Orochimaru's Zelda 3 Hacking Compendium

Special thanks to MathOnNapkins, Euclid, Puzzledude, Spane, XaserLE, NEONswift, ghillie, RedScorpion, wiiqwertyuiop, GameMakr24, Erockbrock, Reshaper256, sorlokreaves, d4s, and of course Sephiroth 3 for creating Hyrule Magic!

And also to everyone else who gathered data for that game over the years without you, this compendium wouldn't have been a reality!

"May Zelda 3 romhacking live long and prosper for at least another decade!"

Version 1.0

TABLE OF CONTENTS

~ Chapter VII - Additional knowledge

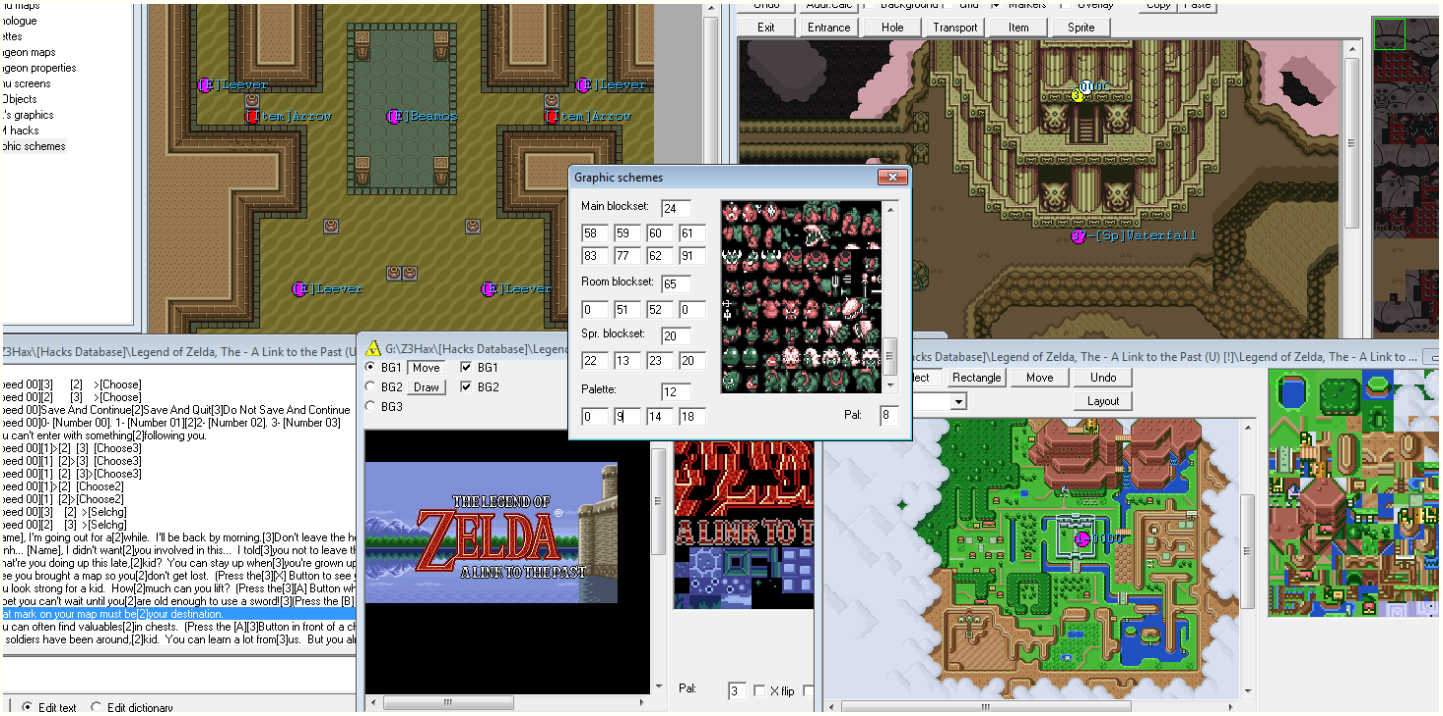
~ Chapter VIII - Hacks database

~ Epilogue

~ Preface

Hello everyone, Orochimaru here to give his old faq a major overhaul. It's been seven years since my last faq update, so I figured it might be a good time to update it all. A lot has happened in those seven years and we always need new recruits, hence the need for a newer and much improved faq!

~ Chapter 1 - The Hyrule Magic Editor



Hyrule Magic is a full fledge editor for "The Legend of Zelda: A Link to the Past" created more than a decade ago by Sephiroth3. It lets you edit many aspects of the game like the overworld, dungeons, dialogue, world maps and many more!

Many bugs in the Hyrule Magic editor can however cause random crashes!

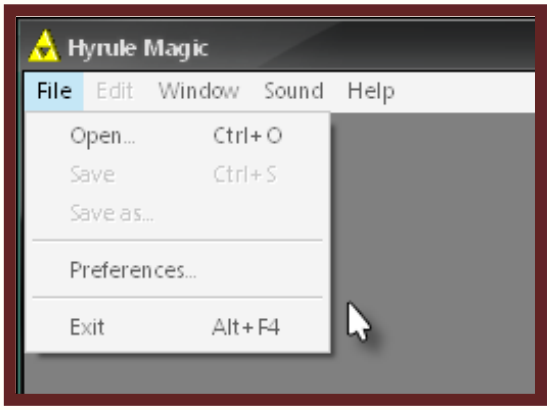
So be sure to make backups of your hack VERY OFTEN ☺

Of course most of these bugs and crashes can be avoided once you get the hang of that editor and it's very steep learning curve ;)

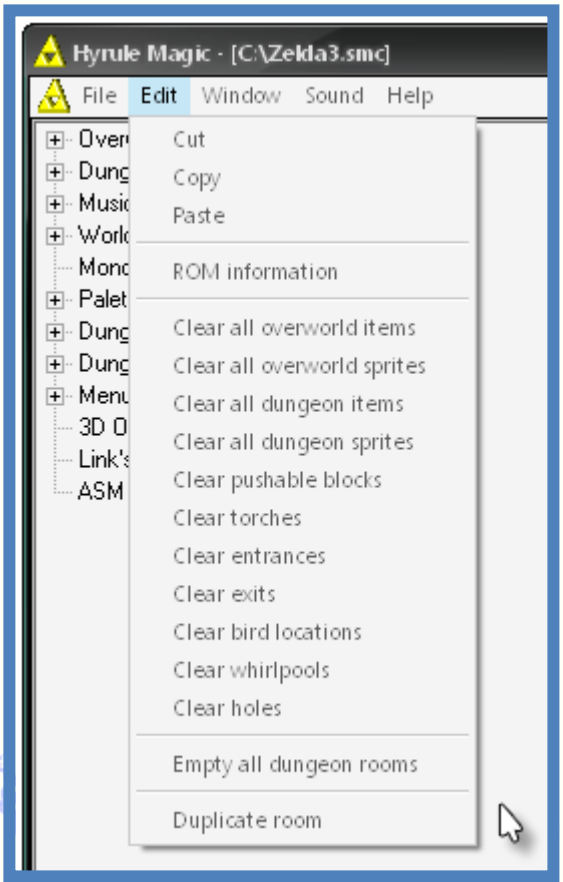
1) The interface

by Sephiroth3

File



Open: *Opens a ROM*
*(Must be a valid **Zelda 3: A Link To The Past USA Version ROM**).*
Save: Saves a ROM with the current name.
Save as: Saves a rom with the name of your choice.
Preferences: Allows you to edit the displayed sprite names, the sound volume, the instruments mapping and the FSNASM path.
Exit: Exits the program.



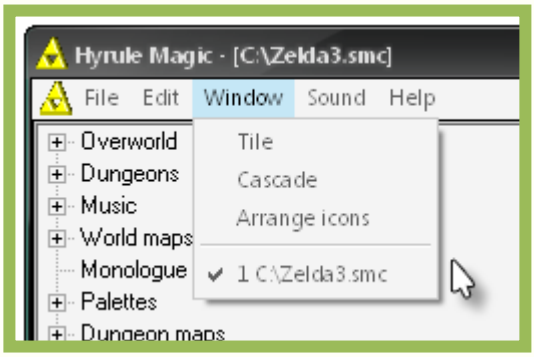
Edit

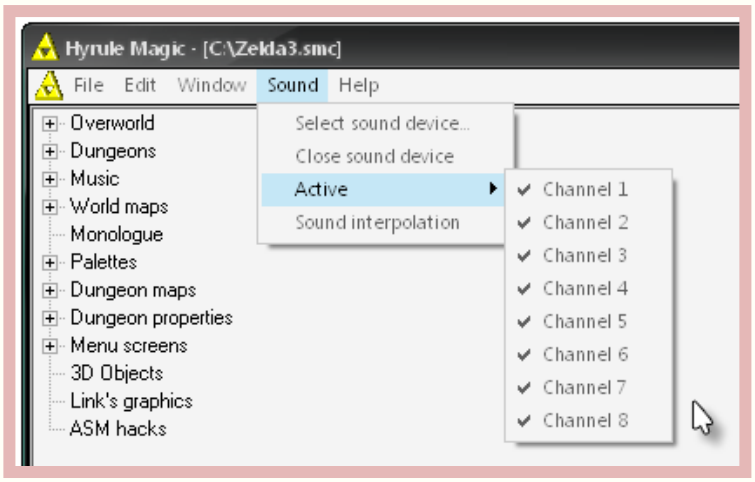
Cut: Self-Explanatory
Copy: Self-Explanatory
Paste: Self-Explanatory
ROM information: Allows you to change the internal ROM name and view the current free space for sprites, items, entrances, exits , etc...
Clear all overworld items: Removes all overworld items.
Clear all overworld sprites: Removes all overworld sprites.
Clear all dungeon items: Removes all dungeon sprites.
Clear all dungeon sprites: Removes all dungeon sprites.

Clear pushable blocks: Removes all dungeon pushable blocks.
Clear torches: Removes all dungeon torches.
Clear entrances: Removes all overworld entrances.
Clear exits: Removes all overworld exits.
Clear bird locations: Removes all overworld bird locations.
Clear whirlpools: Removes all overworld whirlpool locations.
Clear holes: Removes all overworld holes.
Empty all dungeon rooms: Self-Explanatory (But not a good idea).
Duplicate room: Copy/Paste a room.

Window

Tile: Arranges the windows in a tiled fashion.
Cascade: Arranges the windows in a cascaded fashion.
Arrange icons: Arranges minimized windows.
You can select a window in the list to activate it.



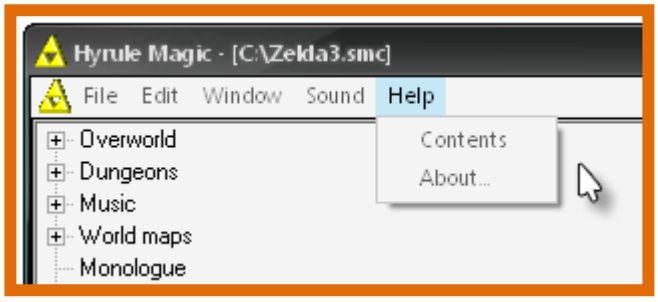


Sound

Select sound device: Choose between a wave and a midi device for the music editor.
Close sound device: Closes the sound device.
Active: Turn sound channels ON/OFF.
Sound interpolation: Turns sound interpolation ON/OFF.

Help

Contents: Displays help contents.
About: Displays copyright information.



2) An introduction to Hyrule Magic

by sorlokreaves

Back-story: ROM hacking really constrains your options. Zelda 3 has a fully-featured map editor, so you'd think that you have the power to re-arrange and re-create rooms like a giant jigsaw puzzle… except that some of the pieces must be in one place, and some of them will automatically re-write themselves with something else. So today, we'll just take a very shallow dip into the waters of **Hyrule Magic**.

Goal: Fiddle with the opening scene of Alttp, changing graphics, text, and object placement.

Now, before we get started, let's verify that we have the same file.

On Windows:

- Make a file named "Zelda3.hash" in the same directory as your ROM, with the contents:

```
1a74468291b02729329dd1357afb45af *Zelda3.smc
```

- Make sure you switch **Zelda3.smc** with your ROM's name. You can download "Checksum" from **this link**.
- Install it and double-click on your .hash file. You should get a message that the checksum is valid.

On Linux:

- Make a file named "Zelda3.hash" in the same directory as your ROM, with the contents:

```
Jacksum: Meta-Info: version=1.7.0;algorithm=md5;filesep=/;encoding=hex;

1a74468291b02729329dd1357afb45af Zelda3.smc
```

- (Make sure to replace **Zelda3.smc** with your ROM's name).
- Install the "Jacksum" package:

```
sudo apt-get install jacksum
```

- …and run this simple check from the command line:

```
jacksum -c Zelda3.hash
```

- You should get an "Ok" message.

Normally I'd just post the valid file, but I'm not allowed to distribute it unless you also own Zelda3. Sorry for the inconvenience. By the way, if your checksum fails, you can still try editing the ROM in Hyrule Magic, with some chance of success. (For example, your ROM may be headerless).

Please note that Hyrule Magic can royally screw up your ROM. Sometimes this is your fault (i.e., loading an expanded ROM), sometimes it is a bug in the editor, and sometimes it's a feature of the Zelda3 ROM (*e.g., "whoops, that tile can only be used for doors"*) and you have no idea how to undo such changes.

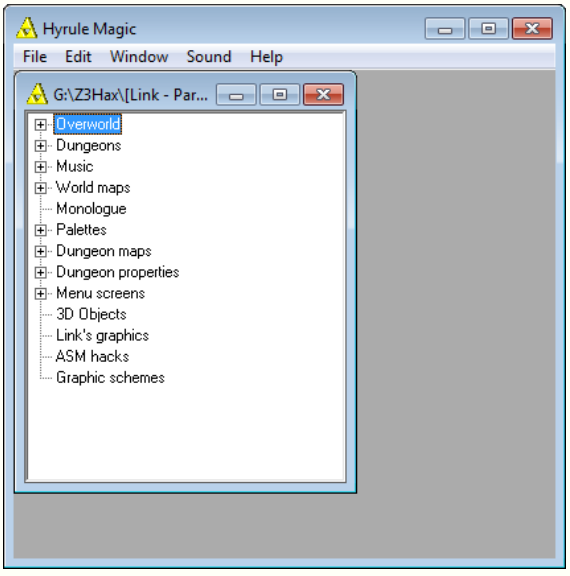
For this reason, it is ***highly*** recommended that you keep rolling backups of your project files, even more so than you would normally. I'm still looking for a VCS that handles binary files well, so for now I'd just recommend daily backups to USB and server-side backups every time you accomplish anything time-consuming.

A Note About Linux

*As you might have guessed, we'll be using Linux for this tutorial. Why, you ask? Well, first of all, **because we can**, and I want to be fair to all homebrew developers. Secondly, I've got a grand goal of making a **totally portable, OS-included thumb drive**, and that requires me to learn Linux. Don't worry, you can follow this tutorial on Windows (I've tested it on both). Here we go!*

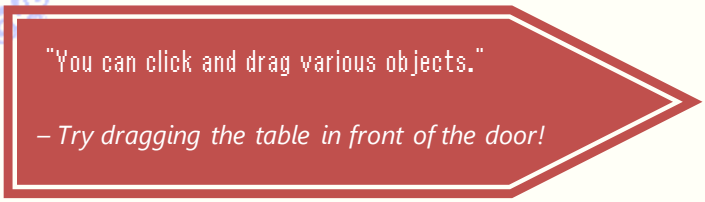
Moving Objects, Changing Message Text

Our goal today is a simple, shallow proof-of-concept hack. Copy the Zelda3 ROM to a new directory, then open Hyrule Magic, click "File -> Open", and browse to the new copy. Once you open the ROM, a whole list of menu items will appear.



"This list is a categorical view of all the data from the Zelda3 ROM that Hyrule Magic understands."

*- Click on "**Dungeons**" and then double click on "**Entrance 00**". Zelda3's dungeons are set up in a grid-like fashion, and the first one just happens to be your Uncle's house.*



Hit "Ctrl+S" to save your ROM and "Ctrl+F4" to close the dungeon editor window.

Double-click on "Dungeons" to compress the list, and then double-click on "Monologue" to edit the game's text. Double-click on the line which reads:

```
[Name], I'm going out for a[2]while.  I'll be back by morning.[3]Don't leave the house.
```

This is the first thing your Uncle says to you after Zelda's message arrives in your head. The **[Name]** will be replaced by your character's name. The **[2]** and **[3]** indicate where the second and third lines begin.
Change the text to read:

```
[Name], you sneak out too much.[2]I've blocked the exit.[3]You can't leave now!
```

Then click the "Set" button.

In the same way, change message 32:

```
[Window 02][Speed 03]Help me...[2]I am in the dungeon of the[3]castle.[Waitkey][Scroll]I  know there is a hidden path[Scroll]from  outside  of  the castle  to [Scroll]the garden inside.
```

...to:

```
[Window 02][Speed 03][2]Zelda here... [3]didn't you learn how to jump?
```

Save your ROM (Ctrl+S) and close Hyrule Magic. You can now test the Zelda3 ROM you just hacked. **Zsnes** works on most Linux boxes:

```
zsnes /media/Windows/myProject/Zelda3.smc
```

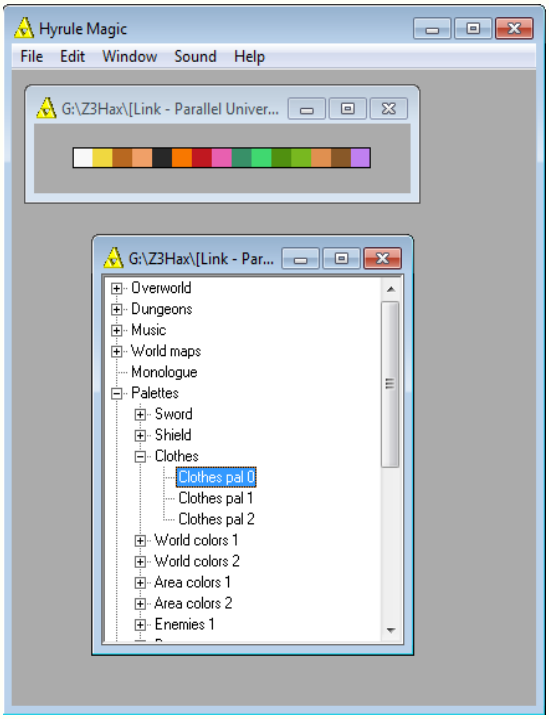
...or just open Zsnes and browse to the ROM. Go through the opening sequence; boy you're stuck now!

Picture: *Link's Uncle can walk over the table, but Link cannot!*

Well, that was fun; let's do one more quick hack and call it a day!

Palette Hacking

Open your ROM in Hyrule Magic, and expand "Palettes", then "Clothes", then double-click "Clothes pal 0". Like most SNES games, Zelda3 stores each pixel it draws by reference to a "palette". If you were to change, say, the "white" color referenced in the "sword" palette, that would affect every white pixel in Link's sword. Right now, we're going to change the color of Link's clothes. You should see:



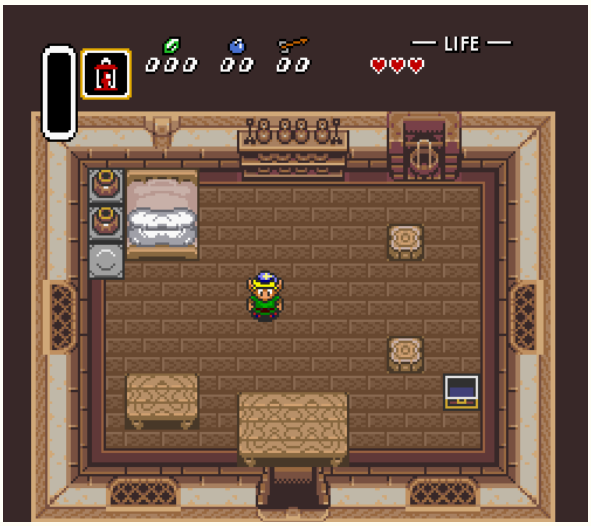
Count the colors from the left: "1, 2, 3...". So color "5" is black.
You can change a color by single-clicking on it and choosing a new one.

Change the following:

- Color 2 from yellow to very dark red.
- Color 6 from orange to dark blue.
- Color 8 from pink to yellow.
- Color 9 from green to a brighter green.
- Color 10 from green to a slightly lighter green than color 9.
- Color 11 from green to matte blue.
- Color 12 from lime green to matte yellow.

This isn't scientific; just pick colors you like, really. Now, save and load your ROM.

Links makeover is complete!



Picture: *Looks a bit like a nightie…*

You may have noticed that Link's face gets all messed up if you pull something. This is not a bug; when we changed orange to dark blue, we affected more than just the trim on Link's hat. If you were serious about changing Link's color scheme, you'd probably have to edit the pixels which make up Link's graphics in addition to the palettes.

Go make yourself a sandwich –or a coffee– we'll be back after a short break!

Required Reading!

You can skip this section if you like; it's about game design, not hacking. But I think you should read it. Yes, you *have* to. That's right.

While you eat your sandwich and drink your coffee, think about your target audience. Realistically, the only people who'd want to play your hack are ones who've already beaten Zelda3 multiple times. Probably, you're one of these people too. So how can you cater to their particular needs?

Well, for one thing, the castle gets pretty boring after you've played the later dungeons.

We're going to use a tried-and-true method of Zelda **power-questers** today: we'll deny the players use of the sword for the first bit of the game. The lantern, unfortunately, is a pretty lousy weapon, so let's give the player some arrows; that should allow for some interesting puzzle construction. But what should we do about Link's uncle? I see two possible solutions:

The Hacker's Solution: Scan the ROM and determine the code which gives Link the sword. Switch it so that Link's uncle gives him the bow. Perhaps alter his sprite to be holding the bow, too.

The Storyteller's Solution: Just give the bow to Link at his house and delete his uncle in the palace basement; the story can be molded to fit around this.

I'm inclined to go with solution number two —it opens up a lot of creative possibilities. Solution 1 is a bit show-off, and it doesn't really fit so well into the story for Link's uncle to give him anything except a sword. At this point, it's good to brainstorm and consider a few potential trip-ups, and some tentative solutions (lest we back ourselves into a corner).

Will Link Meet His Uncle Later? Maybe, but we won't be able to have him in his "wounded" state giving Link anything, since this glitches the game after Zelda is rescued. We have his walkabout graphic; maybe we can replace one of the villagers with him. Or, we can have a mutant in the Dark World say he's him. Or, we can change the ending credits from "Your Uncle Recovers" to "Your Uncle is Found".

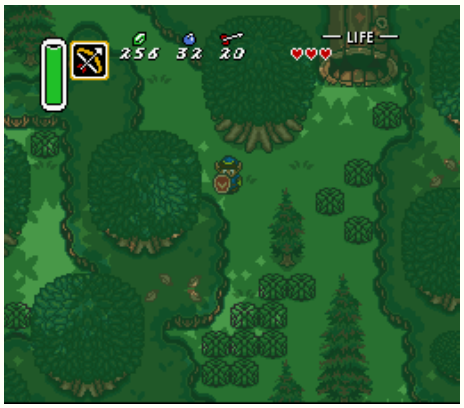
Why Does Link Have a Bow in His House? Maybe he's a hunter? Actually, it makes a lot more sense for him to have a bow than a sword. Maybe we can change the general feel of Zelda3 to be more "woodsy" and less "running from the law".

What About the Game Text? Link's uncle can say anything; even the same message he says now. Zelda's second message is easy to replace; something like "Half past the hour; all citizens to bed!". The first message is tricky. We can have Sora Link monologue ("I've been having these… weird thoughts lately."). Or, we can have him dreaming of his Uncle scolding him for playing around with his bow. Be creative.

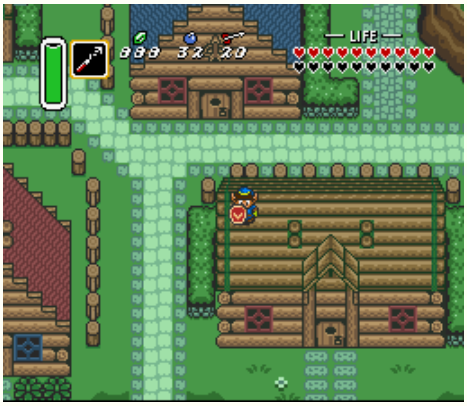
Will Link Be Able to Get the Sword Later? This is a very good question, especially if your Uncle can never give you the sword. Can you put it in a chest? At the very least, you can get the Master Sword, so that'll be our backup.

This was a bit of a digression, but I wanted to impress the importance of thinking creatively and creating a world of your own. The hacker's solution is elegant, but it actually creates a game that "feels" wrong. The lantern was added to your house to make it feel like you were heading out into the rain, to rescue a princess from some unspecific evil. I've always felt this was poorly done in the original game. Grabbing your bow and rushing out into the rain is a lot more romantic. (Stumbling upon your uncle and nabbing his bow is… a non sequitor.)

Even if you were the best ASM hacker in the world, you still shouldn't desire to change everything. At that point, you might as well just develop an entirely new game. Rather, you want to make a game that plays like Zelda3, but feels totally different. I always take Quest for Calatia as my inspiration. Here are two innovations this developer makes:



1) *He uses different trees and houses to imply that Calatia is a colder country.*


2) *He allows you to climb onto house rooftops and hookshot between chimneys.*

Those two simple changes really expand the player's mind as he is exploring. He feels like the fully-equipped link from Hyrule exploring a place which forces him to adapt his strategy.

Another example of great innovation uses **ASM**:

MathOnNapkins modified the game so that you can throw bombs in eight directions, and can throw them either a long distance or a short one.

This mod subtly changes the nature of bombs into something much more utilitarian. I'm sure you could come up with all sorts of puzzles involving bombs and moving platforms, or switches that must be bombed simultaneously.

3) Bow & Arrow Quest

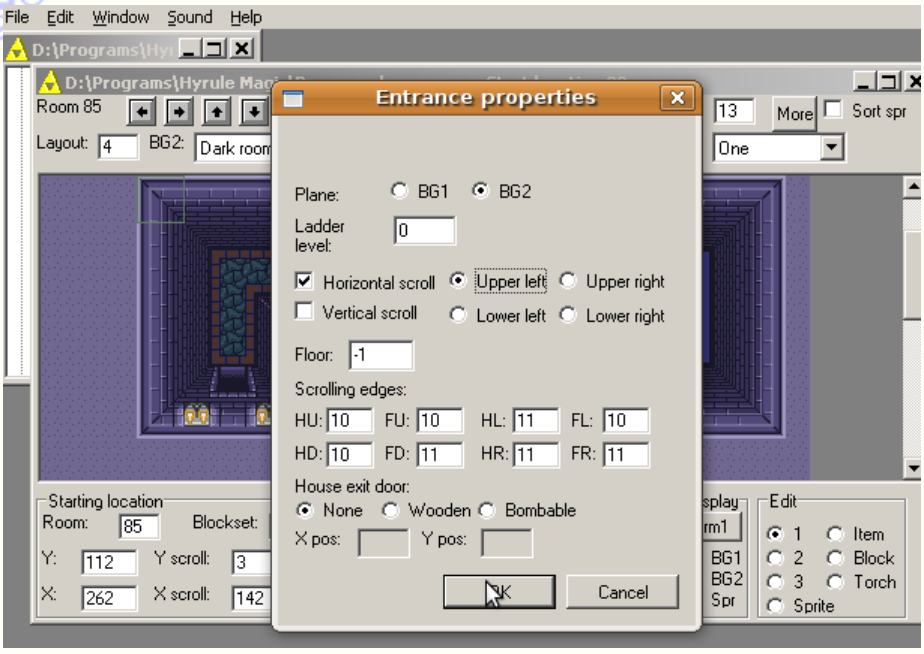
by sorlokreaves

Back-story: We decided in the last post to modify Zelda3 to make the bow-and-arrow the primary weapon for the first part of the game. But how do we actually go about doing this? And, how can we make the first part of the castle more challenging for advanced players? An arrow-related puzzle is in order! But this means we'll have to start learning how to use (***shudder***) the *Dungeon Editor*. (There is no *Dragon Editor*).

Goal: Become proficient in using the *Dungeon Editor*, remove the sword, add the bow, and modify the first dungeon room you fall into to require the clever use of arrows to exit.

Step 1: Understanding Backups

Open a clean copy of Zelda3 in Hyrule Magic. Make SURE this is a ***copy***, because I'm going to show you how simple it is to permanently muck it up. Expand "Dungeons" and click "Starting Location 03", then click on the "More" button at the bottom of the dungeon screen (near "Starting Location"). See the radio button for "Upper Left", next to "Horizontal Scroll"? Click that:



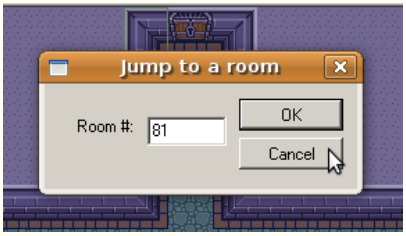
…and then click "Ok" and save. Start the ROM…. and notice that it crashes after the name select screen. Ok, go back to the "More" button and change the setting back to "Upper Right". Save, start the ROM…. it's *still* failing to work.

With one simple click, we have irreversibly corrupted our ROM. Sure, you can try to HEX it back to the *status quo*, but that could take hours. A better solution is to keep regular backups, and to do your most dangerous fiddling directly *after* you backup. Really, I'm not kidding when I say that you *need* to back up your working ROM far more often than usual.

Step 2: Understanding the Dungeon Editor

Open a clean copy of the Zelda3 ROM in Hyrule Magic. It's time to play around with the dungeon editor. Open an MP3 player, too, for your sanity.

Expand the "Dungeons" tab and browse down to "Starting Location 04". Double-click, and you should see a familiar room: the entrance to the sewers. In the top-left corner of the window it says "Room 81". Next to this are some arrows to switch to the next room in each of the four directions. Finally, there is a "Jump" button which allows you to go immediately to a room by number. Click on the arrows a few times to see where it takes you, and then click on "Jump", type in "81" and hit "Ok". You're now back at the throne room.



Close the dungeon view and scroll down to "Entrance 32" & open it. This is the room you fall into after pulling up that bush outside the castle. Actually, that's somewhat inaccurate. This ***entrance*** only takes effect when you walk out of and then back into this room. It is the same **room** as the one you fall into (and the one you start from after saving) but depending on how you enter the room determines a number of minor things like where the camera starts. It's important to remember this when you edit certain properties of this entrance. Make sure you ***always*** access it by double-clicking "Entrance 32", unless I tell you to click "Starting Location 03".

Look at the box in the lower-right corner of the Dungeon Window. There are several options:

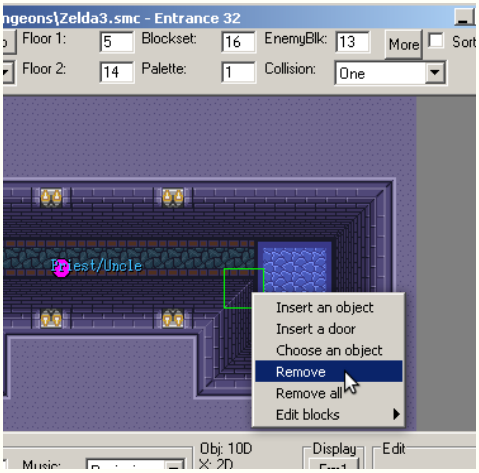
- ***"1"*** *– Layer 1 is where we do most of our editing. Doors can go here, too. Treasure chests often are placed here.*
- ***"2"*** *– Layer 2 is required for some editing tricks (like bridges) –I won't be using it here. Doors can also be placed here.*
- ***"3"*** *– Pretty much only doors can go here. We'll use this layer to make an entrance leading outside –that door has to go on this layer.*
- ***"Sprite"*** *– Things like enemies, crystal switches, and NPCs go here.*
- ***"Item"*** *– Items that can be placed on the ground or in pots go here.*
- ***"Block"*** *– This setting lets you organize your pushable blocks. We won't be using this.*
- ***"Torch"*** *– This setting is for torches. We won't use this, except to delete one torch.*

Click on "1", and then look for the corner jutting out on the left side of the watery place where you land. When you click on it, you will see the following information displayed in the lower-right:

Obj: 10D
X: 2D
Y: 11

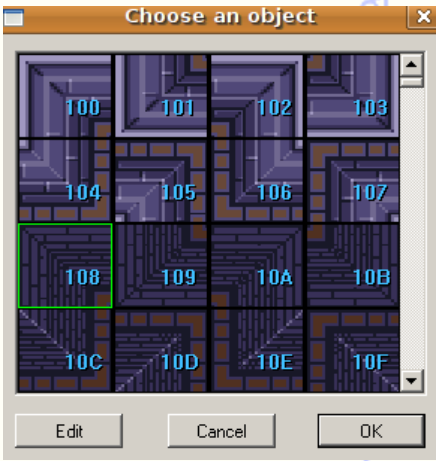
Right-click on this object, and choose "Remove".

You've successfully deleted the corner.



''Right-Click then "Remove" to Delete an Object''
Now click on the piece directly south of the one you deleted. Click & drag it left until its X co-ordinate reads "2A". Now move the mouse over the top edge of the same piece and click+drag to "extend" it until its "Size" reads "03".

Unfortunately, the piece we moved is now *under* another piece. So, left-click on it, and then press "Ctrl+V" to bring it to the front. (Ctrl+B bring it to the back).



Right-click anywhere and choose "Insert an Object".
Scroll up to object 108, click it, and choose "Ok".

It will appear roughly where you right-clicked; drag it to 2A,0B. Makes a nice corner, doesn't it?
Congratulations, we've made our first real change to the room! Save.

''Move the Wall to the Front with Ctrl+V''

We can describe what you just did using a simple table:

Comment	*Before*	*After*
wall	x:2D y:11	—
wall	x:2D y:15	x:2A y:0F size:3 *front*
wall	—	obj:108 x:2A y:0B *front*

That means "Delete object at location 2D,11", then "Take object at 2D,15, move it to 2A,0F, resize it to 3 units, and bring it to the front", then "Add a new object of type 108 at 2A,0B and bring it to the front". It's important that you understand this syntax; we'll be modifying a lot of objects, and it would be horribly cruel to describe each one with a paragraph.

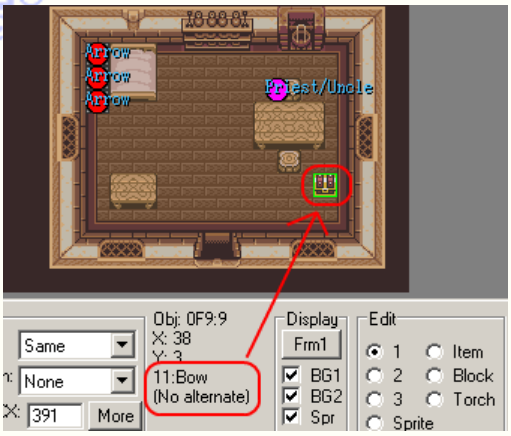
The "Comment" column is an informal addition to help you keep your place.

A note about resizing is in order. Sometimes, when you place a piece down (particularly one that stretches vertically) the piece will stretch to fill up all available vertical space. Please note that you can *still* resize it, you just have to grab the handle at the bottom and drag it *all* the way to the top. This is annoying, but not insurmountable.

Step3: Enabling The Bow

Back in Entrance 32, click on the "Sprites" radio button, then click on "Priest/Uncle". Then, right-click on him, and choose "Remove". Our Uncle will now no longer be there to give us the sword.

Close Entrance 32 and open Entrance 00. Click once on the chest with the lantern. You'll see the word "Lamp" under the "Obj" section of the window. Press "+" (or "\") to scroll through the list of possible treasure chests until you reach "Bow" (do *not* choose "Bow&Arrow" –it won't display a message when you take it.) Then change the editing mode to "Item" and click on each heart, then using "N", "M", "J", and "K" to switch the item to "Arrows". Save.



You Can Change Treasure Chest Contents With + and "

Link now has access to the bow, 15 arrows, and not much else. Play through your game a bit, and you'll notice that arrows are a bit hard to come by. Also, we're stuck in the basement, due to that wall we placed in the last section. We'll change that next.

Step 4: A Complete Roomset

Modify the following in Entrance 32. Note that the "size" I specify might require you to stretch the component horizontally, vertically, or both.

Comment	*Before*	*After*
lights	x:1A y:09	size: 00
lights	x:1A y:15	size: 00
lights	x:2F y:25	x:30 y:09
lights	—	obj:80 x:3B y:11 *front*
wall	x:2A y:14	x:27 y:14
wall	x:18 y:09	x:25 y:14 size:01
wall	x:2A y:18	x:27 y:18
cement	x:18 y:18	—
cement	x:28 y:18	x:26 y:18 *back*
wall	x:2A y:1A	x:27 y:1A
wall	x:2D y:17	x:2A y:16 *front*
wall	x:31 y:17	x:31 y:16 size:00
stairs	—	obj:F9D x:2E y:16 *front*
wall	—	obj:113 x:33 y:16 *back*
wall	—	obj:004 x:33 y:17 size:01 *back*
water	x:30 y:0F	x:2E y:0F
water	—	obj:0C8 x:36 y:0F size:01 *back*
edge	x:30 y:0F	x:2D y:0F size:06 *front*
edge	x:30 y:0E	x:2D y:0E size:08 *front*
edge	—	obj:044 x:2D y:16 size:06 *front*
edge	x:30 y:17	x:35 y:17 size:00

Here's what it looks like after you're finished:

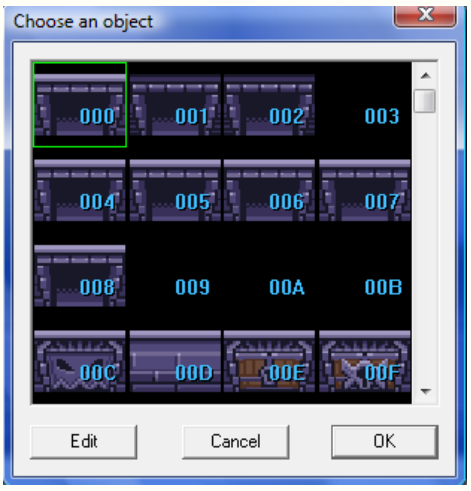
Our Entry Room to the Castle Basement is Complete

You've learned a lot about from this, I hope. For example, you've learned that tiles can overlay like crazy, and the movement rules are determined in a very sensible, visible fashion. (This becomes less intuitive once you start using two layers). You may have also learned that if you left click on an object, then right-click and choose "Add Object", the default object will be the type of the first object you clicked on. This makes it really easy to insert a new tile if you can see a similar one on-screen. It's a real time-saver.

Step 5: Doors and Staircases

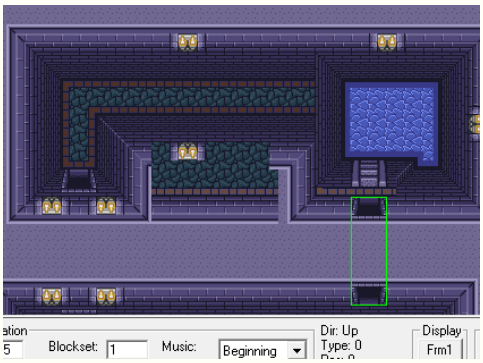
We now move on to one of the most difficult parts of dugeon editing: doors and staircases. Actually, that's a lie: everything's amazingly difficult. But doors (and staircases) require actual explaining, not just hex mangling, to really understand the concept. And then there's that mind-numbing number pad "feature", which I consider to be pure evil.

Backup your ROM and open the working copy in Hyrule Magic. (We can switch to Windows for a while, yes?) Now, click on layer 3 in "Edit", and then right-click and choose "insert a door". Scroll up to door 000, click on it, and click "Ok".



Now, left-click on the door you just placed. This is where Hyrule Magic gets evil. You need to press the arrow keys ***on the number keypad*** in order to move the door around. The regular arrow keys won't work. (On a laptop, you'll have to use the function key, probably). In my opinion, this is a pure sin of interface design —why would you *ever* assign different functionality to these two sets of keys. Even Ctrl+Arrow is a better solution.

Press the right arrow until the "Pos" of your new door reads "8". This may look a bit odd for now, but we'll fix that later.



You can also add this door on layer 1. However, if you do that, the door on the left might disappear; you can manually add it back (it's door type 001, but we'll change that later.) A bigger problem is what happens if the entrance door disappears. If that is the case, you have to re-add it —and it can ***only be added to layer 3***. Also, there are ***two*** doors which must be stacked to enforce the entrance. Do the following:

- First put down door type 02A and align it over where the exit was.
- Then, put down door type 033 and move it directly ***over*** the door you just placed.
- Now save. Your entrance functionality should have been restored. If not, make ***sure*** that you placed ***both*** doors on layer 3.

Now that you've fixed the entrance (or maybe you were lucky and didn't have to) you should add another door, of type 016, and move it to position 7. Finally, click on the left-most door (the one that was already there) and move it to position 6. Right-click on it, select "choose an object", and change it to object 000.

I recommend keeping all doors in layer 3, but it's up to you. Either way, save your game.

Our staircase, by the way, already works. If you look at the very southern part of the room, you can see that we have to climb a staircase before exiting the "Dungeon". This implies that you enter the dungeon on the "ground level". In reality, though, staircases usually work so long as you place the start and end bits on valid walkable ground. (In case you haven't figured it out yet, the "depth" in Zelda3 is just an illusion.) I don't know enough about Zelda3 to know if you can just keep chaining staircases onto each other, but I'm going to try to stay consistent with the game until I feel adventurous. So, when I put that staircase in our small confined room, I knew that meant I was going to delete the other one.

Do that now: edit plane 1 and right-click on the staircase at 0E,34. Choose "Remove". Save. We'll fix the layout later.

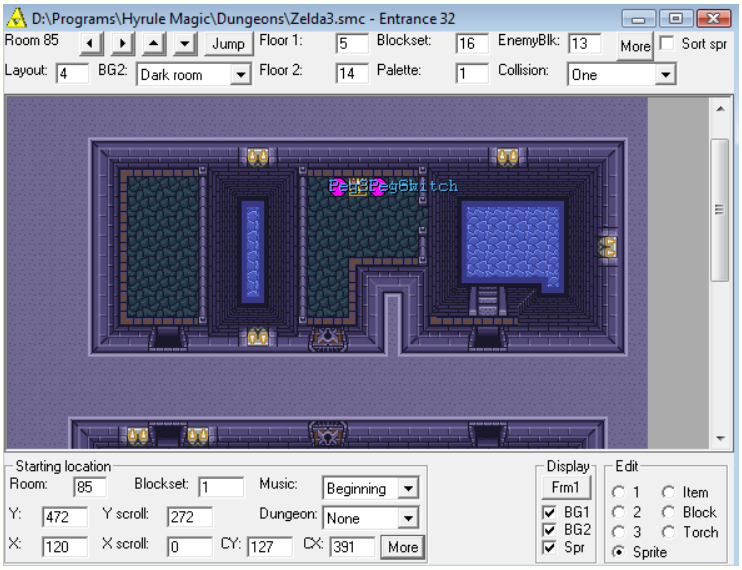
Step 6: An Arrow Puzzle

We want to spice up our first Entrance with an arrow-only puzzle. My idea is to have a crystal switch the player sees as he falls in from the garden, and then have him walk around to the far door, then shoot an arrow over the gap at the crystal switch. The middle door is a red herring; there's no way to get the big key yet. Maybe we can put a treasure chest there to reward the player later when he actually gets the big key.

Perform the following actions:

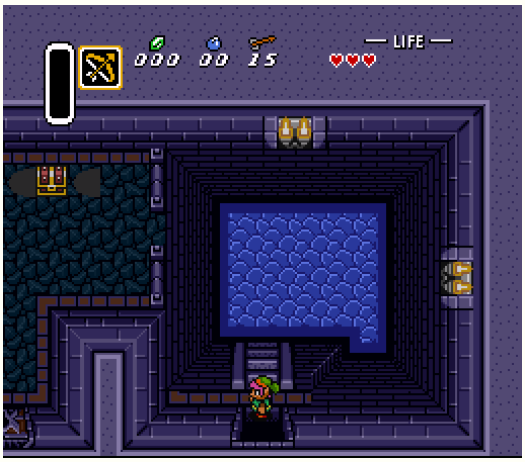
Comment	*Before*	*After*
Comment:	Before:	After:
wall	x:14 y:14	x:22 y:14 *front*
wall	x:14 y:18	x:22 y:18
wall	x:14 y:1A	x:22 y:1A
lights	x:11 y:1B	—
lights	x:0B y:1B	—
wall	x:15 y:11	x:18 y:18 size:00
wall	x:11 y:17	x:19 y:18
wall	x:0B y:17	x:14 y:18
light	x:1A y:15	x:17 y:1B
wall	x:0F y:17	—
wall	x:11 y:15	x:19 y:0E size:04
wall	x:0B y:0F	x:14 y:0E size:04
wall	x:0B y:0B	x:14 y:0B *front*
wall	x:0F y:0B	x:18 y:0B size:00 *front*
wall	x:2F y:0B	x:2D y:0B size:04
wall	—	obj:10A x:19 y:0B *front*
light	x:1A y:09	x:17 y:09
barrier	—	obj:069 x:13 y:0B size:0C
barrier	—	obj:069 x:1D y:0B size:0C
barrier	—	obj:069 x:29 y:0B size:00
barrier	—	obj:069 x:29 y:11 size:00
edge	—	obj:79 x:17 y:0F size:08 *front*
edge	—	obj:7A x:19 y:0F size:08 *front*
edge	—	obj:3F x:17 y:0E size:00 *front*
edge	—	obj:40 x:17 y:18 size:00 *front*
water	—	obj:0C8 x:17 y:0E size:02 *back*
treasure	x:30 y:2C	obj:F99 x:22 y:0C Item: 73:sword 1 *front*

Switch to "Sprite" mode, right-click and choose "Insert an enemy". Choose "PegSwitch" and click Ok. Put this one at x:24 y:0C. Repeat, and place the second one at x:20 y:0C. Then, right-click on each of the "Knight" sprites below and choose "Remove". Now Save.



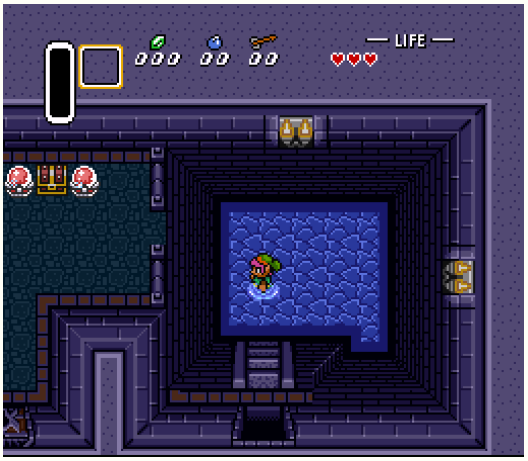
The Top of Our Dungeon Room is Complete!

We're beginning to get a feel for this dungeon: the player drops in, sees a switch, learns there's no way he's getting the big key, and then decides to hit the switch from the other side. Whew, what a lot of work! However, if you load the ROM in zsnes, you'll notice a problem immediately:



Our Crystal Switches are All Shadowy

The problem is, all the enemies on one "level" use the same palette. The crystal switch is considered an enemy, so right now it is using a palette with nulled-out colors. Looking at the Perfect Guide, you can see that several palettes use PegSwitch: 15, 27, 26 and 10, to name a few. Open your ROM and change the "EnemyBlk" to "10". Sometimes you'll have to pair a palette up with a "blockset". Eventually, you should be able to create custom groupings of blocks and colors for use in your dungeons.
By the way, change the "Blockset" from "16" to "1". It's not necessary, but it makes things look a lot more regal:



Proper Crystal Switches and a New Blockset

Step 7: Finishing Up the Arrow Puzzle

We need to edit the lower half of Entrance 32. We could simply render it as a big square, with some blue crystal blocks impeding our exit. But that's boring, and will make our audience groan for its lack of creativity. Let's think: what kind of building did we fall into?

- The original garden entrance was… some weird, useless storage shed.
- Now, we've got a locked door and a chest. (There's a sword inside, but it could be changed to something more interesting… like flippers).
- So we're in a… control shack for Hyrule River?

- Then the lower room should have lots of flowing water, and be a "public space" (no fancy jumping needed to get around).

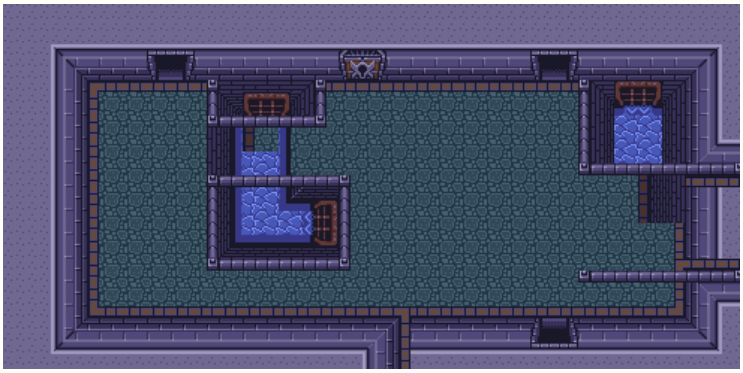
Hmm…. maybe we can also be more creative with the crystal blocks. Let's use them as bridges. Perhaps the big key door is locked because it's a control room for the "bridges". We'll have to think of a better explanation later (water purification? Ph testing?) but at least that's more interesting than a big rectangle.

Ok, make the necessary changes.

- Switch to level 3, click on the door at position 6 (the exit) and press right (on the numpad) ***twice***. Now, click on the *other* door at position 6 and press numpad-right twice. We've now moved the exit doors.
- Switch to "Item" level, right-click on each "Magic" circle, and choose "Remove".
- Switch to "torch" level, right-click on our only torch, and choose "Remove". Switch back to level 1.

Comment	*Before*	*After*
Comment:	Before:	After:
pedestal	x:2C y:2A	—
wall	x:35 y:27	x:37 y:27
wall	x:38 y:28	—
wall	x:28 y:24	—
cement	x:3C y:24	—
wall	x:35 y:2B	x:37 y:29
wall	—	obj:106 x:3A y:2C *back*
wall	—	obj:001 x:3E y:2C
wall	x:35 y:35	—
wall	x:38 y:38	—
wall	x:26 y:38	—
cement	x:22 y:3C	—
wall	x:16 y:30	x:3A y:36
wall	x:1A y:30	x:3E y:36 size:01
wall	x:0D y:27	x:17 y:27 size:00
wall	x:2D y:27	—
wall	x:09 y:27	x:13 y:27
wall	x:13 y:2D	—
wall	x:13 y:31	—
wall	x:13 y:34	—
wall	x:09 y:2B	x:13 y:2B size:03
wall	x:0D y:34	x:15 y:33 size:04
wall	x:17 y:2D	—
wall	x:25 y:2D	—
wall	x:09 y:34	x:13 y:33 *front*
cement	x:1A y:34	—
wall	—	obj:10A x:18 y:27 *front*
wall	—	obj:10E x:18 y:2F
wall	—	obj:003 x:1C y:2F
water	—	obj:0C8 x:15 y:2D *front*
water	—	obj:0C8 x:15 y:31 size:04 *front*
edge	—	obj:079 x:15 y:2B size:08 *front*
edge	—	obj:07A x:19 y:2B size:05 *front*
grate	—	obj:FEF x:1B y:31 *front*
edge	—	obj:045 x:19 y:31 *front*
edge	—	obj:040 x:15 y:34 size:04 *front*
grate	—	obj:FEC x:16 y:28
barrier	x:09 y:38	x:13 y:2A size:06 *front*

barrier	x:12 y:38	x:13 y:2F size:08 *front*
barrier	—	obj:022 x:13 y:36 size:08 *front*
barrier	—	obj:069 x:13 y:27 size:00 *front*
barrier	—	obj:069 x:1C y:27 size:00 *front*
barrier	—	obj:069 x:13 y:2F size:04 *front*
barrier	—	obj:069 x:1E y:2F size:04 *front*
lights	x:11 y:25	—
lights	x:0B y:25	—
wall	x:16 y:34	—
wall	x:16 y:3A	—
wall	x:22 y:38	—
wall	x:22 y:30	x:20 y:3A
wall	x:22 y:34	x:20 y:3E size:01
wall	x:25 y:31	x:33 y:29 size:02
wall	x:25 y:35	—
wall	x:29 y:35	—
pot	x:33 y:31	—
pot	x:33 y:33	—
wall	—	obj:108 x:33 y:27 *front*
wall	x:37 y:27	*front*
water	—	obj:0C8 x:35 y:27 size:01 *front*
grate	—	obj:FEC x:35 y:27 *front*
barrier	—	obj:022 x:32 y:2E size:0A *front*
barrier	—	obj:022 x:32 y:37 size:0A *front*
barrier	—	obj:069 x:32 y:27 size:04 *front*



We're Nearly Done... Should We Add a Third Culvert?

Looks incomplete, no? At this point, I was going to add another minor waterway, but looking at the top screen, I noticed that it wouldn't line up right. Originally, I was just going to say that some sewer line carried away the middle waterway –it looked pretty cool after all. But I think it's actually better to leave this out.

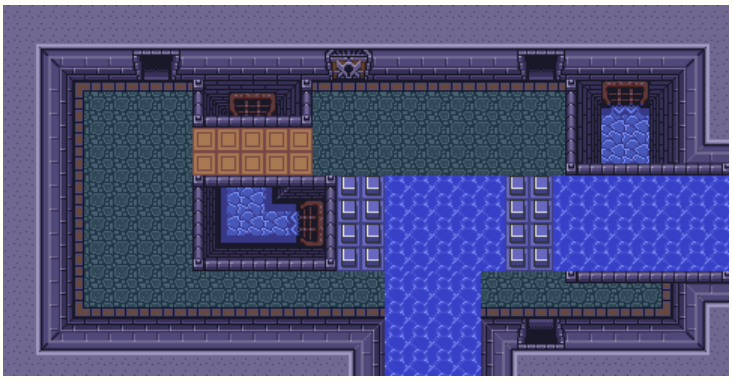
Naturally, I create all of my tutorial ROMs twice: once to make sure the concept works and once to write out instructions for the blog post. But I want to stress the importance of reflecting on your design decisions: I had the third water-way in place on my first ROM, and decided not to add it on the second one.

Ok, let's add the final pieces to our room:

Comment	*Before*	*After*
Comment:	Before:	After:
bridge	—	obj:0B9 x:13 y:2B size:05 *front*
bridge	—	obj:0B9 x:13 y:2D size:05 *front*
water	—	obj:0C9 x:30 y:2F size:0D *front*

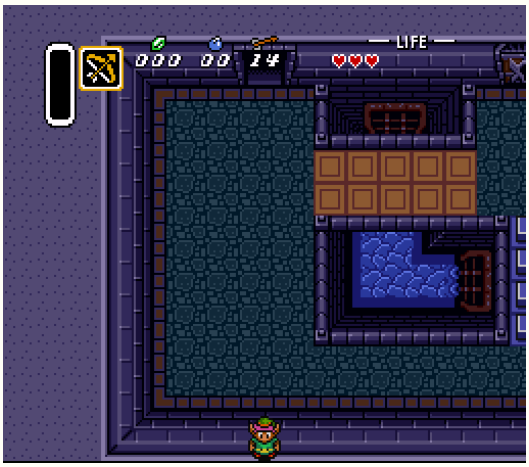
water	—	obj:0C9 x:1F y:2F size:0D *front*
water	—	obj:0C9 x:23 y:34 size:06 *front*
wall	—	obj:107 x:2A y:3A *back*
wall	—	obj:062 x:2A y:3E size:01 *back*
bridge	—	obj:092 x:1F y:2F size:04 *front*
bridge	—	obj:092 x:21 y:2F size:04 *front*
bridge	—	obj:092 x:2D y:2F size:04 *front*
bridge	—	obj:092 x:2F y:2F size:04 *front*

And we're done! (You guys didn't notice that I forgot that corner, did you?)



Our Completed Room

Our puzzle is complete. Back up your ROM, please, since we're going to be hacking the header next. When you run your ROM, you can solve the puzzle easily and exit the room. (There is a weird glitch where sometimes the screen won't scroll properly after walking up the stairs for the first time. I've designed this room so that it won't hinder your progress, but I'm not sure exactly how to fix that bug if it occurs again.) However, something's amiss if you then re-enter the room. You're in the wrong starting location!



Entering From the Original Location, Only Now the Door's Gone

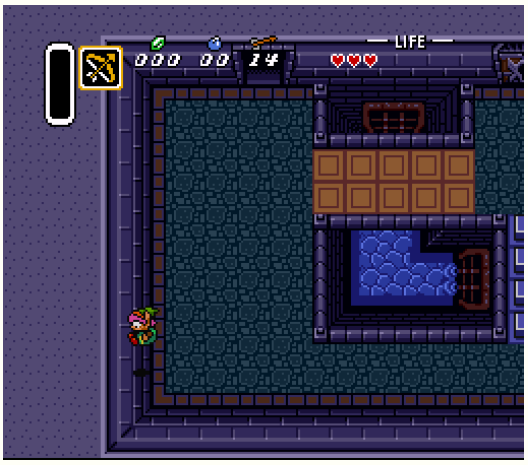
Open Hyrule Magic (make *sure* you've backed up your ROM) and look in the "Starting Location" tab. You'll see "Room", which we won't change, and then "X" and "Y" which we will. You can do the math to set these properly (each tile is 16 pixels wide and high) but I prefer to just switch to a room with a similar entrance and copy the values from there. We'll also need to adjust the x-scroll value to start our camera further over to the right. And, we should fix the CX and CY values.

Change the following:

X = 376
xScroll = 256
CX = xScroll + 128 = 256+128 = 384
CY = yScroll + 112 = 272 + 112 = 384

Please click on the "More" button in the **top** part of the Dungeon editor, then click "Ok". Then Save. Sometimes Hyrule Magic won't persist room values if you don't click the "more" button; I have no idea why.

Open your ROM, and re-test… whoah! You're colliding with obstacles that aren't there, and you're jumping into ground that's really water:

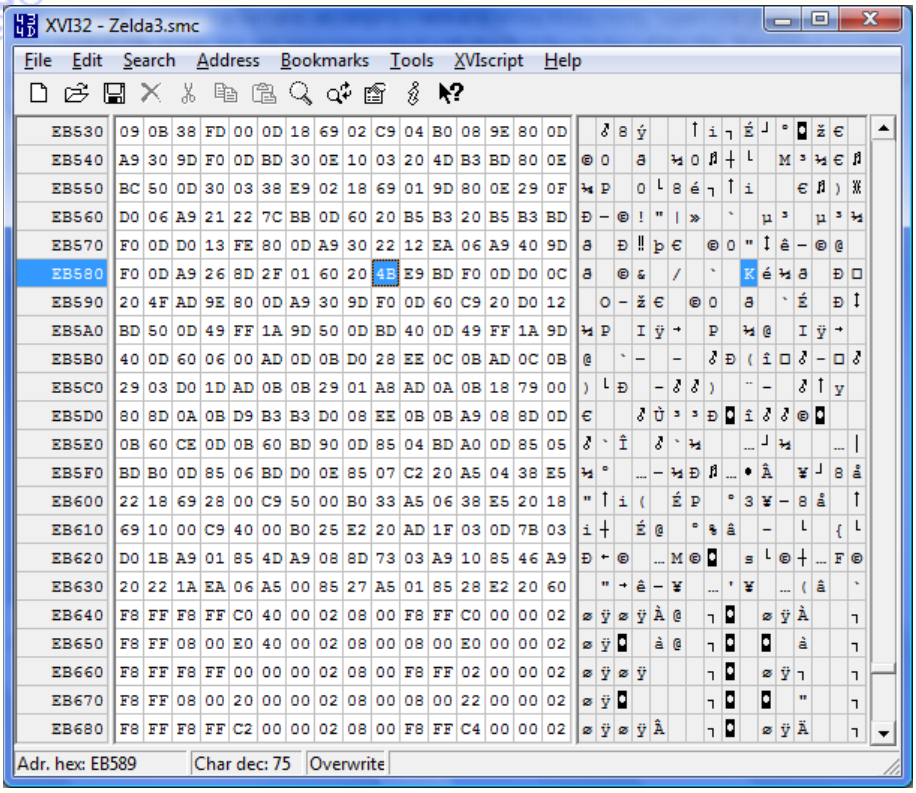


Jumping Into the Wall? Must Have Mis-Aligned the Camera

It's almost like Hyrule Magic is ignoring the value we entered for xScroll. And… if you open your ROM and double-check, ***it is ignoring it***! This is another frustrating feature of Hyrule Magic; sometimes it doesn't really save your data. You'll understand why after we cover the ridiculously complex way that Zelda3 saves the xScroll parameter. For now, it's sandwich time. I recommend a BLT.

Step 8: Fixing the xScroll Wonkiness

You'll need a hex editor. We'll cover Windows first. I recommend using XVI32. Open the program, then open your Zelda3.smc ROM. We are now presented with the first few bytes of the file. They're all header space, so they're not too interesting. On the left is the hexadecimal view, and on the right is the ASCII view. Scroll down a bit and you'll see what I mean:



A Typical Day With Your Hex Editor of Choice

In this example, we are in position 0xEB589, which has the value 4B. In ASCII, that would be the letter "K", but probably this is actually just used as a number for something, not as a letter. A lot of the data in Zelda3 is compressed, so you won't be able to find it even if you know what you're looking for.

At the bottom of the screen, you can see that 0x4B is actually the number 75. Also, you can see that "Overwrite" is on. As you may recall from the early days of Microsoft Word, this means that typing a letter will erase the next letter, so that you never expand the length of the document. Unlike Word, you ***almost always*** want this on when hex editing. Otherwise, you'll offset the entire ROM, and data won't load where it's supposed to. Don't worry, you'll notice if you do this when you open the ROM; it's almost guaranteed not to load at all.

We know that **xScroll** for the dungeons starts at 0x14F45 and each value is 2 bytes in length. Since we want the data for Entrance 0×32, we do some simple math in Windows Calculator (hex mode) and get:

14F45 + 32*2 = 14FA9

Hitting "Ctrl+G" in XVI32, we click "hexadecimal" under "Go to", ensure that our "Go mode" is "absolute", and type in 14FA9. Click "Ok", and our cursor jumps to address 0x14FA9.

The highlighted box reads 00, and the next box to the right reads 0A. Since Zelda3 stores bytes backwards, we read this as 0x0A00, or 2560 …which is ***not*** zero, as we might expect. Enter 2560 in Windows Calculator and click "Bin" to convert this to binary. Now, we have:

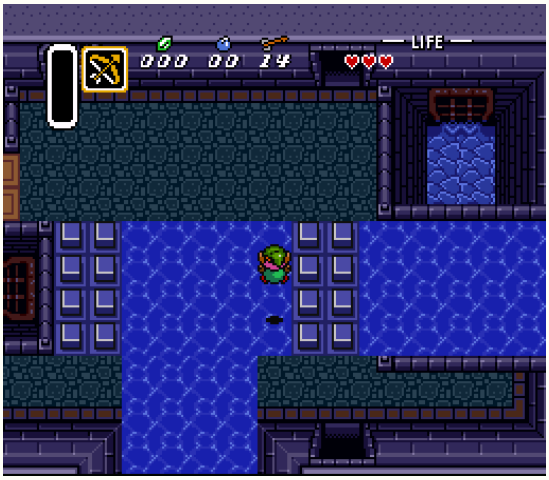
101000000000

Note the green bits. Zelda3 uses these for… something, and it prevents Hyrule Magic from saving the xscroll value properly. The red bits are the actual bits used for saving xscroll. So, if we want an x-scroll of 256, then we need:
101100000000
…since 100000000 in binary is 256 in decimal notation. (Generally, you won't want to mess with the green bits).

If you enter 101100000000 into your calculator and then switch to hexadecimal, you get 0x0B00. Breaking apart and flipping the bytes, we get:

00 0B

So, in XVI 32, click once on the 0A next to the cursor to move it there, type **0B** and then Save. Load the ROM in Hyrule Magic and you'll see that our **xScroll** has, in fact, been updated. Load it in zsnes and you'll notice that our scrolling problem is gone.



Hooray! Let's Go for a Swim!

Step 8.2: Fixing the Wonkiness on Linux

I'm still looking for a good hex editor on Ubuntu, so if you have any suggestions, please let me know. For now, you should use tweak, a console editor with a slight Emacs inclination.

```
sudo apt-get install tweak
```

Please read the previous section on Windows to understand how we do the math. Basically, we just want to replace the **0A** at offset **14FAA** with an **0B**. To do this:

tweak Zelda3.smc
Then, press Ctrl+X then g
Then type **0x14FAA** and hit **Enter**.
Then type **0B** and press Ctrl+X then Ctrl+S
Then press Ctrl+X then Ctrl+C

Okay, we've edited our file, saved it, and exited. Re-run our game, and you'll see that Link is back where he should be when entering the room.

Possible Improvement

Here's a summary of some things I'd like to modify, but had no time to. You can play around with them, if you'd like:

- Change the opening message from Link's uncle to make more sense.
- Re-arrange Entrance 00 (Link's house, pre-Intro) to have a more intelligent layout. The chest should go in front of his bed, for example, or maybe he should have two chests; one with the bow and one with arrows.

- Change the messages from Zelda, and the message when you pick up your bow so that they make more sense.

- Advanced. You actually can't save; it will drop you back off at the introduction. This is (I think) because the "Castle" save flag is set when your Uncle gives you your sword. Read through this guide and *MON's Pinups* until you know what has to be changed, and write a small hack which fixes this. (Maybe you can set the "Castle" save flag when Link falls into the hole?)

- If you play through the intro a bit, you'll notice that arrows are in short supply, and that the ball-and-chain trooper goes down with a mere four arrows. Also, the lamp should be harder to get (maybe in that chest we put between the crystal switches?) and the boomerang should be acquired later. Fiddle around and fix these. Also, consider making more arrow-related puzzles in Hyrule Castle. Just remember that it is a castle, after all, so maintain a believable architecture.

- For the hardcore players, devise a way for them to get into the basement without even the bow. (Make it very tricky to figure out.) This means they have to beat the entire castle by throwing pots. Trust me, hardcore players will love you for this.

- Add a telepathic tile to the water control room which helps to clarify its purpose. (Hint: click on the "more" button on the top of the dungeon window).

4) Sprites Encyclopedia

by Orochimaru, Erockbrox, MathOnNapkins

Erockbrox created the original excel sprite sheet which this database is based on! You can find it on ROMhacking.net: http://www.romhacking.net/documents/629/

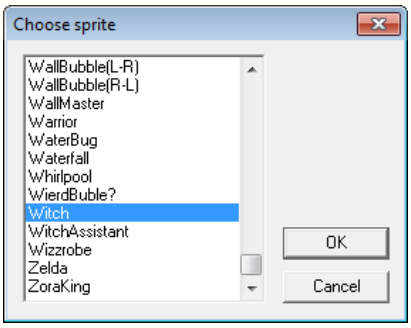
Orochimaru: *There exist a rather large number of sprites in the game. This next part of the guide will thoroughly explain each sprite in the game and which are their overall functions. Thanks to major advancements in knowledge through the years, we are now able to change some sprites in Zelda 3 in ways like we never seen before.*

I expended it trying to fill all the missing pieces of the puzzle and then found MathOnNapkins sprite catalog in his RAM documents… found pretty much all the remaining unknown sprites and what they are used for!

I've taken the courtesy of adding pictures to each and every single sprite that I'm aware of; while adding some valuable info to them… like their gfx tiles locations, monologue locations and palette/gfx#.

Before we get started with our extensive sprite database lets do a quick recap on how to add sprites on both overworlds and dungeons to familiarize ourselves with the sprite menu.

*To add sprites on overworlds, you must first click on the **"Sprite"** button at the top of your Hyrule Magic Menu. Then right click anywhere on your overworld screen and select the option: **"Insert sprite"**. A new window will pop up (**"Choose sprite"**), in it we find out pretty much all the sprites used in the game (minus some some sprites that are exclusive to dungeons and others that need **ASM** to move them).*



*To add sprites in dungeons, you first have to click on the white circle next to **"Sprite"** in the **"Edit"** box in the bottom right corner of the dungeon editor window.*

*Then right click in the location where you want to add your sprite, but this time in the drop down menu, click on **"Insert an enemy"**.*

*You'll arrive at the same **"Choose sprite"** menu as in the overworld editor but with a few more sprites that are exclusive to dungeons.*

Question: What are the overlord sprites?

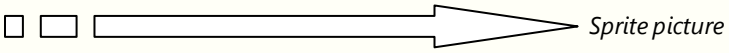
***MathOnNapkins:** "They are not run by the same code as the other sprites. The other main distinction is that they don't really share many of the same properties as normal sprites. You usually can't kill them or even damage them at all.*

They are "Overlords" because they are like mini programs that spawn other sprites and control them. For example, the Armos Knights are controlled by the Armos Knights overlord. If that overlord wasn't there the Armos knights go flying off the screen (try it if you don't believe me).

They are accessible in Hyrule Magic as sprites with numbers above 0xFF (in the dungeons) or 0xF4 (in the overworlds) and above, but I do not like the way he handled that, personally. For one thing, their numbers are separable. Meaning, you can have up 16 sprites in each room and up to 8 overlords in each room, so the number of one that is available doesn't depend on the other."

HOW EACH SPRITE IN THE DATABASE IS STRUCTURED (LEGEND):

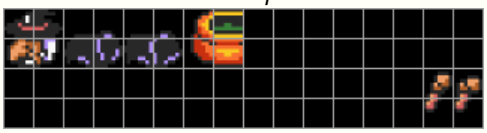
36 - Witch → *Sprite ID and Name (As seen in Hyrule Magic)*



GFX tiles location: The location of the sprite graphics within the sprite sets.
(Each overworld area/dungeon room has four sprite sets).

For example:

GFX tiles location: *3rd sprite set in HM*



*It means that the sprite **"Witch"** is using the graphics contained in the third sprite set to draw its sprite.*

Type: Whether the sprite is a monster, person, element, trigger etc… *(In the case of the witch, it's a person)*

Setting: Whether the sprite is used on overworlds or dungeons by default. (*Overworlds in this case.)*

GFX# and Pal: The sprites usually are either using:

1) ***Spr GFX# and Spr Pal (overworlds)*** (*The Witch is using* ***Spr GFX#: 0 and Spr pal: 1***)

2) ***EnemyBlk and Palette (dungeons)***

or -

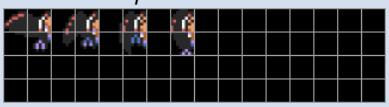
Monologue #: All the dialogue lines that the sprite is using (linked).

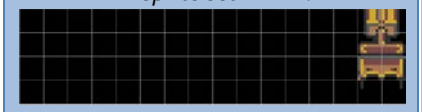
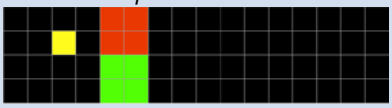
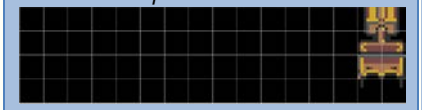
More info: Specific information related to the sprite.

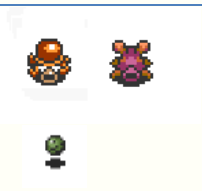
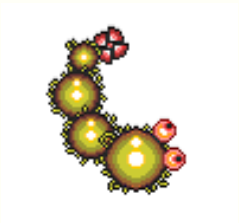
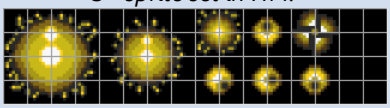
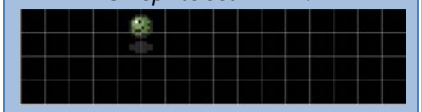
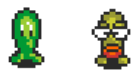
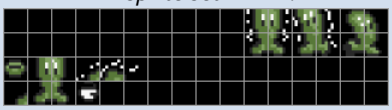
For example: here's the additional information of the witch sprite:
She allows magic powder to spawn in the magic shop. You need to give her a mushroom for that effect to happen.

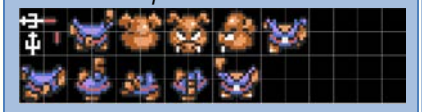
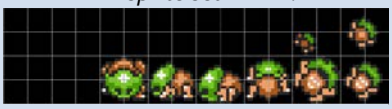
Sprite ASM code: Sprite coding provided by MathOnNapkins source code. Some sprite have ASM data in them while others lack such data. To make use of that ASM data, remember that can always copy those lines of text into a blank text file and rename it as ***"your_file_name.asm"***. Where ***"your_file_name"*** is where you write down the name of your new file. If you want to know how to insert ASM patches there's a really good tutorial in the ASM section down this guide!

a) Sprites usable in both overworlds and dungeons

HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
00 - Raven 	*4th sprite set in HM:* 	*Monster*	***Light World*** ***Dark World***	***4*** ***23***	***3*** ***13***	N/A	*Comes in two versions, one for each world*
01 - Vulture 	*3rd sprite set in HM:* 	*Monster*	***Overworlds***	***8***	***4***	N/A	

02 - 02 Better name: *Stalfos Head* 	*1st sprite set in HM:* 	*Unused*	***Dungeons***	***8***	***11***	N/A	*Strange behaviour as it runs away from you and goes straight into the other rooms/areas.* *Doesn't seem to be used in game.*
03 - BigCanon	N/A	*Unused*	N/A	N/A	N/A	N/A	*Crashes the rom on both overworlds and dungeons.*
04 - PullSwitch 	*4th sprite set in HM:* 	*Element*	***Dungeons***	***6***	***29***	N/A	*If you pull on the switch, it makes the good sound.*
05 - DownSwitch	*4th sprite set in HM:* 	*Unused*	***Dungeons***	N/A	N/A	N/A	*Sort of switch being pulled from above.* *The location of the unused graphic tiles is* ***color coded.***
06 - TrapSwitch 	*4th sprite set in HM:* 	*Element*	***Dungeons***	***6***	***29***	N/A	*If you pull on the switch, it makes the bad sound*
07 - ?FloorMove? Better name: *PullSwitch2* *Can be used with the "bomb wall" trigger in dungeons!* *To do this, insert a door 018 in one on your rooms, then click on "More" in your dungeon editor window and change one of your tags to "Use lever to bomb wall".* *Then add this sprite and feel free of course to find suitable graphics for it, because there simply aren't any tiles for it.*	*3rd sprite set in HM:* 	*Element*	***Dungeons***	***4***	***0***	N/A	*Switch facing up. If you dash into it from the back side, Link pulls a lever making the good sound.* *Unused in the original game.*
08 - Octorok	*3rd sprite set in HM:*	*Monster*	***Light World***	***11***	***3***	N/A	*Comes in two*

			**Dark World**	**22**	**10**		*versions, one for each world.*
09 - Mouldrum 	*3rd sprite set in HM:* 	*Boss*	**Dungeons**	**12**	**6**	N/A	
0A - 4WayOctorok 	*3rd sprite set in HM:* 	*Monster*	**Light World** **Dark World**	**11** **22**	**3** **10**	N/A	*Comes in two versions, one for each world.*
0B - Chicken *Also: The chicken that's transformed into a Lady*  / *[chickenlady pic]*	**Add chickenlady gfx in there:** *4th sprite set in HM:* 	*Animal*	**Light World** *(ChickenLady)* **Dark World**	**6** **21**	**1** **16**	**381** *(ChickenLady)*	*Comes in two versions, one for each world.*
0C - ?HoveringRock? Better name: *Octorok bullet* 	*3rd sprite set in HM:* 	*Monster Animation*	**Light World** **Dark World**	**11** **22**	**3** **10**	N/A	*Hover still in the air and hurts you & explodes if you get too close.* *It can be killed.*
0D - Cucumber 	*4th sprite set in HM:* 	*Monster*	**Overworlds**	**7**	**6**	**378** **379**	*If you sprinkle magic powder on it, it changes its appearance and even talks!*
0E - SnapDragon 	*1st and 3rd sprite sets in HM:* 	*Monster*	**Overworlds**	**19**	**14**	N/A	*This sprite needs gfx tiles in both the first and third sprite sets. It won't display right otherwise.*

0F - OctoBlimp 	*3rd sprite set in HM:* 	*Monster*	***Overworlds***	***4***	***3***	N/A	
10 - 10	N/A	*Unknown*	N/A	N/A	N/A	N/A	*No idea what this sprite does*
11 - Hinox 	*1st sprite set in HM:* 	*Monster*	***Overworlds***	***19***	***14***	N/A	*Hinox's bombs do activate pegswitches. 1 bomb makes the pegs go, up down up down.*
12 - PigSpearMan 	*3rd sprite set in HM:* 	*Monster*	***Overworlds***	***19***	***14***	N/A	
13 - MiniHelmasaur 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***12***	***26***	N/A	

HM Name	14 - GargoyleGrate
Type	*Dungeon entrance trigger*
Setting	***Overworlds***
GFX#	N/A
Pal	N/A
Monologue #	N/A
More info	

Euclid: "The Gargoyle Grate is used to open up a place in the same screen as the grate (The grate disappears when moved out of

*that screen) in that particular **X** and **Y** position. And it will also open up the former place in that ruined town in the dark world using those particular tiles."*

Puzzledude: "You dont need any gloves to actually open as this sprite is a pull sprite (similar to the 5 rupees pull sprite). You just need to grab the fence and pull it down. The gargoyle grate sprite will then open the door, the pull rupee sprite will throw 5 rupees, while Link gets pushed back.

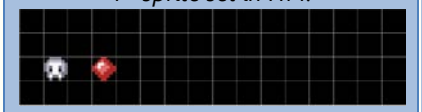
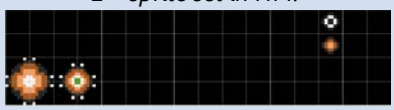
The gate is actually event based. You can pull it to open it any time after rescuing Zelda to the church (part1 mode of the game) or later (part2), but not in the beginning mode.

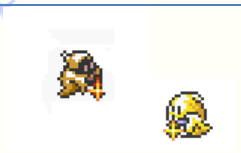
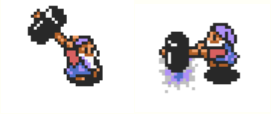
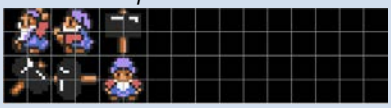
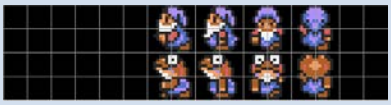
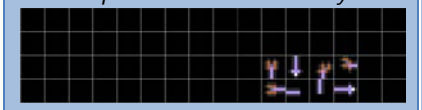
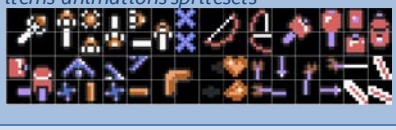
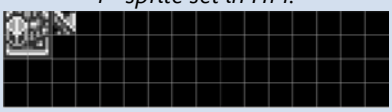
When I was testing this gate long ago, I actually figured out you can make the pull sprite anywhere. But I didn't draw any gargoyle grate statue. The pull sprite was on a random overworld wall (sprite is actually invisible, no gfx change needed).

Once pulled the entire statue appeared on the exact x and y as the original game, but the area was different. Once entered and exited, the entire gate disappeared.

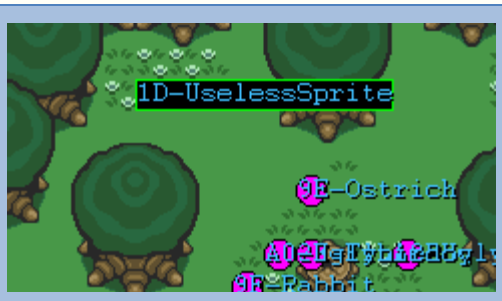
So I guess there are 3 states of the gate. Originally drawn, a second state after pulling, and a third state (fixed overlay after the pull, no further pulling possible).

With this knowledge you could for instance make an invisible statue, that only appears in the specific place of the big area after pulling any object (where the pull sprite is placed). But a fixed overlay after pulling is neccessary in all cases."

HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
15 - Bubble 	*4th sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***10***	N/A	*This sprite is a skull with 4 spinning fireballs; drains health and magic.*
16 - Mutant Better name: *Sahasrahla/Aginah* 	*3rd sprite set in HM:* 	*Person*	***Dungeons*** *(Sahasralah)* ***Dungeons*** *(Aginah)*	***16*** ***16***	***28*** ***7***	**From 048 until 057** *(Sahasrahla)* **293** *(Aginah)*	*Sahasrahla gives you the boots*  **124**
17 - BushCrab 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***4***	***3***	N/A	
18 - Moldorm 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***19***	***13***	N/A	
19 - Poe/Ghini	*4th sprite set in HM:*	*Monster*	***Light World***	***3***	***2***	N/A	*Comes in two versions, one*

			**Dark World**	**21**	**16**		*for each world.*
1A - BlackSmith(Frog 	*1st sprite set in HM:*  *1st sprite set in HM:*  *Graphics loaded in memory:* 	*Person*	**Dungeons** **Overworlds**	**5** **29**	**30** **0**	**From 216 until 226**	*In light world he is a dwarf while in the dark world he is a frog.*
1B - AnArrow? 	*Graphics loaded in memory:*  *- they are located within the inventory items animations spritesets* 	*Trap*	**Dungeons** **Overworlds**	N/A	N/A	N/A	*It's an arrow stuck in a wall.* *It hurts you if you touch it and if you hit it with a weapon it falls off the wall.*
1C - Statue 	*4th sprite set in HM:* 	*Element*	**Dungeons**	**17**	**10**	N/A	*You push it around. Can lay on top of a switch to make it stay activated.*

HM Name	**1D - UselessSprite**
Type	*Useless sprite!*
Setting	***Overworlds***
GFX#	N/A
Pal	N/A
Monologue #	N/A
More info	*Puzzledude: This sprite has no function at all. It is the NOP sprite, that can be useful only to the editor/maker of the game. For instance I couldn't remove the sprites in PW to make the Remodel (since the hex code would be shifted), thus I've put some to NOP, which is this sprite.* *But it can also function as a marker only, to where the ASM code is. So you can remove this sprite entirely, the Flute location and Bird statue are independent from this sprite and entirely ASM based."*



Euclid: The Bird statue in Kakariko Village will only work in a 1024x1024 Area (e.g. Bigger sized ones) and it has to be in that spot. If you change the GFX around it, the bird will still appear there.

Orochimaru: "With the latest Hyrule Magic (version 9.63) you can edit overworld overlays! Meaning that once you move the trigger using ASM, you can then edit the graphics you want to appear once the weathervane exploded."

Weathervane stuff ~ by MathOnNapkins:

Note: *A word about overworld coordinates. The coordinate for 0,0 is at the top left corner OF THE WHOLE OVERWORLD. This applies to areas 0 through 0x7F, not the others. Just because you are looking at something in Hyrule Magic, doesn't mean that top left corner of that map section is (0,0).*

For example, look at the bird's coordinates below. Looking at it in Hyrule Magic, you'd assume that they are roughly the same, but area #$18 happens to be on the left edge of the map, whereas it is about 2-3 whole screens down in terms of Y position. Hence the bird's Y coordinate is about 4 times the size of it's X coordinate.

$48CD5-$48D10 - Initial position data for the weathervane piece sprites (12 of them)

$48CD5 - ???
$48CE1 - ???
$48CED - 12 bytes. *Lower Y coordinate for all 12 pieces*
$48CF9 - 12 bytes. *Lower X coordinate for all 12 pieces*
$48D05 - ???

$48D65 - 1 byte. *Upper Y coordinate byte for all 12 pieces.* Normally **#$07**
$48D72 - 1 byte. *Upper X coordinate byte for all 12 pieces.* Normally **#$02**

Caution: edits to these must be exact and careful, because they are embedded in code. Or rather, they are actually part of the code itself.

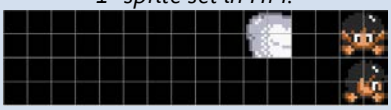
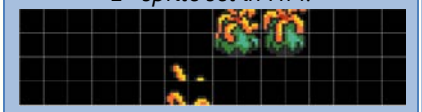
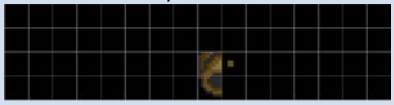
$48DC2 - 16bit. *Y coordinate for the bird to initialize to.* Normally **#$0788**
$48DC7 - 16bit. *X coordinate for the bird to initialize to.* Normally **#$0200**

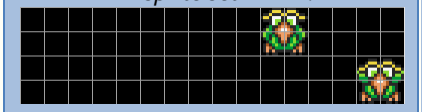
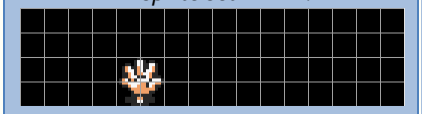
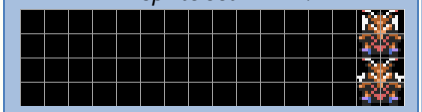
$3A425 - 1 byte. *Overworld area to use with the weathervane.* Normally this reads **#$18**. Make sure to use hex.
$3A42C - 2 bytes. *Overworld Y coordinate lower boundary for triggering the weathervane.* Normally reads **#$0760**
$3A431 - 2 bytes. *Overworld Y coordinate upper boundary* " ... Normally reads **#$07E0**
$3A438 - 2 bytes. " *X* " *lower* " " ... Normally reads **#$01CF**
$3A43D - 2 bytes. " *X* " *upper boundary* " ... Normally reads **#$0230**

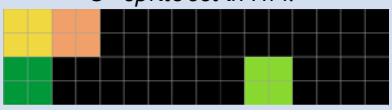
^So basically that defines a square that is **$60 by $80 pixels**, that serves as a trigger location, given the right conditions. Edit these to your heart's content.

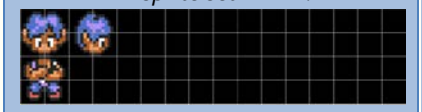
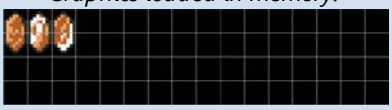
Sprite ASM code

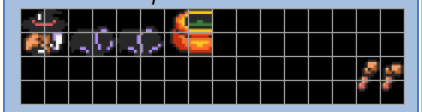
HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info

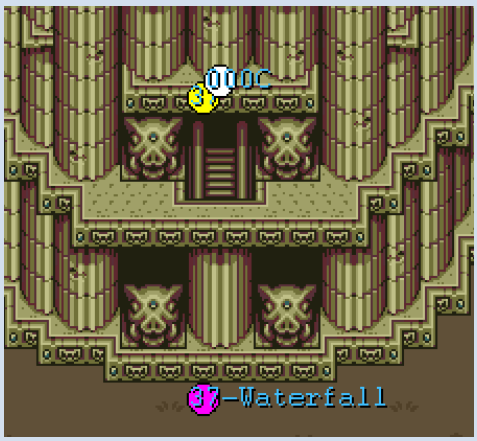
1E - PegSwitch 	*4th sprite set in HM:* 	*Element*	***Dungeons***	***8***	***15***	N/A	*If you hit the pegswitch in the overworlds the entire game rapidly changes colors in a glitched up way.*
1F - SickBoy 	*1st sprite set in HM:*  *Caution: His bedsheet graphics are also used by Link when he wakes up.*	*Person*	***Dungeons***	***13***	***21***	**From 260, until 262**	*Gives you the bug net.*
20 - BombSlug 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***29***	***17***	N/A	*The bombs it makes do activate pegswitches.*
21 - PushSwitch 	*4th sprite set in HM:* 	*Element*	***Dungeons***	***17***	***10***	N/A	*You push left on it to fill up water ways.*
22 - HoppingBulbPlan 	*1st sprite set in HM:* 	*Monster*	***Overworlds***	***19***	***14***	N/A	
23 - RedMiri 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***15***	N/A	
24 - BlueMiri 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***15***	N/A	
25 - LiveTree Better name: *TalkingTree* 	*1st or 4th sprite set in HM:* 	*Person*	***Overworlds***	***21***	***16***	**From 125, until 130**	*The bombs it shoots do activate pegswitches*

26 - BlueOrb Better name: *Octo-Orb* 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***19***	***13***	N/A	*Its color is determined by its X position.*
27 - Squirrel 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***16***	***9***	N/A	
28 - Person?Rm270  (270)  (270)  (274)  (276)  (282)	*2nd sprite set in HM:*  *1st sprite set in HM:*  *1st sprite set in HM:*  *1st sprite set in HM:*  *2nd sprite set in HM:* 	*Person*	***Dungeons***	***5*** ***5*** ***5*** ***7*** ***5***	***7*** ***7*** ***7*** ***7*** ***28***	***???*** ***???*** ***???*** ***???*** ***???***	*This sprite tells you a hint in the game.* *It appears in five different shapes and is entirely room based.* *You can find the rooms they use in the brackets next to their pictures.* *The second and third shapes are also re-used for the archery game, so take caution when editing the graphics.*
29 - Thief  (281)	*1st and 4th sprite sets in HM:* 	*Person*	***Dungeons***	***15***	***35***	***???***	*This sprite is either a village townfolk (only in room 281)*

 (225)  (259)	*1st sprite set in HM:*  *3rd sprite set in HM:* 			**40** **15**	**7** **5**	**???** **???**	*A friendly thief who you encounter in the forest (room 225 only)* *Everywhere else, it is the old man whose son plays the flute. (room 259 by default)*
2A - ?DustGirl? 	*3rd sprite set in HM:* 	*Person*	***Overworlds***	**6**	**1**	**???**	*She sweeps a broom and talks to you.*
2B - TentMan 	*3rd sprite set in HM:* 	*Person*	***Overworlds***	**12**	**8**	**???**	*Gives a bottle to the player.*
2C - Lumberjacks 	*3rd sprite set in HM:* 	*Person*	***Overworlds***	**7**	**6**	**300** **301** **302**	*They saw a tree back and forth*
2D - 2D	*3rd sprite set in HM:* 	*Unused*	***Dungeons***	N/A	N/A	**???**	*Telepathic stone of Sahasrahla talking about the Moon Pearl.*
2E - FluteBoy 	*3rd sprite set in HM:*  *4th sprite set in HM:* 	*Person*	***Light World*** ***Dark World***	**15** **17**	**1** **11**	**???**	*Plays the flute and disappears as you get too close (in the light world).* *He gives you the shovel (in it's dark world form).*
2F - Person? 	*4th sprite set in HM:* 	*Person*	***Overworlds***	**6**	**1**	**???**	*The Labyrinth game; works together with* ***EB-HeartPie*** *And* ***30-Person?***

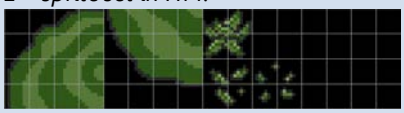
							She starts you off on the maze game. You must talk to her from her right to start the clock.
30 - Person? 	*1st sprite set in HM:* 	*Person*	***Overworlds***	***6***	***1***	***???***	*The Labyrinth game; waiting at the end to claim the price* *He also tells you your time after the game.*
31 - FortuneTeller 	*1st and 2nd sprite sets in HM:* 	*Person*	***Dungeons***	***5***	***30***	***From 234 until 253***	*Must talk to him from a distance to work.*
32 - AngryBrother 	*1st sprite set in HM:* 	*Person*	***Dungeons***	***15***	***2***	***???***	
33 - PullForRupees 	*Graphics loaded in memory:*  *- they are located within the inventory items animations spritesets* 	*Element*	***Dungeons*** ***Overworlds***	***Any*** ***Any***	***Any*** ***Any***	N/A	
34 - ScaredGirl2 	*1st and 4th sprite sets in HM:*  *Caution: Her dress graphics are also used by the sprite **3D- ScaredGirl1**.*	*Person*	***Overworlds***	***6***	***1***	***???***	*Needs a **??-person's door sprite** to work correctly as they run toward that entrance.*

35 - HedgeMan 	*4th sprite set in HM:* 	*Person*	***Dungeons***	***15***	***5***	***???***	*Innkeeper*
36 - Witch 	*3rd sprite set in HM:* 	*Person*	***Overworlds***	***13***	***6***	***074*** ***075*** ***076***	*She allows magic powder to spawn in the magic shop. You need to give her a mushroom for that effect to happen.*

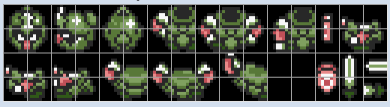
HM Name	37 - Waterfall
Type	*Dungeon entrance trigger*
Setting	***Overworlds***
GFX#	N/A
Pal	N/A
Monologue #	N/A
More info	 *"The Entrance to Ganon's Tower will happen only at the original x/y locations in the original area it's from. Moving the entrance into another area requires a bit of ASM to work."*  *" The sprite is also used to draw a special water animation for the waterfalls. Can be used as a secret dungeon entrance like in the original game and or as a secret area like in Parallel Worlds. The sprite doesn't function on it's own. I believe Euclid edited something in hex to make them work properly."* ***Sprite ASM code***

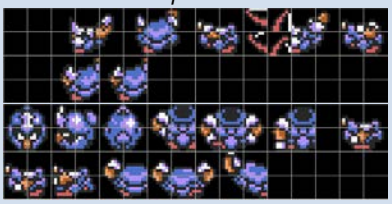
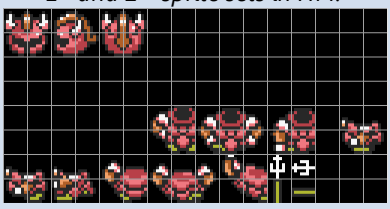
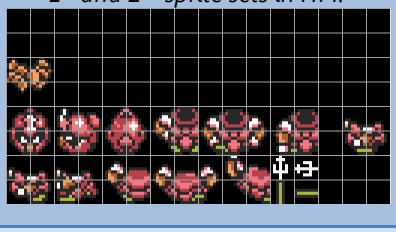
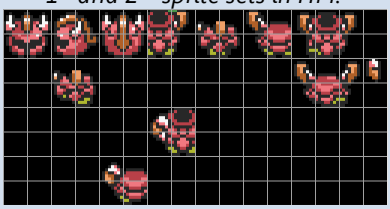
HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
38 - ArrowTarget	N/A (invisible tiles)	*Trigger*	***Dungeons***	N/A	N/A	N/A	*Used w/ the dungeon scrolling wall* *e.g. Big Eye (Dark Palace)* 
39 - GuyByTheSign 	*4th sprite set in HM:*  *Graphics loaded in memory:*	*Person*	***Overworlds***	***10***	***0***	***From 263 until 268*** ***071 (his sign)***	*As soon as you get into the same screen as him, he starts following you around (unless there's a sign next to him).*

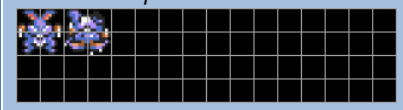
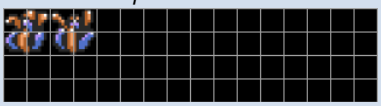
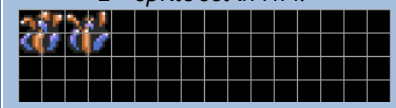
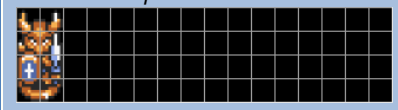
	 *- They are located near the end of the decompressed graphics. Look them up with YY-CHR (They are in the same spriteset as the monkey).*						*He unlocks the **B4-BlueChest** sprite which contains a bottle!*
3A - Person?11,227 Better name: *Magic Powder Bat* 	*4th sprite set in HM:* 	*Person*	***Dungeons***	***9***	***32***	**???**	*Split personality bat who gives you ½ magic.*

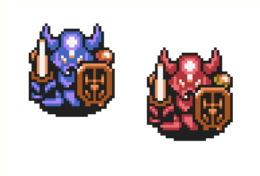
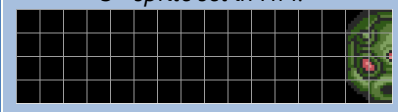
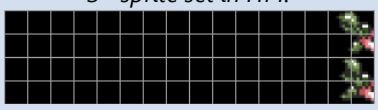
HM Name	**3B - DashItem**   
GFX Tiles location	*1st sprite set in HM:* 
Type	*Special treetop overlay (in the overworld), Book of Mudora and keys (in dungeons)*
Setting	***Overworlds*** and ***Dungeons***
GFX#	***26 (treetop)***
Pal	***6 (treetop)***
Monologue #	N/A
More info	*In the **overworld Area 02** (in Second part), you should notice there's a **3B - DashItem** sprite there (the tree hollow is in the same spot). Moving the sprite only moves the graphics covering the treehole (the top of the tree), you can still run into the **3B - DashItem** bit to remove the cover. Even if you remove the sprite/tree, the hollow tree will still appear in game.* *In dungeons **3B - DashItem** sprite is a key in every room of the game, with the exception of **room 263** where the sprite appears as the Book of Mudora instead.*

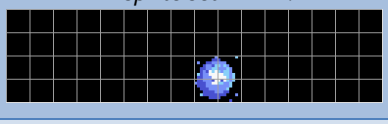
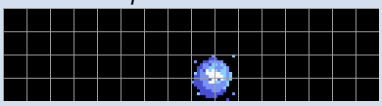
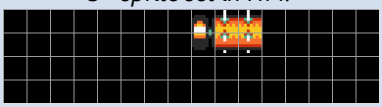
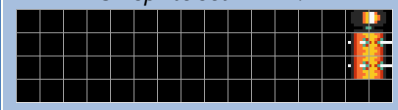
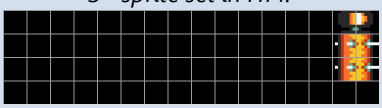
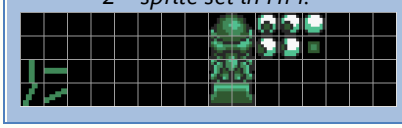
HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
3C - FarmBoy 	*3rd sprite set in HM:* 	*Person*	***Overworlds***	***6***	***1***	**???**	

3D - ScaredGirl1 	*4th sprite set in HM:* 	*Person*	***Overworlds***	***6***	***1***	**???**	*In overworlds she runs away if you touch her and hides in her house while also calling a guard!*
3E - RockCrab 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***10***	***0***	N/A	
3F - PalaceGuard 	*1st and 2nd sprite sets in HM:*  *Caution: the 2nd set graphics are also used by many soldier sprites.*	*Person*	***Overworlds***	***2***	***1***	**???**	*Talking Soldiers from the beginning of the game.* *They can be used to block a certain path.*
40 - ElectricBarrier 	*4th sprite set in HM:* 	*Barrier*	***Overworlds***	***2***	***1***	N/A	*Hyrule Castle Barrier to Agahnim's Tower, You need the MasterSword to destroy the barrier and go through.*
41 - BlueSoldier 	*2nd sprite set in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***3*** ***0***	***0*** ***0***	N/A	*Generic soldiers you find almost everywhere*
42 - GreenSoldier 	*2nd sprite set in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***3*** ***0***	***0*** ***0***	N/A	*Generic soldiers you find almost everywhere*
43 - RedSpearSoldier 	*2nd sprite set in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***39*** ***0***	***0*** ***0***	N/A	
44 - Warrior	*1st and 2nd sprite sets in HM:*	*Monster*	***Dungeons***	***33***	***38***	N/A	

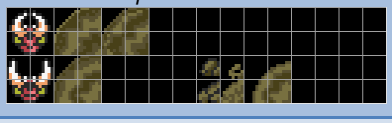
							
45 - HogSpearMan 	*2nd sprite set in HM:* 	*Monster*	***Overworlds***	***23***	***13***	N/A	
46 - BlueArcher 	*1st and 2nd sprite sets in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***39*** ***2***	***0*** ***3***	N/A	
47 - GreenGrassArche 	*1st and 2nd sprite sets in HM:* 	*Monster*	***Overworlds***	***4***	***3***	N/A	
48 - RedSpearKnight 	*1st and 2nd sprite sets in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***33*** ***1***	***38*** ***3***	N/A	
49 - RedGrassSpearSo 	*1st and 2nd sprite sets in HM:* 	*Monster*	***Overworlds***	***4***	***3***	N/A	
4A - RedBombKnight 	*1st and 2nd sprite sets in HM:* 	*Monster*	***Overworlds***	***1***	***3***	N/A	*Doesn't move, trows bombs.* *His bombs do activate pegswitches.*

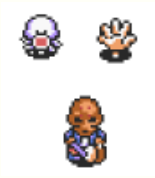
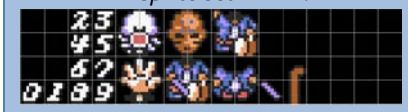
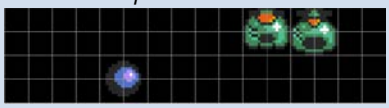
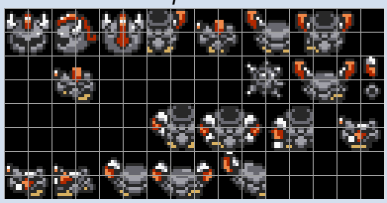
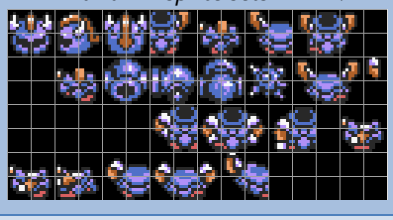
4B - Knight 	*2nd and 3rd sprite sets in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***4*** ***2***	***0*** ***1***	N/A	
4C - Geldman Better name: *Sand Monsters* 	*3rd sprite set in HM:* 	*Monster*	***Overworlds***	***8***	***4***	N/A	*The tan monsters that come out of the desert sand*
4D - Bunny Better name: *GrassBunny* 	*4th sprite set in HM:* 	*Animal*	***Overworlds***	***4***	***3***	N/A	*Without grass it just wiggles on the floor. If you put grass around the sprite it jumps and works.*
4E - Tentacle2 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***11***	N/A	
4F - Tentacle 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***11***	N/A	
50 - GlassSquirrel	**???**	*Element*	*Unused*	N/A	N/A	N/A	*It looks like an invincible sprite that just wiggles back and forth.*
51 - Armos 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***5***	***7***	N/A	*Once they wake up, they hop at you.*
52 - ZoraKing 	*3rd and 4th sprite sets in HM:* 	*Person*	***Overworlds***	***14***	***8***	***???***	*He gives you the flippers in exchange of 500 rupees.*

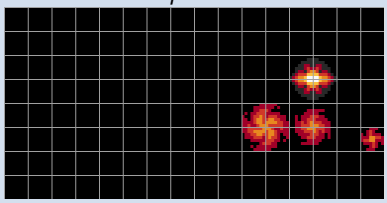
53 - ArmosKnight 	*4th sprite set in HM:* 	*Boss*	***Dungeons*** ***Overworlds***	***9*** ***1***	***11*** ***3***	N/A	*This is the first boss in the game.*
54 - Lanmolas 	*4th sprite set in HM:* 	*Boss*	***Dungeons***	***11***	***4***	N/A	*Second boss in the game.* *On the overworlds this boss is all messed up.*
55 - FireBallZora 	*3rd sprite set in HM:* 	*Monster*	***Light World*** ***Dark World***	***4*** ***22***	***3*** ***10***	N/A	*Comes in two versions, one for each world.*
56 - WalkingZora 	*3rd and 4th sprite sets in HM:* 	*Monster*	***Overworlds***	***14***	***8***	N/A	*Crawls out of the shallow water and walks towards you.*
57 - HyliaObstacle Better name: Desert Palace Barriers 	*3rd sprite set in HM:* 	*Barrier*	***Overworlds***	***8***	***4***	**???**	For use with **B3 - Inscription**; blocks the desert palace entrance in the original game
58 - Crab 	*3rd sprite set in HM:* 	*Monster*	***Overworlds***	***4***	***3***	N/A	
59 - Animal?	*4th sprite set in HM:* 	*Animal*	***Overworlds***	***12***	***1***	N/A	*Bounces away from you and off the screen.*

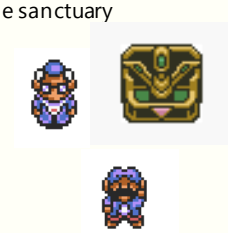
							
5A - Animal? 	*4th sprite set in HM:* 	*Animal*	***Overworlds***	***12***	***1***	N/A	*Flies away and off the screen.*
5B - WallBuble (L-R) 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***10***	N/A	*Fire ball hugs and follows the wall* *Goes clockwise.*
5C - WallBuble (R-L) 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***10***	N/A	*Fire ball hugs and follows the wall* *Goes counter-clockwise.*
5D - Roller 1 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***30***	***24***	N/A	*vertical moving,* ***down -> up***
5E - Roller 2 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***30***	***24***	N/A	*vertical moving,* ***up -> down***
5F - Roller 3 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***30***	***24***	N/A	*horizontal moving,* ***right -> left***
60 - Roller 4 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***30***	***24***	N/A	*horizontal moving,* ***left -> right***
61 - Beamos	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***10***	***9***	N/A	*Static sprite, it's eye rotates and if he sees you he shoots a beam in your direction.*

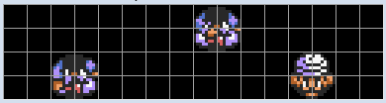
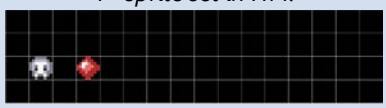
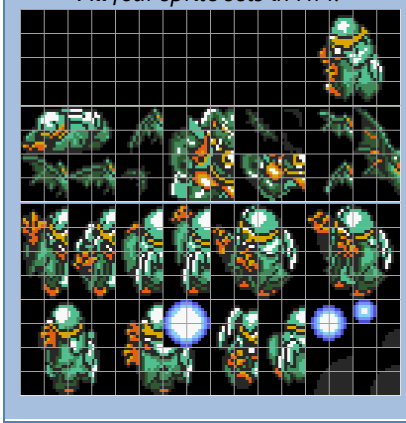
HM Name	62 - MasterSwd  Better name: *Master Sword plus pendants and beams of light*
GFX Tiles location	*3rd and 4th sprite sets in HM:* 
Type	*Event*
Setting	***Overworlds only** (You can add it in dungeons, but the game will crash if you try to grab it.)*
GFX#	***12***
Pal	***1***
Monologue #	**???** **???**(mudora book messages) **???** (sahasrala message after you pick it up)
More info	 *The **B3 – Inscription** sprite in front of the MasterSword has two lines of dialogue, depending on whether or not you have the Mudora book.* *Once you have collected three pendants you can grab it from it's pedestral.*

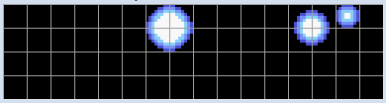
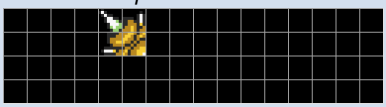
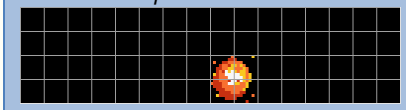
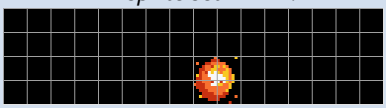
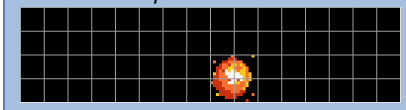
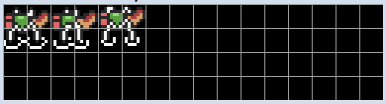
HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
63 - SandCrab1 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***10***	***9***	N/A	*Comes up and back under the ground.*
64 - SandCrab2 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***10***	***9***	N/A	*Comes up and back under the ground and also shoots fireballs.*

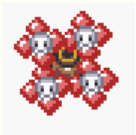
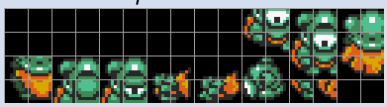
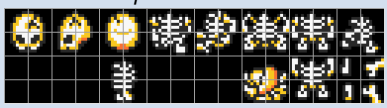
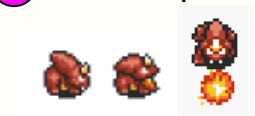
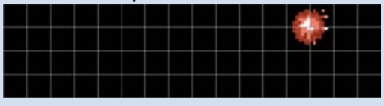
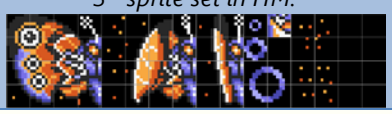
65 - ArcherGame  (273)	*1st sprite set in HM:* 	*Person*	***Dungeons***	***5***	***30***	***???***	
66 - Canon (Right) 	*1st sprite set in HM:* 	*Element*	***Dungeons***	***10***	***9***	N/A	*Shoots cannonballs from west to east*
67 - Canon (Left) 	*1st sprite set in HM:* 	*Element*	***Dungeons***	***10***	***9***	N/A	*Shoots cannonballs from east to west*
68 - Canon (Down) 	*1st sprite set in HM:* 	*Element*	***Dungeons***	***10***	***9***	N/A	*Shoots cannonballs from north to south*
69 - Canon (Up) 	*1st sprite set in HM:* 	*Element*	***Dungeons***	***10***	***9***	N/A	*Shoots cannonballs from south to north*
6A - MorningStar 	*1st and 2nd sprite sets in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***4*** ***1***	***1*** ***3***	N/A	*Swings its ball and chain over his head at you.*
6B - CannonSoldier 	*1st and 2nd sprite sets in HM:* 	*Monster*	***Dungeons*** ***Overworlds***	***4*** ***1***	***1*** ***3***	N/A	*Unused in the original game*
6C - Teleport?		*Warp*	***Overworlds***	**Any**	**Any**	N/A	*Warp Vortex created by Magic Mirror*
6D - Rat	*3rd sprite set in HM:*	*Monster*	***Dungeons***	***1***	***1***	N/A	*It only made the squeak noise in the Light World.*

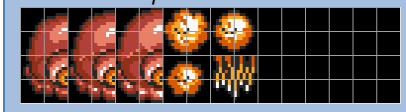
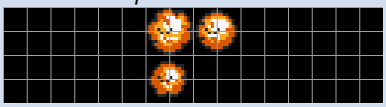
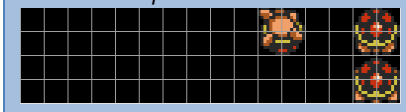
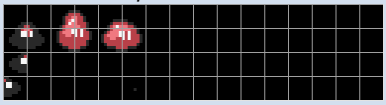
							
6E - Rope 	*3rd sprite set in HM:* 	*Monster*	***Light World*** ***Dark World***	***1***	***1***	N/A	*Comes in two versions, one for each world.*
6F - Keese 	*3rd sprite set in HM:* 	*Monster*	***Light World*** ***Dark World***	***1***	***1***	N/A	*Comes in two versions, one for each world.*
70 - 70 	*3rd and 4th sprite sets in HM:* 	*Boss Animation*	***Dungeons***	***21***	***16***	N/A	*Splitting Fireballs from Helmasaur King*
71 - Leever 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***10***	***9***	N/A	*They come in and out of the ground.*

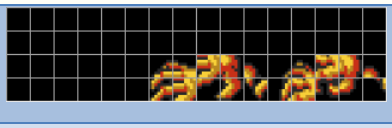
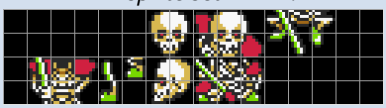
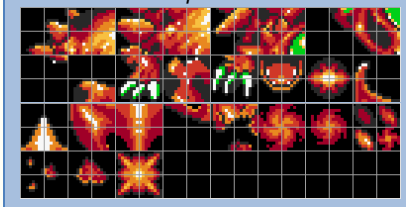
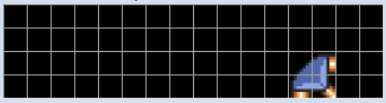
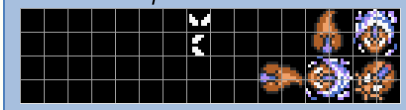
72 - Pond  (276, 277)  (278)	*4th sprite set in HM:*  *4th sprite set in HM:* 	*Person*	***Dungeons***	***7*** ***7***	***7*** ***34***	**???** **???**	*Magic pond where you throw in items in exchange of better ones.* *The pond in room 276 gives you a better boomerang*  *The one in room 277 upgrades your maximum number of bombs and arrows*  *The fat fairy (room 278) let's you upgrade your sword to lvl4 and also upgrades your arrows to silver arrows* 
73 - Priest/Uncle Also that barrier that opens in the sanctuary 	*1st sprite set in HM:*  *????* *uncle*	*Person*	***Dungeons (Priest and Barrier)*** ***Dungeons (Uncle)***	***6*** ***??***	***29*** ***??***	**Priest:** **From 022, Until 027** **Uncle:** **013, 014**	*Link's wounded uncle gives him his sword (lvl1) and shield (lvl1)*  **???**
74 - Runner 	*1st and 4th sprite sets in HM:* 	*Person*	***Overworlds***	***6***	***1***	**???**	*Runs away from you, tells you something if you catch him.*

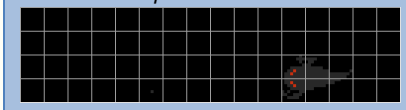
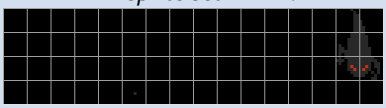
75 - BottleMan 	*3rd sprite set in HM:* 	*Person*	***Overworlds***	***6***	***1***	**From 209, Until 212**	*He sells you a bottle.*
76 - Zelda 	*Graphics loaded in memory:*	*Person*	***Dungeons***	**Any**	**Any**	**From 028, until 042**	
77 - WierdBuble? 	*4th sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***10***	N/A	*This sprite is invincible (magic powder won't turn it into a fairy).*
78 - OldWoman *Better name: Village Elder Wife* 	*1st and 3rd sprite sets in HM:* 	*Person*	***Dungeons***	***5***	***2***	From 043, until 046	*Sahasrahla's wife*
79 - Bee 	*Graphics loaded in memory:*			**Any**	**Any**	N/A	*Place this in a tree or on top of a lantern. When dashed into, bees come out.*
7A - Agahnim 	*All four sprite sets in HM:* 	*Boss*	***Dungeons***	***24***	***27***	**???**	

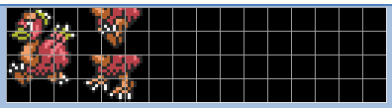
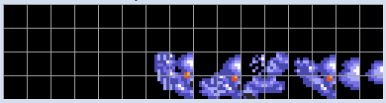
7B - OneShotMagicBal 	*4th sprite set in HM:* 	*Boss Animation*	***Dungeons***	***24***	***27***	N/A	*Agahnim energy blasts (not the duds)* *It sits there until you hit it*
7C - StalfosHead 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***11***	N/A	*Floating head that follows you*
7D - BigSpikeBlock 	*4th sprite set in HM:* 	*Element*	***Dungeons***	***12***	***26***	N/A	*Big 4 way moving block.* *2nd on list in HM*
7E - FireBlade 	*1st sprite set in HM:* 	*Element*	***Dungeons***	***17***	***10***	N/A	*A rotating line of fireballs* ***Goes clockwise or not?!***
7F - FireBlade2 	*1st sprite set in HM:* 	*Element*	***Dungeons***	***17***	***10***	N/A	*A rotating line of fireballs* ***Goes clockwise or not?!***
80 - Lanmola 	*1st sprite set in HM:* 	*Element*	***Dungeons***	***17***	***10***	N/A	*Line of fireballs that travels on the ground*
81 - WaterBug 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***10***	N/A	*Place sprite in shallow or deep water for full effect*

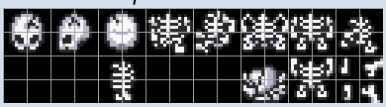
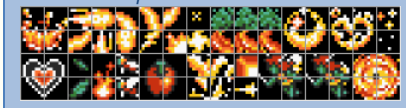
82 - 4Bubbles 	*4th sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***10***	N/A	*All 4 split apart as soon as you walk by. Magic powder does not turn them into fairies*
83 - GreenRocklops  	*3rd sprite set in HM:*  *2nd sprite set in HM:* 	*Monster*	***Dungeons*** ***Dungeons***	***10*** ***8***	***9*** ***39***	N/A	*Comes in two different forms, both need arrows to be killed.* *The second form shoot fireballs.*
84 - RedRocklops  	*3rd sprite set in HM:*  *2nd sprite set in HM:* 	*Monster*	***Dungeons*** ***Dungeons***	***10*** ***8***	***9*** ***39***	N/A	*Comes in two different forms, both need arrows to be killed.* *The second form shoot fireballs.*
85 - BigSpikeBlock Better name: Yellow Stalfos 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***11***	N/A	*Drops to the ground, dislodges head.*
86 - Triceritops 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***25***	***6***	N/A	
87 - FireKeese? Better name: Triceritops Flames 	*3rd sprite set in HM:* 	*Monster Animation*	***Dungeons***	***25***	***6***	N/A	*Rotates in a static position*
88 - Mothula	*3rd sprite set in HM:* 	*Boss*	***Dungeons***	***26***	***14***	N/A	

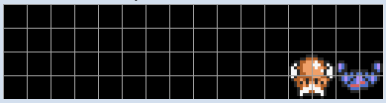
							
89 - 89 Better name: Mothula Ring 	*3rd sprite set in HM:* 	*Boss Animation*	***Dungeons***	***26***	***14***	N/A	*Rotates in a static position and hurts you if you touch it.* *Can't be killed.*
8A - SpikeBlock 	*4th sprite set in HM:* 	*Element*	***Dungeons***	***8***	***15***	N/A	*Spike moves left and right*
8B - Gibdo 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***19***	***13***	N/A	*It walks around and follows you.*
8C - Arrghus 	*3rd sprite set in HM:* 	*Boss*	***Dungeons***	***20***	***8***	N/A	
8D - ArrghusFuzz 	*3rd sprite set in HM:* 	*Boss Animation*	***Dungeons***	***20***	***8***	N/A	*These fly around Arrghus.*
8E - Shell 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***25***	***15***	N/A	*These are the turtles you hit with the hammer.*
8F - Blob 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***8***	N/A	*Pops out of ground and jumps at you.*
90 - WallMaster	*3rd sprite set in HM:*	*Monster*	***Dungeons***	***19***	***13***	N/A	

							
91 - StalfosKnight 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***28***	***19***	N/A	*Skeleton crushes down when you hit it with your sword.* *You then need a bomb to kill it.*
92 - Helmasaur 	*3rd and 4th sprite sets in HM:* 	*Boss*	***Dungeons***	***21***	***16***	N/A	
93 - RedOrb Better name: Blue bumper 	*4th sprite set in HM:* 	*Barrier*	***Dungeons***	***12***	***26***	N/A	*Link bounces back as he touches this sprite.* *You need the magic cape in order to get through it.*
94 - 94 Better name: RightEvil (*Overworlds*) 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***10***	N/A	*Looks like another Right Evil sprite. Plops down and wiggles.* *You need water pathways for them to work properly.* *Sprite usable in overworlds.*
95 - EyeLaser (Right) 	*4th sprite set in HM:* 	*Monster*	***Dungeons***	***29***	***17***	N/A	*Shoots from west to east*

96 - EyeLaser (Left) 	*4th sprite set in HM:* 	*Monster*	***Dungeons***	***29***	***17***	N/A	*Shoots from east to west*
97 - EyeLaser (Down) 	*4th sprite set in HM:* 	*Monster*	***Dungeons***	***29***	***17***	N/A	*Shoots from north to south*
98 - EyeLaser (Up) 	*4th sprite set in HM:* 	*Monster*	***Dungeons***	***29***	***17***	N/A	*Shoots from south to north*
99 - Penguin 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***28***	***19***	N/A	*Its runs then slides on floor*
9A - Splash 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***10***	N/A	*Moving Water Bubble* *Put water around it for full effect*
9B - Wizzrobe 	*3rd sprite set in HM:*  ***???***	*Monster*	***Dungeons***	***29*** ***??***	***17*** ***??***	N/A	*Comes in two different forms.*
9C - 9C Better name: *Vermin1 (Horizontal)* 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***10***	N/A	*Creature in the wall.* *It moves left and right.*
9D - VRat Better name: *Vermin2 (Vertical)* 	*2nd sprite set in HM:* 	*Monster*	***Dungeons***	***17***	***10***	N/A	*Creature in the wall.* *It moves up and down.*
9E - Ostrich	*3rd sprite set in HM:*	*Animal*	***Overworlds***	***15***	***1***	N/A	*The ostrich animal w/ the flute boy*

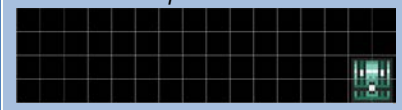
							*It just hops up and down. Doesn't run away.*
9F - Rabbit 	*3rd sprite set in HM:* 	*Animal*	***Overworlds***	***15***	***1***	N/A	*The rabbit animal w/ the flute boy* *It just stands there. Does not run away.*
A0 - Uglybird 	*3rd sprite set in HM:* 	*Animal*	***Overworlds***	***15***	***1***	N/A	*Birds w/ the flute boy* *It just stands there. Does not run away.*
A1 - IceMan 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***28***	***19***	N/A	
A2 - KholdStare 	*3rd sprite set in HM:* 	*Boss*	***Dungeons***	***22***	***20***	N/A	
A3 - A3 Better name: *Kholdstare Shell* 	*(The A3 sprite is invisible)*	*Boss Animation*	***Dungeons***	***22***	***20***	N/A	*Another part of Kholdstare, his shell in this case* *Invisible!*
A4 - A4 Better name: *Falling Iceballs* 	*3rd sprite set in HM:* 	*Boss Animation*	***Dungeons***	***22***	***20***	N/A	*Another part of Kholdstare, Ice balls from above* *This sprite needs* ***A2-KholdStare*** *Or* ***A3-A3*** *In order to work.*

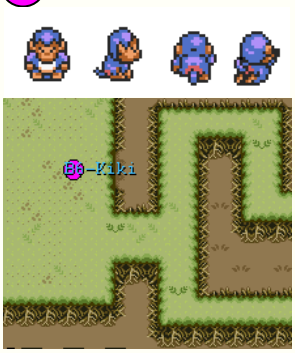
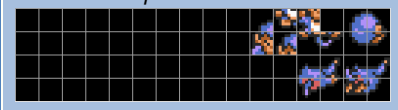
A5 - GreenLizard 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***27***	***23***	N/A	
A6 - RedLizard 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***27***	***23***	N/A	*He shoots fireballs*
A7 - Stalfos 	*1st sprite set in HM:* 	*Monster*	***Dungeons***	***8***	***11***	N/A	*Skeleton who jumps away from you*
A8 - GreenAirBomber 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***19***	***14***	N/A	
A9 - BlueAirBomber 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***19***	***14***	N/A	
AA - LikeLike 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***19***	***14***	N/A	*It steals your items away with its tongue.*
AB - AB Better name: *Crystal maidens* 	*???*	*Person*	***Dungeons***	**??**	**??**	**???**	*Girls inside the 7 crystals.* *Many things are messed up.*
AC - Apples 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 			**Any**	**Any**	N/A	Can be placed in a tree, under a bush/pot. Once you dash into it, apples are thrown on the ground.

AD - OldMan 	*3rd sprite set in HM:*  *Graphics loaded in memory:* ***[located near the end in YY-CHR]***	*Person*	***Dungeons***	***1***	***7***	***???***	*Gives you the mirror* *Needs a* ***??-person's door sprite*** *in order for him to stop following you.*
AE - DownPipe (rooms 20 and 21) 	N/A	*Trigger*	***Dungeons***	N/A	N/A	N/A	*Works only with a certain tileset.* *Freezes game in overworlds.*
AF - UpPipe (rooms 20 and 21) 	N/A	*Trigger*	***Dungeons***	N/A	N/A	N/A	*Works only with a certain tileset.* *Freezes game in overworlds.*
B0 - RightPipe (rooms 20 and 21) 	N/A	*Trigger*	***Dungeons***	N/A	N/A	N/A	*Works only with a certain tileset.* *Freezes game in overworlds.*
B1 - LeftPipe (rooms 20 and 21) 	N/A	*Trigger*	***Dungeons***	N/A	N/A	N/A	*Works only with a certain tileset.* *Freezes game in overworlds.*
B2 - Good-Bee	*Graphics loaded in memory:*			**Any**	**Any**	N/A	*It's a sparkly bee but it can still kill*

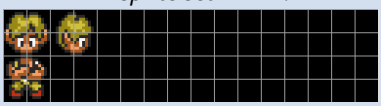
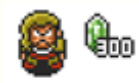
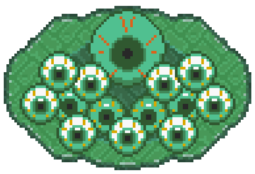
							*you. Place it inside something for it to work.*

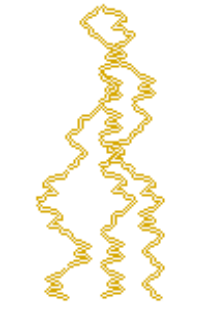
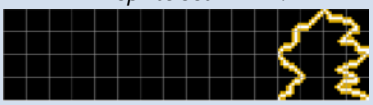
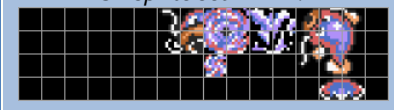
HM Name	B3 - Inscription Better name: *Hylian Writing*
GFX Tiles location	N/A
Type	Dialogue
Setting	***Overworlds***
GFX#	N/A
Pal	N/A
Monologue #	???
More info	 *The **B3 – Inscription** sprite in front of the MasterSword has two lines of dialogue, depending on whether or not you have the Mudora book.*  *Hylian Inscription near Desert Palace*

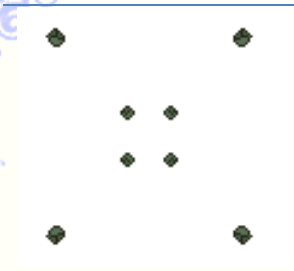
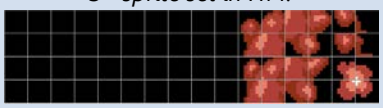
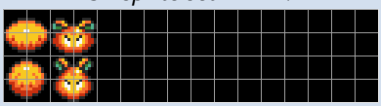
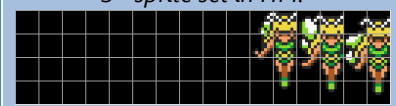
HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
B4 - BlueChest	*1st or 4th sprite sets in HM:* 	*Follower*	***Overworlds***	***21***	***16***		*Contains an empty bottle* *Can walk inside/out of a house with it.*

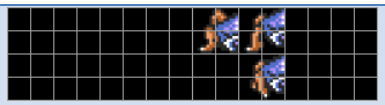
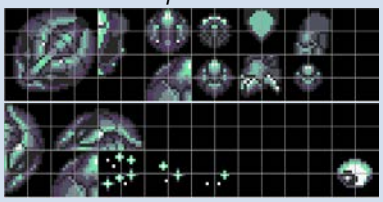
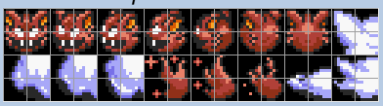
B5 - BombShop 	*2nd sprite set in HM:*  *Graphics loaded in memory:* ***red bomb!!***	*Person*	***Dungeons***	***5***	***31***	**???**	
B6 - Kiki 	*4th sprite set in HM:*  *Graphics loaded in memory:* ***kiki the monkey***	*Person*	***Overworlds***	***23***	***13***	**From 283, Until 289**	*Opens the first dark world dungeon for you.* *The **100 rupees** event to open the dungeon door won't appear if you didn't pay him the first **10 rupees**.*
B7 - BlindMan 	*Graphics loaded in memory:* ***blind maiden***	*Boss Addon*	***Dungeons***	***27***	***23***	**???**	*Maiden following you in Thieves Dungeon (turns into the Blind boss when exposed to sunlight).*
B8 - B8	*- Uses the priest graphics:*	***Degug Artifact*** link to it!!	**Any**	**Same as priest**	**Same as priest**	**???**	*Text Debugger*
B9 - Bully&Whimp(DW) 	*4th sprite set in HM:* 	*Person*	***Overworlds***	***20***	***12***	**From 347, Until 350**	*Big guy kicks the small guy around.*
BA - Whirlpool	*Graphics loaded in memory:*	*Teleport*	***Overworlds***	**Any**	**Any**	N/A	*Link it to whirlpools part*

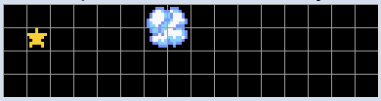
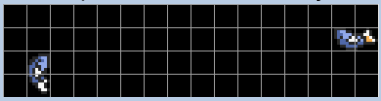
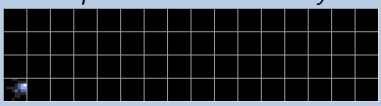
BB - ShopMan  (255, 274, 287)	*1st and 4th sprite sets in HM:* 	*Person*	***Dungeons***	***5***	***7***	***???***	*Vendor that sells red potion, an heart and bombs (rooms 255, 274 and 287)*
 (262)	*2nd sprite set in HM:* 	*Person*	***Dungeons***	***15***	***31***		*chestgame guy (room 262)*
 (271)	*1st, 2nd and 4th sprite sets in HM:* 	*Person*	***Dungeons***	***5***	***31***		*Vendor that sells red potion, a blue shield and bombs (room 271)*
 (272)	*1st, 2nd and 4th sprite sets in HM:* 	*Person*	***Dungeons***	***5***	***31***		*Vendor that sells a red shield, a golden bee and arrows (room 272)*
 (274)	*1st, 2nd and 4th sprite sets in HM:* 	*Person*	***Dungeons***	***5***	***7***		*Vendor that sells red potion, an heart and bombs (room 274)*

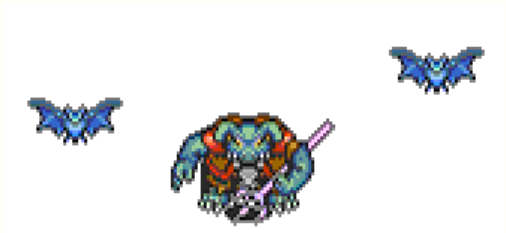
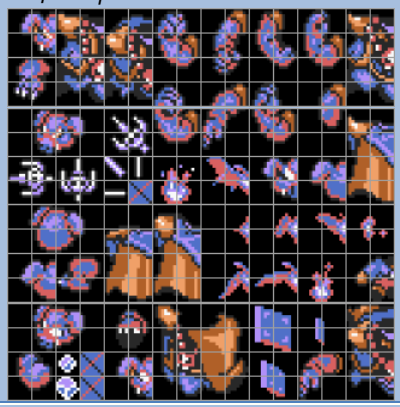
 (256)	*1st sprite set in HM:* 	*Person*	***Dungeons***	***40***	***5***		*Chest game thief (room 256)*
 (280)	*1st sprite set in HM:* 	*Person*	***Dungeons***	***15***	***5***		*Chest game (room 280)*
 (286)	*1st sprite set in HM:* 	*Person*	***Dungeons***	***42***	***32***		*300 rupee giver (room 286)*
 (291)	*1st sprite set in HM:* 	*Person*	***Dungeons***	***40***	***7***		*300 rupee giver (room 291)*
 (292)	*1st sprite set in HM:* 	*Person*	***Dungeons***	***40***	***7***		*thief (room 292)*
BC - OldMan2 	*1st and 3rd sprite sets in HM:* 	*Person*	***Dungeons***	***15***	***5***	***???***	*Drunk Guy in Bar*
BD - Viterous 	*4th sprite set in HM:* 	*Boss*	***Dungeons***	***22***	***18***	N/A	
BE - ??? Better name: *Viterous eyeballs*	*4th sprite set in HM:* 	*Boss Animation*	***Dungeons***	***22***	***18***	N/A	*Floats in the air in a static way*

							
BF - Lightning Better name: *Viterous lightning* 	*4th sprite set in HM:* 	*Boss Animation*	***Dungeons***	***22***	***18***	N/A	*It shoots lighting only once*
C0 - Item Better name: *Quake Medallion Monster* 	*3rd sprite set in HM:* 	*Person*	***Overworlds***	***22***	***10***	**???**	*There must have water in the area/dungeon room where you throw the bush/pot in exchange of the Quake medallion for that trigger to work.*
C1 - AgahTalk 	*1st, 3rd and 4th sprite sets in HM:* 	*Event*	***Dungeons***	***18***	***12***	**???**	*Agahnim teleporting Zelda to dark world, then vanishing.*
C2 - RockChip	*Graphics loaded in memory:* **???**	*Boss Animation*	***Dungeons***	***11***	***4***	N/A	*It's the tiny ground rocks in the lanmolas boss fight.* *In the overworlds it's an invincible monster.*

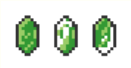
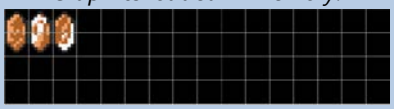
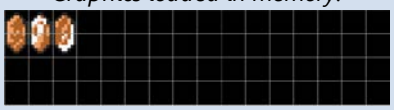
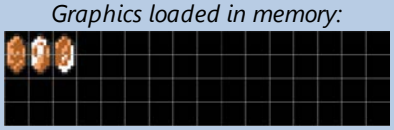
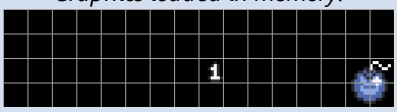
							
C3 - Half-Bubble 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***27***	***23***	N/A	
C4 - Bully 	*1st sprite set in HM:* 	*Monster*	***Light World*** ***Dark World***	***7*** ***??***	***5*** ***??***	N/A	*Steals rupees, bombs and arrows*
C5 - Shooter 	*Graphics loaded in memory:*	*Element*	***Dungeons***	**Any**	**Any**	N/A	*Spits fireballs randomly in your direction.*
C6 - 4WayShooter 	*Graphics loaded in memory:*	*Element*	***Dungeons***	**Any**	**Any**	N/A	*Spits fireballs when you use your sword (in four directions: up, down, left and right).*
C7 - FuzzyStack 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***30***	***24***	N/A	
C8 - BigFairy 	*3rd sprite set in HM:* 	*Person*	***Dungeons***	***7***	***7***	**???**	*Big fairy that refills your health meter.*
C9 - Tektite	*4th sprite set in HM:*	*Monster*	***Overworlds***	***5***	***7***	N/A	

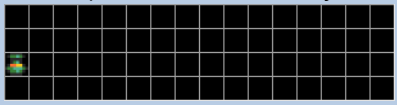
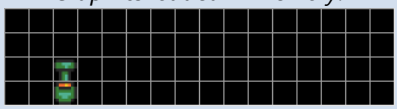
							
CA - Chomp 	*3rd sprite set in HM:* 	*Monster*	***Dungeons***	***30***	***24***	N/A	
CB - TriNexx1 	*1st and 4th sprite sets in HM:* 	*Boss*	***Dungeons***	***23***	***25***	N/A	*Final stage in TriNexx after both heads have been defeated.* *Sprite worked on LW and DW but they were glitched very badly.*
CC - TriNexx2		*Boss*	***Dungeons***	***23***	***25***	N/A	**Fire head or ice?**
CD - TriNexx3		*Boss*	***Dungeons***	***23***	***25***	N/A	**Fire head or ice?**
CE - Blind 	*3rd sprite set in HM:* 	*Boss*	***Dungeons***	***32***	***23***	N/A	
CF - SwampSnake 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***22***	***15***	N/A	*Snake comes in and out of water.*
D0 - Lynel 	*4th sprite set in HM:* 	*Monster*	***Overworlds***	***20***	***12***	N/A	*Lion shoots fireballs at you.*

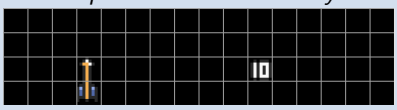
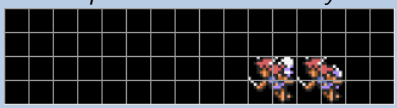
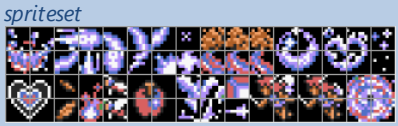
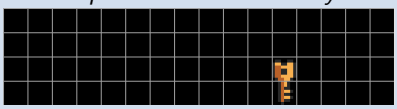
D1 - Transform/Smoke 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	*Transform Link into a Bunny temporarily. (Dungeons)* *Blacksmith chemney's smoke animation. (Overworlds)*
D2 - Fish 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Overworlds***	**Any**	**Any**	N/A	*Flopping fish if placed on ground. (that you can even pick up)* *Or if placed in water, a fish will jump if you trow a pot or bush in it.*
D3 - AliveRock 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Overworlds***	**Any**	**Any**	N/A	*The rock that turns into a hopping skull.* *Can use any palette/gfx# as long as it's located in the Dark World.*
D4 - GroundBomb 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Overworlds***	**Any**	**Any**	N/A	*Bomb implanted into the ground.* *This sprite does activate pegswitches.*
D5 - DiggingGameGuy 	*2nd sprite set in HM:* 	*Person*	***Overworlds***	***27***	***18***	**???**	*On the interior you can dig but you don't get any rupees because the floor is not soil.*

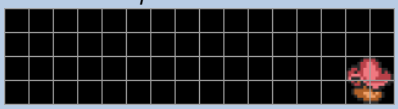
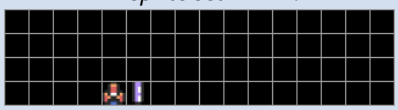
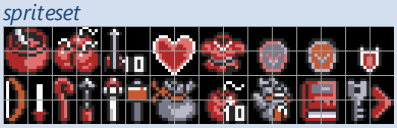
HM Name	D6 - Ganon 
GFX Tiles location	*All four sprite sets in HM:* 
Type	*Boss*
Setting	***Dungeons***
GFX#	***34***
Pal	***33***
Monologue #	**367 and 368**
More info	*Final boss in the game. In order to make the sprite work properly, it needs to be in room 0 along with two room properties. You need to have the effect set to "Ganon room" aswell as tag1 set to "Kill to open Ganon's door". You can enter the room by either using "Hole 7B" or "Entrance 7B".*

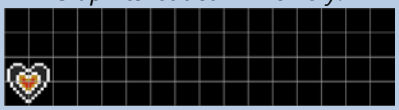
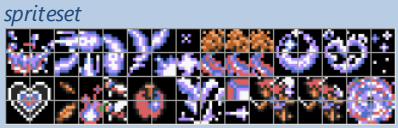
HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
D7 - D7 	*Same as **D6 - Ganon***	*Boss*	***Dungeons***	***34***	***33***	**367 and 368**	Copy of Ganon, except invincible?

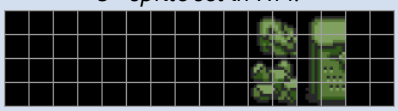
D8 - Heart *[2nd on list in HM]* 	*Graphics loaded in memory:*		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Heart refill In dungeons, it's a heart that sits on the ground. In Overworlds it's a bomb. The bomb activates pegswitches.
D9 - Heart *[1st on list in HM]* Better name: *Green Rupee* 	*Graphics loaded in memory:*  *- they are located within the inventory items animations spritesets* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	*Green rupee* Sits on ground in dungeons. Must place it inside something on overworlds.
DA - BlueRupee 	*Graphics loaded in memory:*  *- they are located within the inventory items animations spritesets* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	*Blue rupee* Sits on ground in dungeons. Must place it inside something on overworlds.
DB - InTree?Rocks? Better name: *Red Rupee* 	*Graphics loaded in memory:*  *- they are located within the inventory items animations spritesets* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	*Red rupee* Sits on ground in dungeons. Must place it inside something on overworlds.
DC - Bomb 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Bomb Refill (1) Sits on ground in dungeons. Must place it inside something on overworlds.

DD - 4 bombs 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Bomb Refill (4) Sits on ground in dungeons. Must place it inside something on overworlds.
DE - 8 bombs 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Bomb Refill (8) Sits on ground in dungeons. Must place it inside something on overworlds.
DF - Magic 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Small Magic Refill Sits on ground in dungeons. Must place it inside something on overworlds.
E0 - BigMagic 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Full Magic Refill Sits on ground in dungeons. Must place it inside something on overworlds.

E1 - Arrow 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Arrow Refill (5) Sits on ground in dungeons. Must place it inside something on overworlds.
E2 - E2 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Arrow Refill (10) Sits on ground in dungeons. Must place it inside something on overworlds.
E3 - Fairy 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	A small fairy floats around in dungeons. Must place it inside something on overworlds.
E4 - Key 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons***	**Any**	**Any**	N/A	Small Key Does not work if placed inside something
E5 - E5 	*Graphics loaded in memory:*		***Dungeons***	**Any**	**Any**	N/A	Big Key Does not work if placed inside something

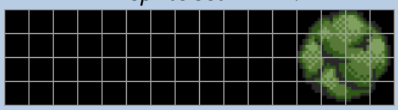
E6 - E6		*Effect*				N/A	Shield-Eater? *Removes Shield (once you pick it up)*
E7 - Mushroom 	*4th sprite set in HM:* 	*Object*	***Overworlds***	***7***	***5***	**???**	*It also works in the dungeons.*
E8 - FakeSword 	*4th sprite set in HM:* 	*Object*	***Overworlds***	***7***	***5***	**???**	*It's a fake master sword that you can pick up and throw.*
E9 - ShopMan2 	*1st and 4th sprite sets in HM:* 	*Person*	***Dungeons***	***5***	***5***	**From 077, Until 080**	*Magic Shop vendor,*  *his items,*  *and the the magic powder*  *(it spawns sometime after you've given a mushroom to the witch)*
EA - WitchAssistant Better name: *Full Heart Container* 	*Graphics loaded in memory:*  *- they are located within that inventory spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Can be placed in a tree, under a bush/pot or lay on the ground.

EB - HeartPie Better name: *Quarter Heart Container* 	*Graphics loaded in memory:*  *- they are located within these animations spriteset* 		***Dungeons*** ***Overworlds***	**Any**	**Any**	N/A	Can be placed in a tree, under a bush/pot or lay on the ground.
EC - PickedObj Better name: *Bush (thrown)*	*Graphics loaded in memory:*			**Any**	**Any**	N/A	*It's the animation of a bush that's being thrown.*
ED - ED Better name: *Somaria platform* 			***Dungeons***			N/A	*Crashes the game if placed in either/both dungeons and overworlds.*
EE - Mantle 			***Dungeons***	***3***	***0***	N/A	*Movable Mantle (in Hyrule Castle).* *Can only be pushed once and with Zelda along your side.*
EF - EF 			***Dungeons***			N/A	*Somaria platform* *Crashes the game if placed in either/both dungeons and overworlds.*
F0 - F0 			***Dungeons***			N/A	*Somaria platform* *Crashes the game if placed in either/both dungeons and overworlds.*
F1 - F1 			***Dungeons***			N/A	*Somaria platform* *Crashes the game if placed in either/both*

							dungeons and overworlds.
F2 - MedallianTablet 	3rd *sprite set in HM:* 	*Overlord*	***Overworlds***	***8***	***4***	***???***	*gives Bombos in every overworld area, except for **Area 03**, in which it gives Ether.* *Hold up your Master Sword to get your spell.*

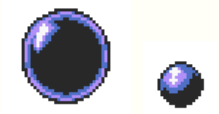
HM Name	F3 - Person'sDoor
GFX Tiles location	N/A
Type	*Overlord*
Setting	***Overworlds***
GFX#	N/A
Pal	N/A
Monologue #	***???*** *(cant enter cave with something following you message)*
More info	   *The **ScaredGirl1** and **ScaredGirl2** sprites are running away in the houses where the **F3 - Person'sDoor** sprite are positioned to.* *The old man on the mountain **(sprite ??)** will not enter any caves with you, but after you meet him in a cave you can exit them, but not enter again. A sprite called **F3 - Person'sDoor** is needed to end that event with that particular message and giving you the mirror.*

HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info

F4 - FallingRocks	*4th sprite set in HM:* 	*Overlord*	*Overworlds only* *(crash in dungeons)*	***5***	***7***	N/A	*Rocks are falling through the screen in the bottom part of a large area. In order to make it work, you need to place the sprite in the top left corner of a large overworld area.* *It won't work on smaller overworld screens.*
F5 - F5 Better name: *CannonBalls (Overworlds)* 		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	*In DW, cannon balls fly down a vertical path. Like in the very first dungeon.*
F6 - F6		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	
F7 - F7		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	
F8 - F8		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	
F9 - F9		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	

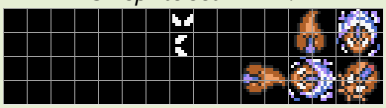
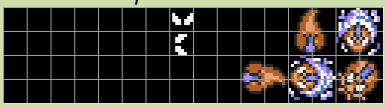
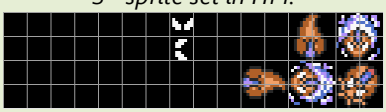
FA - FA Better name: *Blobs drop* (*Overworlds*) 		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	*Endless falling blobs from the ceiling.*
FB - FB Better name: *Wallmaster* (*Overworlds*) 		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	*Wallmaster that puts you in chris houlihan room but in total darkness*
FC - FC Better name: *FloorDrop Square* (*Overworlds*)		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	*Floor tiles fall down in a square pattern.*
FD - FD Better name: *FloorDrop North* (*Overworlds*) 	???	*Overlord*	*Overworlds only* *(crash in dungeons)*	N/A	N/A	N/A	*Floor tiles fall down in vertically upwards,*
FE - FE Better name: *FloorDrop East* (*Overworlds*) 	???	*Overlord*	*Overworlds only* *(crash in dungeons)*	N/A	N/A	N/A	*Floor tiles fall down horizontally to the right.*
FF - FF		*Overlord*	*Overworlds only* *(crash in dungeons)*			N/A	*Unknown properties.*

b) Sprites usable in dungeons only

HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
00 - CannonRoom		*Overlord*	***Dungeons***	***8***	***11***	N/A	
01 - 01		*Overlord*	***Dungeons***				
02 - CannonRoom		*Overlord*	***Dungeons***	***8***	***11***	N/A	
03 - CannonBalls 		*Overlord*	***Dungeons***	***8***	***11***	N/A	
04 - RopeDrp(Snake) Better name: *Stalfos Heads Drop* 		*Overlord*	***Dungeons***	***8***	***11***	N/A	*Stalfos drop down, their heads flying at you*
05 - StalfosDrop 		*Overlord*	***Dungeons***	***8***	***11***	N/A	
06 - BombDrop Better name: *BombDrop1*	*Graphics loaded in memory:*	*Overlord*	***Dungeons***	**Any**	**Any**	N/A	*The first of two BombDrops sprites that appears in the list in Hyrule Magic.*
07 - MovingFloor	N/A	*Overlord*	***Dungeons***	**Any**	**Any**	N/A	
08 - Transformer Better name: *Blobs drop* 		*Overlord*	***Dungeons***	***28***	***17***	N/A	*It's the blobs that fall from the ceiling in the room right before the boss of Misery Mire.*

HM Name	09 - WallMaster Better name: *WallMaster spawner* 
GFX Tiles location	
Type	*Overlord*
Setting	***Dungeons***
GFX#	
Pal	
Monologue #	N/A
More info	***Picture*** *You need to change your room settings to a dungeon (bottom left area of the dungeon editor window)* ***Picture*** *and then edit the starting room location of that same dungeon in the dungeon properties menu in HyruleMagic or else they won't send you back to the entrance of your dungeon and won't work properly!*

HM Name	GFX Tiles location	Type	Setting	GFX#	Pal	Monologue	More Info
0A - FloorDrop(Sqr) Better name: *FloorDrop Square (Dungeons)*		*Overlord*	***Dungeons***	***12***	***26***	N/A	*makes the tiles drop 1 by 1 in the shape of a square*
0B - FloorDrop(Vert) Better name: *FloorDrop North (Dungeons)* 		*Overlord*	***Dungeons***	***12***	***26***	N/A	*makes the floor tiles drop 1 by 1 in a vertical rectangle upwards*
0C - FloorDrop Better name: *FloorDrop East (Dungeons)*		*Overlord*	***Dungeons***	***12***	***26***	N/A	*drops floor tiles down 1 by 1 in a rectangle shape horizontally to the right*

							
0D - FloorDrop Better name: *FloorDrop South (Dungeons)* 		*Overlord*	***Dungeons***	***12***	***26***	N/A	*drops floor tiles down 1 by 1 in a rectangle shape vertically downwards*
0E - FloorDrop Better name: *FloorDrop West (Dungeons)* 		*Overlord*	***Dungeons***	***12***	***26***	N/A	*drops floor tiles down 1 by 1 in a rectangle shape horizontally to the left*
0F - FloorDrop		*Overlord*	***Dungeons***	***12***	***26***	N/A	*drops floor tiles down 1 by 1 in a rectangle shape vertically upwards*
10 - RightEvil 	*3rd sprite set in HM:* 	*Overlord*	***Dungeons***	***17***	***10***	N/A	*you need water pathways for them to work*
11 - LeftEvil 	*3rd sprite set in HM:* 	*Overlord*	***Dungeons***	***17***	***10***	N/A	
12 - DownEvil 	*3rd sprite set in HM:* 	*Overlord*	***Dungeons***	***17***	***10***	N/A	
13 - UpEvil 	*3rd sprite set in HM:* 	*Overlord*	***Dungeons***	***17***	***10***	N/A	

14 - FloorTiles 		*Overlord*	***Dungeons***	***10***	***9***	N/A	
15 - WizzrobeSpawn 		*Overlord*	***Dungeons***	***29*** ***??***	***17*** ***??***	N/A	*Place in center of your room, four of them spawn at the same time*
16 - MiniBats 		*Overlord*	***Dungeons***	***28***	***17***	N/A	
17 - PotTrap Better name: *PotSwitch*		*Overlord*	***Dungeons***			N/A	*floor switch trigger*
18 - StalfosAppear		*Overlord*	***Dungeons***	***8***	***11***	N/A	*May have to place sprite in center of room to work*
19 - ?ArmosKnights?		*Overlord*	***Dungeons***	***9***	***11***	N/A	*Armos Knights Handler* 
1A - BombDrop Better name: *BombDrop2*	*Graphics loaded in memory:*	*Overlord*	***Dungeons***	**Any**	**Any**	N/A	*Second bomb drop on the list.*
1B - 11B		*Unknown*	***Dungeons***			N/A	*Unknown properties.* *Crashes the game.*

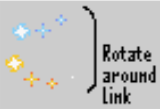
c) Special Sprites Routines

MathOnNapkins: *"At address **$4037F-$40404** there's a jump table for many special sprites (you can't add them with Hyrule Magic)."*

Note: *Parameter A is actually ACC+1, since 0 would indicate NO EFFECT.*

Source : $0C4A, X

Source	Hex Adress	A=	Picture	Description
dw $851B	$4051B*	1		????
dw $86D2	$406D2*	2		fire rod flame (both flying and after hitting something)
dw $8852	$40852*	3		(RTS)
dw $8D19	$40D19*	4		Master sword beam dispersing after hitting something
dw $90FC	$410FC*	5		boomerang
dw $93E8	$413E8*	6		Spark effect that occurs when you hit a wall with a boomerang or hookshot
dw $955A	$4155A*	7		blue bomb
dw $9FB6	$41FB6*	8		rock fall effect (from bombing a cave)
dw $A131	$42131*	9		Flying arrow
dw $A45B	$4245B*	A		Arrow stuck in something (wall or sprite)
dw $A4DD	$424DD*	B		Ice Rod shot
dw $DDC5	$45DC5*	C		Master sword beam
dw $DDCA	$45DCA*	D		The sparkle at the tip of your sword when you power up the spin attack
dw $A60E	$4260E*	E		????
dw $A60E	$4260E*	F		????
dw $A60E	$4260E*	10		????
dw $A536	$42536*	11		Ice rod shot exploding after hitting a surface
dw $A60E	$4260E*	12		????
dw $8435	$40435*	13		????
dw $A80D	$4280D*	14		CONDITIONAL (THIS SEEMS LIKE AN ERROR). SHOULD BE SAME AS $4280F, I THINK.
dw $A80F	$4280F*	15		Splash from jumping into or out of deep water
dw $A8E5	$428E5*	16		The Hammer's Stars / Stars from hitting hard ground with the shovel
dw $A9A9	$429A9*	17		Dirt from digging a hole with the shovel
dw $AAA0	$42AA0*	18		The Ether Effect
dw $B0CE	$430CE*	19		The Bombos Effect
dw $BAB0	$43AB0*	1A		Precursor to torch flame / Magic powder
dw $93FF	$413FF*	1B		Sparks from tapping a wall with your sword
dw $B66A	$4366A*	1C		The Quake Effect
dw $BBBC	$43BBC*	1D		Jarring effect from hitting a wall while dashing
dw $????	$43C92*	1E		Pegasus boots dust flying
dw $BD74	$43D74*	1F		Hookshot

dw $C013	$44013*	20		Link's Bed Spread
dw $C094	$44094*	21		Link's Zzzz's from sleeping
dw $????	$?????*	22		Received Item Sprite
dw $D3BC	$453BC*	23		Bunny / Cape transformation poof
dw $EE01	$46E01*	24		Gravestone sprite when in motion
dw $A8E3	$428E3*	25		????
dw $D65A	$4565A*	26		Sparkles when swinging lvl 2 or higher sword
dw $DDE8	$45DE8*	27		the bird (when called by flute)
dw $C6F2	$446F2*	28		item sprite that you throw into magic faerie ponds.
dw $CA8C	$44A8C*	29		Pendants and crystals
dw $D7B2	$457B2*	2A		Start of spin attack sparkle
dw $D8FD	$458FD*	2B		During Spin attack sparkles
dw $E365	$46365*	2C		Cane of Somaria blocks
dw $E9E8	$469E8*	2D		????
dw $EB3E	$46B3E*	2E		????
dw $EC13	$46C13*	2F		Torch's flame
dw $DB24	$45B24*	30		Initial spark for the Cane of Byrna activating
dw $DC70	$45C70*	31		Cane of Byrna spinning sparkle
dw $AA35	$42A35*	32		Flame blob, possibly from wall explosion
dw $A60E	$4260E*	33		Series of explosions from blowing up a wall (after pulling a switch)
dw $EF9A	$46F9A*	34		Burning effect used to open up the entrance to skull woods.
dw $C25F	$4425F*	35		Master Sword ceremony.... not sure if it's the whole thing or a part of it
dw $CFAA	$44FAA*	36		Flute that pops out of the ground in the haunted grove.
dw $D03D	$4503D*	37		Appears to trigger the weathervane explosion.
dw $D1D8	$451D8*	38		Appears to give Link the bird enabled flute.
dw $EA83	$46A83*	39		Cane of Somaria blast which creates platforms (**sprite 0xED**)
dw $F18D	$4718D*	3A		super bomb explosion (also does things normal bombs can)
dw $C167	$44167*	3B		Unused hit effect. Looks similar to Somaria block being nulled out.

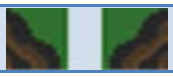
dw $C1EA	$441EA*	3C		Sparkles from holding the sword out charging for a spin attack.
dw $CA01	$44A01*	3D		splash effect when things fall into the water
dw $CBF2	$44BF2*	3E		3D crystal effect (or transition into 3D crystal?)
dw $D519	$45519*	3F		Disintegrating bush poof (due to magic powder)
dw $D49A	$4549A*	40		Dwarf transformation cloud
dw $ECAF	$46CAF*	41		Water splash in the waterfall of wishing entrance (and swamp palace)
dw $C7DE	$447DE*	42		Rupees that you throw in to the Pond of Wishing
dw $CCA0	$44CA0*	43		Ganon's Tower seal being broken. (not opened up though!)

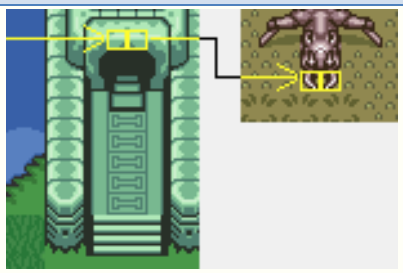
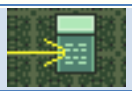
5) Blocks Types Database

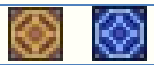
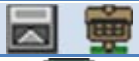
by MathOnNapkins and Orochimaru

Please use caution while editing these as they are shared within all areas/rooms.

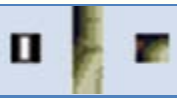
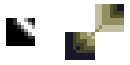
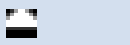
HEX	DEC	Picture	Description
00	000		Can walk **over** or **under** (*Depending on the "In front" setting which is either **off** or **on***)
01	001		Solid (*Nothing can go through*)
02	002		Solid (*Arrows, bombs, etc... can go through*)
03	003		Solid (*Unknown properties*)
04	004		Walking in tall grass
05	005		*Same properties as **00***
06	006		*Same properties as **00***

07	007		*Same properties as **00***
08	008		Deep water (*swimmable*)
09	009		Shallow water (*which you can walk on*) [*Also used for a secret waterfall entrance*]
0A	010		*Same properties as **00***
0B	011		*Same properties as **00***
0C	012		Moving floor (*Mothula's room*)
0D	013		Spike floor (*hurts as you walk on it*)
0E	014		Slippery floor SLOWWW.. (*shiny ice*)
0F	015		Slippery floor FASTERRR.. (*non-shiny ice*)
10	016		Diagonal Edge
11	017		Diagonal Edge
12	018		Diagonal Edge
13	019		Diagonal Edge
14	020		*Same properties as **00***
15	021		*Same properties as **00***
16	022		*Same properties as **00***
17	023		*Same properties as **00***
18	024		Diagonal Edge
19	025		Diagonal Edge
1A	026		Diagonal Edge
1B	027		Diagonal Edge
1C	028		Top of water staircase
1D	029		Stairs (Link walk slower on these)
1E	030		*Same properties as **1D***
1F	031		*Same properties as **1D***
20	032	 / 	Pit *(Dungeons)* / Holes *(Overworlds)* *An additional item* (**80 - Hole**) *is needed for these to work properly on the overworlds.*
21	033		*Same properties as **00***
22	034		*Same properties as **1D***
23	035		Lower half of trigger tile (*Object **1.1.0x35** also uses it, but it seems like a broken mess*)
24	036		Upper half of trigger tiles
25	037		*Same properties as **00***
26	038		Boundary tile for In-floor inter-room staircases
27	039		Solid (*can Hookshot*)
28	040		Ledge leading up
29	041		Ledge leading down
2A	042		Ledge leading left

2B	043		Ledge leading right
2C	044		Ledge leading up + left
2D	045		Ledge leading down + left
2E	046		Ledge leading up + right
2F	047		Ledge leading down + right
30	048		Up Staircase to room 1 of 5
31	049		Up Staircase to room 2 (")
32	050		Up Staircase to room 3 (")
33	051		Up Staircase to room 4 (")
34	052		Down Staircase to room 1 of 5
35	053		Down Staircase to room 2 (")
36	054		Down Staircase to room 3 (")
37	055		Down Staircase to room 4 (")
38	056		Boundary tile for straight up inter-room staircases
39	057		Boundary tile for sraight up inter-room staircases
3A	058		????
3B	059		Star tiles that change up the floor
3C	060		????
3D	061		Inter-floor staircases?
3E	062		Inter-floor staircases?
3F	063		Inter-floor staircases?
40	064		Thick Grass (*and smashed in moles?*)
41	065		????
42	066		Gravestone
43	067		- Tiles used in two very specific dungeon entrances: - **One in Area 3B** (*Entrance 4B ~ that triggers the watergate*). - **The other is in Area 40** (*Entrance 2B*) which requires the firerod to trigger the entrance animation.
44	068		Spike Block *(Dungeons)* / Cactus *(Overworlds)* *- Damages Link each time he touch those* *- You can only travel across that tile type using the hookhot*
45	069		????
46	070		Book of Mudora trigger (Desert dungeon blocked entrance)
47	071		????
48	072		Normal blank ground *(Overworlds)* [*Mirror of 0x00*]
49	073		????
4A	074		????

4B	075		Orange Warp Tile *(Dungeons)* / Blue Warp Tile *(Overworlds)*
4C	076		????
4D	077		????
4E	078		Mountain rock tile found in only a few select areas
4F	079		Mountain rock tile found in only a few select areas
50	080		Green Bush *(Overworlds)*
51	081		Yellow Bush *(Overworlds)*
52	082		Small Rocks Lvl 1 *(Overworlds)*
53	083		Small Rocks Lvl 2 *(Overworlds)*
54	084		Telepathic tiles *(Dungeons)* / Signs *(Overworlds)*
55	085		Big Rocks Lvl 1 (*White*) *(Overworlds)*
56	086		Big Rocks Lvl 2 (*Black*) *(Overworlds)*
57	087		Rock pile (*you can destroy these with your boots*) *(Overworlds)*
58	088		chest 0
59	089		chest 1
5A	090		chest 2
5B	091		chest 3
5C	092		chest 4
5D	093		chest 5
5E	094		upward staircase tile
5F	095		????
60	096		Blue Rupee
61	097		????
62	098		Bombable cracked floor
63	099		minigame chests?
64	100		????
65	101		????
66	102		blue / orange block that is down
67	103		blue / orange block that is up
68	104		conveyor belt that goes up
69	105		conveyor belt that goes down
6A	106		conveyor belt that goes left

6B	107		conveyor belt that goes right
6C	108		used for special dungeon doors (up and down)
6D	109		????
6E	110		used for special dungeon doors (sideways)
6F	111		????
70	112		pot or bush (or mole?)
71	113		pot or bush
72	114		pot or bush
73	115		pot or bush
74	116		pot or bush
75	117		pot or bush
76	118		pot or bush
77	119		pot or bush
78	120		pot or bush
79	121		pot or bush
7A	122		pot or bush
7B	123		pot or bush
7C	124		pot or bush
7D	125		pot or bush
7E	126		pot or bush
7F	127		pot or bush
80	128		open door?
81	129		????
82	130		????
83	131		????
84	132		????
85	133		????
86	134		????
87	135		????
88	136		????
89	137		????
8A	138		????
8B	139		????
8C	140		????
8D	141		????
8E	142		send player to overworld? (well supported so far >_>)
8F	143		????
90	144		screen transition with BG toggle (BG0 <-> BG1)
91	145		screen transition with BG toggle (BG0 <-> BG1)
92	146		screen transition with BG toggle (BG0 <-> BG1)
93	147		screen transition with BG toggle (BG0 <-> BG1)
94	148		screen transition with BG toggle (BG0 <-> BG1)
95	149		screen transition with BG toggle (BG0 <-> BG1)
96	150		screen transition with BG toggle (BG0 <-> BG1)
97	151		screen transition with BG toggle (BG0 <-> BG1)

98	152		????
99	153		????
9A	154		????
9B	155		????
9C	156		????
9D	157		????
9E	158		????
9F	159		????
A0	160		screen transition with dungeon toggle
A1	161		screen transition with dungeon toggle
A2	162		screen transition with dungeon toggle
A3	163		screen transition with dungeon toggle
A4	164		screen transition with dungeon toggle
A5	165		screen transition with dungeon toggle
A6	166		????
A7	167		????
A8	168		????
A9	169		????
AA	170		????
AB	171		????
AC	172		????
AD	173		
AE	174		
AF	175		
B0	176		Cane of Somaria line/Turtle Rock Pipe *(both going up or down)*
B1	177		Cane of Somaria line *(left/right)*
B2	178		line corner / pipe corner
B3	179		line corner / pipe corner
B4	180		line corner / pipe corner
B5	181		line corner / pipe corner
B6	182		Cane of Somaria line stopper ***(question mark shaped*)*** ***Note:** There's two identical gfx tiles with question marks in your tilesets. Be sure to not confuse them since they are not the same type of blocks. A good trick is to edit their graphics so that they don't look the same.*
B7	183		line 3-way (left, down,right)
B8	184		line 3-way (left, up, right)
B9	185		line 3-way (up, right, down)

BA	186		line 3-way (up, left, down)
BB	187		line 4-way
BC	188		stopper (4-way) ***(question mark shaped*)***
BD	189		insersection (crossover)
BE	190		UpPipe/LeftPipe/RightPipe/DownPipe **Tile trigger** (that sets you on your path)
BF	191		????
C0	192		Torch 0x00
C1	193		Torch 0x01
C2	194		Torch 0x02
C3	195		Torch 0x03
C4	196		Torch 0x04
C5	197		Torch 0x05
C6	198		Torch 0x06
C7	199		Torch 0x07
C8	200		Torch 0x08
C9	201		Torch 0x09
CA	202		Torch 0x0A
CB	203		Torch 0x0B
CC	204		Torch 0x0C
CD	205		Torch 0x0D
CE	206		Torch 0x0E
CF	207		Torch 0x0F
D0	208		????
D1	209		????
D2	210		????
D3	211		????
D4	212		????
D5	213		????
D6	214		????
D7	215		????
D8	216		????
D9	217		????
DA	218		????
DB	219		????
DC	220		????
DD	221		????
DE	222		????
DF	223		????
E0	224		????
E1	225		????
E2	226		????
E3	227		????
E4	228		????

E5	229		????
E6	230		????
E7	231		????
E8	232		????
E9	233		????
EA	234		????
EB	235		????
EC	236		????
ED	237		????
EE	238		????
EF	239		????
F0	240		Key door 1
F1	241		Key door 2
F2	242		Key door 3
F3	243		Key door 4
F4	244		????
F5	245		????
F6	246		????
F7	247		????
F8	248		????
F9	249		????
FA	250		????
FB	251		????
FC	252		????
FD	253		????
FE	254		????
FF	255		????

6) Entrances/Exits and HEX values list

by Orochimaru

The following Hex Offsets will work on a headered rom only.

Entrance#	Exit#	Hex Offset	Description
N/A	0003	N/A	Chris Houlihan's Room (Holes without item **80-hole** all lead to this room)
01	0104	15582	Link's House
02	0012	15583	Sanctuary
03	0060	15584	Beginning Dungeon (Hyrule Palace ~ Left Entrance)
04	0061	15585	Beginning Dungeon (Hyrule Palace ~ Main Entrance)
05	0062	15586	Beginning Dungeon (Hyrule Palace ~ Right Entrance)
06	00F0	15587	Cavern (Heading towards Death Mountain)
07	00F1	15588	Death Mountain Arrival (From Hyrule Field)
08	00C9	15589	1st LightWorld Dungeon (East Palace)
09	0084	1558A	2nd LightWorld Dungeon (Desert Palace ~ Main Entrance)
0A	0085	1558B	2nd LightWorld Dungeon (Desert Palace ~ Right Entrance)
0B	0083	1558C	2nd LightWorld Dungeon (Desert Palace ~ Left Entrance)
0C	0063	1558D	2nd LightWorld Dungeon (Desert Palace ~ Boss Entrance)
0D	00F2	1558E	Elder's House (Kakariko Village ~ Left Entrance)
0E	00F3	1558F	Elder's House (Kakariko Village ~ Right Entrance)
0F	00F4	15590	Angry Brothers House (Kakariko Village ~ Left Entrance)
10	00F5	15591	Angry Brothers House (Kakariko Village ~ Right Entrance)
11	00E3	15592	Cavern
12	00E2	15593	Cavern (Right of the Lost Woods)
13	00F8	15594	Cavern (Dark World Death Mountain)
14	00E8	15595	Cavern (Dark World Death Mountain)
15	0023	15596	Cavern (Dark World Death Mountain)
16	00FB	15597	Cavern (Dark World)
17	00EB	15598	Cavern (Dark World)
18	00D5	15599	Cavern (Dark World Death Mountain)
19	0024	1559A	Dungeon Cavern (Dark World Death Mountain)
1A	00FD	1559B	Cavern (Light World Death Mountain)
1B	00ED	1559C	Cavern (Light World Death Mountain)
1C	00FE	1559D	Cavern (Light World Death Mountain)
1D	00EE	1559E	Cavern (Light World Death Mountain)
1E	00FF	1559F	Cavern (Light World Death Mountain)
1F	00EF	155A0	Cavern (Light World Death Mountain)
20	00DF	155A1	Cavern (Light World Death Mountain)
21	00F9	155A2	Cavern (Light World Death Mountain)
22	00FA	155A3	Cavern (Light World Death Mountain)
23	00EA	155A4	Cavern (Light World Death Mountain)
24	00E0	155A5	Hyrule Palace (Aghanim's Tower)

25	0028	155A6	Water Palace (Dark World Dungeon)
26	004A	155A7	Monkey Palace (Dark World Dungeon)
27	0098	155A8	Swamps Palace (Dark World Dungeon)
28	0056	155A9	Lost Woods (Dark World Dungeon ~ Entrance 1)
29	0057	155AA	Lost Woods (Dark World Dungeon ~ Entrance 2)
2A	0058	155AB	Lost Woods (Dark World Dungeon ~ Entrance 3)
2B	0059	155AC	Lost Woods (Dark World Dungeon ~ Boss Entrance)
2C	00E1	155AD	Thief's Hideout (Light World Lost Woods)
2D	000E	155AE	Ice Palace (Dark World Dungeon)
2E	00E6	155AF	Cavern (Light World Death Mountain)
2F	00E7	155B0	Cavern (Light World Death Mountain)
30	00E4	155B1	Cavern (Light World Death Mountain)
31	00E5	155B2	Cavern (Light World Death Mountain)
32	0055	155B3	Hyrule Palace Secret Underway Passage
33	0077	155B4	Aghanim's Tower (3rd Light World Dungeon)
34	00DB	155B5	Thief's Village (Dark World Dungeon)
35	00D6	155B6	Turtle Palace (Dark World Dungeon)
36	0010	155B7	Pyramid Entrance (Dark World)
37	000C	155B8	Ganon's Tower (Dark World Dungeon)
38	0008	155B9	Fairy Cavern
39	002F	155BA	Cavern (Kakariko Village)
3A	003C	155BB	Cavern Stairs (Dark World Death Mountain)
3B	002C	155BC	Cavern Stairs (Dark World Death Mountain)
3C	N/A	155BD	Shopman (Light World Lost Woods)
3D	N/A	155BE	Shopman (Death Mountain)
3E	N/A	155BF	Kakariko Village House
3F	N/A	155C0	Kakariko Village House
40	N/A	155C1	Kakariko Village House (Sick Boy)
41	N/A	155C2	Blue Cane Cavern (Dark World Death Mountain)
42	N/A	155C3	Kakariko Village House (Tavern Main Entrance)
43	N/A	155C4	Kakariko Village House (Tavern Backdoor)
44	N/A	155C5	Kakariko Village House (Hedge Man)
45	N/A	155C6	Sahasrahla House (East Palace Area)
46	N/A	155C7	Kakariko Village House (Farm Boy)
47	N/A	155C8	Shopman (Dark World Chest Game)
48	N/A	155C9	Thief's Village House (Dark World Secret House with 300 Rupees)
49	N/A	155CA	Library (South of Kakariko Village)
4A	N/A	155CB	Kakariko Village House (Secret Storage)
4B	N/A	155CC	Kakariko Village House (Chickens House)
4C	N/A	155CD	Shopman (Potions Hut)
4D	N/A	155CE	Desert Elder Cavern
4E	N/A	155CF	Water Palace Switch (Light World)
4F	N/A	155D0	Chest Cavern (Light world Death Mountain)
50	N/A	155D1	Fairies Cavern
51	N/A	155D2	Cavern (Piece of Heart 1)
52	N/A	155D3	Cavern (Piece of Heart 2)
53	N/A	155D4	Bomb Shop where you get the Red Bomb (Dark World)
54	N/A	155D5	Thief's Village House

55	N/A	155D6	Shopman (Death Mountain)
56	N/A	155D7	Icerod Cavern 1
57	N/A	155D8	Shopman (Three Items)
58	N/A	155D9	Shopman (Dark world Death Mountain)
59	N/A	155DA	Shopman (Dark World Archer Game)
5A	N/A	155DB	Cavern where The Sanctuary was before (Dark world)
5B	N/A	155DC	Cape Cavern (Under A Grave)
5C	N/A	155DD	Waterfall Pond (Before entering Zoras Domain)
5D	N/A	155DE	Fairy Pond (Lake Hylia Isle)
5E	N/A	155DF	Fairy Place (Full Health Recovery)
5F	N/A	155E0	Dungeon/Cavern (Piece of Heart)
60	N/A	155E1	Shopman (Three Items)
61	N/A	155E2	Kakariko Village House (Thief's Hideout)
62	N/A	155E3	Cavern
63	N/A	155E4	Fat Fairy Pond (Dark World Pyramid)
64	N/A	155E5	Blacksmith (Right of Kakariko Village)
65	N/A	155E6	Fortune Teller 1
66	N/A	155E7	Fortune Teller 2
67	N/A	155E8	Shopman (Three Items)
68	N/A	155E9	Monkey Palace Area Small Temple
69	N/A	155EA	Cavern
6A	N/A	155EB	Cavern
6B	N/A	155EC	Lumberjacks House
6C	N/A	155ED	Cavern
6D	N/A	155EE	Rupiees Cavern (Before Entering The Desert)
6E	N/A	155EF	Cavern (Piece of Heart)
6F	N/A	155F0	Cavern
70	N/A	155F1	Shopman (Cavern)
71	N/A	155F2	Fairies Cavern
72	N/A	155F3	Cavern (Piece of Heart)
73	N/A	155F4	[*Event*] Hyrule Palace - Zelda's Cell Area
74	N/A	155F5	[*Event*] Hyrule Palace - Throne Room
75	N/A	155F6	[*Event*] Hyrule Palace - Aghanim's Bed (Where he sends the descendants)
76	N/A	155F7	Lost Woods Palace (Dark World ~ Main Entrance)
77	N/A	155F8	Lost Woods Palace (Dark World ~ Hole 1)
78	N/A	155F9	Lost Woods Palace (Dark World ~ Hole 2)
79	N/A	155FA	Lost Woods Palace (Dark World ~ Other Entrance)
7A	N/A	155FB	Cavern
7B	N/A	155FC	[*Event*] Ganon's Final Battle in The Pyramid (Hole)
7C	N/A	155FD	Cavern (Hole)
7D	N/A	155FE	Hyrule Palace Secret Underway Passage (Hole)
7E	N/A	155FF	Cavern (Hole)
7F	N/A	15600	Fairies Cavern (Hole)
80	N/A	15601	Cavern (Hole)
81	N/A	15602	Hyrule Palace Underground Sewers
82	N/A	15603	Chris Houlihan's Room
83	N/A	15604	Cavern Stairs (Dark World Death Mountain)
84	N/A	15605	Ice Rod Cavern 2

Starting Location 00	N/A	15E08	Link's House
Starting Location 01	N/A	15E09	Sanctuary
Starting Location 02	N/A	15E0A	Hyrule Palace - Zelda's Cell Area
Starting Location 03	N/A	15E0B	Hyrule Palace - Secret Underway
Starting Location 04	N/A	15E0C	Hyrule Palace - Throne Room
Starting Location 05	N/A	15E0D	Old Man Cavern (Light World Death Mountain)
Starting Location 06	N/A	15E0E	Old Man Cavern (Light World Death Mountain)

Overlay 00	N/A	N/A	Holes 01
Overlay 01	N/A	N/A	Holes 02
Overlay 02	N/A	N/A	Holes 03
Overlay 03	N/A	N/A	Holes 04
Overlay 04	N/A	N/A	Holes 05
Overlay 05	N/A	N/A	Holes 06
Overlay 06	N/A	N/A	Holes 07
Overlay 07	N/A	N/A	Holes 08
Overlay 08	N/A	N/A	Holes 09
Overlay 09	N/A	N/A	Holes 10
Overlay 0A	N/A	N/A	Holes 11
Overlay 0B	N/A	N/A	Holes 12
Overlay 0C	N/A	N/A	Holes 13
Overlay 0D	N/A	N/A	Holes 14
Overlay 0E	N/A	N/A	Holes 15
Overlay 0F	N/A	N/A	Holes 16
Overlay 10	N/A	N/A	Holes 17
Overlay 11	N/A	N/A	Holes 18
Overlay 12	N/A	N/A	Holes 19

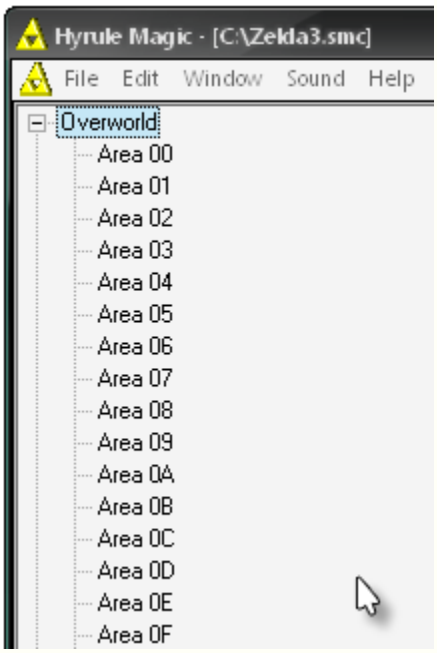
Layout 00	N/A	N/A	4 Squares
Layout 01	N/A	N/A	2 Vertical Rectangles
Layout 02	N/A	N/A	2 Squares + 1 Vertical Rectangle
Layout 03	N/A	N/A	2 Squares + 1 Vertical Rectangle (Inversed)
Layout 04	N/A	N/A	2 Rectangles
Layout 05	N/A	N/A	2 Squares + 1 Horizontal Rectangle
Layout 06	N/A	N/A	2 Squares + 1 Horizontal Rectangle (Inversed)
Layout 07	N/A	N/A	1 Big Square

Watergate Overlay	N/A	N/A	Water Palace Switch (Water Overlay)

7) Overworlds

by Sephiroth3, Orochimaru, DxEdge and Euclid

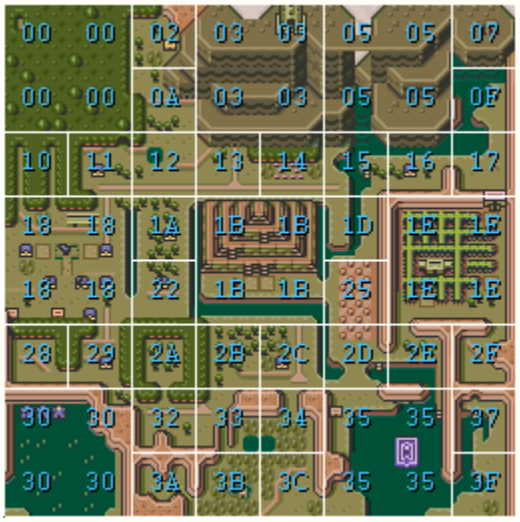
a) The basics



To edit an area, select one from the list:

00-3F: Light world screens.
40-7F: Dark world screens.
80: Area where the Master Sword lies and Area under the bridge.
81: Zora's waterfall (Where you buy the flippers for 500 rupees).
88: Screen with the Triforce (Ending Sequence).
95: Mountain background.
96: Pyramid background.
9C: Lava background.
9D: Fog background.
9E: Forest background.
9F: Rain background.

Above: Light World Screens as shown in HM.



Above: Dark World Screens as shown in HM
(Note that that actual areas numbers differs from these).

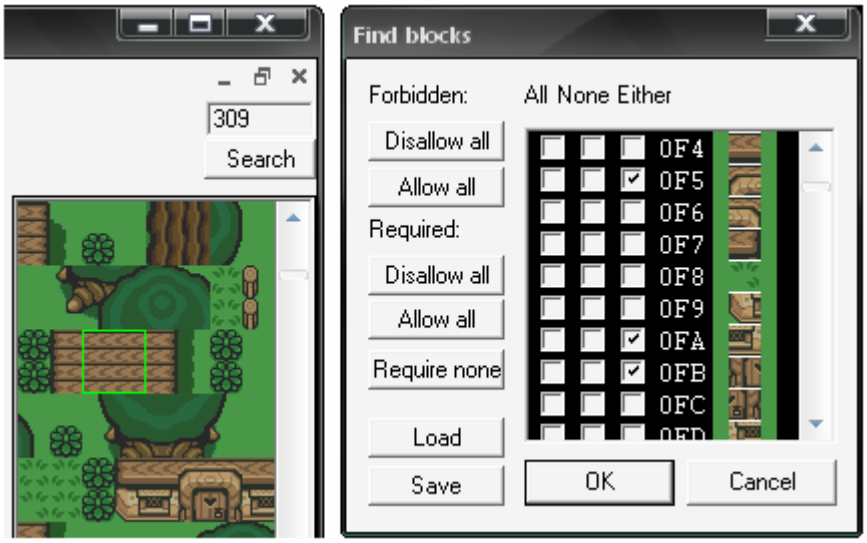
The Map Window

This window shows the selected Area, along with locations of interest.

Entrances: The yellow circles
Exits: The white circles
Transport: The blue circles
Sprites: The pink circles
Holes: The black circles
Items: The red circles

To modify these, you must type a new value with the keyboard. You can add and remove locations by right clicking in the map window with the appropriate tool selected. However, you cannot add more than there are already in the game.

****When you are drawing the map, you can only undo the last operation. So remember to use the "Undo" button anytime you make a mistake.****



!!!!!HM963 screen!!!!

The Block Selector

This window displays all the overworld blocks in the game. You can select a block by clicking on it or typing its number in the above field. You can also edit blocks by double-clicking on them. You can search for specific 16x16 tiles by pressing Search.

Properties

Allows you to change multiple options. First off, you can change the Sign Text you want to

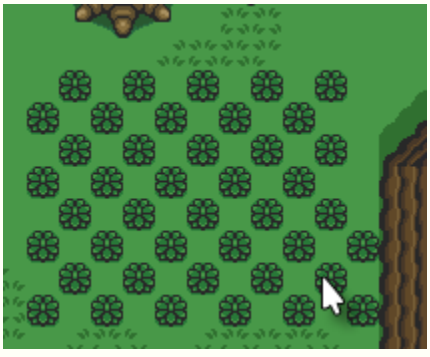
see in that Area. You can see the various texts in the Monologue Editing part of this FAQ. This is also here that you choose which music and ambient sounds will play in that place. You can choose any music that you want depending on the events in the game (eg. You want to use Dark World theme as the Beginning default, but after you rescue Zelda you choose the Town music). Finally, the ambiant sounds, can only be changed if you want to remove the ambiance.. (eg. Changing heavy rain to Nothing; when you will play you won't hear any rain but Thunder SFX Wave).

Draw

When the draw tool is selected, you can place blocks with the left click and select the block underneath with the right click.

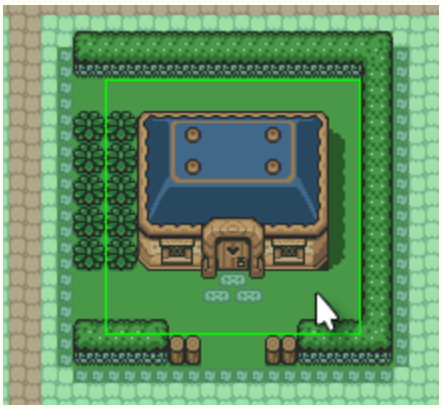
Select

When the select tool is selected, you can select an area with the left click and move it or copy it by pressing the Copy button.



Rectangle

This allows you to draw a filled rectangle using the selected block.



Copy/Paste

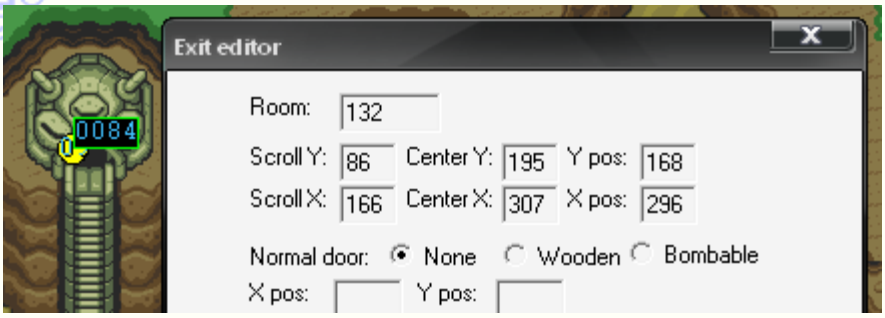
Copy and paste the tiles you select.
(This works best to copy tiles from one screen to another)

Frame

This shows the three animated frames of your tiles.

Address Calculator

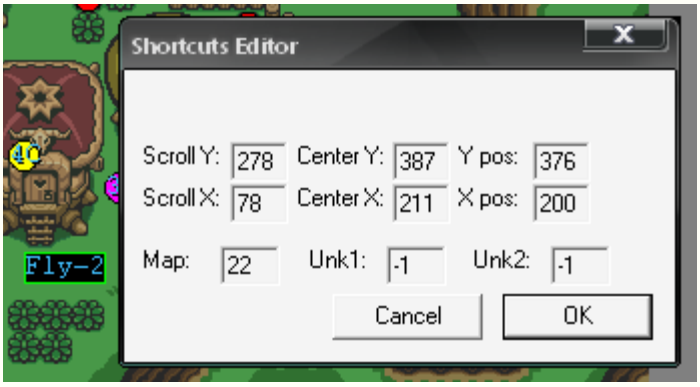
This will tell you the RAM locations of the tile position you click.



Exit

This allows you to move exits.

Note: To edit the Exit circles you have to double click on them.



Transport

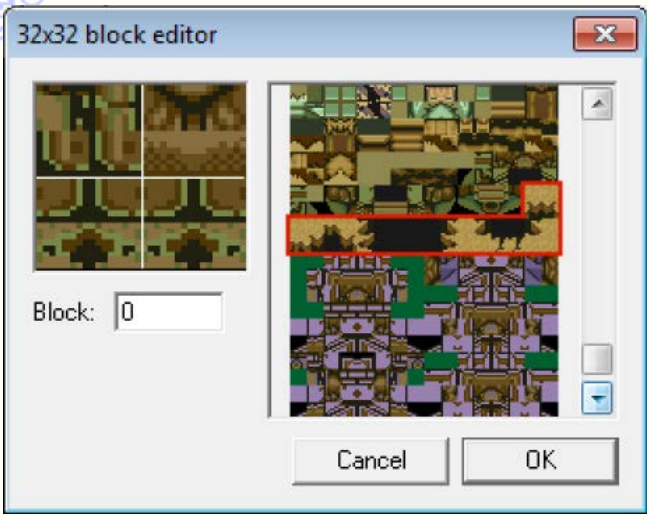
This is similar to the exit tool, but operates on whirlpools and bird locations.

Note: To edit the Transport circles you have to double click on them.

Bird locations

To add a bird/flute location, simply add one wherever you please on any screen of your choosing. You can add a total of eight bird locations. It is to be noted that Fly – 9 is related to the wreckage of the Pyramid where Ganon waits for the final battle (the wreckage will always happen in front of Fly-9).

You can edit the wreckage graphics in the 32x32 editor:



To edit the actual locations that you see on your overworld worldmap, you need to go in HM's world map editor:

Whirlpools

A Whirlpool in A Link to the Past is a magical form of transportation (or form of warping) that transports Link from one body of water to another.

There are **six of them in the Light World** and **two of them in the Dark World**.

Whirlpools work in pairs, meaning that when Link takes one to another, he can go back to the location he warped from by swimming back into the destination whirlpool.

Each whirlpool will always take Link to the same preset destination.

It was long thought that you couldn't add new whirpools using Hyrule Magic because they wouldn't work properly. I've actually managed to make them work! Editing them is actually quite fairly simple.

First let's analyze one of the whirlpools pairs (there are four pairs of two whirlpools that are linked to each other).

The whirlpool south of the Pond of Happiness *in* Lake Hylia – Area 35 (Map 53):

Picture: "The whirlpool south of the Pond of Happiness in Lake Hylia."

When you create a new whirlpool, double-click on it to get to this "Shorcuts Editor" window.

The whirlpool map is the Area of destination. This is where you enter the number of the area you want to teleport to (although you have to write it in decimal). So for exemple if you want to teleport to Area 14 (in hex) you have to write 20 there. Because 20 in decimal is 14 in hex.

The scrolls, center and pos don't need to be changed at all since they are the actual coordinates of your whirlpool on the screen. You can try to move your whirlpool icon around your screen to see these values change imediately according to it's current scrolls, center and pos properties.

The Map value doesn't need to be changed either as it's the current location of your whirlpool. If you're in Area 35 (in hex) then your map value will be 53.
The unknown properties are obviously unknown to me as I have no idea what they actually do.
I don't think you actually need to play with those values in order to create sucessfull whirlpool teleports…

Now, let's analyze the other whirlpool linked to this one.

The whirlpool in Zora's Lake - Area 0F (Map 15):

Picture: "The whirlpool in Zora's Lake."

As you can see, this whirlpool map is 53. Which belongs to Area 35, the area we previously analyzed. Once you have two whirlpools that are linked to each other, you will be able to teleport between the areas.

To create a gateway between two areas, you simply need to create two whirlpools in each of them and then edit their whirlpool map so that they link to each other! Then add a –Whirlpool sprite and place it close to your blue whirlpool icon. Like so:

Voilà, there you have it, a working overworld teleport in one simple edit! Don't forget to write your whirlpool map value in decimal!

Also, it should be noted that the whirlpool don't need to be in water to function! You can place them on grass if you want and they will still work!

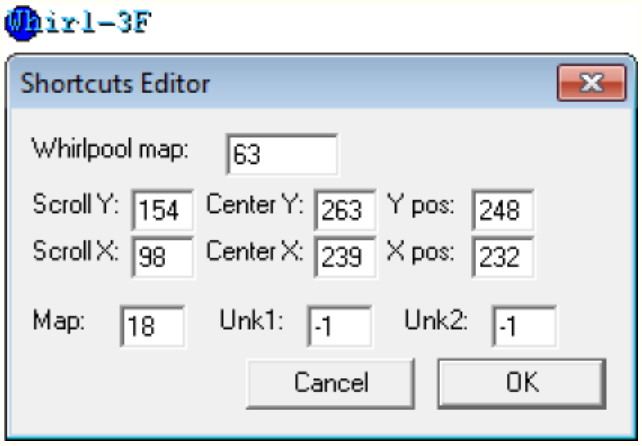
The only thing that then needs to be done in that case is to edit your whirlpool graphics to something else and you'll then have true working teleports on overworlds, just like we can witness in the dungeons!
Let's now take a brief look at the remaining three pairs of whirlpools as seen in the original game.

The whirlpool in the southeastern part of the Lake Hylia - Area 3F (Map 63):

Picture: "The whirlpool in the southeastern part of the Lake Hylia."

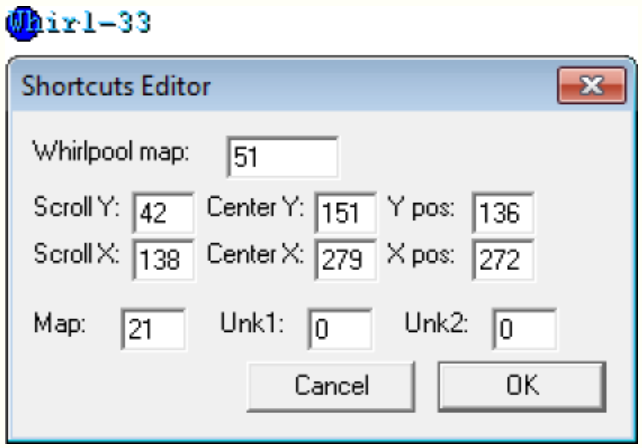
LINKS TO:

The whirlpool in the pond west of the Sanctuary - Area 12 (Map 18):



Picture: "The whirlpool in the pond west of the Sanctuary."

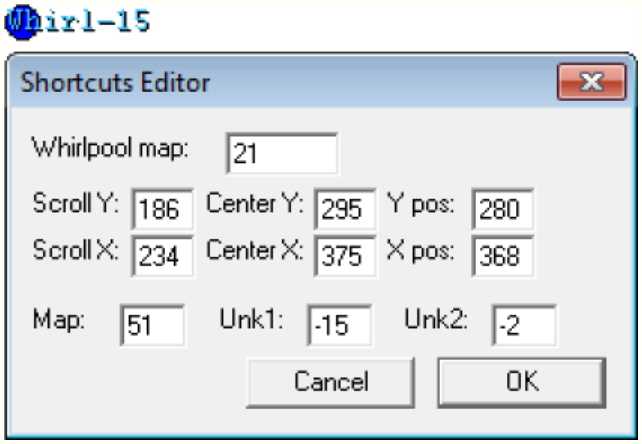
The whirlpool close to the Witch's Hut in Zora's River - Area 15 (Map 21):



Picture: "The whirlpool close to the Witch's Hut in Zora's River."

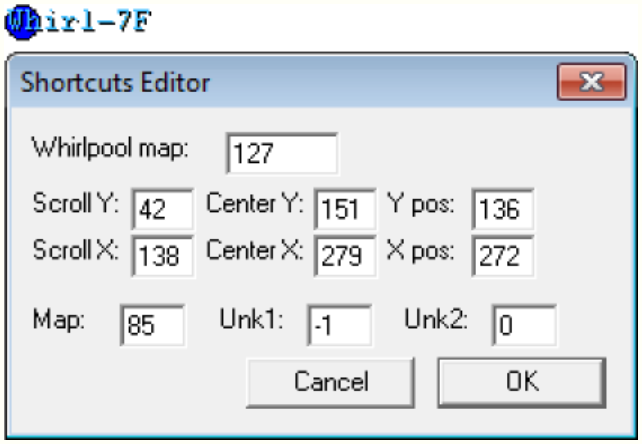
LINKS TO:

The whirlpool in the Great Swamp - Area 33 (Map 51):



Picture: "The whirlpool in the Great Swamp."

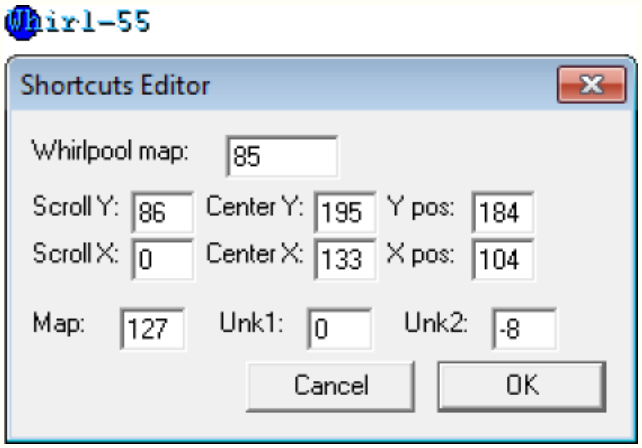
The whirlpool in the Dark World Zora's River - Area 55 (Map 85):



Picture: "The whirlpool in Zora's River."

LINKS TO:

The whirlpool in the Dark World equivalent of Lake Hylia - Area 7F (Map 127):



Picture: "The whirlpool in the Dark World equivalent of Lake Hylia."

That's pretty much it about whirlpools! You can use the clear whirlpools option in HM is want to truly create new teleports. It's safe enough as I was able to create four whirlpool pairs that successfully worked in a mather of minutes!

Also I've just discovered that if you place a whirlpool sprite (you don't need to add a blue whirlpool icon for it to work) in **Area 1B** (The castle area in the original game). Once your character reachs the exact spot where the whirlpool is, it will teleport him to the dark world pyramid area (**Area 5B**).

The whirlpool sprite in that area acts just about the same as that opened castle door trigger that teleports you to the pyramid area. Only now you can place the teleport anywhere in the screen you please using the whirlpool sprite!
Also I've tried using the whirlpools to teleport from a totally random area from one world to another totally random area of the other world and it partially works! I might need to actually play with those unknown properties who knows...

As soon as you arrive in the next world, the dark world music begins to play and the world map changes! The only thing that isn't right is the x/y pos of Link as he arrives into that new world.
You can try to move but what you see on screen isn't the actual place where your sprite is.

Item

This allows you to add or move existing items.

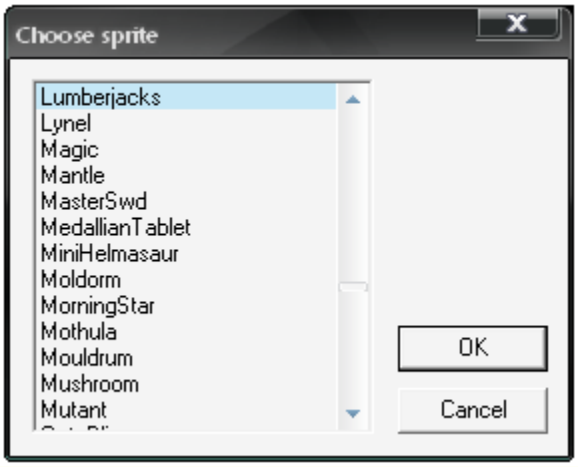
Note: To insert an item, you must right click where you want to add it and select **Insert item**.

Hole

This allows you to add or move holes.

Note: To insert an hole, you must right-click and select **Insert hole**.

Also, holes need a red item with the description "hole" on top of them in order to work. Otherwise, they will send you to Chris Houlihan room.



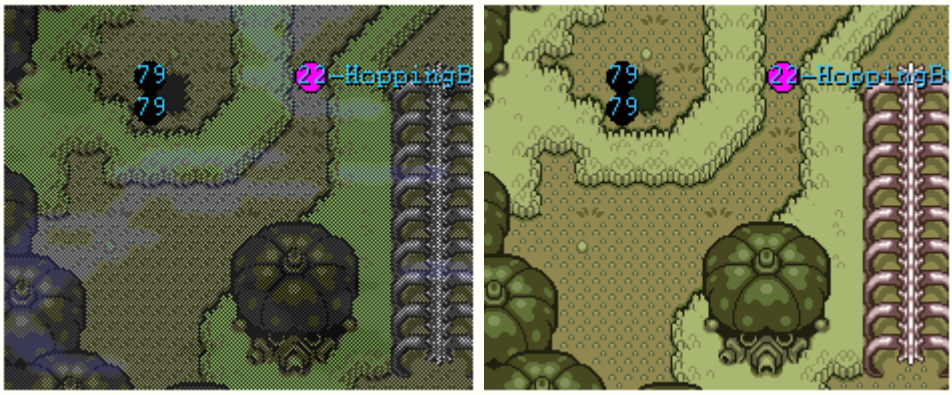
Sprite

This allows you to add or move existing sprites.

Note: To insert an item, you must right-click and select **Insert sprite**.

Warp Switch

Allows you to warp back and forth between Light & Dark Worlds areas.



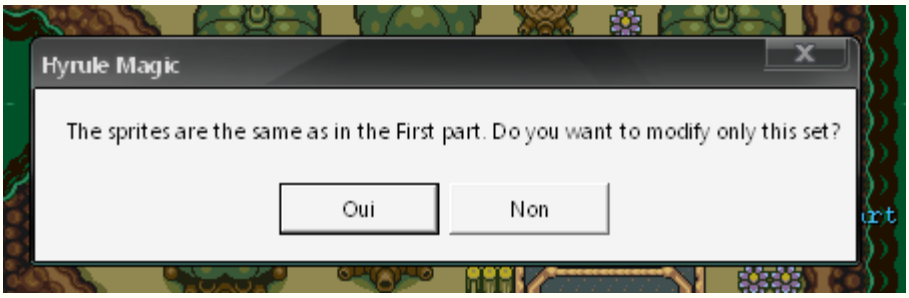
Background

Gives you the choice the view the map with or without the background.
Note: Not applicable to all areas.

Beginning/First part/Second part

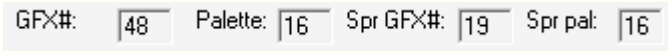
Each sprite set depends on events. For **Light world**, Beginning set shows the sprites as they are in the game before rescueing Zelda. First Part are before getting the Master Sword, and Second Part After getting the Master Sword (eg. You want put fake swords in your forest before getting the Master Sword; After getting it you can simply delete them by selecting the Second part set).

You will get the option:



Choose yes to only modify this set. Otherwise, if you select No, this will result in modifying the First set again... we don't want that to happen, so just select Yes. For **Dark world**, there is only the First set available, thus choosing the second set, will not be available while playing the game.

Beginning: Shows the Beginning sprites set **(Before rescueing Zelda)**.
First part: Shows the First Part sprites set **(Before getting the Master sword)**.
Second part: Shows the Second Part sprites set **(After getting the Master sword)**.



GFX#: Choose the GFX# from 0 to 79. (Many duplicates)
Palette: Choose the Palette from 0 to 999. (Many duplicates)
Spr GFX#: Choose the Sprites GFX according to the monsters/objects surrounding the area.
Spr Pal: Choose the Sprites Pallete according to the monsters/objects surrounding the area.

b) Overworld GFX# Sets

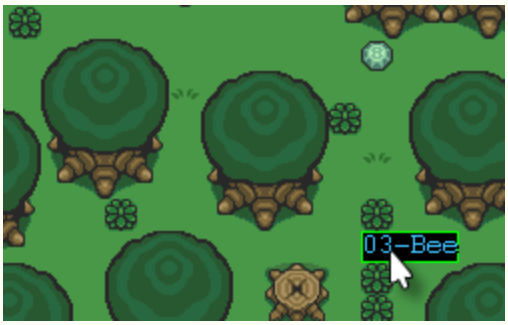
by DxEdge

Within dungeon editor you can right click and select "Edit Blocks" and then you can select from the drop down menu "Other". This will give you a box that you can put in any number between 000 and 219. Those are the sets related to overworlds.

31 > Numbers, Hearts, Menu Items, Water Animations
32 > Kakariko Village
33 > Lost Woods
34 > Death Mountain
36 > Hyrule Castle
37 > Eastern Palace
38 > Desert 1
39 > Water Palace
41 > Catedral
43 > Desert 2
47 > Master Sword Area
59 > Pyramid
60 > Turtle Palace
61 > Monkey Palace
62 > Evil Forest
63 > Thiefs Village
64 > Water Palace, Graveyard (Dark World)
65 > Death Mountain (Dark World)
66 > Swamp Palace

c) Special notes on overworld editing

by Euclid



Items must be placed under objects like rocks, bushes or pots. Items you most likely put under objects are hearts, magic bottles, switches or secret staircases.

Sprites can contain the same thing as items but they don't need to be hidden underneath an object. (e.g. Heart Pieces can just be laced on the ground in clear open view).

Areas 03 & **05** are designed to make the grass brown. If you joined 4 small areas together to make one big one then the gfx sets and background settings all come from the top left area. That's why if you join **02**, **03**, **0A** & **0B** together the grass goes green. It takes the info from area **02** which originally has its grass green. It also means that **03** and **0B** have the rain background and not the mountain background they would have clumped together with areas **04** & **0C**.

Note:

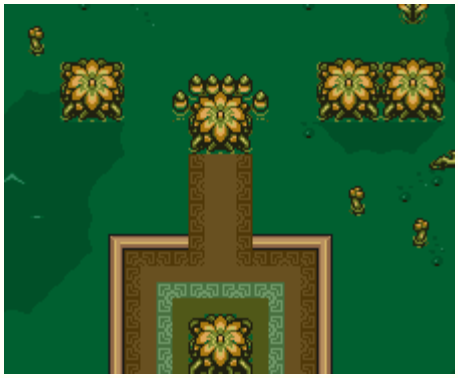
Moving from a brown grass area (e.g. **Area 03**) to a non-brown grass area (e.g., **Area 02**) is **NOT** a good idea, GFX will stuff up until you enter some entrance and exit. However exiting from caves is okay. Unless you have your square on **Area 02** to cancel out the brown grass of **Area 03**, be careful, very careful.

You need **Item: 80-Hole** (The red circles) and **Hole: 7D** (The black circles) under either a bush or a rock next to the castle secret entrance for it to work (Beginning). Doesn't matter which order.

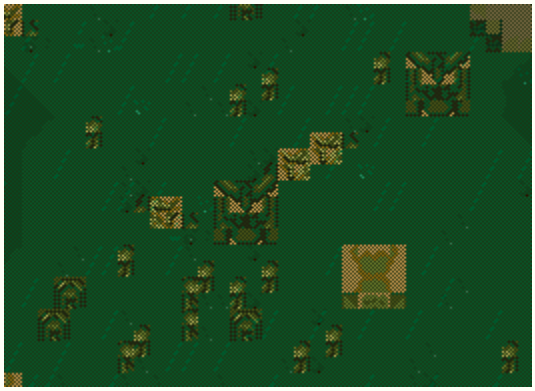
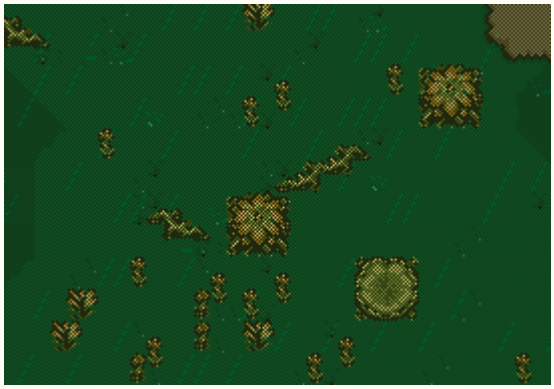
The tree stump/hollow tree entrance to the Master Sword screen clearing needs to be directed to a particular exit.

Shown above are the default Master Sword exit properties. If you get stuck with something like entrances and exits always look back at the original Z3 ROM to compare them to your own work.

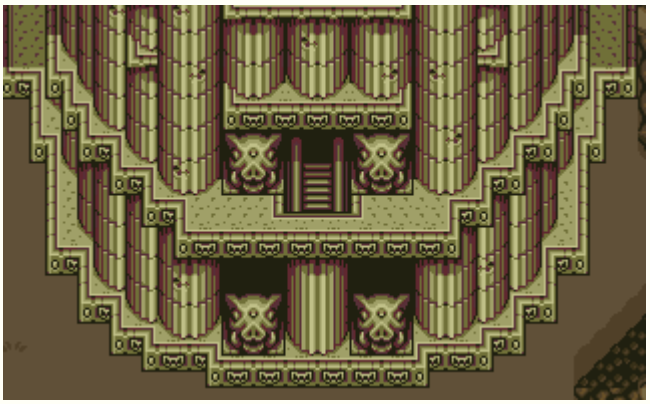
The watergate entrance **4E**, has to stay in **Area 3B** for it to work. In other screens the water inside will not go down after you use the gate

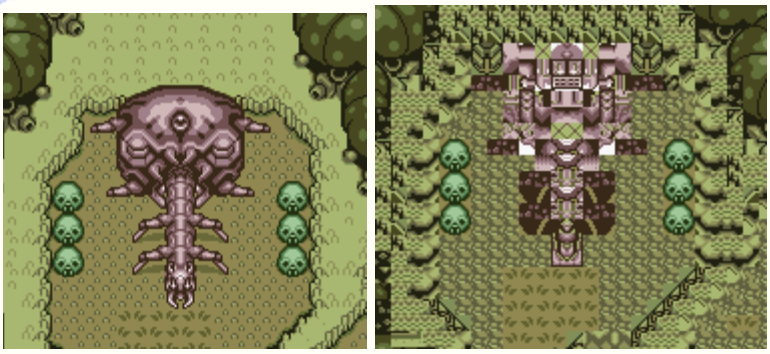
Although you don't have the GFX for the turtle rock or the swamps where you use the **Ether** and **Quake** Medallions, if you do it at the right place the entrance will still appear. One way around is to make sure the user does not get to that tile where you can do the magic to open up the entrance.

Entering **Area 70**, **71**, **78**, **79** from any adjacent overworld pieces will make the background GFX look dizzy. There is a way around that though, head to the Overworld map and come back with the **x button** (Because it's raining until you do the magic, after the magic it will works fine).



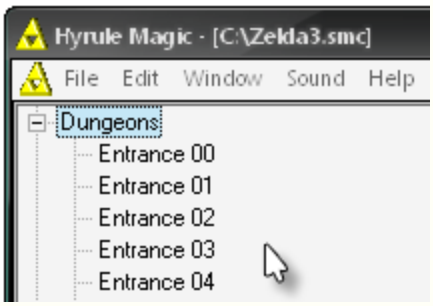
Ganon's fortress won't flash anymore if you move it off screen. In fact if you just move it one tile across, it won't have the "seal" over it anymore.



In **Area 40,** (Dark World Forest). Using the fire rod on the boss entrance (Shown Above) will only work on that tile (at least the 2 front piece of that tile) at that particular place. Moving it will result in not being able to make it work again. After blowing up that place the gfx will be stuffed if you're GFX number is not the same as original.

8) Dungeons

by Sephiroth3, Euclid, Orochimaru, DxEdge and Dude Man



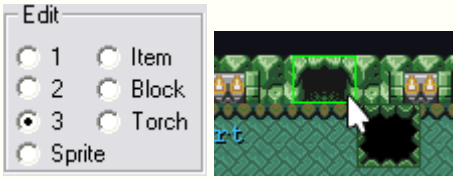
Select an entrance number or starting location in the list to edit it. You can also edit overlays with this editor.

a) The basics

The Room Editor

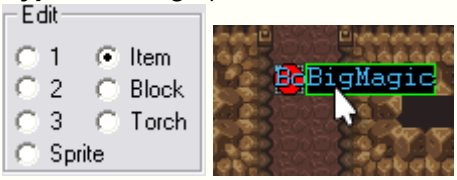
This shows the currently edited room. A room can only be open in one window at a time. You can navigate through the rooms using the arrow buttons and the jump button. Click an object to select it and move it. Some objects can be resized by dragging its border. You can insert and remove objects with the right click button. You can't add more sprites than there already are in the game. The operation of the keyboard depends on the currently selected object.
Note: To add items, sprites, doors, blocks, torches etc... you have to do right click, then add.

Edit 1 and Edit 2 (Backdrop):
Arrows: Move the object.
B: Moves the object towards the back.
V: Moves the object towards the front.
Ctrl+B: Moves the object to the very back.
Ctrl+V: Moves the object to the very front.
N, M, J, and K: Change the object.
Comma, period: Change the size.
Hyphen: Toggles between objects 00-FF and 100-13F.
Plus, backslash: Change the contents of a chest.



Edit 3 (Doors):
Arrows: Move the door.
B: Moves the door towards the back.
V: Moves the door towards the front.
Ctrl+B: Moves the door to the very back.
Ctrl+V: Moves the door to the very front.
N, M, J, and K: Change the door type.

Edit Sprite (Sprites):
Arrows: Move the sprite.
B: Moves the sprite towards the back.
V: Moves the sprite towards the front.
N, M, J, and K: Change the sprite.
Comma, period: Change the parameter.
Hyphen: Change plane.



Edit Item (Items):
Arrows: Move the item.
N, M, J, and K: Change the item.
Hyphen: Change plane.

Edit Blocks (Pushable Blocks):
Self-Explanatory.

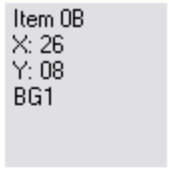
Edit Torch (Torches are for dark areas or triggers):
Self-Explanatory

Object Information

This shows the properties of the currently selected object. This depends on the type of object selected.

Edit 1 and Edit 2 (Backdrop):
Obj: Object number
X: X position
Y: Y position
Size: Object size

Edit 3 (Doors):
Dir: Direction
Type: Door type
Pos: Position



Edit Sprite (Sprites):
Spr: Sprite number.
X: X position.
Y: Y position.
BG: Background 1 or 2.
P: Additional parameter for some enemies.
Note: When it's set to 7, it designates a control sprite.

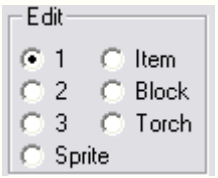
Edit Item (Items):
Item: Item number.
X: X position.
Y: Y position.
BG: Background 1 or 2.

Edit Block (Pushable Blocks):
X: X position.
Y: Y position.
BG: Background 1 or 2.

Edit Torch (Torchs for Dark Areas):
X: X position.
Y: Y position.
BG: Background 1 or 2.
P: The usage of this field is unknown.

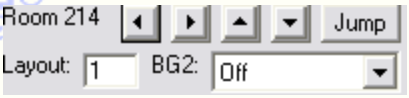
Display Options

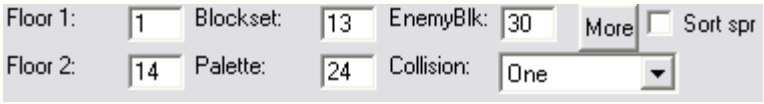
Press the Frm button to show the next animation frame of the animated tiles. You can enable/disable display of BG1, BG2 and Spr (Background 1, Background 2 and Sprites).



Edit Options

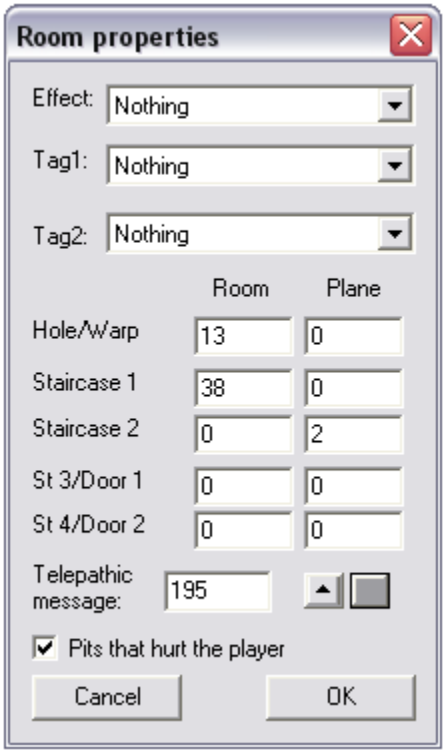
This selects an object type to be edited. 1, 2 and 3 refer to the three backdrop chunks in the room. Chunk 1 is painted on background 1, chunk 2 is painted on background 2 and chunk 3 is painted over background 1.





Room Information and Properties

The current room number is shown in the upper left corner. You can change the floor used on both layers. You can also change the room template. There are **8 room templates** to choose from. In the **BG2** menu you can change how background 2 is drawn. In the collision menu, select One to use only the plane that the player is on for collision. Select Both to include both planes in the collision. If the room uses parallaxing, select **Both w/scroll** to make the player collide with background 2 in the moved position. You can also change the color effect, set up to 2 special properties for the room and change the message on the triangle tile by pressing the More button. If **Sort spr** is checked, sprites that are in the foreground will be displayed on top of sprites that are in the background.



Hole/Warp: Directs the holes or warp tiles. When you fall through a floor or warp to another room it will always be in the exact place in the new room as you left in the old.

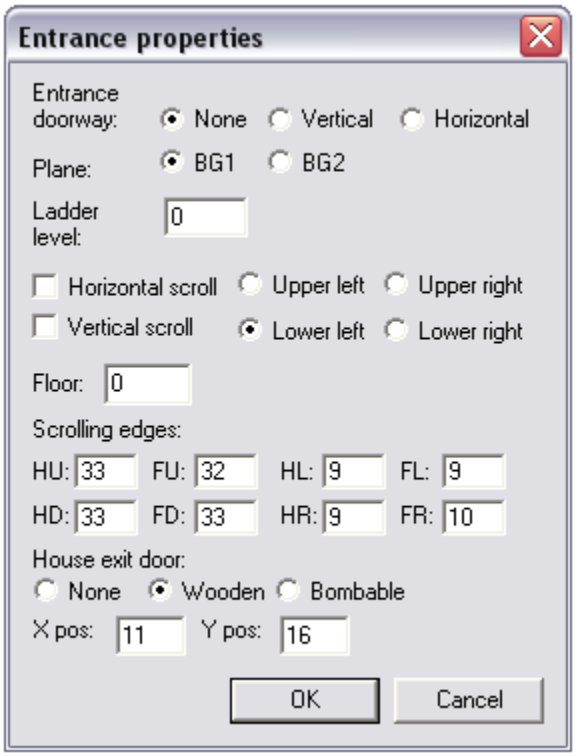
Staircase 1, Staircase 2: Self-Explanatory (**1** = right staircase, **2** = left staircase)

Plane 0 is used for layer 1, plane 1 is for layer 2 and plane 2 is for layer 3..I think this might relate to what layer you placed your doors/stairs.

Door 1, Door 2: Are used for special doors that warp you to a new room (door 0A1 - shown under this text - or door 077). These doors still work normally as your normal dungeon doors but they direct your movement to a new room elsewhere in the ROM. Means you can extend dungeons past their initial area boundaries.

Starting Room and Position

X and Y refer to the player's starting position. X scroll and Y scroll are the coordinates of the upper left corner of the screen as the player enters. CX and CY specify the centre used for scrolling and should be set to X scroll+128 and Y scroll+112 respectively. In the music list you can choose the music in the dungeon, which must be in music bank 2. You can also select the dungeon that is entered. Each dungeon has its unique map and dungeon items.



Entrance Properties

In that window, you first have the Entrance doorway (Usually horizontal, because you always enter from the bottom; none for some cases like holes).

Plane: Self-Explanatory. In other words which plane will Link enter the room in.
Exemple: That Tower dungeon in the Light World you enter in is **BG2**).

Horizontal scroll and **vertical scroll:** Very important settings. These enables some rooms to scroll or else you will be running off the screen. Those are most of the reasons why people's entrance rooms don't scroll properly. The only entrance which refuses to work without some HEX work is entrance **1B**.

Floor: No idea about this setting.

Scrolling Edges : Contains which things to move, and doesn't need to be modified when you're hacking normally.

House Exit door : For rooms 0 - 256 this setting is useless (Because the use of Exits), but for rooms 257+ this determines

where to place the exit door ***(Thus you see some places when you come out of a room, the door exit appears somewhere weird, this is the place to change them)***.It may sounds easy, well there's a catch, HM won't save the values. Whenever you change them and save, when you come back they are as ***Default***.

To fix them, you have to know a bit of HEX and how it relates to the rom.
The offset is **0x15924** for door location and the data goes like this:

At **0x15924**
Exemple: **08 16** (lil endian) **- Entrance 00** (2 bytes)

----yyyy y-xxxxx-

y and **x** represents the positions in bits, and you'll use the same positions as you would if you put an exit there. **-** means <Unknown/Unused>.

So for the Magic Shop you'll have to add that (**Entrance number * 2**) to **0x15924** and you should have that position.

Different dungeons use different blocksets (Not the top one, the bottom one determines which GFX set to load into the rom when you enter through this entrance), if you had all your dungeons in the same blockset, you can join them up without much problems (just make sure the palettes are right).

0x15581 is the offset of the data in the rom (**02: D381**), I guess you'll want to locate the routine which loads this number? Well it's about here in the rom.

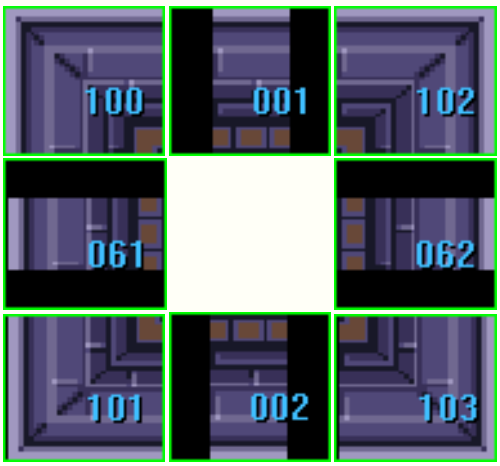
02/DA74: BD 81 D3 LDA $D381,X (this loads **02: D381**, **X** which is probably what you'd expect, unless there's another place).

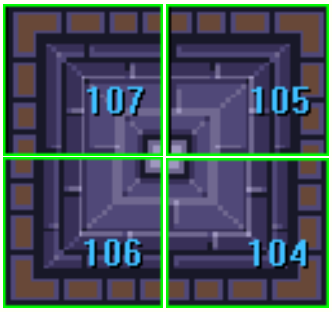
Here's how to change them in HEX: At offset **0x15581** is **Entrance 01**, **0x15582** is **Entrance 2** and so on. You'll get it eventually as it's really easy.

Exemple: If you want to say find the offset for say Entrance **7B**, all you need to do is take the windows calculator, change it to HEX mode and do **7B** + **15581** and you'll have your offset.

b) Objects Database

by Orochimaru



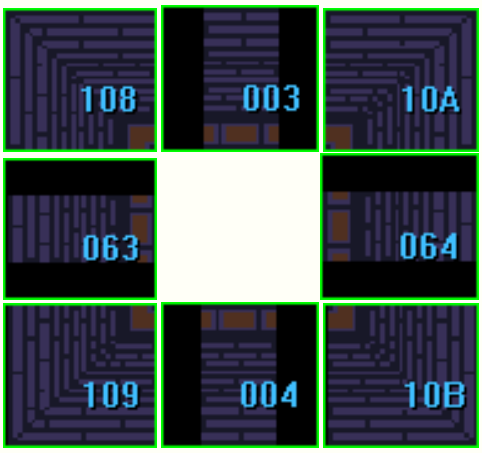


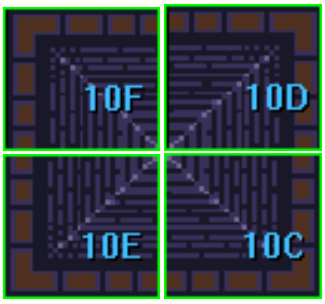
100 =
101 =
102 =
103 =

001 =
002 =

104 =
105 =
106 =
107 =

061 =
062 =



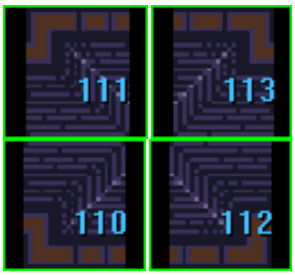


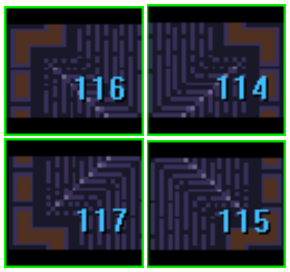
108 =
109 =
10A =
10B =

003 =
004 =

10C =
10D =
10E =
10F =

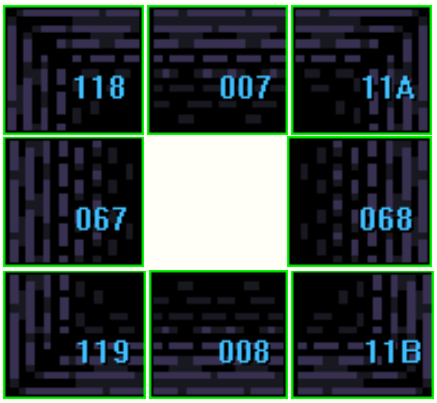
063 =
064 =

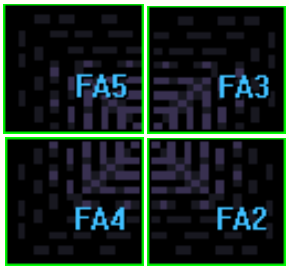




110 =
111 =
112 =
113 =

114 =
115 =
116 =
117 =





118 =
119 =
11A =
11B =

FA2 =
FA3 =
FA4 =
FA5 =

007 =
008 =

067 =
068 =

11C =
11D =
11E =
11F =
120 =

121 = *Wooden barrel*

123 = *Wooden tables and Stone/Rocky structures*

124 = *Royal Throne, Fairy Fountain, Skullhead statue*

127 = *Wooden chairs*

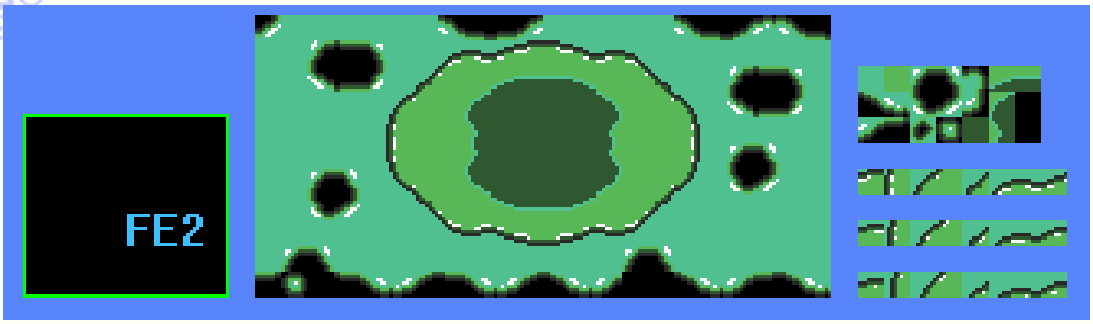
128 = *Bed*

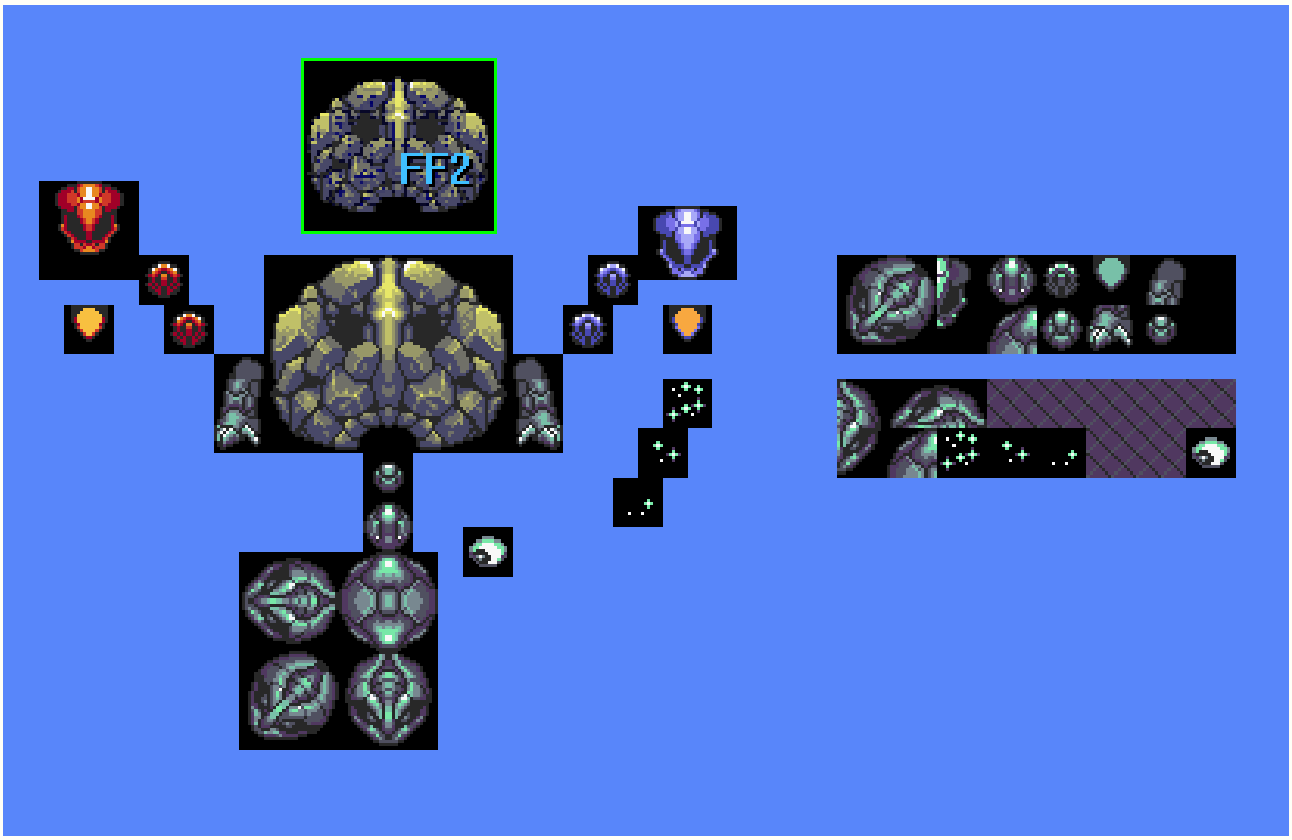
129 = *Old stove*

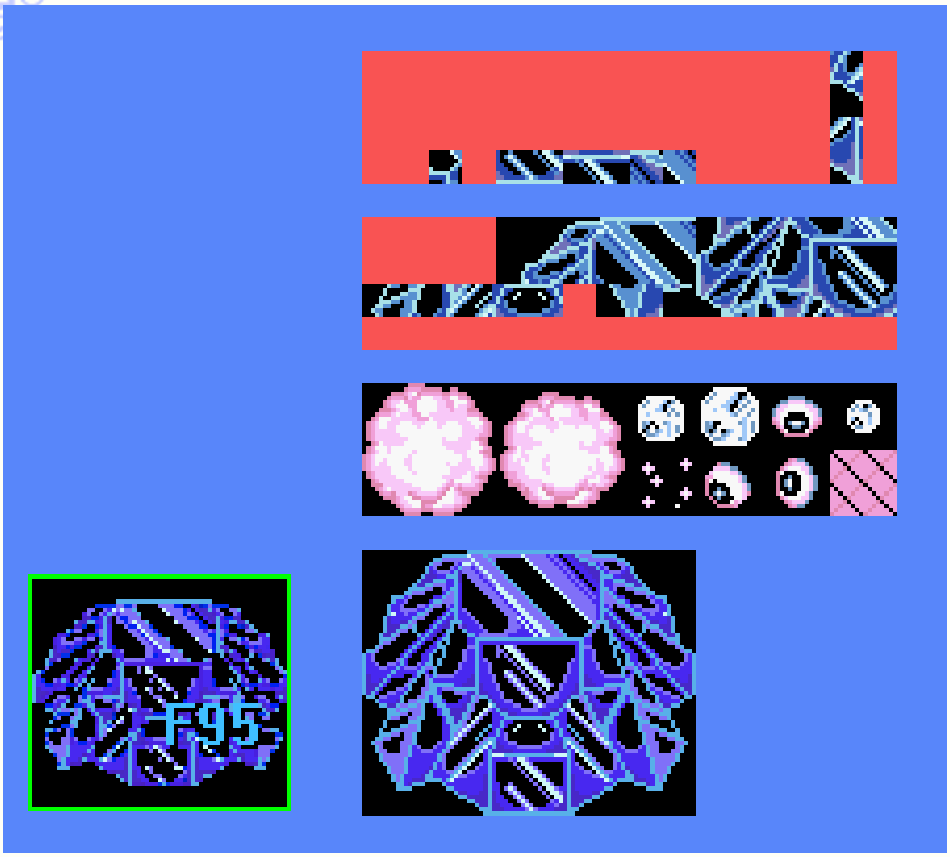
12A = *Mario Picture*

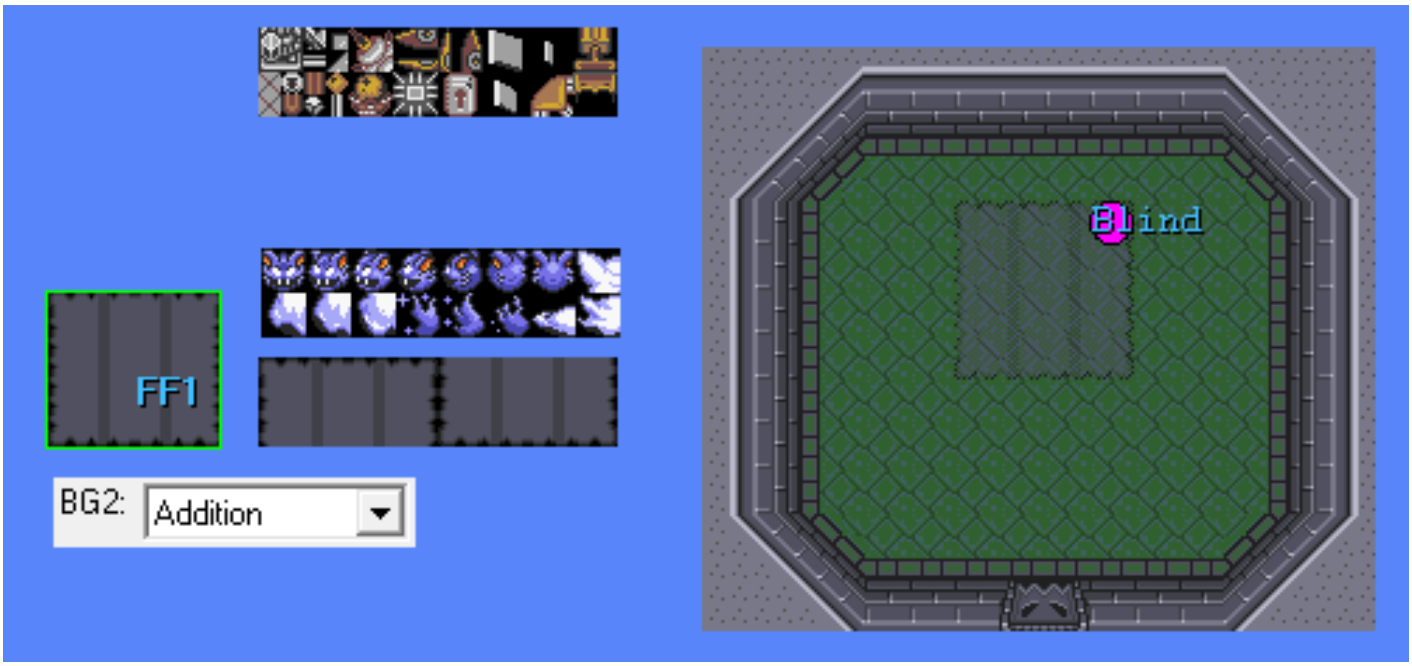
122 = unused?
125 = unused?
126 = unused?
12B = unused?
12C = unused?

FEB =


FE2


FF2


F95


FF1
BG2:
Addition
Blind

c) Doors editing and database

by Spane, Orochimaru and MathOnNapkins

Spane: *The normal doors are exactly 24 bytes long:*

- You have 12 bytes for the tiles
*- and 12 bytes **in front**, **mirror** and **palette**.*

*As an example: At adress **0x4268** and **0x4269 (without header)** you see **88** and **28**.*

*The **88** means the tile that is used. On a default Alttp rom tile **88** is **block 136** in the dungeon piece editor.*

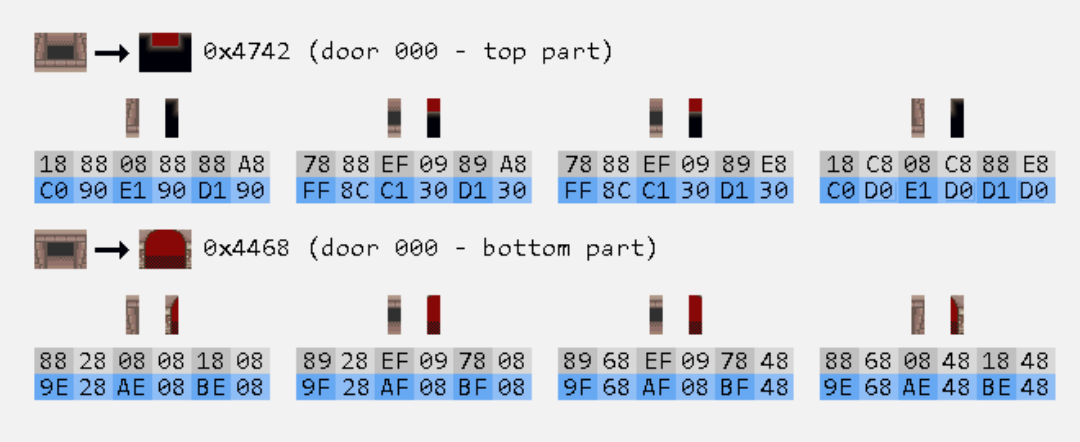
*The **28** stands for **the palette** and **in front**. After these two bytes are the next two bytes for the next tile.*

The next number you see is the 08. That is, the left side of the door. Some doors divide their 24 bytes, but the up doors and down doors can be made up separately.

Legend
08 = normal
28 = in front
48 = mirror on x
68 = in front + mirror on x
88 = mirror on y
A8 = in front + mirror on y
C8 = mirror on x and y
E8 = in front + mirror on x and y

Orochimaru: *At addresses **0x4542 - 0x4559** (door 02A) are the top part for door 000, I'm assuming that like the **0x4268** byte found by Spane that those properties are followed by those for door 001 and so on.. I haven't tested it yet, so it's a wild guess! The grey colors are the original game properties, while the blue ones are my new changes.*

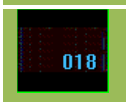
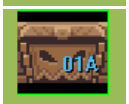
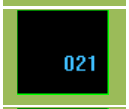
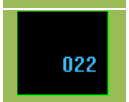
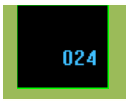
change picture to be properties WITHOUT HEADER.. I got confused for one moment lol

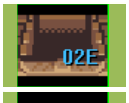
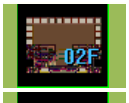
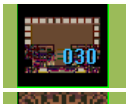
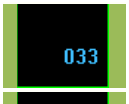
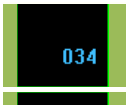
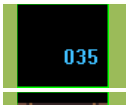


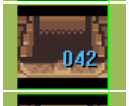
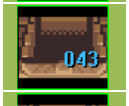
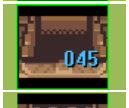
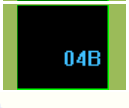
colored in red = double verification!!!!

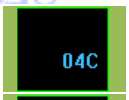
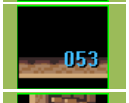
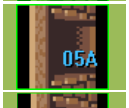
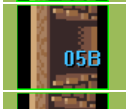
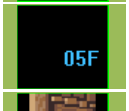
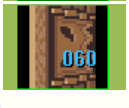
Door #	Hex locations	Description
	0x4268 - 0x427F	*Vertical standard door,* *- door is completely on BG2.*
	0x4280 - 0x4297	*Inset version of door 002* *- Sets priority bits of tilemap entries in a 7 x 4 region leading from the bottom of the door to the edge of the screen.* *- Since this door type insets tile priority, doesn't that mean it can only be used in the positions farther from the edge of the room? (Otherwise causing memory writing problems...?)* *- door is completely on BG1*
	0x4280 - 0x4297	*Cave exit door (inset)* *- why is this different from door 008?*
	0x4298 - 0x42AF	*Unused* *- perhaps this is a debug property / feature. should be investigated* *- it gets a fair amount of attention in the code for each door direction*
	0x4298 - 0x42AF	*Waterfall door (only used in Swamp palace; in one room at that!)* *- probably hides itself via priority bits in tilemap?*
	0x4298 - 0x42AF	*Large palace exit door (BG2)* *- door is completely on BG2* *- only manifests as anything special in down direction*
	0x4298 - 0x42AF	*Large palace exit door (BG1)* *Trap door (probably other types but this seems to be most common)* *- door is completely on BG1* *- only manifests as anything special in down direction*
	0x4298 - 0x42AF	*Cave exit door* *- no comment*
	0x4298 - 0x42AF	*Cave exit door (inset)* *- no comment*
	0x4298 - 0x42AF	*Exit door* *- Adds a property to some doors allowing you to exit to the overworld (this is accomplished by writing to the tile attribute map)*

		How that transition occurs depends on whether the room number is <= 255. If greater than that, Link simply comes out at the location he came in, regardless of what door he exits from.
	**0x4298 - 0x42AF**	*Palace toggle property* *- Use it to modify other doors to give them this property*
	**0x4298 - 0x42AF**	*Floor toggle door* *Transition to dark room?* *- Use it to modify other doors to give them this property*
	**0x42B0 - 0x42C7**	*double sided trap door* *- apparently must be triggered by something to open*
	**0x42C8 - 0x42DF**	*invisible door?* *- only used in Turtle Rock* *- door is completely on BG1*
	**0x42E0 - 0x42F7**	*locked door that opens into normal door* *- versatile, locked on both sides (naturally)*
	**0x42F8 - 0x430F**	*big key door that opens into normal door*
	**0x4310 - 0x4327**	*locked door that obscures staircase type*
	**0x4310 - 0x4327**	*locked door that obscures staircase type* *- Toggles the target BG Link will emerge on.* *e.g. if Link starts on BG0 in the next room he'll be on BG1.* *- door is completely on BG1*
	**0x4328 - 0x433F**	*unused* *- might do something, but not seen in original game*
	**0x4328 - 0x433F**	*locked door that produces staircase?* *- seems like it doesn't draw the top of the door (to help interface with staircases.)* *- only used once in the game.* *- door is completely on BG2*
	**0x4340 - 0x4357**	*bombable vermin door* *- opens when bombed but makes small vermin enemies come out (somehow)*
	**0x4358 - 0x436E**	*bombable exit door* *- notable in that it turns into exit door when bombed open*
	**0x4358 - 0x436E**	*unused* *- Locked door specifically for BG0.*

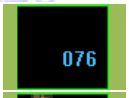
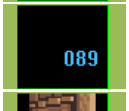
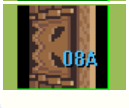
	**0x4370 - 0x4387**	*bombable wall* *- seems to work in a variety of positions*
	**0x473A - 0x4769 (top half)** **0x4478 - 0x44A7 (bottom half)**	*Large exploded pathway resulting from a switch being pulled (unusual to have as a door as it's huge)* *- rarely used* *- door starts off as a wall but a trigger blasts it open*
	**????**	*hidden door that opens with sword slash (e.g. Agahnim's room with the curtain door you have to slash)* *- rarely used* *- door is completely on BG1*
	**0x4388 - 0x439F**	*unused* *- might do something, but not seen in original game*
	**0x4388 - 0x439F**	*BG1 only right side trap door*
	**0x43A0 - 0x43B7**	*BG1 only left side trap door* *- can have two-part form* *- door is completely on BG1*
	**0x43B8 - 0x43CF**	*unused*
	**0x43B8 - 0x43CF**	*unused*
	**0x43B8 - 0x43CF**	*unused* *- might do something, but not seen in original game*
	**0x43B8 - 0x43CF**	*top on BG1 door* *- can have two-part form* *- top of door is on BG1, rest is on BG2*
	**0x43D0 - 0x43E7**	*unused* *- might do something, but not seen in actual game*
	**0x43E8 - 0x43FF**	*trap door triggered by event?* *- can have two-part form* *- seen in positions 9, 10, and 11* *- opened by sprite logic perhaps?*
	**0x4400 - 0x4417**	*arbitrary room link door?* *- designed to specially link to other rooms via dungeon header? (guess)* *- seen in positions 3 and 9* *- top of door is on BG2, rest is on BG1*
	**0x4418 - 0x442F**	*one side trap door* *- two-part door*

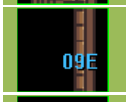
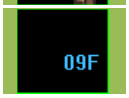
		- only seen in position 11 - top of right-facing door is on BG2, rest is on BG2
	0x4430 - 0x4447	one sided trap door - two-part door - trap door on left, normal (or somewhat normal on right?) - only seen in position 11 - top of right-facing door is on BG2, rest is on BG1 - Trap on left, normal on right
	0x4448 - 0x445F	
	0x4448 - 0x445F	
	0x4448 - 0x445F	
	0x4460 - 0x4477	
	0x4542 - 0x4559	
	0x455A - 0x4571	
	????	
	????	
	0x4572 - 0x4589	
	????	
	????	
	????	
	????	
	????	
	????	
	????	warp door?
	0x458A - 0x45A1	

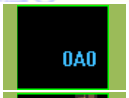
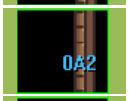
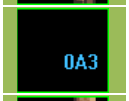
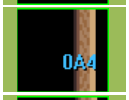
	0x45A2 - 0x45B9	
	0x45BA - 0x45D1	
	0x45D2 - 0x45E9	
	0x45EA - 0x4601	
	0x45EA - 0x4601	
	0x45EA - 0x4601	
	0x45EA - 0x4601	
	0x45EA - 0x4601	
	0x4602 - 0x4619	
	0x461A - 0x4631	
	0x4632 - 0x4649	
	0x464A - 0x4661	
	0x464A - 0x4661	
	0x464A - 0x4661	
	0x464A - 0x4661	
	0x4662 - 0x4679	
	0x467A - 0x4691	
	0x467A - 0x4691	
	0x467A - 0x4691	
	0x467A - 0x4691	
	0x4692 - 0x46A9	

	*0x46AA - 0x46C1*	
	*0x46C2 - 0x46D9*	
	*0x46DA - 0x46F1*	
	*0x46F2 - 0x4709*	
	*0x470A - 0x4721*	
	*0x470A - 0x4721*	
	*0x470A - 0x4721*	
	*0x4722 - 0x4739*	
	*0x47BC - 0x47D3*	
	*0x47D4 - 0x47EB*	
	*0x47D4 - 0x47EB*	
		
	*0x47EC - 0x4803*	
	*0x47EC - 0x4803*	
	*0x47EC - 0x4803*	
	*0x47EC - 0x4803*	
	*0x47EC - 0x4803*	
	*0x4804 - 0x481B*	
	*????*	
	*????*	
	*0x4804 - 0x481B*	

	0x481C - 0x4833	
	0x4834 - 0x484B	
	0x484C - 0x4863	
	0x484C - 0x4863	
	0x484C - 0x4863	
	0x484C - 0x4863	
	0x484C - 0x4863	
	0x484C - 0x4863	
	0x4864 - 0x487B	
	0x4864 - 0x487B	
	0x487C - 0x4893	
	0x4894 - 0x48AB	
	0x4894 - 0x48AB	
	0x4894 - 0x48AB	
	0x4894 - 0x48AB	
	0x48AC - 0x48C3	
	0x48C4 - 0x48DB	
	0x48C4 - 0x48DB	
	0x48C4 - 0x48DB	
	0x48C4 - 0x48DB	
	0x48DC - 0x48F3	

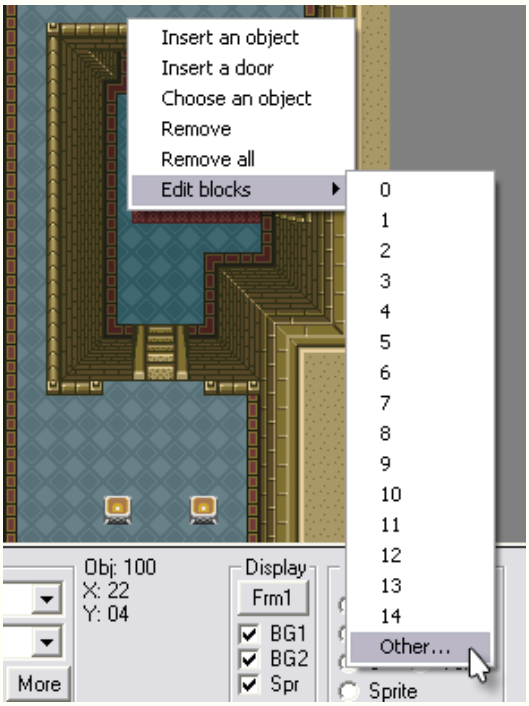
	0x48F4 - 0x490B	
	0x490C - 0x4923	
	0x4924 - 0x493B	
	0x493C - 0x4953	
	0x4954 - 0x496B	
	0x4954 - 0x496B	
	0x4954 - 0x496B	
	0x496C - 0x4983	
	0x49BC - 0x49D3	
	0x49D4 - 0x49EB	
	0x49D4 - 0x49EB	
		
	0x49EC - 0x4A03	
	0x49EC - 0x4A03	
	0x49EC - 0x4A03	
	0x49EC - 0x4A03	
	0x49EC - 0x4A03	
	0x4A04 - 0x4A1B	
		
		
	0x4A04 - 0x4A1B	

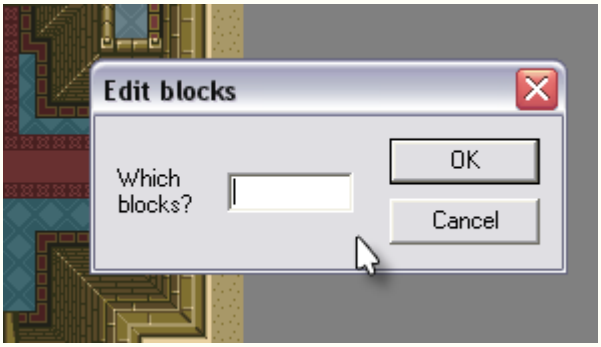
	*0x4A1C - 0x4A33*	
	*0x4A34 - 0x4A4B*	
	*0x4A4C - 0x4A63*	
	*0x4A4C - 0x4A63*	
	*0x4A4C - 0x4A63*	
	*0x4A4C - 0x4A63*	
	*0x4A4C - 0x4A63*	
	*0x4A4C - 0x4A63*	
	*0x4A64 - 0x4A7B*	
	*0x4A64 - 0x4A7B*	
	*0x4A7C - 0x4A93*	
	*0x4A94 - 0x4AAB*	
	*0x4A94 - 0x4AAB*	
	*0x4A94 - 0x4AAB*	
	*0x4A94 - 0x4AAB*	
	*0x4AAC - 0x4AC3*	
	*0x4AC4 - 0x4ADB*	
	*0x4AC4 - 0x4ADB*	
	*0x4AC4 - 0x4ADB*	
	*0x4AC4 - 0x4ADB*	
	*0x4ADC - 0x4AF3*	

	*0x4AF4 - 0x4B0B*	
	*0x4B0C - 0x4B23*	
	*0x4B24 - 0x4B3B*	
	*0x4B3C - 0x4B53*	
	*0x4B54 - 0x4B6B*	
	*0x4B54 - 0x4B6B*	
	*0x4B54 - 0x4B6B*	
	*0x4B6C - 0x4B83*	

d) Dungeon GFX# Sets

by DxEdge





Within dungeon editor you can right click and select "Edit Blocks" and then you can select from the drop down menu "Other". This will give you a box that you can put in any number between 000 and 219. Those numbers are the block numbers for the GFX of the dungeon maps. Then you can edit them.

000 > Doors, Staircases etc..
001 > Inner 1
002 > Inner 2
003 > Floors
004 > Dungeon 1
005 > Ganon Tower 1
006 > Triforce Temple
007 > Dungeon 2
008 > Keys, Items etc..
009 > Soldier 1, Beamer
010 > Soldier 2
011 > Witch, Flute Boy (Dark World Form), Oldman etc..
012 > Enemies 1
013 > Special Effects, Fire, Explosions 1
014 > Piece of Heart, Fairy, Whirlpool
015 > Dungeon 3
016 > Inner 3

017 > Inner/Dungeon
018 > Cavern Wall Pieces
019 > Inner 4
020 > Ganon Tower 2
021 > Ganon Tower 3
022 > Title Screen 1
023 > Title Screen 2, Ganon Tower 4
024 > Aghanim's Bed
025 > Dungeon 4
026 > Dungeon 5
027 > Dungeon 6, Table
028 > Inner 5
030 > Ice 1
031 > Hyrule Castle Throne Room
032 > Dungeon 7
033 > Fairy Pond
034 > Dungeon 8
035 > Inner 6
036 > Dungeon 9
037 > Skeleton Dungeon
038 > Dungeon 10
039 > Dungeon 11
040 > Dungeon 12
041 > Ice Boss, Dungeon Tiles
042 > Ganon Tower 5
043 > Tiles
044 > Temple 1 Tiles
045 > Forest Temple 1 (Dark World)
046 > Forest Temple 2 (Dark World)
047 > Thieves Village (Dark World), Chest
048 > Outside
049 > Swamps 1
050 > Swamps 2
051 > Ganon Tower 6
052 > Ganon Tower 7
053 > ??? - Unknown
054 > Pyramid Background (Dark World)
055 > Mountain 1
056 > Water Temple (Dark World)
057 > Floors
058 > Mountain 2
059 > Mountain 3
060 > Mountain 4
061 > House

062 > Outside 1
063 > Outside 2
064 > Title Screen 3
065 > Title Screen 4
066 > Tree (Dark World)
067 > Mountain (Dark World)
068 > Outside 3
069 > Outside 4
070 > Triforce Ending Screen
071 > MasterSword Place
072 > Lost Woods 1
073 > Hyrule Castle 1
074 > Outside East Palace
075 > Sanctuary 1
076 > Lost Woods 2
077 > Outside Kakariko Village
078 > Water Temple (Light World)
079 > Mountain 5
080 > Sanctuary 2, Potion Hut
081 > Hyrule Castle 2
082 > Hyrule Castle 3
083 > Bird Shrine In Kakariko Village
084 > Desert Temple 1
085 > Desert Temple 2
086 > Mountain 6
087 > Lost Woods 3
088 > Mountain Clouds 1
089 > Mountain Clouds 2
090 > Swamps 3, Jump Steps
091 > Swamps 4, Jump Steps
092 > Floors
093 > Dungeon Water
094 > Ice Floors
095 > Mountain Clouds 3
096 > Turtle Rock (Dark World)
097 > ---
098 > ---
099 > ---
100 > ---
101 > ---
102 > ---
103 > ---
104 > ---
105 > ---

106 > ---
107 > ---
108 > ---
109 > ---
110 > ---
111 > ---
112 > ---
113 > ---
114 > ---
115 > Weapons
116 > Items, Objects
117 > Agahnim Thunder
118 > Special Effects, Fire, Explosions 2
119 > Thunder
120 > Game Over, Fairy, Crystal
121 > Special Effects, Fire, Explosions 3
122 > Piece of Heart, Fairy, Apple
123 > Nintendo Presents
124 > Special Effects, Thunder
125 > Items (Light World)
126 > Items (Dark World)
127 > Zora, Whirlpools
128 > Blue Soldier (Dark World)
129 > Thief, Gravestone
130 > Fake Tree
131 > Big Stone, Enemies
132 > Forest Enemies
133 > Sand Enemies
134 > Knight
135 > Sphinx, Bully And Whimp (Dark World Mountain)
136 > Enemies 1 (Dark World)
137 > Enemies 2 (Dark World)
138 > Enemies 3 (Dark World)
139 > Zoras (Dark World)
140 > Enemies 4 (Dark World)
141 > Dead Aghanim
142 > Enemies 5 (Dark World)
143 > Enemies 6 (Dark World)
144 > Electric Barrier, Boss #1
145 > Enemies 7 (Dark World)
146 > Enemies 8 (Dark World)
147 > Enemies 9 (Dark World)
148 > Ganon 1
149 > Enemies 10 (Dark World)

150 > Enemies 11 (Dark World)
151 > Enemies 12 (Dark World)
152 > Enemies 13 (Dark World)
153 > Enemies 14 (Dark World)
154 > Enemies 15 (Dark World)
155 > Enemies 16 (Dark World)
156 > Enemies 17 (Dark World)
157 > Enemies 18 (Dark World)
158 > ---
159 > Enemies 1 (Dark & Light Worlds)
160 > The End Message, Guy
161 > Enemies 1 (Light World)
162 > Enemies 2 (Light World)
163 > Boss #3, Mouldrum
164 > Boss #2, Lanmolas
165 > Title Screen Master Sword
166 > Ganon 2
167 > People
168 > Ending Sequence People (King, Guards, Zelda...)
169 > Girl, Curtains, Sword
170 > Guy Under The Bridge
171 > Dark Boss #3, Mothula
172 > Dark Boss #2, Big Fairy
173 > Dark Boss #1
174 > Dark Boss #4
175 > Dark Boss #5
176 > Dark Boss #6
177 > Dark Boss #7 1
178 > Dark Boss #7 2
179 > Dark Boss #7 3
180 > Ganon 3
181 > Agahnim 1
182 > Agahnim 2
183 > King Zora
184 > Ganon 4
185 > Red Soldier, Cannon
186 > Priest
187 > Soldier 1, Beamer (Same as **009**)
188 > Red/Blue Soldier
189 > People
190 > Shop Owners
191 > Witch, Flute Boy (Dark World Form), Oldman etc.. (Same as **011**)
192 > People
193 > Animals

194 > Town Folks
195 > Town Folks 2, Chicken etc..
196 > Link's Uncle, Sword, Sick-Bugcatcher Boy
197 > Objects 1
198 > Objects 2
199 > Moveable Block Falldown
200 > Agahnim, Girl
201 > Dungeon Map 1
202 > Dungeon Map 2
203 > Warp Bird, Moveable Chest etc..
204 > Town Folks
205 > Item Screen 1
206 > Item Screen 2
207 > Item Screen 3, Contest Rupees
208 > Moveable Wall, Dead King, Chest, Head?
209 > Sword Slash Effect, Fairy
210 > ZZzz..., Shields, Books
211 > Rupee, Bug-catching Net Effect, Staff
212 > Dungeon Map 3
213 > Dungeon Map 4
214 > Dungeon Map 5
215 > Zelda
216 > Girl, OldMan
217 > Sword Smith, Blacksmith Partner (Frog Form)
218 > Intro Sequence Graphics (Compressed in 2bpp)
219 > Intro Sequence Graphics (Compressed in 2bpp)

> To edit the last two (**218**-**219**), you need to open them up in a Tile Editor, but first you need to decompress the graphics using **Lunar Compress**:

In the command line, type: **decomp zelda3.smc spr103.bin c29ba 0 0**

«**zelda3.smc**» has to be your rom and it has to have a **$200 byte header**.
It's already added in the address for you, the calculator that ships with windows is fine for that.

«**spr103.bin**» will contain the decompressed graphics in **3bpp** SNES tile format.
You'll have to use a tile editor **other than YY-CHR** from here, since it doesn't support **3bpp**.

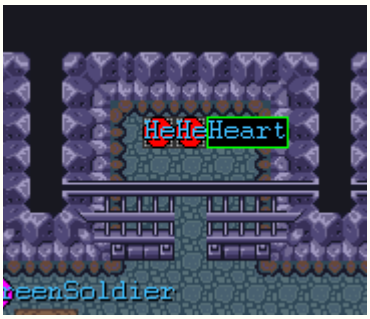
Finally, when you're done with the editing process, here's the command line to reinsert them into the rom, type: **recomp spr103.bin zelda3.smc c29ba 0 0**.

Here's what the untouched GFX tiles 218-219 look like :

Remember, to copy them in an image editor like Jasc Paint Shop Pro or the GIMP, because MSPaint will corrupt the colors, thus you will be losing quality in your graphics and won't be able to paste them back in the editor.

e) Special Notes on dungeons editing

by Euclid and Dude Man



Items must be placed under objects like rocks, bushes or pots. Items you most likely put under objects are hearts, magic bottles, switches or secret staircases.

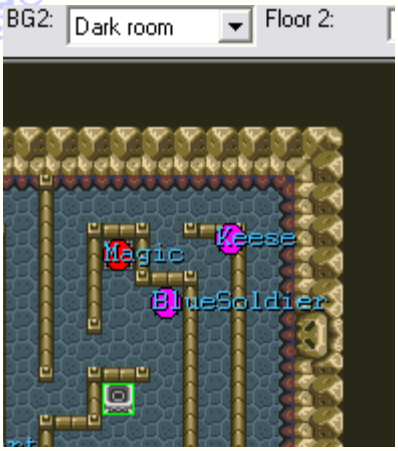
Sprites can contain the same thing as items but they don't need to be hidden underneath an object. (e.g. Heart Pieces can just be laced on the ground in clear open view).

To place a key that will be dropped by an enemy, simply place it in position **00,3C** or **00,3E** on **BG2**. After that an enemy shall drop the key.

00,3C = Small Key
00,3E = Big Boss Key

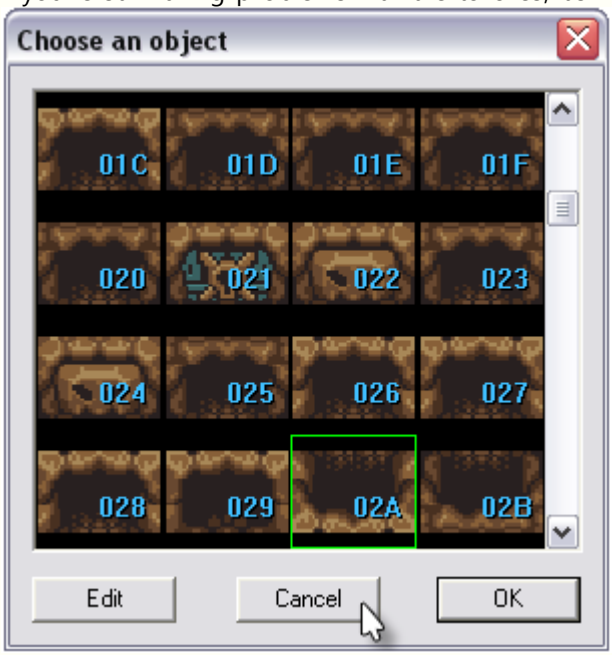
To change the layer that an item/sprite is on you have to select it and press the Hyphen "**-**" key.

To make sure that the order of sprites are correct use the **left and right arrow keys** to move through the sprites. To alter which sprites come first you must press the "**B**" key to move the selected sprite backwards or the "**V**" key to move it forwards.

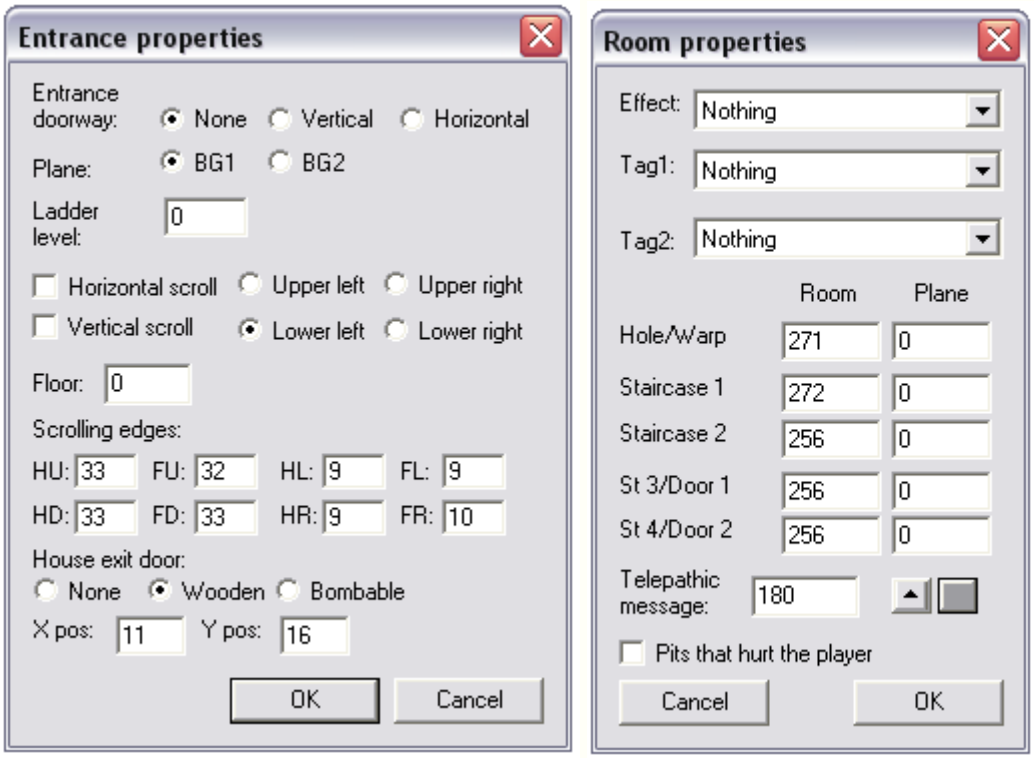


Torches can be very dodgy at times. As long as you have a certain selection of parameters set for that room the torches should work/move fine. (Not sure about this...) Make sure you've got the **BG2** menu set to "**Dark Room**" and the **object** "**FAA**" placed in the room.

If you're still having problems with the torches, it's most likely just HM being buggy as usual.



If you make the insert door box appear (With different doors to use) and instead of selecting **OK** you click **Cancel**, you can sometimes screw up the graphics if you save the ROM. This can be rectified if you clear the room and also clear (and put back together) the layout room that the screwed up room uses.

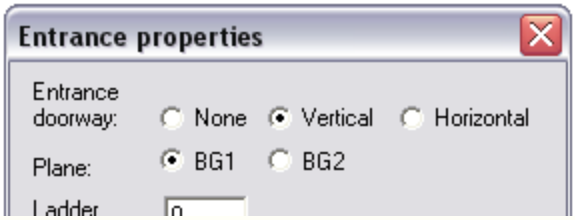


The door at the bottom of Link's room is based on **layer 3** and made up of two different doors (from the insert door menu). These doors are **02A** and **033**.

02A is the visible door (placed on layer 3) - if you have this on its own in a room it will allow link to move from one room to another.
033 is an invisible door (also placed on layer 3) - this door allows link to move out of the room into the overworld and i do believe vice versa.

If you accidentally moved door **033** then that could explain you showing up in the center of the room. Although it's more likely to be the coordinates. The two pictures above are from the original ROM and shows everything as it's supposed to be.

The event where you get the sword of your uncle, has to be done before rescuing Zelda, any time after this, when this event happens, the game will think you just finish the first quest of getting the sword and thus changing all the sprites/etc. back to "**Beginning**", and put rain on the overlay which is very difficult to control since Zelda won't be in the cell anymore.



Planes, which I'm sure you know, are just the **2 kinds of levels** you see in the game. The lower floor is usually **BG2** (Sometimes it is **BG1** too). We don't do it all it in **BG1** because sometimes we want Link to be able to **walk under** some objects/tiles or floors (e.g. See Hyrule Castle **Entrance 05**, it's the perfect example).

Those 2 doors you have there are essential for Link to sometimes (In some cases you don't need the 2 doors) be able to stand on **BG2** when entering another room (must be on the sides). There's a up/down version of it as well but it's not really used much for some reason. (Not sure...)

The sprite of Sahasrahla **(Mutant)** can be relocated in other rooms without encountering problems.

If you place it an important quest item (e.g. Hammer) in a small chest, while in game it will not show a small chest when opened but a big one. A small chest item in a big chest caused the chest to turn into a small one.

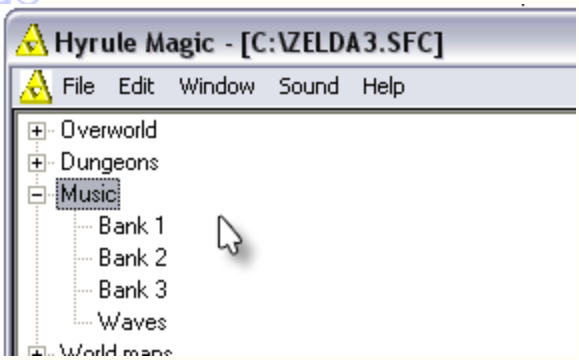
Here are the data for the X Scroll, Y Scroll and the exit door for entrances:

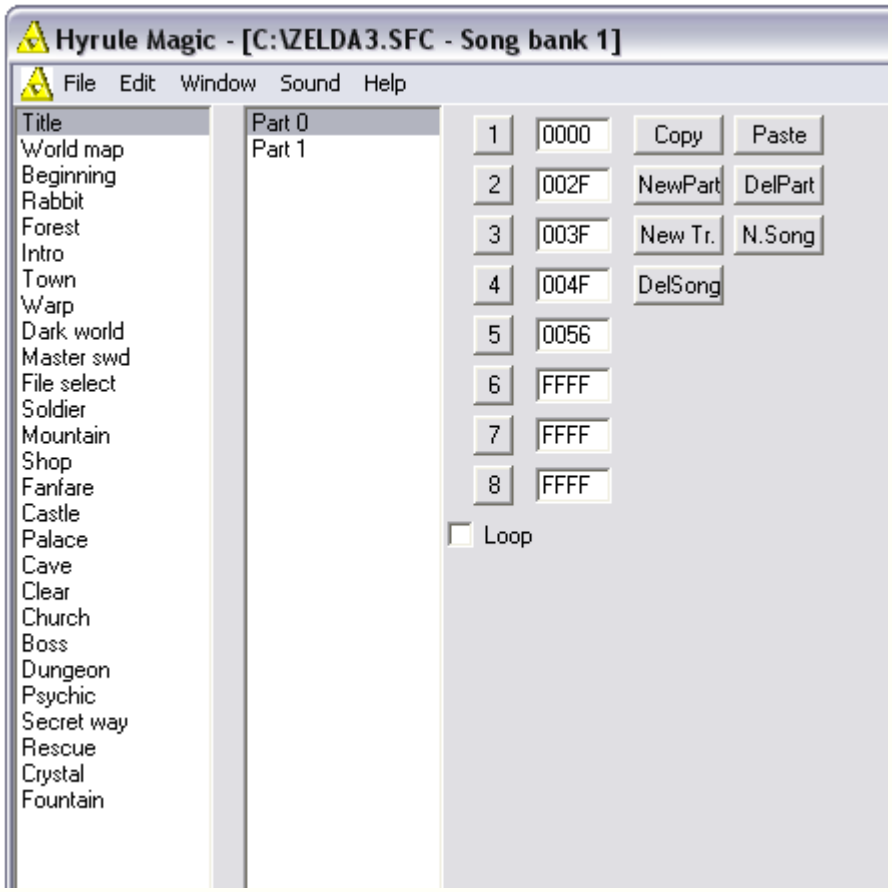
Normal entrance addresses are:
1504F - X Scroll
14F45 - Y Scroll
15924 - Door location

Starting location addresses are:
15DB4 - X Scroll
15DC2 - Y Scroll
15E32 - Door location

9) Music

by Sephiroth3, The4thLime, Euclid and MathOnNapkins





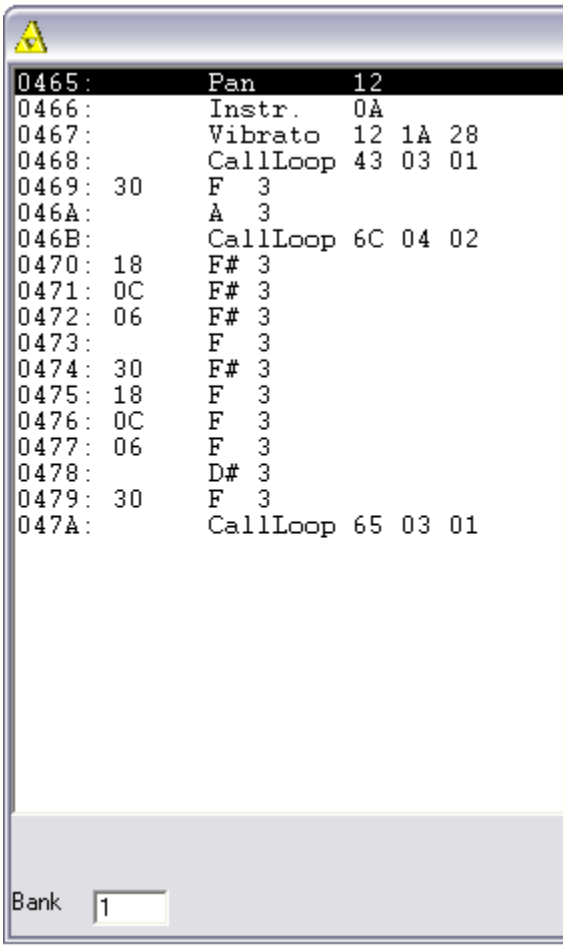
a) The basics

by Sephiroth3

This editor edits music, like its name suggests. The game music is stored in **3 banks**. The list of songs is on the left. If the song exists in the edited bank, a selector for the song parts will be shown. The **loop checkbox** indicates whether the song loops. If checked, the field next to it contains the number of the part that is looped to. When a part is selected, the **button 1-8** can be pressed to edit a channel. The part ends when all channels have ended. The starting address is shown on the

right. This is an unique identifier assigned to each command by the editor. **FFFF** means that the channel is unused. To create a new sequence of commands, press **New track**.
The **Copy** and **Paste** buttons mark and reuse song parts. **New part** inserts a part before the currently selected part or at the end, if no part is selected. **Delete part** deletes a part. New song and Delete song are used to add and remove songs.

Generally speaking, the music data is divided into track data ($D000 in spc ram), samples ($3C00 in spc ram), code ($800), and most work ram is used from $0 to $3FF. There's other stuff too but I don't quite have names for them yet.



b) The track editor

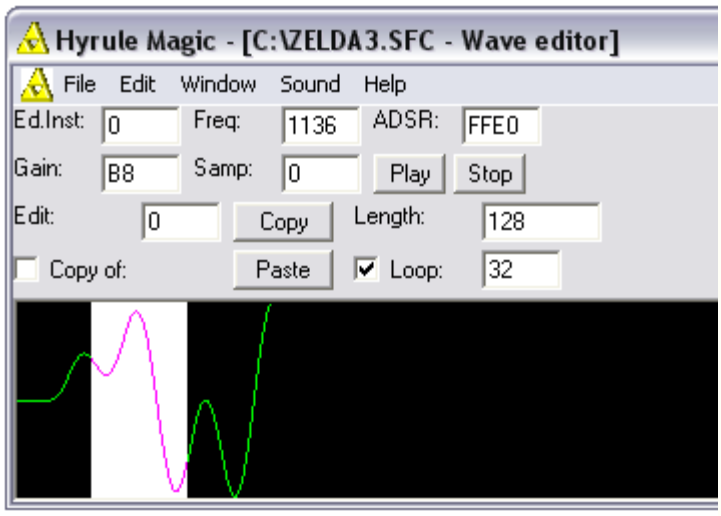
by Sephiroth3

The addresses are shown on the left. These are not consecutive and may change when you save and open the rom again. The next field, if used, sets the new note length. The third field, if used, sets the new key on-time in the high nibble and the volume in the low nibble. If this is used, the second field must also be used. The fourth field specifies the command or note. Two hyphens indicate that there is no command. Use **Z,S,X,D,C,F,V,G,B,H** or **N** and the **digits 1-6** to enter a note or press **M** to enter a command. You can press **Enter** when a CallLoop command is selected to open the destination track. The remaining fields specify command parameters. These are dependent on the command. Fields can be changed by clicking on them. Press the **left mouse button** to decrease the value and the **right mouse button** to increase it. **Holding shift** changes the field by **16**. You can also type the new value with **0-9** and **A-F**.

The other keys are:

Alt+L > Marks commands
Alt+U > Unmarks commands

Alt+M > Moves commands
Alt+C > Copies commands



c) The wave editor
by Sephiroth3 and Euclid

At the display at the bottom, press and hold to select a portion of the edited wave. Press **Delete** to delete the current selection. If nothing is selected, all of the wave will be deleted or copied. Make sure that the loop point and the end point are multiples of **16**. Otherwise, Hyrule Magic will stretch the sample and add zeros at the beginning. At the very top part of the editor, the instruments used for music can be edited.

> The music editor can only play the specific **24** instruments which are mapped in the wave editor.

> The notes for the **24** instruments range from **G2** - **B5** (or **B2** - **A5** - Unsure about this...), some of them will play in HM but won't play in the emulator (e.g. playing Instrument **0E** with note **A5**).

> You can only play around with those notes, pauses etc... but you certainly can't achieve midi quality due to the limited amount of instruments for the whole game.

> If you're for simple changes to the song, you can use the music editor in HM, but if you're talking about something big like rewriting the songs then an hex editor is your friend.

> Quite a few SNES games actually use one or two channels for the sound effects. Sometimes these channels are shared with the music, but sometimes they are isolated. For those games, you can easily achieve what you want by turning the music sound channels off and leaving the sound effects channels on.

> Beyond that, in the case of the SNES, you're going to have to hack the SPC700 program in the ROM. That's probably the best way. That program WILL distinguish between music and sound effects. You could disable the part that plays music and only allow sound effects to play. That seems like the best approach.

> However, you're going to have to learn a heck of alot to do it. You're going to have to find out where the SPC700 program is loaded in the ROM. Then you'll have to disassemble it and try to figure out what it's doing. You'll have to find what needs to be modified to disable music and allow sound effects.

> Then you're going to have reassemble the modified code and then adjust the loading routine to account for additional bytes or any change in size of the SPC700 program. In fact, there probably will not be room for a larger program. You'll have to change it to load from another free space in the ROM.

d) Special notes

by The4thLime and Euclid

> About the numbers preceding a note. The first number I found to be the speed at which all following notes play at, and the note after that controls the volume of all the notes after that. So if you were to find something like **06 D2 A# 4**, you would read it as such:

___06_____D2_____A#_____4
____|_____|_____|_____|
Speed = 06____Volume=D2___Note = A#____Octave= 4

> **Speed:** The lower the number, the faster the song is played.
> **Volume:** The higher the number, the higher the song is played; **Max:** $C8 (200 decimal)

> When comparing the instrument/sample from the waves menu to the instruments in the composition editor (the banks), you need to convert the instrument number from **decimal** (in the waves menu) to **HEX** (for use in the composition editor).

> I have yet to find out what the numbers after commands such as vibrato and echo mean, they're probably just the effect level, and maybe how long it lasts, or something.

> Transposing is nothing special really. It's just like transposing in regular music, or in simpler terms, changing the key. It basically changes the overall sound of the notes in relation to each other. There are two transposition functions, Transpose and G.Transpose. Transpose + a number change the key of the song to whatever the number is. G.Transpose simply changes the song back into the key of G, which is the standard key in music. I don't really know how to explain transposition to anyone else who doesn't really know much about music.

> The music data is compressed, yes compressed. That means that the data mapped into the SPC is decompressed in order for you to work on samples. I've seen the data load from the ROM, it is not decompressed before being mapped in to the SPC during gameplay. It is either decompressed by the SPC itself, or the program that the SPC runs.

> Something that I have yet to try (although in theory should work) is to expand the rom, and put music data at the end of the rom. Now you might be thinking, "But the pointer for music is only 2 bytes..." Well, yes this is true. However, you can use the CallLoop command in the music editor. An example use of this command would be something like "**CallLoop EF 1E 04**". Well, obviously with the CallLoop function, you can access any part of the rom.

> If you write music at the end of the rom, you don't have to worry about the data running into other songs, because it's all freespace. No worries there.

> For the **CallLoop Command** it would be displayed like this:

CallLoop 3D 21 04

Well the first two numbers would go to address **213D** in whatever bank it is. The number after it (the **04**) is **how many times** that will play. So in this instance, it will play whatever is at **213D** four times.

e) N-SPC overview
by MathOnNapkins

I was just going to describe the format from the ground up... all right!
N-SPC is a tracker type of music driver used in lots of first party Super Nintendo games. If you learn how to use it, you can potentially import music and instruments from other games that use the N-SPC driver. E.g. you could take music from Super Mario All Stars and implant it into Kirby Super Star

Is there a quick way to find where that data is stored?

Yes. That requires a general understanding of how the SPC <---> SNES communication works
If I'm looking at a new game, I want to know where the SPC driver is at. Now, the catch is that the N-SPC drivers for each game are not *identical*, but they're pretty close. Let's look at Super Metroid for exemple.

I'm going to use Brinstar1.spc as an example. It's the green brinstar theme (pink too). SPC drivers almost always look like this: two hex bytes 20 CD

That's the start of the one in Super Metroid. Since that's the start of the driver, we can easily search for that in the rom itself but you will likely find more than one instance of "20 CD" in the rom. So what you do is take a few more bytes... in this case:

20 CD CF BD E8 00 5D AF

In a hex editor, that sequence of bytes only shows up once.

Let me back up a second and say that the "driver" is a term that refers to code that runs on the SPC-700. The SPC-700 is a CPU separate from the SNES's main CPU, so if we find this data, we have only found the code that the SPC-700 runs. However, in most N-SPC games, the SPC related data is all lumped together: you find the driver, you find the rest of the music data.

This is true of Link to the Past and Super Metroid, at the very least. I imagine it's the same for Super Mario Kart, etc. Generally speaking the 20 CD will be aligned on a page boundary, like 0x100, 0x200, 0x300, ... etc. SPC files have a 0x100 byte header at the beginning so it's technically 0x1500 for Super Metroid, as far as the SPC RAM goes (The SPC has 64K of its own RAM).

Using Geiger's SNES9x debugger, one can then use read breakpoints to isolate all the music data

The location of the driver in the ROM is 0x278108 (no header). This process should be fairly repeatable on other games since the driver is a logical unit that is, a "chunk". The start of it is just that, the start of a chunk. So in the ROM, we look to the left of that 20 CD and what do we spy but a sequence like this:

E2 41 00 15

This is two 16-bit numbers that tells us important information
1. The length of the chunk is 0x41E2 bytes
2. The location in the SPC's RAM that it will be transmitted to is 0x1500

For the mathematically inept, 64K of RAM ranges from 0x0000 to 0xFFFF, and that's it. Anyways, this chunk will be written to 0x1500 and will stop at 0x56E1.

The length of the "driver" "chunk" varies from game to game, as does its contents but the music data is all basically the same format. The drivers usually differ in how they do sound effects, not music. Super Metroid has an unusually long driver chunk for an N-SPC game… to me, that tells me that it has a very complex sound effects system. Which makes sense, given its late release date and how cool most of the game sounds.

Sound effect wise, Kejardon would probably have more info on that. He mentions some stuff on that in http://jathys.zophar.net/supermetroid/kejardon/SPCRamMap.txt I think.

In your hex editor… we want to skip 0x41e2 bytes forward, there we will find the next chunk.
These chunks are usually consecutively laid out in the ROM. That will bring us to ROM address 0x27C2EA. You should see the following bytes at that location

93 0C 20 58 0E 53 ….

What 93 0c 20 58 means is that it's $0C93 bytes long and going in $5820.

The real issue, is that we don't know what this data is, but we should make a note that it exists… it's clearly relevant.

So we continue moving on. From the 0E byte, we jump 0x0C93 bytes forward and arrive at location 0x27CF81. This next chunk is 0x00A0 bytes long, and gets stored to 0x6D00

Q: *Isn't 27C2EA + C93 = 27CF7D?*

A: You're right it is, but we need to jump from the start of the previous chunk, which would be 4 bytes after 27C2EA. The length and SPC addresses don't count as part of the chunk because they're not going to be written into the SPC RAM… lacks sense I know.

What it's saying is 0x0C93 bytes get written to location 0x5820, and after that are our 0x0C93 bytes to write… so we need to jump from the start of the actual chunk, not the …. "header" data.

Anyways, we're going to jump again. This time from 0x27CF85 in the rom (note: again I've incremented by 4 bytes from 0x27CF81 so we jump ahead 0xA0 bytes and arrive at 0x27D025.

At this point, this is probably getting boring, it will pay off. I'm going to skip to my documentation of the Super Metroid driver (the specifics of the driver tell you what each chunk does).

Q: *Some enemy sound effects change when you change the song in Super Metroid, does this mean that these songs are in a different chunk? When the game changes songs, it loads different instrument samples, sometimes enemies use those samples.*

Songs and sound effect data don't have to be in the same chunk. But if it loads a new song, chances are it's loading new ADPCM samples… which will in turn affect the sound effects.
That's something you guys will have to figure out since this tutorial is mostly on how to locate music data and change it… not sound effects.

I know how to do that for LTTP, but I'd be lost in Super Metroid (sfx). Anyways where were we?

I'm at 0x27D025. This gives us a header with the following data: 0x91E0 bytes long, written to address 0x6E00 in the SPC. So like good little boys we skip ahead over this again to 0x286209. Now here's where the buck stops! We've hit a block with length 0x0000. This is how the SNES CPU knows to stop transmitting data to the SPC.

So we've covered quite a lot of terrain in the ROM. Nearly two 32K banks of data. Kej has a bit more info on these chunks:

```
1500 - 56E1 is the main SPC engine. Copied from ROM at CF:8108 (278108). Some sections may be
changed
1B62 - 1B9F is a jump table of sorts; Song data is interpreted with this. Not changed.
1E1D - 1E31 is panning volume multipliers. Not changed AFAIK.
1E32 - 1E51 is FIR coefficients. 4 groups of 8 bytes. Seems like it'd be changable, but not changed
AFAIK
1E66 - 1E7F is the pitch table. (085F 08DE 0965 09F4 0A8C 0B2C 0BD6 0C8B 0D4A 0E14 0EEA 0FCD 10BE)
5800 - 5807 Note Sound Length percents
5808 - 5817 Note Volume percents?
581E - 5??? Song start addresses for 40. Maximum possible is 80, but usually only 6-8 are used.
5??? - ???? Song note data is stuck here, right after the end of the song start addresses.
6C00 - 6CE9 (Not certain about end address) 6 byte entries for 18F9. Voice Source #, ADSR1, ADSR2,
Gain, and then two bytes stuck in 0220+x (x = 2*channel). Seems to have 1 entry per source #.
6CEA - 6CFF Unused?
6D00 - 6DFF is the sample table. Seems to stop at 6D96 (though 6C00 suggests 6D9C). *Must* stop by
6DA8 - that's all the table at 6C00 can support.
???? - F??? Instrument sample data. Pointed to by 6D00 sample table.
```

The last chunk was really large, 0x91E0 bytes, to be exact. Usually a chunk that big will be your sample (ADPCM) data. That's what most of you would refer to as "instruments". It's sound data that has been prerecorded before being inserted into the game and it's manipulated to produce sound effects, music, etc.

This meshes with Kej's data because he says that the sample table is at 6D00-6DFF. The sample table usually is right before the sample data, but not necessarily.

Let's go to the previous chunk... that was 0x00A0 bytes, written to 0x6D00. There's your chunk that describes the sample table:

```
6D00 - 6DFF is the sample table. Seems to stop at 6D96 (though 6C00 suggests 6D9C). *Must* stop by
6DA8 - that's all the table at 6C00 can support.
```

I don't really want to get into an indepth discussion of the sample table, but let's look at some of the data in it just to make sene of it. The sample table is a table of "pointers" to the ADPCM samples in the SPC RAM.

Each entry is 4 bytes. The first two bytes specify the starting point of the sample, while the last two bytes specify the loop point of the sample. Let's look at the first two entries in the sample table... at 0x27CF85 you'll see 00 6E C4 73 C4 73 88 79. The first four bytes correspond to "instrument" 0. The sample starts at 0x6E00 and loops at 0x73C4.

My intuition tells me that this sample doesn't loop at all as its end point is the same as the start of the next sample... "instrument" 1 which starts at 0x73C4 and loops at 0x7988.

Q: *I don't really follow the logic about: loop point = next sample, so must not loop...*

A: Looped samples are something I don't want to get into at the moment because well be dealing with the DSP at that point. A basic description would go as follows.

The sample is a waveform which can be very short or very long. Let's say that instrument 0 is a piano or rather, a recording of a piano key, either real, or synthed. When you hit a piano key, it makes a percussive noise when your finger first strikes it... but the sound it makes when you hold it down, is different from the sound it makes when you hit it, yes?

[[hit] [held sound]]

Let's say that the hit portion lasts 0.1 seconds and the held sound part lasts 0.5 seconds.
When you input your instrument data, you'd want to make this sample table up in a way that tells it what byte the held sound starts at otherwise if you loop the sample, it will sound like someone hittting a piano key over and over and over again.... rather tahn the sound of someone holding a piano key.

Some samples in the game don't actually ever get used long enough to play "held" notes and the programmers didn't give them that data. So if you tinker with the sound engine and try to make it use one, it sounds like the "hitting over and over again".

Some samples are one offs, they aren't meant to loop. So the loop position is just set to the end of the sample. If you set the base address to the same address as the loop address you really will be doing the whole sample over and over again. e.g. 0x6E00 then 0x6E00 in this case.

Q: *Does this mean that any note would repeat if you let it play long enough? But if it loops, it's told at some point to jump into the sample at an appropriate point.*

A: That's more getting into ADSR settings. I don't want to get into that right now. You can make a sample play forever with ADSR but it won't do that by default. Read anomie's DSP guide if you want to know how that stuff works. It's very thorough but at times hard to understand.

Anyways, so we know what that chunk did. Moving backwards again at 0x27C2EE which is the chunk that will be written to 0x5820. According to Kej... this is the song note data.

What he means, is that this is the tracker data... this is the song data! It consists of commands to play musical notes, to change volume, and a shitload of other things. This is the stuff we're interested in.

Now, as far as we know, this is only one set of song data. There's probably others in the ROM nearby to the chunks we're aware of... but! We now know where the song data is read from in the SPC RAM. For exemple let's pick a super metroid song: Maridia2, the song you hear when you first get into Maridia. In my set it's Maridia2.spc. I probably got the original game spc file from zophar.net ages ago but you can get the hole set from snesmusic.org/v2/

you know where the tube you blow up is, just go two screens up from that then load the spc file that corresponds to that in your hex editor (in mind Maridia1.spc is not the one you're thinking of). I'm just clarifying, so we're all in the same file to be totally clear, it's not the music you'll head in the room right before dragon.

I've opened up my file in my hex editor for convenience sake i'm going to chop off the 0x100 byte header and save it as a different file name... like Maridia_test.spc.

So I do some checking and I look for 20 CD to make sure the driver's at 0x1500 low and behold, it is!

Anyway, the point of chopping off the header was to make sure that the 0x0000 in the file was the same as 0x0000 in SPC RAM. That's what SPC files are, they're dumps of the SPC's RAM along with the DSP (digital sound processor) variables from that an SPC player can begin playing the music.
If you go to 0x5820 in the SPC file you should see the following 0E 53 6E 54 69 55 97 56 ...
What we want to do is see if we can locate this data in the ROM. If not, we'll have to be slightly more clever.

Hehe. That works! That happens to match the chunk we were looking at in the ROM earlier! No surprise there. One thing to keep in mind is that N-SPC can store more than one song in RAM at a time. In Super Metroid, it usually only keeps 5 songs in RAM at a time (SPC RAM) but a game like Zelda LTTP keeps like... 10-14 songs in RAM at a time.

It's just a different way of utilizing the engine. LTTP emphasizes the ability to change the song on the fly whereas Super Metroid probably uses higher quality, larger samples to produce better music. But since the sequence data and the sample data have to be in RAM at the same time.

There's always a trade off. A game like Chrono Trigger only keeps *1* song in SPC memory at a time and afaik, doesn't keep any sound effects in RAM as in, it streams them. But CT is pretty advanced and pushes the system pretty hard to its limit.

Anyways, it's kind of hard with most spc players to change the song. I was going to try to show you guys how to do that. The problem is that they don't simulate input from the SNES CPU to the SPC or rather they do... The point is that a SNES side variable will control which song is currently playing.

I look at the Maridia spc file to find out the number of the song that was playing... that's 06... Where can you find that? At locations 0x0000, 0x0004, and 0x0008 in RAM in the SPC file. Since it's song 6, we look at the bytes at 0x5820 and we skip over 6 times 2 bytes because these are all pointers to the data for each song (pointers are two bytes).

We should end up at 0x582C in the SPC file. The data there is 5A 58, this means we need to go to 0x585A to get to the rest of the song data.

If you haven't noticed yet, the SPC-700 is also little endian, so you have to reverse the bytes to figure out the pointers.

So we go to 0x585A and we see 7D 5F, which is yet another pointer to 0x5F7D. Note that this data is called "part" data. Some songs have multiple parts which are organizational in nature. If you follow this data, at 0x5F7D,you should find the following bytes FA 27 E7 0C E5 B4 .... etc

This to me, looks like what I call track data. It goes "song data", which is a pointer to "part data" which could have more than one pointer, and then each part points to "track data". This might not have been the best example as I can only find one track, but I'll explain anyway, observe:

```
Command_Jump_Table:
0EEA: dw $0C66   ; command E0(1) (instrument)
0EEC: dw $0CBF   ; command E1(1) (pan)
0EEE: dw $0CCD   ; command E2(2) (pan slide)
0EF0: dw $0CE6   ; command E3(3) (vibrato)
0EF2: dw $0CF2   ; command E4(0) (vibrato off)
0EF4: dw $0D0D   ; command E5(1) (global volume)
0EF6: dw $0D1C   ; command E6(2) (global volume slide)
0EF8: dw $0D2E   ; command E7(1) (tempo)
0EFA: dw $0D33   ; command E8(2) (temp slide)
0EFC: dw $0D45   ; command E9(1) (global transpose)
0EFE: dw $0D48   ; command EA(1) (tranpose)
0F00: dw $0D4C   ; command EB(3) (tremelo)
0F02: dw $0D58   ; command EC(0) (tremelo off)
0F04: dw $0D79   ; command ED(1) (channel volume)
0F06: dw $0D82   ; command EE(2) (volume slide)
0F08: dw $0D9F   ; command EF(3) (call loop)
0F0A: dw $0CFD   ; command F0(1) (vibratostep (????????)
0F0C: dw $0D5B   ; command F1(1) (pitch slide to)
0F0E: dw $0D5F   ; command F2(1) (pitch slide from)
0F10: dw $0D75   ; command F3(0) (pitch slide stop)
0F12: dw $0D9B   ; command F4(1) (fine tune)
0F14: dw $0DC2   ; command F5(3) (echo control) [EON, EVOLL, EVOLR]
0F16: dw $0DF9   ; command F6(0) (echo silence) [EDL = 0, EVOLL = 0, EVOLR = 0]
0F18: dw $0E00   ; command F7(3) (Echo and Filters) [sets EDL, ESA, EFB, VxFIR]
0F1A: dw $0DD8   ; command F8(3) (echo volume slide)
0F1C: dw $0E9B   ; command F9(3) (pitch slide)
0F1E: dw $0E68   ; command FA(1) (percussion offset)
Num_Parameters_For_Commands:
0F20: db $01     ; command E0 number of parameters
0F21: db $01     ; command E1 number of parameters
```

```
0F22: db $02     ; command E2 number of parameters
0F23: db $03     ; command E3 number of parameters
0F24: db $00     ; command E4 number of parameters
0F25: db $01     ; command E5 number of parameters
0F26: db $02     ; command E6 number of parameters
0F27: db $01     ; command E7 number of parameters
0F28: db $02     ; command E8 number of parameters
0F29: db $01     ; command E9 number of parameters
0F2A: db $01     ; command EA number of parameters
0F2B: db $03     ; command EB number of parameters
0F2C: db $00     ; command EC number of parameters
0F2D: db $01     ; command ED number of parameters
0F2E: db $02     ; command EE number of parameters
0F2F: db $03     ; command EF number of parameters
0F30: db $01     ; command F0 number of parameters
0F31: db $03     ; command F1 number of parameters
0F32: db $03     ; command F2 number of parameters
0F33: db $00     ; command F3 number of parameters
0F34: db $01     ; command F4 number of parameters
0F35: db $03     ; command F5 number of parameters
0F36: db $00     ; command F6 number of parameters
0F37: db $03     ; command F7 number of parameters
0F38: db $03     ; command F8 number of parameters
0F39: db $03     ; command F9 number of parameters
0F3A: db $01     ; command FA number of parameters
```

The first command in the track data is FA, so we look at the above table: FA(1) = percussion offset. This is the N-SPC driver's command set. Most SPC drivers have a command set. Squaresoft had one, almost everyone has something like this when it comes to playing back music (i.e. tracker data), so the question at this point, is does this data make sense?

FA 27 would say "play drum number 27". Now, given that there's no drums in the Maridia song we're looking at chances are we've made a mistake. So we backtrack and even do a little spc editing. Let me get it set up. You'll have to reload the original maridia.spc file and make a copy of it again. It will need the header.

In this file I'm going to go and edit the 5A 58 that points to the song, to something else. Some nonsense number. If the song refuses to play, I know I've hit the Maridia song.
Note that in the unmodified spc file, the offset will be at 0x592C rather than 0x582C. Anyways
I verified that it was actually the previous entry. A song number of 06 means we need to go to the 05th entry in the song table. That's why nothing made sense (because a song value of 00 indicates to not change the song at all!).

Trying to get back on track, back to the truncated (headerless spc file), at 0x582A in that file we see 29 60, which is a pointer to 0x6029. We follow that pointer and see 43 60 33 60 FF 00. This is the part data. 43 60 (0x6043) is a pointer to part 0. 33 60 (0x6033) is a pointer to part 1 Now... What the fuck is FF 00 (0x00FF)?

Afaik, or can remember, that just means to repeat the previous part indefinitely since the Maridia song loops indefinitely. That makes sense if it was 00 00.... I think it would probably just stop playing some songs will do that, like Samus' Item pickup theme.

And as a matter of course I just verified that it was 00 00, I don't have full documentation of that laid out at the moment. Moving on, part 0 is at 0x6043, as mentioned before. At this location is the "track data" which consists of 8 "track pointers" 7A 63 AE 63 E5 63 and the rest are 00's. This means that the song uses 3 channels of the SPC the first three, even each pointer points to data for each channel.

Now this is only part 0, keep in mind... part 1 might use more channels given how quiet that song is when it starts out. Let's look at the pointer for channel 0... at 0x637A you see FA 27 E7 0F E5 32 E6 A0 96 E0 1F ... Now we run into the same situation as earlier. FA 27... what is this? Well... the best explanation I can give is that Super Metroid has a slightly different music driver than the others. Unless it really is using the percussion command... LTTP doesn't ever use it, but super metroid might.

Anyways, what FA 27 means is: set the percussion offset to 27. That means that samples 27 (instruments 27) and above are for use with the percussion subsystem. I dont' want to get into the percussion details

Moving on to the next bytes: E7 0F. Let's look at the table I pasted in earlier.

```
0EF8: dw $0D2E ; command E7(1) (tempo)
```

Command E7(1) in the syntax I've used indicates it takes one parameter because there's a 1 in the parentheses, so this means, set the tempo of the song to 0F. Chances are that if you change this byte you'll hear a difference in tempo... but you'll have to use an unmodified spc file because spc players require the header to play.

Channel 0 is the channel that usually has the configuration commands, you wouldn't set tempo on channel 1 and global volume on channel 0. It just makes it easier that channel 0 has them all, but you could use the other channels to issue global commands that affet all channels if you wanted to.

Moving on, E5 32 is the next command.

```
0EF4: dw $0D0D ; command E5(1) (global volume)
```

This sets the over all volume of the SPC to 0x32 E6 A0.

```
0EF6: dw $0D1C   ; command E6(2) (global volume slide)
```

Parameter info: byte1 - ticks it takes to get to target volume, byte 2 - target volume, this is all from Zelda_SPCv5.log on math.arc-nova.org. As you can see, I can make sense of Super Metroid's music by making reference to Zelda's even though the games were developed 3 game apart.

What does the global volume slide do? It has a target volume. I also forgot a parameter... it's E6 A0 96. So we're trying to get to a volume of 0x96 and the current volume is 0x32, remember the driver will keep gradually increase the volume to 0x96 over 0xA0 ticks.

The amount of time that 0xA0 ticks takes depends upon the tempo. I don't have exact calculations available, but I'm sure you could all figure it out if you really wanted to. Point is, it's all relative, the tick is the smallest amount of time you can play a note.

The SPC-700 has internal timers that can be set to different frequencies. That's how tempo is controlled. E0 1F is the next command.

```
0EEA: dw $0C66   ; command E0(1) (instrument)
```

So this says, set the instrument to 0x1F. I don't want to get into it yet, but "instruments" in N-SPC aren't the same thing as samples “instruments" reference a sample but consist of additional configuration parameters.

If you want your instrument to sound different, you configure the instrument table, which is a chunk we have not seen yet! In a nutshell, it provides settings for ADSR, the sample to use, and a parameter to "tune" the sample to a certain note. F4 96 is the next command.

```
0F12: dw $0D9B   ; command F4(1) (fine tune)
```

It's a single parameter command that provides a fine tuning (pitch alteration) to the instrument. Next command : EA 00.

```
0EFE: dw $0D48   ; command EA(1) (tranpose)
```

Transpose is a term that means to shift a tuning up or down one or more half notes.

e.g. a transpose of 01 would mean that all the old A notes would become A#'s, B flats would become C's, D#'s would become E's etc. Don't want to get mired in musical language.

Anyways, setting the transpose to 00 means, there is no transpose! This is merely initialization, to ensure that no transpose is being used, as it could make the song sound funky if one of the channels was out of tune. Next command : ED 78.

```
0F04: dw $0D79   ; command ED(1) (channel volume)
```

This sets the channel volume to 78. There is global volume, and then there is channel volume.
I hope it's pretty clear what the difference is... it can make one instrument sound quieter while others sound louder, etc.
Next command : E1 0A.

```
[0EEC: dw $0CBF   ; command E1(1) (pan)
```

Q: Will global volume affect an instrument that has the lowest possible channel volume?

A: yes, it should. Global volume is a base volume. It's kind of icky and complicated and it works off a bunch of multipliers. To know the details you'd have to consult the code of the driver when I say it's a base volume, I mean it's a base "multiplier" e.g. if global volume is 05 and channel volume is still 00 (I think). Because they get multiplied together, but as long as the channel is still audible on some level the global volume will affect it.

Anyways, panning. Panning is the balance of left and right volume in a stereo sound setup. The
N-SPC has 20 levels of panning. 0 means that all the volume for the channel is in the left speaker and 0x14 means that all the volume is in the right speaker. 0x0A is perfectly balanced pan, equal volume between the two speakers.

An easy example is to listen to the final Lavos song in Chrono Trigger, like the final fight... it pans left and right all the time. It's also used for enemy sound effects probably. LTTP uses panning but only three levels of it (far left, middle, and far right). Not sure what Super Metroid does. Next command : F5 07 46 46. This is a rare 3 byte command.

```
0F14: dw $0DC2   ; command F5(3) (echo control) [EON, EVOLL, EVOLR]
```

This one involves a little knowledge of the DSP. Basically, the first byte will get written to EON, which is the echo enable register on the DSP. 0x07 is a bit pattern... meaning, enable echo on channels 0, 1, and 2... but, since we're using only channels 0, 1, and 2, this makes perfect sense.

The second two bytes set the echo volume for the left and right speaker since the echo sound gets mixed in after the rest of the sound has been generated it can be independently configured.

As you probably noticed, a lot of these numbers for volume, panning, etc.. are pretty arbitrary. They're not percentages, they're driver specific values. Chrono Trigger, for example, uses completely different numbers that provide a completely different scale of volume output. 7F in one might correspond to 4A in another...

Moving on. I know we're going through a lot of commands, but you won't see nearly as many on the other channels, this is all initialization. Next command: F7 02 60 00.

```
0F18: dw $0E00   ; command F7(3) (Echo and Filters) [sets EDL, ESA, EFB, VxFIR]
```

This one is getting even more hairy. This sets the EDL (echo delay) to 02. What that means in layman's terms is how large the echo buffer is setting EDL to whatever the hell you want is a bad idea. I suggest keeping it at whatever the game keeps it at because the echo buffer uses SPC RAM. If set to too large a buffer, it will likely overwrite other data too!

Memory usage is typically tight in SPC programming, so you will have 99.999999% of chances to fuck something up. Just follow the game's lead... set it no higher than the highest value you see.

The second byte affects EFB (echo feedback). I don't totally understand this register, but do understand that if you set it too high it can make your ears bleed as the echo signal saturates and sounds like distortion from hell. Here it's set to 60 *shrug*, apparently that works well enough, but it might not work well in another song. Be careful and don't kill your ears.

The third byte sets the Fir Coefficients. Fir is part of the echo system. It's kind of like running the previous echo samples through a polynomial. It can create a different sound depending on how the Fir coefficients are configured. I don't totally understand this, see Anomie's DSP doc if you have any questions. What this does actually is picks from a set of FIR coefficients.

For example, in LTTP, it takes the byte that's passed in (00 in this case) and multiplies it by 8 then uses it as an index to load from a table in SPC RAM. That table consists of FIR coefficients for each *channel* one byte per channel. LTTP only has 4 different sets of FiR coefficients, so that's 0x20 bytes.

But... I don't see why Super Metroid couldn't have more than 4, that would require some investigation. I'm personally interested in modifying the N-SPC to allow input of arbitrary FIR coefficients from the SNES side, but I don't know if I'll ever get around to that. I think I might be able to implement a reverb type of effect... but anyways, moving on.

ED 46 sets channel volume to 46. Either my memory is bad, or we already set channel volume. Unfortunately stuff you see out in the field doesn't always make sense, so just roll with it.

They generated this data with an automated tool most likely, I'm sure it wasn't the most intelligent tool ever. Next command : EE A8 78.

```
0F06: dw $0D82   ; command EE(2) (volume slide)
```

Like the global volume slide, this sets the volume over the course of 0xA8 frames
and it will end up at 0x78 (it started at 0x46 of course).

All right, I'm sure it's been 100% exciting so far but we're going to move on to non commands since the data stream I'm reading is no longer commands but notes and other data.

Okay, 48 7F A7 60 C8 E5 96 ... note the lack of bytes that are 0xE0 and higher until the E5 of course. 0xE0 and up are commands. The commands that are avaialble on each N-SPC driver can vary a light amount. LTTP is missing two SPC commands (FB and FC), but afaik that's because you never pause the music in LTTP... it's always playing.

Here's how it works. If the byte you come to is not a command, you look at it and see if it's
<= 0x7F. In this case 48 is <= 0x7F. Small code snippet:

```
0B9A: 30 20     BMI   $0BBC               ; if(track_byte & 0x80) goto $0BBC;
0B9C: D5 00 02  MOV   $0200+X,A   ; else store the byte to $0200+X
```

That's SPC ASM. BMI is of course "branch if negative", so if it ends up at 0B9C. That means that the byte was <= 0x7F or, as expressed in the comments, (track_byte & 0x80) was zero. The question is, what is $0200+X?

```
; $0200+X = note sustain time (when a voice is keyed on, this is how long it takes to reach release state)
```

Note that $0200 could be something entirely different in Super Metroid's drivers. RAM addresses for things do not tend to be consistent across versions of the driver, but the N-SPC song data and format does!

So anyways, N-SPC implements a sustain time that is in addition to the DSP's sustain system.

It basically means, set the percentage of the note you want to hold this for (e.g. setting it to 50% would totally cut the volume of the channel to 0...after 50% of the time of the note had elapsed).

Let's say that was 0xA0 ticks and you're playing a C#. At 0x50 ticks, that C# cuts out and you're left with silence on the channel.

This actually caused me quite a bit of headache as you can't get 100% sustain in the N-SPC and chrono trigger's engine can so when I converted CT music to LTTP music things sounded just slightly off.

The first byte is the note "length" (i.e. the number of ticks). In this case, 0x48 ticks. Notice there's no concept of quarter note / half note / whole note, etc. It's all in ticks. But you can always make up your own definitions. Typically I see 0x60 as being like a whole note, iirc anyways.

What I said earlier about sustain still applied, but to the second byte provided there is a second byte, the first byte and the second byte are optional. You can just play a note without setting a note lenght or sustain and it will use the most recently set note length or sustain for that channel. This saves space in the tracker data.

So if you're repeating a bunch of notes that are all the same length, you can just specify the pitch (of each in succession). Anyways, the second byte, if present goes as follows: it's 7F.

Another code snippet reveals the following:

```
0B9F: 3F 5C 0C  CALL  get_track_byte       ; grab the next byte.
0BA2: 30 18     BMI   $0BBC                ; if(track_byte < 0)
0BA4: 2D        PUSH  A
0BA5: 9F        XCN   A
0BA6: 28 07     AND   A,#$07
0BA8: FD        MOV   Y,A          ; Y = (XCN(A) & 0x07)
0BA9: F6 96 3D  MOV   A,$3D96+Y    ; A = one of {50,101,127,152,178,203,229,252}
0BAC: D5 01 02  MOV   $0201+X,A    ; $0201+X = that result. what is this!!!?
0BAF: AE        POP   A                    ;
0BB0: 28 0F     AND   A,#$0F               ;
0BB2: FD        MOV   Y,A          ;
0BB3: F6 9E 3D  MOV   A,$3D9E+Y    ; A = one of {25, 50, 76, 101, 114, 127, 140, 152, 165, 178, 191,
203, 216, 229, 242, 252}
0BB6: D5 10 02  MOV   $0210+X,A    ; $0210+X is a note level volume modulator
0BB9: 3F 5C 0C  CALL  get_track_byte       ; retrieves a byte from the track data.
```

Of course this probably looks like gibberish. We take 7F, do an XCN on it (exchange nybbles) which forms F7, then AND with 0x07. Then use it as an index by storing it to the Y register. So in our case we'll be using $3D96 + 7, which is $3D9D. 252 (decimal).

So what the hell is this? (per my comment in the file)

```
; $0200+X = note sustain time (when a voice is keyed on, this is how long it takes to reach release
state)
```

My documentation doesn't always match in all places ;)

So you've got 8 levels of sustain, indicated by the top nybble of the second byte. The bottom nybble provides 16 levels of volume control on the note. Meaning, this one particular note can sound louder, or softer than the channel volume

Q: *Would it be possible to change the SPC driver mid game?*

A: Yeah, you could. It could get hairy though. That's a pretty advanced topic, to be honest! Your best bet would be to alter the N-SPC so that it can go back to the bootstrap ROM... which is a 0x40 byte area in the SPC's RAM that contains the bootloader that handles the first SPC transfer before your custom driver starts working.

CT's driver can go back to the boostram ROM, N-SPC can't, that I'm aware of and I spent a sizeable amount of time studying N-SPC.

Anyways, those two bytes are *completely* optional. Next byte is : A7.

Now, after 3 hours of discussion, we arrive a freaking MUSICAL NOTE! HOLY SHIT... okay. So far we've covered bytes 0x00 through 0x7F, and 0xE0 through 0xFF. For the most part anyways.

0x00 - 0x7F = note modifications
0xE0 - 0xFF = commands

(though not all commands would be available in all drivers, and some in *NO* drivers)

0x80 to 0xC7 are musical notes. Each byte corresponds to a half step.

If 0x80 is A, 0x81 is A#, 0x8C is also A... but an octave up.

Q: *So the notes of the chromatic scale, based on whatever pitch you set the sample instrument at and stuff?*
Chromatic scale = all the notes (A A# B and so on until it loops to A again ect)

A: Well... pretty much. There's 84 keys on a piano. that gives you 6 octaves, if I'm counting right... but global transpose can get you higher or lower than all that. *shrug*, it's a hairy situation because if you set the pitch registers too high, you underflow and your instrument is so low frequency that you can't *hear* it at all (your ears just can't hear it). Matrixz has a detailed document on it somewhere on the web.

Importing music from CT this was one of my biggest problems. The instruments were tuned differently on those two platforms, so I'd take a sample from CT and not know how to tune it in N-SPC properly. My instrument would be offkey, or too high, or too low or inaudible. Quite a clusterfuck and the math formulas can suck. I want to make at some point, a tuning tool for
N-SPC.

Anyways, 0xC8 - a rest (e.g. 70 C8). A rest that lasts 0x70 ticks. 0xC9 – sustain (e.g. 70 A7 C9) that amounts to 0xE0 ticks of whatever note A7 is.
Finally 0xCA and above, up to 0xDF (inclusive)... these are percussion. As I said earlier, percussion doesn't necessarily have to involve drums. The idea behind the percussion system is to not only play a note, but also change to a different sample, with just one byte (e.g. if you wanted to hit a hi-hat drum sound, then a bass drum immediately after that you'd use the percussion system).

Earlier, we saw FA 27 was the first command... this means that instruments 27 and up are considered part of the percussion system. An example in a moment:

```
0906: 3F 66 0C  CALL  $0C66              ; sets the instrument for this voice using an alternate
method
                                                    ; doesn't seem to see much use.
                                                    ; you can use 0xCA through 0xDF to set
the instrument index to
                                                    ; 0x00 through 0x15, respectively.
                                                    ; later games use this feature for
percussion (drum) sets to switch
                                                    ; between different drum notes and ADSR
values quickly.
        0909: 8D A4     MOV   Y,#$A4             ; This is middle C, so a drum instrument always
plays as middle C
                                                    ; (this of course doesn't take into
account the tuning byte in the instrument table.)
```

^ from the Zelda disassembly.

Anyways, if you know any drummers you know they tune their drums before a performance to a certain note. So the instrument table could configure two bass drums, but one at D and another at G flat or whatever but you could switch between those two drums with a minimum of hassle using the percussion system.

Q: *Do you have any experience importing samples from outside what's already in a ROM somehwere? I mean, like if I was going to record a keyboard, would it sound better to record just a C note, or a C chord?*

A: That's what I did for my CT imports. I imported them from the CT spc set directly to a target ROM.

Q: *How does it know when to loop back to a certain point in the track?*

A: That is either handled at the tracker level or the part level

0F08: dw $0D9F ; command EF(3) (call loop)

I'm a little fuzzy on how this works, having messed with CT much more recently than Zelda,
but I think a call loop command with the last parameter as zero is equivalent to a "jump to this place" command normally (e.g. EF 00 63 03). That would jump to location 0x6300 in the SPC RAM
and would do 3 iterations of that loop (the end of the loop is a 0x00 byte).

0x00 terminates a part, by the way unless you're in a call loop it's the exception to the 0x00 to 0x7F rule being note configurators and when I say it terminates a part... it terminates the whole part.

e.g. Let's say we've got 8 tracks. The last 7 channels have been configured to run for 1 minute,
but channel 0 encounters a 0x00 byte (as a command) only 30 seconds into its playback. The whole part is over then... this is the thing I hate most about N-SPC. On the one hand it provides easy synchronization which CT lacks, but it is also hard to tell when track data ends.

Just looking at it in hex. Once a part is over, it continues to the next part... unless you specify a number of times to play that part. I think something like 20 53 02 00, then after that 20 54 FF 00
would play the track data at 0x5320 two times (as in, loop twice, being played 3 times all in all).

Then it would play the track data at 0x5420 indefinitely as 0x00FF means to loop indefinitely.
I don't remember the details on the parts very well. There's a couple of wacky special cases in there (special values that mean different things)

It would probably sound better to record a single C note.

Btw, found this in my SPC notes for Super Metroid:

```
1b96: dw $1af1          ; 0xFA fast instrument? (drum beat according to bh89)
1b98: dw $1af4          ; 0xFB (command does not exist in NSPC-Zelda but this command actually
does nothing at all.)
1b9a: dw $1af8          ; 0xFC channel sustain
1b9c: dw $1afd          ; 0xFD global sustain on
1b9e: dw $1afe          ; 0xFE global sustain off
```

Those are the commands that weren't present in LTTP (except for FA).

Think about it, if you record a C chord, like a C major chord.

C E G

You only have that one chord, you can play it on one channel but it takes up a shitload of space for just one type of chord... you won't have C minor.

C Eflat G

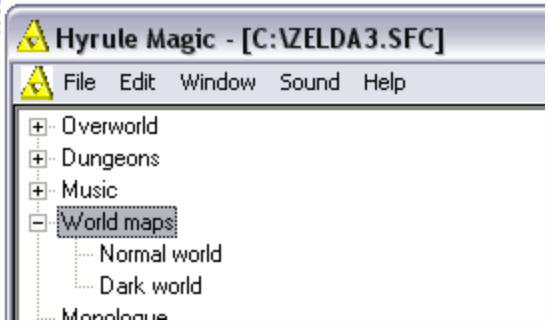
You won't have any other chords... it would be pretty pointless. All you'd have would be at transposable major chord. Of course, if you want a 3 or 4 note chord the other way you need to utilize 3 or 4 channels to play the chord.

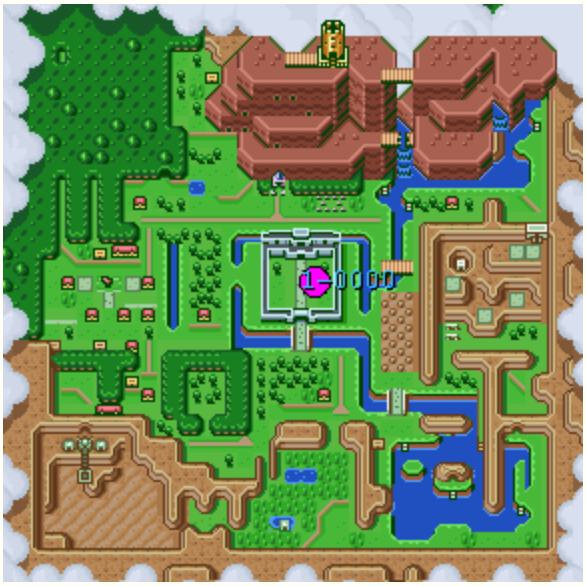
There's always tradeoffs, but recording a chord as a sample is pretty bassackwards
especially with a limited RAM environment like the SPC-700. Typically the digital processing gets bad when you go above or below two octaves of the sampled frequency
which in most cases will be fine.

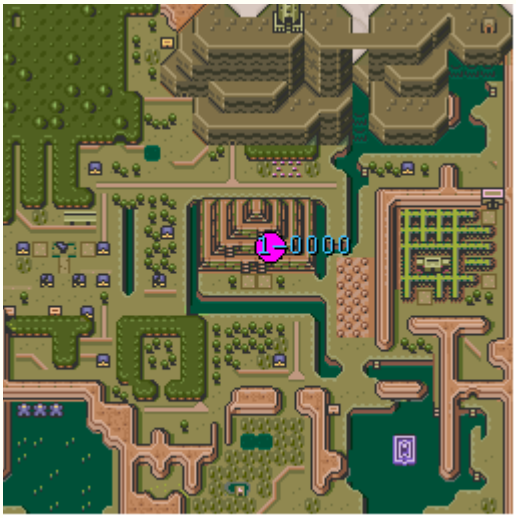
If you've got to have a very low piano part and also a very high piano part, it's usually advisable to have two separate recordings of a piano key... one at a low pitch, and another at a high pitch
and just call them different instruments.

10) World Maps

by Sephiroth3, Orochimaru and sorlokreaves







The world maps contain the maps which you can access in game while in the overworld by pressing the x button (On a real SNES). Editing the tiles sure is easy, just make sure that the **draw button** is on. Use right clicks to select to copy the parts you like and left click to paste that tile where you want. Of course you can also choose them from the right rectangle.

You can also draw by using the select option (Default), use it to select an area, and drag it to wherever you want that area to go. Also note that Dark World map uses the same surrounding clouds as the Light World ones.

Drawing is easy, but what happens if you have some really nice gfx you want to put on the map?

Double clicking on the tiles on the right will lead you to the graphics editor window. This section contains a bunch of colours on the right and the tiles on the left (As shown above). The copy and paste buttons there can be used to import/export the World Map GFX to an external image editor. ***Remember, to copy them in an image editor like Jasc Paint Shop Pro or Adobe Photoshop, because MSPaint will corrupt the colors, thus you will be losing quality in your graphics and won't be able to paste them back in the editor...*** You will need to be careful while choosing the colors; not to use colours which the SNES doesn't use (In other words, stick to the colors you already have there). The bunch of colors on the right is the Overworld Palettes (Not sure about this...).

Left click on a color in the palette editor to edit it.
Right click on it to toggle priority (for the sprites only)

The pink circles are the event signals which you will see on the map. (e.g. Remember at the start of the game you have to head to Hyrule Castle and there's this X thing on the castle? That's it.) To move it, just select the **move button** at the top of the screen and move it to wherever you want it to be. You can also remove it by right clicking. (You can add it back by right clicking and selecting **insert icon #)**. You can select which event you want to edit by that combo box up there which should say "Hyrule Castle" (As shown in the above screen). These events should be pretty Self-Explanatory.

Note: You should try to insert icons which the game doesn't use, for example, trying to put 2 circles at the start; However, trying to put 5 circles on the map which shows the Master Sword and the 3 pendants will result in glitched GFX and will look bad. You should avoid that.

Above: This is what you can do to the flute locations, but remember changing these only affect where they are shown on the world map, not the actual dropping spots.

How to change the flute locations

by sorlokreaves

Changing the 8 flute locations (we ignore the special 9th location) is quite easy to do, but it requires two rather unrelated steps:

1. Change the location of the flashing number on the world map.
2. Change the location the bird drops you off at on the proper Overworld screen.

Neither step is difficult. First, open Hyrule Magic and click on “World Maps” and then “Normal World”. Change the selected marker from “Hyrule Castle” to “Flute Locations” —the very last item in the drop-down list.

Picture: *Eight of the Nine Flute Locations, As Seen On The In-Game Map.*

Personally, I've always found location 8 to be a bit lame. Let's move it to a more useful place: the Great Fairy's island. To do this, click on the "Move" button in the top toolbar, then simply drag the pink 8 to the center of the lake.

Picture: *Drag the Flute Location to the Island.*

Save, and close the world map editor. If you opened the saved game now, you'd see the flashing number 8 in its new location, but the bird would still bring you to the old location. To fix this, open "Overworld" area 3F. Click on the "Transport" button to show the whirlpools and bird locations, and then right-click on "Fly-8" and choose "Remove".

Picture: *Hyrule Magic Will Complain If You Don't Remove the Old Flute Point First.*

Now, move to area 35 and right-click on the island, then choose "Insert bird location". Congratulations! You've moved the drop-off point.

Picture: *Adding a New Flute Location is Easy Once You Know Where to Look For It.*

Save your map. Before testing it out, you should probably short-circuit a few of the doors in the castle, to make evacuating Zelda easier. Also, put the Flute in the treasure chest in Link's house. This way, you can start a new game, rescue Zelda, and activate the bird in Kakariko Village without wasting too much time. The end result is exactly what we expected, except…

The Map Is Properly Up-To-Date.

Link Also Zooms In To the New Location Correctly.

Our rock-bush tile is still in effect from the pre-lunch hack. This has some unintended consequences:

Did Anyone Else Notice That Those Ice Men Were Super-Damaging?

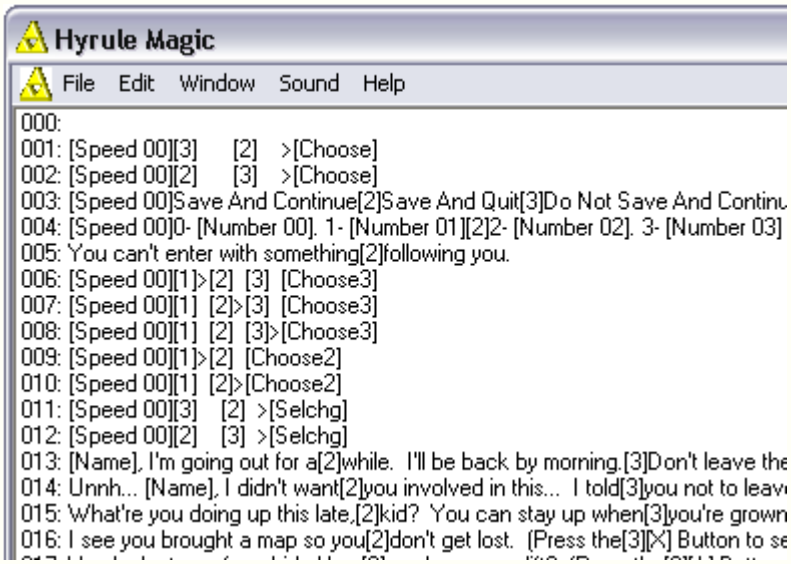
But that's nothing you can't handle, right?

There's one more function in world map editing which is very dangerous to use, it has the potential to forever stuff up your game and you can't do anything about it unless you restore from your backups. It's the **layout button**. You'll see lots of rectangle squares in the map with hex numbers in them. They represent the Overworld and how they're joined up together. (e.g. Clicking on the big Area 00 square in the forest will split the screen in four. Clicking on one of the small screens will make four of them merge into one big screen. It's dangerous because if something like Items, Sprite etc.. is right in the middle of that big Area 00 square, it will cause Hyrule Magic to crash and something really bad will be written to the rom instead.

Note: Sometimes Hyrule Magic doesn't crash in that situation, but it has a high chance that something in your rom is stuffed up. A good tip would be to clean up the ROM (Removing all the Overworld Sprites & Items) before attempting this operation.

11) Monologue

by Sephiroth3, unknown and PuzzleDude



This editor edits the monologue in the game. Double click a sentence to edit it in the field at the bottom. Press Set after the text have been modified. Edit text modifies the dialogue, while Edit dictionary modifies the verbs, nouns, adjectives, yes, no, and many more...

a) Special commands

[NextPic] Displays the next picture in the intro.
[Choose] Choose between lines 2 and 3.
[Choose2] Choose between lines 1 and 2.
[Choose3] Choose among all the lines.
[Item] Choose between items.
[Name] Displays the player's name.
[Window ##] 0=Show frame 2=Hide frame.
[Number ##] Displays a number.
[Position ##] 0=Top 1=Bottom.
[ScrollSpd ##] Sets the scrolling speed.
[Selchg] Changes the selection.
[Crash] Crashes the game.
[Scroll] Scroll one line up.
[1] Output at line 1.
[2] Output at line 2.
[3] Output at line 3.
[Color ##] Sets the color.
- **00** Orange, Grey.
- **01** Red, White.
- **02** Gold, White (Same as border colors).
- **03** Light Blue, White.
- **04** Dark Grey, Grey.
- **05** Red, ?Tan?.
- **06** Dark Blue, White (Default).
- **07** Lime, White.
- **08** Orange, Grey (Same as **00**, and the rest repeats too).

[Wait ##] Waits for some time.
[Sound ##] Plays a sound.
[Speed ##] Sets the text speed (0=Instantaneous).
[Mark1] Nothing of interest.
[Mark2] Nothing of interest.
[Clear] Clears the box.
[Wait] Waits for a keypress.

Misc:
[A] Letters for gamepad buttons.
[B] Same as above.
[Y] Same as above.
[X] Same as above.

The space for the dictionary is limited. If you try to enter too many characters, you will not succeed.

Note: The ending message **(371)** is not complete as well as the rest. You will have to work with an hex editor for completing all that. It's not too hard if you know how to do a relative search.

It only contains **80%** of the text in the game. The rest of the text are the credits, there is no dictionary in the ending text. So refer the "**Ending sequence hex editing by PuzzleDude**" for what you want to change (through an HEX editor).

b) Monologue locations

by unknown

000 > NA - None.
001 > NA - None.
002 > NA - None.
003 > NA - None.
004 > NA - None.
005 > Enter with something behind you.
006 > NA - None.
007 > NA - None.
008 > NA - None.
009 > NA - None.
010 > NA - None.
011 > NA - None.
012 > NA - None.
013 > First message of your uncle.
014 > When you get your uncle's sword.
015 > Palace Guard Message 1.
016 > Palace Guard Message 2.
017 > Palace Guard Message 3.
018 > Palace Guard Message 4.
019 > Palace Guard Message 5.
020 > Palace Guard Message 6.
021 > Palace Guard Message 7 (Rotates back to Message 1 after that one).
022 > Priest's Message before visiting Sahasrahla.
023 > Priest's Message 1 when you meet him with Zelda.
024 > Priest's Message 2 when you meet him with Zelda.
025 > Priest's Message after visiting the Sahasrahla, but before getting the 3 pendants.
026 > Priest's Message after getting the 3 pendants, but before getting Master Sword.
027 > Priest's Message after getting the Master Sword.
028 > Zelda's Message 1 when you meet her in the dungeon.
029 > Zelda's Message when you meet with the Priest.
030 > Zelda's Message before visiting Sahasrahla.
031 > First in-game introduction message.
032 > Message which occasionally appear before finding uncle.
033 > Zelda's Message when you enter the 1st floor in the Hyrule castle.
034 > Zelda's Message when you approach the ornamental shelf.
035 > Zelda's Message when you approach the right switch.
036 > Zelda's Message 3 when you meet her in the dungeon.
037 > Zelda's Message 2 when you meet her in the dungeon.
038 > Zelda's Message after meeting Sahasrahla, but before getting the 3 pendants.
039 > Zelda's Message after getting the 3 pendants, but before getting Master Sword.
040 > Zelda's Message after you got the Master Sword.
041 > Zelda's Message when you enter the water ways under the castle dungeons.
042 > Zelda's Message when you enter the room with 2 levers.

043 > The Old Women in 'Sahasrahla House' in Kakariko Village when you first talk Message 1

Who? Oh, it's you, **[Name]**![2]What can I do for you, young[3]man? The elder? Oh, no one**[Waitkey][Scroll]**has seen him since the wizard**[Scroll]**began collecting victims...**[Scroll]**... ... ...**[Waitkey][Scroll]**What? Master Sword? Well, I**[Scroll]**don't remember the details**[Scroll]**exactly, but...

044 > The Old Women in 'Sahasrahla House' in Kakariko Village when you first talk Message 2

Long ago, a prosperous people**[2]**known as the Hylia inhabited**[3]**this land...**[Waitkey][Scroll]**Legends tell of many treasures**[Scroll]**that the Hylia hid throughout**[Scroll]**the land...**[Waitkey][Scroll]**The Master Sword, a mighty**[Scroll]**blade forged against those**[Scroll]**with evil hearts, is one of**[Waitkey][Scroll]**them. People say that now it**[Scroll]**is sleeping deep in the forest...**[Scroll]**... ... ...**[Waitkey][Scroll]**Do you understand the legend?**[Scroll]** > Yes**[Scroll]** Not at all**[Choose]**

045 > The Old Women in 'Sahasrahla House' in Kakariko Village when you first talk Message 3

Anyway, look for the elder.**[2]**There must be someone in the**[3]**village who knows where he is.**[Waitkey][Scroll]**You take care now, **[Name]**...

046 > The Old Women in 'Sahasrahla House' in Kakariko Village after talking to Sahasrahla

Ohhh,**[Name]**. You've changed!**[2]**You look marvelous... Please**[3]**save us from Agahnim**[Waitkey][Scroll]**the wizard!

047 > Scared Girl1 Message.

Hey! Here is **[Name]**, the**[2]**wanted man! Soldiers! Anyone!**[3]**Come quickly!

048 > Sahasrahla Message after getting 3 pendants

Incredible! At last you have**[2]**the three Pendants... Now, you**[3]**should go to the Lost Woods.**[Waitkey][Scroll]**If you are the true Hero, the**[Scroll]**sword itself will select you.**[Scroll]**... ... ...

049 > Sahasrahla Message after getting master sword

I am too old to fight.**[2]**I can only rely on you.

050 > Sahasrahla Message 1 when you first meet him

I am, indeed, Sahasrahla, the**[2]**village elder and a descendent**[3]**of the seven wise men.**[Waitkey][Scroll]**... ... Oh really?**[Scroll][Name]**, I am surprised a young**[Scroll]**man like you is searching for**[Waitkey][Scroll]**the sword of evil's bane.**[Scroll]**Not just anyone can use that**[Scroll]**weapon. Legends say only the**[Waitkey][Scroll]**Hero who has won the three**[Scroll]**Pendants can wield the sword.**[Scroll]**... ... ...**[Waitkey][Scroll]**Do you really want to find it?**[Scroll]** > Yeah!**[Scroll]** Of Course!**[Choose]**

051 > Sahasrahla message 2 when you first meet him

Good. As a test, can you[2]retrieve the Pendant Of[3]Courage from the East Palace?[Waitkey][Scroll]If you bring it here, I will tell[Scroll]you more of the legend and[Scroll]give you a magical artifact.[Sound 2D][Waitkey][Scroll]Now, go forward to the palace.

052 > Sahasrahla hint message

Other relatives of the wise men[2]are hiding from the evil[3]wizard's followers.[Waitkey][Scroll]You should find them.

053 > Sahasrahla Telepathic Message 1 when you first enter the Dark World

[Window 02][Speed 03][Name], it is I, Sahasrahla.[2]I am communicating to you[3]across the void through[Waitkey][Scroll]telepathy... The place where[Scroll]you now stand was the Golden[Scroll]Land, but evil power turned it[Waitkey][Scroll]into the Dark World. The[Scroll]wizard has broken the wise[Scroll]men's seal and opened a gate[Waitkey][Scroll]to link the worlds[Scroll]at Hyrule Castle. In[Scroll]order to save this half of the[Waitkey][Scroll]world, the Light World, you[Scroll]must win back the Golden[Scroll]Power. You must also rescue[Waitkey][Scroll]the seven maidens who Agahnim[Scroll]sent to the Dark World. As[Scroll]members of the blood-line of[Waitkey][Scroll]the seven wise men, they have[Scroll]power that will surely help you.[Scroll]The maidens are locked in[Waitkey][Scroll]hidden dungeons full of evil[Scroll]creatures and dangerous traps.[Scroll]The Palace of Darkness should[Waitkey][Scroll]be your first goal in this world![Sound 2D][Scroll][Name], I can rely on only[Scroll]you. Please make this old[Waitkey][Scroll]man's wishes come true.[Scroll]I beg you!

054 > Sahasrahla Telepathic Message 2 when you first enter the Dark World (without the Moon Pearl)

[Window 02][Speed 03]But I sense your helplessness...[2]Before you go any further,[3]find the Moon Pearl on[Waitkey][Scroll]Death Mountain. It will protect[Scroll]you from the Dark World's[Scroll]magic so you can keep your[Waitkey][Scroll]heroic figure.

055 > Sahasrahla hint message
A helpful item is hidden in the[2]cave on the east side of Lake[3]Hylia. Get it!

056 > Sahasrahla Message when you return with the Pendant of Courage. (message after having meet him already)
Oh!? You got the Pendant Of[2]Courage! Now I will tell you[3]more of the legend...[Waitkey][Scroll]Three or four generations ago,[Scroll]an order of knights protected[Scroll]the royalty of the Hylia.[Waitkey][Scroll]These Knights Of Hyrule were[Scroll]also guardians of the Pendant[Scroll]Of Courage.[Waitkey][Scroll]Unfortunately, most of them[Scroll]were destroyed in the great[Scroll]war against evil that took[Waitkey][Scroll]place when the seven wise men[Scroll]created their seal. Among the[Scroll]descendants of the Knights Of[Waitkey][Scroll]Hyrule a hero must appear.[Scroll]...I see. [Name], I believe you.[Scroll]You should get the remaining[Waitkey][Scroll]Pendants.[Scroll]And carry this with you.[Scroll]This is a treasure passed down[Waitkey][Scroll]by the families of the wise[Scroll]men. I want you to have it.

057 > Sahasrahla Message when you return with the Pendant of Courage. (message if you haven't meet him already)
You are correct, young man![2]I am Sahasrahla, the village[3]elder and a descendent of the[Waitkey][Scroll]seven wise men.[Scroll]... ... ... What?[Scroll][Name], I'm surprised a young[Waitkey][Scroll]man like you is looking for the[Scroll]sword of evil's bane. Not just[Scroll]anyone can use that sword.[Waitkey][Scroll]According to the tales handed[Scroll]down by the Hylia, only the[Scroll]Hero who has destroyed three[Waitkey][Scroll]great evils and won the three[Scroll]Pendants can wield the sword...[Scroll]... ... ...[Waitkey][Scroll]I see you have acquired the[Scroll]Pendant of Courage. I will tell[Scroll]you about the legend behind it.[Waitkey][Scroll]Three or four generations ago,[Scroll]an order of knights protected[Scroll]the royalty of the Hylia.[Waitkey][Scroll]These Knights Of Hyrule were[Scroll]also guardians of the Pendant[Scroll]Of Courage.[Waitkey][Scroll]Unfortunately, most of them[Scroll]were destroyed in the great[Scroll]war against evil that took[Waitkey][Scroll]place when the seven wise men[Scroll]created their seal. Among the[Scroll]descendants of the Knights Of[Waitkey][Scroll]Hyrule a hero must appear.[Scroll]...I see. [Name], I believe you.[Scroll]You should get the remaining[Waitkey][Scroll]Pendants.[Scroll]And carry this with you.[Scroll]This is a treasure passed down[Waitkey][Scroll]by the families of the wise[Scroll]men. I want you to have it.

058 > Overworld Sign text - on specific areas before meeting with uncle.
059 > Overworld Sign text #1.
060 > Overworld Sign text #2.
061 > Overworld Sign text #3.
062 > Overworld Sign text #4.
063 > Overworld Sign text #5.
064 > Overworld Sign text #6.
065 > Overworld Sign text #7.
066 > Overworld Sign text #8.
067 > Overworld Sign text #9.
068 > Overworld Sign text #10.
069 > Overworld Sign text #11.
070 > Overworld Sign text #12.
071 > Overworld Sign text #13 (GuyByTheSign).
072 > Overworld Sign text #14.
073 > Overworld Sign text #15.
074 > The Witch Message when you don't have the mushroom.
075 > The Witch Message after given the mushroom.
076 > The Witch Message when you have the mushroom.
077 > The Potion Shop guy Message when you don't have any bottles.
078 > The Potion Shop guy Message.
079 > When you try to buy a potion without any bottles.
080 > When you try to buy a potion but got no free bottles.
081 > Receive Lamp Message.

082 > Receive Boomerang Message.
083 > Receive Bow Message.
084 > Receive Shovel Message.
085 > Receive Magic Cape Message.
086 > Receive Magic Powder Message.
087 > Receive Zora's Flippers Message.
088 > Receive Power Glove Message.
089 > Receive Pendant of Courage Message.
090 > Receive Pendant of Power Message.
091 > Receive Pendant of Wisdom Message.
092 > Receive Mushroom Message.
093 > Receive Book of Mudora Message.
094 > Receive Moon Pearl Message.
095 > Receive Compass Message.
096 > Receive Map Message.
097 > Receive Ice Rod message.
098 > Receive Fire Rod Message.
099 > Receive Ether Medallion Message.
100 > Receive Bombos Medallion Message.
101 > Receive Quake Medallion Message.
102 > Receive Magic Hammer Message.
103 > Receive Flute Message.
104 > Receive Cane of Somaria Message.
105 > Receive Hook Shot Message.
106 > Receive Bombs Message.
107 > Receive Magic Bottle Message.
108 > Receive Big Key Message.
109 > Receive Titan's Mitt message.
110 > Receive Magic Mirror Message.
111 > When you pick up a fake Master Sword (Lost Woods).
112 > Sahasrahla message when you pick up the real Master Sword.
113 > Receive Red Potion Message.
114 > Receive Green Potion Message.
115 > Receive Blue potion message.
116 > Receive Bug-Catching Net Message.
117 > Receive Blue Mail (Armor2) Message.
118 > Receive Red Mail (Armor3) Message.
119 > Receive Sword3 Message.
120 > Receive Shield3 Message.
121 > Receive Cane of Byrna Message.
122 > When trying to open the big chest without the Big Key.
123 > When you run out of magic.

124 > Sahasrahla gives you the Pegasus Shoes Message
He gives you the Pegasus[2]Shoes! Now you can execute a[3]devastating dash attack![Waitkey][Scroll](Hold the [A] Button[Scroll]for a short time.)

125 > LiveTree - Talking Tree Message 1
Wow! I haven't seen a normal[2]person in a few hundred years![3]Let me talk to you for a while.

126 > LiveTree - Talking Tree Message 2

Do you know about the[2]Gargoyle statue in the village?[3]People say they can hear a girl[Waitkey][Scroll]calling for help from under the[Scroll]statue. Isn't that a strange[Scroll]story...

127 > LiveTree - Talking Tree Message 3
Surprisingly, the Triforce[2]created this world to fulfill[3]Ganon's wish.[Waitkey][Scroll]What is Ganon's wish,[Scroll]you ask? It is to rule the[Scroll]entire cosmos! Don't you think[Waitkey][Scroll]it might be possible with the[Scroll]power of the Triforce[Scroll]behind you?

128 > LiveTree - Talking Tree Message 4
I once lived in the Lost Woods,[2]until the day I wandered into a[3]magic transporter...[Waitkey][Scroll]The power of the Dark World[Scroll]quickly turned me into this[Scroll]tree shape...[Waitkey][Scroll]I guess the two forests are[Scroll]connected with each other...

129 > LiveTree - Talking Tree Message 5
I heard that using Bombs is the[2]best way to defeat the[3]one-eyed giants.[Waitkey][Scroll]That's all I know!

130 > LiveTree - Talking Tree Message 6
Quit bothering me! And watch[2]where you're going when you[3]dash around!

131 > Receive Pendant of Power Message. Version 2.
132 > Receive Pendant of Wisdom Message. Version 2.
133 > Shooting gallery Message.
134 > Shooting gallery Message when you decided to play.
135 > Thieves Cavern Message.
136 > Shooting gallery Message asking if you want to play again.
137 > Pond of Happiness Introduction Message.
138 > Pond of Happiness select item to throw in Message.
139 > Pond of Happiness did you drop this? Message.
140 > Pond of Happiness saying you're honest Message.
141 > Pond of Happiness when you answer **NO** to question **(139)** Message.
142 > Pond of Happiness telling that you are lying Message.
143 > Receive Magical Boomerang Message.
144 > Receive Shield2 Message.
145 > Receive Silver Arrows Message.
146 > Receive Green potion message from Pond of Happiness.
147 > Receive Sword4 Message.
148 > Pond of Happiness saying you donated that much cash Message.
149 > Pond of Happiness asking you arrows or bombs? Message.
150 > Pond of Happiness receive more bombs Message.
151 > Pond of Happiness receive more arrows Message.
152 > Pond of Happiness when you reach max bombs/arrows Message.
153 > Old Man Message when travelling with him Message 1.
154 > Old Man Message when travelling with him Message 2.
155 > Old Man Message when travelling with him Message 3.
156 > Old Man Message - Intro Message.
157 > Old Man Message - End Message.
158 > Old Man cave Message 1.
159 > Old Man cave Message 2.
160 > Old Man cave Message 3.
161 > Flute boy father in the bar... sleeping Message.
162 > Flute boy father in the bar... woke up Message 1.
163 > Flute boy father in the bar... woke up Message 2.

164 > Sleeping guy under the bridge Message.
165 > Old Woman in Kakariko Village Message.
166 > That guy which runs away from you in Kakariko Village Message.
167 > Kakarikko village maze (message when you read a sign when the clock is started)
168 > You need the cape to get this Piece of Heart (Mountain).
169 > Overworld Sign text #15.
170 > Overworld Sign text #16.
171 > Overworld Sign text #17.
172 > Overworld Sign text #18.
173 > Overworld Sign text #19.
174 > Overworld Sign text #20.
175 > Overworld Sign text #21.
176 > Overworld Sign text #22.
177 > Overworld Sign text #23.
178 > Soldier Outside Hyrule Castle.
179 > Guard in front of Hyrule Castle.
180 > Telepathic Message #1.
181 > Telepathic Message #2.
182 > Encoded Message text.
183 > Inscription in front of Master Sword text.
184 > Telepathic Message #3.
185 > Telepathic Message #4.
186 > Telepathic Message #5.
187 > Telepathic Message #6
188 > Encoded Message text (you know, those weird symbols).
189 > Inscription in the desert.
190 > Telepathic Message #7.
191 > Telepathic Message #8.
192 > Telepathic Message #9.
193 > Telepathic Message #10.
194 > Telepathic Message #11.
195 > Telepathic Message #12
196 > Telepathic Message #13
197 > Telepathic Message #14
198 > Telepathic Message #15
199 > Chris Houlihan's Room Message.
200 > Catch and caught Bee Message.
201 > Catch and caught Fairy Message.
202 > Catch and caught Message when you don't have any bottles.
203 > Time Report.
204 > Time Report starter.
205 > Time Report - Success.
206 > Time Report - Fail.
207 > Time Report - Success after taking heart.
208 > Time Report - Can't start timer message.
209 > Bottle Shop - Buy or not message?
210 > Bottle Shop - Buy message.
211 > Bottle Shop - Not enough rupees.
212 > Bottle Shop - Outta bottles.
213 > Bottle Shop - Sell Bee for 100 Rupees Message.

214 > Bottle Shop - Fish Exchange Message.
215 > Sleeping guy under the bridge Message 2. (gives you a bottle)

216 > Blacksmith - Temper or not?
Hey you![2]Welcome![3][Waitkey][Scroll]Ask us to do anything![Scroll] > Temper my sword[Scroll] I just dropped by[Choose]

217 > Blacksmith - Temper sword... 10 bucks?
I'll give you a big discount![2] >Sword Tempered... 10 Rupees[3] Wait a minute[Choose]

218 > Blacksmith - Temper sword, are you sure?
Tempered, eh? Are you sure?[2] > Yes[3] I changed my mind[Choose]

219 > Blacksmith - Can't temper anymore
Well, we can't make it any[2]stronger than that... Sorry!

220 > Blacksmith - Ask again when your sword is tempering)
Drop by again anytime you[2]want to. Hi ho! Hi ho![3]We're off to work!

221 > Blacksmith - Agree to temper
All right, no problem.[2]We'll have to keep your sword[3]for a while.

222 > Blacksmith - Finish temper
Your sword is tempered-up![2]Now hold it!

223 > Blacksmith - Welcome note
If my lost partner returns,[2]we can temper your sword,[3]but now, I can't do[Waitkey][Scroll]anything for you.

224 > Blacksmith - Found partner
Oh! Happy days are here again![2]You found my partner![3]... We are very happy now...[Waitkey][Scroll]Drop by here again![Scroll]At that time, we will temper[Scroll]your sword perfectly!

225 > Blacksmith - That guy out there in dark world
Ribbit ribbit... Your body did[2]not change! You are not just[3]an ordinary guy, are you?[Waitkey][Scroll]I used to live in Kakariko Town.[Scroll]I wonder what my partner is[Scroll]doing there without me...[Waitkey][Scroll]Ribbit! I have a request of[Scroll]you.[Scroll]Please take me to my partner![Waitkey][Scroll]Please! Ribbit! Please!

226 > Blacksmith - Not yet finish temper
I'm sorry, we're not done yet.[2]Come back after a while.

227 > Thank You Message.
228 > ??? - Unknown.
229 > Flute Boy - Welcome note.
230 > Flute Boy - Agree to look for flute.
231 > Flute Boy - Didn't agree to look for flute.
232 > Flute Boy - Message when looking for flute, but haven't find it.
233 > Flute Boy - Ending Message.
234 > Fortune - You will find elder.
235 > Fortune - You will need book to get into desert.
236 > Fortune - Find mushroom lover.
237 > Fortune - Find a person in desert.
238 > Fortune - Meet Zora king.
239 > Fortune - Find moon pearl in tower.
240 > Fortune - Sword can't harm wizard.

241 > Fortune - Shall find the 1/2 magic thing.
242 > Fortune - Not well right now.
243 > Fortune - Intro message.
244 > Fortune - Finish message.
245 > Fortune - Didn't agree to take one fortune.
246 > Fortune - Meet strange man in desert.
247 > Fortune - Info booth in Dark World has treasure.
248 > Fortune - Place to find that smith's partner.
249 > Fortune - Find treasure in graveyard.
250 > Fortune - Buy new bombs in bomb shop.
251 > Fortune - Things in the pyramid.
252 > Fortune - Barrier in front of Ganon's castle.
253 > Fortune - Need silver arrows to kill Ganon.
254 > Info Booth (Dark World) - Welcome Message.
255 > Info Booth (Dark World) - Info Message 1.
256 > Info Booth (Dark World) - Don't agree Message.
257 > Info Booth (Dark World) - Info Message 2.
258 > Info Booth (Dark World) - Info Message 3.
259 > Info Booth (Dark World) - Info Message 4.

260 > SickBoy - When you don't have bottle
Sniffle... Hey brother [Name]![2]Do you have a bottle to keep[3]a bug in?[Waitkey][Scroll]...[Scroll]I see. You don't have one...[Scroll]Cough cough...

261 > SickBoy - When you do have bottle; he gives you the bug-catching net
I can't go out 'cause I'm sick...[2]Cough cough...[3]People say I caught this cold[Waitkey][Scroll]from the evil air that is coming[Scroll]down off the mountain...[Scroll]Sniff sniff...[Waitkey][Scroll]This is my bug-catching net.[Scroll]I'll use it when I'm better, but[Scroll]for now, I'll lend it to you.

262 > SickBoy - Message after giving you his net
Sniffle... I hope I get well[2]soon...[3]Cough cough...

263 > GuyByTheSign - ...
264 > GuyByTheSign - When you take the sign
265 > GuyByTheSign - When you have the chest
266 > GuyByTheSign - When you say no to his offer
267 > GuyByTheSign - When you talk to him again after accepting his offer
268 > GuyByTheSign - When you say yes

269 > Encoded message.
270 > Decoded message - Ether Medallion.
271 > Decoded message - Bombos Medallion.
272 > When you throw something into the Dark World pond.
273 > Getting the 1/2 magic.
274 > Game Intro message 1.
275 > Game Intro message 2.
276 > Game Intro message 3.
277 > Game Intro message 4.
278 > The Unopenable chest Message.
279 > Bomb Shop - First welcome message.
280 > Bomb Shop - new bomb welcome message.
281 > Bomb Shop - buy bombs.
282 > bomb Shop - buy big bomb.

283 > Monkey - 100 rupees to open stage 1 door.
284 > Monkey - When you disagree with his offer.
285 > Monkey - Agree with offer.
286 > Monkey - First meet Message.
287 > Monkey - Agree to give him 10.
288 > Monkey - Disagree to give him 10.
289 > Monkey - When you go somewhere instead of stage 1.
290 > Stage 4 boss - Intro Message.
291 > Stage 4 boss - Blind Message.
292 > Stage 4 boss - When you run around to specific places Message.
293 > Guy in the cave in the desert Message (Aginah)

I am Aginah. I sense something**[2]**is happening in the Golden Land**[3]**the seven wise men sealed...**[Waitkey][Scroll]**This must be an omen of the**[Scroll]**Great Cataclysm foretold by**[Scroll]**the people of Hylian blood...**[Waitkey][Scroll]**... ... ...**[Scroll]**The prophecy says, "The Hero**[Scroll]**will stand in the desert holding**[Waitkey][Scroll]**the Book Of Mudora."**[Scroll]**If you have the Book Of Mudora**[Scroll]**you can read the language of**[Waitkey][Scroll]**the Hylia People.**[Scroll]**It should be in the house of**[Scroll]**books in the village...**[Waitkey][Scroll]**You must get it!**[Scroll]**If you are the person who will**[Scroll]**be The Hero...

294 > When you collected the 3 pendants, you get this Message.
295 > ??? - Unknown.
296 > ??? - Unknown.
297 > ??? - Unknown.
298 > ??? - Unknown.
299 > ??? - Unknown.
300 > Tree cutter guys Message 1.
301 > Tree cutter guys Message 2.
302 > Tree cutter guys Message 3.
303 > Angry Brother (right).
304 > Angry Brother (right) after bombing the door.
305 > Angry Brother (left).
306 > Beaten Dark World Dungeon Message 1.
307 > Beaten Dark World Dungeon Message 2.
308 > Beaten Dark World Dungeon Message 3.
309 > Beaten Dark World Dungeon Message 4.
310 > Beaten Dark World Dungeon Message 5.
311 > Beaten Dark World Dungeon Message 6.
312 > Beaten Dark World Dungeon Message 7 (Rescues Zelda Message).
313 > Beaten Dark World Dungeon - Finish Message.
314 > Beaten Dark World Dungeon - Do you understand? Message.
315 > Break Barrier on 8th Dungeon Message.
316 > Beaten Dark World Dungeon Message 7 (without completing one or more dungeons).
317 > Aghanim's Message when you enter the room (which he makes Zelda disappear).
318 > Aghanim's Message when he gets rid of Zelda.
319 > Aghanim's Message when you enter the room to fight him.
320 > Aghanim's Dying Message.
321 > Aghanim's Message when you fight him again in last stage.
322 > Zora King - Welcome message.
323 > Zora King - When you say you want flippers.
324 > Zora King - After buying flippers.
325 > Zora King - Not enough rupees to buy flippers.
326 > Zora King - When you say nothing after the welcome message.

327 > The Village guy Message (Required Point, to mark the map with a new place to go to).
328 > The Village guy Message after the required point.
329 > ??? - Unknown.
330 > Mysterious Pond - Welcome Message.
331 > Mysterious Pond - Don't do it Message.
332 > Mysterious Pond - Don't do it Message.
333 > Mysterious Pond - Give back item Message.
334 > Mysterious Pond - How much will you toss? Message.
335 > Mysterious Pond - Message for the silver arrows.
336 > Mysterious Pond - Great Luck Message.
337 > Mysterious Pond - Good Luck Message.
338 > Mysterious Pond - A Little Luck Message.
339 > Mysterious Pond - Big Trouble Message.
340 > Mysterious Pond - Luck Message.
341 > NA - None.
342 > NA - None.
343 > NA - None.
344 > NA - None.
345 > Receive Heart container message.
346 > Fairy pond Message.
347 > Whimp (Dark World Mountain) when you are a bunny Message.
348 > Whimp (Dark World Mountain) when you are yourself Message.
349 > Bully (Dark World Mountain) when you are a bunny Message.
350 > Bully (Dark World Mountain) when you are yourself Message.
351 > Dark World shop welcome Message.
352 > Dark World massive chest game welcome Message.
353 > Dark World massive chest game don't want to play Message.
354 > Haven't paid to play Message.
355 > Can't open anymore chest Message.
356 > Pay to open chest, good luck Message.
357 > Shop Message.
358 > Shop Message, When you don't need the item.
359 > Shop Message - Buy shield.
360 > Shop Message - Buy red potion.
361 > Shop Message - Buy arrows.
362 > Shop Message - Buy bombs.
363 > Shop Message - Buy bee.
364 > Shop Message - Buy recovery heart.
365 > Shop Message - Run out of bottles.
366 > Shop Message - Can't carry more.
367 > Final Boss - Intro Message.
368 > Final Boss - Mid-Way Message.
369 > Thieves Hideout in Lost Woods 1.
370 > Thieves Hideout in Lost Woods 2.
371 > Ending Message.
372 > Message when you run into those trapped jumpy things in the grassland.
373 > The other guy in the pub.
374 > Secret Cavern Thief who gives you 300 rupees.
375 > Another Thief's Message.
376 > ??? - Unknown.

377 > Telepathic message #16.
378 > Cucumber message 1 when you throw magic powder on him. (before finding Sahasrahla)
379 > Cucumber message 2 when you throw magic powder on him. (after finding Sahasrahla)
380 > ??? - Unknown.
381 > Chicken Lady message

Cluck cluck... What?!**[2]**You turned me into a human.**[3]**I can even speak!**[Waitkey][Scroll]**Aha, it must be you who is**[Scroll]**always teasing my friends.**[Scroll]**The Weathercock is always**[Waitkey][Scroll]**watching you harass them.**[Scroll]**Well, this human shape is**[Scroll]**uncomfortable for me.**[Waitkey][Scroll]**Ahhh, I want to be a chicken**[Scroll]**again! Cluck cluck...

382 > Light World chest game Message - 20 rupees to play.
383 > Light World chest game Message - Agree to open chest.
384 > Light World chest game Message - Don't feel like opening one.
385 > Light World chest game Message - 100 rupees to play.
386 > Hedgeman Message - That guy which is in the room with lots of beds.
387 > ??? - Unknown.
388 > Starting screen choice Message with 2 options.
389 > Starting screen choice Message with 3 options.
390 > Continue/save and quit Message when you press select.
391 > Digging game guy - Intro Message.
392 > Digging game guy - Agree to dig.
393 > Digging game guy - Disagree to dig.
394 > Digging game guy - Time is up Message.
395 > Digging game guy - Come back again Message.
396 > Digging game guy - Message when the field is still dug up.

c) Ending sequence hex editing

by PuzzleDude

Part I) The correct values

Ganon entrance to the triforce shrine,
15E26 = 89
15E27 = 01

GFX in triforce shrine
15E76 = 88
15F14 = 00

Scroll up-down in triforce shrine
15FB1 = 00
15FB2 = 00

Scroll left-right in triforce shrine
1604F = 00

Example: First text is:
the return of the king (small text)
HYRULE CASTLE (BIG TEXT)

Go to WinHex32, go to relative search and type: return,
it will tell you, that the return is **2B 1E 2D 2E 2B 27**

THE RETURN... = **2D 21 1E 9F 2B 1E 2D 2E 2B 27**...

Now you have the location

Go to WinHex32, go to relative search and type: Castle
it will tell you that the castle is **5F 5D 6F 70 68 61** and **85 83 95 96 8E 87**

Now you have the locations

ALPHABET

Small text (2 values - lower row)

SMALL TEXT COLOUR1/2/3
A=1A, **A=38**, **A=00**
B=1B, **B=39**, **B=01**
C=1C, **C=3A**, **C=02**
D=1D, **D=3B**, **D=03**
E=1E, **E=3C**, **E=04**
F=1F, **F=3D**, **F=05**
G=20, **G=3E**, **G=06**
H=21 **H=3F**, **H=07**
I=22, **I=40**, **I=08**
J=23, **J=41**, **J=09**
K=24, **K=42**, **K=0A**
L=25, **L=43**, **L=0B**
M=26, **M=44**, **M=0C**
N=27, **N=45**, **N=0D**
O=28, **O=46**, **O=0E**
P=29, **P=47**, **P=0F**
Q=2A, **Q=48**, **Q=10**
R=2B, **R=49**, **R=11**
S=2C, **S=4A**, **S=12**
T=2D, **T=4B**, **T=13**
U=2E, **U=4C**, **U=14**
V=2F, **V=4D**, **V=15**
W=30, **W=4E**, **W=16**
X=31, **X=4F**, **X=17**

Y=32, **Y=50**, **Y=18**
Z=33, **Z=51**, **Z=19**
APOSTROPH=34
EMPTY=9F

Big text (2 values - upper row and lower row)
But upper row is always together and lower row is always together.

UP LOW
A=5D, **83**
B=5E, **84**
C=5F, **85**
D=60, **86**
E=61, **87**
F=62, **88**
G=63, **89**
H=64, **8A**
I=65, **8B**
J=66, **8C**
K=67, **8D**
L=68, **8E**
M=69, **8F**
N=6A, **90**
O=6B, **91**
P=6C, **92**
Q=6D, **93**
R=6E, **94**
S=6F, **95**
T=70, **96**
U=71, **97**
V=72, **98**
W=73, **99**
X=74, **9A**
Y=75, **9B**
Z=76, **9C**
EMPTY= 9F

Let's say you want to change:
the return of the king (22 letters)

to this:
the evil is vanishing (21 letters) 22 is 21+1

With relative search find the starting byte and replace the 22 old bytes with this:

2D 21 1E, 9F, 1E 2F 22 25, 9F, 22 2C, 9F, 2F 1A 27 22 2C 21 22 27 29, 9F

Let's say you want to change:
HYRULE CASTLE (13 letters)

to this:
DARK TEMPLE (11 letters) 13 is 1+11+1 (first and second value must be empty because of the centering)

With relative search find the starting byte of the First row and replace the 13 old bytes with this:
9F, 60 5D 6E 67, 9F, 70 61 69 6C 68 61, 9F

With relative search find the starting byte of the Second row and replace the 13 old bytes with this:
9F, 86 83 94 8D, 9F, 96 87 8F 92 8E 87, 9F

Part II-B) Special exits

(this is not related to door exits but the location of the frames of the ending)

01.) EXIT 1000 (place it in big area, lower middle position of the exit, picture travels up, place it so that the sprites are not seen)
02.) Room 18 (automatic)
03.) EXIT 1002 (place it in big area, upper right, travels down and left)
04.) EXIT 1012 (big area, lower middle, travels up)
05.) EXIT 1004 (big area, middle up, travels down)
06.) EXIT 1006 (small area, upper right, travels down left)
07.) EXIT 1010 (big area = Zora domain, upper left, travels right)
08.) EXIT 1014 (small area, upper middle, travels down)
09.) EXIT 100A (small area, middle right, travels left)
10.) EXIT 1016 (big area, middle low, travels up)
11.) Room 277 (must be fountain, automatic)
12.) Room 289 (must be smith, automatic)
13.) EXIT 100E (big area, upper left, travels right)
14.) EXIT 1008 (big area, upper right, travels left)
15.) EXIT 1018 (big area, upper left, travels down right)
16.) EXIT 0180 (master sword, automatic)
PS = exit of the small area can be placed in a big area to avoid the sprites

Texts:
01.) The return of the king, HYRULE CASTLE
02.) The loyal sage, SANCTUARY
03.) Sahasrahla's homecoming, KAKARIKO TOWN
04.) Vultures rule the desert, DESERT PALACE
05.) The bully makes a friend, DEATH MOUNTAIN
06.) Your uncle recovers, YOUR HOUSE
07.) Flippers for sale, ZORA'S WATERFALL

08.) The witch and assistant, MAGIC SHOP
09.) Twin lumberjacks, WOODSMEN'S HUT
10.) Flute boy plays again, HOUNTED GROVE
11.) Venus. queen of faeries, WISHING WELL
12.) The dwarven swordsmiths, SMITHERY
13.) The bug-catching kid, KAKARIKO TOWN
14.) The lost old man, DEATH MOUNTAIN
15.) The forest thief, LOST WOODS
16.) And the master sword, sleeps again..., FOREVER

Part III) Credits and dungeon names

(all editable in hex)

Some things to type in the relative search, because they are subject to change

Executive Producer
Hiroshi Yamauchi

Producer
Shigeru Miyamoto

Director
Takashi Tezuka

Script Writer
Kensuke Tanabe

Assistant Directors
Yasuhisa Yamamura
Yoichi Yamada

Screen Graphics Designers
Object Designers
Soichiro Tomita
Takaya Imamura

Back Ground Designers
Masanao Arimoto
Tsuyoshi Watanabe

Program Director
Toshihiko Nakago

Main Programmer
Yasunari Soejima

Object Programmer
Kazuaki Morita

Programmers
Tatsuo Nishiyama
Yuichi Yamamoto
Yoshihiro Nomoto
Eiji Noto
Satoru Takahata
Toshio Iwawaki
Shigehiro Kasamatsu
Yasunari Nishida

Sound Composer
Koji Kondo

Coordinators
Keizo Kato
Takao Shimizu

Printed Art Work
Yoichi Kotabe
Hideki Fujii
Yoshiaki Koizumi
Yasuhiro Sakai
Tomoaki Kuroume

English Script Writers
Daniel Owsen
Hiroyuki Yamada

Special Thanks To
Nobuo Okajima
Yasunori Taketani
Kiyoshi Koda
Takamitsu Kuzuhara
Hironobu Kakui
Shigeki Yamashiro

Quest History
Location Games

Castle of Hyrule
Castle Dungeon
East Palace
Desert Palace
Mountain Tower

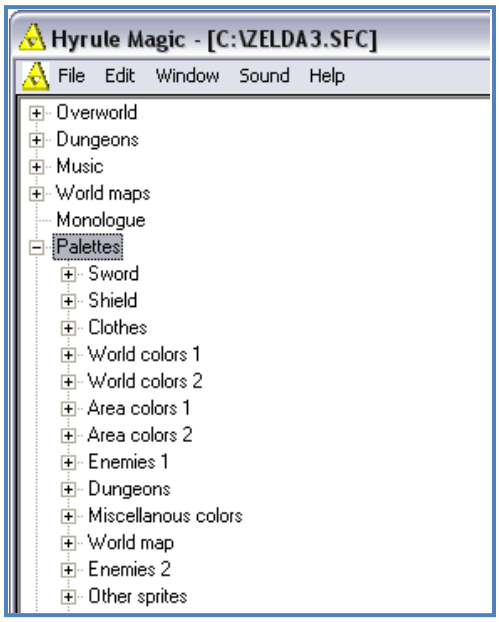
Level 1 DARK PALACE
Level 2 SWAMP PALACE
Level 3 SKULL WOODS
Level 4 THIEVES' TOWN
Level 5 ICE PALACE
Level 6 MISERY MIRE

Level 7 TURTLE ROCK
Level 8 GANON'S TOWER

TOTAL GAMES PLAYED

12) Palettes

by Orochimaru



"This menu edits almost all the palettes in the game with a few exceptions (some are hardcoded and you need to change them in hex instead of using Hyrule Magic)."

SWORD:

Palette as seen in HM	Palette Name	Description

	Sword pal 0	Uncle's Sword (Level 1)
	Sword pal 1	Master Sword (Level 2)
	Sword pal 2	Tempered Sword (Level 3)
	Sword pal 3	Golden Sword (Level 4)

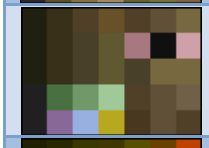
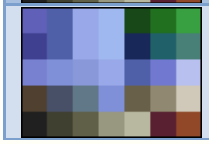
SHIELD:

Palette as seen in HM	Palette Name	Description
	Shield pal 0	Uncle's Shield (Level 1)
	Shield pal 1	Red Shield (Level 2)
	Shield pal 2	Mirror Shield (Level 3)

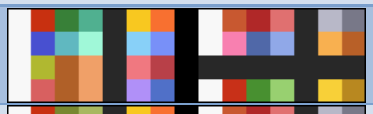
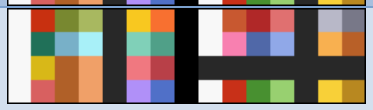
CLOTHES:

Palette as seen in HM	Palette Name	Description
	Clothes pal 0	Green Tunic (Level 1)
	Clothes pal 1	Blue Mail (Level 2)
	Clothes pal 2	Red Mail (Level 3)

WORLD COLORS 1: (General world colours)

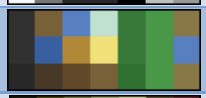
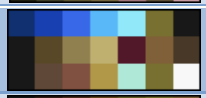
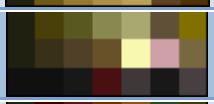
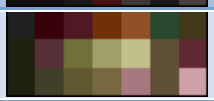
Palette as seen in HM	Palette Name	Description
	World Colors 1 pal 0	Light World Rocky/Grass Tiles
	World Colors 1 pal 1	Dark World Rocky/Grass Tiles
	World Colors 1 pal 2	Light World Mountain Area
	World Colors 1 pal 3	Dark World Mountain Area
	World Colors 1 pal 4	Introduction/Ending sequences (Beginning dialogue/Triforce Screen)
	World Colors 1 pal 5	Title Screen Related

WORLD COLORS 2:

Palette as seen in HM	Palette Name	Description
	World colors 2 pal 0	Many sprites use those colors (LightWorld)
	World colors 2 pal 1	Many sprites use those colors (DarkWorld)

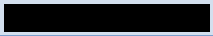
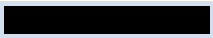
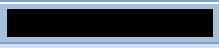
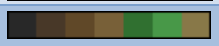
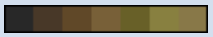
AREA COLORS 1: (Colors for the Palette# which you can edit in the Overworld screens)

Palette as seen in HM	Palette Name	Description
	Area Colors 1 pal 0	Overworld General Palettes (Used Almost Everywhere)
	Area Colors 1 pal 1	Overworld Palette #1 (Church Area)

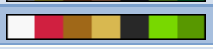
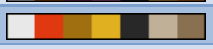
	Area Colors 1 pal 2	Overworld Palette #2 (Hyrule Castle)
	Area Colors 1 pal 3	Overworld Palette #3 (Title screen related eg. ZELDA etc..)
	Area Colors 1 pal 4	Overworld Palette #4 (Light World - Swamp Areas)
	Area Colors 1 pal 5	Overworld Palette #5 ???? unused ????
	Area Colors 1 pal 6	Overworld Palette #6 (Light World - Lost Woods)
	Area Colors 1 pal 7	Overworld Palette #7 (Light World - Death Mountain)
	Area Colors 1 pal 8	Overworld Palette #8 (Light World - Kakariko Village)
	Area Colors 1 pal 9	Overworld Palette #9 (Light World - The Desert)
	Area Colors 1 pal 10	Overworld Palette #10 (Light World - Zora's Domain)
	Area Colors 1 pal 11	Overworld Palette #26 (Dark World - Monkey Area)
	Area Colors 1 pal 12	Overworld Palette #27 (Dark World - Skull Woods)
	Area Colors 1 pal 13	Overworld Palette #28 (Dark World - Ice/Swamp Palace)
	Area Colors 1 pal 14	Overworld Palette #29 (Dark World - Swamp Areas)
	Area Colors 1 pal 15	Overworld Palette #30 (Dark World - Thieves Town)
	Area Colors 1 pal 16	Overworld Palette #16 (Dark World - General Palettes)
	Area Colors 1 pal 17	Overworld Palette #18 (Dark World - Pyramid Area)
	Area Colors 1 pal 18	Overworld Palette #23 (Dark World - Ganon's Tower)
	Area Colors 1 pal 19	Overworld Palette #24 (Dark World - Turtle Rock)

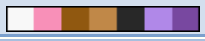
AREA COLORS 2: (Special overworld palettes)

Palette as seen in HM	Palette Name	Description

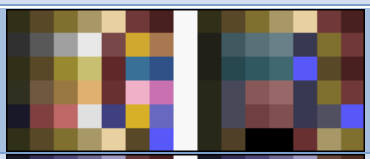
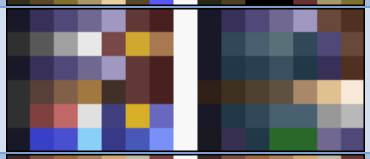
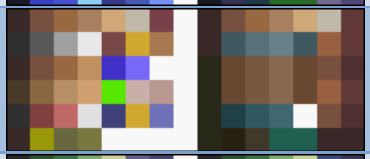
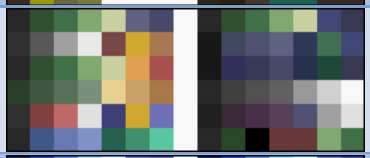
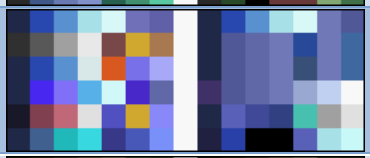
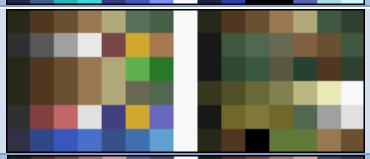
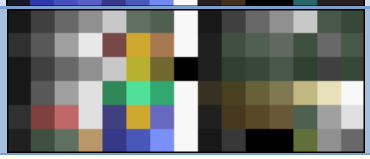
	Area Colors 2 pal 0	Unknown
	Area Colors 2 pal 1	Nothing/Unused
	Area Colors 2 pal 2	Nothing/Unused
	Area Colors 2 pal 3	Nothing/Unused
	Area Colors 2 pal 4	Overworld Palette #23 (Animations)
	Area Colors 2 pal 5	Overworld Palette #7 (Animations)
	Area Colors 2 pal 6	16x16 Block Editor Palette #7 (Dark World)
	Area Colors 2 pal 7	16x16 Block Editor Palette #7 (Light World)
	Area Colors 2 pal 8	Nothing/Unused
	Area Colors 2 pal 9	Nothing/Unused
	Area Colors 2 pal 10	Nothing/Unused
	Area Colors 2 pal 11	Nothing/Unused
	Area Colors 2 pal 12	Nothing/Unused
	Area Colors 2 pal 13	Nothing/Unused
	Area Colors 2 pal 14	16x16 Block Editor Palette #2 (Light World)
	Area Colors 2 pal 15	16x16 Block Editor Palette #2 (Dark World)

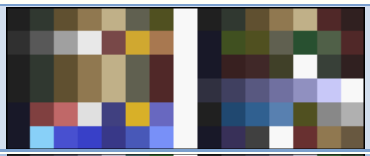
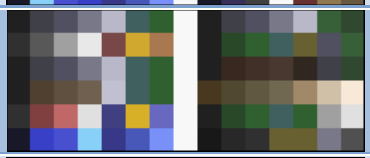
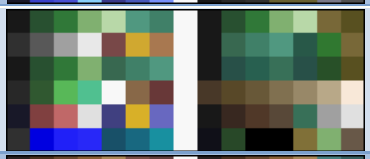
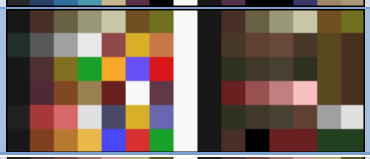
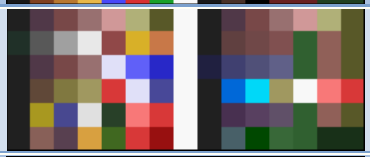
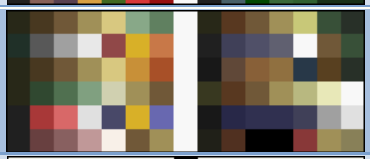
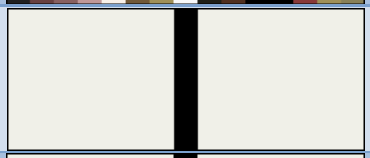
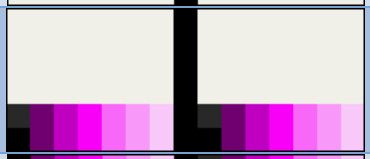
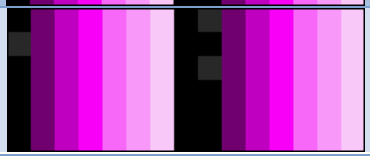
ENEMIES 1: (Sprites palettes)

Palette as seen in HM	Palette Name	Description
	Enemies 1 pal 0	Unknown
	Enemies 1 pal 1	Unknown
	Enemies 1 pal 2	Unknown
	Enemies 1 pal 3	Green Soldier, Link's Uncle
	Enemies 1 pal 4	Unknown
	Enemies 1 pal 5	Unknown
	Enemies 1 pal 6	Unknown
	Enemies 1 pal 7	Unknown
	Enemies 1 pal 8	Unknown
	Enemies 1 pal 9	Unknown
	Enemies 1 pal 10	Unknown
	Enemies 1 pal 11	Unknown
	Enemies 1 pal 12	Unknown
	Enemies 1 pal 13	Unknown
	Enemies 1 pal 14	Unknown
	Enemies 1 pal 15	Unknown
	Enemies 1 pal 16	Unknown

	Enemies 1 pal 17	Unknown
	Enemies 1 pal 18	Unknown
	Enemies 1 pal 19	Unknown
	Enemies 1 pal 20	Unknown
	Enemies 1 pal 21	Unknown
	Enemies 1 pal 22	Unknown
	Enemies 1 pal 23	Unknown

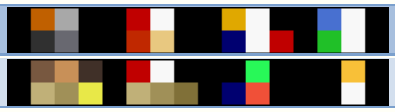
DUNGEONS: (Dungeon GFX# Palettes)

Palette as seen in HM	Palette Name	Description
	Dungeons pal 0	Unknown
	Dungeons pal 1	Blue Caves
	Dungeons pal 2	Houses
	Dungeons pal 3	Green Dungeon
	Dungeons pal 4	Ice Dungeon
	Dungeons pal 5	Desert Dungeon
	Dungeons pal 6	Unknown
	Dungeons pal 7	Unknown
	Dungeons pal 8	Unknown

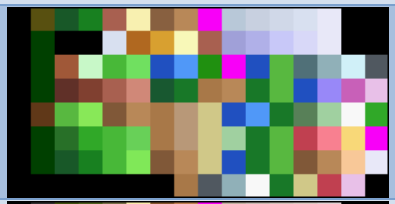
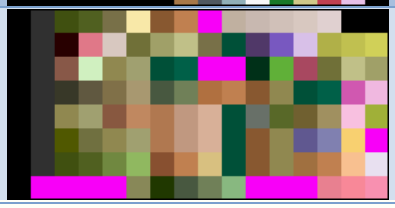
	Dungeons pal 9	Unknown
	Dungeons pal 10	Unknown
	Dungeons pal 11	Misery Mire (Dark World)
	Dungeons pal 12	Unknown
	Dungeons pal 13	Unknown
	Dungeons pal 14	Church/Sanctuary
	Dungeons pal 15	Unknown
	Dungeons pal 16	Unknown
	Dungeons pal 17	Unknown
	Dungeons pal 18	Unknown
	Dungeons pal 19	Unknown

MISCELLANOUS COLORS: (Font Colors, Inventory and HUD overlays)

Palette as seen in HM	Palette Name	Description

	Miscellanous colors pal 0	Font Colors, Inventory and HUD overlays
	Miscellanous colors pal 1	Font Colors, Intro sequence (2pbb pictures)

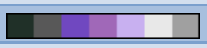
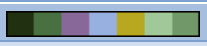
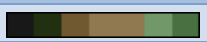
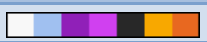
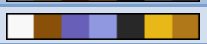
WORLD MAP:

Palette as seen in HM	Palette Name	Description
	World map pal 0	World Map (Light World)
	World map pal 1	World Map (Dark World)

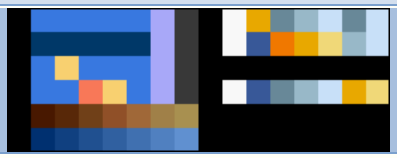
ENEMIES 2: (Dungeon enemy's sprites and Title screens palettes)

Palette as seen in HM	Palette Name	Description
	Enemies 2 pal 0	Unknown
	Enemies 2 pal 1	Unknown
	Enemies 2 pal 2	Unknown
	Enemies 2 pal 3	Unknown
	Enemies 2 pal 4	Unknown
	Enemies 2 pal 5	Unknown
	Enemies 2 pal 6	Trinexx (core)
	Enemies 2 pal 7	Unknown
	Enemies 2 pal 8	Unknown
	Enemies 2 pal 9	Unknown
	Enemies 2 pal 10	Unknown
	Enemies 2 pal 11	Title Screen Palette 8 (Master Sword)
	Enemies 2 pal 12	Unknown
	Enemies 2 pal 13	Unknown
	Enemies 2 pal 14	Unknown
	Enemies 2 pal 15	Unknown

OTHER SPRITES: (Special sprites like the bird statue, soldiers, animations, objects you pickup)

Palette as seen in HM	Palette Name	Description
	Other sprites pal 0	Unknown
	Other sprites pal 1	Unknown
	Other sprites pal 2	Unknown
	Other sprites pal 3	Unknown
	Other sprites pal 4	Unknown
	Other sprites pal 5	Unknown
	Other sprites pal 6	Objects you pick up (Lighter) (Light World)
	Other sprites pal 7	Objects you pick up (Darker) (Light World)
	Other sprites pal 8	Objects you pick up (Lighter) (Dark World)
	Other sprites pal 9	Objects you pick up (Darker) (Dark World)
	Other sprites pal 10	
	Other sprites pal 11	
	Other sprites pal 12	
	Other sprites pal 13	
	Other sprites pal 14	PEOPLE 167 - 189 - 192
	Other sprites pal 15	
	Other sprites pal 16	
	Other sprites pal 17	

DUNGEON MAP:

Palette as seen in HM	Palette Name	Description
	Dungeon map pal 0	Self-Explanatory

TRIFORCE:

Palette as seen in HM	Palette Name	Description
	Triforce pal 0	Self-Explanatory

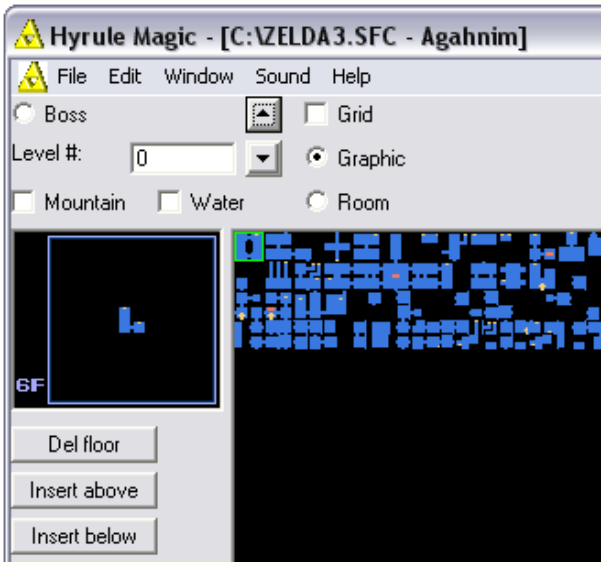
CRYSTAL:

Palette as		

seen in HM	Palette Name	Description
	Crystal pal 0	Self-Explanatory

13) Dungeon Maps

by Sephiroth3



In this editor you can change the dungeon map. Click on a block in the selector and then on the room to change it. Select the room you want to change and type the room number (In hex) or press **N** to remove it from the map. You can move the entire dungeon up or down by clicking with the left or the right mouse button on the floor number.

Grid:
This check box shows the grid.

Boss:
This allows you to select anywhere in the map on the left and shows in-game the location of the dungeon boss (If one exists). The boss cannot be placed outside of a room.

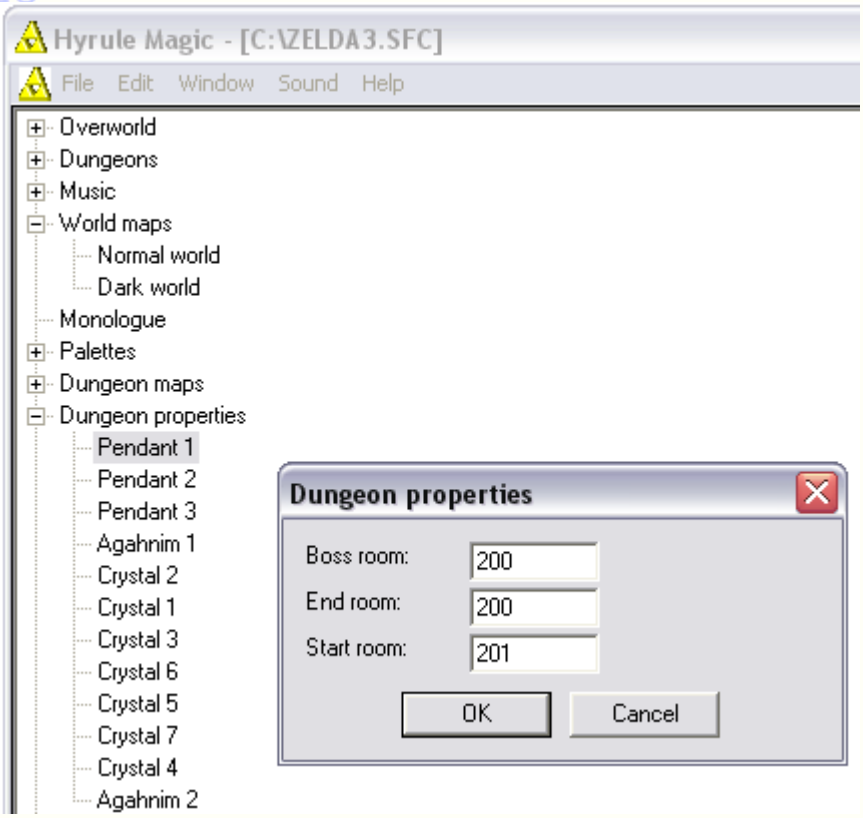
Room:
This will show location of the rooms. A part of the map can be designated to a Room reference. These however are done in Hexadecimal co-ordinates so you will need to convert the Room numbers from the dungeon editor. (e.g. The first room in the game. Link's house. This is Room number 260. In the editor if you wanted to make that one room on the dungeon map then you would select the map area and input the number **104** (Which is the Hex equivalent). If you can't calculate the numbers in your head then you could use the Windows basic scientific calculator).

Graphic:
This allows you to select any of the map GFX on the right and place it down in the map box on the left. The problem with the graphics is that they rarely fit the layout you want. This means you have to edit all the gfx to fit.

201 > Dungeon Map 1 GFX
202 > Dungeon Map 2 GFX
212 > Dungeon Map GFX
213 > Dungeon Map GFX
214 > Dungeon Map GFX

14) Dungeon Properties

by Orochimaru



Legend:

Boss Room > *The room where you fight the big boss.*
End Room > *The room that ends the dungeon.*
Start Room > *The first room you see when you enter the dungeon.*

Pendant 1
Boss Room: ***200***
End Room: ***200***
Start Room: ***201***

Pendant 2
Boss Room: ***51***
End Room: ***51***
Start Room: ***99***

Pendant 3
Boss Room: ***7***
End Room: ***7***
Start Room: ***119***

Agahnim 1
Boss Room: ***32***
End Room: ***32***
Start Room: ***32***

Crystal 1
Boss Room: ***90***
End Room: ***90***
Start Room: ***74***

Crystal 2
Boss Room: ***6***
End Room: ***6***
Start Room: ***40***

Crystal 3
Boss Room: ***41***
End Room: ***41***
Start Room: ***89***

Crystal 4
Boss Room: ***172***
End Room: ***172***
Start Room: ***219***

Crystal 5
Boss Room: ***222***
End Room: ***222***
Start Room: ***14***

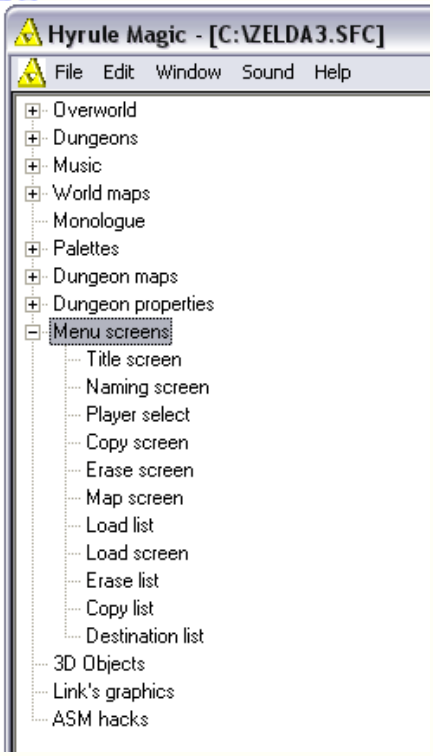
Crystal 6
Boss Room: ***144***
End Room: ***144***
Start Room: ***152***

Crystal 7
Boss Room: ***164***
End Room: ***164***
Start Room: ***214***

Agahnim 2
Boss Room: ***13***
End Room: ***13***
Start Room: ***13***

15) Menu Screens

by Sephiroth3, Orochimaru and MyPiop



Information by Sephiroth3:

This editor allows you to change the menus screens of the game. With the **move button** you can click and drag to move stretches of tiles. Pressing **M** changes between horizontal and vertical. Pressing "**,**"decreases while "**.**" increases the length. Pressing "**-**" changes between normal and fill. It is recommended that you don't change the size of anything these characters in the last 5 screens as this might cause the game to crash. With the **draw button**, the selected tile can be painted on a stretch. By pressing the right mouse button, a new stretch can be placed.

a) Title screen

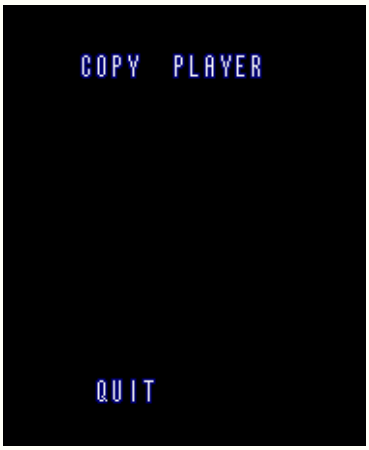
Valuable information by MyPiop: *On the title screen, two similar palettes can throw you off guard.*

2-4 are palettes that are used **"behind the logo"** (appears after the sword comes down).
5-7 are palettes that are used **"for the logo"** (appears after the Triforce dance).

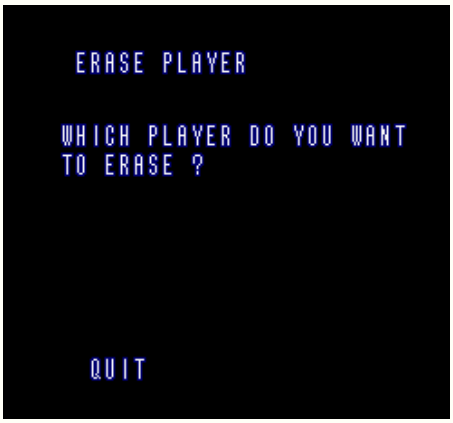
b) Naming screen

c) Player screen

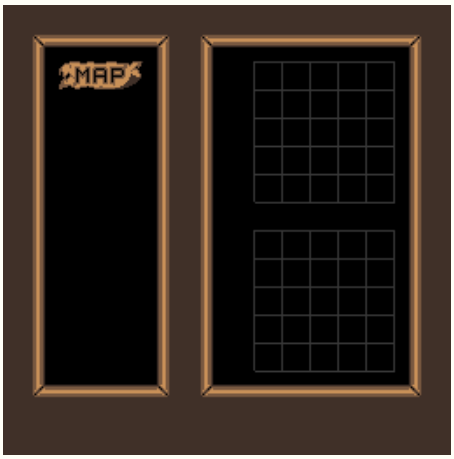
d) Copy screen



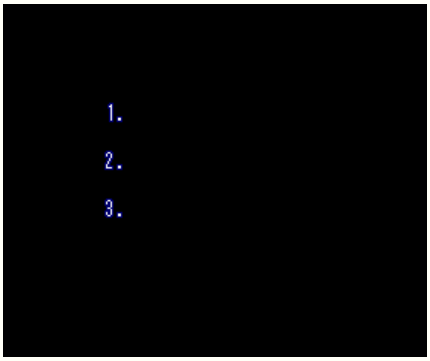
e) Erase screen



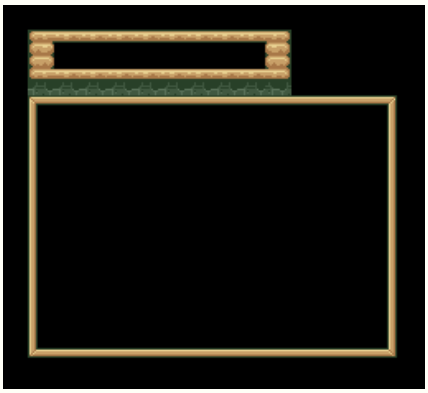
f) Map screen



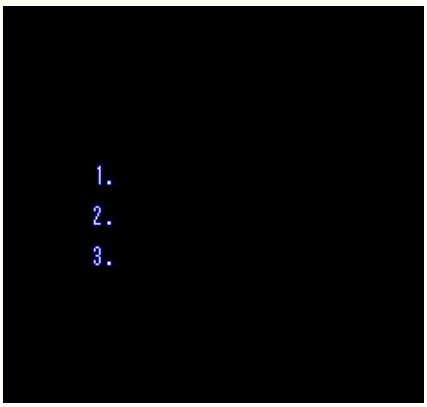
g) Load list



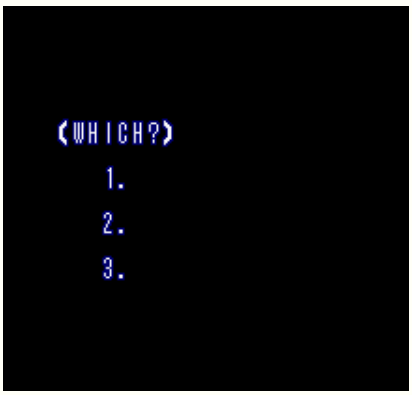
h) Load screen



i) Erase list



j) Copy list



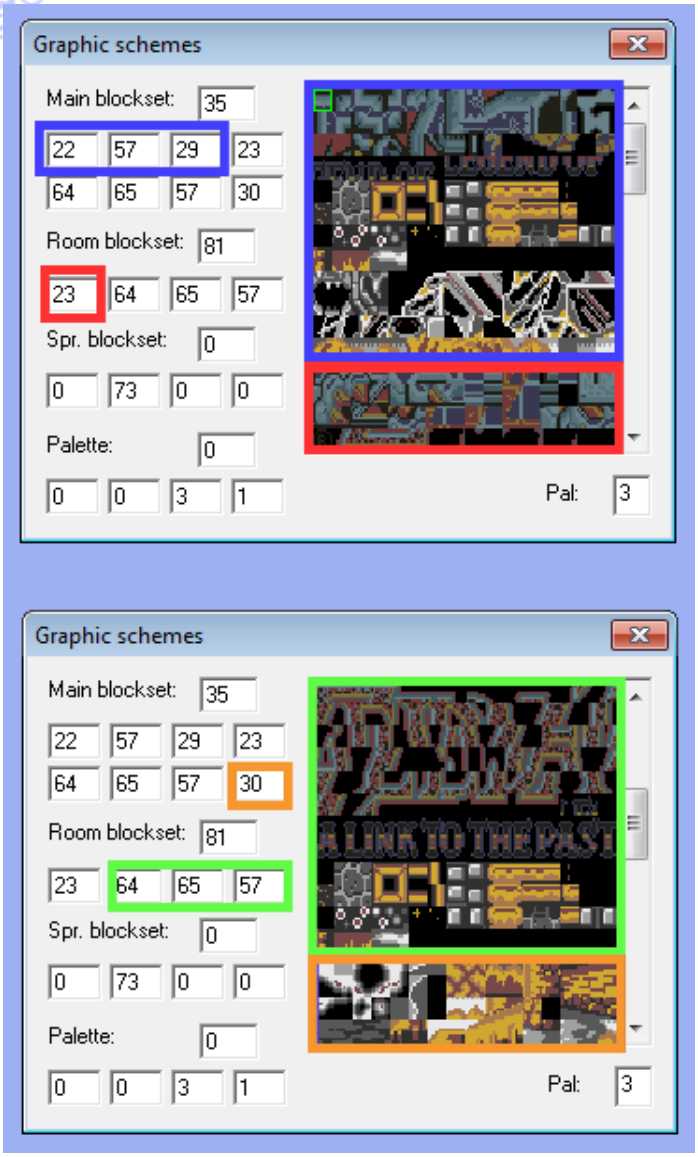
k) Destination list

1) Expending the menu screens graphics space

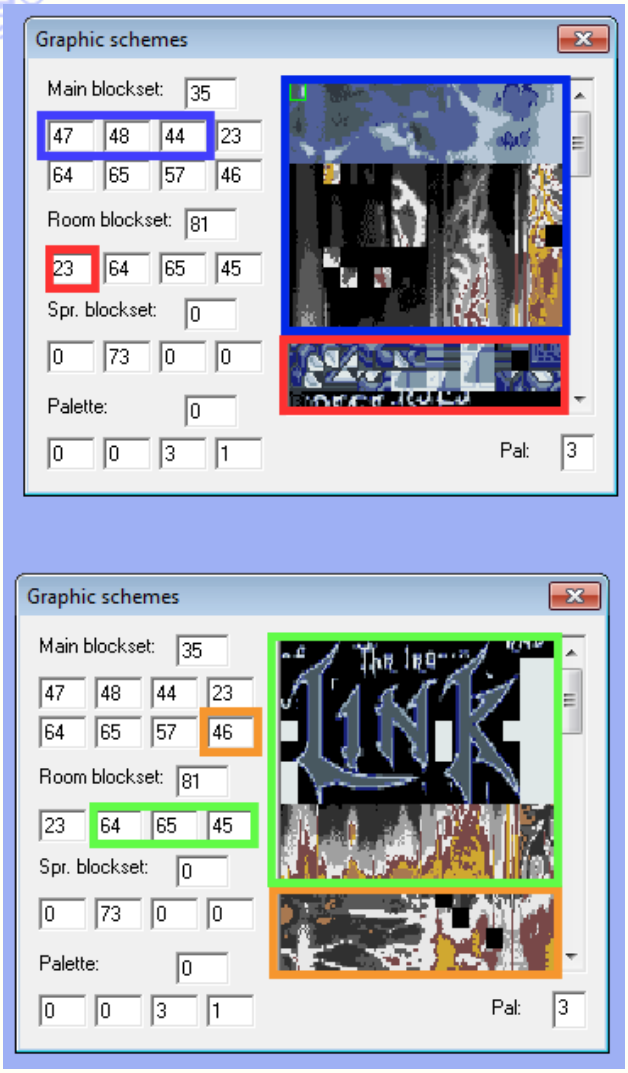
by Orochimaru

I've come up with something great using the graphic shemes editor in the version 0.963 of Hyrule Magic. Go in the graphic shemes editor, change the "Main blockset" value to 35 and the "Room blockset" to 81... don't these graphics ring a bell? They should as they are the menu screens default graphics set. Now open up the title screen editor and there you go, the same graphic sets.

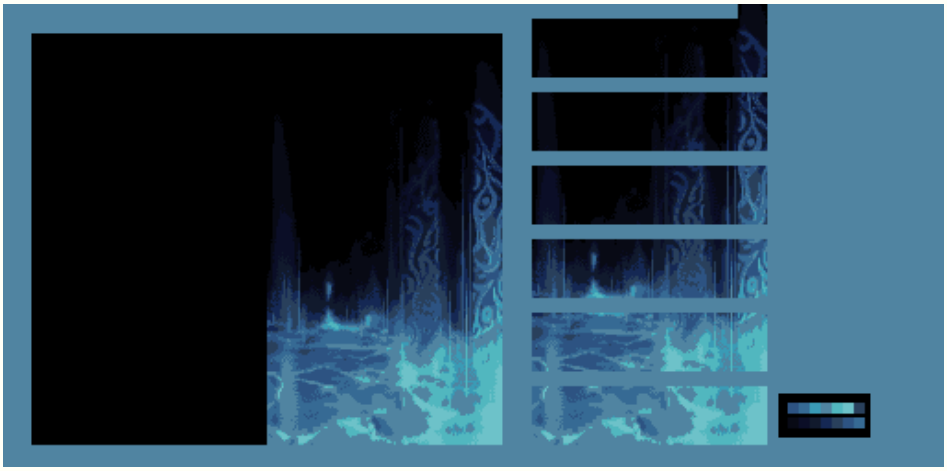
Here's how the original game menu screens graphics are setup:



Then here's mine in my romhack:



The overall space it all takes:

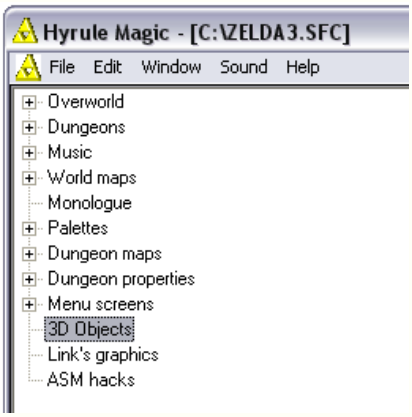


And the end result:

There's also some more work involved afterward. Like creating another main blockset and room blockset for the objects used in those we just changed. But there's nothing the graphic shemes doesn't let you change...

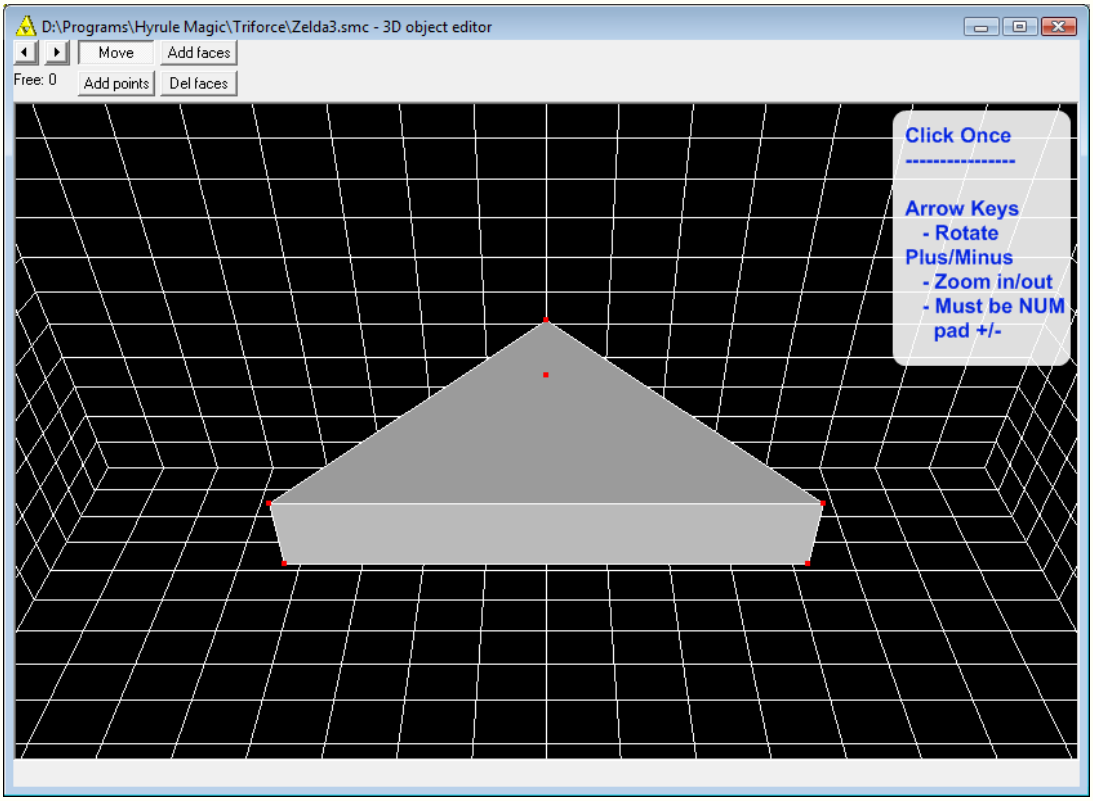
16) 3D Objects

by sorlokreaves



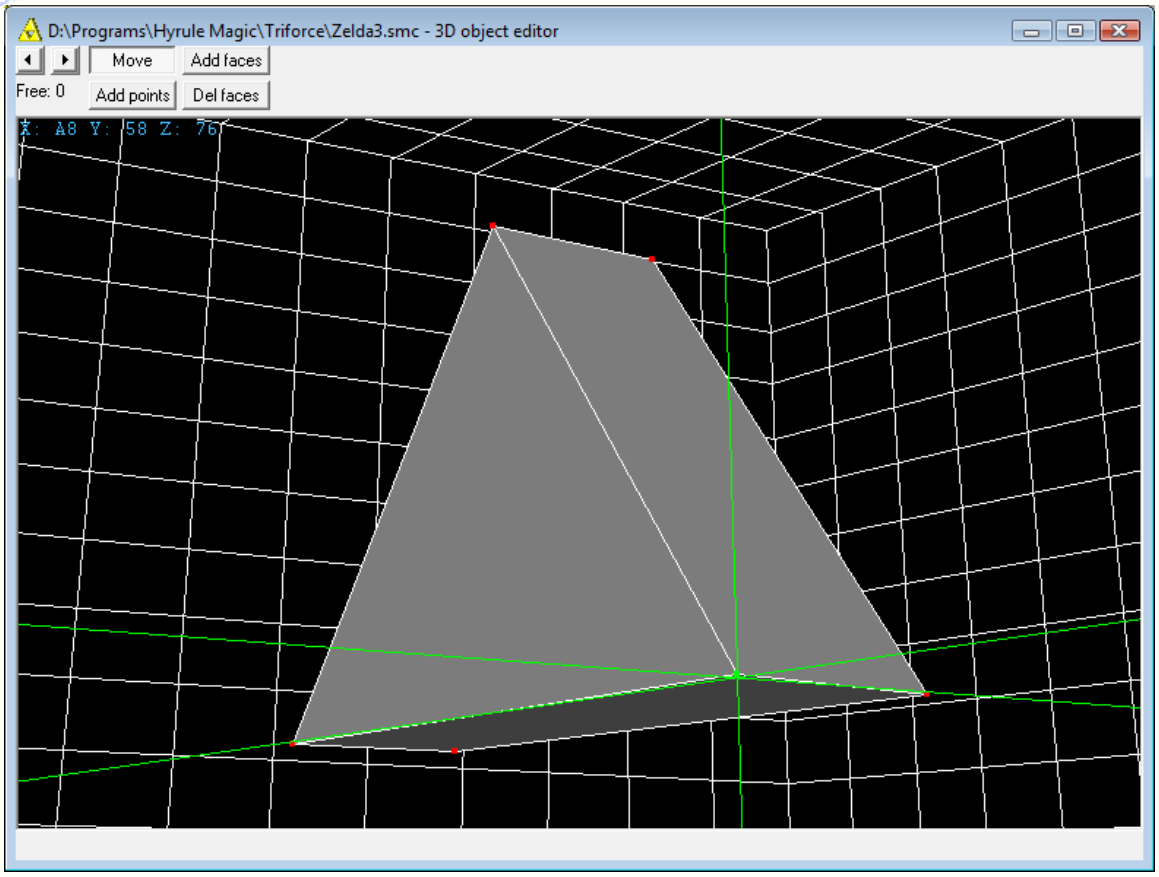
This editor lets you edit the crystal and the triforce.

Open a clean Zelda3 ROM in Hyrule Magic and click the "3D Objects" tab. Then click once on the black, gridded area to focus it. Now, you can rotate the scene (arrow keys) and zoom (num pad plus/minus). The current view, of the crystals, is somewhat difficult to grasp, so hit the right arrow in the upper-left of the screen to switch to editing the triforce shards.



Picture: Hyrule Magic's 3D shape editor is impressive, but slightly tricky to use.

You should then click on each point on the top surface of the triforce to see its location in 3D-space, which will appear in the upper-left corner of the screen.

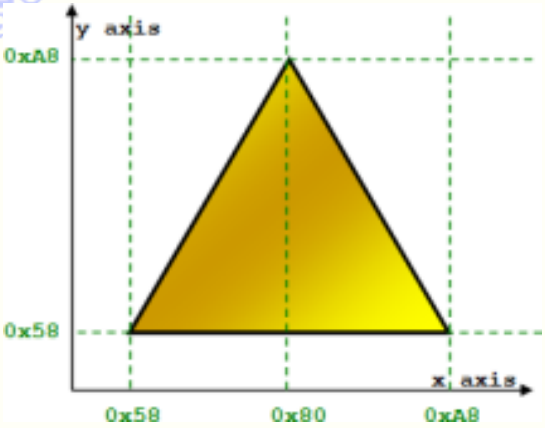


Picture: Click on a point to show its position in 3D space.

Here, we can see that this point is located at X:A8, Y:58, Z:76. Click on the remaining points and jot them down. These coordinates, of course, are hex values.

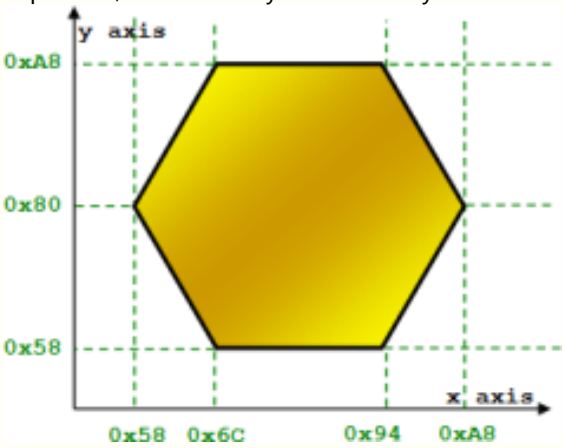
Face	***X***	***Y***	***Z***
Top Face	A8	58	76
	58	58	76
	80	A8	76
Bottom Face	A8	58	8A
	58	58	8A
	80	8A	8A

The triangle is 0×14 points thick, with the bottom face flat on 0x8A and the top face flat on 0×76. The remaining Cartesian co-ordinates are distributed as an equilateral triangle:



Picture: Our equilateral triforce piece.

We would like to replace the triangle with a nice hexagonal shape. However, if you try to add a point, you'll notice that we don't have any more points to work with. So, click the left arrow (at the top-left of the window) to switch back to the crystals, and then click on any one of the points. Hit "Backspace" to delete this point —the cursor will then automatically jump to the next point. Continue hitting "backspace" until all points have been deleted. Then, hit the "right" arrow to go to the triforce triangle, and delete all points here too.
Now, draw a rough sketch of your hexagon; it should be roughly the same size as the triangle. Drawing a sketch is important, as it allows you to identify vertices easily. You should end up with something like this:

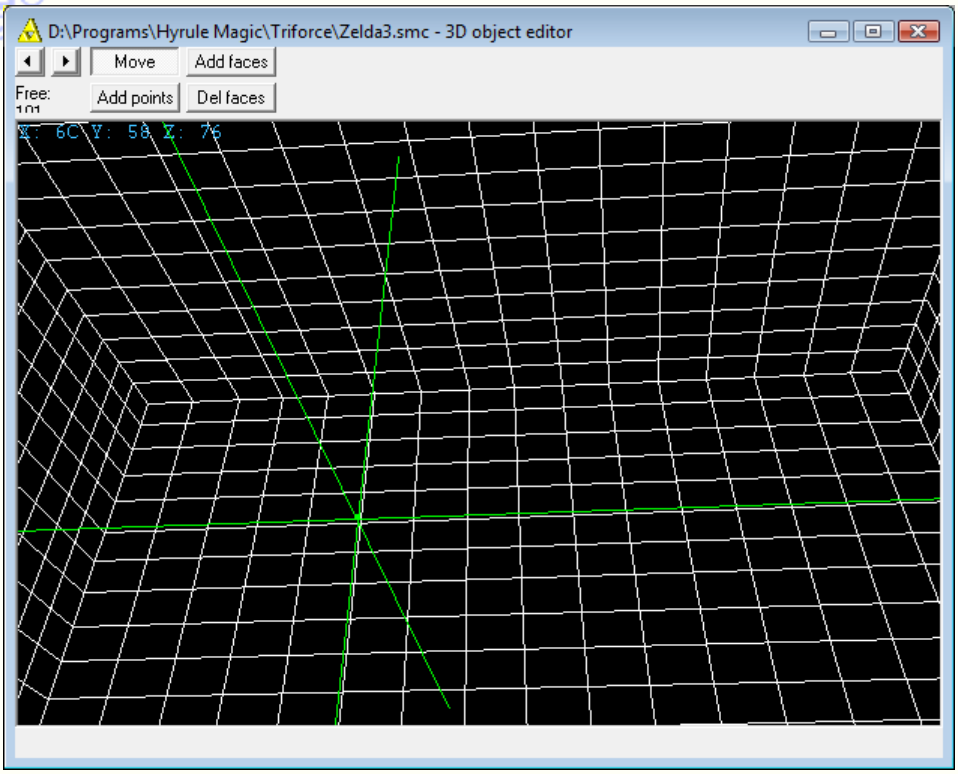


Picture: Our Hexa-Force is at least twice as powerful as the triforce!

Now we have our six points in a single z-plane:
(6C,58), (94,58), (A8,80), (94,A8), (6C,A8), (58,80)
You'll notice that I wrote these out in a counter-clockwise order. The reason will be important later: Hyrule Magic creates an "outward facing" shape when points are connected counter-clockwise.

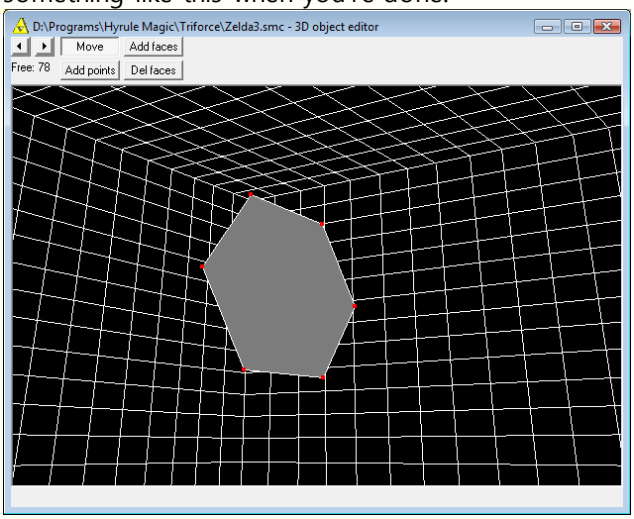
Now, we need to enter these points. To enter the first point, (6C,58), do the following:

1. Click the "Add Points" button, and click once on the black area. This will pull up a blue line and two green lines, indicating the temporary placement of your point.
2. Click on one of the green lines to place your point. Place it anywhere; we'll move it manually to make sure it ends up at the exact position we choose.
3. Click the "Move" button, then left-click once on your new point. Now, press the A or Z key until the point's "Z coordinate" is at 76. Then, use the numeric keypad (this part sucks on a laptop) to shift the point's X/Y coordinates to 6C and 58, respectively.



Picture: Our first point, positioned correctly In 3D space.

Continue to add all the points; the order you **add** them in is not important (only the order you **connect** them in). The Z-component will always be 76, since we're only building the top face for now. Then, after all points have been added, rotate the view to the origin (the easy way: save, close, and re-open your ROM in Hyrule Magic), click the "Add Faces" button and click on each point in order. It doesn't matter where you start, so long as you go in a **counter**-clockwise direction, and end at your starting point. As you click on each point, a blue line will trace the current shape you're creating. You should have something like this when you're done:



Picture: Our first hexagon surface. 1-Dimensional.

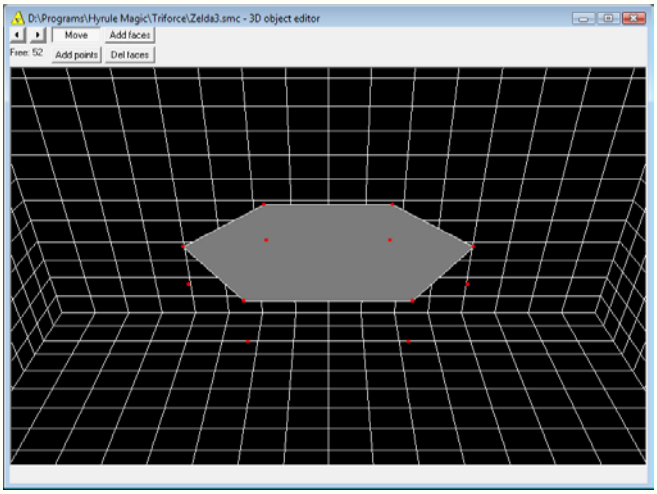
Save your file and close the editor. Start up the ROM, and you'll see some blinking hexagons:

Picture: Our 1-D hexagons often vanish, but prove the concept.

This is fine; the back face of the hexagons is invisible; we'll fix that later. If, instead of loading, your game shows some glitched graphics and then freezes up, you probably edited the crystal's graphics instead of the triforce graphics (d'oh!). Just delete all points, hit the "right" arrow, and start again.

Let's add the second set of points. Load your ROM in Hyrule Magic and go to the 3D Objects editor, pressing "right" to go to the triforce graphics. You should see the hexagon facing you directly; hold "Down" to rotate the screen until you are seeing the back of the hexagon. Now, add six new points to the object. They have the same X and Y coordinates as the previous points, but their Z-coordinates are all the same: 8A.

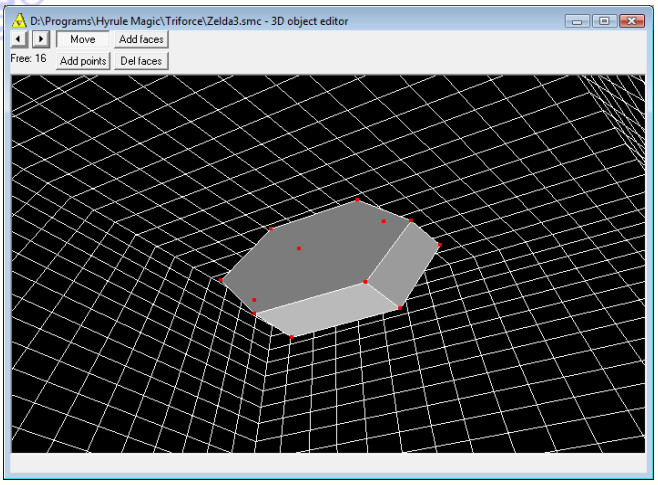
After you've added these points, hit "Up" two or three times so that you can tell these points apart from the first set. Then, connect all six in a counter-clockwise direction using the "Add Face" button. You now should have two hexagons facing away from each other:



Picture: Our 2 faces of the coin have been added. One should always be invisible.

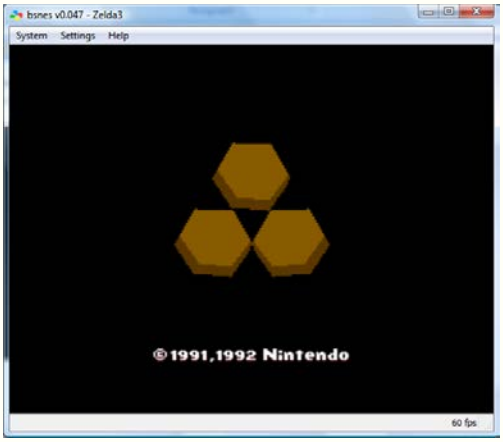
There's a small chance that you'll corrupt your first face by adding the points for the second. In this case, the "Delete Face" button might not even work! You'll have to delete one of the ***points*** contained in the face. (In my case, there was a weird triangular shape; I just deleted and re-added one of the points of that triangle.) In general, you'll want to add all points before you connect any faces.

Either way, you only have one more task before we're done: just add each of the six rectangles that connect the top and bottom two hexagons. Rotate the window as you go, and make sure you connect all points in a counter-clockwise fashion. When you're done, you'll have a nice, coin-ish object.



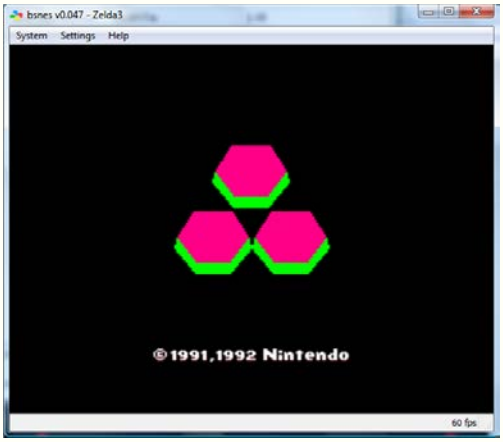
Picture: My favourite pic from this entire post: Your complete coin!

Save and exit, then load your game. My only regret is that I didn't actually do the math properly, so our coin is a bit "longer" than it should be.



Picture: Our coins rotate into position. Hooray, 3-D Editor!

Does anyone else find it interesting that **bsnes** and **zsnes** display the opening zoom-in differently? Also, although it should be painfully obvious by now, you can change the palette used for the triforce just like anything else in the game:

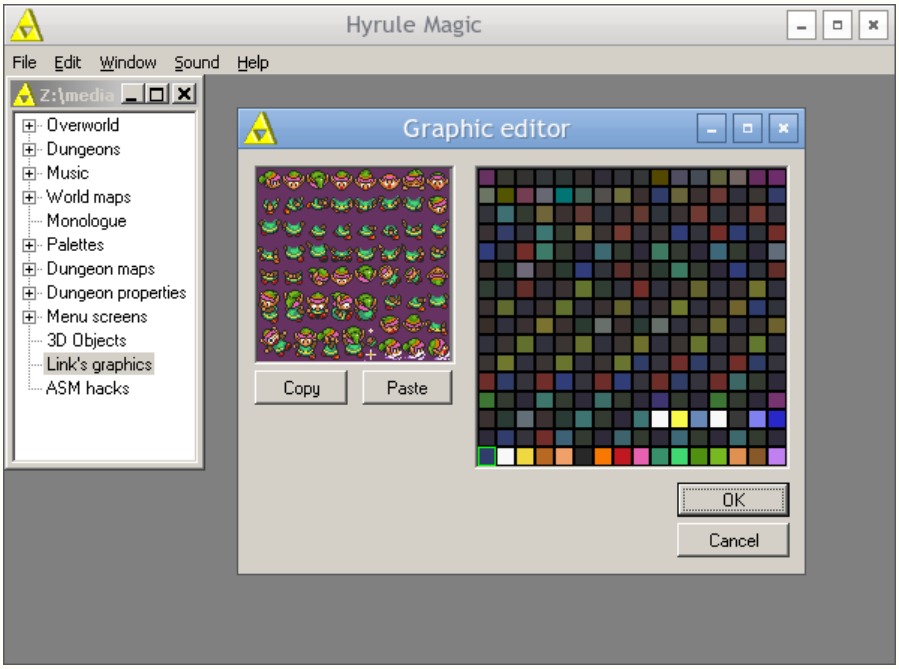


Picture: Go rainbow laser Triforce!

17) Link's Graphics

by sorlokreaves

One of the latest additions to Hyrule Magic is a "Link's Graphics" tab which gives you access to the 4bpp image data associated with our hero. It's a bit tricky to use properly, though.



Picture: Exporting Link's graphics. Palette is somewhat messed up, since we're using wine. But it should still work ok.

The poses on the left are Link's animation graphics, and the squares on the right are potential palettes. (You can click on various squares to mess with Link's graphics, but make sure you click on the lower-left most square before doing any serious work.) To export graphics click the "copy" button, and then go to your favourite image editor and choose "paste".

You might want to do this in Windows for two reasons. First, although Link's palette (the bottom row) is shown properly in Wine, all other palette colors are messed up. Although the graphics editor doesn't directly handle palette data, I personally wouldn't risk having Wine corrupt your project; Hyrule Magic does enough corruption on its own. Second, **one of the best** free raster editing programs for Windows has only spotty success under Wine. On Linux, I've yet to find a native equivalent —I consider The Gimp to be terrible for pixel-based editing. Either way, you're on your own for ***how*** you edit the copied graphics. Just make sure you save your file as a 4-bit (16-color) BMP file. If you're using an editor that doesn't support 4bit BMPs, just save it as a True Color BMP and then convert it later (see the next question).

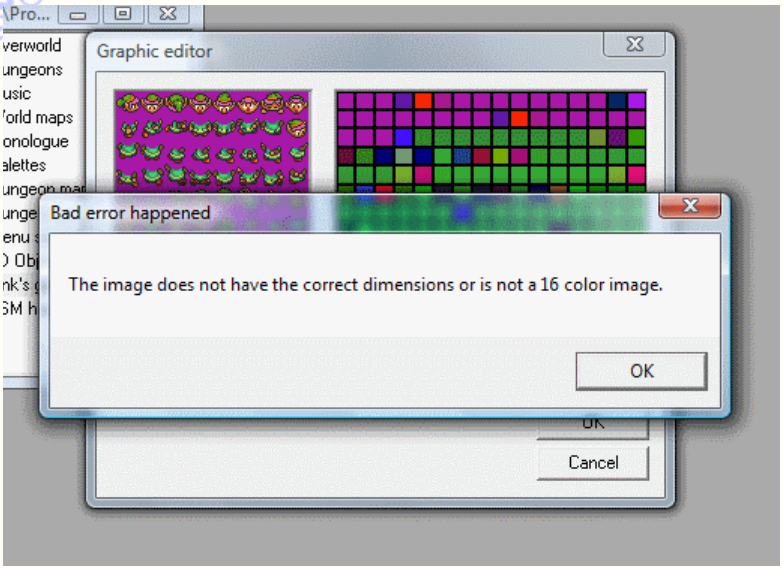
Let's give Link a diving-style helmet when he swims. Simply edit the three side-facing and two forward-facing heads with water around them. Give Link the helmet, and also remove the "mouth opening" graphics, like so:

Picture: Some of Link's Graphics, Edited to Add a Helmet. (Background Faded For Emphasis.)

Make sure that when you edit these graphics, you don't use any new colors. ***Only*** use colors that are already in Link's graphics. When you're done, perform a "Select All" and then a "copy" from your image editor, then go into Hyrule Magic and choose "Paste" from the "Link's Graphics" window. Save your game, and try it out.

Picture: Link is swimming with his new safety helmet ON.
It's not perfect (several more graphics need to be changed to make the animation accurate), but it shows the concept.

If Link's colors are messed up, or if you got the following error when you pasted in Link's graphics, then you probably failed to save the image in proper 16-color format.



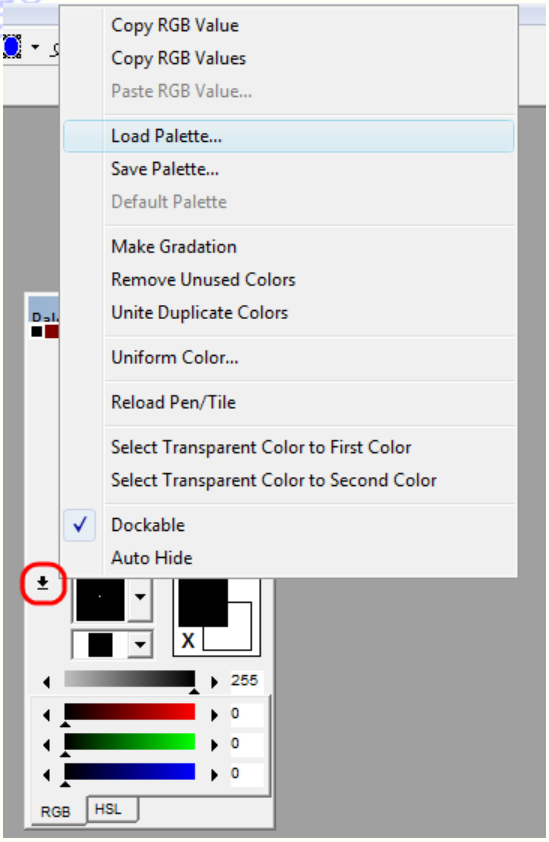
Picture: The error you get when you try to import a picture with too many colors.

Hyrule Magic is very picky about the format of its images. Personally, I think it should have exported and imported **files** instead of clipboard data, but whatever their rationale, the reality is that you often get a "Bad error" message or screwed up colors when you re-import your graphics.

The best way to avoid this problem is at the very beginning: by saving the copied graphics properly as a 4bpp, 16-color bitmap before you perform any editing at all.

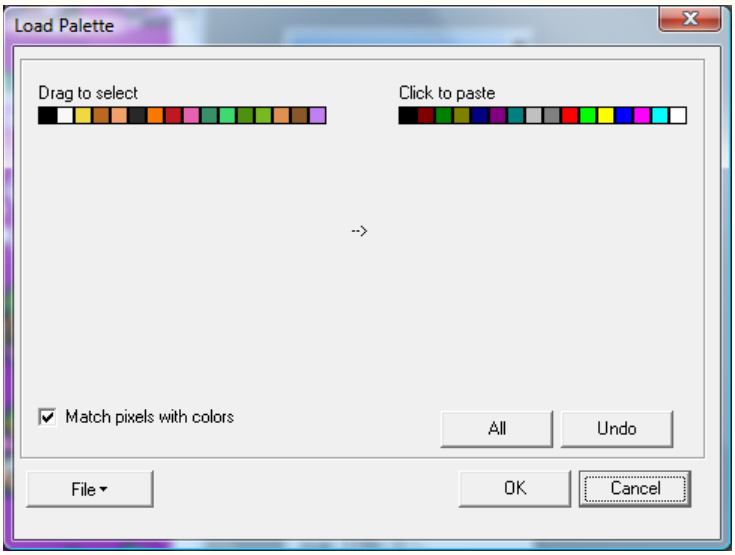
We're going to use the free version of **Graphics Gale**, an excellent raster editing program. First, open the program. Then, choose "File->New", and enter 128 for the width, 448 for the height, and 4-bit (16 color) for the bit depth. Choose "Ok".

Now, open a clean Zelda3 ROM in Hyrule Magic, go to the "Link's graphics" window, and select "Copy". Then, go back to Graphics Gale. Click on the down-pointing arrow in the palette window, circled below. Then, choose "Load Palette".



Picture: Graphics Gale hides its menus in non-standard context buttons.

This will bring up a new menu; here, click on "File" and then choose "Import from Clipboard". You'll see the palette from Link's graphics side-by-side with your current palette.



Picture: Copying Link's palette in Graphics Gale.

Since we want to use all the new (left-hand-side) colors, click and drag to select all of them, and then click once on the first color of the right-hand palette to paste them in order. (You might also try the "All" button, though I've never used it.) Then, click Ok. Now, you can simply paste Link's graphics into the main drawing window (Ctrl+V) and then save the file. At this point, you should be able to open this file using another program; most good editors won't change the palette without asking you. If that doesn't work, you can always edit your sprites in Graphics Gale; it's a decent stand-alone editor.

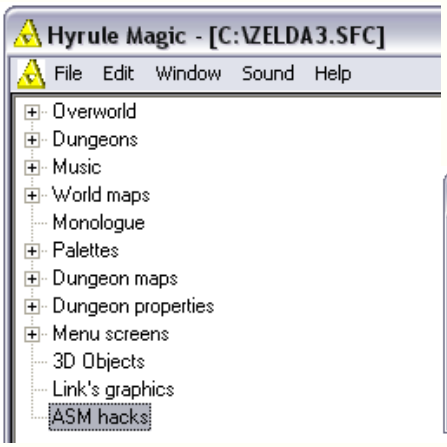
I've heard that Graphics Gale works on Linux under Wine, at least well enough to perform the steps listed above. If you prefer to use another editor on any OS, **here's some instructions** on how to save in 16-color format, in The Gimp, Photoshop, and other popular programs.

Remember, though, that saving in 16-color format won't fix the error; you also have to **copy** it in 16-color format. So you should always copy from Graphics Gale before pasting into Hyrule Magic. This is what makes Hyrule Magic difficult to use at times.

18) ASM Hacks

by Sephiroth3 and Euclid

The Hyrule Magic editor allows you to add ASM patches to your hacks but only if you have created them using the FSNASM assembler by Sephiroth3 (it needs to be placed in the same directory as your Hyrule Magic editor).



To add a source file to the list, press the **Add button**. Similarly, to remove a source file, press the **Remove button**. The **Build button** is Self-Explanatory.

Some info regarding FSNASM

by Sephiroth3

FSNASM (C) 1999,2001-2002 Sephiroth of Gigasoft
Usage: FSNASM [option] srcfile
Options:
-l: Produce LO-ROM
-o : Select output filename

You can use .b, .w, and .l to be explicit about what version of a particular instruction you want (so, for example, LDA.w). If you don't use those, the assembler will make its best guess.

Commands are: charset, defchar, endb, base, data, block, align, end, code, global, zram, incbin and org

"block" corresponds to resb in nasm
data, code, ram, zram and org define sections
I believe 'end' ends these segments.

defchar "EXAMPLECHARSET" "AZ"=1,"az"=27,"09"=118," "=0,s".'!?:,-;&/"=63,"\n"=129
charset "EXAMPLECHARSET"
;You can use \ to escape, for characters like "
ZRAM corresponds to addresses 0-$1fff i think, or maybe it's 0-$ff

the base directive tells it where the code really is
endb ends the base directive

If you're coding for the NES, you can use org $c00000 to put the code at the beginning of the ROM. With -l (the LoROM option), that would be org $808000

You can use incbin to include data. For example:
incbin "NESFONT.dat"
to include some NESFONT.dat file, which would presumably be your font graphics.

Use dc.x to include data, where x is b, w, or l. Data can be a mix of numbers (hex ($) or dec) and labels (or operations on labels, such as yourlabel-1)

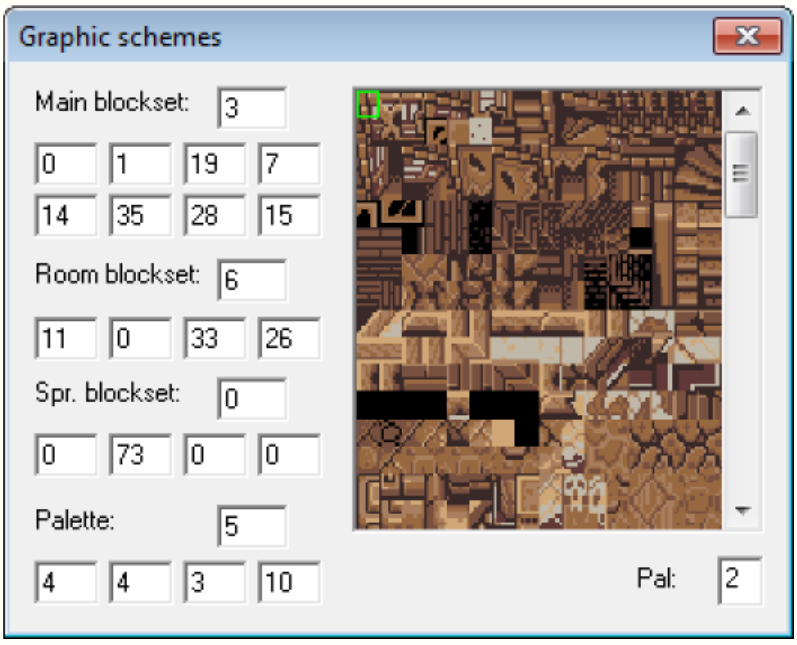
I've used this for NES development before and it works well. I had to do org $c00000 at the start, define the header (example: dc.b "NES",26,2,1,1,0,0,0,0,0,0,0,0,0), base $8000, and then you're ready to code (just be sure to endb, end, and set up your interrupt vectors at the end of the file). You can do variable = $blah to set up your variables. I think labels require a colon after the declaration unless they're naming a table, in which case it's just yourlabel dc.x yourdata.

Here's an exemple of an ASM hack of Euclid using FSNASM:

```
; FSNASM module by Euclid - 16/10/2006
; - makes boss drop nothing no matter what happens
;  but makes crystal drop if the room is tagged with clear level room header
; - side effects:
;  - boss drop nothing, always,
;  - boss will never revive itself even with kill boss again header

.ORG $9EE4F
LDA #$DA      ; give out 5 rupees just in case this patch fails
.end

.code
somewhere:
LDA $0403    ; teh code
ORA #$80
STA $0403
JSL $1DF65F  ; replacement code
RTL
.end


.ORG $9EE53
JSL somewhere ; overwrote a JSL
.end
```

19) Graphic schemes

by Orochimaru

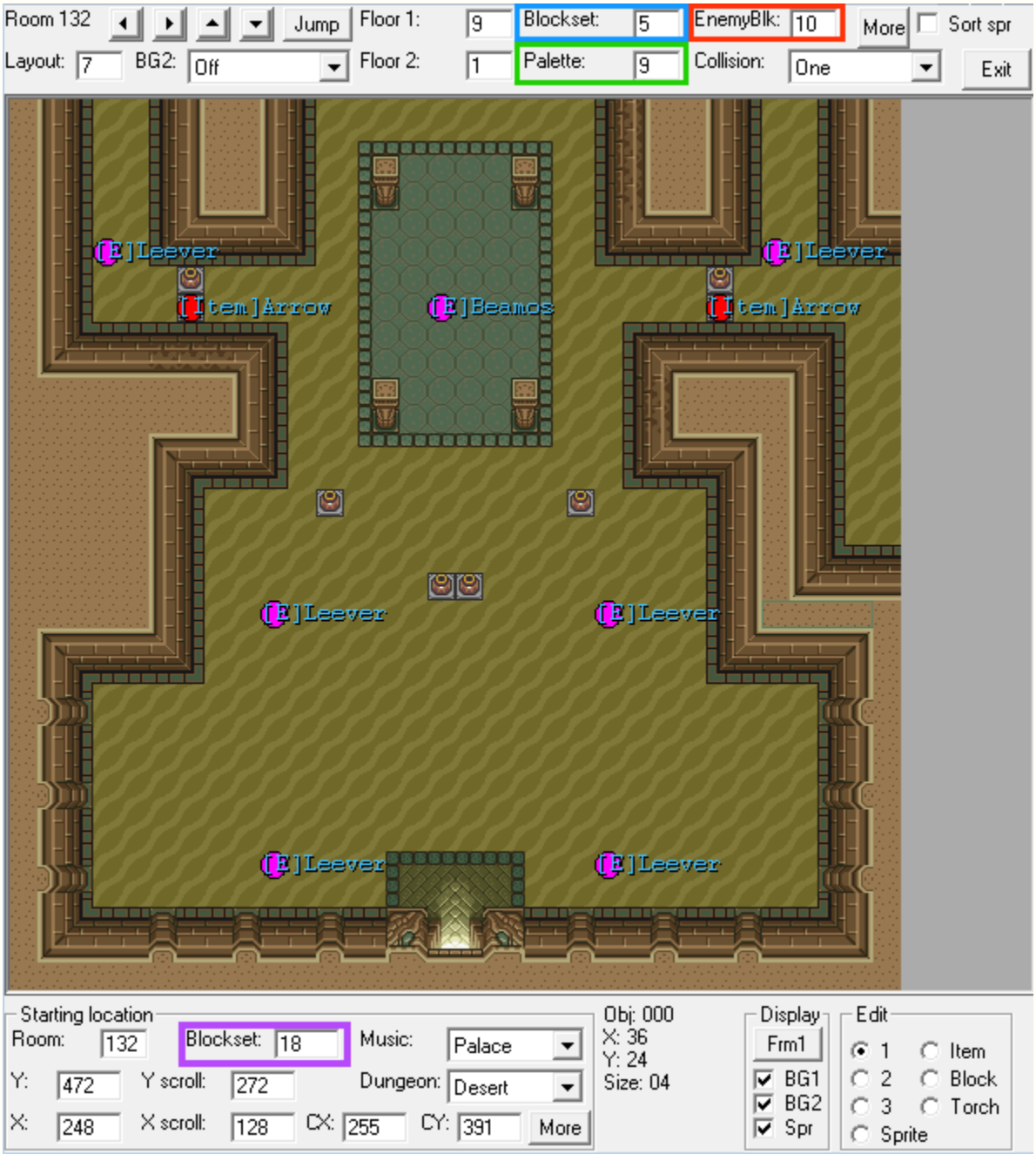


The graphic schemes is the latest addition to Hyrule Magic. It was added in version 0.963 which you can get here: http://math.arc-nova.org/HMagic_v963.zip. That menu let's you edit all the graphics sets in the game along their palettes. It's a very complex editor in nature and it's also very easy to get confused, so I'll break down this tutorial in several parts for easier understanding!

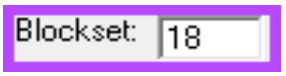
a) A simple introduction to graphic schemes

They say that the best way to learn is by example… I kinda agree with that statement, so let's start this tutorial by taking apart an whole dungeon room as an exemple to what each of the digits do! Open up your Hyrule Magic editor (make sure it's v0.963), go in the dungeon editor and open "Entrance 09". That dungeon in question is the 2nd dungeon in the original game.

Notice the various elements I've put an emphasis on:



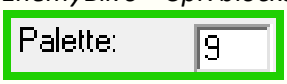
Legend:



= This is your "Main blockset"

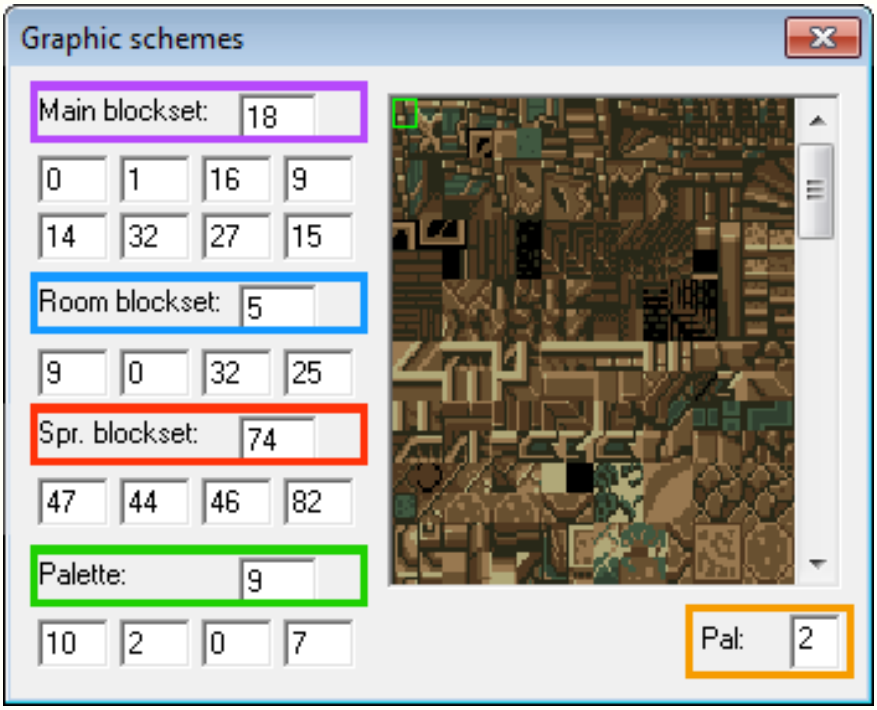
= This is your "Room blockset"

= This is your "Spr. Blockset" *(If you're editing dungeons, like in the current case, remember that EnemyBlk 0 = Spr. blockset 64, EnemyBlk 1 = Spr. blockset 65 etc...)*



= This is your "Palette"

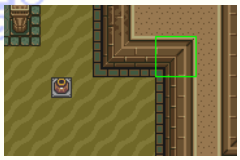
Now let's open another Hyrule Magic executable so we have two of them running at the same time. In your second editor, open the graphic schemes editor and enter the various information we just put an emphasis on, on their correct locations.



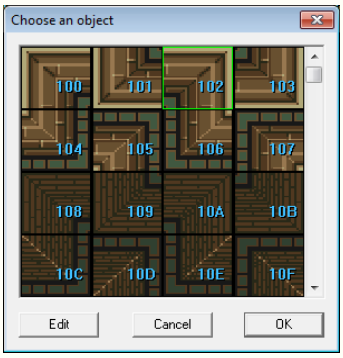
Within the graphic schemes window, enter the same values as your current dungeon room.

- Set your "Main blockset" value to 18;
- Your "Room blockset" value to 5;
- Your "Spr blockset" value to 74 (because EnemyBlk 10 = Spr blockset 74);
- Your "Palette" value to 9;
- And finally in the bottom right corner, set your "Pal" to 2 (we'll use this for later).

Now let's go back to our other Hyrule Magic editor (the one with our dungeon room window).

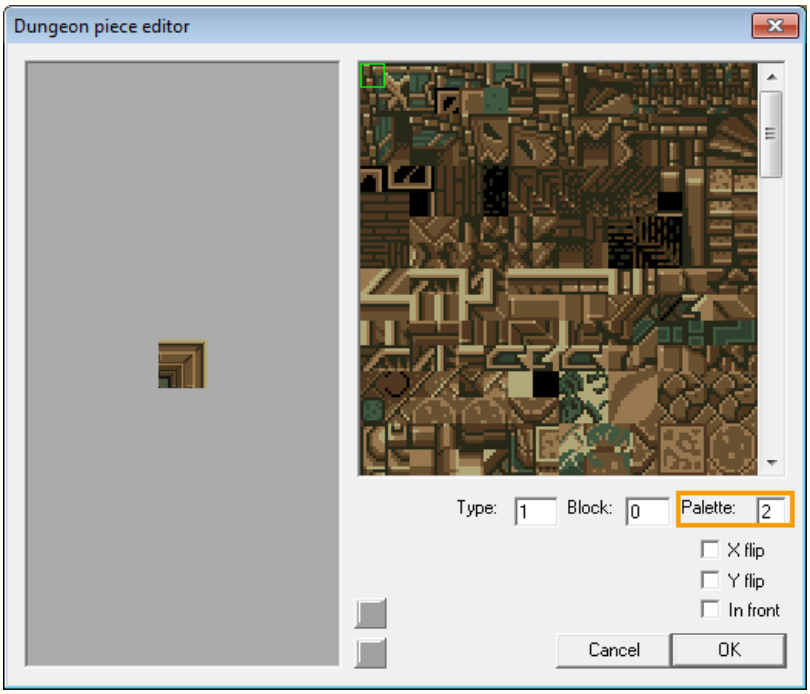


Select a tile (can be any tile that you want) and double click on it.

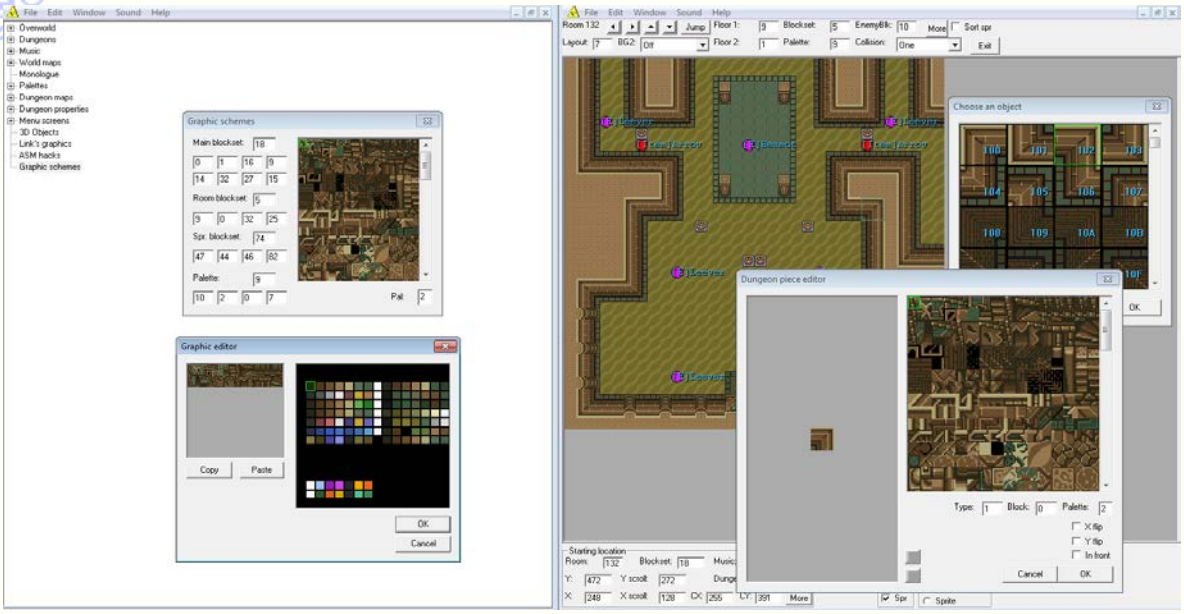


This will open the "Choose an object" window. That window contains all the dungeons objects in the game (theirs graphics are shared through all the dungeons, so be careful when editing them).

But for now, let's just click on the edit button in the bottom left corner. This will open the dungeon piece editor.



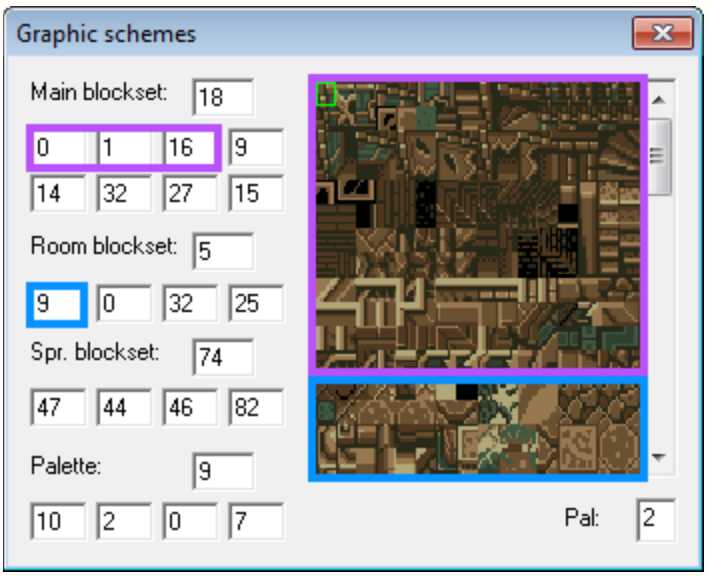
Remember our "Pal" value in the bottom right corner of the graphic schemes window? This is where you write it down. This is in fact the palette for the walls and some objects.
Now both editors should be displaying the same data!



Switch back to your Hyrule Magic editor that contains the graphic schemes window. It's time to delve deeper into the values we haven't touched yet.

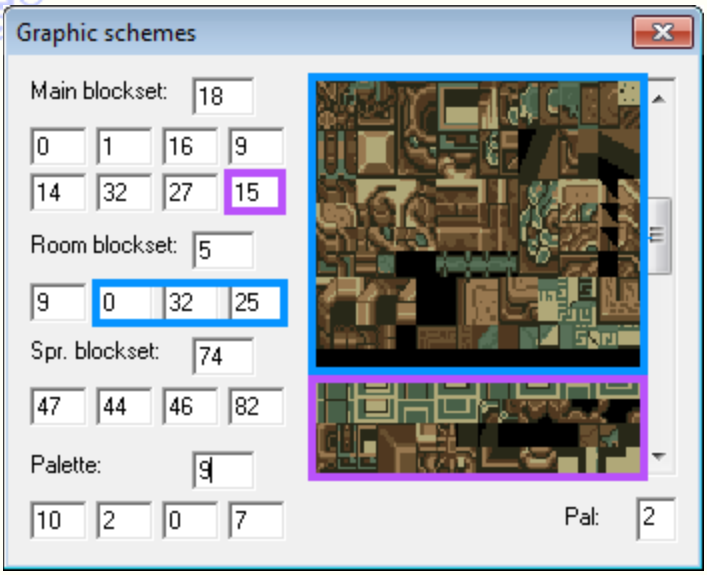
The "Main blockset" and "Room blockset" are intwined with each other. Meaning if you're planning on editing a "Main blockset" then you have to edit your "Room blockset at the same time so that it's as less confusing as posible.

Let's take for exemple a look at our first four tilesets we see in our graphic schemes window:



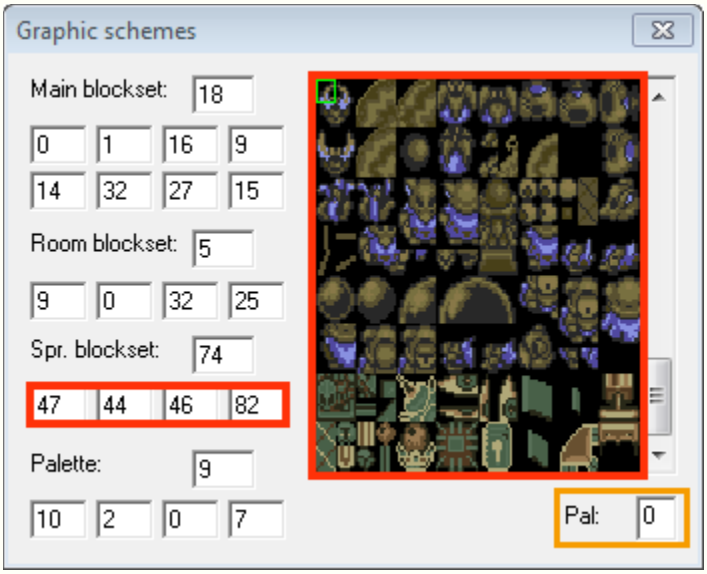
The first three tilesets belong to the "Main blockset", while the 4th one to the "Room blockset".

Now, scroll down your graphic schemes window to the next four tilesets.



The first three tilesets belong to the "Room blockset", while the 4th one to the "Main blockset".
Let's continue our current dungeon room tilesets investigation by scrolling down again.

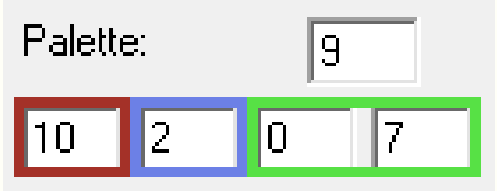
This time we scroll down at the very bottom of the graphic schemes window:



The "Spr blockset" four values allow you to change the various enemies that load in your room. Switch your "Pal" value to 0 to see your spritesets (otherwise with value 2 you will see nothing).

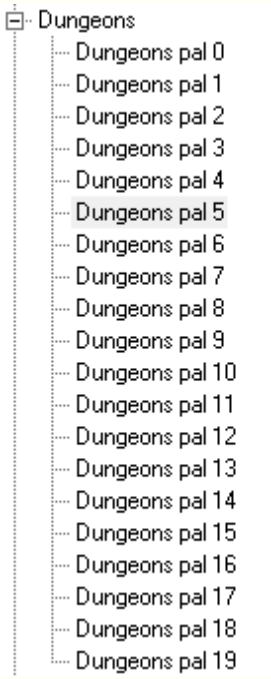
One more setting to describe, then I'll explain each section in details!

Let's now take a look at the "Palette" section. This part is somewhat different then the other settings.



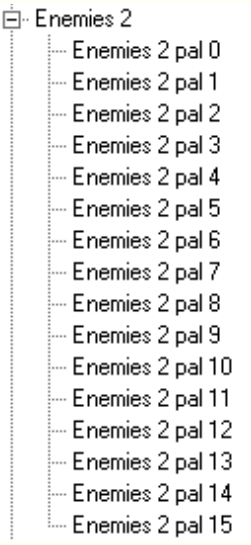
The **first value** determines which dungeon palette your current dungeon tileset use (in our case it's set to 9). If you change the value to something else, you can load another dungeon palette.

Take caution that the palettes aren't in alphabetical order in the dungeons palettes list so even if you change your first "Main Palette" value to for exemple 6 and then go double-click on "Dungeon pal 6" you'll find out that those colors aren't the same ones.

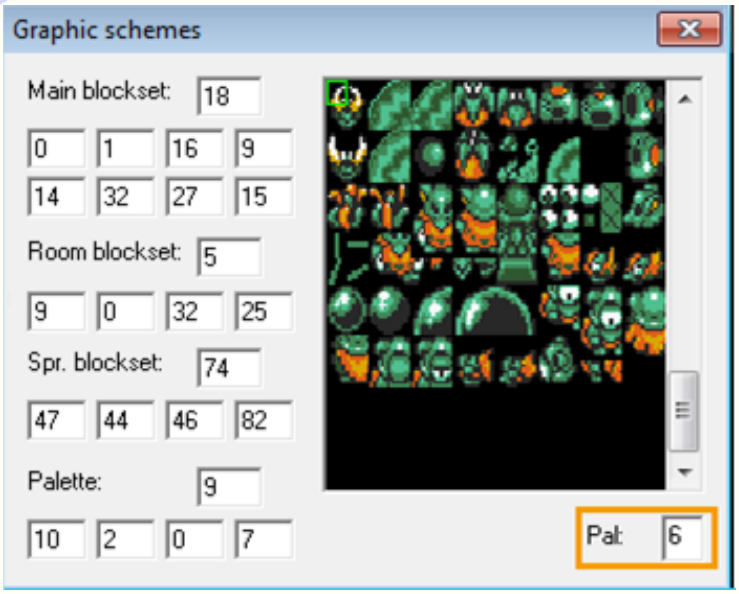


For now let's look at the other palette values.

The **second value** is related to enemies/sprites palettes. In fact you can edit that palette in the "Enemies 2" section of your palette editor in HM. The values themselves range from 0-15.



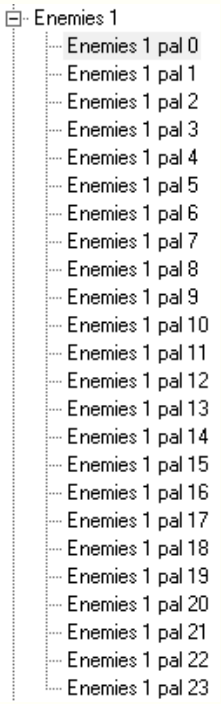
The **third** and **fourth values** are also related to enemies/sprites palettes. You can quickly notice that by changing your "Pal" value to 6:



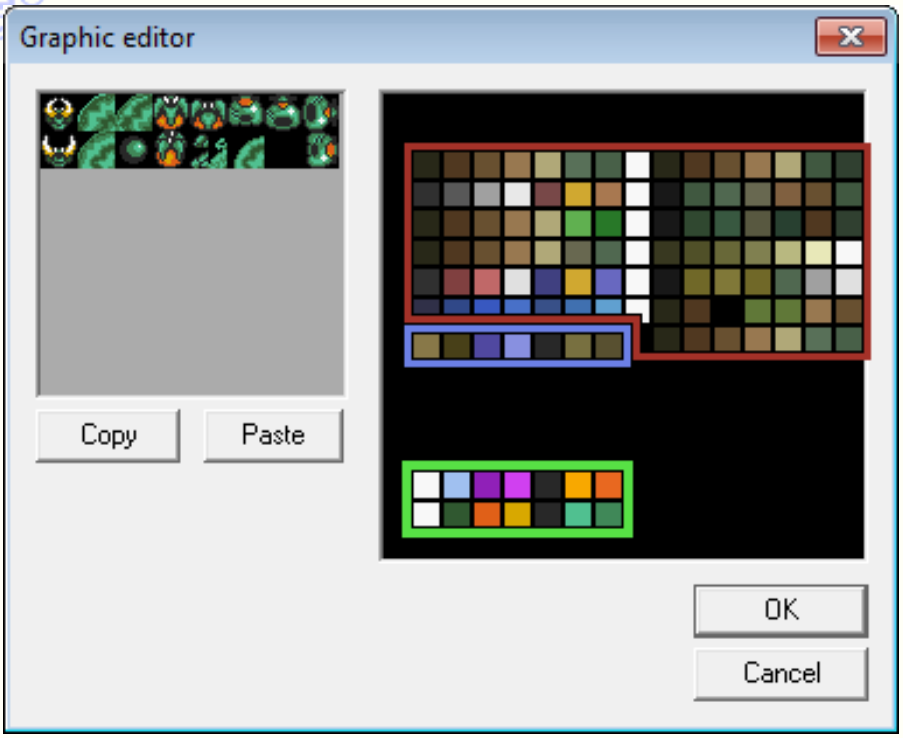
Here you can chose to attribute two different sprites palettes.

You can set you "Pal" to either 5 or 6 to see the actual sprite palettes. While the values themselves range from 0-23. The values are again located in the palettes editor in Hyrule Magic.

"Enemies 1" are what you are looking for this time around:



We can take a quick look at our current palettes in use by double-clicking on any tile of our tilesets/spritesets (in your graphic schemes window) to open up the graphic editor window.



Palette: 9

10 2 0 7

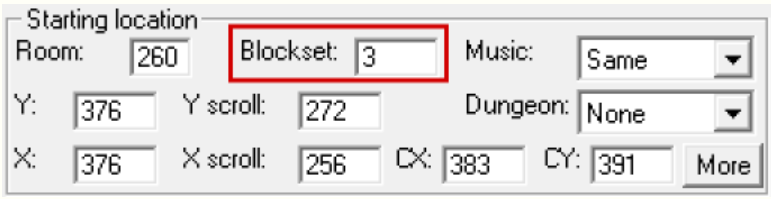
It's time for our final analyst of our dungeon room!

Here's all the tilesets that our dungeon uses in order to be like so:

And that concludes our simple introduction to the graphic schemes! I hope it wasn't too hard, because it's going to get more complicated from now on! Time to learn about each of the parts in details now!

b) Main blockset

The main blockset option edits the dungeon main blocksets. You can edit that data in the dungeon editor window as seen in the screenshot below:

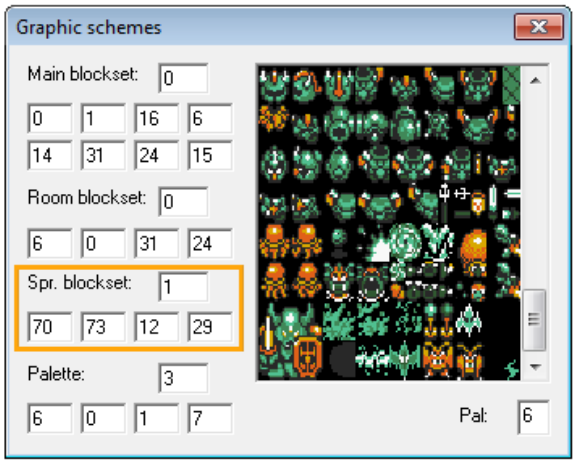


There are 30 of them in total, here they are:

c) Room blockset

goes from 0-23

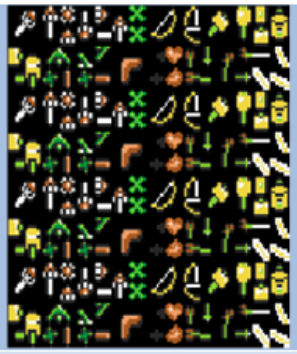
d) Spr. Blockset

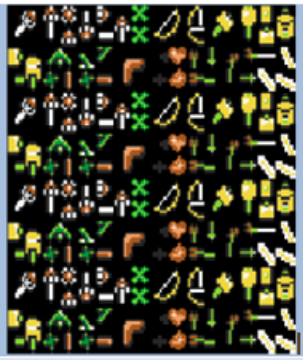


In the graphic schemes menu, the Spr. Blockset lets you reorganize all your spritesets in the way you like!

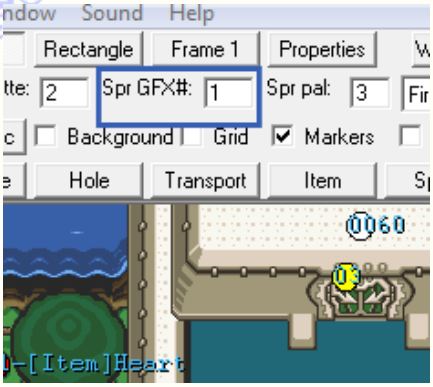
~ *It basically goes from **0-143*** ~

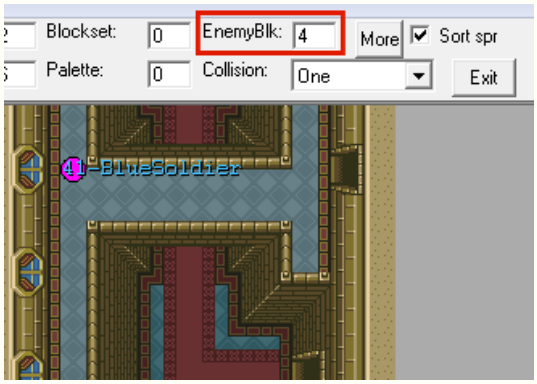
It should be noted that ***freespace*** in the Spr. Blockset can be found by locating those sprite sets:

Name	Spr. blocksets 47-63
Graphics	
1st digit	0: *Freespace*
2nd digit	0: *Freespace*
3rd digit	0: *Freespace*
4th digit	0: *Freespace*
More info	*Much freespace in there for custom overworlds sprite sets.*

Name	Spr. Blocksets 108-124/EnemyBlks 44-60
Graphics	
1st digit	0: *Freespace*
2nd digit	0: *Freespace*
3rd digit	0: *Freespace*
4th digit	0: *Freespace*
More info	*Much freespace in there for custom dungeon sprite sets.*

In **overworlds** you need to edit the **Spr GFX# tag** in the editor main menu
while in the **dungeons** you need to edit the **EnemyBlk tag**





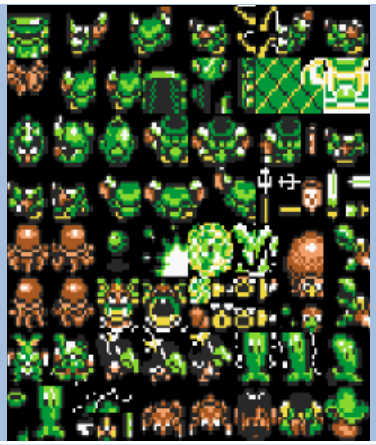
Pictures above: *The **Spr GFX# tag** is the same thing as the **Spr. blockset one** but named differently in the graphic schemes editor (**the dungeons** also use a different name).*

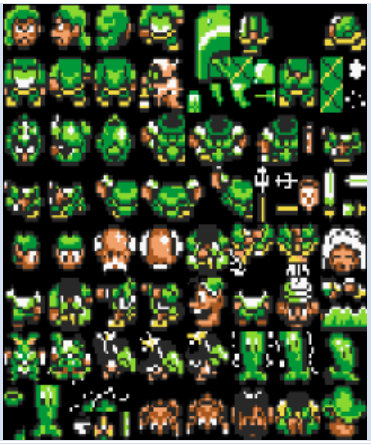
If you want to edit the sprite sets you need to keep that in mind:

Spr. blockset 64 = EnemyBlk 0
Spr. blockset 65 = EnemyBlk 1
Spr. blockset 66 = EnemyBlk 2
Spr. blockset 67 = EnemyBlk 3 ... and so on.

Name	Spr. blockset 0	Spr. blockset 1
Graphics		
1st digit	0: *Freespace*	70: *CannonSoldier, GreenGrassArcher, MorningStar*
2nd digit	73: *Blue/GreenSoldier (Light World forms), RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	0: *Freespace*	12: *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*
4th digit	0: *Freespace*	29: *ArmosKnight (Boss), ElectricBarrier, Person?11,227 (1/2 magic)*
More info		

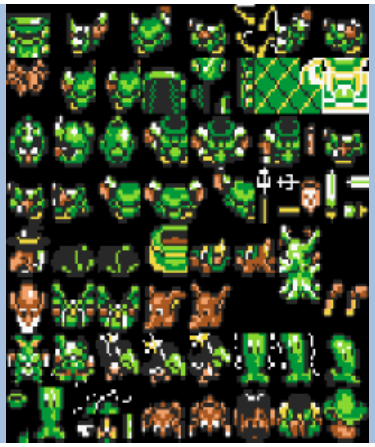
Name	Spr. blockset 2	Spr. blockset 3
Graphics		
1st digit	72: *BlueArcher, PalaceGuard**	70: *CannonSoldier, GreenGrassArcher, MorningStar*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	19: *Knight, SWITCH????*	19: *Knight**
4th digit	29: *ArmosKnight (Boss), ElectricBarrier, Person?11,227 (1/2 magic)*	14: *Poe/Ghini*
More info	* = *The PalaceGuard sprite requires tiles in both the 1st and 2nd sprite sets to show up properly.*	

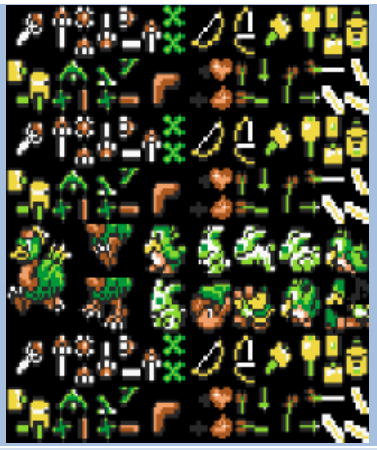
Name	Spr. blockset 4	Spr. blockset 5
Graphics		
1st digit	72: *BlueArcher, PalaceGuard**	72: *BlueArcher, PalaceGuard**
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	12: *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*	12: *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*
4th digit	17: *Bunny, Cucumber, FakeSword, GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	16: *Armos, FallingRocks, Bush/RockCrab, Squirrel, Tektite*
More info		

Name	Spr. blockset 6	Spr. blockset 7
Graphics		
1st digit	79: *AngryBrother, Person?, Runner*	14: *Bully (Light World form)*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	74: *Bottleman, Farmboy, Lumberjacks, OldWoman*	74: *Bottleman, Farmboy, Lumberjacks, OldWoman*
4th digit	80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*	17: *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*
More info		

Name	Spr. blockset 8	Spr. blockset 9
Graphics		
1st digit	70: *CannonSoldier, GreenGrassArcher, MorningStar*	0: *Freespace*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	18: *Geldman, HyliaObstacle, MedallianTablet, Vulture*	0: *Freespace*
4th digit	0: *Freespace*	80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*
More info		

Name	Spr. blockset 10	Spr. blockset 11
Graphics		
1st digit	0: *Freespace*	72: *BlueArcher, PalaceGuard**
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	0: *Freespace*	12: *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*
4th digit	17: *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	0: *Freespace*
More info		

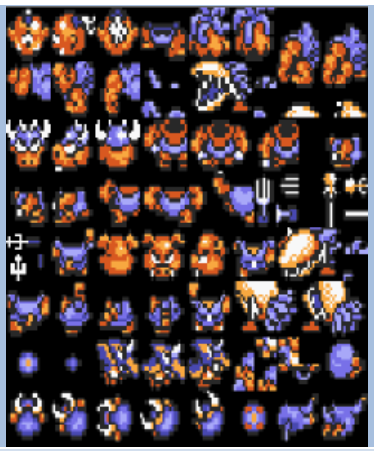
Name	Spr. blockset 12	Spr. blockset 13
Graphics		
1st digit	0: *Freespace*	72: *BlueArcher, PalaceGuard**
2nd digit	0: *Freespace*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	55: *Tentman, MasterSwd**	76: *Fluteboy (Dark World form), Mutant, Witch*
4th digit	54: *Animal? (Both of them), MasterSwd*, Pond (Big fairies)*	17: *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*
More info	* = *The MasterSwd sprite requires tiles in both the 3rd and 4th sprite sets to show up properly.*	

Name	Spr. blockset 14	Spr. blockset 15
Graphics		
1st digit	**93:** *Mantle, Beginning/Ending sequence sprites*	**0:** *Freespace*
2nd digit	**44:** *Beamos, Tentacle*	**0:** *Freespace*
3rd digit	**12:** *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*	**76:** *Fluteboy (Dark World form), Mutant, Witch*
4th digit	**68:** *WalkingZora, ZoraKing*	**0:** *Freespace*
More info		

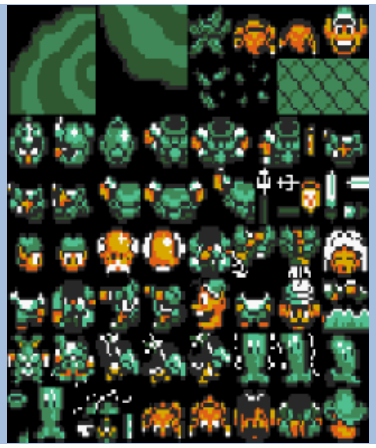
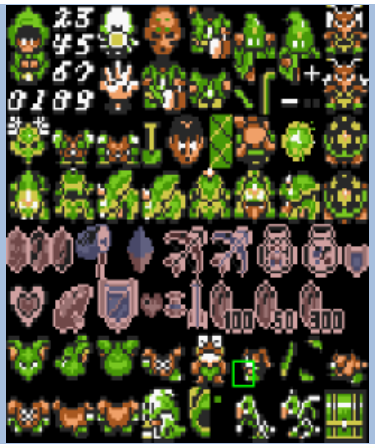
Name	Spr. blockset 16	Spr. blockset 17
Graphics		
1st digit	**15:** *DashItem (Tree hollow in Area 02),* Ending sequence sprite (Lumberjack looking at camera), *Bush/RockCrab **	**0:** *Freespace*
2nd digit	**0:** *Freespace*	**0:** *Freespace*
3rd digit	**18:** *Geldman, HyliaObstacle, MedallianTablet, Vulture*	**0:** *Freespace*
4th digit	**16:** *Armos, FallingRocks, Bush/RockCrab*, Squirrel, Tektite*	**76:** *Freespace (All sprites are using the wrong digit, they are supposed to be using the 3rd digit)*
More info	* = *The Bush/RockCrab sprite requires tiles in both the 1st and 4th sprite sets to show up properly.*	

Name	Spr. blockset 18	Spr. blockset 19
Graphics		
1st digit	0: *Freespace*	22: *Hinox, HoppingBulbPlant*
2nd digit	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*
3rd digit	23: *PigSpearman*	23: *PigSpearman*
4th digit	0: *Freespace*	27: *Blue/GreenAirBomber, LikeLike, LiveTree*
More info		

Name	Spr. blockset 20	Spr. blockset 21
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	21: *Bully (Dark World form)*
2nd digit	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*
3rd digit	23: *PigSpearman*	23: *PigSpearman*
4th digit	20: *Bully&Whimp, Lynel*	21: *Chicken (DW Form)*
More info		

Name	Spr. blockset 22	Spr. blockset 23
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	22: *Hinox, HoppingBulbPlant*
2nd digit	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*
3rd digit	24: *4WayOctorok/FireballZora/Octorok (Dark World forms), Item (Quake medallion fish)*	23: *PigSpearman*
4th digit	25: *Raven (DW Form), Swampsnake*	25: *Raven (DW Form), Swampsnake*
More info		

Name	Spr. blockset 24	Spr. blockset 25
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	22: *Hinox, HoppingBulbPlant*
2nd digit	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*
3rd digit	0: *Freespace*	24: *4WayOctorok/FireballZora/Octorok (Dark World forms), Item (Quake medallion fish)*
4th digit	0: *Freespace*	27: *Blue/GreenAirBomber, LikeLike*
More info		

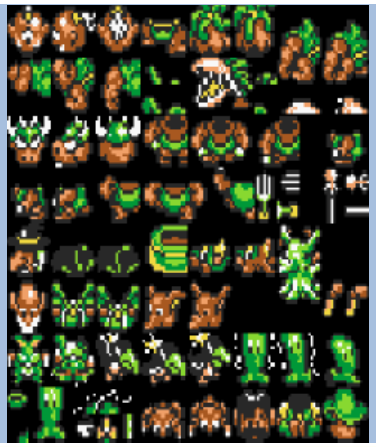
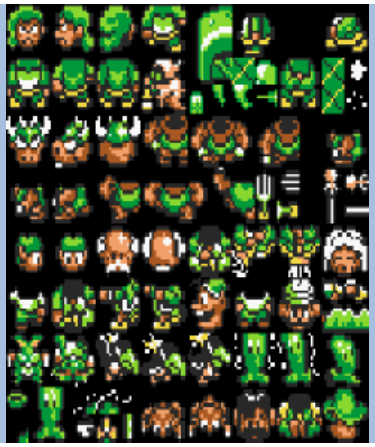
Name	Spr. blockset 26	Spr. blockset 27
Graphics		
1st digit	15:	75: *ArcherGame*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	42: *DiggingGameGuy , Fireblade, Lanmola, Shell, Triceritops, WallBubble*
3rd digit	74: *Bottleman, Farmboy, Lumberjacks, ?DustGirl?, Thief*	92:
4th digit	17: *Bunny, Cucumber , FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	21: *Chicken (DW Form)*
More info		

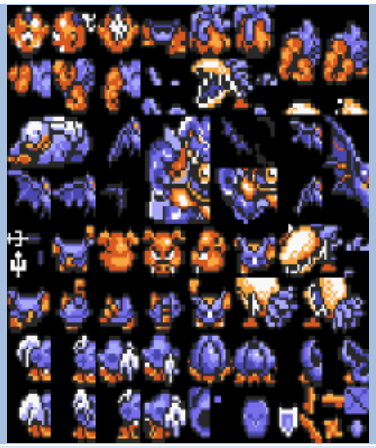
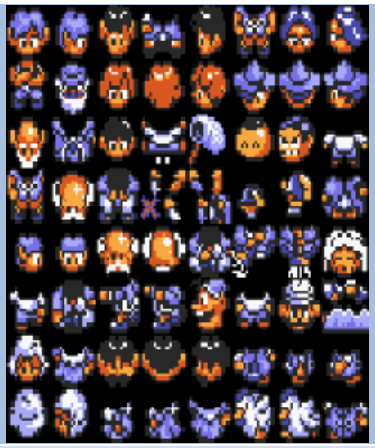
Name	Spr. blockset 28	Spr. blockset 29
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	0: *Freespace*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	0: *Freespace*
3rd digit	23: *PigSpearman*	0: *Freespace*
4th digit	29: *ArmosKnight (Boss), ElectricBarrier, Person?11,227 (1/2 magic)*	21: *Chicken (DW Form)*
More info		

Name	Spr. blockset 30	Spr. blockset 31
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	22: *Hinox, HoppingBulbPlant*
2nd digit	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	23: *PigSpearman*	18: *Geldman, HyliaObstacle, MedallianTablet, Vulture*
4th digit	16: *Armos, FallingRocks, Bush/RockCrab, Squirrel, Tektite*	0: *Freespace*
More info		

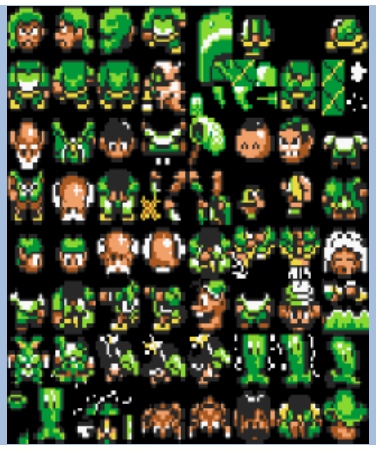
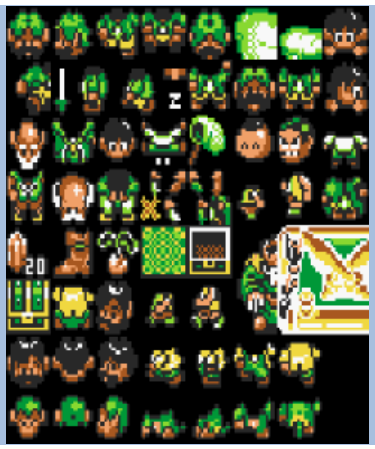
Name	Spr. blockset 32	Spr. blockset 33
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	0: *Freespace*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	0: *Freespace*
3rd digit	12: *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*	18: *Geldman, HyliaObstacle, MedallianTablet, Vulture*
4th digit	17: *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	16: *Armos, FallingRocks, Bush/RockCrab, Squirrel, Tektite*
More info		

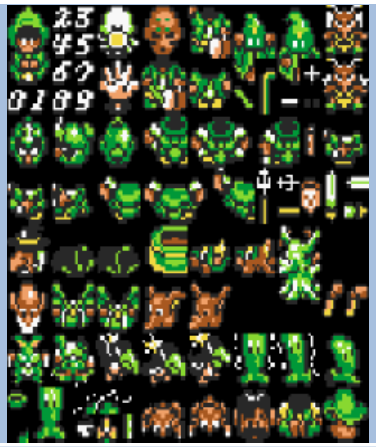
Name	Spr. blockset 34	Spr. blockset 35
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	22: *Hinox, HoppingBulbPlant*
2nd digit	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	0: *Freespace*	12: *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*
4th digit	17: *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	0: *Freespace*
More info		

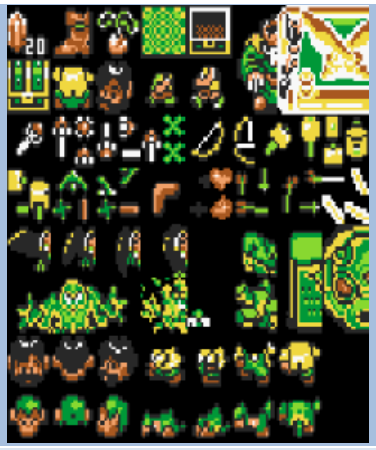
Name	Spr. blockset 36	Spr. blockset 37
Graphics		
1st digit	22: *Hinox, HoppingBulbPlant*	14: *Bully (Light World form)*
2nd digit	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*	13: *Blue/GreenSoldier (Dark World forms), HogSpearMan*
3rd digit	76: *Fluteboy (Dark World form), Mutant, Witch*	74: *Bottleman, Farmboy, Lumberjacks, ?DustGirl?, Thief*
4th digit	17: *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	17: *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*
More info		

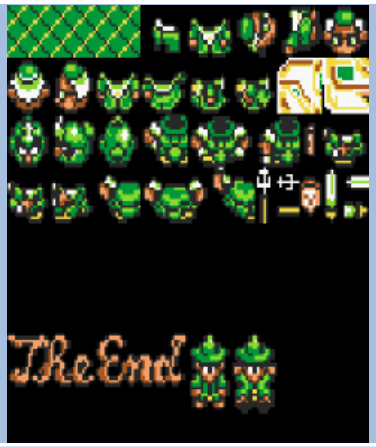
Name	Spr. blockset 38	Spr. blockset 39
Graphics		
1st digit	**22:** *Hinox, HoppingBulbPlant*	**79:** *AngryBrother, Person?, Runner*
2nd digit	**26:** *Agahnim (Death animation)*	**52:** Ending sprites
3rd digit	**23:** *PigSpearman*	**74:** *Bottleman, Farmboy, Lumberjacks, ?DustGirl?, Thief*
4th digit	**27:** *Blue/GreenAirBomber, LikeLike*	**80:** *Chicken (LW Form), Hedgeman, ScaredGirl1/2*
More info		

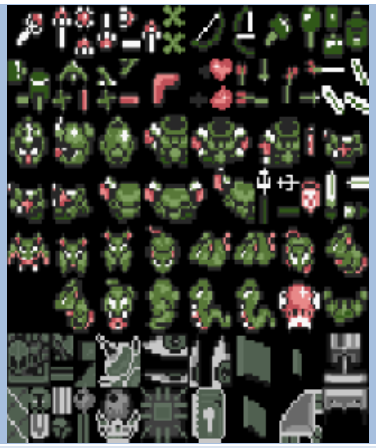
Name	Spr. blockset 40	Spr. blockset 41
Graphics		
1st digit	**53:** Ending sprites	**74:**
2nd digit	**77:**	**52:** Ending sprites
3rd digit	**101:** *Blindman, OldMan (followers)*	**78:** *Fluteboy (Light World form), Ostrich, Rabbit, Uglybird*
4th digit	**54:** *Animal (Both of them), MasterSwd, Pond (Big Fairies)*	**0:** *Freespace*
More info		

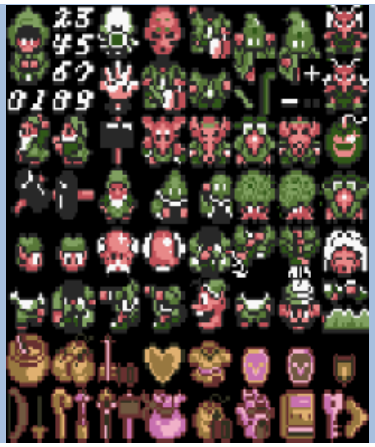
Name	Spr. blockset 42	Spr. blockset 43
Graphics		
1st digit	**14:** *Bully (Light World form)*	**81:** Priest/Uncle (Uncle), SickBoy
2nd digit	**52:** Ending sprites	**52:** Ending sprites
3rd digit	**74:** *Bottleman, Farmboy, Lumberjacks, ?DustGirl?, Thief*	**93:** Intro/Ending sprites, Mantle
4th digit	**17:** *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	**89:** GuyByTheSign, Kiki (followers)
More info		

Name	Spr. blockset 44	Spr. blockset 45
Graphics		
1st digit	**75:** *ArcherGame*	**45:** Ending sprites
2nd digit	**73:** *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	**0:** *Freespace*
3rd digit	**76:** *Fluteboy (Dark World form), Mutant, Witch*	**0:** *Freespace*
4th digit	**17:** *Bunny, Cucumber, FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*	**0:** *Freespace*
More info		

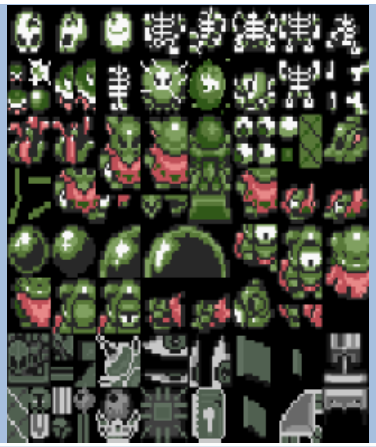
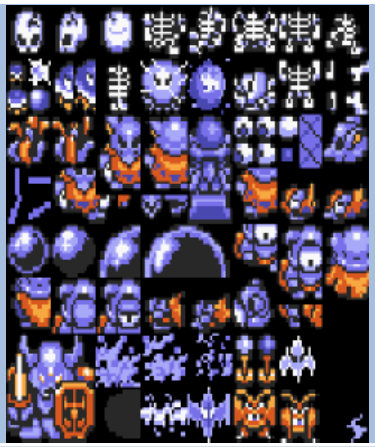
Name	Spr. blockset 46	Spr. blocksets 47-63
Graphics		
1st digit	**93:** Intro/Ending sprites, Mantle	**0:** *Freespace*
2nd digit	**0:** *Freespace*	**0:** *Freespace*
3rd digit	**18:** *Geldman, HyliaObstacle, MedallianTablet, Vulture*	**0:** *Freespace*
4th digit	**89:** GuyByTheSign, Kiki (followers)	**0:** *Freespace*
More info		*Much freespace in there for custom overworlds sprite sets.*

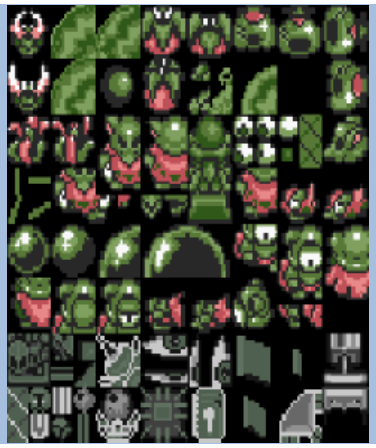
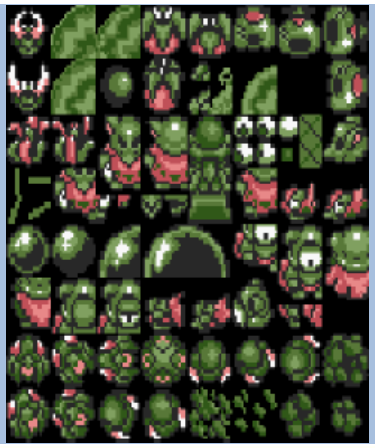
Name	Spr. Blockset 64/EnemyBlk 0	Spr. Blockset 64/EnemyBlk 1
Graphics		
1st digit	**71:** Uncle/Priest (Priest)	**70:** *CannonSoldier, GreenGrassArcher, MorningStar*
2nd digit	**73:** *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	**73:** *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	**43:** Empty?	**28:** *Keese/Rat/Rope (Light World forms)*
4th digit	**45:** Ending sprites	**82:**
More info	*Dungeons sprite sets begin here.*	

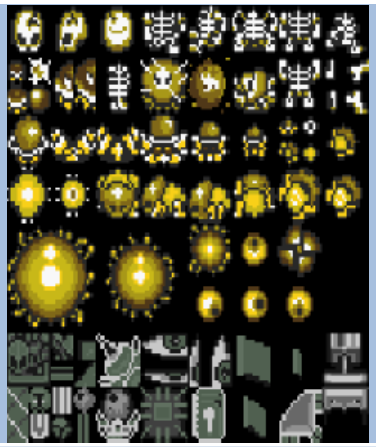
Name	Spr. Blockset 66/EnemyBlk 2	Spr. Blockset 67/EnemyBlk 3
Graphics		
1st digit	0: *Freespace*	93:
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	28: *Keese/Rat/Rope (Light World forms)*	0: *Freespace*
4th digit	82:	82:
More info		

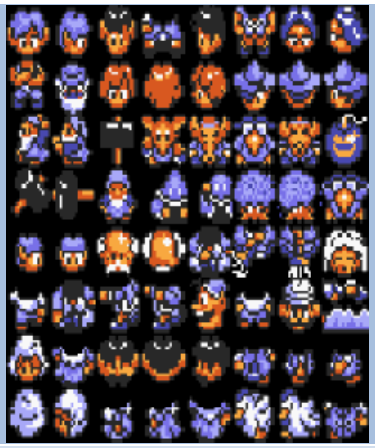
Name	Spr. Blockset 68/EnemyBlk 4	Spr. Blockset 69/EnemyBlk 5
Graphics		
1st digit	70: *CannonSoldier, GreenGrassArcher, MorningStar*	75: *ArcherGame*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	77:
3rd digit	19: *Knight*	74: *Bottleman, Farmboy, Lumberjacks, OldWoman*
4th digit	82:	90:
More info		

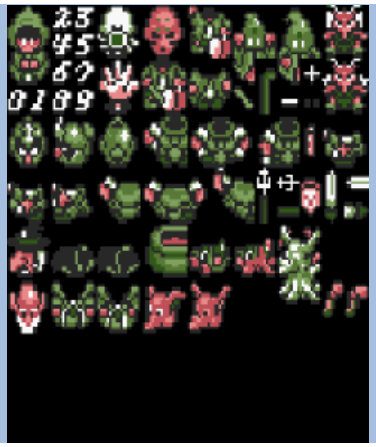
Name	Spr. Blockset 70/EnemyBlk 6	Spr. Blockset 71/EnemyBlk 7
Graphics		
1st digit	71:	75: *ArcherGame*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	77:
3rd digit	28: *Keese/Rat/Rope (Light World forms)*	57: *Arrghus (Boss), BigFairy*
4th digit	82:	54: *Animal, MasterSwd*
More info		

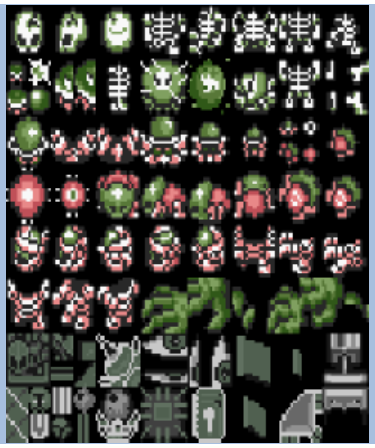
Name	Spr. Blockset 72/EnemyBlk 8	Spr. Blockset 73/EnemyBlk 9
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	44: *Beamos, Tentacle*	44: *Beamos, Tentacle*
3rd digit	46:	46:
4th digit	82:	29: *ArmosKnight (Boss), ElectricBarrier, Person?11,227 (1/2 magic)*
More info		

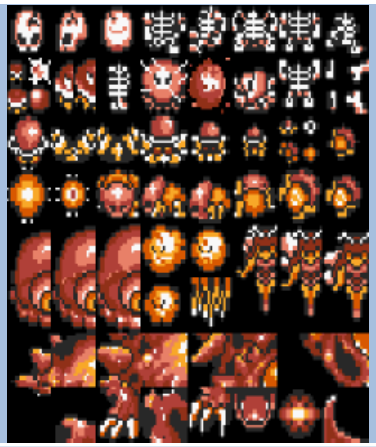
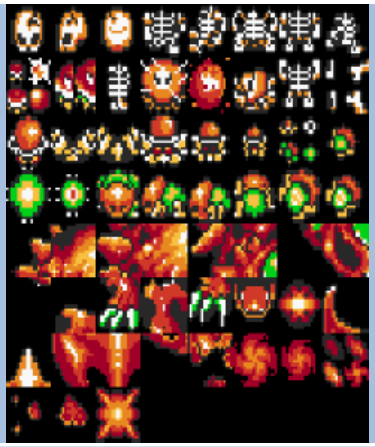
Name	Spr. Blockset 74/EnemyBlk 10	Spr. Blockset 75/EnemyBlk 11
Graphics		
1st digit	47: *CannonUp/Down/Left/Right, Leever, SandCrab*	47: *CannonUp/Down/Left/Right, Leever, SandCrab*
2nd digit	44: *Beamos, Tentacle*	44: *Beamos, Tentacle*
3rd digit	46:	46:
4th digit	82:	49: *Lanmolas (Boss)*
More info		

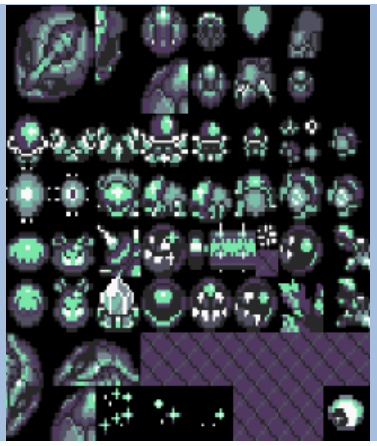
Name	Spr. Blockset 76/EnemyBlk 12	Spr. Blockset 77/EnemyBlk 13
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	81:
2nd digit	30: *BlueOrb, MiniHelmasaur, Moldorm*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	48: *Mouldrum (Boss)*	19: *Knight*
4th digit	82:	0: *Freespace*
More info		

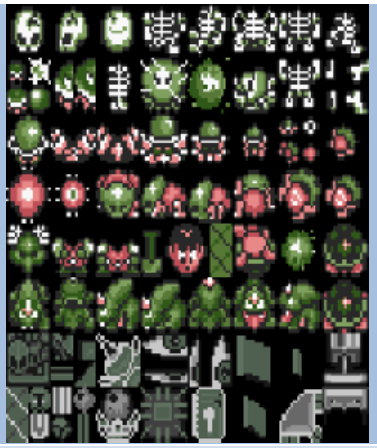
Name	Spr. Blockset 78/EnemyBlk 14	Spr. Blockset 79/EnemyBlk 15
Graphics		
1st digit	79: *AngryBrother, Person?, Runner*	79: *AngryBrother, Person?, Runner*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	77:
3rd digit	19: *Knight*	74: *Bottleman, Farmboy, Lumberjacks, OldWoman*
4th digit	80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*	80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*
More info		

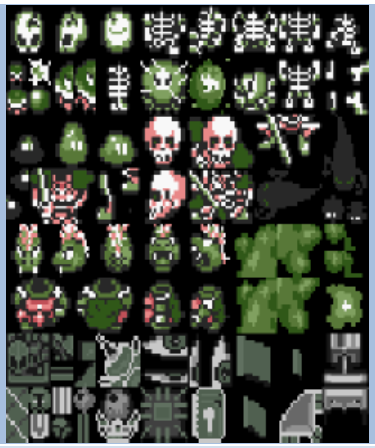
Name	Spr. Blockset 80/EnemyBlk 16	Spr. Blockset 81/EnemyBlk 17
Graphics		
1st digit	75: *ArcherGame*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	32: *Blob, StalfosKnight, VRat*
3rd digit	76: *Fluteboy (Dark World form), Mutant, Witch*	34: *Splash, WaterBug*
4th digit	43:	83:
More info		

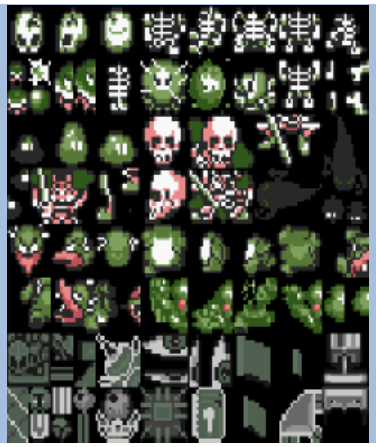
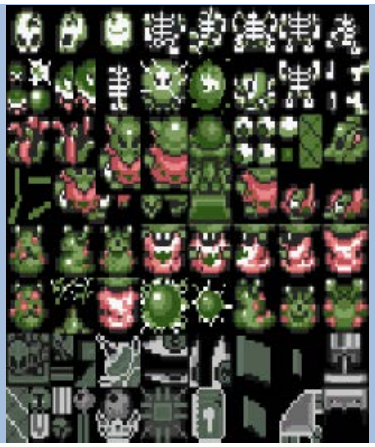
Name	Spr. Blockset 82/EnemyBlk 18	Spr. Blockset 83/EnemyBlk 19
Graphics		
1st digit	85:	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	61:	30: *BlueOrb, MiniHelmasaur, Moldorm*
3rd digit	66:	35: *Gibdo, WallMaster*
4th digit	67:	82:
More info		

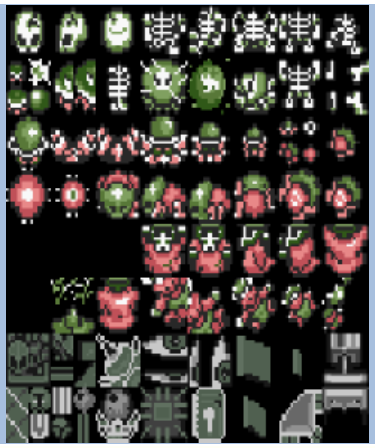
Name	Spr. Blockset 84/EnemyBlk 20	Spr. Blockset 85/EnemyBlk 21
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	30: *BlueOrb, MiniHelmasaur, Moldorm*	30: *BlueOrb, MiniHelmasaur, Moldorm*
3rd digit	57: *Arrghus/ArrghusFuzz (Boss), BigFairy*	58: *Helmasaur (Boss)**
4th digit	58:	62: *Helmasaur (Boss)**
More info		* = The Helmasaur (Boss) sprite requires tiles in both the 3rd and 4th sprite sets to show up properly.

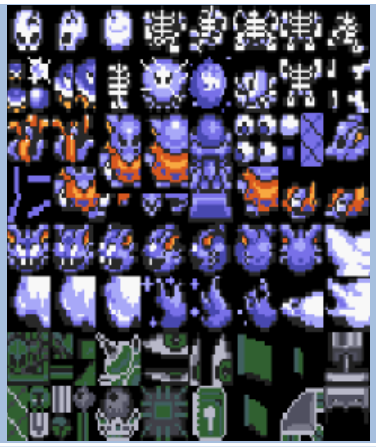
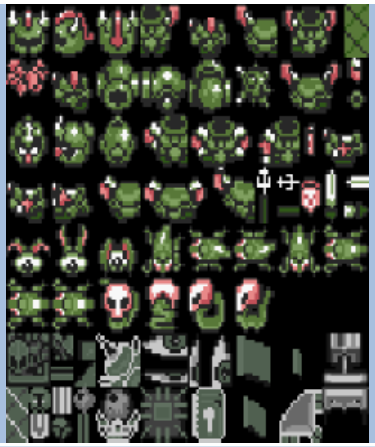
Name	Spr. Blockset 86/EnemyBlk 22	Spr. Blockset 87/EnemyBlk 23
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	64: *TriNexx1/2/3 (Boss)**
2nd digit	30: *BlueOrb, MiniHelmasaur, Moldorm*	30: *BlueOrb, MiniHelmasaur, Moldorm*
3rd digit	60: *Kholdstare (Boss)*	39: *Chomp, FuzzyStack, Roller1/2/3/4, pegswitch?, platform for the red cane?*
4th digit	61: *Viterous (Boss)*	63: *TriNexx1/2/3 (Boss)**
More info		* = The TriNexx1/2/3 (Boss) sprite requires tiles in both the 1st and 4th sprite sets to show up properly. NOTE: ----redo picture^^

Name	Spr. Blockset 88/EnemyBlk 24	Spr. Blockset 89/EnemyBlk 25
Graphics		
1st digit	85:	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	26: *Agahnim (Death)*	30: *BlueOrb, MiniHelmasaur, Moldorm*
3rd digit	66:	42:
4th digit	67:	82:
More info		

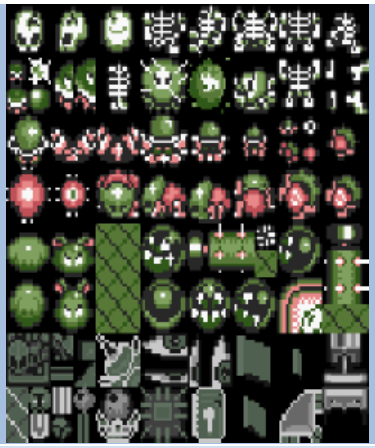
Name	Spr. Blockset 90/EnemyBlk 26	Spr. Blockset 91/EnemyBlk 27
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	30: *BlueOrb, MiniHelmasaur, Moldorm*	32: *Blob, StalfosKnight, VRat*
3rd digit	56: *Mothula (Boss)*	40: *Green/RedLizard, Half-Bubble*
4th digit	82:	82:
More info		

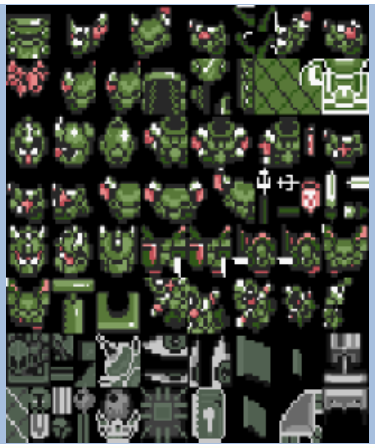
Name	Spr. Blockset 92/EnemyBlk 28	Spr. Blockset 93/EnemyBlk 29
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	32: *Blob, StalfosKnight, VRat*	44: *Beamos, Tentacle*
3rd digit	38: *Iceman, Penguin*	37: *Bombslug, Wizzrobe (Light World form)*
4th digit	82:	82:
More info		

Name	Spr. Blockset 94/EnemyBlk 30	Spr. Blockset 95/EnemyBlk 31
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	32: *Blob, StalfosKnight, VRat*	30: *BlueOrb, MiniHelmasaur, Moldorm*
3rd digit	39: *Chomp, FuzzyStack, Roller1/2/3/4*	41: *Wizzrobe (Dark World form)*
4th digit	82:	82:
More info		

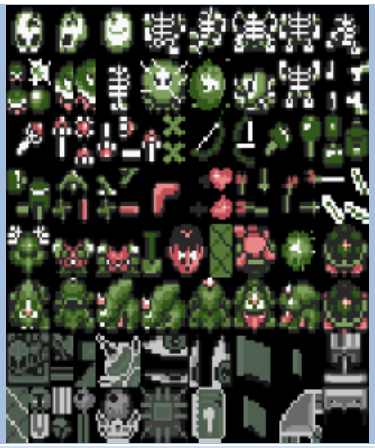
Name	Spr. Blockset 96/EnemyBlk 32	Spr. Blockset 97/EnemyBlk 33
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos, LANMOLA*	70: *CannonSoldier, GreenGrassArcher, MorningStar*
2nd digit	44: *Beamos, Tentacle*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	59: *Blind (Boss)*	36: *Keese/Rat/Rope (Dark World forms)*
4th digit	82: SpikeBlock, floordrops, switches, RedOrb, BigSpikeBlock, Bubble, 4Bubbles, EyeLasers	82:
More info		

Name	Spr. Blockset 98/EnemyBlk 34	Spr. Blockset 99/EnemyBlk 35
Graphics		
1st digit	33: *Ganon*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	65: *Ganon*	44: *Beamos, Tentacle*
3rd digit	69: *Ganon*	40: *Green/RedLizard, Half-Bubble*
4th digit	51: *Ganon*	49: *Lanmolas (Boss)*
More info	*All four sprite sets are used by Ganon (final boss in the game).*	

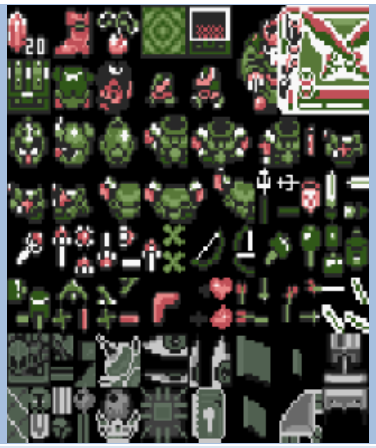
Name	Spr. Blockset 100/EnemyBlk 36	Spr. Blockset 101/EnemyBlk 37
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	13: *Blue/GreenSoldier (Dark World forms)*	30: *BlueOrb, MiniHelmasaur, Moldorm*
3rd digit	41: *Wizzrobe (Dark World form)*	39: *Chomp, FuzzyStack, Roller1/2/3/4*
4th digit	82:	82:
More info		

Name	Spr. Blockset 102/EnemyBlk 38	Spr. Blockset 103/EnemyBlk 39
Graphics		
1st digit	31: *BlueMiri, RedMiri, Stalfos*	72: *BlueArcher, PalaceGuard**
2nd digit	32: *Blob, StalfosKnight, VRat*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	39: *Chomp, FuzzyStack, Roller1/2/3/4*	19: *Knight*
4th digit	83:	82:
More info		

Name	Spr. Blockset 104/EnemyBlk 40	Spr. Blockset 105/EnemyBlk 41
Graphics		
1st digit	14: *Bully (Light World form)*	31: *BlueMiri, RedMiri, Stalfos*
2nd digit	30: *BlueOrb, MiniHelmasaur, Moldorm*	32: *Blob, StalfosKnight, VRat*
3rd digit	74: *Bottleman, Farmboy, Lumberjacks, OldWoman*	38: *Iceman, Penguin*
4th digit	80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*	83:
More info		

Name	Spr. Blockset 106/EnemyBlk 42	Spr. Blockset 107/EnemyBlk 43
Graphics		
1st digit	21: *Bully (Dark World form)*	31: *BlueMiri*, *RedMiri*, *Stalfos*
2nd digit	0: *Freespace*	0: *Freespace*
3rd digit	0: *Freespace*	42:
4th digit	0: *Freespace*	82:
More info		

Name	Spr. Blocksets 108-124/EnemyBlks 44-60	Spr. Blockset 125/EnemyBlk 61
Graphics		
1st digit	0: *Freespace*	50:
2nd digit	0: *Freespace*	0: *Freespace*
3rd digit	0: *Freespace*	0: *Freespace*
4th digit	0: *Freespace*	8:
More info	*Much freespace in there for custom dungeon sprite sets.*	

Name	Spr. Blockset 126/EnemyBlk 62	Spr. Blockset 127/EnemyBlk 63
Graphics		
1st digit	93:	85:
2nd digit	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*	73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
3rd digit	0: *Freespace*	66:
4th digit	82:	67:
More info		

Name	Spr. Blocksets 128/EnemyBlks 64	Spr. Blockset 129/EnemyBlk 65
Graphics		
1st digit	97:	97:
2nd digit	98:	98:
3rd digit	99:	99:
4th digit	80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*	80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*
More info		

Name	Spr. Blockset 130/EnemyBlk 66	Spr. Blockset 131/EnemyBlk 67
Graphics		
1st digit	97:	97:
2nd digit	98:	98:
3rd digit	99:	99:
4th digit	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*
More info		

Name	Spr. Blocksets 132/EnemyBlk 68	Spr. Blockset 133/EnemyBlk 69
Graphics		
1st digit	97:	97:
2nd digit	98:	98:
3rd digit	99:	99:
4th digit	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*
More info		

Name	Spr. Blockset 134/EnemyBlk 70	Spr. Blockset 135/EnemyBlk 71
Graphics		
1st digit	97:	97:
2nd digit	86:	98:
3rd digit	87:	99:
4th digit	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*
More info		

Name	Spr. Blocksets 136/EnemyBlk 72	Spr. Blockset 137/EnemyBlk 73
Graphics		
1st digit	97:	97:
2nd digit	98:	86:
3rd digit	99:	87:
4th digit	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*
More info		

Name	Spr. Blockset 138/EnemyBlk 74	Spr. Blockset 139/EnemyBlk 75
Graphics		
1st digit	97:	97:
2nd digit	86:	86:
3rd digit	99:	87:
4th digit	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*
More info		

Name	Spr. Blocksets 140/EnemyBlk 76	Spr. Blockset 141/EnemyBlk 77
Graphics		
1st digit	97:	97:
2nd digit	86:	86:
3rd digit	51:	87:
4th digit	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*
More info		

Name	Spr. Blocksets 142/EnemyBlk 78	Spr. Blockset 143/EnemyBlk 79
Graphics		
1st digit	97:	97:
2nd digit	98:	98:
3rd digit	99:	99:
4th digit	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*	80: *Chicken (LW Form)*, *Hedgeman*, *ScaredGirl1/2*
More info		

e) How to create your own sprite sets

As you may have noticed already, the Spr. Blockset in broken down into four digits. In order to create your very own sprite blocksets, you need to choose which sprites you want to have in each of the digits. Here's a database of all the sprites in the game along with the digit(s) they use:

1st digit:
14: *Bully (Light World form)*
15:
21: *Bully (Dark World form)*
22: *Hinox*, *HoppingBulbPlant*
31: *BlueMiri*, *RedMiri*, *Stalfos*
45:
47: *CannonUp/Down/Left/Right*, *Leever*, *SandCrab*
50:
53:
64:
70: *CannonSoldier*, *GreenGrassArcher* , *MorningStar*
71:
72: *BlueArcher*, *PalaceGuard**
74
75: *ArcherGame*
79: *AngryBrother*, *Person?*, *Runner*
81:
85:
93:

97: dungeon maps related

2nd digit:

13: *Blue/GreenSoldier (Dark World forms)*
26: *Agahnim (Death)*
30: *BlueOrb, MiniHelmasaur, Moldorm*
32: *Blob, StalfosKnight, VRat*
42: *DiggingGameGuy , Fireblade, Lanmola, Shell, Triceritops, WallBubble*
44: *Beamos, Tentacle*
52:
61:
73: *Blue/GreenSoldier (Light World forms), PalaceGuard*, RedSpearKnight*
77:
??: *HogSpearman*
86: dungeon maps related???????????
98: dungeon maps related

3rd digit:

12: *4WayOctorok/FireballZora/Octorok (Light World forms), Crab, Octoblimp*
18: *Geldman, HyliaObstacle, MedallianTablet, Vulture*
19: *Knight*
23: *PigSpearman*
24: *4WayOctorok/FireballZora/Octorok (Dark World forms), Item (Quake medallion fish)*
28: *Keese/Rat/Rope (Light World forms)*
34: *Splash, WaterBug*
35: *Gibdo, WallMaster*
36: *Keese/Rat/Rope (Dark World forms)*
37: *Bombslug, Wizzrobe (Light World form)*
38: *Iceman, Penguin*
39: *Chomp, FuzzyStack, Roller1/2/3/4*
40: *Green/RedLizard, Half-Bubble*
41: *Wizzrobe (Dark World form)*
42:
43:
46:
48: *Mouldrum (Boss)*
51:
55: *Tentman*
56: *Mothula (Boss)*
57: *Arrghus (Boss), BigFairy*
58:
59:
60: *Kholdstare (Boss)*
66:
74: *Bottleman, Farmboy, Lumberjacks, OldWoman*
76: *Fluteboy (Dark World form), Mutant, Witch*
78: *Fluteboy (Light World form), Ostrich, Rabbit, Uglybird*
87: dungeon maps related???????????
92:
93:

99: dungeon maps related
101:

4th digit:
8:
14: *Poe/Ghini*
16: *Armos, FallingRocks, Bush/RockCrab, Squirrel, Tektite*
17: *Bunny, Cucumber , FakeSword , GuyByTheSign, Mushroom, Raven (LW Form), Bush/RockCrab*
20: *Bully&Whimp, Lynel*
21: *Chicken (DW Form)*
25: *Raven (DW Form), Swampsnake*
27: *Blue/GreenAirBomber, LikeLike*
29: *ArmosKnight (Boss), ElectricBarrier, Person?11,227 (1/2 magic)*
43:
45:
49: *Lanmolas (Boss)*
54: *Animal, MasterSwd*
58:
61: *Viterous (Boss)*
62:
63:
67:
68: *WalkingZora, ZoraKing*
76:
80: *Chicken (LW Form), Hedgeman, ScaredGirl1/2*
82:
83:
89:
90:
??: *EyeLaser 82???*
??: RedOrb
??: Statue 82???
??: SpikeBlock 82??
TrapSwitch 82???

Sprites that use multiple digits:

22, 24 (1st and 3rd digits): *SnapDragon*
44, 46 (2nd and 3rd digits): *Green/RedRocklops*
58, 62 (3rd and 4th digits): *HelmasaurKing (Boss)*
63, 64 (1st and 4th digits): *Trinexx (Boss)*
66, 67, ??, ?? (??? and ??? digits): *Agahnim (Boss)*
33,51,65,69 (All four digits): *Ganon (Boss)*

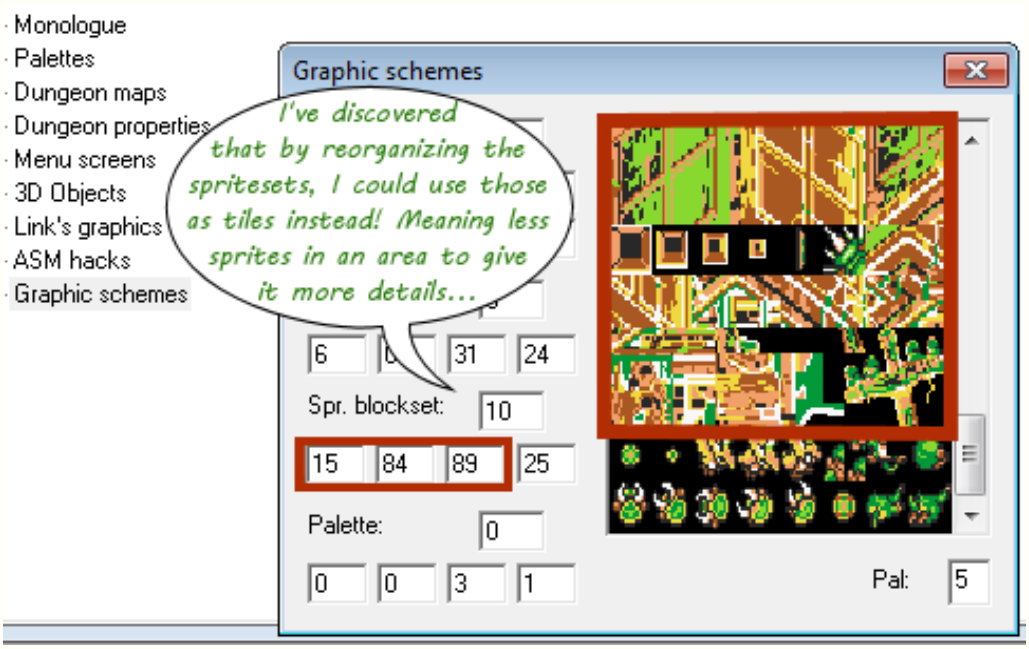
Sprites to sort out:

- *Blind (Boss) [59] ---unknown digit*
- *Priest [71] ---unknown digit*
- *Uncle [81] ---unknown digit*
- *SickBoy [81] ---unknown digit*
- Shooter ??

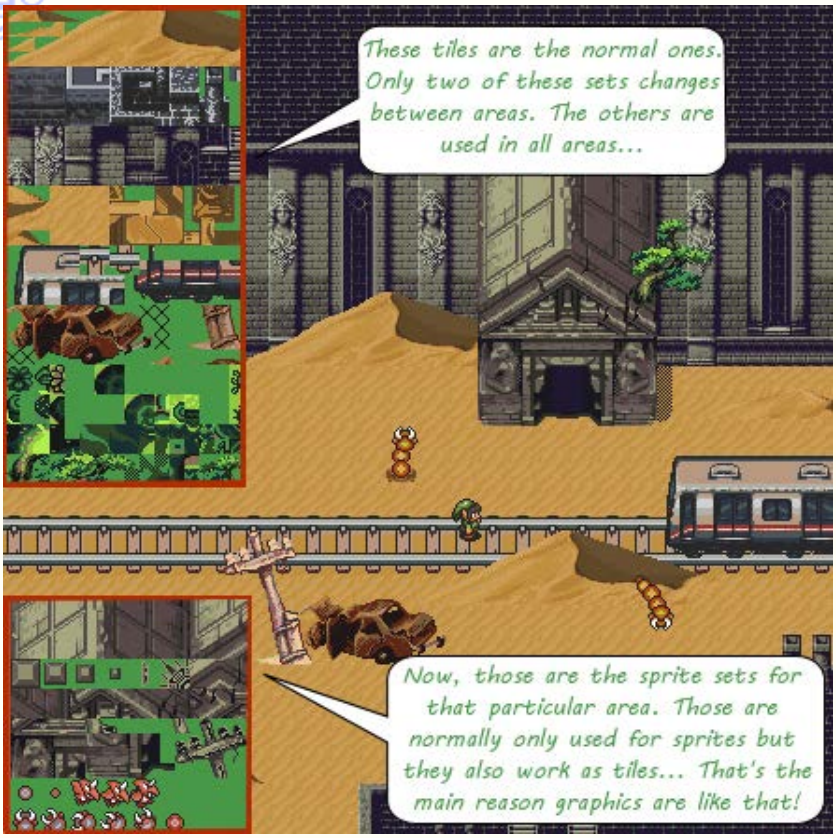
f) Palette

hfhcfdfnnfgnfngfn

g) Using the sprite sets as tilesets



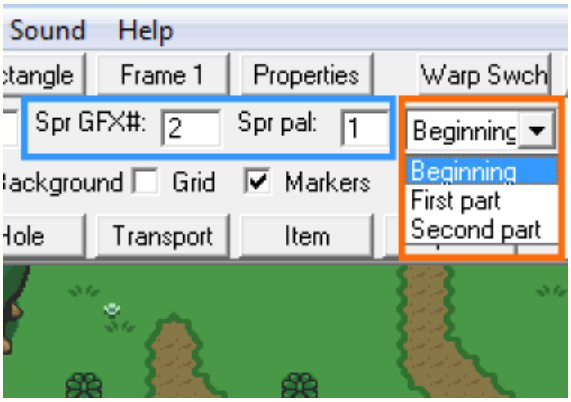
hfhcfdfnnfgnfngfn



h) Creating evolving areas

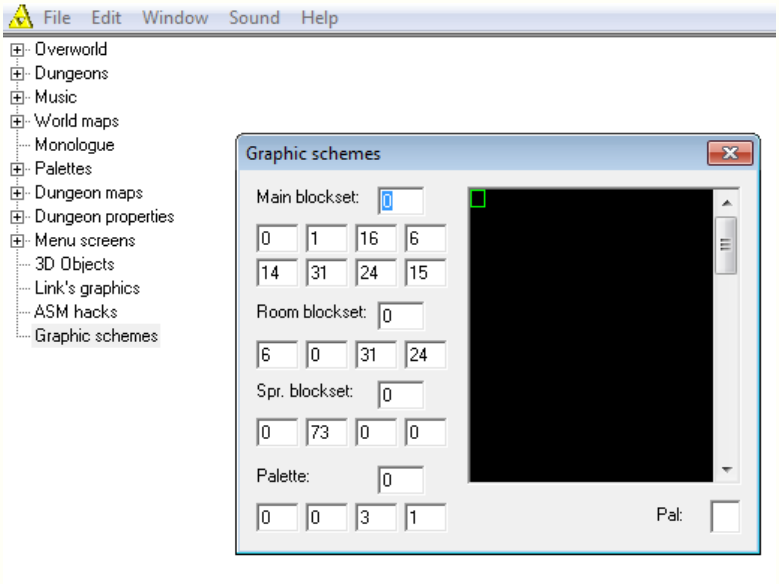
A long time ago while playing with the spritesets and the graphic schemes I've discovered a method to have areas that evolve over time in the game! The main idea is to use the spritesets in one of your areas as tilesets instead and later use the graphic schemes to make two more different variations to your current tileset. You can then assign a different Spr GFX# and Spr pal for each of the parts in the game to load different graphics each time a new event is accomplished!

You can have different graphics in all the different parts of the game, for either the beginning, first or second parts.



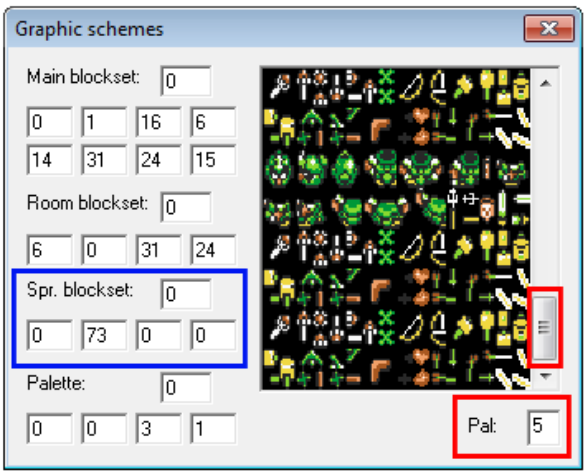
Picture: *You can set the current **Spr GFX#, Spr pal** and the **current part in the game** in the upper dialog boxes of the overworld editor main window.*

To get started, double click on "Graphic schemes" at the bottom of the Hyrule Magic menu.



Scroll down to the bottom of the graphic schemes default menu and change the palette in the bottom right corner to **5**.

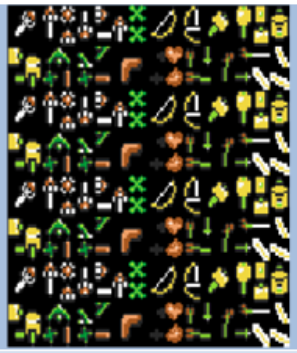
That will actually allow us to see something like this:



What we are actually seeing are the four spritesets that are connected to sprite blockset **0**. If we remember in the last chapter, it was mentioned that ***freespace*** in the Spr. Blockset can be found by locating those sprite sets:

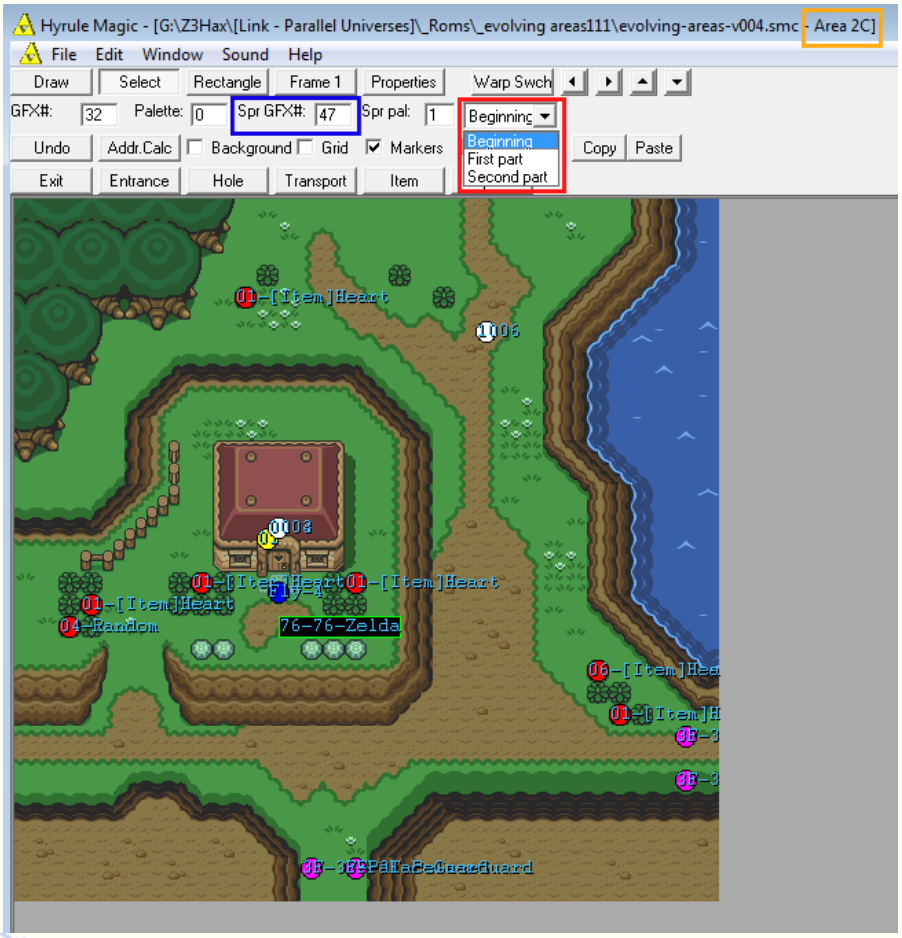
You can indeed change the 0 values to other digits and get other sprites to load in each of your overworld areas and dungeons! But… for this tutorial though we're only going to need the ***freespace*** that can be found in the overworlds since you can't really have evolving areas in dungeons :-P

Beginning with sprite blocksets 47 and going until 63 are free overworld tilesets (unused) so for our tutorial on evolving overworlds we will be using a few of those!

Name	Spr. blocksets 47-63
Graphics	
1st digit	0: *Freespace*
2nd digit	0: *Freespace*
3rd digit	0: *Freespace*
4th digit	0: *Freespace*
More info	*Much freespace in there for custom overworlds sprite sets.*

Let's remember that in the **overworlds** editor, the **Spr GFX# tags** are the same thing as the sprite blocksets found in the graphics schemes menu!

For now let's just close the graphic schemes menu, we'll come back to it later!



- Create a new hack of Zelda 3 and name it evolving-areas-v001.smc or something ☺
- Open the overworld tab and double-click on Area 2C which is links house.
- Click on the scrolling tab (highlighted in red) and select **Beginning.**
- Change the Spr GFX# (highlighted in blue) into 47.
- Change the entrance number on the door to 02 and the 0003 exit to 0012. Save your changes then go to Area 13 and delete the 0012 exit at the top of the church.
- Finally back in Area 2C change one of those palace guard soldiers sprite into Zelda and place her somewhere near the links house. Save your game again.

After that, go try it in game, go outside your house and save Zelda, enter again to deliver her to the priest inside the church, exit to get back into the overworld near your house again!

That event in question is the “beginning” into “First part” transition.

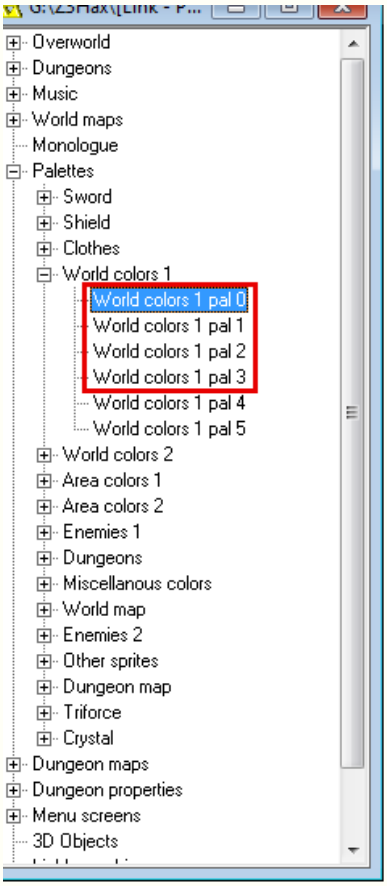
There is another one of those in the game and that’s after defeating Agahnim the first time and he transports you to the dark world. The next time you’ll come back to the lightworld, a new set of sprites will have loaded!

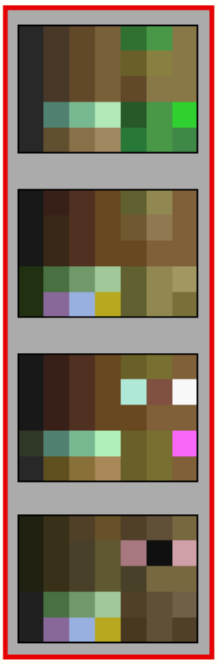
That’s where the joy suddenly comes in. Sometime ago while editing the tilesets I’ve found out that all the graphics located in those four spritesets can change of appearance between the beginning, first part and second parts of the game! Not limited to sprites per say, because if we want we can edit some of those spritesets to have both sprites and tiles to build houses/temples in them!

For example Spane requested to be able to have a buried temple in the beginning part that later looks partially excavated in the first part and finally in the second part it’s fully excavated and opened! It’s fully possible with graphics schemes only, no asm involved! If we wanted an asm hack to complement it, we could go as far as implement the red bomb opening on it in part two :-P

Before continuing our tutorial let’s make sure we understand each other correctly ☺

When using spritesets as tilesets the palettes loaded for your objects are always those found in the “World colors 1” palettes!



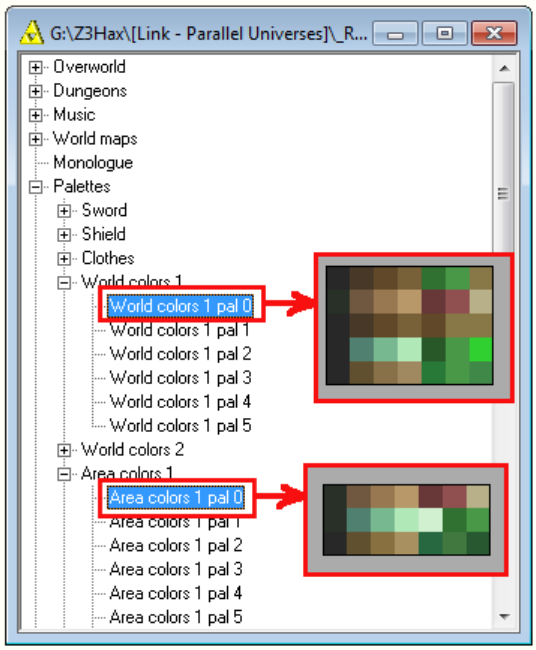


World Colors 1 pal 0
Light World Rocky/Grass Tiles

World Colors 1 pal 1
Dark World Rocky/Grass Tiles

World Colors 1 pal 2
Light World Mountain Area

World Colors 1 pal 3
Dark World Mountain Area



- For the purpose of learning, open the palettes editor and go on Area colors 1. Double-click then Areas colors 1 pal 0.

- Copy the first palette (the red one which is used for houses, red trees and other objects)...

- ...and paste it over the second palette in World colors 1 pal 0 (which is used for some areas that have dark brown grass and their cliffs).

- Save your game again.

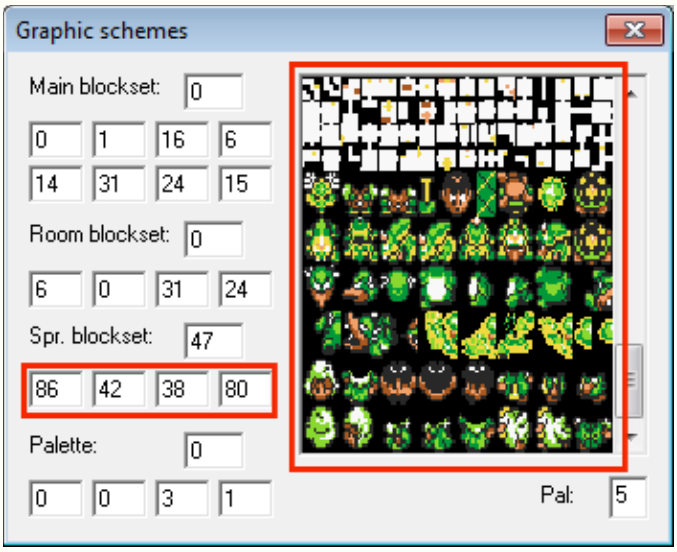
Now, let's make three different spritesets that contains different tiles for the three different parts! For this tutorial, I've created three different sets of houses and trees that have different shapes and styles overtime.

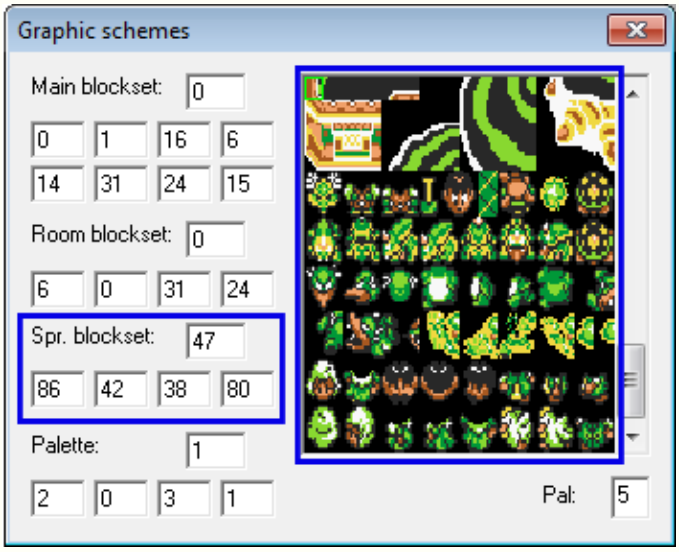
For now enjoy the picture but sometime later I'll add a gfx .bin file of those or an ips of the whole project to demonstrate how it fully looks inside the game! *or a video actually lol*

Our beginning tileset will be using Spr GFX# 47 while the first part will be 48 and second part 49.

I'll also include enemy sprites in those spritesets to show you how they can co-exist in the same space!
But first let's find three free space tilesets (those are actually harder to find). Since you can only use the space reserved for the sprites (not the tilesets at all), let's actually use the maps graphics for now since they can't be edited in Hyrule Magic anyway!

If you've followed all the graphic schemes tutorials to get to this point, you might remember the tutorial that lets you create your own sprite sets, if that's not the case I suggest you go read it before proceeding further!





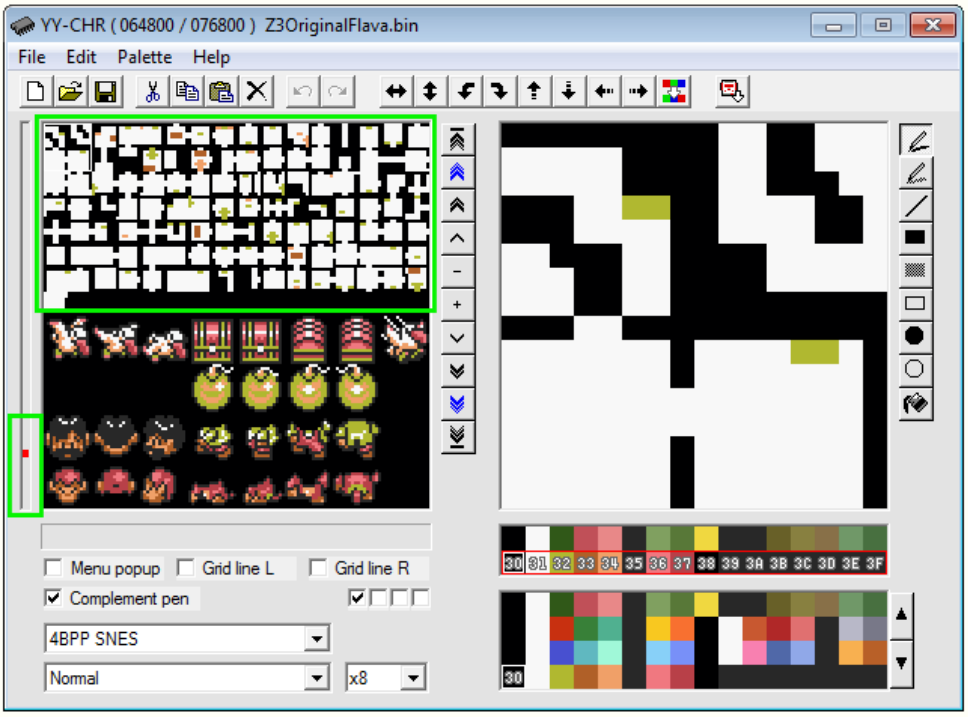
- Choose three (out of four) of your spritesets locations to be used as enemy sprites... be sure to use the right digits in the right locations so that the sprites show up fine. The last remaining spriteset will be for our evolving tileset!

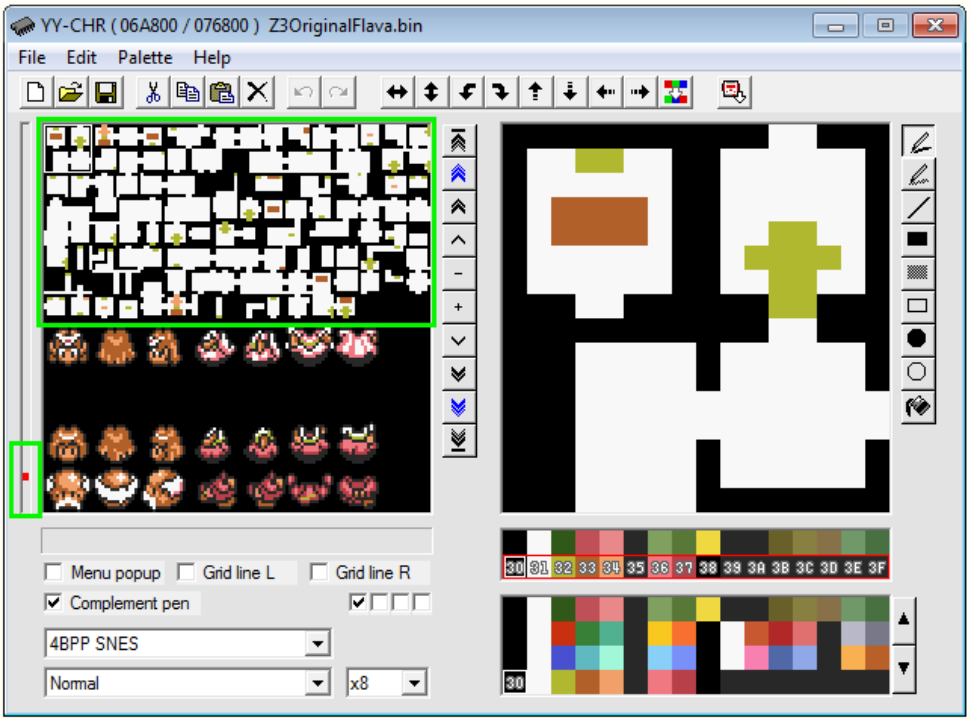
- Once you've chosen all your spritesets, press "X" and go save your game right away.

- After that use zcompress to decompress your game graphics and start editing them in yy-chr afterwards.

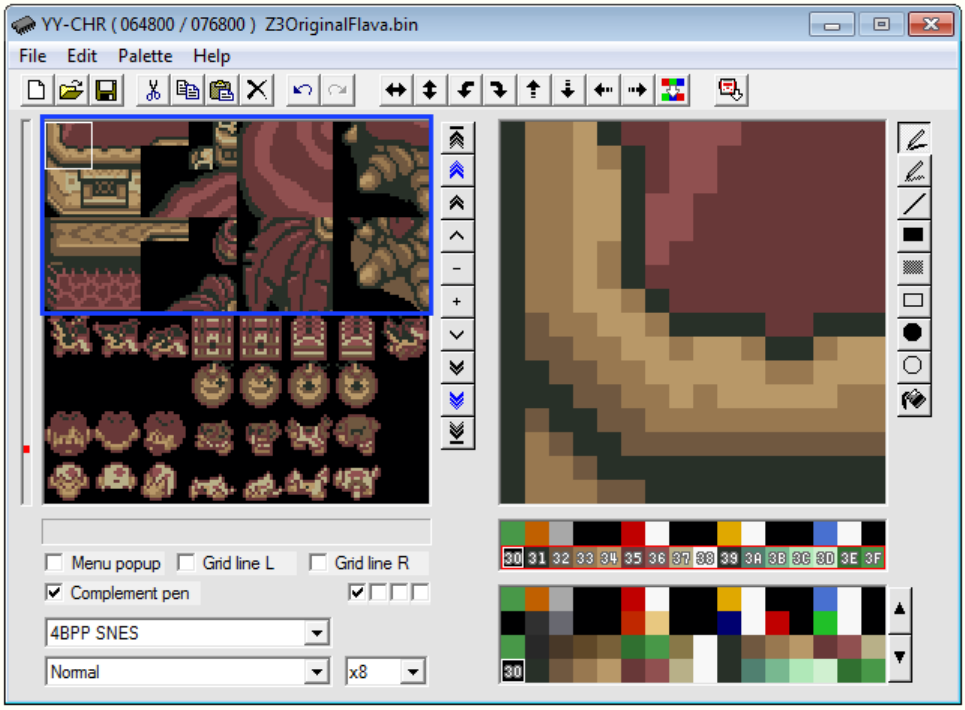
- For my house and tree evolving example, the Spr. blockset 47 will be using the maps graphics 86... while Spr. blockset 48 will be using the maps graphics 87 and in case of Spr. blockset 49 that will be 98 (again map graphics).

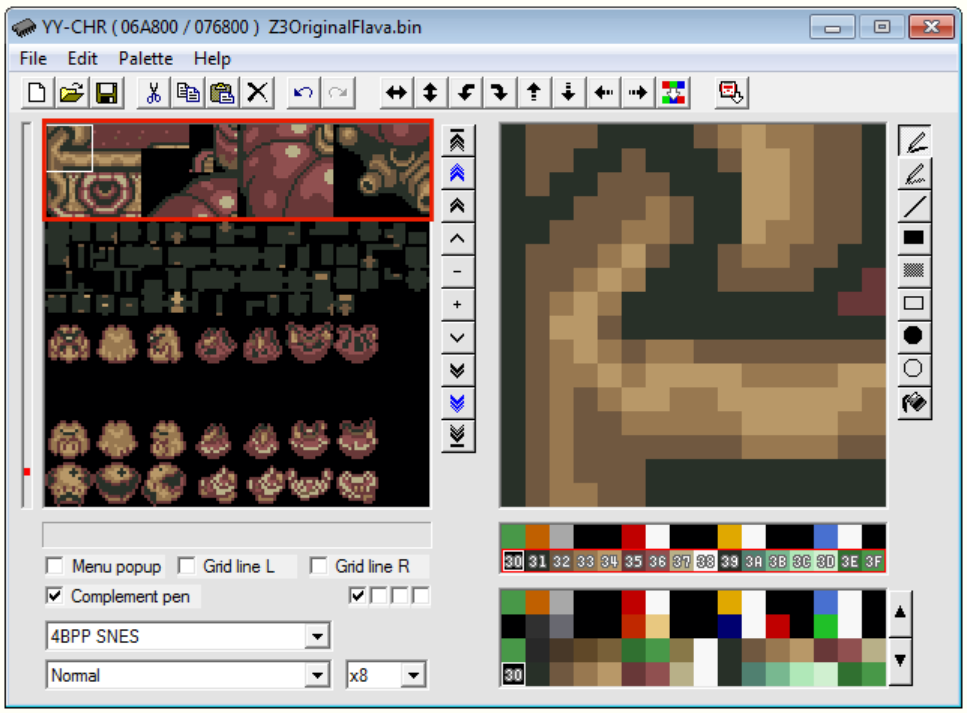
Now copy your previously created tilesets (or download the ones I created) from paint shop pro 9 into yy-chr and copy them over the dungeon map spritesets... they should be near this location in your decompressed graphics file:
(there are a total of four maps spritesets in the game)



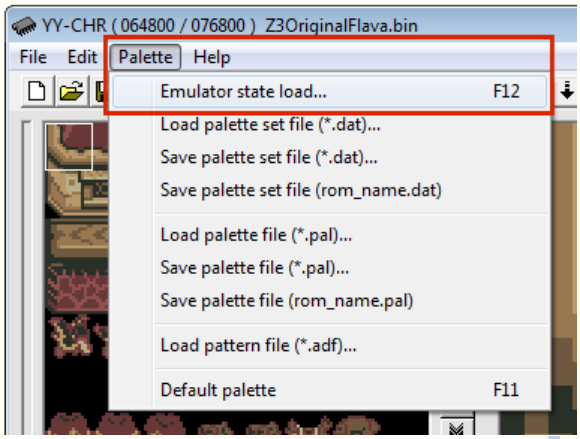


Once you have copied your new evolving tilesets over the old maps in yy-chr it should look like this:

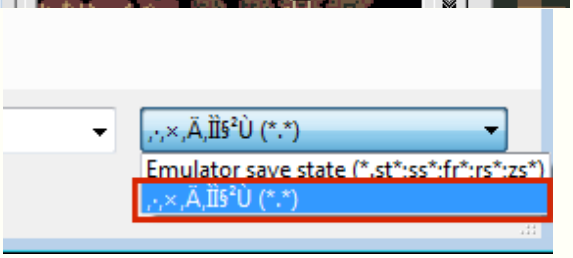




- To get the palettes to load correctly in yy-chr, all you have to do is make a savestate of the game once you are in an overworld area.



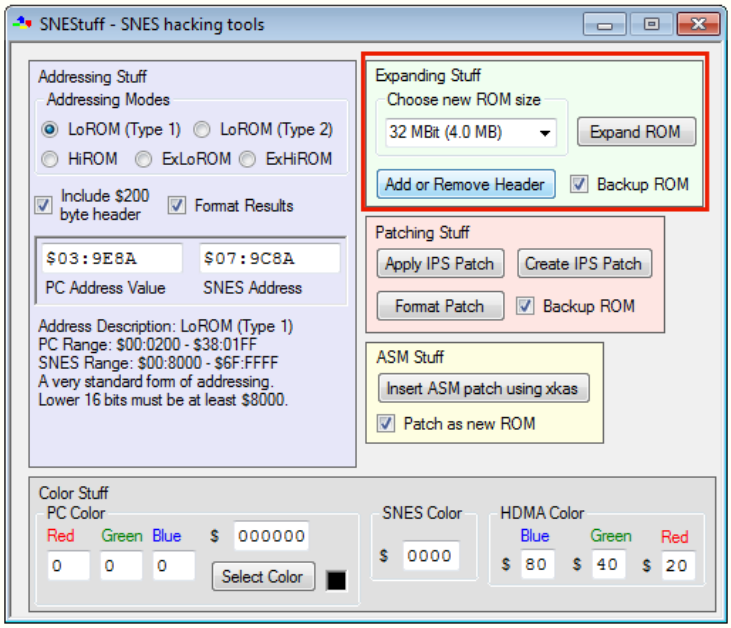
- You can load that palette in yy-chr using the top menu.



- And once the search window opens, click on the bottom right corner area and select the second option. Then go select the savestate you've just made on the overworld.

To get the right palettes to load, you can scroll through them using the arrows in the bottom right corner. Once your three tilesets are in place in yy-chr, save your graphics file. You are now almost ready to reinsert your graphics.

It would be wise before using zcompress to expend your rom to 4mb and adding a header to it or else you might have trouble reinserting graphics in zcompress because that tool needs a header most of the time and can be picky.



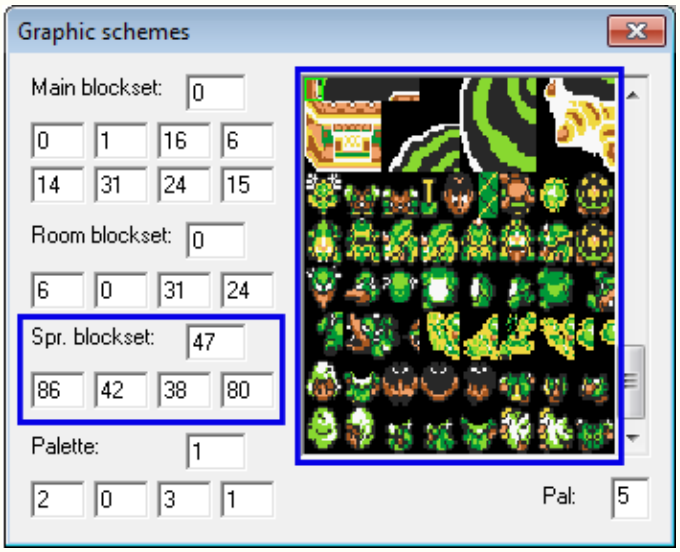
- Fortunately for us there's many ways for us to remove or add headers/expending it etc.. if I we're to suggest one program, I'd say SNEStuff since it got it all!

- You can both expand your rom and remove/add your header almost painlessly. It works with Windows 7 (unlike a certain SNESTool which I used to use) and probably with Windows 8 as well.

- You can also insert asm patches with this tool easily and convert addresses… but for now let's just expand your rom to 4mb and add a header to it.

- Reinsert your graphics in your game using zcompress and when you are ready, open the graphics schemes menu to get started on creating our basic evolving tilesets at long last! ☺

Once inside the graphic schemes scroll to the bottom, change the palette to 5 and also change the Spr. blockset to 47… You should see something similar to this (based on the sprites you've chosen):



- My Spr. blockset 47 is composed of four different spritesets:

- The 1st one is used for our kakkariko red houses tileset and tree, **write 86** there.

- The 2nd one contains the digging game guy, the shell and triceritops enemies and others. Can be anything in your case (that fits there of course).

- My 3rd ones are penguins and icemans, once again you can have any sprites that fit in the 3rd spriteset to show here.

- Finally, my 4th spriteset is composed of townfolks and chickens. You can of course have anything here as you wish.

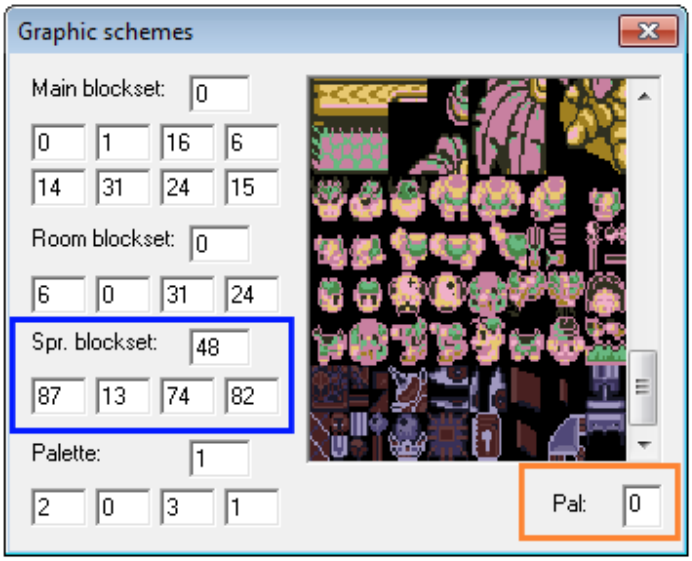
** *You can use the previous parts of the graphics schemes tutorial as a reference on how to properly re-organize your spritesets!* **

- We're now going to create two other new sprite blocksets before saving the game.

- *Don't exit the graphics schemes window yet as you can actually edit many of them before quitting or saving.*

- Change the Spr. blockset value 47 to 48.

- In the first digit, **write 87**, the second one can be any sprite that is appearing in the 2nd spriteset and the same thing for your 3th and 4th digits.



Or alternatively you can use my own choices:

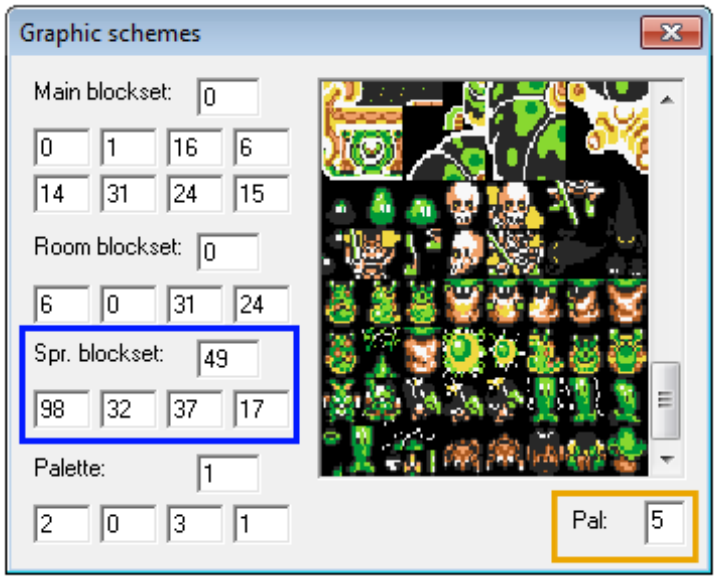
To view the 4th digit, change the highlighted palette to 0

- The first digit will be for my beach tileset and tree.

- The 2nd one will have the dark world enemies that are replacing the light world soldiers ☺

- The third one will contains some more townfolks.

- And the last digit will some dungeon sprites but on overworlds this time!



Finally lets go create our last spriteset for our evolving areas experiment!

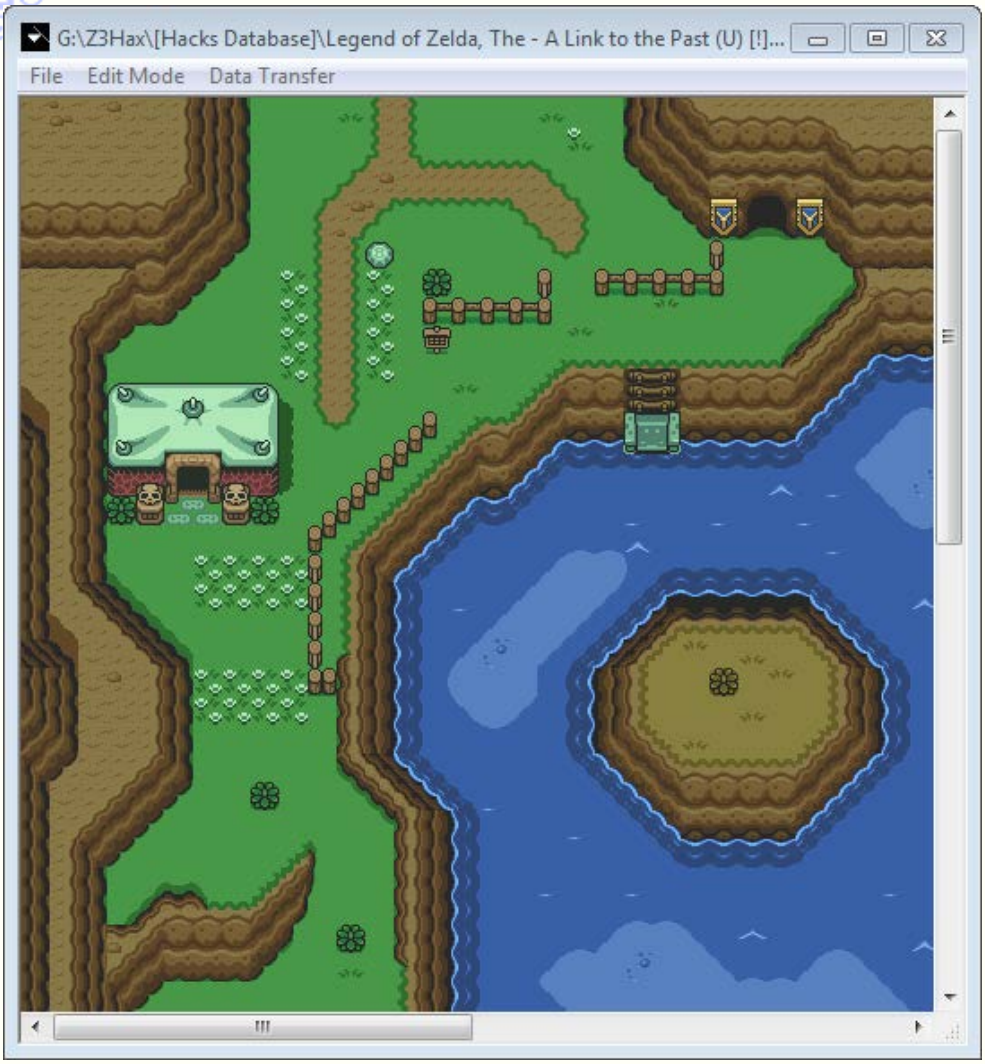
- Change your Spr. blockset value to 49.

- Change the first value into **98**. This is our dark world tileset complete with house and tree.

- The 2th, 3rd and 4th digits can be any of your choosing or use the ones I predefined!

Now it's time to save everything and start working on our evolving tilesets! So close the graphics schemes windows, save your game and if you haven't made one already, make a new rom backup aswell (we are never too cautious with Hyrule Magic).

~ Chapter II - The Black Magic editor



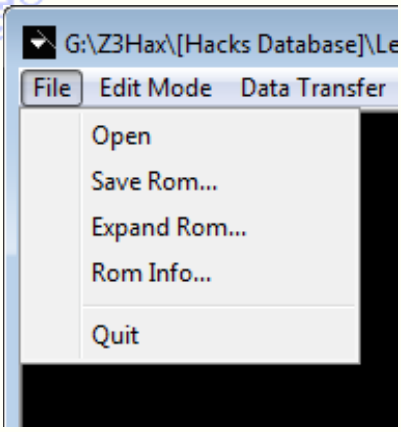
1) A little history about the editor

Hyrule Magic's legendary buggy nature, made many people stay away from that editor. To this day even, very few people use it to the fullest since its learning curve can be quite hard.

Fortunately for the more common people amongst us, MathOnNapkins is hard at work to give us a remedy.

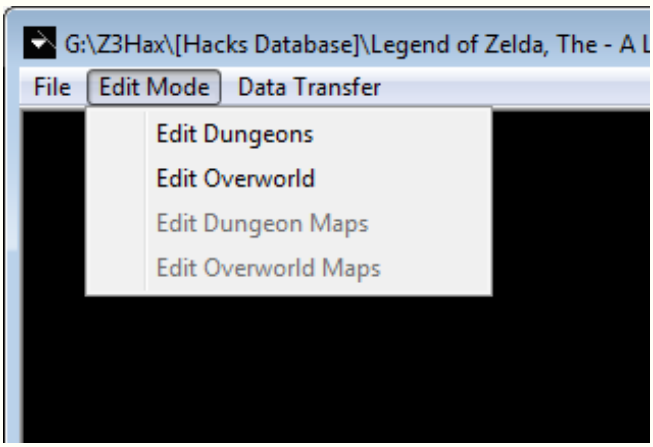
Coming in with the name Black Magic (no pun intended!), this new editor will address many problems that Hyrule Magic has, while bringing its lot of new features as well.

2) The interface



File

Open: *Opens a ROM (Must be a valid **Zelda 3: A Link To The Past USA Version ROM**).*
Save Rom: *Saves the ROM.*
Expand Rom: *Expands the ROM, choose which size you want, click the "Change" button and finally, click on the "Done" button to return to the previous menu.*
Rom Info: *Current checksum.*
Quit: *Exits the program. If you've made changes to your rom and haven't saved yet, it'll ask you if you want to save your game.*



Edit Mode

Edit Dungeons: *Self-Explanatory*
Edit Overworld: *Self-Explanatory*
Edit Dungeon Maps: *Self-Explanatory (Option not available yet)*
Edit Overworld Maps: *Self-Explanatory (Option not available yet)*

[picture]

Data Transfer

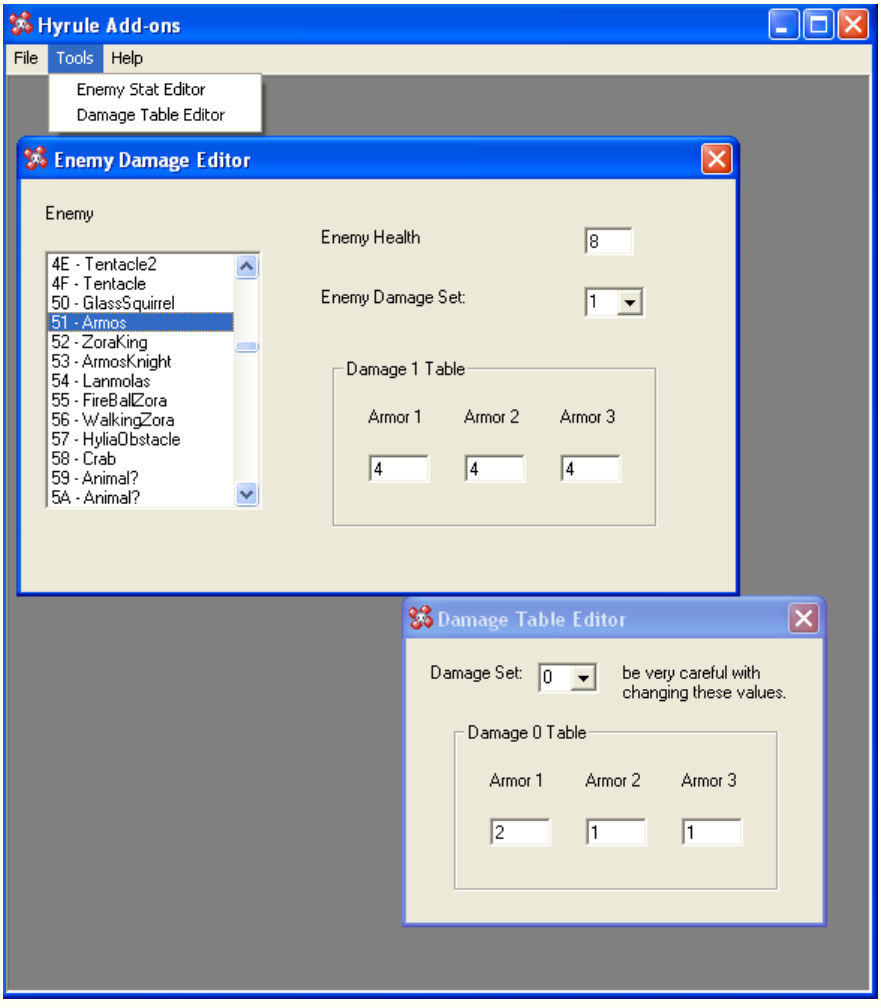
124124
214214214
412214
214214214

3) Latest news and developments

fdhdfbndfbhdfbasascas sd dsfg d
ds sdd dfg dfg dfd fdgdf

~ Chapter III – Additional tools

1) Hyrule Add-ons
by Euclid



Hyrule Add-ons is a program which lets you edit all sprites stats (health, damage set and damages tables

<u>Brief Instructions:</u>

One program load one rom, no more, it's a lot more work to allow more than 1 rom to be loaded at the same time the way I've programmed this in VB! Oh and don't use this program at the same time as Hyrule Magic (you can have HM open, just don't save in it).

After you open the rom, everything is under that "tools" menu.

Oh and if you want to change the names of some sprite, go change them in the names.cfg file.

Requires some VB6 runtime files to run properly.

Some tips and and warnings:

- There's about 10 different "types" of damage (more like 16, but I suspect the rest aren't used due to the ridiculous amount of damage you'll take if they used it).

- Each enemy chooses one of these "types" as to how much damage they'll do to Link.

- You're free to change the damage, but I've put them into 2 separate forms so you can't accidentally change them in the enemy stats editor.

- Do make sure you don't play around with damage type 0 and 2 (they're somewhat special since sprites which aren't enemies use them).

- Also don't attempt to change sprites F3-FF since I believe those aren't really sprites.

- Everything is loaded in memory, so even if it crashes, your rom is safe, and also your rom is not modified until you hit that save button or Ctrl+S. Don't trust me? make a backup then.

- One more thing, the program doesn't know if you have changed the values so there won't be a confirm exit box when you hit that close button.

Here is how to resolve comdlg32.ocx missing error:

Download comdlg32.zip and extract comdlg32.ocx from zip file
Move comdlg32.ocx to c:\Windows\system32 folder.
For 64bit Vista/Win7, move comdlg32.ocx to c:\Windows\SysWOW64

Open a command line window and run following command:

regsvr32 c:\Windows\system32\comdlg32.ocx

Note: a) On 64bit Vista/Win7, run this command instead:

regsvr32 c:\Windows\SysWOW64\comdlg32.ocx
regsvr32 c:\Windows\SysWOW64\msstdfmt.dll

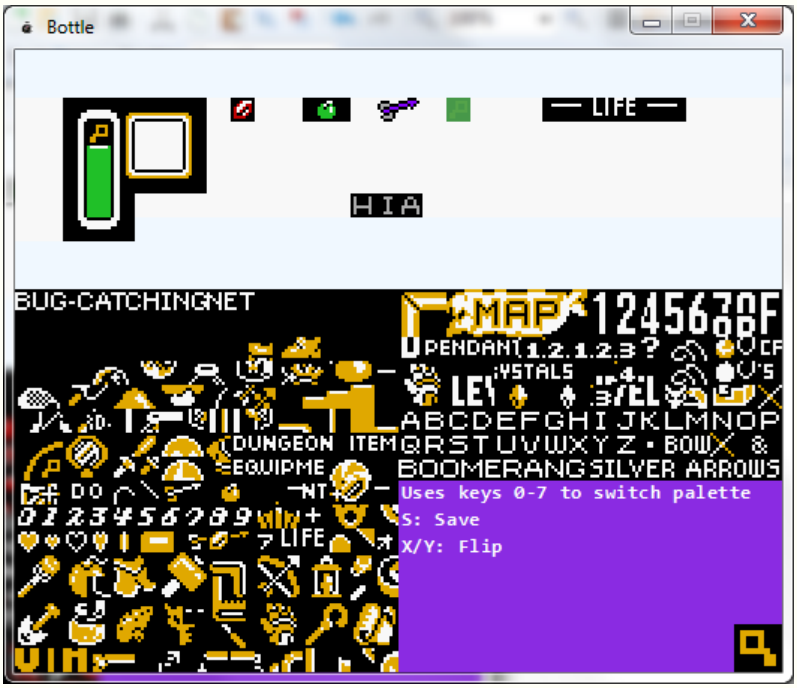
b) For Vista/Win7 with UAC turn on, the above command needs to be run from elevated command prompt.
Once comdlg32 is registered successfully, following message will prompt, DllRegisterServer in C:\WINDOWS\System32\comdlg32.ocx succeeded.

Now launch the application again you should not see file missing error.

Update: Important note if you use WIN7: when registering comdlg32.ocx, it must be done as administrator, or regsvr32.exe will fail with error 0x8002801c.

To do this as administrator, Go to 'All Programs ->Accessories ->Command Prompt', right click on 'Command Prompt' icon, and click on 'Run as administrator' to start a command prompt, then run the regsvr32 command

2) Bottle
by Vitor Vilela

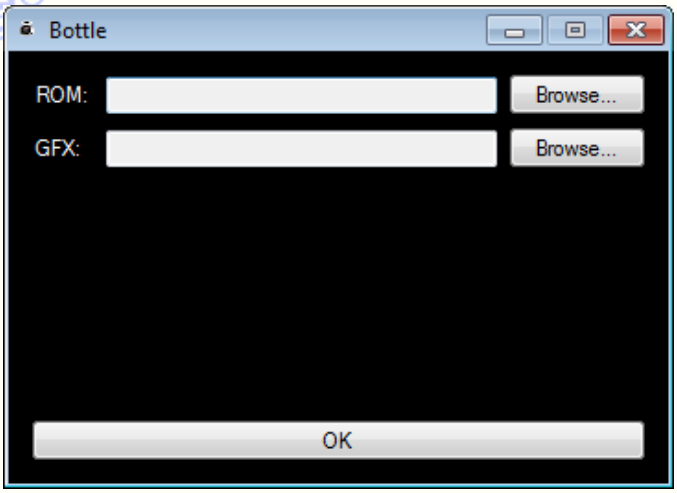


This tool will allow you to partially edit the status screen (hud).

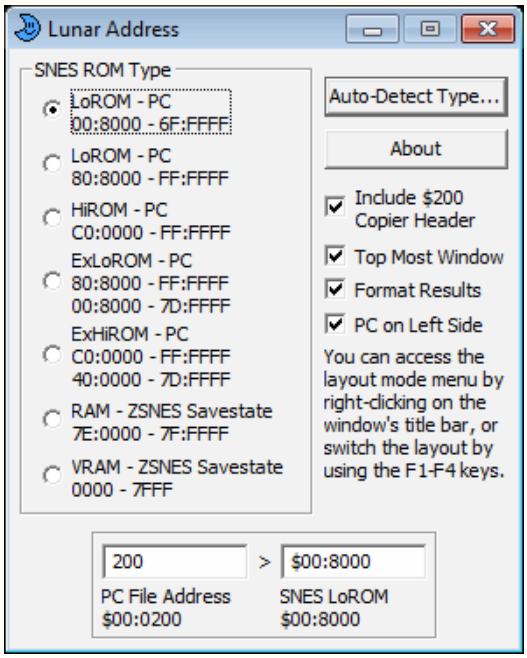
Take note that the counters or the item select screen cannot be edited with this tool... that requires a bit of ASM.

Refer to **chapter seven** for more information on how to change those parts along with information on how to edit the inventory itself.

Instructions:
Using ZCompress, extract the GFX from your ROM. Now open the tool and give it your ROM and GFX file, after that it should be pretty self-explanatory.



3) Lunar Address
by FuSoYa



Lunar Address is basically a SNES address converter with a layout based on the useful Hex to SNES program from J2E, but with some additional functionality.

First off, the text fields are standard edit controls that you can copy and paste into using the windows clipboard. Also, the program ignores extra non-hexadecimal characters, so you can input formatted addresses straight from an assembly trace if you like. Thus 8000, $00:8000, and 80M00N are all the same as far as the application is concerned.

Conversion is done automatically as you type, and a text field below the edit control will describe the address and identify what part of the ROM it accesses. Note that you can convert from RAM to ZSNES save states at any time in one of the ROM modes without having to be in the RAM - ZSNES conversion mode, simply by typing in a SNES RAM address.

The program also allows you to convert SNES addresses from outside the "standard" range for the ROM type selected. For example, 00:8000 is a perfectly valid pointer in a HiROM game, even though it doesn't access the same PC location that it would in a LoROM game.

And Lunar Address includes support for ROM sizes above 32 Mbits up to 64 Mbits using the ExHiROM and ExLoROM options.

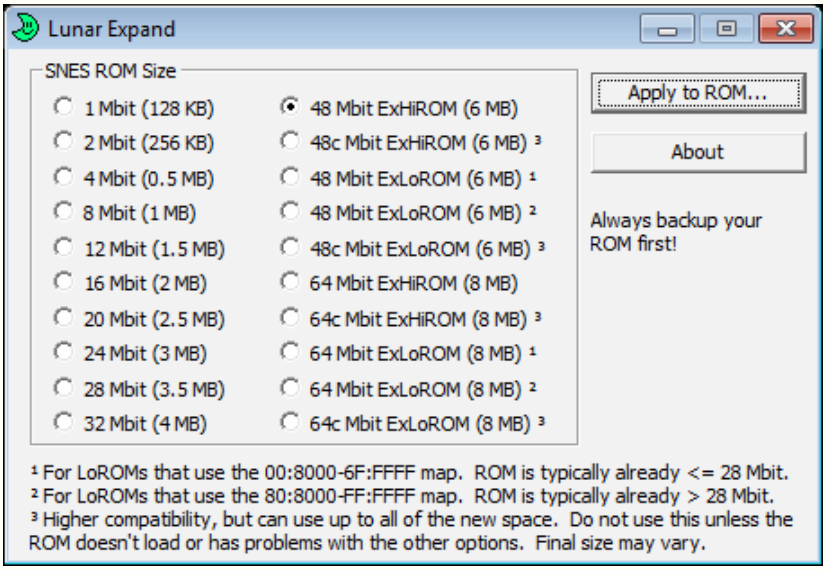
If you don't know what type of ROM you're working with, you can use the Auto-Detect feature to let the program try and figure it out for you. This will detect both the ROM map type and the header. It should work for most ROMs, but if the ROM is interleaved or uses an extra chip it may not be of much use.

The program actually has 4 different layouts that you can choose from in the system menu (right-click on the window's title bar). "Expanded" is the default mode that the program starts in. "Large" moves the ROM type selection and other options into menus, which considerably reduces the size of the window. "Medium" is much like "Large", except without the menu. And "Small" removes everything except for the 2 edit controls... when combined with the "Top Most Window" option, this can be extremely useful for keeping the address converter present but out of the way while you work.

You can also use the F1-F4 keys to quickly switch between the different layouts available.

All settings are saved to the registry, so when you run the program again it'll be in the same state you left it in.

4) Lunar Expand by FuSoYa



Expansion of SNES ROMs to sizes over 32 Mbit has traditionally been a bit of a headache for most ROM hackers, partly because you have to do it a bit differently, but mostly because emulator support for it has been relatively recent. And since the mirrored areas of the memory map are used to hold the extra data, it's not even guaranteed that expansion beyond 32 Mbit sizes is going to work in all cases with all ROMs.

I've put together this program that provides a couple of options to try and deal with expansion of this sort.

On compatibility with emulators: if your emulator can play ToP, it supports the 48 Mbit ExHiROM option. Both Zsnes and Snes9x have supported this one for a while now. Snes9x 1.39a (or 1.39mk3) or higher is required for support of 48 & 64 MBit ExLoROM and 64 Mbit ExHiROM expansion (Zsnes doesn't currently support these).

Keep in mind that if your ROM uses any extra chips (SFX, SFX2, DSP1, etc), then you're most likely out of luck with expansion as these chips tend to use their own memory maps. The program also doesn't support interleaved ROMs. It does support ROMs with or without a 0x200 byte copier head.

In the program, the left column gives you choices for expansion up to 32 Mbit. These can be used on either HiROM or LoROM games. The right column gives you choices for expansion to 48 or 64 Mbit. If you have a LoROM game, make sure to choose an ExLoROM (Expanded LoROM, AKA JumboLoROM) option to expand your game. And likewise, if you have a HiROM game, choose an ExHiROM option. The program does attempt to verify which ROM type it is first, but it's better to be safe than sorry.

The ExHiROM options are also capable of converting an 8 Mbit or smaller LoROM game to ExHiROM, though it's unlikely that anyone would want to do that.

Now, all the choices in the right hand column do the basic steps for expansion beyond 32 Mbit (expand the ROM, change the byte size in the header, and copy a 0x8000 bank of data to a section at the start of the expanded space). But you'll notice that there are up to 3 different options for each ROM size.

For ExHiROM, the first option (no note) is just a typical expansion. The compatibility option (note 3) copies every second 32K bank into the expanded space so that the 00:8000 map accesses the same data it would in the game before expansion. This can take up to half of the new space, but it's useful for evil HiROM games that have pointers using the 00:8000 map. Not all the copied banks may be necessary, so if you _really_ need more space you can try clearing out some of them, but be careful...

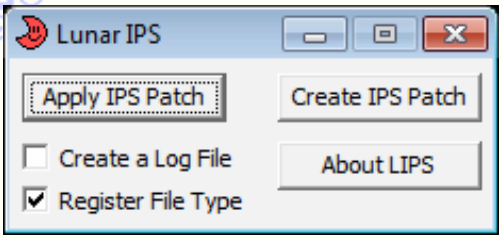
You can access the space above the 32 Mbit mark in the ExHiROM map at 40:0000 - 7D:FFFF. Note that banks 7E and 7F are RAM, thus the last 128K of your 64 Mbit ROM is not accessible from there, though you can access the upper 32K of the two banks at 3E:8000 - 3E:FFFF and 3F:8000 - 3F:FFFF. Also, the area from 0000-7FFF of banks $70 - $77 is usually used by SRAM, so the corresponding areas in the ROM are not accessible.

For ExLoROM, the first option (note 1) is for LoROM games that use the 00:8000 - 6F:FFFF memory map. Typically ROMs already <= 28 Mbit in size use this map. In this case, the original ROM will be copied into the expanded space at 40:0000. The second option (note 2) is for LoROM games that use the 80:8000 - FF:FFFF memory map. Typically ROMs > 28 Mbit in size use this map. If you have an evil LoROM game that uses both memory maps, you can try using the third compatibility option (note 3), but this simply creates a second copy of your ROM in the expanded space. If the ROM was already 32 Mbit before, you won't gain much from this... you'd have to figure out manually which banks in the first or second half of the ROM can be cleared out. >_<

You can access the space above the 32 Mbit mark in the ExLoROM map at 00:8000 - 7D:8000. Note that banks 7E and 7F are RAM, thus the last 64K of your 64 Mbit ROM is not actually accessible in the game.

Anyway, the program seems to function fairly well in most cases. Games like Chrono Trigger and Mario World have been converted successfully into 48 Mbit ExHiROM games. RoboTrek is one that had to use the compatibility ExHiROM option. Always try the other options before resorting to the compatibility options.

5) Lunar IPS
by FuSoYa



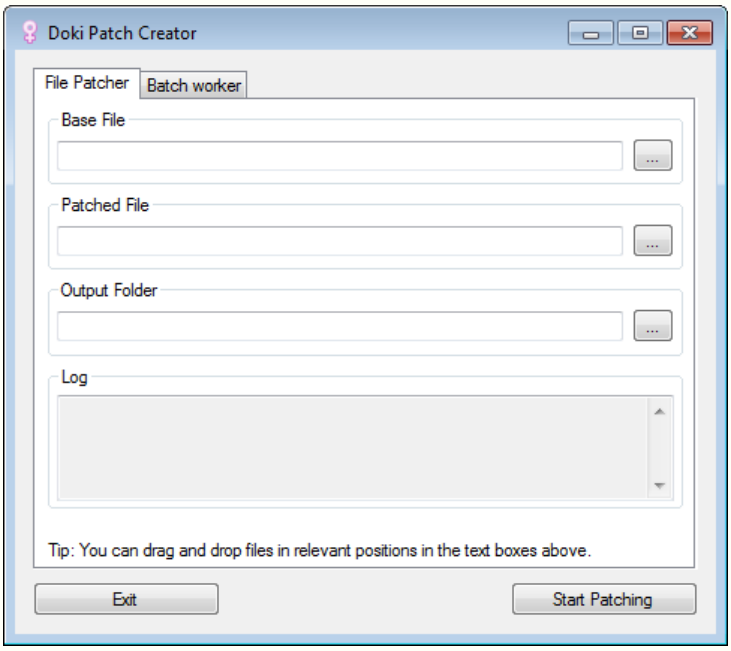
Lunar IPS is intended as an easy to use, lightweight IPS patch utility for windows to replace the SNESTool DOS program. It can both create and apply IPS patches.

As far as features go, LIPS has:

-IPS patch creation/application support.
-full RLE encoding/decoding support.
-file expanding/truncating support.
-the IPS encoder creates files that are the same size or smaller than files created with SNESTool.
-the IPS encoder avoids the rare "0x454F46 (EOF) offset bug" that SNESTool's IPS encoder has.
-logging feature for applying IPS patches (ROMFileName.log).
-registers the ".IPS" file type so that you can just double click on an IPS file and choose the file to apply it to for convenience.
-support for patching files up to 16 MB in size, which is the limit of the IPS format. The files can technically be larger than that, but the IPS format cannot record changes beyond the 16 MB mark due to 24-bit addressing. The IPS file itself can be any size.

Note that the logging and file registration options are saved to the registry.

6) Doki xdelta patch creator
by Jonatan Nilsson



Information by PuzzleDude:

This surely is one of the best if not the best patch creator and autopatcher I've ever seen. We all know, what the main problems of any file patching are: using the wrong original rom, problems with renaming the new patched file, problems with the extension of the new file and problems with the original rom being overwritten!

The standard IPS patchers fail completely in all of the above mentioned problems. The new UPS, xdelta, etc. patchers are better, but everyone will surely miss out something. Most of them include the option: ignore the checksum, which really is a strange choice by the authors, allowing the dumb user to click this and corrupt the rom.

Others (in their wish not to overwrite the original rom) force the user to input a new name of the new file. This is a problem for a dumb user who will MOST surely forget to input the .smc or whatever the extension the rom has. The user will then wonder what is wrong, since the file doesn't show up in the load menu of the emu. Plus, why should we enter the name manually.

Incredible as it sounds, most patchers will corrupt the rom if the patch is accidently patched twice. They don't attempt to cancel the operation. This can happen to a dumb user, especially because if he forgot to rename the file (which still has the original name).

Now let us take a look at the Patch creator by Jonatan Nilsson:
- It will NOT allow any false original roms (crc checksum is mandatory and automatic when creating or applying the patch).
- It will NOT corrupt or overwrite any of the roms (original or patched) and will do nothing to the files if the patching process is done multiple times.
- It will NOT force you to rename the patched file. It will auto name it for you, the way the author named it.
- It will NOT force you to input the file extension, like .smc or similar. It will auto determine the file extension. It is the same as the file, which the author used, so it is automatically correct.
- It uses the delta patch system, which provides the best possible shrinking of the patch.
- It uses the automated patching (no annoying browsing for the files). A dumb user can select an original rom, when asked to select the modified one. This cannot happen here. Automated means you double click on one file only. If anything goes wrong, you are kindly informed about it and the process is terminated and nothing is lost or corrupted.

How it works:

This Patch creator makes a batch file which executes a command done by the xdelta3 exe file in a combination with the actual .delta patch. So the auto patch uses 3 files in a folder to operate. The fourth file (which is the original rom) must be inserted by the user.

When double clicking the batch file, it will tell you which name it expects. Rename the original file to the exact name as specified by the program, and everything is done automatically.

It is up to the author to define the version of the rom: EU, US, header, no header.
Luckily only the correct version will be executed.

Creating a patch:

The program itself uses more than one file to operate, so a folder with the same name as a program and no other programs is expected.

Under the Base file, we must browse to our Original UN-modified file. Its name and crc checksum is important.

Under the Patched file, we must browse to our Modified file (hack). Its name and file extension will be preserved when patched.

The output folder is automatically formed as the "patch" folder. Sometimes the name is different.

Finally we click on the "Start patching" and we're all done.

What was created in the folder?

Actual patch, exe file and 2 files to auto apply the patch, one is for the Linux. We can rename these 2 files only. It is logical that these 2 files (and the patch folder) have the name of the modified game. For instance Super Demo World TLC Auto Patcher. However, this is not mandatory. The default names is apply_patch.

Applying a patch:

Double click on the auto patcher (apply_patch by default) to determine the NAME of the original file. It will tell you: Attempting to patch "name".

For instance: Attempting to patch "Yoshi's Island.smc"
Copy the original file into the same folder, where the auto patcher is, and rename it into this specific name: Yoshi's Island.smc (usually the .smc is not seen).

Now double click on the auto patcher again. If the crc is matching, a program will auto create a new .smc file, with the final name and extension, ready to be played. Run it in an emulator and it will surely work, because its crc, name and extension are the same, as the author made it.

Problems:

1.) The file cannot be found
You didn't put the original file in the folder, or you didn't RENAME it to the EXACT NAME, as specified by the program. The process has stopped and nothing is lost or corrupted.

2.) Target window checksum mismatch: XD3_INVALID_INPUT
You don't have the correct original rom, you have for instance EU or J instead of U, or your file has a header, when it shouldn't; or it has no header, when it should. The process has stopped and nothing is lost or corrupted.

Get the correct version and use the TUSH header remover/adder to fix this and repeat the auto patching.

3.) To overwrite output file specify...
You double patched the file. The process has stopped and nothing is lost or corrupted.

4.) xdelta3.exe is not recognized
You messed with the xdelta3.exe file which must not be touched (re-download the patch).

5.) Failed to read patch.delta
You messed with the patch.delta file which must not be touched (re-download the patch).

Correct patching:

The command line reads: Attempting to patch...

The new file is auto created, has a correct name/extension and it is ready to be played.

Happy patching and creating patches!

7) ALttp Room Header Expander
by XaserLE

[missing picture + link]

Information:

This is the Zelda ALttP Room Header Expander.
Cause Hyrule Magic lacks the possibility to expand the room headers so you can connect nearly
all rooms you want, i wrote this script to manage this.
It copies the room headers to another position at the end of the rom, changes the bank which the
game uses to load the headers and changes the header pointer table.
It is not possible to patch the rom and then edit it in Hyrule Magic further. But this is no problem.
You have your actual ROM and export your actual room headers (the 5 bytes for the doors) with this program.
After this there is a new file called 'RoomHeaders.txt'. Open it with Notepad++ or a similar program (not
Windows Notepad). You will find a comment line at the beginning to see how it works.
For each room you have a line in that you can change doors/staircases/warp. I suggest you to keep it
open while editing with Hyrule Magic. After saving this you can import it. You don't need to back up
your ROM cause the program creates a new file called '_YourROMname'. The new file has all what you set
in Hyrule Magic for Effect, Tag1, Tag2 and so on and the new rooms from the file "RoomHeaders.txt".
So nothing goes lost.
The new headers are written at offset 0x1F8000 (beginning of bank 0x3F), so make sure there is no data there.

Important: Note that you cannot get from the rooms 0-255 to the rooms 256-295 and vice versa.
So for example if you write in the file that a staircase in room 256 leads to 128
then this will be interpreted as 256 + 128 = 384 and the game will crash.
You only need to remember that for rooms 0-255 the only valid values are 0-255
and for rooms 256-295 the only valid values are 256-295.
Don't care about rooms 296-319, the game will crash if you go in one of these rooms.
****About that part: PuzzleDude found a way to use them!!****

Problems:

-If your ROM has a 0x200 Bytes header, you need to remove it, otherwise it will not work.

-You need the following ROM: "Zelda 3 (U).smc" (date of last modification: 23.08.2004).
So your room header pointers should start at 0x27502 with the value 0x82F7.

-You need to delete whitespaces in your ROM filename.

I think i will fix the problems in the future if there is someone interested in this program.

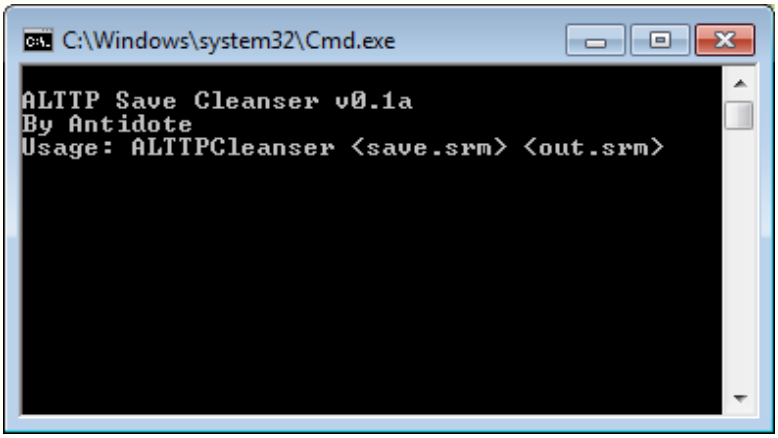
Usage:

Import: 'RHE.exe ROMname'
Export: 'RHE.exe -export ROMname'

Credits:

Coded by XaserLE (www.le-softworks.com)
Thanks goes to the great MathOnNapkins for his Zelda documents and his personal help.

8) ALTTP Cleanser
by Antidote

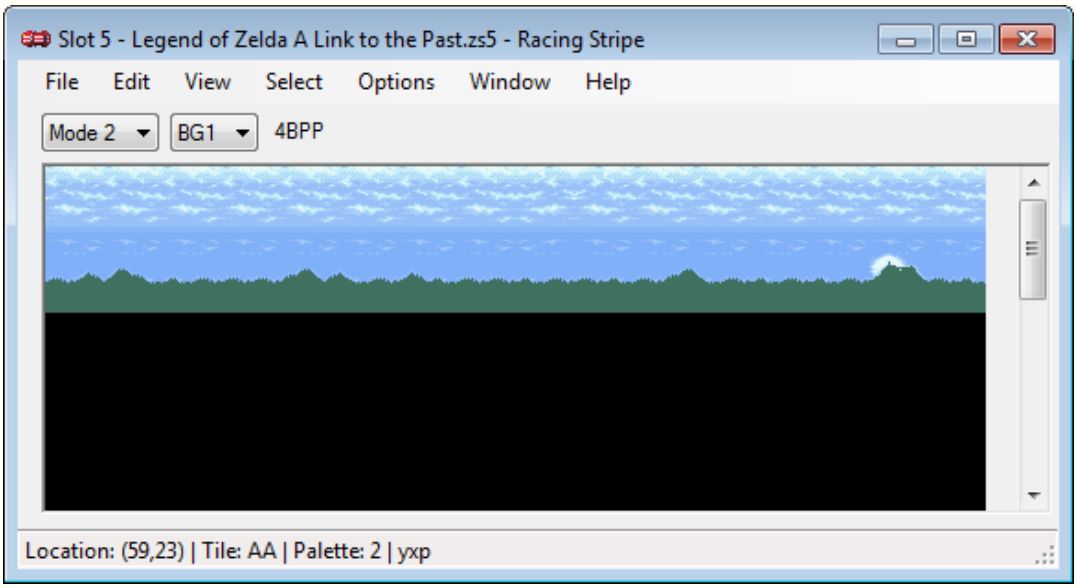


Using MathOnNapkins documents as well as his disassembly, I managed to make this little gem.

What it does is takes a "corrupt" ALTTP save and corrects it's checksum, it has no sanity checking so it may throw an error. This is more "proof of concept" at any rate.

--link--

9) Racing Stripe
by editl754



Racing Stripe, at the most basic level, is a ZST tile map editor with Stripe Image support. Stripe Images are used in Super Mario World, as well as a few other games, as a method of uploading tile maps to VRAM.

Menu Items:

Open ZST

Go to File > Open ZST and browse for your ZST file. By default, *.zs* is selected. This will display ZSNES save state files #0-9. To view #10-99 as well, change "Files of type" to *.z**. However, this will also cause a few extra files to show up, such as .zip files.

Save to ZST / Save To New ZST

Go to File > Save to ZST and a dialog will come up. You will see three checkboxes, all checked by default. The first one, "Use New Tile map Location", will insert the BG into the ZST at the new tile map Location set in Options > Change Tile map Address. This only applies if you have changed it. The second one, "Use New Screen Mode" will save the screen mode you set in the "Mode" drop-down box. This only applies if you've changed the mode. The third checkbox, "Remember for this BG" will prevent the dialog from coming up every time you save, and remember the options you selected. You can undo this by going to the Options menu and unchecking "Remember ZST Save Options". If you have any "blank tiles" in your BG, you will see that the "Fill Blanks With" option is enabled. This lets you choose what tile data to fill those blanks with. If you're saving to a new ZST, it will ask you if you want all further changes to be saved to the new ZST instead of the original.

Load Tile map/Stripe

This lets you import a tile map file or stripe image file. It will be rendered with the current BG's properties on the same size tile map. If you open a stripe image file, another dialog will come up asking for the location of the tile map the stripe image draws to. It will make a guess based on where the majority of the tiles are, and the size of the current tile map.

Save Tile map/Stripe As & File > Save Tile map/Stripe

Let's you save your currently-open BG as a compressed stripe image file or as a raw tile map file. If you choose stripe image, a dialog will come up asking you for the tile map address the BG will be drawn to in-game. The default value will be your BG's current tile map address. If you check the box "Remember", going to File > Save Tile map/Stripe will not bring up the dialog. You can reverse this by going to Options and unchecking "Remember Stripe Image Save Options". Going to Save As will always bring the dialog up. If you choose Raw Tile map File, and you have blank tiles in your BG, a dialog will come up asking what to fill them with. The "Remember" Checkbox serves the same purpose as the one for stripe images, and you can reverse it by unchecking Options > Remember Raw Tile map Save Options.

Export Image

This allows you to save the currently open tile map as a PNG or BMP image

Preview Stripe Image Size

This option will tell you how many bytes your compressed stripe image file will take up.

Replace

in Edit > Replace, you can replace certain tiles with other tiles. You can specify what info to look for in a tile (just tile number, tile number + palette, etc.) and specify which information to replace. To fill the slots automatically, first click on the tile you want to replace (either in the main window or the tile selector) and then hover over the tile you want to replace it with (again in either window). In the dialog, the checkbox before each option in the first one indicates whether or not to check for that property, and in the second box, it indicates whether or not to change that property.

Change BG Properties

This allows you to change various info about the BG, including where in the VRAM to load tiles from, the tile size to load/display, and the tile map size

Change Tile map Address

This lets you change the VRAM address the tile map is located in, and gives you the option to reload tile map from the new location.

Offset Tiles

This lets you add a certain value to all the tiles. Any tiles that go beyond 0x3FF will wrap back around, so if you select 0x3F0, for example, it's like selecting -0x10.

Convert 16x16 Tile map to 8x8

For tile maps using 16x16 tiles, such as those in Yoshi's Island, this lets you split up the 16x16 tiles into 8x8 tiles. NOTE: tile maps like this will NOT compress very well into stripe images, due to lack of double-tile RLE (ABABAB...) so they will require a bit of editing.

Please note that this is not all you can do with this program. It is designed to be a very general tool, and should have many more uses.

How to use this tool to help rip FGs:

1. In ZSNES, make a lot of save states in the level you want to rip the FG from. I recommend making the save states with specific parts you want to rip on-screen.

2. Open one instance of Racing Stripe for each save state, and open your save states.

3. Optionally, you can turn on a grid (recommended if you're ripping from an 8x8 tile map)

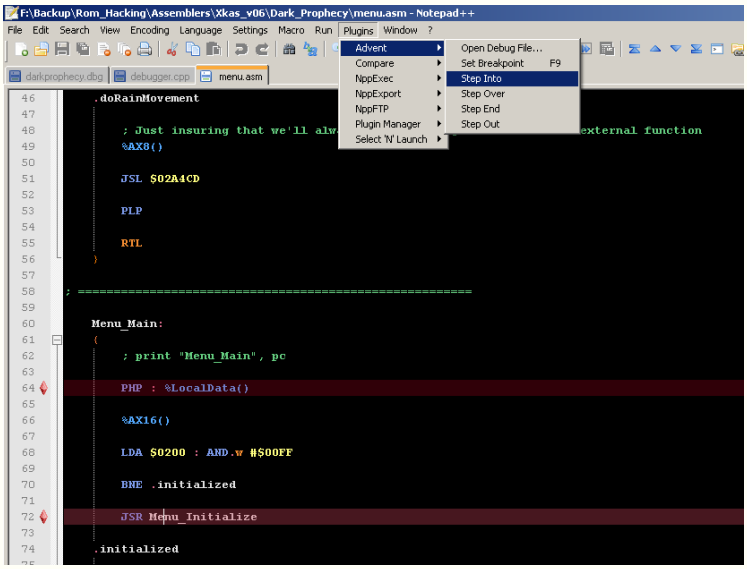
4. Start copy pasting all the tiles you need onto one tile map. You can increase the size to 512x512, and use up to 512x432 tiles of space.

5. When you're done, rip it with my online BG Ripper at http://edit1754.co.cc/bgripper.php (keep in mind that you're still limited to 256 unique tiles when ripping FGs with this method)

6. Insert with LM. insert the Map16Page onto one of the FG pages with F2. (you might first want to extract a blank page to generate a blank Map16PageG file)

7. Start editing the behaviour settings and whatnot

10) Advent - Integrated SNES Debugger Plugin
by MathOnNapkins



Important: *While it could be used by a novice SNES 65c816 hacker, this set of tools is primarily aimed at experienced users that are jaded with the SNES debugging status quo.*

Advent is a plugin for Notepad++ that bridges the gap between the textual aspect of coding assembly language for the SNES and actually running it in an emulator debugger. It is modeled to some extent after Visual Studio (and other similar graphical frontends to debuggers). Advent as a whole could be thought of as an entity depending upon three modules:

- The Advent plugin DLL (authored by yours truly)
- A special build of byuu's xkas version 0.06 (see http://byuu.org/programming/)
- A special build of byuu's bsnes version 0.67 (the debugger build)

This is quite a specific set of tools, but nothing about this setup necessarily dictates that they be used. Notepad++ is Windows only but a comparable editor on Linux or Mac could have a similar plugin developed. Geiger's Debugger could be interfaced with this debugging protocol without too much fuss. Other assemblers could output the same debugger information. In fact the debug information format ought to be rewritten in the future to be more compact and contain more information, but that's getting ahead of ourselves.

One thing I should note is that xkas v0.06 has macros and this presented a bit of a stumbling block for outputting debugging information. So stepping over / into a macro may make it seem like you've lost the current line marker, but it will re-emerge after the end of the macro. Currently only the first instruction of a macro will register in the debug file.

~ Chapter IV - Graphics editing

1) Zcompress and YY-CHR
by sorlokreaves and ghillie

Link's graphics are unique in that they are encoded with 4 bits-per-pixel (a format supported by most major image editors) and they are (I believe) uncompressed. The remaining graphics are much harder to edit. You basically have to follow three steps:

1. Decompress the graphics from your ROM to a .bin file.
2. Edit the graphics using YY-CHR.
3. Recompress the graphics in your .bin file and insert them into the ROM.

This method is so general that it works for almost any game, so long as you have the right decompression tool. **Lunar Compress** can handle several popular formats as well as Zelda 3, but for this tutorial we're going to use Zcompress which works for Zelda 3 only. The only other thing you need to remember is how many bits-per-pixel each graphic is encoded in.

All right, let's edit Link's shield graphics. First, open up a command prompt (Start->Run, type "cmd.exe" and hit Enter. On Vista just click Start, type "cmd" and hit Enter. On Linux… you should know how to do this.)

Browse to the directory you saved Zcompress to. Make sure you have also saved a clean Zelda3.smc ROM file there as well. Now, type:

zcompress.exe 0 Zelda3.smc Zelda3Gfx.bin

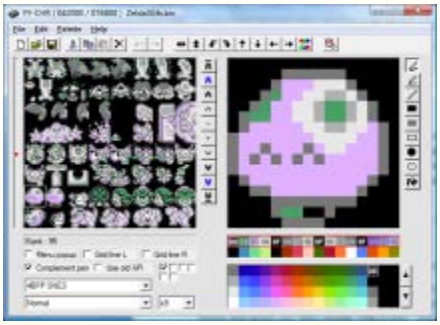
…on Linux, you'll have to do:

wine zcompress.exe 0 Zelda3.smc Zelda3Gfx.bin

Now, open YY-CHR and select "File->Open", then browse to your Zcompress directory. In the "File Name" box, type the following and press Enter:

***.bin**

This allows you to see .bin files. Now, select your Zelda3Gfx.bin file. The window will load with apparent garbage. You can fix this by clicking on the "Format List" (currently, it shows "4BPP MSX/Genesis) and selecting "4BPP SNES". Hit PgDown or PgUp to browse through the tile set; you'll notice that the shapes are familiar but the colors are all wrong.



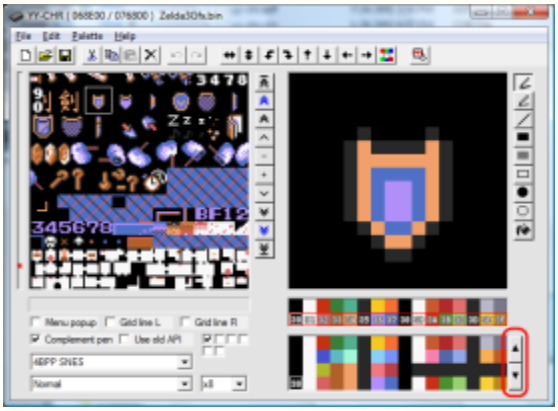
Picture: *We've found Mushroom Boy's ghost!*

It's perfectly fine to edit the sprites using these colors, but you can make them much more accurate with just a little extra effort.

1. Decide which sprite you want to edit, and locate that sprite in-game. We're going to edit Link's shield, so play through the game in Zsnes until Link's uncle gives you his shield. Now, save a state (F2) and note the state's id.

2. Close Zsnes, and locate the file that was just saved. For example, if you see "State **0** saved", you will be looking for Zelda3.zs**t**; if it's "State **7** changed", you'll need Zelda3.zs**7**. Copy this file into the Zcompress directory, to make things easier.

3. Now, click on "Palette" then "Emulator State Load". Browse to the Zcompress directory and load your .zs* file. (You might not be able to see it; just type ***.zs*** into the file window, similar to how you located the .bin file.)
4. Scroll to your sprite in YY-CHR (the shield is roughly 85% of the way down the screen). Now, click on the palette arrows (circled in red, below) to scroll through all possible palettes. You might not find the exact palette; just find one that's close.



Picture: *Link's shield, with a slightly-distorted palette.*

Now, edit the shield. Make it bigger. As you do this, make sure that you only use the colors already in the palette. Stealing from colors further along is a bad idea, as it makes it much more difficult to edit those other palettes later. Just make something simple, then choose "save".
After you've saved, close YY-CHR; our Zelda3Gfx.bin file now contains our new shield graphics. We just have to re-insert them. Switch back to your command prompt (in the Zcompress directory) and type in:

zcompress 1 87200 Zelda3.smc Zelda3Gfx.bin

This will re-insert all of your graphics (including the shield) –it will start doing so at 0×87200, which is where a normal Zelda3 ROM expects to see graphics data. Zcompress has the ability to export all graphics as separate files; in this case, you will have to figure out their insertion addresses yourself. But if you always use one big .bin file, the address should always be 87200.
Open your ROM, and play until you get the shield. Don't your new graphics look nice?

Picture: *We've added a blue face to Link's shield, and enlarged It.*

Of course, we can do better than that! Go into the palette editor in Hyrule Magic and scroll down to "Shield 0". Change the dark blue to white, the light blue to navy, and the white to red. Now, re-arrange your graphics in YY-CHR to look spookier. Ooh~~~ skull shield!

We've Changed the Face Into a Skull! Note That Link's Uncle Uses the Same Shield As His Nephew.

Are we having fun yet? Keep in mind that some graphics are more stubborn than others. I still can't manage to figure out how to change the treasure chest graphics. Also, some tiles are mirrored, so if you're editing, say, tombstones, you don't have to bother looking for the "right half" graphics —they're simply mirrors of the "left half" graphics.

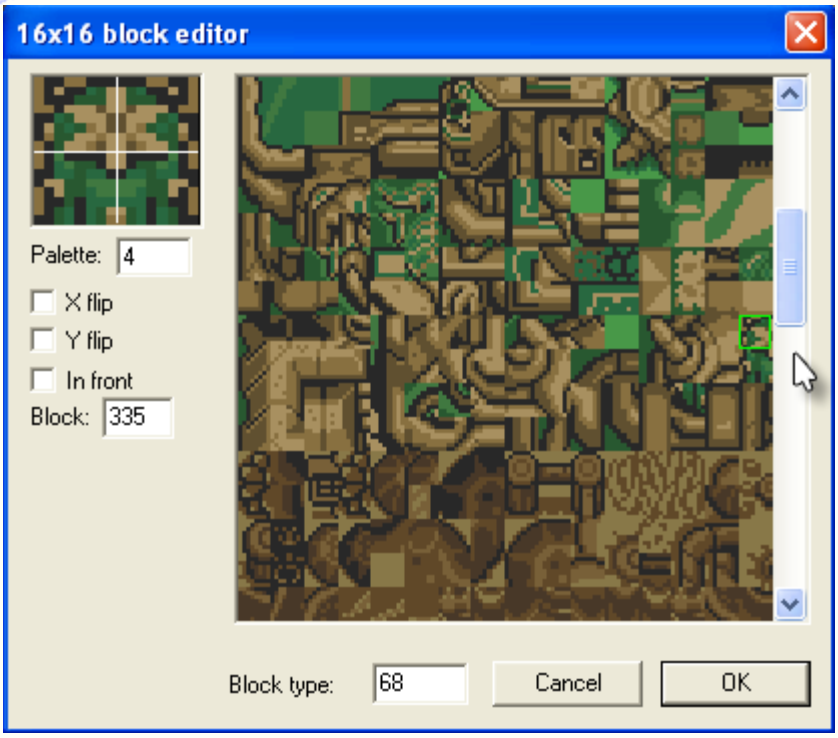
2) Paint Shop Pro 9 [*outdated ~ needs updating*]

by Orochimaru

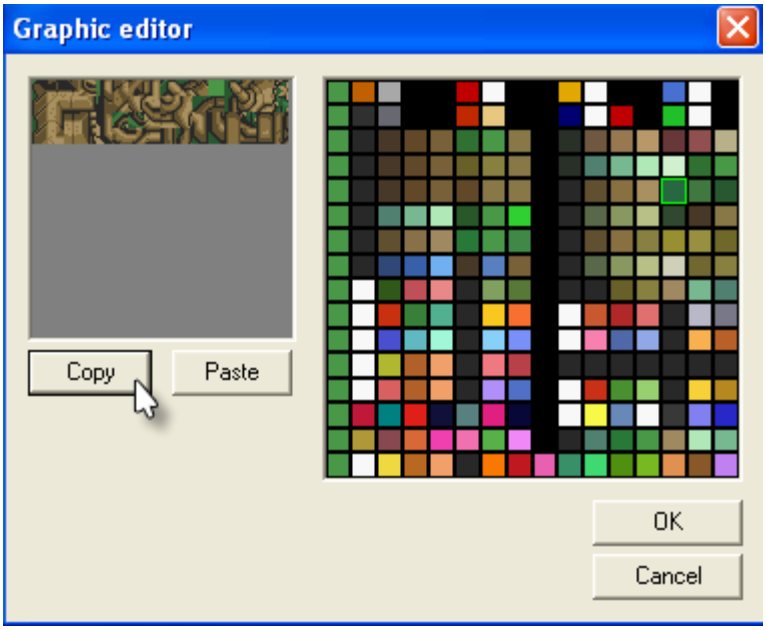
To change the GFX tiles across your overworld you need to do the following
(After having found the GFX tiles you want to use):

01 > Open your rom.
02 > Go to Overworld Editing.
03 > Select the area containing the tiles you want to change.
04 > When in that area, right click on the tile which contains the tiles you want to modify. (In that example I'm going to change the desert trees for some bush).
05 > Double-click on the tile you want to change on the right rectangle.

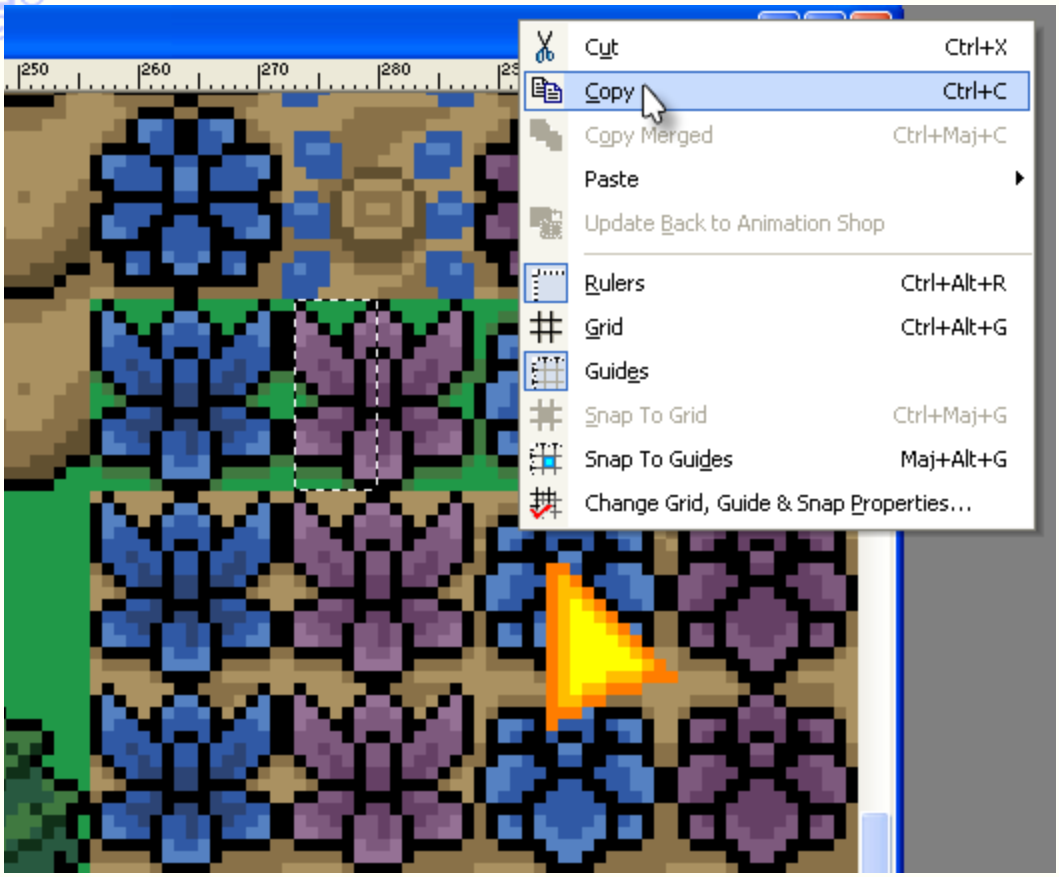
06 > In the 16x16 block editor window, right click on the first tile of the 4x4 square and then double click on the same tile on the right rectangle.



07 > In the graphics editor window, move through the palettes until you find the appropriate palette for your custom tile.



08 > Press the copy button, and go in Paint Shop Pro.
09 > Go to Edit and choose Paste As a new image.
10 > Open the custom GFX tile you want to put in the game.
11 > Copy the tile (Right click on the title bar, choose copy).



12 > Paste the tile in your Zelda 3 sprite sheet (Right click on the title bar, choose Paste As a new selection)

13 > Move the tile where you want it to be.
14 > Unselect the file by right clicking anywhere in the picture.
15 > Chose Copy in the right click context menu of the title bar.
16 > Go back in HM and select Paste this time. Close everything/Save all progress and re-open the rom as sometimes it

doesn't save. (HM is being evil)
17 > If you followed this tutorial, you should come up with something like this:

~ Chapter V - Hex editing

1) Hex basics
by PuzzleDude

Hex is really easy to learn:

0 - 9 = 0 - 9
A = 10
B = 11
C = 12
D = 13
E = 14
F = 15

Hex basis
00,10,20,30,40,50,60,70,80,90,A0,B0,C0,D0,E0,F0,100,110...

Hex address 6F152 = address 6F150, byte 02.

Example
Address 1C060 = has 16 bytes = 00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0F.

Address 1C06B or 0x1C06B or $1C06B = address 1C060, byte 0B.

Header is from 000 to 1F0, 200 bytes, all with value 00.

Working with Hex is easier with no header, but removing the header might cause problems in extracting the graphics from the rom.

FREE PROGRAMS

-HxD Hexeditor by Mael Horz (has red colour bytes after editing)
-WindHex32 editor (has relative search)

2) Hex lessons
by solid-tbone

The Beginning

Now this is the part that scares people… counting hex numbers. It's so simple after you get it. You will smack yourself for not learning it before. Hex goes a little something like this:

00,01,02,03,04,05,06,07,08,09,0A,0B,0C,0D,0E,0F…..

You really don't need the 0 when counting, but for romhacking reasons, we will put them in there. When you get so high it gets a tad confusing.
It goes like this:

D0,D1,D2,D3,D4,D5,D6,D7,D8,D9,DA,DB,DC,DD,DE,DF…

After DF comes E0. People have the hardest part with that. Always ends in F and always begins with 0 when counting. It's from 00 to FF. In decimal the total is 255. It NEVER goes any higher, ever!! Just do not forget that after EF would came F0. Another is 49 comes 4A and CD comes CE. I would recommend writing down all the hex numbers so you get the idea. Takes time, but do not give up. Hex addresses are read in the same way except the numbers rise in value. This number depends on the size of the rom. At hex address 1001A, the byte there is 39. Addresses and bytes are 2 different things and never get them mixed up. Addresses hold bytes. That should make it simple enough. You can't change addresses, but you can change bytes. Bytes are what hold the rom data we go looking for. It becomes easier, just read on.

The Next Step

Open up a hex editor and open a rom (In this example I will use Castlevania). You will see a bunch of hex numbers. All these numbers hold important data for the rom. The first 10 bytes in ALL roms are for the header so emulators can read them. Never touch them. This will be important when I go over adding Game Genie Codes to a rom. There is nothing in a rom you can't find and change. Searching through a rom can be a major pain in the ass. If you do it correctly you can find good usable data for the rom. Just take every 8 bytes or so, except for the first 10, and change them all to 00. Make sure you copy the 8 bytes you changed so that after you see what it does you can put them back in. I use Hex Workshop and Translhextion for my hex editing needs. I ALWAYS make copies of the rom I am using and I write all my information down. Always save after you get some great info. It happens to the best of us, believe me I know.

Finding level data

There really isn't much to finding data in a rom. Here is an example of what I did to find the level data. I searched every 8 bytes until I found the data I was looking for. At address &H1001A you will find the byte 39. Change that byte and save it. Open up the rom in your favourite emulator and play the first stage. You will see the very first tile changed to an all-black tile. Isn't that totally awesome?! I was very excited when I found this data. It just takes time, don't give up. Let's move on.

Finding more level data

This is a little easier since you found the level data for Castlevania. This takes a little time, but just change so many bytes and keep checking the rom. You should notice that level 2 begins right at the end of level 1. The only difference is that the rooms are not in order. The last byte in that room is where the fish men jump out of the water. After that, go to level 2. You will see that the first byte in level 2 is changed. I am pretty sure it's around &H10865. I haven't been playing with

hex in a while and I lost my CV document that I had with all my info. I will find it though. Using the same steps I am telling you here. Level 2 level data is also not in order so mess with all the bytes from there and you will find the level data. That is really it. Search.

Finding a title screen

This is easier than looking for level data. I did this easily and was surprised how quick I found it. Every title screen has some kind of text info in it. I took Castlevania title screen and looked up words like push start now in a hex editor. Most hex editors support text search, like hexposure. After that, you have part of the title screen. Just keep going back until you find the very first byte in the screen. Unlike level data, title screen data is usually 1 sprite big. So changing it to your specifications can be a little time consuming, but definitely worth the trouble in the end. I used Nesticle95 and TLP to view the pattern table and find out what byte is where. You can click on a byte in Nesticle and find out what byte is assigned to a sprite. There are 2 different sides. They change due to changes in RAM I think. The title screen data in a pattern table is usually on the right hand side. Just click and find out what byte is assigned to what sprite. It becomes easy after you do it. Everything does technically. Just keep plugging.

Finding Graphic data

This is the easiest part and the hardest part. Open up TLP and Castlevania. Scroll down until you begin to see sprites. You will see a sprite that looks like a line. That is the beginning graphical data for Castlevania. It should begin at &H3508 and end around &H10000. I been up for 15 straight hours and I am too lazy to open it. You just saved yourself a ton of time looking through a rom for all this information. Now when looking for data, you can skip over a ton of bytes that were once unknown data. Some nes roms have what they call compression. They compress graphics to save space. There are a good amount of different formats that you have to learn. I don't have time to teach them here, because I don't know them all and this is a hex document. They do have programs that help you look for it and FuSoYa has made a program called Lunar Compress that decompresses certain roms. Nifty program and I suggest you pick it up. I sure hope that someone else can pick up on this subject. Sorry if I don't know that much here. Getting help from people can be such a task sometimes. Let's move on to our final subject(s).

Finding other types of data

You be amazed by how one byte can hold important data. I am going to use Megaman 5 as my example. When I was learning how to convert Game Genie codes I learned that Megaman's jump height data was held by one byte. You could find this the hard way by changing all bytes one by one until you come to hex address 360BE. The byte there is 05. If you change it to 06 he will jump higher. The higher you change it, the higher Megaman jumps. It's pretty fun to do. I had help from fellow tekhacks member DahrkDaiz and another member named Gil_Galad I met in acmlm's mIRC #romhacking channel. I used a Game Genie code converter. Open it up and select NES. Enter the code and you should get the address and the bytes. It will give you the address it's at in memory and the byte it is and the byte you need to change it too. Now the address you have is not the address that needs changing. You must add 10 bytes to the address for the header. This next part is a little tricky. You must use the mathematical operation "AND" to the new address. So if you got the hex address 9123 in your Game Genie Code Converter, you would open your calculator that comes with windows. For the AND operation, you always use the hex address 1FFF. I don't know why it is, but it just is. I believe that it also has to deal with RAM moving in and out of the rom. If you do 9123 AND 1FFF you get 1123. Now never forget to add 10 bytes for the header. After that you get hex address 1133. Now, because of something called banking you must search for the address by adding 2000 or 8000 bytes. Sometimes you can just add 8000 bytes and you got your address, but sometimes different roms use different amounts of banking. So just add 2000 bytes to the address until you find the desired result. For some Game genie codes, you don't get a starting byte value. You get a byte to change it to and not a byte that you have to look for. So just do what I said before and just change the byte held at every 2000 or 8000 bytes to the one that Game Genie Code Converter gave you. You will eventually come up with the result you want.

3) Hex to Snes address conversion and Pointer Info
by Unknown Author and PuzzleDude

(all hex addresses are for NON headered rom)

****
WRITTEN BY UNKNOWN AUTHOR

This chart is for standard lo-roms only:

HEX----------SNES (Lorom format)

000000-007FFF--------008000-00FFFF
008000-00FFFF--------018000-01FFFF
010000-017FFF--------028000-02FFFF
018000-01FFFF--------038000-03FFFF
020000-027FFF--------048000-04FFFF
028000-02FFFF--------058000-05FFFF
030000-037FFF--------068000-06FFFF
038000-03FFFF--------078000-07FFFF
040000-047FFF--------088000-08FFFF
048000-04FFFF--------098000-09FFFF

050000-057FFF--------0A8000-0AFFFF
058000-05FFFF--------0B8000-0BFFFF
060000-067FFF--------0C8000-0DFFFF
068000-06FFFF--------0D8000-0CFFFF
070000-077FFF--------0E8000-0EFFFF
078000-07FFFF--------0F8000-0FFFFF
080000-087FFF--------108000-10FFFF
088000-08FFFF--------118000-11FFFF
090000-097FFF--------128000-12FFFF
098000-09FFFF--------138000-13FFFF

0A0000-0A7FFF--------148000-14FFFF
0A8000-0AFFFF--------158000-15FFFF
0B0000-0B7FFF--------168000-16FFFF
0B8000-0BFFFF--------178000-17FFFF
0C0000-0C7FFF--------188000-18FFFF
0C8000-0CFFFF--------198000-19FFFF
0D0000-0D7FFF--------1A8000-1AFFFF
0D8000-0DFFFF--------1B8000-1BFFFF
0E0000-0E7FFF--------1C8000-1CFFFF
0E8000-0EFFFF--------1D8000-1DFFFF

0F0000-0F7FFF--------1E8000-1EFFFF
0F8000-0FFFFF--------1F8000-1FFFFF
100000-107FFF--------208000-20FFFF
108000-10FFFF--------218000-21FFFF
110000-117FFF--------228000-22FFFF
118000-11FFFF--------238000-23FFFF
120000-127FFF--------248000-24FFFF
128000-12FFFF--------258000-25FFFF
130000-137FFF--------268000-26FFFF
138000-13FFFF--------278000-27FFFF

140000-147FFF--------288000-28FFFF
148000-14FFFF--------298000-29FFFF
150000-157FFF--------2A8000-2AFFFF
158000-15FFFF--------2B8000-2BFFFF
160000-167FFF--------2C8000-2CFFFF
168000-16FFFF--------2D8000-2DFFFF
170000-177FFF--------2E8000-2EFFFF
178000-17FFFF--------2F8000-2FFFFF
180000-187FFF--------308000-30FFFF
188000-18FFFF--------318000-31FFFF

190000-197FFF--------328000-32FFFF
198000-19FFFF--------338000-33FFFF
1A0000-1A7FFF--------348000-34FFFF
1A8000-1AFFFF--------358000-35FFFF
1B0000-1B7FFF--------368000-36FFFF
1B8000-1BFFFF--------378000-37FFFF
1C0000-1C7FFF--------388000-38FFFF
1C8000-1CFFFF--------398000-39FFFF
1D0000-1D7FFF--------3A8000-3AFFFF
1D8000-1DFFFF--------3B8000-3BFFFF

I'm sure you see the pattern now. So let's say you have a text block located at $080600. Subtract the header, so it becomes $080400, and then look it up on the chart and find its new address. so $080400 is actually $108400. Now split it up into pairs -> 10 84 00, and re-arrange them backwards -> 00 84 10. Now you'll have to search for that, and remember, it might appear more than once in the rom, so the best thing to do change all of these addresses that appear in the rom to the new one.

A little hint: These kind of just USUALLY have "BF" before them, so 00 84 10, might look like BF 00 84 10.

****
****

ADDED BY PUZZLEDUDE

It's much easier to think of the HEX and SNES addresses, if you consider one to be the actual one. The "actual" address is a HEX address or hex offset in the rom. This is also known as the PC address, or the address, the way the humans (that's

us, lol) could read a certain spot in the rom.

Obviously a human would start at 0. And if the data is at 27502, then this is 27502 in hex from 0. This is the normal PC address, or the hex address.

But the SNES is a "machine", that reads it in a dumb way. It starts at 008000. So 0 in hex is 008000 for SNES. And 27502 in hex is then 04F502 in SNES. Even more dumb is a way this address is read (by SNES CPU) from a file. It is little endian = reversed.

So what a human would understand as 27502, you have to tell the machine as 02 F5 04. Because you enter the address in bytes, so you need 3 bytes to tell the machine the address. It will then read the 3 bytes 02 F5 04 as a SNES address 04F502, which is 27502 in hex, so it will know to go to 27502.

SePH wrote
Are assembly addresses the same as snes cpu addresses?

ASM addresses are SNES addresses (luckily they are not reversed, so they are not for CPU, but for an Assembler like xkas, who converts the whole code into bytes, for the SNES CPU to read later).

Example by XaserLE (overlays asm) = ORG $3F9180; go to expanded space which is at 1F9180 as hex offset. While this address is RAM = LDA $7EF3C5; load game state. ASM can do all sorts of stuff, not just addressing.

SePH wrote
Is SNES lorom the same thing as a hex address?

Not really. Lorom is just formatting. This is the method you would organize your data in a file; while hex address is a certain spot in this data. So hex address can be a spot in the file of any format: LoROM, HiROM, ExLoROM etc...

What Euclid meant as select Lorom is because most Zelda3 games are Lorom. So at Lunar address or SNEStuff (I use this address calculator), you have multiple options for format. So you need to select Lorom before calculation.

1DE1C3 ~ 0xEE1C3 unheadered rom
SNES-----HEX

1DE1E5 ~ 0xEE1E5 unheadered rom.
SNES-----HEX

This is calculated with Lunar Address.

But SNES address 1DE1C3 is usually written as C3 E1 1D as 3 bytes in the rom. (Machine, or rather the SNES CPU will read it this way). It is however written normally (1DE1C3) in ASM.

But a human does not need any bytes to determine an address and is not dumb enough to read it backwards. When we see 0xEE1C3, we know it is on EE1C3 in the rom (no bytes needed, just the brain, lol).

LIST of addresses:

-HEX or PC or ("human") = 27502
-SNES (not reversed, for ASM) = 04F502
-SNES (reversed for CPU) or ("machine") = 02 F5 04
So 27502, is 04F502 for SNES, is 02 F5 04 as bytes.

-RAM = 7EF3C5 (this is NOT connected to the upper 3)

It's all about human VS the machine.

****

When converting HEX to SNES the last 3 digits are always the same. The 4th digit (from right to left) is sometimes changed, sometimes not, the same for first 2.

HEX TO SNES (UNDERSTANDING THE CONVERSION)

Example1 (EASY bank)
Hex address Snes address
028253---------058253

Example2 (COMPLEX bank)
Hex address Snes address
027253---------04F253

EASY BANK

An easy bank is the 8000 bank in hex that goes from 8000-FFFF.
The last 4 digits remain the same (--8253), while the first two (02----) change as presented in this table: So 02 will change to 05 If the left increases normally (00, 01, 02), factor is +01, starting at 00 the right increases doubled, starting with 01 = (01, 03, 05), factor is +02, starting at 01.

00---01
01---03
02---05
03---07
04---09
05---0B
06---0D
07---0F
08---11
09---13
0A---15
0B---17
0C---19
0D---1B
0E---1D
0F---1F

ETC

formula: (current address times 2) + 1, so 09 x2 = 18dec = 12 hex + 1 = 13. So 09 --> 13

COMPLEX BANK

027253 = example

A complex bank is the 8000 bank in hex that goes from 0000-8000. The last 3 digits remain the same (---253), while the first two (02----) change as presented in this table: So 02 will change to 04 If the left increases normally (00, 01, 02), factor is +01, starting at 00 the right increases doubled, starting with 00 = (00, 02, 04), factor is +02, starting at 00.

00---00
01---02
02---04
03---06
04---08
05---0A
06---0C
07---0E
08---10
09---12
0A---14
0B---16
0C---18
0D---1A
0E---1C
0F---1E
ETC

formula: (current address times 2), so 09 x2 = 18dec = 12 hex = 12. So 09 --> 12

The THIRD digit (--7---) will change as presented in this table: So 7 will change to F If the left increases normally (0, 1, 2), factor is +1, starting at 0 the right increases normally, starting with 8 = (8, 9, A), factor is +1, starting at 8.

0---8
1---9
2---A
3---B
4---C
5---D
6---E
7---F
formula: (current address plus 8), so 4 +8 = 12dec = C hex = C. So 4 --> C

Let's convert a random Hex address into Snes manually, no use of calculator:

106781 (this is just over 1MB).

last bank is 6000 so between 0 and 8000 = complex bank.

-last 3 digits will stay ---781

-first 2 digits are 10
formula: (current address times 2), so 10 hex x2(hex/dec) = 20 hex
-first 2 digits will convert to 20

-third digit is 6
formula: (current address plus 8), 6 +8 = 14dec = E in hex
-third digit will convert to E.

So SNES address is 20E781.

****

Lets convert another random Hex address into Snes manually, no use of calculator:

6A2EB = 06A2EB

last bank is A000 so between 8000 and FFFF = easy bank.

-last 4 digits will stay --A2EB

-first 2 digits are 06
formula: (current address times 2) + 1, so 06 hex x2(hex/dec) +1 = 0D hex
-first 2 digits will convert to 0D

So SNES address is 0DA2EB.

****

For the CPU version of the SNES address break it up in pairs and reverse them.

So SNES address 20E781 = 20 E7 81 = reversed = 81 E7 20
So SNES address 0DA2EB = 0D A2 EB = reversed = EB A2 0D

CPU SNES addresses are 81 E7 20 and EB A2 0D. This is the way any address is presented in the ROM file (read only memory that the SNES CPU reads).

But the normal SNES addresses like 20E781, are usually read by Assemblers to produce the code later on.

****
****

POINTERS

What is a pointer, you ask? Nothing complex. These are all of those bytes that actually contain an address, usually with the command Load infront.

See this: BF 81 E7 20, this is a pointer. It means load/look up in SNES address 81 E7 20 = reversed = 20 E7 81 = 20E781. Converted to hex this is 106781 (calculated with Lunar address directly).

(Knowing this you can actually do ASM with HEX, nice isn't it. But it is just easier to make a normal Asm file and let Xkas do the conversion thing.)

This process of converting is the opposite of the one presented in the previous chapter.

Sometimes a pointer like that can point to an address with more pointers. So at 106781 we can have more pointers in a row, making them secondary pointers, while the one after the BF is a primary one.

Sometimes there are more primary pointers, pointing at the same address or close to it. All the above are whole or 3 bytes pointers or global pointers = any address can be defined with them.

Other type are local pointers or 2 byte pointers, since the third bytes will always be the same.

81 E8 20, 62 E9 20, 35 EA 20, gets reduced to 81 E8, 62 E9, 35 EA (third value is 20 for all).

81 E8 is E881 = would be 006881, but with 02 it makes 106881.

Sometimes the global bank is hardcoded and the 2 byte pointer defines an offset from the global bank. So 81 E8 can mean the offset from 100000 to 106881, or an offset from 40000 to 46881.

Local pointers are used if all the data is in one bank or close together to save space, with the repeated third byte.

Global pointers are used for larger data blocks.

4) Vital dungeon hex data

by PuzzleDude

After about a week of hex study, here are my findings. Hope this will help you all on your Zelda3 hacking journeys. You may use this document for repointing data, porting data between games, having more room for data; make use of rooms 296-319 etc.

I hope someone, who is more experienced in coding/programing, can use this data to make some sort of utility (program) to be used with Hyrule Magic or after using HM. With this hex info you can use maximum possible header (for all 320 rooms), the entire 8000 bank for items (calculates to 33 items per room, for 320 rooms = more than you'll ever need), and can repoint room object data to have "no limit" = all 320 rooms can be filled this way (but with sanity limit that the emulator will allow, which is around 300 in hex). But we are still stuck with the "regular" space for sprites, chests, blocks, torches etc.

What would also be nice would be to have some sort of simple utility to recognize room data pointers at F8000, read them, and then export the room data separately (since the rooms are somewhat mixed and filled-up into the file). Putting the data in into new locations is easy due to the possibility of precalculated pointers, which can use round up addresses.

After the repointing of any data the file is no longer compatible to Hyrule Magic!
The addresses are fixed in Hyrule Magic. The program will not read new pointers!

All hex offsets are for a NON-headered rom and for Zelda3 A link to the Past.
Hacks can have the data shifted!

Room data (Object data for BG1, BG2 and BG3)

-bank not fixed (can be reallocated)
-3 byte pointers

***Primary pointers at:**

1. 8746, 01 80 1F --> F8001
2. 874C, 00 80 1F --> F8000
3. 883F, 01 80 1F --> F8001
4. 8845, 00 80 1F --> F8000 (all correspond to F8000)

For reallocation change all pointers accordingly.

***Secondary pointers at:**

F8000 (block is 3C0) = 960 in decimal => 960:3 = 320 --> for 320 rooms (room 0 to room 319)

(3 byte pointers for one room)
***Pointer Read:**
6A 84 0A = reversed **0A 84 6A** --> points to 5046A

(address calculator needed for determine this)

E2 90 0A = reversed **0A 90 E2** -->points to 510E2
etc

***Room code:**

start (pointer points to it)....bg1....FF FF.....bg2.....FF FF.....bg3.....FF FF. (end)

Data are objects: x position, y position, size, type.
Data holds room layout (2nd byte) and floor1 and floo2 value (1st byte).

***No bg2**
start (pointer points to it)...bg1...FF FF FF FF.....bg3.....FF FF. (end)

***No bg3**
start (pointer points to it)...bg1...FF FF.....bg2.....FF FF FF FF. (end)

***Empty room**
start (pointer points to it)...bg1 basic data...FF FF FF FF FF FF. (end)

***End of room data is always the third FF FF value after pointer!**
(there are some minor exceptions though).

***All pointers are 3bytes long** = full address can be defined. Reallocation of room data to anywhere between **0** and **400000** is possible.
Reallocation of secondary pointers is also possible.
****Reallocation not compatible with Hyrule Magic****

Sprite data (Indoors)

-the global bank is somehow stuck at **40000**.

***Primary pointer at:**
4C298, but 2 bytes only = **2E D6**
code is:

B9 **2E D6** 85 00 AD 8E 04 4A

No third value 09, to make it 2E D6 09 = 09 D6 2E = 4D62E.
no value is 09 to define 40000. If anyone can find the value, which defines 40000, let me know.

***Secondary pointers at:**
4D62E (block is 300) = in dec is 768:2 = 384 (for 320 room + some additional data).

***Data is at:**
4D92E (block is 1371) = ends at **4EC9E**
***Pointer read:**
2E D9 = reversed D9 2E = 09 D9 2E --> points to **4D92E** = room 0.

***Sprite Code:**
for room 0 = Ganon
00 05 17 D6 FF

00 = sort sprite

05 = y

17 = x

D6 = type (Ganon)

FF = end

The **05** is not only y coordinate, but also: if the sprite is on bg1 or bg2.

***Empty room (no sprites)**
00 FF

Comparison to Goddess of Wisdom hack
*bank shifted, at **4C298** is DB CE, so sec. pointers at **4CEDB***
*but data end also at **4EC9E**.*

Comparison to Parallel Worlds hack
*bank shifted, at **4C298** is 6C D4, so sec. pointers at **4D4C6***
*but data end also at **4EC9E**.*

So sprites cannot be seen in HM for PW, because they were shifted.

Item data (Indoors)

-These are red dots in HM, to define stuff under jars (heart, magic, key, bomb etc)

***Primary pointer at E6C1**
Code: BF **69 DB 01**, 85 00, A9 **01**

69 DB 01 points to **DB69**

01 defines the global bank. Can be reallocated anywhere between **0** and **400000**.
For instance BF 69 DB **23**, 85 00, A9 **23**; new address is **11DB69**.

***Secondary pointers at:**
DB69 (block is 282), 282 is 642 in dec :2 = 321 rooms (for 320 rooms + 2 bytes)

***Data at:**
DDEB (block is 8C7)

***Pointer read:**
EB DD reversed is DD EB = direct address = DDEB

***Item code:**
CE 13 0B FF FF

CE = x

13 = y

0B = type

FF FF = end

A jar object is necessary at the same location for the item to work!

Pushable blocks data (Indoors)

***Number of bytes to define block data at:**
8896,
default is 8C 01 = 18C = 396 bytes :4 = 99 blocks, but
maximum is 00 02 = 200 = 512 bytes :4 = 128 blocks (29 more).

max was tested, 04 02 = block not displayed, too much data.
So 80 in hex for one pointer is max.

***Primary pointers at:**

1. 15AFA, DE F1 04, --> 271DE (but with hardcoded block = 80)
2. 15B01, 5E F2 04, --> 2725E (but with hardcoded block = 80)
3. 15B08, DE F2 04, --> 272DE (but with hardcoded block = 80)
4. 15B0F, 5E F2 04, --> 2735E (but with hardcoded block = 0C by default, 80 max)

Pointers can be changed to another place, but it will only read the 80 bytes in hex after this location.

***No secondary pointers, but Data directly at:**

271DE (block is 80+80+80+0C) = 18C = 396 in dec :4 = 99 pushable blocks.
max block is 80 x 4 = 200 in hex.

***Code (4bytes for one block)**
0B 01 3A 19 (no ending value! FF), room values are mixed!, so the space must be hardcoded, that the game knows where to stop reading values.

0B 01 is 10B = room 267. 3A is x, 19 is y.

Want to insert a block in hex, let's say the x=08, y=0A in Hyrule Magic.

x times 2 is 08 x 2 = 10 in hex
y divided by 2 is 0A : 2 = 05 in hex

So x value will always be bigger than y.

let's take room 80 for instance, 80 is 50 in hex.

new code is 50 00 10 05 for block in room 80, on HM coordinates x=08, y=0A.

Torches data (Indoors)

***Number of bytes to define torch data at:**
88C1,
default is 2C 01 = 288 bytes, but
maximum is 80 01 = 384 bytes. (around 28 torches more)

max was tested, 8C 01 crashes the game, too much data for torches.
So 80 in hex for one pointer is max.

***Primary pointers at:**

1. 15B16, 6A F3 04 = 04 F3 6A = 2736A (block 80)
2. 15B1D, EA F3 04 = 04 F3 EA = 273EA (block 80)
3. 15B24, 6A F4 04 = 04 F4 6A = 2746A (block 20, max 80)

If 04 is changed to 24, new data is at 12736A and at 88C1 to 80 01,
the block can be 384 bytes long.

***No secondary pointers, but Data directly at:**

2736A (block is 80+80+20) = 12C = 288 in dec.
max block is 180 in hex

1 torch in a new room is 6 bytes, then +2 bytes for each torch.
2 torches is 8bytes, 3 is 10 and 4 is 12 etc.

***Code (example)**
43 00, AA 03, B6 03, AA 09, B6 09, FF FF

43 00 = room 67,
AA is x, 03 is y for the first torch
B6 is x, 03 is y for the second torch etc
FF FF is end.

The upper room has 4 torches.

Room header properties

***Primary pointer at B5DC**
Code BF, **02 F5 04**, 85 0D, E2 20 C2 10, A9 **04**

02 F5 04 points to **27502**

04 defines the global bank. Can be reallocated anywhere between **0** and **400000**.
For instance BF, 02 F5 **24**, 85 0D, E2 20 C2 10, A9 **24**; new address is **127502**.

***Secondary pointer at:**
27502 (block is 280)
,
27782 (block is 87E)

***Pointers + data**
27502 (block is AFE)

***Pointer read**
82 F7 = F7 82 = 04 F7 82 = 27782

***Header code (14 bytes maximum)**

1st byte,
bg properties + collision

2nd byte, 3rd byte, 4th byte, 5th byte, 6th byte, 7th byte
PAL BLK EnemyBlk Room effect Tag1 Tag2

8th byte
Plane properties for hole/warp and staircase 1, 2, 3

9th byte
Plane properties for staircase 4

10th byte, 11th byte, 12th byte, 13th byte, 14th byte.
hole/warp staircase1 staircase2 staircase3 staircase4.

This is essentially related to room transit values as a part of a room header.

Q: *What defines which room uses which header?*

A: To recognize the order of the rooms, you need to read the pointers at 27502 (27702 headered). First two pointers are for room 0, second two for room 1 etc. But if you use the Headered rom, read the pointer, for instance 82 F7 = F7 82 = 04 F7 82 = 27782, and add 200 bytes = 27982 in your case, for room 0. Then next pointer for room 1. Start of room1, one byte before is the end of room0.

In your case, the first pointer points to
20 18 0E 22 07 3D 00 00 00 10 (4 bytes missing to maximum)

The next pointer points to where the C0 is, so 10 is end room 0.
The C0 is the typical BG properties on "Normal".

You can mix the data though, but Alttp tends to have all the data from room 0-319. But all the pointers are hardcoded from 0-319.
Your case of room0:
20 is parallaxing BG
18 is 24 pal
0E is 14 blk
22 is 34 eblk (this is eblk for Ganon)
07 is effect (Ganon room effect)
3D is tag1 (kill to open Ganon door tag)
00 is tag2 (empty = "Nothing")
00 is plane(empty or set to 0)
00 is plane(empty or set to 0)
10 is room16 hole/warp (drop down from Ganon)

end of room0, because next pointer to C0, C0 is BG properties on "Normal" for room 1.

****If repointed, you can use 14 bytes per room, but HM does not read this****

Essentially it is better to keep the header limit and everything is compatible. But the upper info can be used if putting multiple rooms/dungeons together in hex and needing more header space. Or to port header data between games.

Q: *Alright so, 10 bytes per block for the header (without repointing) and I have space to create all 319 room headers in that block?*

A: Not exactly 10 bytes. You have 87E (hex) block for 320 rooms. But maximum for one room is 14 (dec). Optimal space for header would be 14 bytes times 320 rooms = 4480 in dec or 1180 in hex. (Black Magic already uses this). Compare it to 87E. 87E is only 2174 in dec, so twice as small.

Every room uses a different amount of bytes for header. Room 0 uses 10 bytes, room 1 uses 14 (maximum), room 2 uses 11, while room 3 only 7 (minimum). Every room must have at least 7 bytes for header. Calculated average = 2174 in dec :320 rooms = only 6,8 (under minimum), that's why only 295 rooms are used. 2174 :295 = 7,3 per room. Only 0,3 for special effects. Since the 8th and the 9th bytes are only plane info, you need 10 bytes for actual hole/warp effect.

So 3 rooms must have a minimum 7 bytes header, so that the 4th room can have 10 bytes (defining the hole/warp).

This basically means if you want to use 320 rooms, some rooms must have the same header, bringing the average from 6,8 to at least 7,1.

Chest definitions

***Primary pointer at:**

1. EBFB, 6E E9 01 = E96E
2. EC09, 70 E9 01 = E970
3. EC0F, 6E E9 01 = E96E all for E96E

Repoint all 3 accordingly. 01 to 23 is 11E96E for instance.

max bank is 1F8 = 504 in dec : 3 for one chest = 168 chests possible in the game.

***No secondary pointers, but Data directly at:**
E96E (block is 1F8)

***Code is: (3 bytes for one chest definition)**
04 00 23, 04 00 = room 4, 23 is a content of the chest.
23 in hex is 35 in dec is 35: Armor 3 in Hyrule magic.

04 80 23, 04 00 = room 4, but 80 is big chest, 23 is Armor 3.

04 01 23, 04 01 is 104 in hex is room 260, 23 is Armor 3.

***Definition will only work with the chest object in the same room!**

***Problems**
Sometimes the definition is set, but no chest object is inserted. Insert chest in the rooms

with chest definitions to solve this problem.

Damage pits (Indoors)

***Couldn't find any pointers here**

***Data at 190C (block is fixed to 72)**

72 in hex is 114 in dec :2 = 57 rooms with pits are possible.

2 bytes for a pit definition.

***Code**:
72 00 = room 114

No pointers, fixed location and block size. No FF ending, just a list of rooms in hex.

Telepathic messages (Indoors)

***Couldn't find any pointers here**

***Data at 3F61D (block is fixed to 280)**

280 in hex is 640 in dec :2 = for 320 rooms. Rooms go from 0-319.

2 bytes for a message definition.

***Code**:
72 00 = message (monologue number) 114

No pointers, fixed location and block size. No FF ending, just a list of message values in hex.

Basic starting location properties

***Wall blockset for entrances:**
15381 (block is 85 for 85 entrances in hex)

***Music definitions for entrances:**
1582E (block is 85 for 85 entrances in hex)

***Dungeon definition for entrances:**
1548B (block is 85 for 85 entrances in hex)

FF is no dungeon or None. Hyrule magic can make a false byte FE,
change it manually to FF to fix this.

5) Dungeons items study
By PuzzleDude

(avoid any items outside! if using Hyrule Magic)

* = most in use

```
TYPE:---------------WHAT YOU GET---------------------INSIDE / OUTSIDE

-00-nothing----------nothing--------------------------inside and outside item
-01-heart------------green rupee (not a heart)--------inside and outside item
-02-rock crab--------rock crab (bugged gfx indoors)---outside item
-03-bee--------------bee appears----------------------inside and outside item
-04-random-----------rupee/heart/fairy etc------------inside and outside item
*05-bomb-------------one bomb-------------------------inside and outside item
*06-heart------------falling heart--------------------inside item
-07-blue rupee-------5 rupees--------------------------inside and outside item
*08-key--------------small key-----------------------inside item
-09-arrow------------5 arrows--------------------------inside item

-0A-bomb-------------one bomb (same as 05), rep-------inside and outside item
*0B-heart------------heart standing on the floor------inside item
*0C-magic------------small green magic----------------inside item
*0D-big magic--------full green magic-----------------inside item
-0E-chicken----------chicken (gfx must be accurate)----inside item (in house)
-0F-green soldier----green soldier appears------------inside item

-10-alive rock-------rock that starts jumping---------outside item (dark world)
-11-blue soldier-----blue soldier appears-------------inside item
-12-ground bomb------mine on the floor----------------outside item
-13-heart------------green rupee (not a heart), rep---inside and outside item
*14-fairy------------fairy appears--------------------inside item
-15-heart------------falling heart (same as 06), rep--inside item
-16-nothing----------nothing (same as 00), rep--------inside and outside item

-80-hole-------------a hole in ground under bush------outside item
-82-warp-------------a warp under the rock------------outside item
-84-staircase--------a staircase under big rock-------outside item
-86-bombsble---------bombable cave or house entrance--outside item
*88-switch-----------a switch for doors---------------inside item
```

Coordinates

Item can be placed on a "accurate" = "small" coordinate. The definition of this coordinate is much more accurate, since it has a full amount of values, going from 00 to 3E in hex = 63 small units. Item can be placed on the 64 small units as well but this is Not recommended. So 64 small units in theory. = 8x8!

64 divided by 2 = 32. One Zelda3 dungeon room has 32 squared (normal) units, one of those units is a wide as a jar (pot), or a small chest. So a small chest is one unit wide and one unit long = 1x1 unit or 2x2 small units.

Calculation
Using Hyrule magic as a coordinate calculator, we can traverse the thing into hex code.

Let's take an item on the coordinate X = 23, Y = 2E (both in hex); 35, 46 in dec.

This is 35 small units' right and 46 small units down, since 00, 00 is a far left-upper corner.

To calculate the hex value in the actual rom, you must multiply X with 2, and divide Y by 2.

23 in hex, times 2 in hex = 46 in hex
2E in hex, divided by 2 in hex = 17 in hex

So the code for bomb item on position 23, 2E in Hyrule magic is:
46 17 05 FF FF in hex in the actual rom.

Code in the rom coordinates
00-7C, one small unit counts as 02, one normal unit as 04 --- this is for X

00-1F, 2 small units count as 01, one small unit is actually written as 80 00 (x,y). adds a +80 factor on X.

---
complex example: X = 17, Y = 17.

17 in hex, times 2 in hex = 2E in hex
17 in hex, divided by 2 in hex = 0B in hex

But y must be even, 17 is odd, so the +80 factor will be added on X, if the Y is odd. The +80 on x = +80 00 = nothing added on y, 80 on x. No factor is added on X if the Y is even.

2E + 08 = AE, final result for (17, 17) = AE 0B

2E 0B would be x = 17, y = 16.

BG factor

If an item is put on bg2, a +20 is added on the y coordinate = +00 20 (nothing added on x, 20 on y). Ironically if you put an item on bg2 with Hyrule magic, you can no longer touch it. You just have to go to hex and reduce the y for -20 again (Or, use "remove all" items in the room and start all over, it shouldn't take long). For instance from 3C, to 1C.

Let's take a look at the most complicated case (item on bg2 and an odd y, + high values):

bomb item on x = 1D, y = 3B, on bg2
(3B = 59 in dec = odd)

1D x2 = 3A (all in hex)
3B :2 = 1D
05 = item type = bomb

3A 1D 05 so far

+80 00 factor because of the odd y = BA 1D 05
+00 20 factor because of the bg2 = BA 3D 05

Final result = BA 3D 05 FF FF
for the bomb item on x = 1D, y = 3B, bg2

Empty string for debugging bugged items

Copy paste this at **DB69** (no header) to *actually* remove all items. The item code gets "purified". It is completely empty and clean.

```
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD
E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD E9
DD E9 DD E9 DD E9 DD E9 DD E9 DD E9 DD FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

```
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

At **DB69** (with block 280 in hex) we have a number of pointers, pointing at **DDE9** = 2 bytes reserved for empty room = FF FF. Then comes **DDEB** (block 87C).

The entire data is **DB69** (block B49 long = 2889 in dec).

This is something that HM fails to do. When you select remove all items in the game, only the pointers will change, but not the data itself. The same for a command *remove all* in the room. Just the pointer is changed, while the data remains. But you see a room with no items. So you obviously want to add them, not knowing, that the game actually has no room left. So adding new on already filled data block makes the bugs.

Using any remove all functions is therefore not recommended.

Using Hyrule magic as an Item position calculator

Make a copy1 of Alttp, go to **DB69** and paste in the upper code. Go to room 0 (Ganon room), empty all objects, sprites, torches etc. Fill the entire room (everything to the very edge) with floor, such as appears in room 0 (you know, that squared floor). This will give you the feeling of the x and y in the room. Open copy1 in Hex editor and go to **DDEB**.

Now make a copy2 of Alttp, and go to room 117. You have 3 items there, magic at 02 AD, arrow at 10 AD and heart (type 0B) at 16 2D. Insert those items into room 0 of copy 1 and save. Click on hex editor and the data will refresh/reload automatically at **DDEB** to your hex code of this item data:
94 16 0C A0 16 09 AC 16 0B FF FF (this is your code) = item code for items in room 117 of Alttp.

To reset to empty, fill this with FF bytes and close HM, and reopen copy1 in HM.

This is how you manually export the item code.
This is very useful if constructing one dungeon in one file (to avoid bugs).

Putting it all together in hex

Let's say the dungeon has 15 rooms. Export items for all 15 rooms, like.

room18
code

room54
code

etc

When all dungeons are done, the code is put all together in order for all 295 or 320 rooms, starting with room 0. Must not be longer than 8C7, otherwise repoint the primary pointer at **E6C1**, Code BF 69 DB 01, 85 00, A9 01 (change both 01 to for instance 23, to have data at **11DB69**)

Room0
code

room1
code

etc

Now everything gets erased except for the code, which is then inserted at **DDEB** of the final file with empty item block. Empty space in the txt will be ignored by hex editor.

Hex display of rows is then put on 1, and searching of the FF FF is repeated. Every byte after FF FF is a starting byte of the new room, and the address is displayed on the left. Code is now vertical.

Like so

```
DDEB 94 start of room0
DDEC 16
... ..
DDF4 FF
DDF5 FF
DDF6 3A (start) because of the FF FF earlier, so DDF6 is pointer position, starting byte of room1.
.....
entire code is vertical
```

Starting addresses are written down in order, **DDEB**, **DDF6**. This is converted to SNES addresses and reversed, in this case it is just reversed, because of the 2 byte local pointers and first 8000 bank.

EB DD, F6 DD etc, These are the pointers. Copy the entire string of pointers to **DB69** (block 280), and that's it. All the items of the entire game, put together from multiple other files, with finished rooms.

If the primary pointer is repointed, you have virtually unlimited items per room (actually you have 8000 in hex, so around 33 items per 320 rooms at maximum = way to much for the emulator to handle; 4 per room is a normal/high average). You can also fill all 320 rooms (from 0 to 319) with that.

At the end of this document we can say, that we have "hacked" the indoor items of Alttp. We can now do whatever we want with them.

6) Overworlds items study

By PuzzleDude

Hex data

*Primary pointer at **DC8B8**,
Code BF, F9 C2 1B, 85 00, A9 1B

F9 C2 1B points to **DC2F9** = start of secondary pointers (2 byte pointers)

1B defines the global bank. Can be reallocated anywhere between **0** and **400000**. For instance BF, F9 C2 3B, 85 00, A9 3B; new address is **1DC8B8** .

*Secondary pointers at:
DC2F9 (block is 100), 100 is 256 in dec, (minus 2 bytes) = 254 :2 = 127 areas = 7F areas, 3F in light world and 3F in dark world = 7F areas.

*Reserved for area with no items:
DC3F9 (block is 2) = FF FF bytes

*Data at:
DC3FB (block is 4A1 long)

*ALL together (Pointers + FF FF + Data):
DC2F9 (block is 5A3 long)

*Pointer read
FB C3 reversed is C3 FB = 1B C3 FB = DC3FB, so pointer FB C3 points to DC3FB.

*Item code
D0 1A 01 FF FF, D0 = x, 1A = y, 01 = type, FF FF = end

A bush or rock object is necessary at the same location for the item to work!

Testing the items

* = most in use

This is the same as in the Dungeon items document, since the same items can be used indoors and outdoors.

```
-TYPE:------------WHAT YOU GET-------------------INSIDE / OUTSIDE

-00-nothing-------nothing------------------------inside and outside item
*01-heart----------green rupee (not a heart)-------inside and outside item
*02-rock crab------rock crab (bugged gfx indoors)--outside item
*03-bee------------bee appears---------------------inside and outside item
*04-random---------rupee/heart/fairy etc-----------inside and outside item
*05-bomb-----------one bomb------------------------inside and outside item
-06-heart----------falling heart-------------------inside item
*07-blue rupee-----5 rupees-------------------------inside and outside item
-08-key------------small key-----------------------inside item
-09-arrow----------5 arrows-------------------------inside item

-0A-bomb-----------one bomb (same as 05), rep------inside and outside item
-0B-heart----------heart standing on the floor-----inside item
-0C-magic----------small green magic---------------inside item
-0D-big magic------full green magic----------------inside item
-0E-chicken--------chicken (gfx must be accurate)---inside item (in house)
-0F-green soldier--green soldier appears-----------inside item

*10-alive rock-----rock that starts jumping--------outside item (dark world)
-11-blue soldier---blue soldier appears------------inside item
*12-ground bomb----mine on the floor---------------outside item
-13-heart----------green rupee (not a heart), rep--inside and outside item
-14-fairy----------fairy appears--------------------inside item
-15-heart----------falling heart (same as 06), rep-inside item
-16-nothing--------nothing (same as 00), rep-------inside and outside item

-17-nothing--------------------empty
-18-
-19-
-1A-
-1B-
-1C-
-1D-
-1E-
-1F-
-20-
-21-
-22-
-23-
-24-
-25-
-26-
-27-nothing--------------------empty
```

-28-nothing----------empty
-29-nothing----------empty
-2A-nothing----------empty
-2B-nothing----------empty
-2C-nothing----------empty
-2D-pullswitch----------spawns next to the bush/pot (right)
-2E-
-2F-
-30-pullswitch----------spawns next to the bush/pot (right)
-31-nothing----------empty
-32-pullswitch----------spawns under the bush/pot
-33-pullswitch----------spawns next to the bush/pot (right)
-34-
-35-
-36-
-37-pullswitch----------spawns under the bush/pot
-38-
-39-
-40-nothing----------empty
-41-
-42-
-43-
-44-
-45-
-46-
-47-
-48-
-49-
-4A-
-4B-
-4C-
-4D-
-4E-
-4F-
-50-nothing----------empty
-51-
-52-
-53-
-54-
-55-
-56-
-57-
-58-
-59-10----------doesn’t work
-5A-
-5B-
-5C-
-5D-
-5E-
-5F-10----------doesn’t work
-60-10----------doesn’t work
-61-10----------doesn’t work
-62-10----------doesn’t work
-63-nothing----------empty
-64-10----------doesn’t work
-65-4wayoctorok----------spawns next to the bush (right)
-66-10----------doesn’t work
-67-nothing----------empty
-68-nothing----------empty
-69-nothing----------empty
-6A-nothing----------empty
-6B-10----------doesn’t work
-6C-nothing----------empty
-6D-nothing----------empty
-6E-nothing----------empty
-6F-moldorm----------crashes the game
-70-roller4----------doesn’t work
-71-greenlizard----------doesn’t work
-72-anarrow----------doesn’t work

```
-73-lynel---------------------crashes the game
-74-octorok-------------------spawns to the far right
-75-hoppingbulbplant----------spawns under the bush
-76-leever--------------------crashes the game
-77-whirlpool-----------------spawns to the far right
-78-cucumber------------------spawns to the far right
-79-thief(flute boy father)----spawns next to the bush (right)
-7A-octorok-------------------spawns to the far right
-7B-lynel---------------------crashes the game
-7C-medallian tablet----------crashes the game
-7D-apples--------------------doesn't work
-7E-vermin--------------------crashes the game
-7F-chicken-------------------spawns under the bush

*80-hole-----------a hole in ground under bush-----outside item
-81-hole-----------
*82-warp-----------a warp under the rock-----------outside item
-83-warp-----------
*84-staircase------a staircase under big rock------outside item
-85-staircase------
*86-bombable-------bombable cave or house entrance-outside item
-87-bombable-------
-88-switch---------a switch for doors--------------inside item

-89-switch--------------------spawns under the bush
-8A-8A------------------------unknown tile (???)
-8B-8A------------------------unknown tile (???)
-8C-8C------------------------unknown tile (???)
-8D-8C------------------------unknown tile (???)
-8E-8E------------------------unknown tile (???)
-8F-8E------------------------unknown tile (???)
-90-hole----------------------spawns under the bush



-A0-hole
-B0-hole
-C0-hole
-D0-hole
-E0-hole
-F0-hole
-FF-8E
```

Coordinates

Big area

start value -->x = 00, y = 00

On X, one unit in HM is 02 units in hex, max X = 7E. But 7E covers a pair 7E, 7F, the way the 00 covers the pair 00, 01, so that the next value is 02. So 7E corresponds to 64 units. This is Not a normal hex to dec conversion!

Hex string
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F = 16 bytes.
The X rises like so: 00, 02, 04, 06, 08, 0A, 0C, 0E = 8 (in dec) units,
This gets repeated 8 times, so max value is 7E = corresponds to 64 units.

On Y, two units in HM is 01 unit in hex, max Y = 1F.
The Y rises like so: 00, 00, 01, 01, 02, 02, 03, 03 = 8 (in dec) units,
This gets repeated 8 times, so max value is 1F = corresponds to 64 units.

The entire grid is therefore 64 by 64 units to place the item.

On Y, one unit adds a +80 00, so +80 on x (odd Y)
On Y, two units add a +00 01, so +01 on y (even Y)

So the machine really doesn't see it as X or Y. It knows to start at 00, and can have 7E jumps to the right, 7E covers 7F, so the second line is 80 on X to correspond what a human would see as one unit of Y. So the second line goes from 80 to FE (covers FF), when finally switching to 100 in hex. But the CPU reeds it reversed, namely 00 01.

So the X and Y are simplified! It is really Not X and Y, it is just one value, rising from 00 to 3F in one line, or from 00 to FF in two lines, reaching 100 in the third line.

But reversed it can be simplified as Y = 1, since 100 is 0100 is 00 01 reversed. This is why an odd and even Y must be defined. Only the lines 00, 02, 04 ... = even lines, will display the Y correctly, the odd ones: 01, 03, 05 ... will actually add a +80 on X to define a Y. Now we all know why this is so.

Small area

One would think the 4 lines are necessary to reach the value 100. This would really be complicated, and would have to drop the X and Y simplification, since every 4th line would only be accurate. Luckily this is not so. Every new line still starts with 80 00 offset, just the first line has a 3E limit, rather than 7E.

So it is the same thing as a big area, but the max X = 3E, corresponds to 32 units. The max Y is then 0F, corresponds to 32 units.

The entire grid is therefore 32 by 32 units to place the item.

So the X simply doesn't use the 40 to 7E values, and the Y doesn't use the 10 to 1F values, making the area 4 times (not 2 times!) smaller. So 64x64 is 4 times bigger than 32x32.

Example of the coordinate calculation

Example is D0 1A, this can be seen as X = D0, Y = 1A, or as a value 1AD0.

Because X = bigger than 7E, an 80 00 must be reduced, so X = 50, divided by 2 = 28, 40 in dec = 40 units to the right, out of 64 units.

Y is 1A (odd Y) = 1A times 2 = 34 (but +1, because odd) = 35, 53 in dec = 53 units down, out of 64 units.

This actually refers to the jump! rather than actual value, so 1 is 1 unit jump from 0, 40 units mean 40 jumps by one unit to the right, all together this is 41st unit; and the 53 jumps down is actually a 54th unit on Y = 54th line.

Other strategy: Value 1AD0 = 6864 in dec divided by 128 (64 units times 2) = 53,625. So the jump is 53, which makes it a 54th line = Y, as suggested before. Regarding the column (X):

1000 ~ 625
64 ~ x

x = 40, as suggested before, but equals to the jump, so the actual unit is 41.

So D0 1A, is 41st unit on X, and 54th unit on Y.

This is actually a calculation for the Item type 01-Heart (actually green rupee) in the Area 00 = Forest. The coordinates of this item are D0 1A in hex, so string is D0 1A 01, 01 is type.

Open Alttp in Hyrule Magic and count the units to the right, to the bush, using the global grid. It is in the 21st square, times two is 42. But minus 1, since it is on the left side of the square. So actual unit is 41, or 40 jumps from 0.

Do the same for vertical value. Bush is in the 27th square, but on the lower part of the square, which makes it 27 times 2 = 54th unit.

Empty string for debugging bugged items

Paste (+overwrite) this string (5A3 long) at **DC2F9** in hex to actually remove all items. This will clear the code, so you can start adding items in the overworld. The remove all function in HM doesn't do this, but simply repoints the pointers. The data is still there, and there is no more rooms for items, the result are bugs (items start respawning all over the place). Remove all will of course not remove them, but this lower code will.

Important:
First go to **17FA8**, and change the values to F9 C3 (the old values in Alttp were 94 C8), This is the value of the last 2 byte pointer. If you don't do this, HM will always tell you, no room for items, because the default value is too big (for all areas filled with items).

Note: if you are using the very original Alttp (not ever saved with HM), you must first open it in HM, and save the game, despite no changes. This is the so called initial save (HM will adopt the rom), making the **17FA0** segment filled with new code. Original Alttp (with no initial save in HM) has this area empty (FF bytes).

Then go to **DC2F9** and copy/rewrite the lower code (5A3 long)

```
F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9
C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3
F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9
C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3
F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9
C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3
F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9
C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 F9 C3 FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

```
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

NOTE: If you put an item like heart (type 0B) under the bush, it will NOT always appear! The game is smart enough not to give you the item, so that it feels like it is random for the player. The bushes can also contain random sprites, like soldier in light world or jumping skull in the dark world.

So if you put items like heart, bomb etc under bushes, only the probability of this item under this bush is achieved, or better said, the probability rises. The bush can still be empty! Just note, that the default item is as seen in Hyrule Magic.

Only items like: hole, bombable, staircase (under big rock) and warp (under small rock) always work. Bombable and staircase work only once, then a fixed overlay is put over it (crack wall, or staircase over the big rock).

7) Inventory/HUD editing

by RedScorpion, Euclid, Spane, PuzzleDude and Conn:

a] Inventory Editing

Euclid: *The basic concept for HUD is - tiles (in the snes vhoppptt tttttttt format) are written in* **7EC700 - 7EC84F** *(for play screen) and* ***1100 to 177F*** *(for subscreen)*

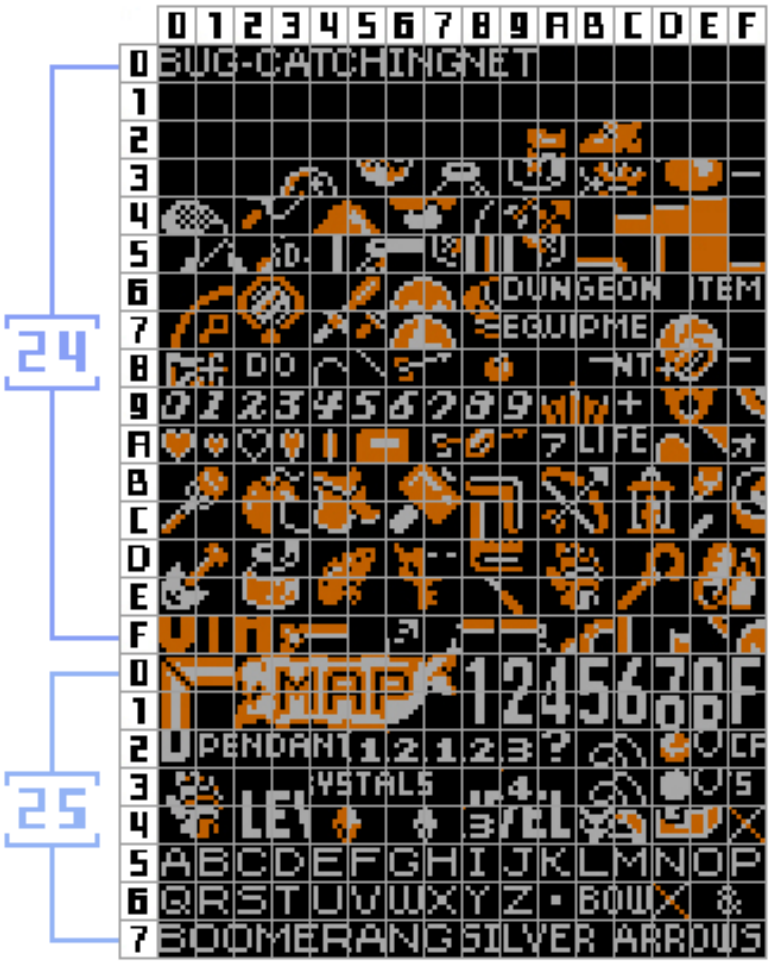
For the subscreen all components are done via assembly/code. Note this is pure disassembly, for what the actual memory address does refer to MathOnNapkins RAM Addresses.

sub screen

```
Subscreen disassembly details

Euclid
10/6/2012

Entrance routine is from this point - do not change this location.

$0D/DDAB 20 99 E3 JSR $E399 [$00:E399] ; unsure exactly what this does yet.
$0D/DDAE A9 01 LDA #$01
$0D/DDB0 20 C8 E3 JSR $E3C8 [$00:E3C8] ; in my old notes it says "change palette".
$0D/DDB3 20 D9 E3 JSR $E3D9 [$00:E3D9] ; draws the top left (item) are of the subscreen (note this
function is called in the refresh)
$0D/DDB6 A9 01 LDA #$01
$0D/DDB8 20 C8 E3 JSR $E3C8 [$00:E3C8]
$0D/DDBB 20 47 E6 JSR $E647 [$00:E647] ; top right (item box) border only.
$0D/DDBE A9 01 LDA #$01
$0D/DDC0 20 C8 E3 JSR $E3C8 [$00:E3C8]
$0D/DDC3 20 B6 E6 JSR $E6B6 [$00:E6B6] ; most of the bottom left box - excluding the 4 items and
the words (eg/ run, swim etc)
$0D/DDC6 20 B7 E7 JSR $E7B7 [$00:E7B7] ; most other items in the bottom left box - excluding the
very right one in the bottom box.
$0D/DDC9 20 C8 E9 JSR $E9C8 [$00:E9C8] ; triforce/crystals box.
$0D/DDCC 20 E9 EC JSR $ECE9 [$00:ECE9] ; that very right item in the bottom left box.
$0D/DDCF 20 04 ED JSR $ED04 [$00:ED04] ; wasted bytes (Just look at the subroutine!)
$0D/DDD2 A9 01 LDA #$01
$0D/DDD4 20 C8 E3 JSR $E3C8 [$00:E3C8]
$0D/DDD7 20 29 ED JSR $ED29 [$00:ED29] ; borders of the bottom right box
$0D/DDDA 20 21 EE JSR $EE21 [$00:EE21] ; shield
$0D/DDDD 20 3C EE JSR $EE3C [$00:EE3C] ; Armor
$0D/DDE0 20 57 EE JSR $EE57 [$00:EE57] ; big key (dungeon)
$0D/DDE3 20 39 EF JSR $EF39 [$00:EF39] ; compass (dungeon)
...
$0D/DE35 20 3A EB JSR $EB3A [$0D:EB3A] ; words


Here's the redraw code:
$0D/DF8A 20 C8 E3 JSR $E3C8 [$0D:E3C8] ;
$0D/DF8D 20 D9 E3 JSR $E3D9 [$0D:E3D9] ; draws the top left (item) are of the subscreen
$0D/DF90 20 3A EB JSR $EB3A [$0D:EB3A] ; words!
```

Data:

The items inventory names are at **6F3D0** while the items themselves are at **6F629 - 6F9DD**.



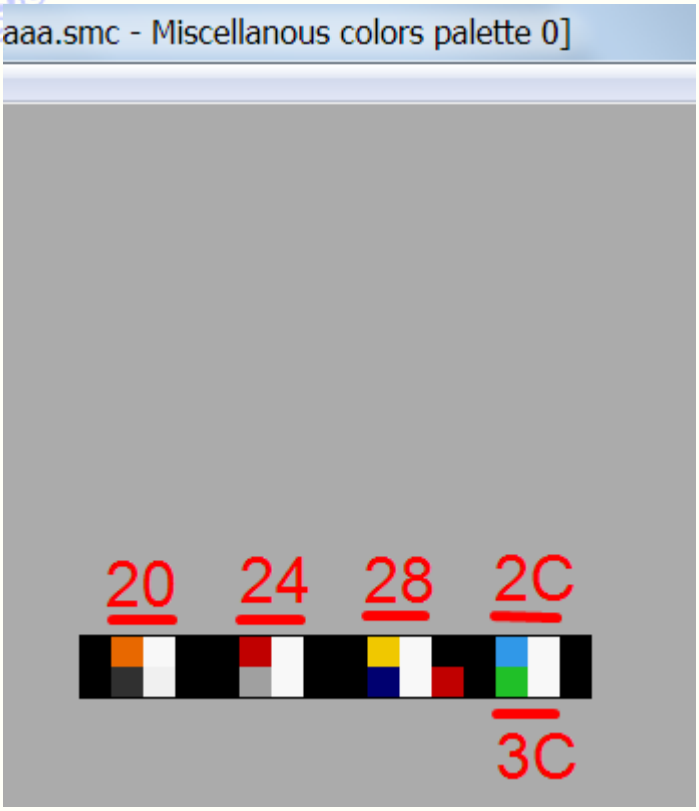
LOADING gfx into inventory

For instance Fire rod
Coordinates B0, B1, C0, C1 (B0, B is x, 0 is y for first block)

24 is red colour + white
Code is B0 24, B1 24, C0 24, C1 24 for fire rod.

Regarding the 24 and 25 values in the chart above. It is actually one of the values possible, to define the colour.

Here are the colours:



Other first byte values are 2, 6, A, E:

orange
20 = orange normal
60 = orange y mirror (mirror according to the vertical line)
A0 = orange x mirror (mirror according to the horizontal line)
E0 = orange xy mirror (mirror according to both lines)

red
24 = red normal
64 = red y mirror
A4 = red x mirror
E4 = red xy mirror

yellow
28 = yellow normal
68 = yellow y mirror
A8 = yellow x mirror
E8 = yellow xy mirror
blue
2C = blue normal
6C = blue y mirror
AC = blue x mirror
EC = blue xy mirror

*green
3C = green normal
7C = green y mirror
BC = green x mirror
FC = green xy mirror

---

20 defines 2 colours (orange and white) as presented in the miscellaneous colours (under the palettes in HM). This can be changed.

Addresses for graphics - Note these are snes ppu pointer things, 2 bytes each - follows the vhopppcc cccccccc pattern.

v = vertical mirror
h = horizontal mirror
o = sprite ordering (not really applicable to BG3)
p = palette number
c = tile number

8 bytes each - top left, top right, bottom left bottom right in this order.

6DD5F - 6DD60 - Invisible frame
6E3DC - 6E3DD - Corners
6E3FC - 6E3FD - Left and right frames
6E419 - 6E420 - Top and bottom frames
6E432 - 6E433 - Black frame
6E461, 6E462, 6E467, 6E468 - Green Y
6E64A - 6E64B - Item name (frame corners)
6E66A - 6E66B - Item name (left and right frames)
6E687 - 6E688 - Item name (top and bottom frames)
6E6A0 - 6E6A1 - Item name (black)
6E6BF - 6E6C0 - Ability (black)
6E74D - 6E75E - Ability (frame corners)
6E76D - 6E76E - Ability (left and right frames)
6E78A - 6E78B - Ability (top and bottom frames)
6E79D, 6E79E, 6E7A3, 6E7A4 - Red A
6E7A9, 6E7AA, 6E7AF, 6E7B0 - Do
6E860 - 6E913 - Yellow Pendant frame
6E914 - 6E9C7 - Yellow Crystal frame
6ED09 - 6ED18 - Equipment Text
6ED19 - 6ED28 - Dungeon Item Text
6FDEF - 6FE76 - Magic Bar Animation:
F6
3C F6 3C F6 3C F6 3C F6 3C F6 3C F6 3C A9 3C F6
3C F6 3C F6 3C AA 3C F6 3C F6 3C F6 3C AB 3C F6
3C F6 3C F6 3C AC 3C F6 3C F6 3C A9 3C AD 3C F6
3C F6 3C AA 3C AD 3C F6 3C F6 3C AB 3C AD 3C F6
3C F6 3C AC 3C AD 3C F6 3C A9 3C AD 3C AD 3C F6
3C AA 3C AD 3C AD 3C F6 3C AB 3C AD 3C AD 3C F6

3C AC 3C AD 3C AD 3C A9 3C AD 3C AD 3C AD 3C AA
3C AD 3C AD 3C AD 3C AB 3C AD 3C AD 3C AD 3C AC
3C AD 3C AD 3C AD 3C

6FE77 - 6FFC0 - Hud Layer 2
6F629 - Bow slot (when empty)
6F631 - Bow (when all used up)
6F639 - Bow (normal)
6F641 - Bow with silver arrows (when all used up)
6F649 - Bow with silver arrows (normal)
6F651 - Boomerang slot (when empty)
6F659 - Boomerang (blue)
6F661 - Boomerang (red)
6F669 - Hookshot slot (when empty)
6F671 - Hookshot
6F679 - Bombs slot (when empty)
6F681 - Bombs
6F689 - Mushroom slot (when empty)
6F691 - Mushroom
6F699 - Magic Powder
6F6A1 - Fire rod slot (when empty)
6F6A9 - Fire rod
6F6B1 - Ice rod slot (when empty)
6F6B9 - Ice rod
6F6C1 - Bombos slot (when empty)
6F6C9 - Bombos
6F6D1 - Ether slot (when empty)
6F6D9 - Ether
6F6E1 - Quake slot (when empty)
6F6E9 - Quake
6F6F1 - Lamp slot (when empty)
6F6F9 - Lamp slot
6F701 - Hammer slot (when empty)
6F709 - Hammer slot
6F711 - Shovel/flute slot (when empty)
6F719 - Shovel
6F721 - Flute
6F729 - Flute (again)
6F731 - Bug-catching net slot (when empty)
6F739 - Bug-catching net
6F741 - Mudora Book slot (when empty)
6F749 - Mudora Book
6F751 - Bottle slot (when empty)
6F759 - Mushroom (?)
6F761 - Empty bottle
6F769 - Red Potion
6F771 - Green Potion
6F779 - Blue Potion
6F781 - Fairy
6F789 - Bee
6F791 - Good Bee
6F799 - Red cane slot (when empty)
6F7A1 - Red cane

6F7A9 - Blue cane slot (when empty)
6F7B1 - Blue cane
6F7B9 - Magic Cape slot (when empty)
6F7C1 - Magic Cape
6F7C9 - Mirror slot (when empty)
6F7D1 - Letter

The Letter (called てがみ (tegami) in the Japanese version, but untranslated and blank in the English version) is another item making a return from *Zelda 1*. As in that game, it uses the same sprite as the Map. It can actually be added to the inventory in all versions of the game; it occupies the spot of the Magic Mirror, suggesting that you needed the Letter to acquire the Mirror at some point in development, in the same way you need the Shovel to acquire the Flute, which then takes the Shovel's spot in the inventory. It's unknown exactly why the letter was canned. To add it to your inventory, use Action Replay code **7EF35301**. In both versions, the item acts exactly like the Magic Mirror when used.

6F7D9 - Mirror
6F7E1 - Bomb and arrows counter
6F7E9 - Power glove slot (when empty)
6F7F1 - Power glove
6F7F9 - Titan glove
6F801 - Pegasus boots slot (when empty)
6F809 - Pegasus boots
6F811 - Zora's Flippers (when empty)
6F819 - Zora's Flippers
6F821 - Moonpearl (when empty)
6F829 - Moonpearl
6F831 - ??? (when empty)
6F839 - Sword (when none)
6F841 - Sword 1
6F849 - Sword 2
6F851 - Sword 3
6F859 - Sword 4
6F861 - Shield (when none)
6F869 - Shield 1
6F871 - Shield 2
6F879 - Shield 3
6F881 - Armor 1 (green)
6F889 - Armor 2 (blue)
6F891 - Armor 3 (red)
6F899 - Compass slot (when empty)
6F8A1 - Compass

6F8A9 - Big key slot (when empty)
6F8B1 - Big key
6F8B9 - Big chest is opened
6F8C1 - Map slot (when empty)
6F8C9 - Map
6F8D1 - Red pendant slot (when empty)
6F8D9 - Red pendant
6F8E1 - Blue pendant slot (when empty)
6F8E9 - Blue pendant
6F8F1 - Green pendant slot (when empty)
6F8F9 - Green pendant
6F901 - Green glove? slot (when empty)
6F909 - Green glove?
6F911 - Empty heart container
6F919 - 1/4 heart container
6F921 - 1/2 heart container
6F929 - 3/4 heart container
6F93B - Lift.2
6F94F - Lift.3
6F963 - Lift.1
6F979 - Read
6F98D - Talk
6F9B3 - Pull
6F9C9 - Run
6F9DD - Swim
7537F - 75390 - Monologue Text Frame:
F3 28 F4 28 F3 68 C8 28 7F 38 C8 68 F3 A8 F4 A8 F3 E8

Item descriptions

6F1C9 – 6F619
Same format as above,
2 lines per item, 16 bytes per line.
Some descriptions uses custom graphics (e.g./ BOOMERANG because it doesn't fit in a line)
Will not go into the details, but will give **the order in which items come up:**

1. bow
2. boomerang
3. hookshot
4. bomb
5. mushroom
6. fire rod
7. ice rod
8. bombos
9. ether
10. quake
11. lamp
12. magic hammer
13. shovel
14. bug catching net
15. book of Mudora
16.
17. blue cane

18. red cane
19. magic cape
20.
21. mushroom (again)
...

And the table

24=
25=
50=A
51=B
52=C
53=D
54=E
55=F
56=G
57=H
58=I
59=J
5A=K
5B=L
5C=M
5D=N
5E=O
5F=P
60=Q
61=R
62=S
63=T
64=U
65=V
66=W
67=X
68=Y
69=Z
6A=-

Each tile is like this: Bug-Catching net = 0024 0124 0224 0324 ....

With this info, you can change the name of an item in the inventory.
Load A is 50 25, B is 51 25 etc.

play screen

For the playscreen there's some code already which performs a memory move (MVN) instruction from a set memory address (6Fxxx in the rom or around there somewhere) - this paints the "base" which includes most of the graphics (magic meter border, rupee symbols etc) - also the place which the HUD editor which was (Bottle) around edits mostly.

For everything else, custom assembly/code is written to put the right tile values in those memory addresses before the screen refresh occurs (also known as the NMI interrupt). You'll also notice the tile list doesn't go to the bottom of the playscreen - the original developers only used memory up to the end of the bottom right corner of the magic meter.

```
HUD screen disassembly details

Euclid
23/5/2012


This is the start of the routine which redraws the screen every frame.

$0D/DD21 20 94 FB JSR $FB94 [$0D:FB94]


This one is for the coming back from start. Note the addresses are so close!

$0D/FA8A 20 91 FB JSR $FB91 [$0D:FB91]


Now here's the routine to draw HUD:

$0D/FB91 20 FD FA JSR $FAFD [$00:FAFD] ; first routine, NOT being reused on a redraw, draws misc
items, details later.

$0D/FB94 E2 30 SEP #$30 ; from here onwards the code is being reused on a redraw
$0D/FB96 A9 FD LDA #$FD
$0D/FB98 85 0A STA $0A [$00:000A]
$0D/FB9A A9 F9 LDA #$F9
$0D/FB9C 85 0B STA $0B [$00:000B]
$0D/FB9E A9 0D LDA #$0D
$0D/FBA0 85 0C STA $0C [$00:000C] ; Note 0DF9FD - graphic address
$0D/FBA2 A9 68 LDA #$68
$0D/FBA4 85 07 STA $07 [$00:0007]
$0D/FBA6 A9 C7 LDA #$C7
$0D/FBA8 85 08 STA $08 [$00:0008]
$0D/FBAA A9 7E LDA #$7E
$0D/FBAC 85 09 STA $09 [$00:0009] ; target start addr 0x7EC768
$0D/FBAE C2 30 REP #$30
$0D/FBB0 AF 6C F3 7E LDA $7EF36C[$7E:F36C]
$0D/FBB4 29 FF 00 AND #$00FF
$0D/FBB7 85 00 STA $00 [$00:0000]
$0D/FBB9 85 02 STA $02 [$00:0002]
$0D/FBBB 85 04 STA $04 [$00:0004]
$0D/FBBD 20 AB FD JSR $FDAB [$00:FDAB] ; this draws the HP "base" in the location (for details on
the loop seelater)

$0D/FBC0 E2 30 SEP #$30
$0D/FBC2 A9 03 LDA #$03
$0D/FBC4 85 0A STA $0A [$00:000A]
$0D/FBC6 A9 FA LDA #$FA
$0D/FBC8 85 0B STA $0B [$00:000B]
$0D/FBCA A9 0D LDA #$0D
$0D/FBCC 85 0C STA $0C [$00:000C] ; Note 0DFA03 - graphic address
$0D/FBCE A9 68 LDA #$68
```

```
$0D/FBD0 85 07 STA $07 [$00:0007]
$0D/FBD2 A9 C7 LDA #$C7
$0D/FBD4 85 08 STA $08 [$00:0008]
$0D/FBD6 A9 7E LDA #$7E
$0D/FBD8 85 09 STA $09 [$00:0009] ; target start addr 0x7EC768
$0D/FBDA AF 6C F3 7E LDA $7EF36C[$7E:F36C] ; was Nintendo trying something here? i don't see the
use of this lot of code at all.
$0D/FBDE CF 6D F3 7E CMP $7EF36D[$7E:F36D] ; the branches all go to the same spot
$0D/FBE2 F0 09 BEQ $09 [$FBED] ; no variables are saved
$0D/FBE4 38 SEC ; no subroutines etc.
$0D/FBE5 E9 04 SBC #$04 ; sounds like stupid code
$0D/FBE7 CF 6D F3 7E CMP $7EF36D[$7E:F36D] ; especially this CMP and branch which does nothing.
$0D/FBEB B0 00 BCS $00 [$FBED] ; end stupidness
$0D/FBED AF 6D F3 7E LDA $7EF36D[$7E:F36D]
$0D/FBF1 18 CLC
$0D/FBF2 69 03 ADC #$03
$0D/FBF4 C2 30 REP #$30
$0D/FBF6 29 FC 00 AND #$00FC
$0D/FBF9 85 00 STA $00 [$00:0000]
$0D/FBFB 85 04 STA $04 [$00:0004]
$0D/FBFD AF 6C F3 7E LDA $7EF36C[$7E:F36C]
$0D/FC01 29 FF 00 AND #$00FF
$0D/FC04 85 02 STA $02 [$00:0002]
$0D/FC06 20 AB FD JSR $FDAB [$00:FDAB] ; déja vu right? This draws the HP "colored bits" in the
location.


$0D/FC09 C2 30 REP #$30
$0D/FC0B AF 7B F3 7E LDA $7EF37B[$7E:F37B]
$0D/FC0F 29 FF 00 AND #$00FF
$0D/FC12 C9 01 00 CMP #$0001
$0D/FC15 90 15 BCC $15 [$FC2C]
$0D/FC17 A9 F7 28 LDA #$28F7
$0D/FC1A 8F 04 C7 7E STA $7EC704[$7E:C704]
$0D/FC1E A9 51 28 LDA #$2851
$0D/FC21 8F 06 C7 7E STA $7EC706[$7E:C706]
$0D/FC25 A9 FA 28 LDA #$28FA
$0D/FC28 8F 08 C7 7E STA $7EC708[$7E:C708] ; this draws the 1/2 symbol on top of the magic bar.


$0D/FC2C AF 6E F3 7E LDA $7EF36E[$7E:F36E]
$0D/FC30 29 FF 00 AND #$00FF
$0D/FC33 18 CLC
$0D/FC34 69 07 00 ADC #$0007
$0D/FC37 29 F8 FF AND #$FFF8
$0D/FC3A AA TAX
$0D/FC3B BD EF FD LDA $FDEF,x[$00:FDEF]
$0D/FC3E 8F 46 C7 7E STA $7EC746[$7E:C746]
$0D/FC42 BD F1 FD LDA $FDF1,x[$00:FDF1]
$0D/FC45 8F 86 C7 7E STA $7EC786[$7E:C786]
$0D/FC49 BD F3 FD LDA $FDF3,x[$00:FDF3]
$0D/FC4C 8F C6 C7 7E STA $7EC7C6[$7E:C7C6]
$0D/FC50 BD F5 FD LDA $FDF5,x[$00:FDF5]
$0D/FC53 8F 06 C8 7E STA $7EC806[$7E:C806] ; this code here draws the magic bar, very smart by the
programmers btw.

$0D/FC57 AF 62 F3 7E LDA $7EF362[$7E:F362]
$0D/FC5B 20 F7 F0 JSR $F0F7 [$00:F0F7] ; see binary_to_decimal Routine,
$0D/FC5E C2 30 REP #$30
$0D/FC60 A5 03 LDA $03 [$00:0003]
$0D/FC62 29 FF 00 AND #$00FF
$0D/FC65 09 00 24 ORA #$2400
$0D/FC68 8F 50 C7 7E STA $7EC750[$7E:C750]
$0D/FC6C A5 04 LDA $04 [$00:0004]
$0D/FC6E 29 FF 00 AND #$00FF
$0D/FC71 09 00 24 ORA #$2400
$0D/FC74 8F 52 C7 7E STA $7EC752[$7E:C752]
$0D/FC78 A5 05 LDA $05 [$00:0005]
$0D/FC7A 29 FF 00 AND #$00FF
```

```
$0D/FC7D 09 00 24 ORA #$2400
$0D/FC80 8F 54 C7 7E STA $7EC754[$7E:C754] ; this code here draws rupees.

$0D/FC84 AF 43 F3 7E LDA $7EF343[$7E:F343]
$0D/FC88 29 FF 00 AND #$00FF
$0D/FC8B 20 F7 F0 JSR $F0F7 [$00:F0F7] ; see binary_to_decimal Routine,
$0D/FC8E C2 30 REP #$30
$0D/FC90 A5 04 LDA $04 [$00:0004]
$0D/FC92 29 FF 00 AND #$00FF
$0D/FC95 09 00 24 ORA #$2400
$0D/FC98 8F 58 C7 7E STA $7EC758[$7E:C758]
$0D/FC9C A5 05 LDA $05 [$00:0005]
$0D/FC9E 29 FF 00 AND #$00FF
$0D/FCA1 09 00 24 ORA #$2400
$0D/FCA4 8F 5A C7 7E STA $7EC75A[$7E:C75A] ; this code here draws # Bombs.

$0D/FCA8 AF 77 F3 7E LDA $7EF377[$7E:F377]
$0D/FCAC 29 FF 00 AND #$00FF
$0D/FCAF 20 F7 F0 JSR $F0F7 [$00:F0F7]
$0D/FCB2 C2 30 REP #$30
$0D/FCB4 A5 04 LDA $04 [$00:0004]
$0D/FCB6 29 FF 00 AND #$00FF
$0D/FCB9 09 00 24 ORA #$2400
$0D/FCBC 8F 5E C7 7E STA $7EC75E[$7E:C75E]
$0D/FCC0 A5 05 LDA $05 [$00:0005]
$0D/FCC2 29 FF 00 AND #$00FF
$0D/FCC5 09 00 24 ORA #$2400
$0D/FCC8 8F 60 C7 7E STA $7EC760[$7E:C760] ; this code here draws # arrows.

$0D/FCCC A9 7F 00 LDA #$007F
$0D/FCCF 85 05 STA $05 [$00:0005]
$0D/FCD1 AF 6F F3 7E LDA $7EF36F[$7E:F36F]
$0D/FCD5 29 FF 00 AND #$00FF
$0D/FCD8 C9 FF 00 CMP #$00FF
$0D/FCDB F0 03 BEQ $03 [$FCE0] ; 0 keys (which is applicable to overworld) don't calculate - also
means don't draw.
$0D/FCDD 20 F7 F0 JSR $F0F7 [$00:F0F7] ; see binary_to_decimal Routine,
$0D/FCE0 C2 30 REP #$30
$0D/FCE2 A5 05 LDA $05 [$00:0005]
$0D/FCE4 29 FF 00 AND #$00FF
$0D/FCE7 09 00 24 ORA #$2400
$0D/FCEA 8F 64 C7 7E STA $7EC764[$7E:C764]
$0D/FCEE C9 7F 24 CMP #$247F ; see why it set 7F into it earlier?
$0D/FCF1 D0 04 BNE $04 [$FCF7]
$0D/FCF3 8F 24 C7 7E STA $7EC724[$7E:C724] ; keys.

$0D/FCF7 E2 30 SEP #$30 ; Finished. Some very tightly packed code!
$0D/FCF9 60 RTS ; the amount of extra things you can fit into this is minimal!
; however if the subscreen is done then this becomes a lot easier

There is some bits of code scattered around which deals with HUD, here it is:

$0A/FD0C C2 30 REP #$30 ; NOTE entrance to this routine is a JMP/JSR, don't move this line.
$0A/FD0E AD A0 04 LDA $04A0 [$00:04A0]
$0A/FD11 29 FF 00 AND #$00FF
$0A/FD14 F0 7A BEQ $7A [$FD90] ; this branches to a RTL.
$0A/FD90 C2 20 REP #$20
$0A/FD92 A9 7F 00 LDA #$007F
$0A/FD95 8F F2 C7 7E STA $7EC7F2[$7E:C7F2] ; 1F!
$0A/FD99 8F 32 C8 7E STA $7EC832[$7E:C832] ; 1F!
$0A/FD9D 8F F4 C7 7E STA $7EC7F4[$7E:C7F4] ; 1F!
$0A/FDA1 8F 34 C8 7E STA $7EC834[$7E:C834] ; 1F!
$0A/FDA5 E2 30 SEP #$30
$0A/FDA7 6B RTL


$0A/FD7C B9 E0 FC LDA $FCE0,y[$0A:FCE4]
$0A/FD7F 9F F2 C7 7E STA $7EC7F2,x[$7E:C7F4]
$0A/FD83 B9 F6 FC LDA $FCF6,y[$0A:FCFA]
```

```
$0A/FD86 9F 32 C8 7E STA $7EC832,x[$7E:C834]

$0A/FD25 A9 1E 25 LDA #$251E
$0A/FD28 8F F2 C7 7E STA $7EC7F2[$7E:C7F2]
$0A/FD2C 1A INC A
$0A/FD2D 8F 34 C8 7E STA $7EC834[$7E:C834]
$0A/FD31 1A INC A
$0A/FD32 8F 32 C8 7E STA $7EC832[$7E:C832]
$0A/FD36 A9 0F 25 LDA #$250F
$0A/FD39 8F F4 C7 7E STA $7EC7F4[$7E:C7F4]

Hearts refilling

$0D/F14F E2 30 SEP #$30 A:2436 X:00FF Y:00FE P:eNvMXdizc
$0D/F151 A9 68 LDA #$68 A:2436 X:00FF Y:00FE P:eNvMXdizc
$0D/F153 85 00 STA $00 [$00:0000] A:2468 X:00FF Y:00FE P:envMXdizc
$0D/F155 A9 C7 LDA #$C7 A:2468 X:00FF Y:00FE P:envMXdizc
$0D/F157 85 01 STA $01 [$00:0001] A:24C7 X:00FF Y:00FE P:eNvMXdizc
$0D/F159 A9 7E LDA #$7E A:24C7 X:00FF Y:00FE P:eNvMXdizc
$0D/F15B 85 02 STA $02 [$00:0002] A:247E X:00FF Y:00FE P:envMXdizc
$0D/F15D CE 08 02 DEC $0208 [$0D:0208] A:247E X:00FF Y:00FE P:envMXdizc
$0D/F160 D0 4F BNE $4F [$F1B1] A:247E X:00FF Y:00FE P:envMXdizc

$0D/F1B1 18 CLC A:247E X:00FF Y:00FE P:envMXdizc
$0D/F1B2 60 RTS A:247E X:00FF Y:00FE P:envMXdizc

As part of the HUD - the dialog box border:

$0E/D2AB C2 30 REP #$30 A:6244 X:0000 Y:0000 P:envMXdiZc
$0E/D2AD AD D0 1C LDA $1CD0 [$0E:1CD0] A:6244 X:0000 Y:0000 P:envmxdiZc
$0E/D2B0 EB XBA A:6244 X:0000 Y:0000 P:envmxdizc
$0E/D2B1 9D 02 10 STA $1002,x[$0E:1002] A:4462 X:0000 Y:0000 P:envmxdizc
$0E/D2B4 E8 INX A:4462 X:0000 Y:0000 P:envmxdizc
$0E/D2B5 E8 INX A:4462 X:0001 Y:0000 P:envmxdizc
$0E/D2B6 EB XBA A:4462 X:0002 Y:0000 P:envmxdizc
$0E/D2B7 18 CLC A:6244 X:0002 Y:0000 P:envmxdizc
$0E/D2B8 69 20 00 ADC #$0020 A:6244 X:0002 Y:0000 P:envmxdizc
$0E/D2BB 8D D0 1C STA $1CD0 [$0E:1CD0] A:6264 X:0002 Y:0000 P:envmxdizc
$0E/D2BE A9 00 2F LDA #$2F00 A:6264 X:0002 Y:0000 P:envmxdizc
$0E/D2C1 9D 02 10 STA $1002,x[$0E:1004] A:2F00 X:0002 Y:0000 P:envmxdizc
$0E/D2C4 E8 INX A:2F00 X:0002 Y:0000 P:envmxdizc
$0E/D2C5 E8 INX A:2F00 X:0003 Y:0000 P:envmxdizc
$0E/D2C6 B9 7F D3 LDA $D37F,y[$0E:D37F] A:2F00 X:0004 Y:0000 P:envmxdizc
$0E/D2C9 9D 02 10 STA $1002,x[$0E:1006] A:28F3 X:0004 Y:0000 P:envmxdizc
$0E/D2CC E8 INX A:28F3 X:0004 Y:0000 P:envmxdizc
$0E/D2CD E8 INX A:28F3 X:0005 Y:0000 P:envmxdizc
$0E/D2CE C8 INY A:28F3 X:0006 Y:0000 P:envmxdizc
$0E/D2CF C8 INY A:28F3 X:0006 Y:0001 P:envmxdizc
$0E/D2D0 A9 16 00 LDA #$0016 A:28F3 X:0006 Y:0002 P:envmxdizc
$0E/D2D3 85 0E STA $0E [$00:000E] A:0016 X:0006 Y:0002 P:envmxdizc
$0E/D2D5 B9 7F D3 LDA $D37F,y[$0E:D381] A:0016 X:0006 Y:0002 P:envmxdizc
$0E/D2D8 9D 02 10 STA $1002,x[$0E:1008] A:28F4 X:0006 Y:0002 P:envmxdizc
$0E/D2DB E8 INX A:28F4 X:0006 Y:0002 P:envmxdizc
$0E/D2DC E8 INX A:28F4 X:0007 Y:0002 P:envmxdizc
$0E/D2DD C6 0E DEC $0E [$00:000E] A:28F4 X:0008 Y:0002 P:envmxdizc
$0E/D2DF D0 F7 BNE $F7 [$D2D8] A:28F4 X:0008 Y:0002 P:envmxdizc
$0E/D2E1 C8 INY A:28F4 X:0032 Y:0002 P:envmxdiZc
$0E/D2E2 C8 INY A:28F4 X:0032 Y:0003 P:envmxdizc
$0E/D2E3 B9 7F D3 LDA $D37F,y[$0E:D383] A:28F4 X:0032 Y:0004 P:envmxdizc
$0E/D2E6 9D 02 10 STA $1002,x[$0E:1034] A:68F3 X:0032 Y:0004 P:envmxdizc
$0E/D2E9 E8 INX A:68F3 X:0032 Y:0004 P:envmxdizc
$0E/D2EA E8 INX A:68F3 X:0033 Y:0004 P:envmxdizc
$0E/D2EB 60 RTS A:68F3 X:0034 Y:0004 P:envmxdizc

First routine:

$0D/FAFD E2 30 SEP #$30
$0D/FAFF AF 40 F3 7E LDA $7EF340[$7E:F340] ;
$0D/FB03 F0 37 BEQ $37 [$FB3C]
```

```
$0D/FB05 C9 03 CMP #$03
$0D/FB07 90 24 BCC $24 [$FB2D]
$0D/FB09 A9 86 LDA #$86
$0D/FB0B 8F 1E C7 7E STA $7EC71E[$7E:C71E]
$0D/FB0F A9 24 LDA #$24
$0D/FB11 8F 1F C7 7E STA $7EC71F[$7E:C71F]
$0D/FB15 A9 87 LDA #$87
$0D/FB17 8F 20 C7 7E STA $7EC720[$7E:C720]
$0D/FB1B A9 24 LDA #$24
$0D/FB1D 8F 21 C7 7E STA $7EC721[$7E:C721]
$0D/FB21 A2 04 LDX #$04
$0D/FB23 AF 77 F3 7E LDA $7EF377[$7E:F377]
$0D/FB27 D0 0E BNE $0E [$FB37]
$0D/FB29 A2 03 LDX #$03
$0D/FB2B 80 0A BRA $0A [$FB37]
$0D/FB2D A2 02 LDX #$02
$0D/FB2F AF 77 F3 7E LDA $7EF377[$7E:F377]
$0D/FB33 D0 02 BNE $02 [$FB37]
$0D/FB35 A2 01 LDX #$01
$0D/FB37 8A TXA
$0D/FB38 8F 40 F3 7E STA $7EF340[$7E:F340] ; this code deals with the bow/arrows change to "no
arrows" graphic.


$0D/FB3C C2 30 REP #$30 ; this code draws the item box.
$0D/FB3E AE 02 02 LDX $0202 [$00:0202]
$0D/FB41 F0 4D BEQ $4D [$FB90] ; no item selected, ignore.
$0D/FB43 BF 3F F3 7E LDA $7EF33F,x[$7E:F33F] ; pick item, position 1 = bow, etc.
$0D/FB47 29 FF 00 AND #$00FF
$0D/FB4A E0 04 00 CPX #$0004
$0D/FB4D D0 03 BNE $03 [$FB52] ; hookshot is position 4, have to set the number to 1 (probably
graphic issues)
$0D/FB4F A9 01 00 LDA #$0001
$0D/FB52 E0 10 00 CPX #$0010
$0D/FB55 D0 0A BNE $0A [$FB61] ; that's the bottles!
$0D/FB57 9B TXY
$0D/FB58 AA TAX
$0D/FB59 BF 5B F3 7E LDA $7EF35B,x[$7E:F35B] ; load the correct bottle graphic.
$0D/FB5D 29 FF 00 AND #$00FF
$0D/FB60 BB TYX
$0D/FB61 85 02 STA $02 [$00:0002] ; store position of graphic.
$0D/FB63 8A TXA
$0D/FB64 3A DEC A
$0D/FB65 0A ASL A
$0D/FB66 AA TAX ; relative position is ((mem_addr_of_item - 7EF340) * 2)
$0D/FB67 BD 93 FA LDA $FA93,x[$00:FA93] ; loads graphic pointer.
$0D/FB6A 85 04 STA $04 [$00:0004]
$0D/FB6C A5 02 LDA $02 [$00:0002]
$0D/FB6E 0A ASL A
$0D/FB6F 0A ASL A
$0D/FB70 0A ASL A
$0D/FB71 A8 TAY
$0D/FB72 B1 04 LDA ($04),y[$00:5555] ; loads the graphic in $0D bank, upper left NOTE see data area
for a description of where they are.
$0D/FB74 8F 4A C7 7E STA $7EC74A[$7E:C74A]
$0D/FB78 C8 INY
$0D/FB79 C8 INY
$0D/FB7A B1 04 LDA ($04),y[$00:5555] ; upper right
$0D/FB7C 8F 4C C7 7E STA $7EC74C[$7E:C74C]
$0D/FB80 C8 INY
$0D/FB81 C8 INY
$0D/FB82 B1 04 LDA ($04),y[$00:5555]
$0D/FB84 8F 8A C7 7E STA $7EC78A[$7E:C78A] ; lower left
$0D/FB88 C8 INY
$0D/FB89 C8 INY
$0D/FB8A B1 04 LDA ($04),y[$00:5555]
$0D/FB8C 8F 8C C7 7E STA $7EC78C[$7E:C78C] ; lower right
$0D/FB90 60 RTS
```

hearts routine (only used by the 2 places above)

```
$0D/FDAB A2 00 00 LDX #$0000
$0D/FDAE A5 00 LDA $00 [$00:0000]
$0D/FDB0 C9 08 00 CMP #$0008
$0D/FDB3 90 0F BCC $0F [$FDC4]
$0D/FDB5 E9 08 00 SBC #$0008
$0D/FDB8 85 00 STA $00 [$00:0000]
$0D/FDBA A0 04 00 LDY #$0004
$0D/FDBD 20 D9 FD JSR $FDD9 [$00:FDD9]
$0D/FDC0 E8 INX
$0D/FDC1 E8 INX
$0D/FDC2 80 EA BRA $EA [$FDAE]
$0D/FDC4 C9 05 00 CMP #$0005
$0D/FDC7 90 05 BCC $05 [$FDCE]
$0D/FDC9 A0 04 00 LDY #$0004
$0D/FDCC 80 0B BRA $0B [$FDD9]
$0D/FDCE C9 01 00 CMP #$0001
$0D/FDD1 90 05 BCC $05 [$FDD8]
$0D/FDD3 A0 02 00 LDY #$0002
$0D/FDD6 80 01 BRA $01 [$FDD9]
$0D/FDD8 60 RTS

$0D/FDD9 E0 14 00 CPX #$0014
$0D/FDDC 90 0B BCC $0B [$FDE9]
$0D/FDDE A2 00 00 LDX #$0000
$0D/FDE1 A5 07 LDA $07 [$00:0007]
$0D/FDE3 18 CLC
$0D/FDE4 69 40 00 ADC #$0040
$0D/FDE7 85 07 STA $07 [$00:0007]
$0D/FDE9 B7 0A LDA [$0A],y[$55:5555]
$0D/FDEB 9B TXY
$0D/FDEC 97 07 STA [$07],y[$55:5555]
$0D/FDEE 60 RTS
```

binary_to_decimal Routine:
NOTE Never - change the entrance position of this routine, it's being reused in MANY PLACES.
quite simple, takes in number in accumulator, spits out result in addresses, 03, 04, and 05.

```
$0D/F0F7 C2 30 REP #$30
$0D/F0F9 9C 03 00 STZ $0003 [$00:0003]
$0D/F0FC A2 00 00 LDX #$0000
$0D/F0FF A0 02 00 LDY #$0002
$0D/F102 D9 F9 F9 CMP $F9F9,y[$00:F9F9]
$0D/F105 90 08 BCC $08 [$F10F]
$0D/F107 38 SEC
$0D/F108 F9 F9 F9 SBC $F9F9,y[$00:F9F9]
$0D/F10B F6 03 INC $03,x [$00:0003]
$0D/F10D 80 F3 BRA $F3 [$F102]
$0D/F10F E8 INX
$0D/F110 88 DEY
$0D/F111 88 DEY
$0D/F112 10 EE BPL $EE [$F102]
$0D/F114 85 05 STA $05 [$00:0005]
$0D/F116 E2 30 SEP #$30
$0D/F118 A2 02 LDX #$02
$0D/F11A B5 03 LDA $03,x [$00:0003]
$0D/F11C C9 7F CMP #$7F
$0D/F11E F0 02 BEQ $02 [$F122]
$0D/F120 09 90 ORA #$90
$0D/F122 95 03 STA $03,x [$00:0003]
$0D/F124 CA DEX
$0D/F125 10 F3 BPL $F3 [$F11A]
$0D/F127 60 RTS
```

b) Changing the objects palettes (higher quality items)

You may have noticed already but the items use two different graphic sets of tiles. One for the items which are used in the inventory and the other one for the items you get in chests/while they are being used. The following table will help you change the later ones (aka: the higher quality ones).

The palette values for the items as seen when you open a chest are as follow:

00 = transparent
01 = red
02 = blue
03 = fully black
04 = green
05 = brown
06 = lighter brown
07 = glowing color
FF = glowing blue (Master Sword; If you set this item in a treasure and open it, the game freeze)

While the palette values for the items while they are being used:

02 = red
04 = blue
08 = green

00, 06 and 10 = not recommended (black and dark brown). Odd values (bugged gfx).

12 repeats the cycle of 02, so 12 is the same as 02 (with possible different sub colours).

Hex	Dec	Description	Graphics	Seen in chest (hex location)	Default value	Being used (hex location)	Default value
00	00	Fighter's Sword and Shield		0x4849C	05		
01	01	Master Sword		0x4849D	FF		
02	02	Tempered Sword		0x4849E	05		
03	03	Golden Sword		0x4849F	05		
04	04	Fighter's Shield		0x484A0	05		
05	05	Red Shield		0x484A1	05		
06	06	Mirror Shield		0x484A2	05		
07	07	Fire Rod		0x484A3	01		

08	08	Ice Rod		0x484A4	02		
09	09	Magic Hammer		0x484A5	01		
0A	10	Hookshot		0x484A6	02		
0B	11	Bow		0x484A7	01		
0C	12	Blue Boomerang		0x484A8	02		
0D	13	Magic Powder		0x484A9	02		
0E	14	Bee (needs bottle or you get nothing)		0x484AA	02		
0F	15	Bombos Medallion		0x484AB	04		
10	16	Ether Medallion		0x484AC	04		
11	17	Quake Medallion		0x484AD	04		
12	18	Lantern		0x484AE	01		
13	19	Shovel		0x484AF	01		
14	20	Flute		0x484B0	02		
15	21	Cane of Somaria(red)		0x484B1	01		
16	22	Bottle		0x484B2	01		
17	23	Heart Piece		0x484B3	01		
18	24	Cane of Byrna(blue)		0x484B4	02		
19	25	Magic Cape		0x484B5	01		
1A	26	Magic Mirror		0x484B6	02		
1B	27	Power Glove		0x484B7	01		
1C	28	Titan's Mitt		0x484B8	04		
1D	29	Book of Mudora		0x484B9	04		
1E	30	Zora's Flippers		0x484BA	02		
1F	31	Moon Pearl		0x484BB	01		
20	32	Crystal		0x484BC	06		
21	33	Bug-Catching Net		0x484BD	01		
22	34	Blue Mail		0x484BE	02		
23	35	Red Mail		0x484BF	01		
24	36	Small Key		0x484C0	02		
25	37	Compass		0x484C1	02		
26	38	Liar Heart (Full heart container w/message)		0x484C2	01		
27	39	Bomb		0x484C3	02	41F00	04
28	40	3 Bombs		0x484C4	02		
29	41	Mushroom		0x484C5	04		
2A	42	Magical Boomerang		0x484C6	01		
2B	43	Medicine of Life		0x484C7	01		
2C	44	Medicine of Magic		0x484C8	04		
2D	45	Medicine of Life and Magic		0x484C9	02		
2E	46	Medicine of Life (needs bottle or you get nothing)		0x484CA	01		

2F	47	Medicine of Magic (needs bottle or you get nothing)		0x484CB	04		
30	48	Medicine of Life and Magic (needs bottle or you get nothing)		0x484CC	02		
31	49	10 Bombs		0x484CD	02		
32	50	Big Key		0x484CE	04		
33	51	Map		0x484CF	04		
34	52	Green Rupee (1)		0x484D0	04		
35	53	Blue Rupee (5)		0x484D1	02		
36	54	Red Rupee (20)		0x484D2	01		
37	55	Pendant of Courage		0x484D3	04		
38	56	Pendant of Power		0x484D4	01		
39	57	Pendant of Wisdom		0x484D5	02		
3A	58	Bow & Arrows		0x484D6	02		
3B	59	Bow & Silver Arrows		0x484D7	01		
3C	60	Normal Bee		0x484D8	02		
3D	61	Fairy		0x484D9	02		
3E	62	MuteHeart (Full heart container w/ no message)		0x484DA	01		
3F	63	Heart Container (Full heart container w/message)		0x484DB	01		
40	64	100 Rupees		0x484DC	04		
41	65	50 Rupees		0x484DD	04		
42	66	Small Heart		0x484DE	01		
43	67	Arrow		0x484DF	02		
44	68	10 Arrows		0x484E0	02		
45	69	Small Magic Jar		0x484E1	04		
46	70	300 Rupees		0x484E2	04		
47	71	20 Rupees		0x484E3	04		
48	72	Golden Bee		0x484E4	02		
49	73	Sword 1		0x484E5	05		
4A	74	Flute		0x484E6	02		
4B	75	Pegasus Boots		0x484E7	01		

8) Various effects you can change in hex

*** All hex offsets are for a NON-headered rom! ***

+200 to all addresses if you have a header to your rom!

Hex offsets found by Euclid:

Disable all overworld overlays

headerless rom - 0x12F5D change from 90 to 80. Note this gets rid of all overlays including the bg clouds in area 03, 05, 07, lost woods area 00 (before master sword), area 70 after clearing the rain. Also gets rid of rain in the beginning of the game.

If you want to just get rid of the pyramid, try changing 0x12FFA from 96 to 00 (note 96 = area 96 bg to load for area 5B)

Overworld areas always blank-out

Remember how when you enter/exit Area 00 or 40 you'll get a "blank out" before going into the next screen? This actually does good when going say, from a brown grass area to a green grass area, or go from a non-raining bg to a raining bg (the swamp in dark world is a good example, if you have joined that screen with the other screens and don't feel like getting rid of the raining overlay.)

Well here's how to make it always blank-out.

02AAE3EA
02AAE4EA

Enter these 2 codes, or if you want them permanently, just hack these codes into your rom.

rom with header offset: 12CE3 and 12CE4 - change to EA

If you want them in particular spots, I'm afraid that requires some asm programming work and some free space in the

rom, checking for 0 is actually takes less space to program than say, check 1B and 0

Here's a rough dump of the code

```
# pop the "Current area code" off the stack
$02/AADA 68 PLA
# check if the area is 0 or #40, (only #40& 3F and 0 & 3F gives 0) and that jumps to
02/AAE5.
$02/AADB 29 3F AND #$3F
$02/AADD F0 06 BEQ $06
# if it is 0/#40, load the current area code (stored in $8A, anything other than 0 or 40
will make it jump to another section of the code.)
$02/AADF A5 8A LDA $8A
$02/AAE1 29 BF AND #$BF
$02/AAE3 D0 0F BNE $0F
# here's the initialisation routine for the "blank out" sequence by setting the "sub
module" bits to appropriate values (thanks to MathOnNapkins for those bits, which made
finding this way easier)
$02/AAE5 64 B0 STZ $B0
$02/AAE7 A9 0D LDA #$0D
$02/AAE9 85 11 STA $11
$02/AAEB A9 00 LDA #$00
$02/AAED 85 95 STA $95
$02/AAEF 8F 11 C0 7E STA $7EC011
$02/AAF3 60 RTS
```

Price for the flippers

At address **29A9A**

value of the bytes **F4 01** = 500 rupees because 1F4 is 100 = 256, + f4 is 244, = 500

Use the windows calculator to convert from Hex to dec
Hex dec
10 16
A0 160
F4 244
BC 188
100 256
200 512
A0 160
A1 161
A2 162
64 100

Let's say you want 700 rupees:

Select dec, type 700, select hex = 2BC = bytes **BC 02**
200 = 512
BC = 188

change bytes **F4 01** to **BC 02** and the flippers will cost 700 rupees.

Parallel worlds (the flippers are only 50 rupees)
50 in dec is 32 in hex or 032 or 32 00

so Euclid changed it from **F4 01** to **32 00**.

Mirror warping

At address **3A951**, original values must be **29 40 D0 07** (mirror working in dark world only)

New values you can change into
9F 9F 9F 9F (mirror NOT working in both worlds)

Warping from light to dark world and from dark to light world is only possible by placing warps. Warp placed in light world takes you to dark world. Warp in dark world takes you to light world.

This is optimal, because the possibility of bugs is low and dark world can be completely different from the light world, not just a copy.

Another new set of possible values
EA 4C 5C A9 (mirror working in both worlds = not optimal)

In all cases: in the dungeon, mirror warps to the beginning of the dungeon.

No Music when warping:

At address 2F1 , original values must be **F0 10 8D 40 21**
Change to **4C 03 81 EA EA**

Custom sprites under bushes

You can change which sprites randomly appear under bushes (overworlds)
At address **301F4** – **30208**, change any of the digits to anything you like!

D9 3E 79 D9 DC D8 DA E4 E1 DC D8 DF E0 0B 42 D3 41 D4 D9 E3 D8

Y Val effect
01 D9 1 rupee/heart
02 3E rock crab
03 79 Bee
04 D9 1 rupee/heart
05 DC bomb (item)
06 D8 heart (again?)
07 DA 5 rupees
08 E4 key?
09 E1 Arrow (item)
0A DC bomb (item)
0B D8 heart (again?)

0C DF Magic (small)
0D E0 big Magic (gee, must be darn rare)
0E 0B Chicken (gee must be darn rare)
0F 42 Green soldier
10 D3 Alive rock (dark world only)
11 41 Blue soldier
12 D4 ground bomb
13 D9 1 rupee/heart
14 E3 Fairy (must be darn rare)

PS: item DB is 20 rupees.

Graveyard hex

Open Hyrule Magic, go to overworld Area 14, then toggle the address calculator button ON and click next to the Push Grave you want to work out co-ords.

7E0020, C0
7E0021, 04
7E0022, 47
7E0023, 02

Those are the x and y coordinates of the grave.

For example:

25, 11 is the original position in the game and we want to have 04, 11 there instead.
That's -21 in the x position.
yyyy yyxx xxxx <-- the format

The first two values are the **x** and **y positions** **(7E0020-23)**

The 3rd one is **0592**

05B2 that's the default value already in the rom for the staircase in Area 14
0592 left of where the original is supposed to be located
x, y = 05, 92

4th value = 3rd value – **80** ... so in this case it would be **0512**

7E0020, C0
7E0021, 04
7E0022, 47
7E0023, 02

05 92

05 12

jump to hex offset **49B46**

you should see **C0 04**, you should replace it with the values in **7E0020**, then **7E0021**.

This also happens to be **C0 04**.

At **49B64,** you should see 90 09, replace it with what you see in **7E0022** replacing the last digit with 0 and then **7E0023** for the 09.

At **49BBE,** you should see B2 05, put 92 05.

At **46FD9**, you should see 32 05, put 12 05.

The values for the hole one is like right after every single one of the stairs offset with the exception of the 4th value. **46FE0**, a bit after the other 4th value.

Pegasus boots useless on gravestones: At **49C42** change from D0 to 80

Adjusting the speed of the rain

At 0x1270D are the values to make the rain faster/slower.

faster .... slower 00 01 03 07 0F 1F 3F 7F FF

03 is the default value
00 is really fast (like swapping every frame)
01 is swapping every 2 frames
03 is swapping every 4 frames
and it goes up

Transparent corners for grass (overworlds)

At **600A9** you should see **69 26** written there at that address
just put in a new value to change the color to anything of your choosing

600A9 - Light Overworld

600B2 - Dark Overworld

75825 - enter master sword/under the bridge
75840 - enter zora's domain
75845 - exit to light overworld
7584F - exit to dark overworld

Overworld area which has holes that hurt

You can change the area to which holes will hurt the player!
currently it only allows you to choose one area

396DB, should be a **05** - Change to another area hex number

Change where you can call the bird

(Glitch if you call it from the Dark World, since the game think you're in the Light World)
At address **3A604**, original values must be **D0 D4** = Can call bird in light world only

Change to **EA EA** = Can call bird in both worlds

Change to **F0 D4** = Can call bird in Dark World only.

Link can walk through all backdrops

0x3DE37 : F0
Change to D0

Always shoot sword beams regardless of heart level

0x39E7B - ?? ??
Change to EA EA

Magic usage required for items

0x80 = Maximum Mana
0x3B29A - Lamp/Torch

Bug Catching-Net Boy gives Quake Medallion

(Note: The GFX shown will still be the net.)
0x4872A : 4D

Change to 49

Remove flashing effect (Areas 43, 45 and 47)

0x77787 : A5
Change to 6B

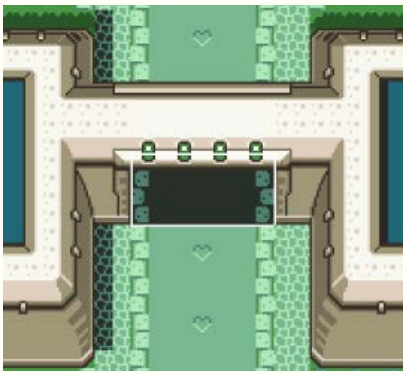
Remove a warp event that isn't in HM (Area 33)

0x17131 : D0
Change to 80

Remove a warp event that isn't in HM (Area 2F)

0x1713A : D0
Change to 80

Remove Hyrule Castle doors overlay in part two (Area 1B)



0x779FE : A9
Change to 60

Make Hyrule Castle doors open again after beating Agahnim

0xDBE9E : B0 21
Change to EA EA

Make the "Kill boss again" header to work in all rooms

0x10B08 : ??
Change to 80

Bosses don't drop hearts

2F14C : D0 07 (or something similar)
Change to EA EA

Remove the beginning sequence message reminder

3F698 : 38
Change to 18

HM won't shrink the rom again

0xFF at the very last byte of the rom after expanding.

Turn Area 03 into grass

1489A - change to EA EA

Hex offsets found by MathOnNapkins:

Fixing the camera shakes when Armos and Ganon pounds or when the walls move

At address **E8000**, the first two bytes must have the value **01 FF** (normal shake)
False shake = **7F 80** (done by Hyrule Magic)

Chrono Trigger/ Final Fantasy style chests (Can open them from any side)

If it ever annoyed you that Zelda 3 chests can only be opened from the front (which sort of makes sense, but regardless) there is a simple two byte change you can do to eliminate this.

Go to address **3B576** and change that byte and the following one to **EA EA**.

The instruction before this looks at $2F (Link's direction) and the instruction you're NOPing out branches if it's not up (0 = up, and it's using BNE).

The other directions are down = 2, left = 4, right = 6.

So there you have it, Chrono Trigger/ Final Fantasy style chests using only two bytes.

0x3B576 : D0 47
Change to EA EA

Sword damage indices

At address **36D33-36D3E**, there is a data table that contains the damage indices for the sword.

01 02 03 04 02 03 04 05 01 01 02 03

1, 2, 3, 4 ; – normal strike damage indices
2, 3, 4, 5 ; – spin attack damage indices
1, 1, 2, 3 ; – stabbing damage indices

Removing the pits damage

By default, the pits deal one heart. For those that didn't played Majora's Mask, the creators changed the pits to deal 0 damage as in it just warps you back to the beginning.

Here's a code snippet in Bank 07 (0x38000 to 0x3FFFF in the rom):

```
LDA #$14; Return Link from the damaging pit.
STA $11;

LDA $7EF36D; Load His HP.
SEC;
SBC #$08;
STA $7EF36D; Subtract one heart.
CMP #$A8;

BCC BRANCH_EPSILON2; Could replace this with a BPL... maybe.
```

Notice this "SBC #$08". This is what causes Link to lose one heart. If we replace some of this code with instructions that don't do anything (NOP instructions), it effectively will reduce the damage to zero.

To do this, change the bytes at address 0x3950C (unheadered ROM)
from "38 E9 08" to "EA EA EA".

Alternatively we could just have it subtract zero, by changing the byte
at 0x3950E from "08" to "00"

As long as pits are set to damage, that should do the trick.

Intro Setup Screen

At address **6415D-6419F**, you can change various effects that belong to the intro screen. You can even change the Nintendo "twinkle" sound to something else!

22 00 80 02 A9 **0F** 85 **13** 64 **B0** E6 **15** E6 **11** A9 **0A** 8D **2F 01** 20 82 ED A5 B0 E6 B0 C9 0B B0 7A 22 9C 87 00 A0 C1 0C A0 C1 0C A0 C1 0C A0 C1 0C A0 C1 0C A0 C1 0C A0 C1 0C A0 C1 0C 16 81 02 31 D2 00 23 D4 00

JSL; $10000 IN ROM

; Push the screen to full brightness next frame
LDA.b #$**0F** : STA $**13**

```
; Initialize the sub-submodule index.
STZ $B0

; Indicates that CGRAM needs to be updated next frame
INC $15

; Move into the next submodule.
INC $11

; Plays the twinkle as the 'Nintendo' logo pops up. (At hex address 6416C)
LDA.b #$0A : STA $012F
```

sound effects db:

0A = *'Nintendo' logo twinkle*
EF = *Pickup dungeon key*
20 = *Moving in the menus*
30 = *Blob (enemy)*
31 = *Moldorm (boss)*
32 = *RedOrb (Bouncy static things)*
37 =

Dummied out Frame control

Change addresses **$39** and **$3A** to 0x**EAs** (Default = 80 16). This NOPs out a Branch Always instruction. That's it. Now load up your game in an emulator.

Press the L button to freeze the game. Music will continue running as will animations like water and flowers, but sprites and Link will be frozen on the screen.

To advance by a frame, press the R button. To unfreeze the game, simply press L again. Only reason I never mentioned this was because I for some reason always thought the buttons it was checking were Select and Start (d'oh). Hope someone gains some amusement from this.

Hex offsets found by PuzzleDude:

Dungeon maps removal

At hex address **10908**,

value **F0** = maps are displayed when pressing X button in any dungeon

value **80** = maps are NOT displayed when pressing X button in any dungeon (optimal, since the dungeon maps are always bugged or inaccurate).

The Fire Sword

After studying the bytes which control the sword's power, I've come across something interesting. Aside the normal damage values, there are also some other possibilities. A lot of them are not actually recommended for use, but some are.

Like the 0B value, which makes the enemies burn. It is the same as they were hit by a fire rod. With this knowledge the implementation of the Fire sword is possible, but spin attack must be redefined to normal damage 03 or above, to be able to beat Ganon and some other flame resisting enemies.

address 36D33 = start

order = sword 1, 2, 3, 4;
spin attack with sword 1, 2, 3, 4
stab with sword 1, 2, 3, 4 = 12 bytes.

values:
00 - enemy freezes, but on second hit Link freezes (not recommended)
01 - damage level 1
02 - damage level 2
03 - damage level 3
04 - damage level 4
05 - damage level 5
06 - damage level 2
07 - enemy freezes, but on second hit Link freezes (not recommended)
08 - damage level 2
09 - damage level 5
0A - enemies will change into a hoping thing (but will display bug before that), no effect on strong enemies
0B - fire sword! = same as hitting the enemy with fire rod
0C - ice sword! = same as hitting the enemy with ice rod (some enemies are suddenly defeated, but don't get iced)
0D - fire sword
0E - ice sword
0F - enemies will change into a hoping thing (but will display bug before that), no effect on strong enemies
10 - sword cannot hit the enemy

0B - no effect on Ganon
minimum for Ganon is level-3 damage = 03

logical combination = 0B normal hit sword4
----------03 spin for sword4
----------01 stab for sword4

10 - sword cannot hit the enemy

11 - sword cannot hit the enemy
12 - sword makes half of level 1 damage, enemy is not bounced back
13 - sword makes half of level 1 damage, enemy is not bounced back
14 - sword cannot hit the enemy
15 - level 2 damage, enemy is not bounced back
16 - sword makes half of level 1 damage, enemy is not bounced back
17 - sword makes half of level 1 damage, enemy is not bounced back
18 - sword cannot hit the enemy
19 - sword cannot hit the enemy
1A to 1F - sword cannot hit the enemy
20 - enemy freezes, but on second hit Link freezes (not recommended)
21 - damage level 1, enemy is not bounced back
22 - damage level 2, enemy is not bounced back

further values to FF are supposed to be repeated or irrelevant (not recommended or not useful).

Green Magic Auto refill

This code is a modified version of the Woot2 ASM file by MathOnNapkins. This ASM makes various effects: like faster scrolling text, extensive modifications regarding bombs etc. This document only deals with green magic auto refill part of the new code.

The first thing you should do is remove the header and expand your game to 2MB. Open the file in hex, and delete the first 200 (in hex) 00 bytes = header, if the file has one. Then go to the end of the file (is 0FFFFF), choose insert bytes, number of bytes is 100000 in hex, with the value 00.

First step = New pointer
****

Go to hex address **000034** (this is right at the beginning).

You should see these bytes: **A5 12 F0 FC**.
Change them to **22 00 80 23** = New pointer, points to 118000.

This is mimicked from the woot2 ips file. Of course you can repoint this if you plan to have some other data at 118000.

Second step = New Code
****

Copy this new code at 118000 using a HxD hex editor. The code is 190 long in hex = 400 bytes in dec.

```
08 20 0F 80 20 80 80 20 66 81 20 62 80 28 6B 08 E2 20 A5 F6 29 20 F0 06 A5 9B 49 20 85 9B A5 F6 29
10 F0 3C AF EA B0 7F F0 04 C9 15 90 11 A9 00 8F 08 B0 7F AD 02 02 F0 27 8F 08 B0 7F 80 21 48 AD 02
02 48 8F 08 B0 7F 68 68 8D 02 02 22 6C FA 0D AE 02 02 BF 15 FA 0D 8D 03 03 A9 20 8D 2F 01 28 60 E2
20 AF 0D B0 7F CF 0C B0 7F F0 05 A9 20 8D 2F 01 A5 12 F0 FC AF 0D B0 7F 8F 0C B0 7F 60 08 E2 30 A5
11 05 5D D0 46 A6 10 BF 52 81 23 F0 3E AD 02 02 C9 0F D0 13 AF 04 B0 7F 18 69 10 8F 04 B0 7F 90 06
A9 FF FF 04 B0 7F AF 6E F3 7E AA BF D1 80 23 18 6F 04 B0 7F 8F 04 B0 7F AF 6E F3 7E 69 00 10 02 A9
80 8F 6E F3 7E 8F 05 B0 7F 28 60 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 08 AF 00 B0 7F
3A C9 00 10 02 A9 1F 8F 00 B0 7F 0A AA C2 20 BF 15 82 23 8F 06 B0 7F BF 95 8D 23 8F 0A B0 7F 28 60
EB 00 00 00
```

This code was written by MathOnNapkins, and modified by PuzzleDude for optimal performance. Open the game in emulator, and the green magic will fill up automatically.

Adopting the speed of auto fill
****

Do you see the group of 01 bytes? It starts at **1180D1** and is 95 (in hex) long. This is the speed of the auto fill. You need to fill the entire 95 (hex) block with the same value. Here's the table:

00 - no fill, speed at 0
01 - quite slow, but still normal for the game (selected as optimal)
02 - a bit faster, but still normal for the game

03 - faster, the limit of recommended
04 - faster, the limit of recommended

05 - very fast, not recommended, slightly too fast
06 - very fast, not recommended, slightly too fast
07 - even faster, not recommended, too fast
08 - much faster, not recommended, too fast

09 - much much faster, not recommended, too fast
10 - much much faster, not recommended, too fast
11 - much much faster, not recommended, too fast
..
1A - much much much faster, not recommended, way to fast
1B - so fast, that Cane of Byrna can be used to infinity, if you have half magic
..
3A - so very fast, that the repeated use of Fire rod will drain the green magic with difficulty
..
5A - super speed, infinite green magic
..
9A - ultra speed, infinite green magic

..
FF - maximal speed, infinite green magic

Complex auto fill
****

Do you want to make the green magic fill fast first and slow later? This is suitable, since when you have no magic, you want it to fill fast, but once you have some, it doesn't need to go fast any more, but normal, and when you have like 70 percent, the slow fill is acceptable.

Example

```
08 20 0F 80 20 80 80 20 66 81 20 62 80 28 6B 08 E2 20 A5 F6 29 20 F0 06 A5 9B 49 20 85 9B A5 F6 29
10 F0 3C AF EA B0 7F F0 04 C9 15 90 11 A9 00 8F 08 B0 7F AD 02 02 F0 27 8F 08 B0 7F 80 21 48 AD 02
02 48 8F 08 B0 7F 68 68 8D 02 02 22 6C FA 0D AE 02 02 BF 15 FA 0D 8D 03 03 A9 20 8D 2F 01 28 60 E2
20 AF 0D B0 7F CF 0C B0 7F F0 05 A9 20 8D 2F 01 A5 12 F0 FC AF 0D B0 7F 8F 0C B0 7F 60 08 E2 30 A5
11 05 5D D0 46 A6 10 BF 52 81 23 F0 3E AD 02 02 C9 0F D0 13 AF 04 B0 7F 18 69 10 8F 04 B0 7F 90 06
A9 FF FF 04 B0 7F AF 6E F3 7E AA BF D1 80 23 18 6F 04 B0 7F 8F 04 B0 7F AF 6E F3 7E 69 00 10 02 A9
80 8F 6E F3 7E 8F 05 B0 7F 28 60 04 04 04 04 04 04 04 04 04 04 04 04 03 03 03 03 03 03 03 03 03 03
03 03 03 03 03 03 03 03 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02 02 02 02 02 02 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 08 AF 00 B0 7F
3A C9 00 10 02 A9 1F 8F 00 B0 7F 0A AA C2 20 BF 15 82 23 8F 06 B0 7F BF 95 8D 23 8F 0A B0 7F 28 60
EB 00 00 00
```

This will fill green magic quite fast, when you have none, then a bit slower, even slower later etc, until reaching the quite low/normal speed of 01 around half way.

The BOOK OF MUDORA factor
****

The upper code actually contains 3 ASM mods, that are somehow connected, I've just put the other two on NOP (no operation), so you can easily turn them off or on, it is off by default.

The first one is The Book of Mudora factor. Once equipped it will fill your magic very fast. That's why this is off by default. If you use this mod, the green auto fill is really not recommended, but you must NOT put it on 00, or the book wont fill magic at all. Once you gain the book, the green potion from the Witch is really not necessary anymore.

You could for instance gfx-change the book into a green ruby, not having any green magic jars or green potions in the game, and no special signs, to read with the book. So you would then produce a new item basically. This is suitable for the dark world environment and big dungeons (if you run out of magic in a big dungeon it can be a problem, but now just equip the book/ruby, and the problem is solved).

This kind of item is also very useful when fighting Trinexx.

TURN the Book of Mudora feature ON = (fast green magic auto fill when equipped), at 1180A7.
FF = NOP = no operation = feature is OFF
8F = OP = operational = feature is ON.

But the 8F will only work if the standard green magic auto fill is bigger than 00. Obviously 01 is the best choice here, since the difference between the two fills must be big, for the book-fill to have sense.

The R-button factor
****

This feature will swap between two different items in the inventory. Open the menu, select an item and press R. Then move on the second item. Now press R repeatedly to swap between the two. Close the menu and the R will still swap in the game directly.

TURN the R button feature ON = at 118025.
EA = NOP = no operation = feature is OFF
08 = OP = operational = feature is ON (not recommended though, see problems)

PROBLEMS
****

I can Not recommend to have this feature for various reasons:
-if your game has a sub-menu for bottles, the swap between a bottle (sub menu) and a standard item (in the main menu) can confuse the game, or simply make false displays.

-if you swap while the inventory in opened, some emulators will not be able to handle the rendering, like snes9x, which is not a fan of this feature and will produce a small display bug in the very upper horizontal line of the entire game, so every swap will make a black line for a moment.

-to change the second item in use, you have to be somewhat smart to grasp how to do this, so a typical player will think that the swap is just random and will probably give up on the feature, since he will think that everything is randomly switching.

The problem is, that when switching, you don't know what your primary item is. When equipping the secondary item using the R swap, you must leave the selected item, to make the former primary item your secondary item. This is somewhat to complicated in comparison to one item idea. The thing selected is in use.

Original code by MathOnNapkins
****

```
08 20 0F 80 20 80 80 20 66 81 20 62 80 28 6B 08 E2 20 A5 F6 29 20 F0 06 A5 9B 49 20 85 9B A5 F6 29
10 F0 3C AF 08 B0 7F F0 04 C9 15 90 11 A9 00 8F 08 B0 7F AD 02 02 F0 27 8F 08 B0 7F 80 21 48 AD 02
02 48 8F 08 B0 7F 68 68 8D 02 02 22 6C FA 0D AE 02 02 BF 15 FA 0D 8D 03 03 A9 20 8D 2F 01 28 60 E2
20 AF 0D B0 7F CF 0C B0 7F F0 05 A9 20 8D 2F 01 A5 12 F0 FC AF 0D B0 7F 8F 0C B0 7F 60 08 E2 30 A5
11 05 5D D0 46 A6 10 BF 52 81 23 F0 3E AD 02 02 C9 0F D0 13 AF 04 B0 7F 18 69 10 8F 04 B0 7F 90 06
A9 FF 8F 04 B0 7F AF 6E F3 7E AA BF D1 80 23 18 6F 04 B0 7F 8F 04 B0 7F AF 6E F3 7E 69 00 10 02 A9
80 8F 6E F3 7E 8F 05 B0 7F 28 60 1B 1A 1A 19 18 18 18 18 17 17 17 17 16 16 15 15 14 14 14 14 13 13
13 13 12 12 12 12 11 11 10 10 09 09 09 09 08 08 08 08 07 07 07 07 07 07 07 07 06 06 06 06 06 06 06
06 05 05 05 05 05 05 05 05 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 03 03 03 03 03 03 03 03
03 03 03 03 03 03 03 03 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 00 00 00 00 00 00 00 00 01 00 01 00 01 00 00 00 00 00 00 00 00 08 AF 00 B0 7F
3A C9 00 10 02 A9 1F 8F 00 B0 7F 0A AA C2 20 BF 15 82 23 8F 06 B0 7F BF 95 8D 23 8F 0A B0 7F 28 60
EB 00 00 00
```

R button function = ON, Book of Mudora function = ON, Green magic auto fill = from 1B (very very fast = to fast) to 02 (normal for the game) The green magic auto fill is far too fast here (allows infinite cane of Byrna, when having half magic).

"For Parallel Worlds, the auto fill code was put as part of the redraw code of the hud - quite lazy I know since I have to disable it when the dialog box is up. Afraid that I can't help you much but the only thing I can tell you is that it fills up 1 magic every 255 cycles of the 1A (timer) variable, i.e. the code below gets put somewhere and get called."

```
LDA $1A A5 1A
AND #$7F 29 7F
BNE $0D D0 12
LDA $7EF36E AF 6E F3 7E
CMP #$80 C9 80
BEQ $05 F0 05
INC 1A
STA $7EF36E 8F 6E F3 7E
```

~ Euclid

Spinning hearts in the HUD (when gaining health)

At hex address **6F152**,

value of the byte is 68 (hearts spin on the right = A Link to The Past ''original'' position)
value of the byte is 06 (hearts spin on the left = Zelda - Parallel Worlds position)

Old HUD and MENU location

Bytes from **6DDA0** to **6FFC0** (on original rom):
Location of the hud and submenu as it is presented in A Link to The Past.

You must search for the values as they are in Alttp to match it with the hack,
Might not be on the exact same place but close to it.

Parallel Worlds HUD + Alttp Menu

A) spinning hearts in the HUD
address **6F152**, value must be 06 (spinning on the left = Zelda - Parallel Worlds)
old value is 68 (spinning on the right = Zelda – A Link To The Past)

B) new HUD position, from **6FB75** to **6FFC0**

For the header addresses simply go 32 lines down from the current address!

*** after hex editing = graphics must be changed!**, so that the new magic bar is horizontal!

***Old menu position:**

From **6DDA0** to **6FB74** (graphics must be compatible)

Globally

-MENU = from **6DDA0** to **6FB74**
-HUD = from **6FB75** to **6FFC0**

-MENU + HUD = from **6DDA0** to **6FFC0**

If you want to insert the New hud or menu, go to acmlm's board2, uploader, hack ips patches, and find Gates of Darkness by SePH. It has new menu and hud (like in parallel worlds), on these locations, compatible with the original rom + you have access to the data. But graphics must be changed to new menu and hud graphics.

The new menu (parallel worlds) has one problem = one space is empty

Old Menu position

from 6DDA0 to 6FB74 (no header + graphics must be compatible)

globally (no header)
-MENU = from 6DDA0 to 6FB74
-HUD = from 6FB75 to 6FFC0
-MENU + HUD = from 6DDA0 to 6FFC0

If you want to insert the New hud or menu, go to Acmlm's board2, uploader, hack ips patches, and find Gates of Darkness by SePH. It has new menu and hud (like in parallel worlds), on these locations, compatible with the original rom + you have access to the data. But graphics must be changed to new menu and hud graphics.
The new menu (parallel worlds) has one problem = one space is empty

Sound effects fix for Gates of Time/Darkness

Bytes from **C940E** to **C950B** on original rom = normal sound effects

On the Gates of Darkness romhack, the address is shifted from **C9402** to **C94FF**

When you hit with the sword against the wall or another (guard's sword) in Gates of Darkness/Time, you hear like something is hitting the wood. But in Alttp is sounds normal, like hitting on iron.

Also, if you collect a rupee in Gates of Time, the sound is to high and strong, it is different/normal in Alttp. Another thing if the arrow hits your shield (again the wrong sound, in comparison to Alttp).

So I've found the hex location of sound effects in Alttp and Gates of Time (some bytes shifted from the original), and copy/pasted the ones from Alttp to regain the correct sound effects.

Silver arrows fix for Gates of Time/Darkness

In Gates of Time, when you collect the silver arrows, they are in the false hud location. This is because the Hud is different, and the silver arrows were not adopted to the new hud. The new hex values fix this by moving them a bit to the right, to their correct place.

silver arrows go to rupees in the Hud

	Alttp	gates
6FB0C	1E	2A
6FB12	1F	2B
6FB18	20	2C
6FB1E	21	2D

Ganon tower – stairs overlay

At address **16F50** bytes 00 01

wrong 16 0C (no stairs)
ok 22 52 (stairs overlay after unlock)

Ganon gate barrier must be in area 43 on the exact same place as original,
the sprite 37-waterfal must be at the exact same place as the original.
7 crystals will then unlock the barrier and the upper bytes will make
a stairs overlay after the unlocking.

Room 0 effect

At address **27787**

value 02 is NE defeat enemy to open

value 3D is Defeat to open Ganon's door

hole/warp is room16

You can make the Ganon's door open in every room, but it would not lead to Triforce shrine. You can get there only from room0, with the special byte that controls this.

But with asm, this can be changed: In Parallel Worlds, you can come to Triforce shrine from room 14 as well.

Room 0, fight with Ganon, Special Torches

Sometimes if you edit room 0, Hyrule Magic destroys certain bytes, so the torches don't work.

At address **27380**, are bytes 0A,0B,0C,0D,0E,0F

Correct values are: **00 00 CA 8C F2 8C**

Hyrule Magic might change that to **wrong** values like such: **6A 00 3E 0E 4A 11**

3D Objects

From **4FF90** to **4FFFF** (7 lines)
normal = Zelda – A Link To The Past
hacked = Zelda - Goddess of Wisdom

Hex offsets found by Spane:

Change maximum of rupees

On address - 0x6DBEC - you see E8 03 <--- Maximum amount of rupees (in dec = 1000)
On address - 0x6DBF1 - you see E7 03 <--- There where the rupees counter stops (in dec = 999)

A little test:

Open your windows calculator and choose dec. Now type 100 and convert to hex. You see 64 so your bytes are 00 64 <--- Maximum amount of rupees.

Type 99 and convert and you see 63 so your bytes are 00 63 <--- rupees counter will stop here.

Go to address 0x6DBEC and replace the E8 03 by 00 64.
Go to address 0x6DBF1 and replace the E7 03 by 00 63.

Save the rom in your hex editor and open it in Hyrule magic. Go to entrance 01 and put 300 rupees (number 70) in a chest. Play the rom on your favourite emu and open the chest. The rupees counter will stop at 99.

Euclid managed to change the bytes to 10 27 (10000) and 0F 27 (9999) so you can hold up to 9999 rupees in parallel worlds but i think the rupees counter needs asm to hold the 1000 digits

Remove annoying low life beeping sound

Go to the address 0x6DCA1 and you see the byte 2B. Change it to 00 and you will have no annoying low life beeping sound or replace it with other bytes and you will have other sounds Play a little bit with the bytes

Hex offsets found by XaserLE:

Big keys edit

At address **EC1A** big chest is opened with
64 is compass, 66 is big door key, 68 is map
Default is 66

At address **EB8D** small lock in prison is opened with
64 is compass, 66 is big door key, 68 is map
Default is 66

Added by PuzzleDude
Map in the inventory is at
(**6F9D1**) and **6FAC9** = DE 28, DF 28, EE 28, EF 28
28 is yellow colour, DE, DF, EE, EF are gfx coordinates, load different gfx for big chest key here

Compass in inventory is at
6FAA1, BF 24 (is red), BF 64 (is red mirrored), CF 2C (is blue), CF 6C (is blue mirrored)
Change all colours to 28 (is yellow for the big key)

Big key in the inventory is at
6FAB1 = D6 28, D6 68 (yellow mirrored), E6 28, E7 28.
28 is yellow colour, D6, D6, E6, E7 are gfx coordinates.

Hex offsets found by NEONswift:

Intro cutscenes debugging

add pictures!!!!

CPU LoROM Address	Hex Address	Description
$02:C8FB	0x148FB	First room of Intro Cutscene (2 bytes) (default = 51)
$02:CF37	0x14F37	First room vertical starting co-ordinates (2 bytes)
$02:CE2D	0x14E2D	First room Horizontal starting co-ordinates (2 bytes)
$02:C8F9	0x148F9	Second room of Intro Cutscene (2 bytes)
$02:C8FD	0x148FD	Final room of Intro Cutscene (2 bytes)

```
$02/D9A3 BD 13 C8    LDA $C813,x[$02:C8FB]   A:03A0 X:00E8 Y:03A0 P:envmxdizc <-Loads room # into
Accumulator
$02/D9A6 85 A0       STA $A0    [$00:00A0]   A:0051 X:00E8 Y:03A0 P:envmxdizc <-Stores room # in
RAM [Current Area]
$02/D9A8 8D 8E 04    STA $048E  [$02:048E]   A:0051 X:00E8 Y:03A0 P:envmxdizc <-Stores room # in
RAM [Grab Hand Sprite]
$02/D9AB BD 4F CE    LDA $CE4F,x[$02:CF37]   A:0051 X:00E8 Y:03A0 P:envmxdizc
$02/D9AE 85 E8       STA $E8    [$00:00E8]   A:0B10 X:00E8 Y:03A0 P:envmxdizc
$02/D9B0 85 E6       STA $E6    [$00:00E6]   A:0B10 X:00E8 Y:03A0 P:envmxdizc
$02/D9B2 8D 22 01    STA $0122  [$02:0122]   A:0B10 X:00E8 Y:03A0 P:envmxdizc
```

```
$02/D9B5 8D 24 01    STA $0124  [$02:0124]   A:0B10 X:00E8 Y:03A0 P:envmxdizc
$02/D9B8 BD 45 CD    LDA $CD45,x[$02:CE2D]   A:0B10 X:00E8 Y:03A0 P:envmxdizc
$02/D9BB 85 E2       STA $E2    [$00:00E2]   A:0280 X:00E8 Y:03A0 P:envmxdizc
$02/D9BD 85 E0       STA $E0    [$00:00E0]   A:0280 X:00E8 Y:03A0 P:envmxdizc
$02/D9BF 8D 1E 01    STA $011E  [$02:011E]   A:0280 X:00E8 Y:03A0 P:envmxdizc
$02/D9C2 8D 20 01    STA $0120  [$02:0120]   A:0280 X:00E8 Y:03A0 P:envmxdizc
$02/D9C5 AF C5 F3 7E LDA $7EF3C5[$7E:F3C5]   A:0280 X:00E8 Y:03A0 P:envmxdizc
$02/D9C9 F0 0A       BEQ $0A    [$D9D5]      A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/D9D5 BD 6D D1    LDA $D16D,x[$02:D255]   A:0000 X:00E8 Y:03A0 P:envmxdiZc <-Loads y cam scroll
co-ords (0188) into Accumulator
$02/D9D8 8D 18 06    STA $0618  [$02:0618]   A:0188 X:00E8 Y:03A0 P:envmxdizc <-Stores y cam scroll
co-ords in RAM (lower bound)
$02/D9DB 1A          INC A                   A:0188 X:00E8 Y:03A0 P:envmxdizc <-Increments y cam
scroll (Accumulator) by 1
$02/D9DC 1A          INC A                   A:0189 X:00E8 Y:03A0 P:envmxdizc <-Increments y cam
scroll (Accumulator) by 1 again
$02/D9DD 8D 1A 06    STA $061A  [$02:061A]   A:018A X:00E8 Y:03A0 P:envmxdizc <-Stores y cam scroll
co-ords in RAM (upper bound)
$02/D9E0 BD 77 D2    LDA $D277,x[$02:D35F]   A:018A X:00E8 Y:03A0 P:envmxdizc <-Loads x cam scroll
co-ords (00FF) into Accumulator
$02/D9E3 8D 1C 06    STA $061C  [$02:061C]   A:00FF X:00E8 Y:03A0 P:envmxdizc <-Stores x cam scroll
co-ords in RAM (lower bound)
$02/D9E6 1A          INC A                   A:00FF X:00E8 Y:03A0 P:envmxdizc <-Increments x cam
scroll (Accumulator) by 1
$02/D9E7 1A          INC A                   A:0100 X:00E8 Y:03A0 P:envmxdizc <-Increments x cam
scroll (Accumulator) by 1 again
$02/D9E8 8D 1E 06    STA $061E  [$02:061E]   A:0101 X:00E8 Y:03A0 P:envmxdizc <-Stores x cam scroll
co-ords in RAM (upper bound)
$02/D9EB A9 F8 01    LDA #$01F8              A:0101 X:00E8 Y:03A0 P:envmxdizc <-Loads #$01F8 value
intermediate into Accumulator ???
$02/D9EE 85 EC       STA $EC    [$00:00EC]   A:01F8 X:00E8 Y:03A0 P:envmxdizc <-Stores $01FB into
address $00:00EC ???
$02/D9F0 BD 24 D7    LDA $D724,x[$02:D80C]   A:01F8 X:00E8 Y:03A0 P:envmxdizc
$02/D9F3 8D 96 06    STA $0696  [$02:0696]   A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/D9F6 9C 98 06    STZ $0698  [$02:0698]   A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/D9F9 A9 00 00    LDA #$0000              A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/D9FC 8D 10 06    STA $0610  [$02:0610]   A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/D9FF A9 10 01    LDA #$0110              A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/DA02 8D 12 06    STA $0612  [$02:0612]   A:0110 X:00E8 Y:03A0 P:envmxdizc
$02/DA05 A9 00 00    LDA #$0000              A:0110 X:00E8 Y:03A0 P:envmxdizc
$02/DA08 8D 14 06    STA $0614  [$02:0614]   A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/DA0B A9 00 01    LDA #$0100              A:0000 X:00E8 Y:03A0 P:envmxdiZc
$02/DA0E 8D 16 06    STA $0616  [$02:0616]   A:0100 X:00E8 Y:03A0 P:envmxdizc
$02/DA11 AD 0E 01    LDA $010E  [$02:010E]   A:0100 X:00E8 Y:03A0 P:envmxdizc
$02/DA14 29 FF 00    AND #$00FF              A:0074 X:00E8 Y:03A0 P:envmxdizc
$02/DA17 AA          TAX                     A:0074 X:00E8 Y:03A0 P:envmxdizc
$02/DA18 E2 20       SEP #$20                A:0074 X:0074 Y:03A0 P:envmxdizc
```

Exits to new areas – Lost Woods Master Sword Area and under the bridge

CPU LoROM Address	Hex Address	Description
$0E:DE31	0x75E31	The area for 8x8 367 & 261 (Entrance to Master Sword Area - Exit: 0180)
$0E:DE33	0x75E33	The area for 8x8 173 (Entrance to Zora Domain Area - Exit: 0182)
$0E:DE35	0x75E35	The area for 8x8 484 (Entrance to Bridge Area - Exit: 0181)

```
$0E/DE66 A5 8A       LDA $8A    [$00:008A]   A:0105 X:0000 Y:0514 P:envmxdiZC <-Load Accumulator
$8A (Current Overworld Location = 00)
$0E/DE68 DF 31 DE 0E CMP $0EDE31,x[$0E:DE31] A:0000 X:0000 Y:0514 P:envmxdiZC <-Compare
$8A/Accumulator with $0E:DE31 (most likely also 00)
```

```
$0E/DE6C D0 EC       BNE $EC    [$DE5A]      A:0000 X:0000 Y:0514 P:envmxdiZC <- Zero Flag "Z" is
not clear so no Branching to $DE5A
$0E/DE6E BF 41 DE 0E LDA $0EDE41,x[$0E:DE41] A:0000 X:0000 Y:0514 P:envmxdiZC <-Load A from
$0E:DE41 (0180..The exit at Master Sword Area)
$0E/DE72 85 A0       STA $A0    [$00:00A0]   A:0180 X:0000 Y:0514 P:envmxdizC <-Store A (0180) in
RAM $A0 (Current Dungeon Area)
$0E/DE74 E2 20       SEP #$20                A:0180 X:0000 Y:0514 P:envmxdizC <-SEP #$000100000
Sets the Memory/Accumulator to 8 bit (M=1)
$0E/DE76 BF 39 DE 0E LDA $0EDE39,x[$0E:DE39] A:0180 X:0000 Y:0514 P:envMxdizC ???<-Load $0E:DE39
(0108) into the Accumulator
$0E/DE7A 85 67       STA $67    [$00:0067]   A:0108 X:0000 Y:0514 P:envMxdizC ???<-Stores 0108 into
RAM $67
$0E/DE7C 8D 10 04    STA $0410  [$00:0410]   A:0108 X:0000 Y:0514 P:envMxdizC ???<-Stores 0108 into
RAM $0410
$0E/DE7F 8D 16 04    STA $0416  [$00:0416]   A:0108 X:0000 Y:0514 P:envMxdizC ???<-Stores 0108 into
RAM $0416
$0E/DE82 A2 04 00    LDX #$0004              A:0108 X:0000 Y:0514 P:envMxdizC ???<-Loads #$0004
immediate into X Memory
$0E/DE85 CA          DEX                     A:0108 X:0004 Y:0514 P:envMxdizC <-Subtracts one from
X Memory (0004 becomes 0003)
$0E/DE86 4A          LSR A                   A:0108 X:0003 Y:0514 P:envMxdizC <-Logical Shift right
making carry flag clear (c=0)
$0E/DE87 90 FC       BCC $FC    [$DE85]      A:0104 X:0003 Y:0514 P:envMxdizc
$0E/DE89 8E 18 04    STX $0418  [$00:0418]   A:0100 X:0000 Y:0514 P:envMxdiZC
$0E/DE8C 8E 9C 06    STX $069C  [$00:069C]   A:0100 X:0000 Y:0514 P:envMxdiZC
$0E/DE8F A9 17       LDA #$17                A:0100 X:0000 Y:0514 P:envMxdiZC
$0E/DE91 85 11       STA $11    [$00:0011]   A:0117 X:0000 Y:0514 P:envMxdizC
$0E/DE93 A9 0B       LDA #$0B                A:0117 X:0000 Y:0514 P:envMxdizC
$0E/DE95 85 10       STA $10    [$00:0010]   A:010B X:0000 Y:0514 P:envMxdizC
```

Hex offsets found by Jonwil:

What Zora King Sells To You

EE1C3 (Default value = **1E** = Zora's Flippers)

What The Zora King's Item Looks Like

EE1E5 (Default value = **11** = Zora's Flippers graphics)

Magic Shop Items (potion shop) so they obviously sell potions...

Magic shop item 1: **2F595** (Default value = **02 = ??**)potion which one?
Magic Shop Item 2: **2F5C6** (Default value = **03 = ??**) potion which one?
Magic Shop Item 3: **2F5F7** (Default value = **04 = ??**) potion which one?

Hex offsets found by Gideon Zhi:

Make Link Sink

$009A - BG blend settings - set to 72 to enable sprite blending on BG2
6A129/0D:A128 - Link's sprite priority (before it gets chunked into RAM) - Change it from $20 to $18 to place him

underneath BG2
3D7D8/07: D7D8 - Jump table for handling movement stuff. The pointer at D7E8 ($BCDC) points to the splash-into-water handler. The pointer at D852 ($9DDD) points to the walk-down-stairs handler.

To implement the hack, simply, change the blend settings in a save state, Link's sprite priority in the rom, and the pointer for the splash-into-water handler to the walk-down-stairs handler.

Of course, to implement this stuff in a practical, useful way, you'd have to do a whole lot more, but it's kind of neat to see

Oh, and by the way, the OAM table is stored at $0800 in wram. Fun!

CPU Addresses (convert all into hex!!!!)

How Much You Have To Pay Zora King for The Flippers **0x29A9A in hex** (Default **F4 01** = **500**).

Minimum Sword Level To Shoot Sword Beams

0x3958A or 0x39E8B **(?)cfdfbdnbc use Euclid asm and convert to hex!!!!**

Swords default damage + Spin attack

Sword 1 Damage
0x6BAFA

Sword 1 Spin Damage/Sword 2 Damage
0x6BB02

Sword 2 Spin Damage/Sword 3 Damage
0x6BB0A

Sword 3 Spin Damage/Sword 4 Damage
0x6BB12

Sword 4 Spin Damage

0x6BB1A

Text Layout

0x73B3D onwards.

Start of Credits Data

0x73378

Start of Overworld Scrolling Text Data

0x74150

Layout of the outdoors

0x1717E - 0x30000

Magic Usage rom offsets

(Includes header)
1/2 magic - +1
1/4 magic - +2

Note: Maximum amount of mana is 0x80

0x3B287 - Fire/Ice Rods
0x3B28A - 3 spells
0x3B28D - Magic Powder
0x3B290 -
0x3B293 - Red Staff (block making one)
0x3B296 -
0x3B299 - Torch
0x3B29C -
0x3B29F - Blue Staff (start using)

The blank ones are data as well but we don't know yet what they do.

0x7EF37B - 01 = 1/2 Magic, 02 = 1/4 Magic (I did a rough search on **STA 7EF47B** and find nothing which stores a 2 since they all have **LDA #01** in front of it)

As with the cape, CPU address **07/AE32** - change to **EA** to have Infinite Cloak (or change it to **INC opcode** to make you gain magic!)

9) GameGenie codes to HEX converting guide by Weasel

Programs required:
1) Windows Calculator (Set to scientific mode)
2) GGConvC (This utility converts a game genie code to a raw hex code)
3) Gg (This one will make GG codes turn into Pro Action Replay codes)
4) gg-hex (This utility can convert GG to PAR and PAR to GG)

Game Genie to Pro Action Replay!

Okay, let's say you want to use a game genie code. For a game called Final Fight 2, there's a code that makes you start a new game with 10 lives. But you can't get very far with 10 lives. That's sad, my friend. But beside the point. Altering GG codes can be a pretty bad thing to mess with as some GG codes can erase save files, mess with the batteries, and just screw things up.
You don't want THAT to happen, right? Of course not. That's why **Gg** and **gg-hex** were invented.

Gg must be used from the DOS command line. If you don't know what that is, you need to be slapped with a large trout. So, at the command line, run **Gg** like so:

gg dbcf-c7d6

The program will print:

82a21c09

The printed part of the program is a PAR code. Now, if you know anything about PAR codes, their formats are **xxxxxx:yy**. The **x**'s are the RAM offsets. This means next to nothing. The **y**'s are the modified bytes. THAT is what you want to mess with, peoples! Instead of screwing with the GG code and possibly f-ing your game up, you can mess with the y bytes. This doesn't
screw up the game because it has nothing to do with the game's actual code. It changed memory, no big deal. Cool, huh? So in the PAR code above, change the **09** (which is **10** in decimal) to, let's say **10**. Remember, this is in hexadecimal, so **10** is
actually **16**. Now, in the game, use the code **82121c10**. You will start the game with 16 lives. It's that simple.

Note: Since PAR codes affect memory, they are outside the range of the actual locations of the game. You can't convert PAR to GG for this reason. Don't try, it'll just waste your time.

Game Genie to Hex address!

If you're a rom hacker, THIS is what you were probably looking for. There's a code for a game called Seiken Densetsu 3 that can give Duran a bit for Vitality points. **25** to be exact. FB78-80A5 is that code you want. Now, open up **GGConvC** and

select SNES. For this example, I am using a Hirom. Hirom and Lorom examples are different as their range for offsets can vary greatly.

Back to the example, enter the code above into the proper area in the program. It will convert it to **D13EB0:19**. This looks a lot like a PAR code, doesn't it? It has the format **xxxxxx:yy**. It works the same way, the **y**'s being the modified byte. **But this ISN'T the actual hex address.** If you test this, you know it's out of range. So what to do? Here's the method.

Open the calculator, in scientific mode. Select **hex** on the left side. Type in **D13EB0**. Subtract **C00000**. Add **200**. Guess what, you maniac! That's the actual offset! Now, open the rom in a hex editor, go to the new address (it's **1140B0**) and type **19** for the byte you find there, it should be a **05**. Now, save it. The code is permanently patched to the rom. You beast, you did it! Yay!

Lorom conversions are a bit different but not by much. Simple type in the code into the program like above. But this time, press the Show 64k button. Subtract and add like above. Viola! you have the offset for the code.

This process is wonderful for finding data in roms. If there is a code that screws up a room's graphics, chances are that the code moves stuff around and changes level data.

ADDITIONAL NOTES by Euclid:

0780C7CF - Monster touch you one time and your dead.
0780F7CF - Give you infinite energy.
098133CF - Give you infinite bombs.
1EF3AECF - Shops don't take your money when you buy.

Those ones are probably CPU addresses which you can do in the rom straight away.

You first probably have to know how these codes work:

xxyyyyzz - Codes are structured like this

xx - Bank number (7E means it's in the work ram, which gets saved in the .sav file i think, 0 - 3F or 80 - BF usually refers to rom banks i.e. actual data in the.smc file)

yyyy - Refers to the actual address in the rom, 8000 - FFFF refers to values in the .smc file. Usually anything below that refers to the ram.)

zz - The value to write in this offset.

You first have to know that these codes just changes your rom in memory slightly, overwriting some asm instruction so instead of doing say a subtraction, do a useless thing (e.g./ CMP) which means whatever the original code there will not work.

Most codes starting with 7E you probably can't.

~ Chapter VI – ASM Hacking

If you've hacked (or tried to) Zelda 3, you've probably heard the names MathOnNapkins and Euclid by now. It's fair to say, every time my rom screw up over, these guys were always there to help. They usually fixed your problem in a simple post! If you thought something was impossible due to the Hyrule Magic editor being limited, their out-of-this-world knowledge has proved otherwise with their awesome ASM hacks!

Without further-ado let's start this ASM section with their rom hacking/ASM teachings! These lessons may prove more than useful if you intend on start making Zelda 3 ASM hacks yourself!

~Orochimaru

1) MathOnNapkins Teachings
by MathOnNapkins

~ MathOnNapkins ™ Lesson 1 ~

Introduction to SNES LTTP Hacking

Over the years I've seen a lot of questions about how to get started with ASM hacking, and a lot of people tend to shy away from it. I'm going to assume for now that it's because many people don't know the basics and don't know where to learn them.

I'm not sure how productive this tutorial will be but I decided to write it to see if it would help anybody out. Several years ago I started writing a tutorial like this, but it was on a forum that is now defunct and I had little relation to the forum in the first place - I can't even remember what it's called but it was Zelda related. So let's try that again, shall we?

A) SNES ROMS

Disclaimer: *I am not a hacking scene historian, I make no claim to the absolute accuracy of what follows in this section.*

In the beginning there were ***SNES cartridges***. If you grew up when the SNES was new, you went to your favourite rental store and hoped "The Legend of Zelda: A Link to the Past" was in.

Often it wasn't.

If you managed to rent it, you made sure you beat the game during your rental because the next dude or dudette would most likely be erasing all the games on the cart and starting their own games by the next time you rented it - assuming you could get the same copy of the cart in the first place.

Or if you or your parents had the money, you waltzed down to a game store and just bought the damn thing. Nobody overwriting your game now, eh? No matter how you got it, you stuck it in your system and had a grand old time.

You never really know how the game was stored on the cartridge but it didn't matter because you were having fun.

Fast-forward a few years into the life of the SNES and companies in China / Hong Kong began making ***"copier"*** hardware which had the capability to dump the contents of SNES carts to one or more floppy disks, or even to a PC hard drive with the right hardware setup.

Inevitably, copies of these ROMs made it onto the rapidly growing internet. If you were lucky enough to own a copier unit, you could dump your own cartridges yourself. Many people naturally took the easier route and downloaded copies of their favourite SNES games from the internet, though.

*****I'm not going to address the legality of this action in this tutorial.*****

With SNES emulators being written by a wide variety of hobbyists, people could finally play their SNES games on systems other than an SNES.

Interest in SNES homebrew programming sprouted out of the ***"demo"*** scene, where hobbyist programmers would try to make new SNES programs from scratch. The demos were not usually games. They were typically introductions (***"intros"***) designed to execute right before popular games and were redistributed on the internet in different form.

Essentially these were hacked roms that didn't change the actual gameplay in any way. The reason people made these was to show off their programming skills among other programmers.

Some demos were ***standalone ROMs*** demonstrating different features of the SNES hardware. Typically these were graphics, but sometimes roms that played music were crafted as well.

The interest coming out of the demo scene motivated research of the SNES hardware. Soon many documents describing various features of the SNES were made available on the web.

These documents included tutorials on the ***65c816 processor***, which is the ***main CPU*** of the SNES, sometimes referred to as the ***S-CPU*** by SNES enthusiasts.

Documents on the ***SPC-700*** (the sound module) could also be found, though most were of poor quality. The SPC-700 is actually made up of two components: the ***S-SMP***, a microprocessor that runs code that runs a game's sound engine, and the ***S-DSP***, the digital sound processing unit that actually mixes and outputs the sound for the system.

Eventually ***ROM modders*** started to appear. Unlike the demo scene, these people sought to modify their favourite commercially released games and bend them to their own dreams and imaginations.

These ***ROM "hackers"*** are people that do whatever it takes to produce a desired result in a game. I generally consider modding and hacking of a game to be identical.

Some would make a distinction that modding is of a lower level, but I'm not going to here. Rom Hacking is the skill of figuring out how a classic game works and trying to modify it to produce an intended result. The ends don't really matter, if you ask me.

Generally, ROM Hackers modify one of two things - ***data*** or ***code***. *Using a* ***level editor*** *would be considered hacking data.* ***Modifying how a power up item in a game works*** *would be considered hacking code.* Things can get blurry, because sometimes hacking code requires modifying data.

Changing values in a hex editor *is considered hacking data - but it is very much possible to* ***edit code in a hex editor as well.*** *(And woe to you if you've edited some code and didn't realize it! I hope your ROM doesn't crash.)*

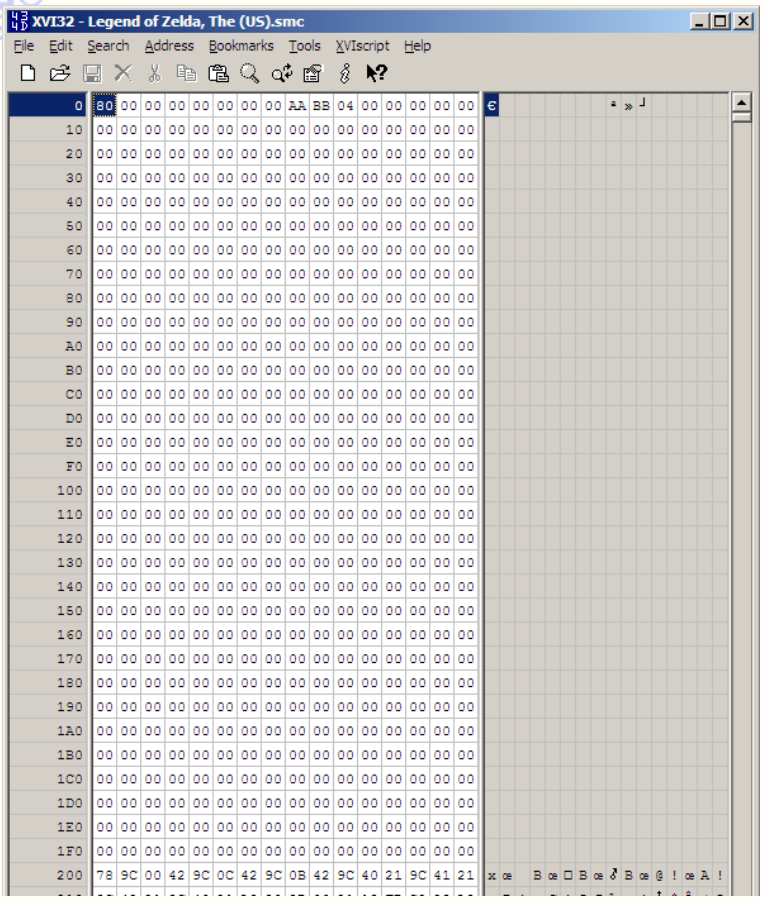
We're going to examine the ***LTTP ROM*** for starters. ***ROM extensions*** are usually "**.sfc**" or "**.smc**". I hope it goes without saying that ***.zip/.rar/.7z*** are not rom extensions. If your file is compressed in one of these formats, make sure to extract a .smc or .sfc file from the archive.

B) ROM HEADERS

There were multiple companies that produced SNES Cartridge copiers of ***"dumpers"*** as they are sometimes called. Most if not all of these inserted information before the ROM. These are usually called ***headers***. More properly they are called ***"copier headers"*** for clarity, because there is an actual ***"internal"*** header stored in the ROM that Nintendo required all developers to include in games they produced. Copier headers, when present, are always ***0x200 bytes in length***, as you'll see below.

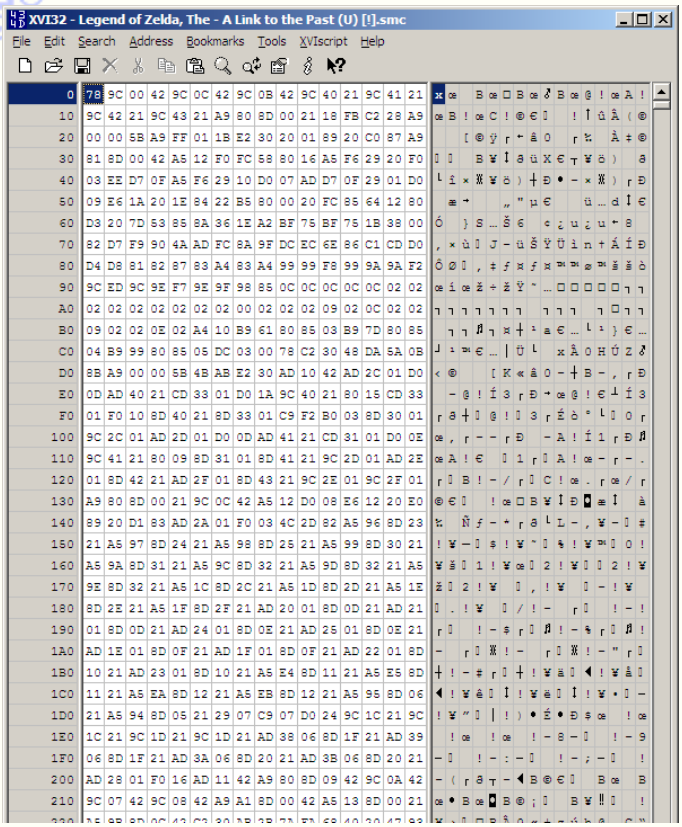
So why bring this up? Because headers can cause confusion when one ROM Hacker talks to another. Copier headers contain information that is not useful to emulation. The information is only useful to the cartridge copier device when it needs to read a file back in and play it (via your SNES or Super Famicom).

Let's look at the very beginning of ***a ROM that has a header***. They generally look like this in a hex editor:



Notice how the bytes from address ***0x0*** to ***0x1FF*** are mostly zero bytes.

A ROM without a header looks like this in a hex editor:



Note that the bytes at ***0x200*** in the headered ROM match the bytes at ***0x0*** in the unheadered ROM. Namely, this:

```
78 9C 00 42 9C 0C 42 9C 0B 42 9C 40 21 9C 41 21
```

This is true for all offsets in the ROM. The actual data that is part of the game starts at ***0x200 in the headered ROM***, and at ***0x0 in the unheadered ROM***. This can cause confusion when someone refers to a location within a file.

If I were to say, go to address ***$10000*** and modify bytes there in your hex editor, naturally you'd want to know if I was talking about a headered ROM or an unheadered ROM.

Because of this confusion, some hackers strip the headers off of their ROMs to simplify finding locations in the file.

It's becoming more common these days to be able to download a ROM that doesn't have a header, but it seems to be a relatively recent phenomenon by my estimation.

Headers also complicate things for people writing game editors, like ***Black Magic***. Black Magic does some math on the size of the file to figure out whether your ROM has a header or not.

If you know how *bitwise* ***AND*** works in the C programming language, it looks similar to this:

```
if(fileSize & 0x200)
{
    hasHeader = true;
}
```

As you edit your ROM in an editor, such as Black Magic, it has to remember whether there was a header on the ROM. When it saves the game after you're done editing, it makes sure it's left in the edited ROM.

Some editors, like Lunar Magic for Super Mario World, actually require a copier header be present. There is a rationale for why the editor creator did this. If you mandate that all ROMs have a header, then ***IPS patching*** is generally much simpler.

> *"Trying to use an **IPS patch** created using headered ROMs with an unheadered ROM generally will not work, and will likely corrupt your ROM. The same works in reverse when patching headered ROMs with IPS patches created with unheadered ROMs."*

The discussion on whether headers are bad or good is something of an ongoing political struggle in the ROM Hacking community, and my personal position is that they should be discarded while you're doing your ROM Hacking.

If you finish a hack, it may be advisable to put a header back on before creating an IPS patch of your work. Either way, it's courteous to at least provide a readme stating what type of ROM the IPS should be applied to.

Ideally you would provide IPS patches for both types, as some hackers like *Drewseph did for Super Metroid Redesign*. A more ideal solution would be to use a patching technology that is smart enough to make headers not matter, like ***xdelta*** for example.

The next section will give instructions on how to remove and how to add a header to a ROM.

C) ADDING OR REMOVING A HEADER

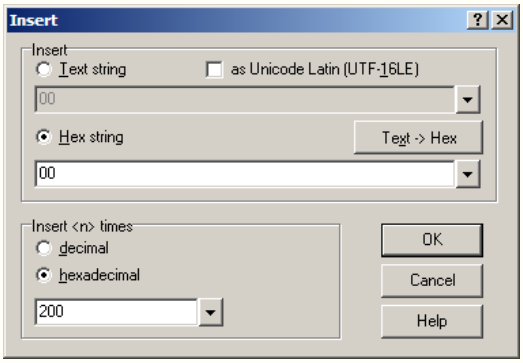
I'm well aware that there are many ***hex editors*** out there you could use. They all have their own merits and drawbacks. Myself, I've been using ***XVI32*** for years, and while it has some drawbacks, it has suited me well.

Your hex editor may or may not have the functionality I'm about to describe. If it doesn't, you may want to download XVI32 just for the sake of learning here.

Adding a header

Because copier header information is not typically used by emulators, we can just insert ***0x200*** zero bytes at the beginning of our ROM. Now, I want to make the obvious point that you should check first whether your ROM has a header already before trying to add one. As stated earlier, the first ***0x200*** bytes will be nearly all zeroes if you have a header. This is just a rule of thumb, though. I can't speak for 100% of all other games out there.

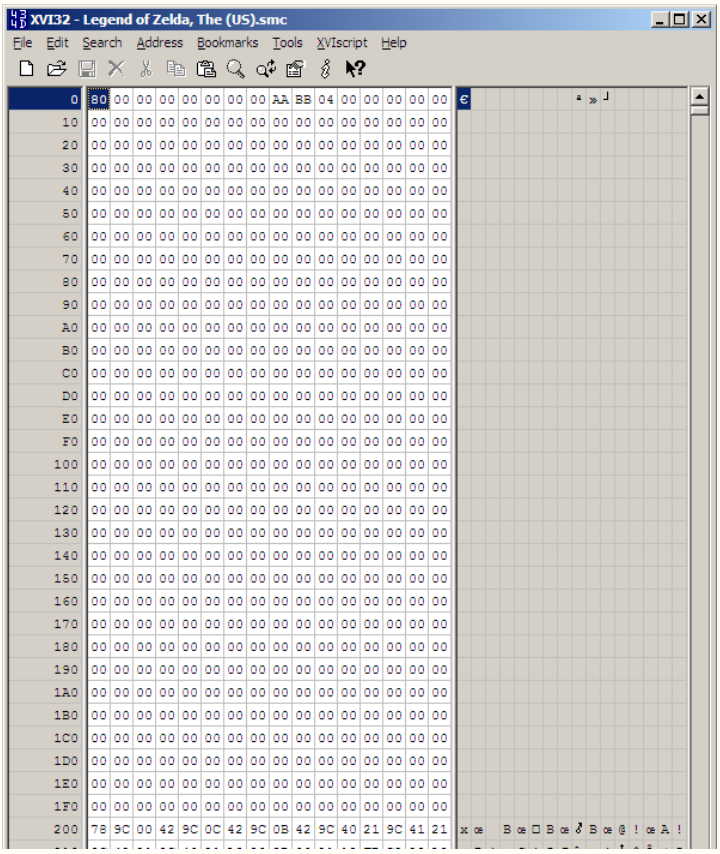
*"Navigate to the first byte (byte **0x0**) of your headerless ROM. The cursor should be hovering over a "**78**" byte. Choose the menu command "**Edit** > **Insert String**..." which will bring up a dialog that looks like this:*

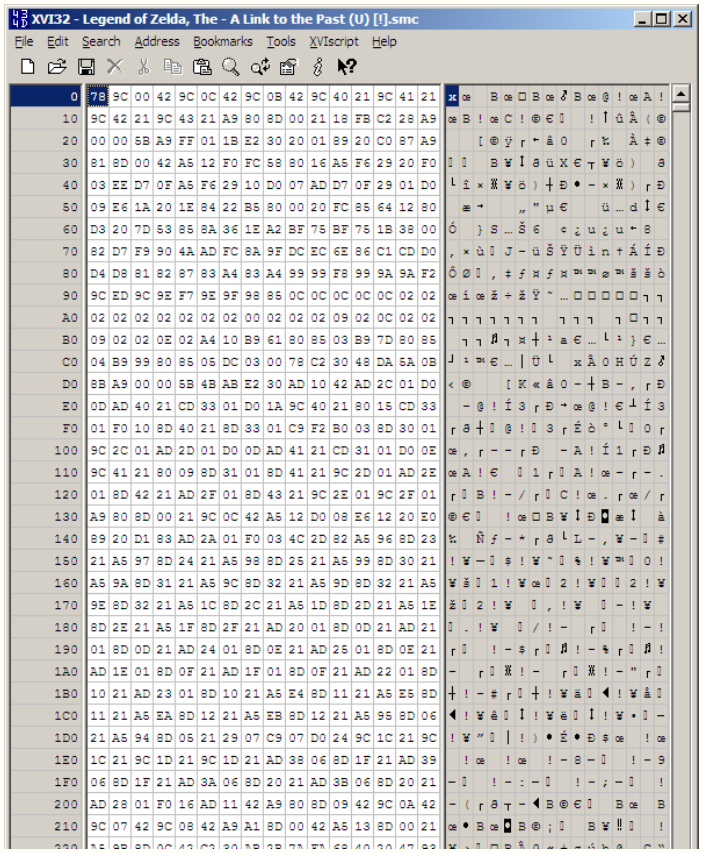


*Make sure to put a single "**00**" byte in as a hex string, and make sure hexadecimal is selected in the bottom left section. Also make sure the value in the edit box is **200** (it might read as **$200** as well if the edit box gets refreshed). Hit the OK button. You should now have a simulated copier header at the beginning of your ROM."*

*You can do a "**Save As...**" if you don't want to chance ruining your original copy.*

The ROM should look like this:





Removing a header

Using ***XVI32***, navigate to byte ***0x1FF***. The cursor should be hovering over a "***00***" byte. The very next byte should be the aforementioned "***78***" byte. Choose the menu command "***Edit*** > ***Delete to cursor***".

Make sure it's this option because the other option "***Delete from cursor***" will annihilate everything *but* the header and you'll have to reload your ROM in the hex editor. If performed successfully, the first byte in your ROM should now be the "***78***" byte. It should look like on the screenshot on the left.

D) ANATOMY OF A SNES ROM

Now that we understand where the ROM data begins, we have another problem. The ROM loaded in our hex editor just looks like a huge jumble of numbers! Well cheer up... imagine how much more formless it would look if we were looking at the ROM in binary ones and zeroes.

Earlier I mentioned that there are two types of information you can edit in a ROM: code and data. In this sea of hexadecimal numbers, how do we tell what is code and what is data? Recall the first line* of ROM data I mentioned earlier:

```
78 9C 00 42 9C 0C 42 9C 0B 42 9C 40 21 9C 41 21
```

(* Side note: it's common, but not universal, to define a line of hexadecimal numbers as 16 bytes as above. But that's what a "line" means in this context.)

As it turns out, this is the very beginning of the code in the game. But that doesn't really tell us anything. We don't know what this series of hexadecimal numbers means, we don't know what "78" means, or "9C" means, etc. How do we even know this is the start of the game's code? I'll get to that soon. I want to go over the general structure of a ROM first, and how it actually gets accessed from the SNES CPU (the 65c816 processor).

Banks:

At first glance it might seem that the ROM has no structure, just a linear stream of bytes. However, it was created with a very specific structure in mind. You may have seen the terms LoROM or HiROM used to refer to different types of ROMs.

The truth is that there are no real systems called LoROM or HiROM, but they're abstractions that simplify our understanding of how ROMs are laid out.

When you load up your LTTP ROM in say, Snes9X, if you're a fast enough reader you'll see "LoROM" on screen right after the game loads for a second or two.

We could talk about ***HiROM*** as well, but for now we're going to focus on ***LoROM*** because that's what LTTP is considered to be from an emulator standpoint.

Let's first get into what a "bank" is. In years past a memory "bank" was often heard of in popular culture, such as a robot saying "I have stored that information in my memory banks."

The term is a bit out of date these days, as computers often don't use banks anymore.

In *Star Trek 2: The Wrath of Khan*, a scientist refers to data stored in memory banks. Yes, that movie was written in the 80s. How did you guess?

Even the old CPUs in Intel PCs didn't use banks. They used something else called a "segment." It's common these days for RAM (random access memory) to be accessed in a flat fashion.
But back in the old days, one common way to organize the layout of a processor's RAM was to divide it into banks.

The SNES' processor, the 65c816, divides its RAM into separate blocks, each being 64 kilobytes in size. Each of those blocks is called a "bank." For reference, that's 0x10000 bytes in hexadecimal, and 65,536 bytes in decimal.

The 65c816 has 256 (0x100) banks at its disposal. That adds up to 16 Megabytes of memory (0x100 banks * 0x10000 bytes per bank) = 0x1000000 bytes, or 16 megabytes.)

Now you might be asking this question: "The SNES has 16 Megabytes of memory? I read it only had 128 Kilobytes of RAM on some site!" Well you read correctly, it only does have 128 Kilobytes (128K for short) of memory.

If they had put 16 Megabytes of RAM into the system, it probably would have been so expensive that no one would have bought it. The RAM in the SNES had to be fast for high performance, and faster RAM generally costs more to manufacture.

The main RAM in the SNES is "mapped" to banks 0x7E and 0x7F. It's often called WRAM, short for "Work RAM". It's general purpose RAM.

Whereas there are other types of RAM in the system, like VRAM (Video RAM) and CGRAM (color graphics RAM, which stores the colors used in palettes).

Again, note that the size of these two WRAM banks is 64K * 2 = 128K, as we talked about earlier. This means that the addresses of WRAM range from 0x7E0000 to 0x7FFFFF *.

(* side note: A dollar sign is another common notation used to denote that a number is in hexadecimal (base 16), just like the prefix "0x".

With the dollar sign notation, those same addresses could be expressed as $7E0000 to $7FFFFF. A common misconception is that the dollar sign means "address".

This is totally not true, so please don't fall into this trap. I prefer to use "0x" to avoid this problem, but sometimes $ is more appealing to look at.)

So you're probably wondering: "If only two out of 256 banks are occupied by RAM, what's in the rest of them?". This is a question that is difficult to answer 100% accurately without writing a bunch of stuff you're probably not interested in hearing at this stage of the tutorial.

For now we'll rather answer the related question: "Can the ROM be found in these banks?". The answer to that is yes, and we'll be examining how the ROM is "mapped into" the 65c816 banks.

Where is the ROM data?

Let me first say that "mapped into" is misleading. That is, the data from the ROM is *never* copied into the SNES' banks. The SNES has nowhere to store it! The only reasonably sized place it can store data into is in WRAM or VRAM.
So when we say "mapped" what we really mean is a gateway. The ROM data, which is generally much larger than can fit into WRAM, always stays on the cartridge*. The way it works is that the SNES CPU uses various addresses as gateways to the ROM data.

(* Side note: It is indeed possible for the SNES to copy a portion of the ROM to the WRAM in banks 0x7E and 0x7F, in fact it happens all the time.

Code can even be copied from the ROM to WRAM and run from WRAM, but that's a bit beyond what we're trying to figure out here.)

Intuitively, you might think that that first line of ROM data would show up in the first byte of bank 0x00. That is, address 0x000000 (sometimes written as $00:0000 with the colon used to separate the bank number from the offset inside that bank.

I will be using this notation to describe CPU addresses simply because it helps you note which bank we're looking at). It turns out that this is not true. Instead, we can find the first line at address $00:8000 to $00:800F.

Now remember that I said that the ROM data remains on the cartridge. Here's generally how things play out between the SNES and the Cartridge:

1. The SNES CPU is told to read some data from address $00:8000
2. The CPU sends a request to the cartridge
3. The cartridge looks at the address ($00:8000) and decides what data to send back. The cartridge has a low level controller that determines what data to send.
4. The cartridge sends back the first byte of the the ROM. (address 0x0 in the ROM)
5. The CPU now has the data, which is the "78" byte we saw before, and carries on its merry way.

So we know where to access the first byte in the ROM from the CPU. How do we find the rest of the ROM? This goes back to the concept of "LoROM." Older documents often referred to "LoROM" games as having 32 Kilobyte banks instead of 64 Kilobyte banks.

Now we saw earlier that all the banks on the SNES are 64 Kilobytes in size, right? Does a LoROM game shrink these banks? Absolutely not!

What those documents really meant was that the ROM was split up into 32K pieces and mapped to the second halves of banks. Here's a diagram to help explain:

```
Address in ROM file     Address from the CPU's perspective
$000000-$007FFF         $00:8000-$00:FFFF
$008000-$00FFFF         $01:8000-$01:FFFF
$010000-$017FFF         $02:8000-$02:FFFF
...
$0F0000-$0F7FFF         $1E:8000-$1E:FFFF
$0F8000-$0FFFFF         $1F:8000-$1F:FFFF
```

So now we know how the ROM looks from the SNES CPU's point of view. Notice that the ROM spans banks 0x00 to 0x1F, but only the upper halves of each of those banks. You may say "What's in the lower halves of those banks?" We'll find out later.

In the meantime we're going to re-examine something we took for granted earlier. Namely, that the place where the game starts executing, the very beginning of the code, is found at ROM address* $000000 (CPU address $00:8000).

(* Side note: If you need help converting between ROM addresses and CPU addresses, or if you're too cool to do math in your head, there's a program out there called Lunar Address which will do the conversions for you, if you have everything set up right.)

How did I know where it started? It wasn't just luck, because not all ROMs start executing at their first byte. The answer is that there is something called a "Vector table" stored in the ROM. A vector is another name for a pointer or address.

The CPU is hardwired when it's powered on to look for a starting point at the address $00:FFFC. There's more than one vector, and the whole vector table starts at address $00:FFE0.

The vector we're interested in is called the "Reset Vector", because that's where to start execution when the system resets or gets powered on. It is a 2 byte number that leads to another location in bank 0x00.

All the other vectors are 2 byte numbers as well.

Now you might be itching to go to location $00FFFC in your ROM to find that vector. But we can't forget what we just learned about how the ROM is mapped into the CPU address space. $00:FFFC in the CPU is $007FFC in the ROM.

For convenience I have printed out the 0x20 bytes starting at ROM address $007FE0 and ending at $007FFF here. The reset vector is highlighted in red.

```
FF FF FF FF 2C 82 FF FF 2C 82 C9 80 00 80 D8 82
FF FF FF FF 2C 82 2C 82 2C 82 2C 82 00 80 D8 82
```

So is the reset vector 0x0080? No, because the 65c816 processor is "Little Endian". This means that for going left to right, the left most numbers have lesser value. Humans have a tendency to read numbers in the opposite direction, with the larger, more significant places coming first.

This is not the place for a tutorial on endianness, so if you're confused, do a search on "little endian" and "big endian". If you don't feel like it, all you have to do for now is just "flip the bytes".

That is, to get the final hex number, just flip the order they appear in in the hex editor. So in this case, the red number is **80 00**, which combines into a 16-bit number to become **0x8000**.

The reset vector points to a location in bank $00, always (same goes for all the other vectors.) So the number we just looked up, **0x8000**, means we need to look 0x8000 bytes into bank 0x00. Thus, the very beginning of the code in the game will start at address $00:8000*. And the CPU address $00:8000 corresponds to ROM address $000000, as I was saying earlier.

(* Side note: Here's a hypothetical example to help reinforce the concept.

If the bytes for the reset vector read out as "23 A2" instead of "00 80", the reset vector would be 0xA223 instead of 0x8000, and we'd have to look at CPU address $00:A223, which is ROM address $002223.)

Zelda 3 Disassembly & Editing Sprites Behaviour (AI)

A) ZELDA 3 SOURCE CODE

http://math.arc-nova.org/temp/Banks_10_04_2012.7z

This is the most up-to-date version of my disassembly of the game's code. It is a massive amount of text to sort through, so don't be surprised if it takes a while to find what you're looking for. Of course, Bank00.asm is a good place to start!

B) EDITING SPRITES BEHAVIOR (AI)

Moving on... *DiscoPeach* requested information on how he might change the behaviour of the Lake of Ill Omen Monster (the one that gives you the quake medallion).

This will be our first test case.

I should stress that hacking the sprite AI requires a bit of in-depth knowledge of the sprite system itself and the sprite that is to be edited in particular. There are some consistent things that each sprite or a subgroup of sprites will do similarly, while others march to the beat of their own drum and use pretty proprietary methods to get things done.

Without further ado, we will see how to modify the monster so that it gives an empty bottle rather than the quake medallion.

```
$EDF49-$EDFDD JUMP LOCATION
{
          ; ILL OMEN MONSTER / QUAKE MEDALLION
          LDA $0D90, X;
          BPL BRANCH_ALPHA
          JSR $E37D; $EE37D IN ROM;
          RTS
BRANCH_ALPHA:
          .... ; and so on
```

That is the start of the monster's AI code. This routine also controls the AI of the quake medallion you see pop out of the water later. You wouldn't know this unless you had dicked around with the code and some memory a bit...

Anyways, It's clear to me that $0D90, X is an important memory address for this sprite. In fact, it's how the game differentiates the normal catfish looking monster from the quake medallion itself. The sprite number (ID) of this monster is 0xC0.

When the medallion pops out, there will be two sprites with the ID 0xC0 in memory. The difference will be that the quake medallion has $0D90, X set to 0x11. So we need to find out how the quake medallion is being generated.

```
$EE16C-$EE1A9 LOCAL
{
          LDA.b #$C0

          JSL $1DF65D ; $EF65D IN ROM;

          BMI BRANCH_ALPHA

          JSL $09AE64 ; $4AE64 IN ROM;

          PHX : TYX

          LDA.b #$18 : STA $0D50, X

          LDA.b #$30 : STA $0F80, X

          LDA.b #$11 : STA $0D90, X

          LDA #$20 ; play a sound effect

          JSL $0DBB7C ; $6BB7C IN ROM;

          LDA.b #$83 : STA $0E40, X

          LDA.b #$58 : STA $0E60, X
          AND.b #$0F : STA $0F50, X

          PLX : PHX
          PHY

          LDA #$1C

          JSL $00D4ED ; $54ED IN ROM;

          PLY
          PLX

BRANCH_ALPHA:

          RTS
}
```

From previous experience I know that the call to $1DF65D (via the **JSL** / Jump to Subroutine Long instruction) is the most widely used method of spawning a sprite. In fact, I can't think of any others off the top of my head.

You simply load the accumulator with the ID of the type of sprite you want to create, and then call the subroutine and it will spawn it. In this case, we see that it's spawning another sprite with ID 0xC0. This must be the quake medallion! Observe:

```
LDA.b #$11 : STA $0D90, X
```

This is what needs changing to make the game give you a different item. Changing the #$11 to #$10 or #$12 will give you some immediate effects you can see. However, the graphics for the sprite will only be correct once they reach Link's hands... They'll still look like the Quake medallion until he picks it up. This is annoying.

```
          LDA #$1C

          JSL $00D4ED ; $54ED IN ROM;
```

My instincts told me this was what was controlling the graphics of the sprite. So I fiddled with it and changed the #$1C to a #$1D. Strangely enough I got a bottle on the first try. But is it correct? The subroutine called after the LDA #$1C is intended to decompress graphics, so I just told it to load a different graphics set. I got lucky, so to speak. But we should always do things correctly...

After digging a little deeper, I figured out that the routine $0054ED has the purpose of decompressing graphics only to copy a single bit of data out of the graphics set. (A graphics set in this game consists of 64 8x8 pixel tiles). So the #$1C value helped determine which set to load, and what tiles to grab out of that set. The rest of the graphics set is then discarded. So this works out all fine and dandy. Change this to $1D and all will be well. We're not quite finished though....

The monster itself obviously behaves differently based on whether you've gotten the quake medallion or not. If you already have it, it just shoots a fireball or throws a bomb at you when you toss something into its circle of stones.

So it must be checking your inventory to see if you have the quake medallion. Naturally we don't want to be basing the decision of giving Link a bottle on whether or not he has the quake medallion!

This leads to a bit of a conundrum - there are four bottles in the game, so how do we tell them apart? For this purpose we should inspect the bottle vendor and see how he does it, because he will only sell you one. Some flag in memory must be set telling him not to sell it again, but what is it...? A bit of explanation on how the bottle guy does his stuff.

```
LDA $7EF3C9 : AND.b #$02
```

This code checks a specific flag that tells the game whether Link has been sold the bottle from the Vendor. If you check my SRAM guide, you'll see under location $3C9 (the $7E3 part at the beginning is assumed in that guide) that this flag is unknown. Well, it's known now. (It of course will appear the next time I upload my documents.)
So I did a little digging and found out that some of the bits in this memory location are not used anywhere in the game.

That's pretty convenient! Specifically, bits 2 and 6, which correspond to masks 0x04 and 0x40 when it comes to ASM.

So we can rewrite the monster's code to check for bit2 in $7EF3C9 rather than the presence of the quake medallion. An excerpt from the monster's code.

```
        LDY.b #$2A
        LDA $7EF349

        BEQ BRANCH_ZETA

        LDY.b #$2B

BRANCH_ZETA:

        STY $1CF0

        LDA.b #$01 : STA $1CF1

        JSL $05FA8E

        RTS

BRANCH_EPSILON:

        CMP.b #$50

        BNE BRANCH_THETA


        PHA

        LDA $7EF349

        BEQ BRANCH_IOTA
```

Okay... let's do a basic overview of this code. Subroutine $05FA8E is called whenever the game wants to display a dialogue message. naturally, we know that the monster has two things it says.

The first time you wake it up it gives you something and tells you to piss off and let it go back to sleep. Once you have the quake medallion, it will just throw something harmful at you and dive back in the lake.

From the code, we can see that it invokes message 0x012A if you don't have the quake medallion and 0x012B if you do. Follow along with Hyrule Magic and convert these numbers to decimal. They should be the expected messages from the monster if you go look them up.

In any case, it's using $7EF349 to determine whether you have the quake medallion or not. If you check my SRAM guide you can verify that this is what the whole game uses to verify if you have the quake medallion.

This brings up an interesting point. If you somehow fail to pick up the quake medallion and leave the screen, when you come back the monster will try to give you the quake medallion again. Keep that in mind...

Herein lies a problem, you see the Quake medallion sprite itself is what sets the variable saying that you have it. Let's say we change the condition of the dialogue and the action of the monster to this:

```
 LDA $7EF3C9 : AND.b #$04 ; using an independent flag now rather than the quake medallion
indicator!
```

Both instances should be replaced and doing that is another matter entirely as it won't fit in the original space reserved in the ROM, so you have to hook into the ROM at those places and link to this code at another location. If you need an explanation of how to do that, ASK!

The problem with this is that in sprites like the Bottle Vendor guy in the village, the transaction is discrete, he controls the whole act of giving you the bottle. Here, we could just walk away from the bottle on the ground (provided we were not in the line of fire in the first place).

If the monster will be reprogrammed to set this flag after it throws it out, then if you leave the screen without picking up the bottle, it will be gone when you come back, and the monster won't ever give it to you again!

Now, I know you're thinking, most players would pick up whatever item was given to them. They wouldn't just walk away. But think if this was a critical item that you needed to beat the game. That would certainly not be very professional programming.

Here's a possible solution: the main AI pointer that controls the monster is the same as the item that gets thrown out (they're both 0xC0, remember?). We could change some code in the part that handles the sprite being an item.

When Link goes to touch it, we should determine whether the sprite is a bottle, and also whether Link is on the overworld, and also check which area we're in. If it matches the area with the Lake of Ill Omen (area 0x7F), we set this flag like so:

```
LDA $7EF3C9 : ORA.b #$04 : STA $7EF3C9
```

Now we could leave the screen and the monster will still give us the bottle when we come back a second or third or fourth time, etc.

This is a bit of extra work to do for something so minor, but if someone plays your hack and they complain that it's buggy with unexpected things like this, you have only yourself to blame.

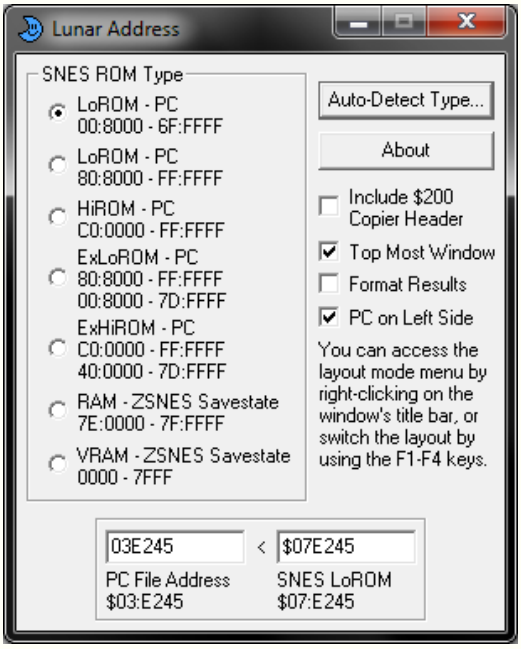
2) Euclid's Teachings

~ Euclid™ Lesson 1 ~

HEX, ASM and CPU Address Basics

A) RAM AND LUNAR ADDRESS

One of the quickest ways to convert your addresses is to use Lunar Address. If your addresses start with 7E or 7F, you can't convert them to hex. 7E refers to the current memory (RAM) so it wouldn't be in the ROM hex.



So if you want to convert any codes pretty much drop the first 6 hex digits into the right box and your address with show up on the left (the last 2 hex digits is the byte to write to that address with).

For example

008000FF is just writing FF into 008000 in snes cpu address (which translates to hex offset 0)
+ 0x200 if you have a header

The "Address Calculator" button (in the overworld editor of Hyrule Magic) which just tells you the x and y positions... pretty much the technical side of it.

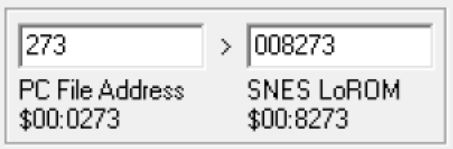
The easiest way of describing **RAM** and **ROM** is:

"RAM is memory, starts out to be 00 when you start the game or when you reset the game"

"ROM is what's in the .smc file"

Then after knowing that, you need to learn about snes addresses.

For example, open Lunar Address and type **273** in the bottom left box (PC File Address). Notice how it now says $00:**8273** right below the bottom right box.



00 is the bank
8273 is an address in the bank

B) SNES MEMORY

Let me now explain this bank stuff in details.

In the following document, you should find two sets of different modes to the snes address model. The top one is called **lorom** while bottom is **hirom**. Zelda 3 uses lorom. That is in banks $00-3F. There's no difference if you use 3F4000 to access 004000. That is pretty much self-explanatory.

```
                    +=-=-=-=-=-=-=-=-=-=-=-=+
                    | SNES Memory Mapping |
                    |    By: ]SiMKiN[     |
                    |         v3.5        |
                    +=-=-=-=-=-=-=-=-=-=-=-=+


• FastROM's can execute at 3.58Mhz
• SlowROM's can only execute 2.68Mhz

  • The SNES lets you access ROM through bank $00 onwards and bank
    $80 onwards such that locations $00:8000 and $80:8000 are congruent,
    (they access the same locations.)
  • When accessing bank $00 onwards the 65816 runs at 2.68Mhz. However,
    when accessing bank $80 onwards the 65816 can run at 2.68Mhz or
    3.58Mhz depending on how you set bit 0 of $420D.

  • This Document Contains Information Regarding ROM's up to 48mbit.
    If you have any information not contained in this document please
    send E-Mail to 'simkin@innocent.com'

+=================================================================+
| Mode 20: LoROM Memory Model (32k Banks) - 24Mbit Max            |
| ------------------------------------------------                |
|  • $00-$5F : $8000-$FFFF                                        |
|              Mirrored to $80-DF                                 |
|    -------                                                      |
|    $60 * 1/2 banks = 24Mbit                                     |
+=========+============+=====================================+=========+
| Bank    | Offset     | Definition                          | Shadow  |
+=========+============+=====================================+=========+
| $00-$2F | $0000-$1FFF | LowRAM, shadowed from $7E          |   $7E   |
|         | $2000-$2FFF | PPU1, APU                          | $00-$3F |
|         | $3000-$3FFF | SFX, DSP, etc.                     | $00-$3F |
|         | $4000-$41FF | Controller                         | $00-$3F |
|         | $4200-$5FFF | PPU2, DMA, etc.                    | $00-$3F |
|         | $6000-$7FFF | • RESERVED                         | $00-$3F |
```

	$8000-$FFFF	(Mode 20 ROM)	-------
$30-$3F	$0000-$1FFF	LowRAM, shadowed from $7E	$7E
	$2000-$2FFF	PPU1, APU	$00-$3F
	$3000-$3FFF	SFX, DSP, etc.	$00-$3F
	$4000-$41FF	Controller	$00-$3F
	$4200-$5FFF	PPU2, DMA, etc.	$00-$3F
	$6000-$7FFF	• RESERVED	-------
	$8000-$FFFF	(Mode 20 ROM)	$80-$BF
$40-$5F	$0000-$7FFF	• RESERVED	-------
	$8000-$FFFF	(Mode 20 ROM)	$C0-$DF
$60-$6F	$0000-$7FFF	• RESERVED	-------
$70-$77	$0000-$7FFF	(Mode 20 SRAM) 256KBytes	-------
	$8000-$FFFF	• RESERVED	-------
$78-$7D	$0000-$FFFF	• RESERVED	-------
$7E	$0000-$1FFF	LowRAM	$00-$3F
	$2000-$7FFF	HighRAM	-------
	$8000-$FFFF	Expanded RAM	-------
$7F	$0000-$FFFF	Expanded RAM	-------
$80-$DF	$0000-$FFFF	Mirror of $00-$5f	$00-$5F
$E0-$FF	$0000-$FFFF	• RESERVED	-------

Mode 21: HiROM Memory Model (64k Banks) - 48Mbit Max

- $C0-$FF : $0000-$FFFF
  High Parts ONLY '($8000-$FFFF)' are Shadowed to $00-3F
- $40-$5F : $0000-$FFFF

  $60 banks = 48Mbit

Bank	Offset	Definition	Shadow
$00-$2f	$0000-$1FFF	LowRAM, shadowed from $7E	$7E
	$2000-$2FFF	PPU1, APU	$00-$3F
	$3000-$3FFF	SFX, DSP, etc.	$00-$3F
	$4000-$41FF	Controller	$00-$3F
	$4200-$5FFF	PPU2, DMA, etc.	$00-$3F
	$6000-$7FFF	• RESERVED	$00-$3F
	$8000-$FFFF	(Mode 21 ROM) from $C0-$EF	$C0-$EF
$30-$3F	$0000-$1FFF	LowRAM, shadowed from $7E	$7E
	$2000-$2FFF	PPU1, APU	$00-$3F
	$3000-$3FFF	SFX, DSP, etc.	$00-$3F
	$4000-$41FF	Controller	$00-$3F
	$4200-$5FFF	PPU2, DMA, etc.	$00-$3F
	$6000-$7FFF	(Mode 21 SRAM) 128KBytes	-------
	$8000-$FFFF	(Mode 21 ROM) from $F0-$FF	$F0-$FF
$40-$5F	$0000-$FFFF	(Mode 21 ROM)	-------
$60-$6F	$0000-$7FFF	• RESERVED	-------
$70-$77	$0000-$7FFF	(Mode 20 SRAM) 256KBytes	-------
	$8000-$FFFF	• RESERVED	-------
$78-$7D	$0000-$FFFF	• RESERVED	-------
$7E	$0000-$1FFF	LowRAM	$00-$3F
	$2000-$7FFF	HighRAM	-------

```
|         | $8000-$FFFF | Expanded RAM                          | ------- |
+---------+-------------+---------------------------------------+---------+
|   $7F   | $0000-$FFFF | Expanded RAM                          | ------- |
+---------+-------------+---------------------------------------+---------+
| $80-$BF | $0000-$FFFF | Mirror of $00-$3f                     | $00-$3F |
+---------+-------------+---------------------------------------+---------+
| $C0-$FF | $0000-$FFFF | (Mode 21 ROM)                         | ------- |
+=========+=============+=======================================+=========+

• ROM: The SNES ROM Image
• RAM: The SNES Work Memory (WRAM)
       LowRAM, HighRAM, & Expanded RAM
       All together = 128 Kilo-Bytes
• SRAM: Save RAM (Extra RAM added by Cart)
        The SNES only utilizes 256 Kilo-bits.
        However 256 Kilo-Bytes are provided for MODE 20 SRAM,
        and 128 Kilo-Bytes are provided for MODE 21 SRAM.

• APU: Audio Processing Unit
       SPC700, Inside which has a DSP
• PPU: Picture Processing Unit
       PPU1: 5c77-01
       PPU2: 5c78-03

• SFX: Super FX Cart Chip, by Nintendo
• DSP: Digital Signal Processing Cart Chip
       a.k.a. 'NEC mUPD77C25'

• Shadow: "Congruent Bank".  Same meaning as Mirror.

          ____________________________________________
       .o(_Thanks to: zsKnight, Lord Esnes, Y0SHi, and MintaBoo_)o.
```

Let me go through all those banks 00 – 3F. You should see LowRAM shadowed from $7E.

Q: *That's saying the bank on the left says 00 to 2F?*

A: *00/0000 is the same address as 7E/0000*

The second box has the same thing written under it… xx/0000 to access that address, xx can be 00-3F or 7E. For take address 7E008A for example. If you type write in 00008A it'll have the same effect or even 3F008A.

Addresses banks 00-3F (2000 - 5FFF) those are used to do special stuff, I'll get to them later
the ones with PPU1, APU, SFX, DMA etc. What I wanted to point out is the (mode 20 ROM) in $8000-FFFF. That is the actual code of the game, like what you see in the .smc file.

Usually the calculation goes like this:

```
bank  hex     offset
00   0000   - 7FFF
01   8000   - FFFF
02   10000  - 17FFF etc..
```

So say $01/C710 in snes address is really just 0xC710 in hex offset (+0x200 for header) and that's 200 in hex… but if you use Lunar Address, it can calculate it for you without any fuss. Just type in something like 20/8000.. you'll notice it's 0x100000 hex offset (0x100200 when you tick the +200 to header box).

That's that, banks $7E and $7F you'll noticed they're all RAM and also
$7E 0000-1FFF is mirrored to banks 00-3F.

Ok that's that for the snes technical stuff, ready for a quick fix in ASM?

C) QUICK ASM FIX

Let me take a piece of code...

```
$00/8BD3 B9 7A 93 LDA $937A,y[$00:937D] A:0220 X:0000 Y:0003 P:envMXdIzC
$00/8BD6 85 00 STA $00 [$00:0000] A:0280 X:0000 Y:0003 P:eNvMXdIzC
```

Note that it's a piece a code created using Geiger's Snes9x Debugger trace logger function.

On the left is pretty much the snes address.

For example take a look at "Bank 00" address 8BD3. Notice how it says B9 7A 93. If you go open up Zelda 3 in a hex editor and go to that address $00/8BD3 (is 0xDD3 in hex), you'll notice the same values written here: B9 7A 93.

In Translhextion:	In HxD:
ctrl + g	ctrl + g
or Offset -> Jump to	or Search -> Go to...
type "xdd3"	type "DD3"

If you notice after B9 7A 93 is 85 00 which is the next line of code, that's pretty much how code is done in the rom.

Another piece of code:

```
$02/D9A3 BD 13 C8 LDA $C813,x[$02:C8FB] A:03A0 X:00E8 Y:03A0 P:envmxdizc
$02/D9A6 85 A0 STA $A0 [$00:00A0] A:003F X:00E8 Y:03A0 P:envmxdizc
```

Type 02D9A3 in Lunar Address and you'll get 0x15BA3 and in hex it shows up as BD 13 C8.
Every address in the rom is described by one of these snes addresses.

D) SNES REGISTERS, GEIGER'S SNES9X, BREAKPOINTS AND OPCODES

Now to more technical stuff about the snes apart from all the addresses in the RAM. There's also a few special memory registers to help out (that's A:03A0 X:00E8 Y:03A0 that stuff there).

In snes there's 3 commonly used registers, called A, X and Y, and they can take in a 2 byte value

So in that case, it's showing :
the A has 0x03A0 in it
the X has 0x00E8 in it
the Y has 0x3A0 in it

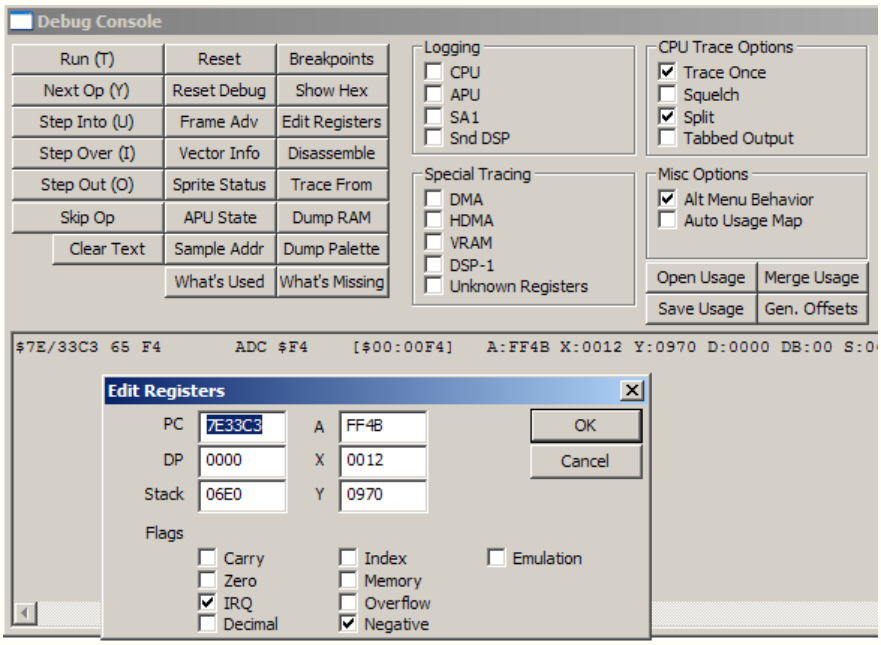
The Debugger itself: Geiger's Snes9x Debugger

You might need those 3 files msvcp71.dll, msvcr71.dll and mfc71.dll in order to run the program correctly, you can find them on Zophar http://www.zophar.net/snes/geiger-s-snes9x-debugger.html , just stick them in the same folder as the emulator.

Oh and I have to warn you that the emulator doesn't like headers and will get rid of them.
Anyway make a copy of your Zelda 3 rom and just open it up in the emulator.

After that debug console pops up, go check Trace Once and Squelch on the right. Let me explain all those functions.

First, the buttons.



Run is self-explanatory
Reset is reset, self-explanatory
Next Op shows the next line of code to run
Show Hex, if you click on it, you can select either **ROM**/**RAM**/**ARAM**/**VRAM**.
- if you select **ROM** it pretty much shows you what's in the rom
  notice the address on the top, it's 808000 BFFFFF
  if you see that document earlier, bank $80-BF is mirrored to $00-3F
- if you go click on **RAM** it's pretty much banks $7E and $7F
- the **ARAM** and the **VRAM** you don't have to worry about much yet
  since they're the the sound RAM and video RAM.

Step Into will step into the next instruction
if you press that the instruction will move onto the next one
think when you're playing the game this button gets pressed 99999999 times a second
Step Over/Step Out is pretty much the same as step into
Dump RAM copies banks 7E - 7F to a file, I haven't used it much.
And the rest of the buttons I don't use much with the exception of **Breakpoints**.

The **Logging** means to write down a log file of each of the step into instructions taken place
so if you check it, then let the game run for a while, uncheck it, you would have logged the instructions used to run at that particular time frame.

And since you know how I mentioned running the game means executing lots of instructions
the trace once option you checked pretty much tells it to not write down the same instruction again. If you have that off, your log file will climb into the mbs when you let the game run.

The APU/SA1/Snd DSP I haven't used much, the special tracing shouldn't be much use, and I don't use those things under misc options.

Now to that piece of code I mentioned earlier

```
$02/D9A3 BD 13 C8 LDA $C813,x[$02:C8FB] A:03A0 X:00E8 Y:03A0 P:envmxdizc
$02/D9A6 85 A0 STA $A0 [$00:00A0] A:003F X:00E8 Y:03A0 P:envmxdizc
```

Say you want to know when that line of code is run or executed that's when you'll use a breakpoint. So click on **Breakpoints**.

Type 02D9A3. There's three options there.

Exec means when this memory address gets executed.
Read means when this memory address gets read.
Write means when this memory address gets written (but not possible in this case since this is in the rom).

So check **Exec**, click **ok**, then let the game run (just let it play the intro).

Press **run** on the debug console, the game should start. Just let it run the intro. The game suddenly stops after it zooms in from the map.

That's telling you, that address is getting run at that time. That's pretty much the basic intro debugging as that address is just some code in the game.

http://www.obelisk.demon.co.uk/6502/

It's for the NES but SNES is just the same thing but with banks. The registers, instructions, and addressing are the important pages. Once you understand those, you can pretty much start to ASM hack. It pretty much explains all those LDX stuff and how they work.

E) UNUSED RAM

There's unused RAM everywhere which you can use for your counter.
$1Dxx-$1Fxx
Those should be unused.

~ Euclid™ Lesson 2 ~

Opcodes, Processor Status and Instructions

All those snes instructions are somewhat confusing and that's the next step, understanding those instructions, it's not that hard really. This step is easier if you're a programmer.

A quick explanation for them:
http://www.obelisk.demon.co.uk/6502/instructions.html
http://www.obelisk.demon.co.uk/6502/reference.html

Looks slightly more understandable. You'll probably be looking at ADC.
That's just adding values. The list below that under Addressing Mode, that's the ways which this instruction can be executed.

Don't worry too much about the process status there with, the Carry Flags and stuff… the opcode there is basically the byte to tell the cpu that the instruction is a ADC.

You'll notice that each value 00-FF represents a different opcode. The bytes next to it is telling it how much bytes this instruction will use like how absolute is 6D.

A) IMMEDIATE, ZERO PAGE & ABSOLUTE

Understand the differences between immediate, zero page, absolute and stuff. With ADC it's basically, the A register added with something… that something is determined by which addressing mode it is in.

Immediate is when you give it a 1 byte value. For example when A has 0x00 in it and you do
ADC #$23, A would have 0x23 in it. The # we write usually say this is a hex value not a memory address. So basically adding 23 to the A register can be done with 69 23.

$23 usually means memory address $23 (variable is also a name for it) so that's immediate out of the way.

Zero page is known as the direct page, basically a commonly accessed bank of memory in most snes games. Zero page is 7E0000 to 7E00FF.

Ao if you see ADC $00, pretty much means adding what's in 7E0000 to the A register
Zero Page,x is basically the same thing, it's just the address + whatever is in the X register.

ADC $00,x if X has 0x10 in it, $7E0010 will be accessed. This is useful when you have to step through stuff in a loop.

Absolute is when you give it an 2 byte address. Say ADC $1000 is saying add to memory address $7E1000.

Absolute,x Absolute,y works the same as before.

(Indirect,X) is possibly one of the most annoying ones. When you see ($00,x)…
…it pretty much says look inside what's at $00 and $01. Say they have $20 and $80 respectively
and X in this case X is 0, then the address accessed is $8020. In that case it'd be 7E8020.

(Indirect,x) and (Indirect,y) in those cases X or Y is added before looking into it (so if X is 1 in the above case it'll be $01 and $02 instead).

Whereas (Indirect),x or (Indirect),y the address is added after, so if x or y is 1 in the above case, it'll be 7E8021.

I think that pretty much covers some of the addressing modes

B) OPCODE INSTRUCTIONS

Let's go through those instructions.

ADC something is obviously adding something to the A register (also called the accumulator).

AND logical AND bitwise operations.
You have to know that 0x23 in hex is 0010 0011 in bits

with AND
0 and 0 = 0
0 and 1 = 0
1 and 0 = 0
1 and 1 = 1

Pretty much says if you AND 0x23 with say 0x89.
The same thing applies here. Yeah and OR works kind of opposite.

ASL x2 per shift

Scrolling down the page... I'll skip the branches for now.

BIT is pretty much an AND operation

stuff like **BRK** you won't even have to deal with

DEC is just decrement the memory address, in snes you can decrement X, Y or A as well

EOR is XOR

INC is increment

LDA/LDX/LDY is putting a value onto the A,X,Y register

LSR is divide by 2

NOP is telling it to do nothing

ORA is OR

ROL is pretty much the same as ASL but the last bit gets swapped with something

SBC is subtract

STA/STX/STY is storing values from the register to a specified memory address

Txx xx can be
XY - transfer value from x to y
YX - y to x
AY - etc
AX
XA
YA

C) PROCESSOR STATUS

Ok now I got to teach you about the processor status. Besides the three registers, there's a few special memory which special stuff gets stored. One is the stack pointer, one is the processor status... there's a few more which I forgot atm.

If you have noticed the outputs from that emulator:

```
$0C/F796 EA NOP A:B723 X:012E Y:0001 P:envMxdizc
```

The **P:envMxdizc** is the processor status. They're basically bits. 000100000 in the above case.

The M is on because it's in caps.

P:envMxdizc <-- this thing

They're just a special memory with 1 bit associated with each letter

e - 0
n - 0
v - 0
M - 1
x - 0
d - 0
i - 0
z - 0
c - 0

If its capital it means it's a 1. I forgot what some of them mean, but the important ones n, M, x, z, c. You may have noticed them here http://www.obelisk.demon.co.uk/6502/reference.html.
Say under the Processor status they have Czidbvn and they even got their names next to them, it becomes quite self-explanatory. The M mean is not listed there because M is exclusive to snes.

The Carry flag, that's used in a lot of times, you can set it or clear it with **SEC** or **CLC**
with instructions like **ROL**/**ROR**.

The last bit or the bit which is getting pushed off is taken to the carry flag.

Helps visualise:

CLC
ROL $30

First clear the carry flag, then rotate. Say $30 has **0010 0011**. What would happen after the instruction is the first 0 replaces the carry flag (which is 0), then whatever which was on the carry flag gets put onto the end **0100 0110**. If the instruction before that is **SEC** (set carry), then it'll instead be: **0100 0111**.

D) NEW M SNES PROCESSOR STATUS

You noticed how the accumulator is 16 bits.

When M is 1 (M) it'll work like the old nes one, stuff like LDA #$20 will work.
When m is 0 (m) it'll work 16 bits.

Instructions like LDA will need to be 3 bytes.
So stuff like LDA #$1234 is possible.

Well pretty much everything is possible with the accumulator in 16 bits when m is down, when X is down everything with the X/Y registers are working in 16 bits.

If you need an example of it...

```
$00/8661 BD 96 93 LDA $9396,x[$00:9396] A:B7A8 X:0000 Y:00FC P:envmxdiZc
$00/8664 8D CC 0A STA $0ACC [$00:0ACC] A:8080 X:0000 Y:00FC P:eNvmxdizc
```

see how in the next instruction, 16 bits gets loaded in the A register

```
$00/8697 18 CLC A:9000 X:0000 Y:00FC P:eNvmXdizc
$00/8698 69 80 01 ADC #$0180 A:9000 X:0000 Y:00FC P:eNvmXdizc
```

E) BRANCHES, JUMPS AND COMPARE FUNCTIONS

Ok going to get into branches.
Branches are pretty much **if** statements.

I should introduce you to the compare functions first.
The compare functions (**CMP**, **CPX**, **CPY**) are mostly used right in front of the branches to do an **if** statement. You'll need to know when to do it and where to go to.

Ok so about **CMP**, **CPX**, **CPY**. In each branch statement only the where to go to state those 3 instructions there helps determine when to do it.

CMP basically compares whatever with what's in the A.
The same applies to the other 2 (to X and Y). What that really does is that it puts the compare result to the carry and zero flags.

Zero flag gets turned on if the 2 things are equal.
Carry gets turned on if what's the in the A is >= M

As stated there the branch instruction usually looks at those 2 flag.

For example:
CMP #$04
BEQ blah

First it compares with 4, then if it is the branch will activate.

BMI looks at the negative flag.
BCS/**BCC** looks at the carry flag (good for >= < compares).
BEQ/**BNE** looks at the zero flag (== and !=).
There's a few more branches like **BVC** but I hardly see them.

CMP #$04 compares compares the value **04** with the accumulator
The # means it's a hex value or immediate address mode

Now I've gotten the condition of the **if** statement out of the way, now it's the where to go to.

BNE $10 means it's going to move **0x10** away from the next instruction.

So.. for example this:
D0 03 A9 32 60 A9 20 60

Translates to this:
BNE $03
LDA #$32
RTS
LDA #$20
RTS

The **$03** in the branch will end up at the second **LDA #$20** (it'll skip over three, **A9, 32, 60**, onto the second **A9**).

D0 03 A9 32 60

D0 03 is the **BNE $03** instruction.
A9 32 60 is **LDA #$32 RTS** instructions.

The counter for **$03** addresses starts from the A9. D0 is the opcode.

The **JSR/JMP** instructions and the **RTS**.

JSR is jumping to a subroutine.
RTS is returning from the subroutine (what it really does is that it pushes the positions of where it's executing on the stack).

Say for example:
$00/8444 - **JSR** $8900

That's saying jump to subroutine $00/8900. So 44 and 84 gets pushed onto the stack.
When it sees the **RTS** in the $00/8900 routine, it'll pop these addresses out and returns back to $00/8444 to continue executing.

JMP is the same thing but no pushing, so no going back if you use a **JMP**

That's pretty much all the opcodes... all the NES ones anyway.

F) SNES OPCODES

http://www.zophar.net/documents/65816/65816-info.html

This will cover the snes ones as well

If you scroll down a bit you'll see ***"2.00 New 65816 Instructions"***.
Hardest ones to get is the **MVN/MVP** which is rarely used.

STZ is basically writing 0 instead of doing LDA #$00 STA somewhere.

BRA is branch always.
I think **BRL** is a 16 bit version of the **BRA** instruction.

Where should I start with the snes ones?!

Scroll down and look for that ***"6.10 New 65816 Specific Addressing Modes"***

They have a quite decent description of what they do there.

G) CODE EXAMPLE: DEATH MOUNTAIN OVERLAY

Another piece of code...

```
$02/AFC6 A5 8A LDA $8A [$00:008A] A:0022 X:0095 Y:0390 P:envmxdizc
$02/AFC8 C9 03 00 CMP #$0003 A:0022 X:0095 Y:0390 P:envmxdizc
$02/AFCB EA NOP A:0022 X:0095 Y:0390 P:envmxdizC
$02/AFCC EA NOP A:0022 X:0095 Y:0390 P:envmxdizC
$02/AFCD C9 05 00 CMP #$0005 A:0022 X:0095 Y:0390 P:envmxdizC
```

You should also know by now what $8A in Zelda 3 is.

Remember 7E008A = current map.

Here's the bit after that bunch of code

```
$02/AFD0 F0 39 BEQ $39 [$B00B] A:0022 X:0095 Y:0390 P:envmxdizC
$02/AFD2 C9 07 00 CMP #$0007 A:0022 X:0095 Y:0390 P:envmxdizC
$02/AFD5 F0 34 BEQ $34 [$B00B] A:0022 X:0095 Y:0390 P:envmxdizC
```

Ok, first it loads the area and it's 0x22 I'm at when i did this.
Then it compares with 0003 0005 0007 (Areas 03 05 07).

Does that ring a bell? Using the overlay, those are the death mountain screens
and yes, this is the middle of the overlay code.

That's pretty much how code is done.

You might've noticed the NOP right after the **CMP #$0003**.
That's to delete the overlay for Area 03.

It was like this before...
CMP #$0003
BEQ somewhere
CMP #$0005
BEW Somewhere
...

So what I did was replaced the 2 bytes it took to write **BEQ** somewhere with **NOP** (do nothing).
So in a sense I've just broken the **if** statement.

I will go through each line in that snippet explaining what's doing what...that way it would be easier for you to understand all the opcodes and accumulators doing their stuff

So...

```
$02/AFC6 A5 8A LDA $8A [$00:008A] A:0022 X:0095 Y:0390 P:envmxdizc
$02/AFC8 C9 03 00 CMP #$0003 A:0022 X:0095 Y:0390 P:envmxdizc
$02/AFCB EA NOP A:0022 X:0095 Y:0390 P:envmxdizC
$02/AFCC EA NOP A:0022 X:0095 Y:0390 P:envmxdizC
$02/AFCD C9 05 00 CMP #$0005 A:0022 X:0095 Y:0390 P:envmxdizC
```

Ok first line. It reads **A5 8A**

A5 is the opcode **LDA** in the direct page.
That opcode takes 2 bytes, so the cpu reads in another byte, which is **8A**.
so it becomes **LDA #8A**.

So then it looks in the memory address 7E/008A (or 00/008A, same thing),
and finds the value, 0x22 in this case.

Puts that onto the bottom 8 bits of the accumulator, then it looks into 7E/008B (because m is 0)
that's 00, so it puts 00 into the top 8 bits of the accumulator.

Next instruction, so it looks at the next byte **C9**.
C9 is a **CMP**, since m is down it has to read 3 bytes.

03 00 gets read, so the value 0003 gets compared with the accumulator.
Not equal so keep z down, smaller so c goes up.

Next instruction, reads **EA**. that's do nothing, so it reads again, **EA** again, do nothing

Next instruction, **C9**. Does the same thing again, this time it reads **CMP #$0005**, and since 0x0005 is smaller than 0x0022 c goes up, not equal so z is down.

Then the next instruction... etc..

~ Euclid™ Lesson 3 ~

HUD Assembly

A) TILE FORMAT - HVPP PBBB

First you need to learn tile layout or the data for each tile.
Each tile is 2 bytes - 16 bits.

Their layout: **HVpp pbbb tttt tttt** (16 bits there starting from the right)

t is the tile number.

Basically, if you say open up the ALttP gfx file in yychr you may have noticed when you mouse over the gfx tiles, it'll say "bank $xx" somewhere. The xx basically refers to this.

The gfx is already loaded to the PPU already. All you got is this value to point it.
Also the 3 b bits to determine which block of gfx to use out of about 7 of them.

p is the palette, one out of 7 already defined palettes.

H and **V** are horizontal and vertical mirroring of the gfx tile.

B) YY-CHR AND THE TILESET BLOCKS

First boot up yy-chr, load ALttP gfx, switch it to 2gb and find those menu items, load a palette file if you want. Let's take... something which is shown on the hud.

Top left you should see bug catching net.
Ok let's see, the numbers are good... they're 90 to 99.

Boot up ALttP.. according to MoN's documents you should find out that the data which is written straight to the PPU is at about 7EC700.

So in your zsnes (or snes9x) cheat box type in: 7EC70090 and see the change on your screen.

If you want say the first tile of the bug catching net word you would write 00.

That's the low 8 bytes - it's on that format thing. You may notice the 0 has a different palette than your other numbers so to change that we have to look in the high byte, 7EC701 with no mirroring.

So say we want palette 0 just write 00 to 7EC70100. The palette is black, try something else say palette 5 what would you write in there.

Hip pbbb 5 is 101
0010 1000 = 0x28

So type in 7EC70128

Lets try mirroring 0 (is probably not the best letter to mirror)
say... 7 instead! Would be good enough, so go ahead change it to a 7.

7EC70097

Then now flipping the first 2 bits on in the second byte:

7EC701E0

Notice the change in the 7?

All the game does is pick out stuff from there and draw it onto the screen.

That's one tile you're modifying, imagine the whole screen is made of these tiles!

So in the actual rom it's all stored in $6FE77 - $6FFC0.
Changing that is ok when you want it to always appear.

For stuff which changes, you need code.

C) HUD DISPLAY AND DISTANCE

First you need to know the 7EC700 RAM area only goes up to a specific spot.

Should go up to only 164 8x8 blocks right.

For example in my "Zelda 3 Tower of The Triforce" romhack, I just made it so it goes from 7EC500 all the way to like 7ECC00 or so.

Q: Wouldn't that clash with existing ram data?

A: I didn't know what the existing RAM data was used for, but I did it anyway and it seemed to work fine at first. My first problem was the monologue box, when it pops up on the bottom it flashes like crazy because it's writing both my rupee count and the box on the same HUD.

That sucked, I managed to like make sure my rupee counter doesn't get updated when the box is up. Apparently that area of the RAM is used for scrolling the secret wall.

Creates a rather nasty effect, but it works fine afterwards and so far, apart from these 2 bugs, I see no problems with it. In the end if you want to expand it I can tell you how to fix the monologue problem.

D) RUPEE DISPLAY

Anyway let me start doing some quick explanation on the hud code.

First do a trace of the ALttP rom anywhere.

Just get into the rom, have Link staying somewhere, trace, run for a while, stop trace.

Ok say I want to find out where the money gets drawn.

Go search for some **LDA** on 7EF362

```
$0D/FC4C 8F C6 C7 7E STA $7EC7C6[$7E:C7C6] A:3C5E X:0080 Y:0012 P:envmxdizc
$0D/FC50 BD F5 FD LDA $FDF5,x[$0D:FE75] A:3C5E X:0080 Y:0012 P:envmxdizc
$0D/FC53 8F 06 C8 7E STA $7EC806[$7E:C806] A:3C5E X:0080 Y:0012 P:envmxdizc
$0D/FC57 AF 62 F3 7E LDA $7EF362[$7E:F362] A:3C5E X:0080 Y:0012 P:envmxdizc
$0D/FC5B 20 F7 F0 JSR $F0F7 [$0D:F0F7] A:03E7 X:0080 Y:0012 P:envmxdizc
```

Yep that looks right now you're in the middle of the code.
Ok let me explain the **JSR** $F0F7.

It's a routine which takes a hex number (the number of rupees in this case) and writes the tile number which is needed to draw that number in some memory.
Basically a hex -> decimal converter.

So scroll down a bit.
Upon return from that routine, you'll see a **LDA** $03 **AND #**$00FF **ORA #**$2400...
...then a **STA** $7EC750, then the same thing repeats for $04 and $05.

$03 in this case has the 100th digit
$04 has the 10th digit
$05 has the last digit

They just write it to $7EC750/52/54 which is the 3 tiles for the money.
They **ORA #**$24 is to make it use palette 6.

So basically that's that. If you EA those code there, the money won't appear.

E) ARROWS AND BOMBS DISPLAY

You may notice the stuff afterwards.

LDA $7EF343 that address ring any bells? That's the bomb counter.

After that you will see the same routine JSR $F0F7.
The hex -> decimal routine gets called again and numbers are written into 7EC758 7EC75A.

Then it loads 7EF377, the same thing happens.
7EF377 is the arrow counter.

F) DUNGEON FLOOR DISPLAY

Scroll down a bit more.

$7EF36F

Keys: $36F. Number of Keys you have in a dungeon.
You can earn keys on the Overworld but they don't do anything.
If you're in a non-keyed dungeon it will generally read FF.

Scroll down a bit more, you've got **RTS** and some more code afterwards:

```
$0A/FD92 A9 7F 00 LDA #$007F A:0000 X:00FF Y:00FE P:envmxdiZC
$0A/FD95 8F F2 C7 7E STA $7EC7F2[$7E:C7F2] A:007F X:00FF Y:00FE P:envmxdizC
$0A/FD99 8F 32 C8 7E STA $7EC832[$7E:C832] A:007F X:00FF Y:00FE P:envmxdizC
$0A/FD9D 8F F4 C7 7E STA $7EC7F4[$7E:C7F4] A:007F X:00FF Y:00FE P:envmxdizC
$0A/FDA1 8F 34 C8 7E STA $7EC834[$7E:C834] A:007F X:00FF Y:00FE P:envmxdizC
```

Notice that bunch of **STA**s which are related to that $7ECxxx area?
Well if you did the trace in the Overworld that wouldn't show up because that's to clear the area which it writes "1F" etc.

Anyway scrolling down ends the code.

G) MAGIC BAR DISPLAY

Scroll up this time from the rupees.

The magic bar uses a somewhat complicated method to draw itself, I rewrote it because I have a more convenient method and also mine is horizontal!

Each row in the screen is 0x40 bytes long.
That's actually one vertical row of tiles it's writing to.

You may notice 4 **STA**s:

746 786 7C6 806

You'll see it writes to 7EC704 06 08. Then a bit above that you see a $7EF37B. If you look in MoN's documents again: 37B is the half magic trigger. That's basically drawing that.

H) HEARTS AND ITEM DISPLAY

Up a bit more you'll see reads from 7EF36C/6D. Not entirely sure what it is but my guess is the hearts. That's about it for the code.

Scroll up a bit more you won't see much writes into the 7ECxxx area. You might be thinking when that item in the item box get updates. It gets updated when you move the cursor in the subscreen. That's it for the code which runs every frame to keep the screen looking right.

For other stuff which doesn't change it just gets loaded once from that 6FBxx area at the start of the game. The code is shorter than you'd think. The subscreen one is way longer though if you want to learn that one day.

I) FUTURE AREA HUD DISPLAY

Basically if you want to put code to make say text appear in a place (Area names), just make sure it goes around the same place where all these updating code is.

If you have trouble writing the ASM code, try thinking how you would do something like it in another programming language, then slowly convert it to ASM.

With the block set to 0 you get that page of gfx in yychr. The letters are in block 1
and if you scroll down in yychr until you see the font, line up the top left with that MAP gfx...
...that's where block 1 starts.

so A is block 1 0x50

~ Euclid™ Lesson 4 ~

ASM Patches, Pointers and Loops

A) ASM PATCHES & LABELS

For **CMP**, **BNE**, **BEQ** etc… use labels when writing ASM patches as follows:

org $02AB33
JMP NEW_CODE
org $208000
NEW_CODE:

After applying an ASM patch sometimes you should lunar expand it to 1.5mb or some even number.

In case you don't know how to apply ASM patches, here's a good tutorial to walk you through the whole process!

B) INSERTING ASM PATCHES USING XKAS

by wiiqwertyuiop

Things you will need:

xkas
Slogger or some other freespace finder
Lunar Expand
Command Prompt (CMD.exe)
A clean ROM (or whatever), preferably header-less
An ASM patch

Ok first open your patch. Unless the patch says otherwise, you will need to change the freespace in it. Now in the patch you should see something that says "freespace" or "change this!!" or something. Lets say it looks like this:

```
!Freespace = $xxxxxx ; Change this!!!
```

First open Lunar Expand and expand your ROM to 32Mbit (4mb) and apply the changes to your ROM. Next, drag and drop your ROM over slogger. You should get a text file named the same name as your ROM. Open it and you should see something like this.

```
PC offset     LoROM offset  Size

0x0A7FFF      0x14FFFF      0x0001
0x0BFFFF      0x17FFFF      0x0001
0x100000      0x208000      0x8000
0x108000      0x218000      0x8000
0x110000      0x228000      0x8000
...
```

It may be different, but yeah.

PC offset: This is the SNES address converted to a PC address, ignore it, usually you won't need it (for patching anyway).

LoROM offset: This is our freespace.

Size: This is how big our freespace is. The first two are useless and not recommended to use.

Now copy the address you want to use (e.g. 218000), and paste it over the one in the ASM file. So now we should have this:

```
!Freespace = $218000 ; Freespace (this should be changed)
```

Ok, now save and close the file. Now open CMD and type this in it:

xkas.exe The_name_of_the_patch.asm The_name_of_your_ROM.smc

Then hit enter.

That's it! If it worked, you successfully patched your ROM!

C) USING POINTERS TO DISPLAY HUD AREA NAMES

First it's better to know the addresses which the words are going to get written into the hud.

I'm placing it at 7EC804 upwards. First thing to do, you have to find out a name for each of the areas. In that case you should write them into some sort of format for the game. Here's a pearl script that should work along the names.txt file that contains the area names.

area name inserter.pl

```
# Work on NON HEADERED roms ONLY!

use Fcntl 'O_RDWR';
use Fcntl 'SEEK_SET';
use Fcntl 'SEEK_CUR';

die "Usage thing rom script\n"  if (@ARGV!=2);


sysopen DEST, "$ARGV[0]",O_RDWR or die "can't open $ARGV[0]!";
binmode DEST;

read DEST, $temp, 1;
die "I told you NON headered!\n" if ($temp ne chr(0x78));

@table = ( 0x2D,0x51,0x52,0x53,0x6C,0x6D,0x6E,0x6F,0x7C,0x86,
       0x87,0x88,0x89,0x8B,0x8C,0x8F,0x9C,0xA7,0xA9,0xAA,0xAB,0xAC,0xD7
       ,0xFA,0xF7,0x2E);

sub decodeLower {
       my $val = shift;
       if ($val ge 'A' and $val le 'Z')
       {
              return $table[ord($val) - ord('A')];
       }
       elsif ($val ge 'a' && $val le 'z')
       {
              return $table[ord($val) - ord("a")];
       }
       elsif ($val eq '-')
       {
              return 0x3F;
       }
       elsif ($val eq '~')
       {
              return 0x2F;
```

```
	}
	elsif ($val eq '\'')
	{
		return 0x30;
	}
	elsif ($val eq '?')
	{
		return 0xAF;
	}
	else
	{
		return 0x00;
	}
}

sub decodeUpper {
	return 0x20;
}



open ROM, "$ARGV[1]" or die "can't open $ARGV[1]!";

$i = 0;
while ($line = <ROM>) {
	chomp $line;
	($useless, $name) = split(" ",$line,2);
	@line = split('',$name);
	while (@line < 20) {
		$name = " ".$name;
		@line = split('',$name);
	}
	# $name should be 20 characters long

	sysseek DEST, 0x102800 + hex($useless) * 40, SEEK_SET;
	foreach $s (@line)
	{
		syswrite DEST, chr(decodeLower($s)), 1;
		syswrite DEST, chr(decodeUpper($s)), 1;
	}
	print hex($useless)," complete\n";
}

print "done.\n";
```

names.txt

```
00 ~Ancient Pyramid~
02 ~Pyramid Side~
03 ~?~
04 ~Triforce Shrine~
05 ~Sky Isles~
07 ~Parallel Tower~
0A ~Pyramid Falls~
0B ~Hyrule Castle~
0F ~Hidden Falls~
10 ~Victory Beach~
11 ~The Oasis~
15 ~Castle Guardhouse~
16 ~Lake Saria~
18 ~Endless Beach~
1B ~Lupo Intersection~
1C ~Merchant's Way~
1D ~Merchant's Way~
20 ~Beach Path~
```

```
21 ~Lupo Quarry~
23 ~Kakariko Village~
25 ~Village Cemetary~
27 ~Village Church~
28 ~Potion Shop~
2F ~Church Path~
30 ~Hyrule Docks~
32 ~Vanilla Beach~
34 ~Village Entrance~
35 ~Misty Forest~
37 ~Lost Woods~
3C ~Your House~
3F ~Secret Meadow~
40 ~Frozen Pyramids~
42 ~Fairy Fountain~
43 ~Lava Isles~
44 ~Triforce Shrine~
45 ~Lava Isles~
47 ~Parallel Tower~
4A ~Pyramid Falls~
4B ~Castle Site~
4F ~Hidden Ruins~
50 ~Victory Beach~
51 ~Mystery Hideout~
55 ~Merchant's Tent~
56 ~Ice Fortress~
58 ~Anti-sword Beach~
5B ~A Dead End~
5C ~Merchant's Way~
5D ~Merchant's Way~
60 ~Cold Beach Path~
61 ~Lupo Quarry~
63 ~Kakariko Village~
65 ~Village Cemetary~
67 ~Village Church~
68 ~Potion Shop~
6F ~Church Path~
70 ~Lake Ruins~
72 ~Icy Beach~
74 ~Village Entrance~
75 ~Snowy Forest~
77 ~Lost Woods~
7C ~?'s House~
7F ~Secret Meadow~
80 ~Ancient Temple~
81 ~Watefall cave~
82 ~Zoraking's domain~
```

Say for example, HYRULE CASTLE, what would you represent the letters with?

Like what **LDA** you have to do to load a H?

There should be a hex associated with every symbol/letter.

A is 50
B is 51
etc
H is 57

In that case you should have an area of the rom, where you insert these data
the way you do it is to have each of these area names separated by a special symbol
in this case we'll use FF because it's easy to use.

50 51 52 FF 50 51 52 FF ...
If Area 00 and 01 are both called ABC, you'll have this sort of data table for 0x80 areas
then it's the pointers.

50 51 52 FF 50 51 52 FF ...

Area00 will need a pointer there.
Area01 will need a pointer there.
etc…

So if this is the start of the data, 0000 is the pointer to Area00 while 0004 is the pointer to Area01. Then all you have to do is load that area pointer and you can find the start of the string, read it once you see a FF you know area name is at the end.

As for inserting the data, I would not suggest db-ing all these bytes for the data although you can. I would write it in the rom with a table.

A table file is a just a file which tells the hex editor what symbol is what letter.

In this case you'll have A=50, B=51 etc

Then in your rom, pick a good spot, and write the data. Make sure you have a character for space as well. Obviously that'll take a while so I'll just do a few for Area00, Area01 and 02

I'll be inserting the data at the end of the rom I guess (e.g. 0x104000).

First we reserve some space for those 2 byte pointers.

104000 - 104110 we'll leave out for pointers.
fills them with 00s
So we'll make the real data start at 104110.

The first byte should be FF.
It tells those 00s up there already that there's no area name.

So from the next byte onwards we write a few area names. How about Links house?

Ok so: South Spring

Now.. hex values:
62 5e 64 63 57 11 62 5f 61 58 5d 56 = SOUTH SPRINGS (11 is a space)

So from 104111, we type that. After that, we want to know what area(s) which has this name, Area 02 where Link starts. That area pointer starts at 0x104004.

Here we have two choices. Our pointer can be an absolute address (in this case C111) or a relative address (in this case 0001). Either way works, but it's easier to work with one you're comfortable with.

Relative means the other areas won't go look at 0x100000 for the address right now.

Let's do relative.

0001 would be the pointer.
So at 0x104004 you have 01 00.

Ok, we've made data for one area!

Now for some code. In this new routine, the job is to load area pointer, load area name, then write it to that RAM.

So first finding out some addresses.

104000 -> $20:C000

So a LDA $20C000,something, the first line of code should be to load the area.

LDA $8A

Then knowing each of the area number is a 2 byte pointer.
We need to multiply it by 2. ASL or shift left by 1. Then we
need to put that value onto the X or Y, we'll use X for now.

TAX

Then now we have the relative position. We can do a LDA $20C000,x
This will load the pointer if we're in Area 02. The code will load 02, then *2 = 04, then load 20C004... which is the right position. Now that we have the right pointer. We load it and we can now access to the data.

So after that LDA $20C000,x , we put a TAX to put it onto X register. So now $20C110,x is the start of the SOUTH SPRING.

Now we need some sort of loop. We clear Y register first LDY #$0000.

Before everything, REP #$30. It's important because when loading the pointer,
we want to load the 16bit address 0001 not just the 01.

Anyway after the clearing Y register with the LDY, all we need now is a little loop label:

LDA something
STA something
INX
INY
CMP #$some value
BNE label

That's usually what a loop looks like. We'll use the X register as the position to the actual data in 0x10411x positions and we'll use the Y register as the position to the 7EC804.

We write it into 7EC804 then 7EC806 etc...

The accumulator has to work in 8 bits because our data is 8 bits, so after the LDY:

SEP #$20 (keep the X on because the x and y registers can still be working in 16 bit mode)

Then in the loop because we're incrementing the 7Exxxx section by 2 at a time we'll need 2 INYs.

The LDA something and STA something of the loop LDA $204110,x should get you the start of the data. Then after each increment of X you'll get the next byte.

After the LDA though, we'll need a way to get out of the loop if we see FF label:

LDA $204110,x
CMP #$FF
BEQ endloop
STA something
INX
INY
INY
CMP #$some value
BNE label
endloop:
BEQ endloop just gets out of the loop
then the **STA**
STA $7EC804,y will do the trick

Then after the **INY**s, we don't need checking there, instead we should just go straight back so a
BRA label will do the trick.

label:
LDA $20C110,x
CMP #$FF
BEQ endloop
STA $7EC804,y
INX
INY
INY
BRA label
endloop:

Now you might think the loop is complete... well it's not finished yet. There is actually a problem with it. The **STA** $7EC804,Y, the main problem with this **STA** is that doesn't have a mode where it uses a long address + Y.

It means we have to use X

so we should do this:

LDA ....
...
BEQ
<save X here>
<put Y into X>
STA $7EC804,x
<restore X here>
INX
....
in these situations we could use the stack:
PHX - push X onto stack
PLX - pull X from stack

As for the put Y into X, just **TYX**.
The stack btw, usually sits at $1FF area.

Anyway you'll end up with it looking like this:

```
label:
LDA $20C110,x
CMP #$FF
BEQ endloop
PHX
TYX
STA $7EC804,x
PLX
INX
INY
INY
BRA label
endloop:
; now at endloop, you've done your job and need to set that X back to how it's started
; so you add a
SEP #$10
; then a
RTL
```

And that's it. Now we need to find the entry point for this code. I'll give you a sample address where the m is down.
Looking at the trace, why aren't we using **0DDD24** instead?

When picking these spots, it's often good to pick spots where the X and Y aren't used anymore
so we don't have to save them. The way to tell is just to see if there's any loading into Y before something replaces it.

Anyway **0D/DD24** looks good.

```
$0D/DD24 E2 30 SEP #$30 A:247F X:00FF Y:00FE P:envMXdiZC
$0D/DD26 E6 16 INC $16 [$00:0016] A:247F X:00FF Y:00FE P:envMXdiZC
```

A **SEP** #$30 which we're doing already + a **INC** $16
so we put **INC** $16 just before the **REP** #$30.

Your final ASM file should look something like this:

```
INC $16
REP #$30
LDA $8A
ASL
TAX
LDA $20C000,x
TAX
LDY #$0000
SEP #$20

label:
LDA $20C110,x
CMP #$FF
BEQ endloop
PHX
TYX
STA $7EC804,x
PLX
INX
INY
INY
```

```
BRA label
endloop:

SEP #$10
RTL
```

Anyway if you compile that code into $0D/DD24, then try the game the name should display there when you're at Area 02. At the moment it doesn't disappear yet... That'll just require a bit of extension to this ASM file. Don't worry about moving indoors because it cleans it for you.

~ Euclid™ Lesson 5 ~

Advanced Assembly

A) PLUG-IN ASM CODING

First you need to know what you're going to put in the RAM addresses.

What you need to do is to first find a place to do a **JSR/JSL**.
The trick is to look around where you have some **STA** $xxxxxx.
Which is 4 bytes, you can easily change that to a **JSL** to another bank.

After you done that the first thing to do is repeat that missing instruction

; so
STA $xxxxxx
; and then add your own code
LDA #$0090
STA $7EC700
RTL

You'll usually not find enough existing space, unless their programming is really crap and you can rewrite it... but rather than rewriting, sometimes it's easier to just plug code in like that.

The difference between **JSR** and **JMP**:
If you use **JMP** you'll need to end it with a **JMP** to come back to the code.
If you use JSR you can use **RTS/RTL.**

Most of the time you would want **JSL**.
Because you would want to jump to another area of the rom say the end of the rom.

Q: *How do you go about expanding a rom in terms of jumping to the end*

of the rom to write new code?

A: Just write some 0s to the end of the rom.
If you want, you can use Lunar Expand the same thing happens.
If you want some free space, I think around 0x3FF00 there's a lot there.

Q: *What kind of code gets read when you enter a new area?*

A: hmm tile layout, palettes also the blank out, maybe overlays as well since blank out is loaded.

B) MUTIPLE CMPs AS CONDITIONS

Q: *How can u have multiple **CMP**s as conditions for comparing the Accumulator, X/Y memory?*

A: This will require some more code:

LDA $8A
CMP #$3F
BEQ code
CMP #$02
BEQ code
RTS or **BRA** below the code
code:
LDA #$21
STA $0AA1

C) SRAM ELEMENTS (HEARTS ETC) AND FRAMES

Q: *Do u know how i can write a certain tile to 7EC7xx-7EC8xx if I have more than 10 hearts?*

A: Ah you want like the RAM address like try 7EF36C
It's in the sram document, everything in the sram document is copied to 7EFxxx when the game starts

Q: *How would I time something to appear on the hud for like say only 3 seconds?*

A: 3 seconds is about 180 frames, should be 60 frames per sec and each frame $1A gets incremented by 1. You would have to keep some value to find out when the $1A has reached the original value + 180 then clear it.

If you need some free memory go look in the 7E1CFx section there is supposed to be lots of free memory lying around, not sure if they're documented.

1CF2 onwards to 1CFF is definitely unused, make that 1CF3 to be safer.
If you're doing the exact same thing as what the original display code is doing.

One of the big differences between my hud implementation to others is that I actually went in there and cleaned out all the code related to the hud and rewrite most of it.

Of course some stuff like calculating hearts etc are left in place, just their position to draw to is changed.

What you should do is find the places where it draws to those positions (just breakpoint on write), then change the position it writes to.

Q: *I've been trying to work it out but how do i do an IF statement that says IF A >= #$50 BRANCH?*

A: Let's see if I remember correctly.
BCS is branch greater than or equal.
BCC is branch smaller than.
But right before that you need to do a **CMP** or **CPX** etc...

Q: *So use a **CMP**? When would I use **XOR** **OR** or **AND**?*

A: XOR OR AND is when you need to find common patterns.
Say you want to test if the rupee counter is even or odd.

You would do:
LDA rupee
AND #$01
BEQ even

Like you know that big list of instructions in the hud code we explored before? The big list of STAs. I bet you, if you rewrite that using loops you could save a lot of space ;)

Not that it really matters though. Make use of the X and Y registers.
Use one of them as the loop counter, of course make sure they aren't being used.

Don't worry about the ASM expansion size, its limitless.
It's achievable with all the opcodes you know, think of it as programming.

In this case you would have some large string characters names

"area00name","area01name" ... etc

And you would want a list of pointers to point to the beginning of each of these area names
then in your actual code you would load your area code into the X or Y register and do a relative lookup onto the pointers to see where the name is and load that.

D) HEX AND BITS FOR RGB TRANSPARENCY

Let's talk about hex and bits while we're at it.

First you need to know that 0x2629 in hex

is 0010 0110 0010 1001 in bits.

The format of the RGB in this case is
xBBB BBGG GGGR RRRR

x is not used, RGB is rgb

So,
B - 01001 = 9
G - 10001 = 17
R - 01001 = 9

Then multiply those values by 8:
B - 9 * 8 = 72
G - 17 * 8 = 136
R - 9 * 8 = 72

So the rgb for 0x2629 is 72,136,72

You'll also notice HyruleMagic stores all palette information in multiples of 8.
So you'll never get something like 73,191,71 as RGB, because that is not a valid colour in the old snes.

E) OVERLAYS AND THEIR LOCATIONS

131B6 - change to overlay you want to test (9F I'm guessing)
131B9 - area to use the overlay 1
131BE - area to use the overlay 2
131C3 - area to use the overlay 3

Areas 00, 03, 05, 07 and 40 won't work because the code for them is loaded beforehand.

F) TRANSITION BETWEEN OW AREAS

Need to alter the way Overworld moves from area to area?
At hex address 12CDD change the value to 80 = Changes the scroll to fade out.

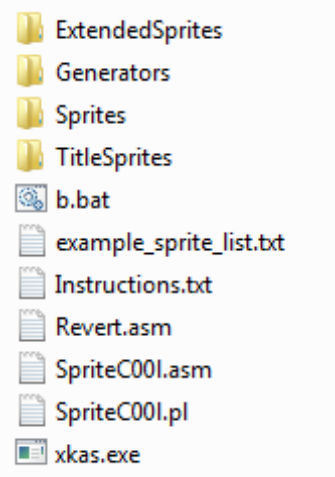
3) Custom Sprite Tutorial

by wiiqwertyuiop

This is a tutorial I made for inserting and making custom sprites. Since none of the current level editors for ALttP (Hyrule Magic/Black Magic) support these types of sprites inserting them can be a little tricky, but if you follow along carefully it shouldn't be that hard to do.

What you will need:

A) THE SPRITECOOL TOOL (CUSTOM SPRITE INSERTER)



https://www.dropbox.com/s/fgxuxry6xa8iqx2/CustomSpriteInserter.zip

Note: *This tool will NOT work with headered ROMs.*

Instructions:

1. Create a .txt file a name it to your sprite list name (I don't think spaces will work in the name, but I haven't tried it).

2. In the sprite list you need to type in your sprite type, number and file. The format is like this:

[Sprite type] [Sprite number] [Sprite file]

Sprite numbers: Any number from 0-255 (in decimal) is valid.
Sprite file: This is the sprite file along with the .asm extension
Sprite types:
s - Normal (Custom) Sprites - This will take the sprite files from the ***"Sprites"*** folder.
g - Generators - This will take the sprite files from the ***"Generators"*** folder.
e - Extended Sprites - This will take the sprite files from the ***"ExtendedSprites"*** folder.
t - Title screen sprites - This will take the sprite files from the "TitleSprites" folder.

3. Open "SpriteC00l.asm" and change the freespace and freeram if needed.

4. Get a ROM and if you haven't already done, expand it using Lunar Expand.

5 Edit the batch file (**b.bat**) according to your current rom name and sprite list and then run it! ** make sure you also have Perl installed on your Operating System! If that's not the case, I suggest you do it right now as it will be necessary for some scripts! **

6. It will ask you for your ROM name, type in the name along with the .sfc/.smc/whatever extension.

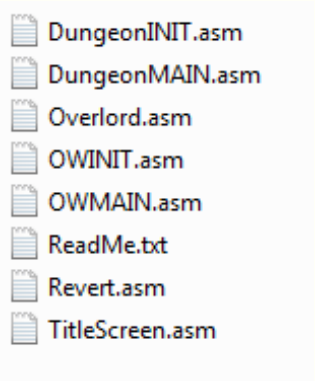
7. Then type in your sprite list name, along with the .txt extension, and hit enter.

8. After that it will install the sprites and you are ready to use them!

perl SpriteC00l.pl s.sfc s.txt
@pause

"**s.sfc**" is where you enter your current rom name/extension,
while "**s.txt**" is actually your sprite list name.

B) THE OVERLORD PACKAGE



https://www.dropbox.com/s/qh4nmtpcn8h1yv9/Overlord.zip

What is Overlord?

Overlord will run a specific code in any dungeon/OW screen of your choosing.

The MAIN codes will run every frame in a level.

The INIT codes will run before the MAIN code for one frame. It can be used to initialize your codes, or run a certain code once in the beginning.

- What file do I patch to my rom?

Patch Overlord.asm. If you try to patch any of the others asm files alone, it'll likely crash your ROM. In fact if you look at the end of the Overlord.asm file you'll notice those lines at the end:

```
incsrc DungeonINIT.asm
incsrc DungeonMAIN.asm
incsrc OWINIT.asm
incsrc OWMAIN.asm
incsrc TitleScreen.asm
```

Those lines of text means that the Overlord.asm file is including those files when you're actually applying the patch!

Also make sure those files are in the same folder as xkas!

- How do I work this?:

For dungeons:
First open your ROM in HM (Hyrule Magic), and find the dungeon you want to run the code for. At the top left of the dungeon window you should see "Room: XXX", take that number and convert it to hex (you can use windows calculator or something).

e.g. if I want to run a code in room 260, I'd convert that to hex then Ctrl+F it in dungeon MAIN/INIT, and put my code there.

For Overworlds:
The Overworld has already been converted to hex so just find the Overworld level you want ("Area XX") to run the code at and Ctrl+F it in OW MAIN/INIT.

Also note that every time you add new code you must patch the patch again!

Want to save some space? If you find a spot in the ROM where you want to use its code, first copy the address (e.g. $00803B) then open Overlord.asm. Ctrl+F the dungeon/OW number you want to run the code at and replace the label with address (e.g. "dl Dungeoninit2C" -> "dl $00803B").

- Any code examples?

There is one in "OWmain2C:"

```
OWmain2C:
       INC $E0             ; Increase BG1 X pos
       LDA $1A             ;\
       LSR #4        ; | Every couple of frames...
       BCC +         ;/
       DEC $E6             ; Make BG1 scroll down
       +
       RTL
```

Before proceeding further

If you want to know how to make sprites you will both need the **SpriteC00l tool** and **Overlord** as well as "**Basic ASM knowledge**" (if you don't know what **LDA** and **STA** do, try to learn those first)... and **xkas** (A SNES S-CPU cross assembler).

C) USING THE SPRITES

This tutorial explains how to use the sprites after you have inserted the sprites with SpriteC00l.asm. Now how do you use them you say? Well since Hyrule Magic or any other level editor currently doesn't support these sprites we will need to use Overlord.asm.

If you haven't done so already first open up DungeonINIT.asm if you are using this sprite for a dungeon or OWINIT.asm if you are using it in the OW. Go to the level this is for. Once you are there copy and paste this there:

```
LDA #$SpriteState
STA $00
LDA #$SpriteSpawn
STA $01
LDA #$Xpos
STA $02
LDA #$Ypos
STA $03
LDA #$XposHigh
STA $04
LDA #$YposHigh
STA $05
LDA #$ExtraSettings
STA $06
LDA #$SpriteType
JSL SpawnSprite
```

Now before you do anything you need to replace those words I have with your values:

SpriteState = Unless the sprite you have says otherwise just leave it at #$08.

SpriteSpawn = This is the sprite to spawn/sprite number. You replace this with the custom sprite number. We are going to assume the sprite I inserted was the meat sprite I included with SpriteC00l, and that I inserted it as sprite number #$00.

Xpos = This is the sprite's X pos, or horizontal pos. How do we find this? Well if you go to Hyrule Magic switch to sprite editing mode, select at sprite then move it to where you want your sprite, then at the bottom you should see this:

Spr XX
X: xx
Y: xx
BG1
P: xx

Get the X pos. and open a calculator, then switch to hex (for windows go to view>programmer then switch to hex), and multiply the X pos by 8. That's the X pos for your sprite.

Ypos = Same as above, but use the Y pos.

XposHigh = If you want to use the left half of the screen use #$08 if you want to use the right half use #$09. Also when multiplying if you get a number like 1xx make the high byte #$09.

YposHigh = If you want to use the top part of the screen use #$20 if you want to use the bottom part use #$21. Also when multiplying if you get a number like 1xx make the high byte #$21.

Note: When in the OW you need to get the pos's differently (It's actually a bit easier). In the menu click "**Addr. Calc**"

then click where you want your sprite. A window will open up telling you the position. The first number is the **X pos**, and the second one is its **Y pos**. You'll see you get a bigger number though. Well the first two numbers (**XX**YY) are the high byte of your pos, so just stick them in your high byte. Then for the last two numbers you need to round it to its closest number divisible by 10. So if you get E5, that's E0, if you get 44 that's 40, etc.

ExtraSettings = Unless the file says otherwise leave it at #$01. The meat sprite we are using says:

```
;; Uses extra settings? Yes
;;
;; #$01 - Small meat
;; #$02 - Big meat
;; #$03 - Small meat (bad)
;; #$04 - Big meat (bad)
```

So if we set it to #$01 it will be small meat, if we set it to #$02 it will be big meat, etc.

SpriteType = This is the type of custom sprite the sprite is.
#$00 = Normal (Custom) Sprite
#$01 = Extended sprite
#$02 = Generator
#$03 = Title Screen Sprite

This might be a good time to explain what each of these sprites do.

Normal custom sprites and extended sprites aren't really anything special, extended sprites are usually used for special effects and projectiles, but that's about it.

Generators run every frame in the level and don't normally don't use GFX.

Title screen sprites will run during the title screen *only*.

Now insert Overlord.asm again enter your level and you should see your sprite!

D) MAKING CUSTOM SPRITES

Ok let's first start by making a generator, since they are a bit easier to make. The first thing you need to do is create a .txt file in the generators folder.

After that rename it to your sprite name and replace the ".txt" extension with ".asm".
Make sure there are no spaces in the name. Ok now open the file and type your MAIN label in the sprite (there is no INIT in generators). Like this:

```
.MySprite
```

It's better to use periods in front of labels since it will stop inserting errors if someone else uses the same label name as you. Ok now let's make it do something. Let's make the rupee counter always stay at #$00. For this we just need to make $7EF360 #$00. So:

```
.MySprite
LDA #$00
STA $7EF360
```

```
RTL
```

Insert your sprite and it should work! Let's make another one. How about one that gives you a bomb after you get 10 rupees then resets it back to #$00? Try this:

```
.MySprite
LDA $7EF362
CMP #$0A
BCC .Return

LDA #$01
STA $7EF375

LDA #$00
STA $7EF360
.Return
RTL
```

Now what that code does is first check if we have 10 (#$0A in hex) rupees, if we have less than 10 coins (BCC) return, if we have 10 rupees we increase the amount of bombs we have by 1, and then reset the counter.

So yeah that's about it for making generators. How about we move on to custom sprites?

First put in your INIT and MAIN labels, like so:

```
.INIT
RTL

.MAIN
RTL
```

Now let's first make a static sprite, meaning one that only has GFX but no function. First it's good to keep your code clean so let's JSR to our GFX routine like this:

```
.INIT
RTL

.MAIN
JSR .GFXRoutine
RTL

.GFXRoutine
RTS
```

4) Some info regarding the WLA DX Assembler

by d4s

If you want a fast routines patcher, flexible and -most important- easy to maintain tool, you have to get **WLA DX**, it's a multiple platform assembler sporting **65816** and **spc700** support
and it also has the ability to patch roms directly.

Here's an example, should be fairly easy to understand.

Just for the record, this routine replaces the "Get next pointer for decompressed graphics block" loader in Zelda 3.

This was coded because I wanted to move the graphics in the rom around, especially to get around the problem with recompressed GFX blocks that are bigger than the original ones.
This way, **WLA DX** relocates all graphics blocks dynamically every time I compile my code.

I could even move it from one bank to another in a matter of seconds.

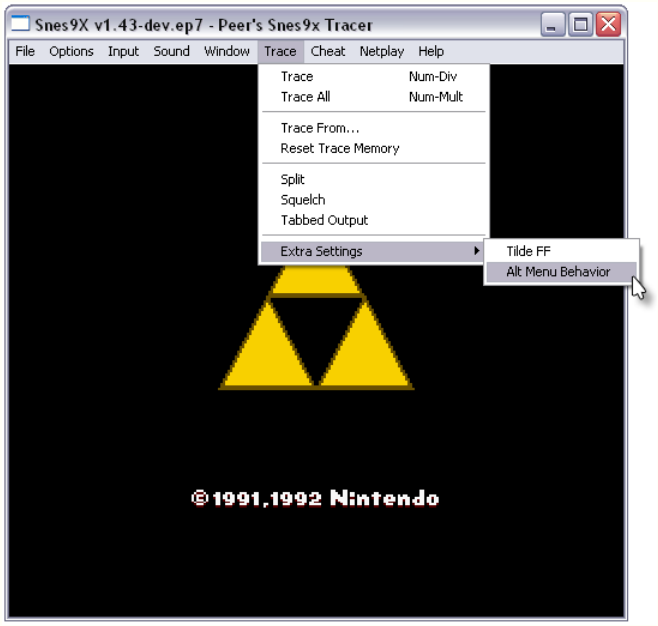
Here is the example:

```
;=============================================================
; Hack
;=============================================================
; Hooktrap for compressed GFX pointer loading routine

.BANK 0 SLOT 0 <----Apply patch on bank 0, offset $E7783
.ORG $E783

JSL (CompGfxPointerLoader+10354688) <------Jump to my code, hirom (So I have more space
per bank, since lorom is so annoying)
JMP $E79C
;=============================================================
; Patch
;=============================================================
;Custom loader for Zelda 3 compressed 3BPP GFX

.BANK 33 SLOT 0 <----Selects which bank to use
.ORG 0
.Section "Main Intro Code" overwrite <-------Let WLA DX dynamically handle the ORGing and
overwrite rom (Although Zelda 3 has nothing in bank 33 cause it's just 8mbits.)

CompGfxPointerLoader:

PHP ;Save cpu status
PHB ;Save last bank
LDA.B #(:CompGFXBankTable+192) ;Get bank where the pointertable is stored
PHA ;Push that shit on the stack
PLB ;Get current data bank from stack
SEP #$30 ;Index/mem 8bit
STZ $00 ;
LDA.B #$40 ;Set up destination
STA $01 ;Decompression buffer
LDA.B #$7F ;In wram, 7f400
STA $02 ;
STA $05 ;
LDA CompGFXBankTable,y ;Get next pointer
STA $CA ;Store bank
REP #$30 ;Index 16bit
TYA
ASL A ;Wish asl y was possible.
TAY ;Multiply pointerselector by two cause that's
LDA CompGFXTable,y ;A two byte table.
STA $C8 ;Store offset in $c8 and $c9
```

```
TYA
LSR A
TAY
LDA $C8
PLB ;Restore data bank
PLP ;Restore cpu status
RTL ;
```

Now, we also have a nice 2 byte pointertable + separate banktable, instead of the ugly splitted shit that was there in the first place.(A separate pointertable for each bank, lo address and hi address, which is hard to follow).

5) Geiger's Snes9x Debugger



This version of Snes9x was compiled by Evil Peer. The official Snes9x team will not support it. PST does not support either Glide or Fmod.

PST was built with Visual Studio 2003 and the latest Microsoft development kits. As such, you may also need:

- mfc71.dll
- msvcp71.dll
- msvcr71.dll
- DirectX 9.0b (Probably not necessary though...)

DLL Libraries: http://www.zophar.net/Files/msvcp71.zip
http://www.zophar.net/Files/msvcr71.zip
http://www.zophar.net/Files/mfc71.zip

Features:

- Trace . trace each instruction once only
- Trace All . trace every instruction, every time
- Trace From, Too . trace from a SNES address after its been executed a certain number of times until it reaches another SNES address a certain number of times. Place a zero in any unwanted fields.

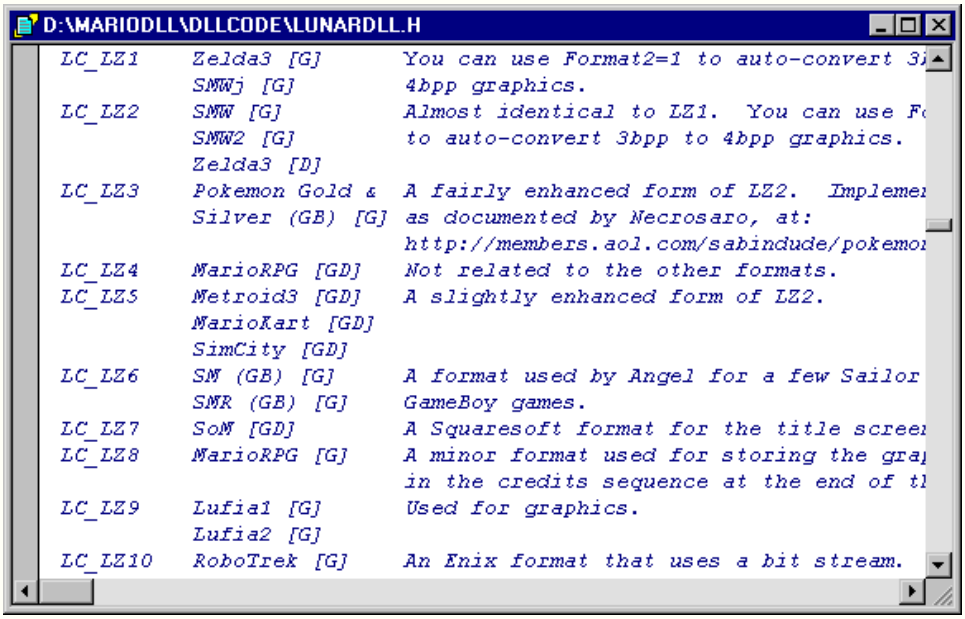
- Capture Every Pass . available on the Trace From dialog, this setting will trace a section of code to file every time it's executed.
- Reset Trace Memory . resets all internal trace variables back to their bootup values.
- Split . splits trace files after 65535 lines (around 5 megs in squelch mode)
- Squelch . squelches some of the less useful information in trace files (reduces file size by 25%)
- Tabbed Output . produces tabbed fields for spreadsheets or databases
- Tilde FF . use the tilde key (~) for fast forward, like ZSNES
- Alt Menu Behavior . causes the escape key to call the menu and pause emulation

Known Issues:

- Sound: Not sure if it's my build or just 1.42 in general, but long sound buffers (640ms) will cause considerable delay between onscreen action and sound.40ms was tested and seemed to work well.

URL: http://www.zophar.net/Files/snes9x1.43-dev.ep7.zip

6) Lunar Compress



Lunar Compress is a decompression and recompression DLL written in C for a few compression formats that have been known to show up in certain SNES/GB games. It's intended primarily as a programmer's resource, so it even includes a few common functions that may be useful for SNES ROM editing (such as ROM/PC address conversion, ROM expansion, bpp/indexed GFX conversion, etc.).

The zip file contains the source code required for accessing the DLL, and two simple command line utilities that may prove useful if you just want to test or use the DLL's compression capabilities without having to code anything. The source code for these two programs has been provided, as well as the source to a simple win32 GUI sample program for Super Mario World so you can examine how to use the DLL correctly.

A small brute force tool called "sniff" that can occasionally be useful for locating compressed data offsets is also included (check the sniff.txt file).

For documentation on the DLL function calls and the values used to represent each format, please read the files "LunarDLL.h" and "LunarDLL.def". While the DLL and utilities have all been written in C, other languages should be able to access the DLL just fine, including VB.

Lunar Compress DLL + Development Files v1.61:

http://fusoya.eludevisibility.org/lc/download/lc161.zip

7) Miscellaneous Notes

Every 60th of a second an "interrupt" is called called the NMI. In Zelda 3 the interrupt is located at **$000069** in rom and **$00:8069** in SNES memory (for the American rom). Now I'm no expert on this, but I believe that is called VBlank or NMI. It's where your graphical data gets updated to the screen.

There is also Hblank (horizontal blank) where you can alter graphics as they are being drawn every scanline. If you don't know what a scanline is feel free to ask. That is essentially how things like oval shaped black spot lights are created when Link goes into and out of a building. The width of a black object is changed every scanline. I believe that Hblank effects are handled using HDMA or possibly just DMA. I need to research this a bit more as well, but I've come to terms with it enough to recognize it when I see it.

You don't have to use the nmi. As most regs (Like the video regs, the audio ports etc) have buffers in wram. That means you don't have to write to the regs during vblank, you can just write to these buffers when you want to and the values will automatically be written to the regs during vblank.

There's also similar stuff for dma transfers, but that's a bit more complex.

Most vram stuff is also buffered in vram, like the pause menu, the status display (i mean on BG3) and the complete oam table(for sprites). You can just write there and everything will get transferred during the next vblank. Don't know about hdma, but there should be a buffer for that, too. I'm going to have a look at it later as I need it.

8) Links to some useful guides

Super NES Programming - by Wikibooks

http://en.wikibooks.org/wiki/Super_NES_Programming

This book is designed for people interested in learning to program for the **Super Nintendo Entertainment System** and, for now, assumes some basic knowledge on how to write an assembly language program, how to use a command prompt, and how to use an emulator.

Qwertie's SNES Documentation

link

description

anomie docs

9) ASM Hacks Database

screen screen

dgdfhdzfbcvbfcnb
cvbzcvnbcvncvn
vncvncvncvcvnvc
dhzdfhzdfdfhzdf

- ASM Hacks by Euclid -

Hack: Palette Patch
Author: Euclid
Information: Patch world colour loading to be area dependant.
Screenshots:

screen screen

Rom: ALTTP (US), without header
Code Addresses:
@ **3F8C0**

world color 1 pal 0 is light world non death mountain areas (areas other than 3,5 and 7)
world color 1 pal 1 is dark world non death mountain areas (areas other than 43,45 and 47)
world color 1 pal 2 is light world death mountain areas (areas 3,5 and 7)
world color 1 pal 3 is dark world death mountain areas (areas 43,45 and 47)

Map 1, indicates which pal to use, only proven to work with 0,1,2,3.
Areas 00 to 3F

```
0 0 0 2 2 2 2 2
0 0 0 2 2 2 2 1
0 0 0 0 1 1 1 1
```

```
0 0 0 0 1 1 1 1
0 0 0 3 3 1 1 1
3 3 3 3 3 1 1 1
3 3 3 3 3 3 1 1
3 3 3 3 3 3 1 1
```

Areas 40 to 7F

```
0 0 0 2 2 2 2 2
0 0 0 2 2 2 2 1
0 0 0 0 1 1 1 1
0 0 0 0 1 1 1 1
0 0 0 3 3 1 1 1
3 3 3 3 3 1 1 1
3 3 3 3 3 3 1 1
3 3 3 3 3 3 1 1
```

Areas 80+

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

@ **3F9C0**

Map 2, if it's a dark world (using 1 or 3) set it to 2, otherwise 0.
Areas 00 to 3F

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2
0 0 0 0 2 2 2 2
0 0 0 0 2 2 2 2
0 0 0 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
```

Areas 40 to 7F

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2
0 0 0 0 2 2 2 2
0 0 0 0 2 2 2 2
0 0 0 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
```

Areas 80+

```
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

----------------
```

ASM:

Code change to load Map 1:

```
$02/C694 A5 8A       LDA $8A    [$00:008A]   A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C696 29 3F       AND #$3F                A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C698 C9 03       CMP #$03                A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C69A F0 0A       BEQ $0A    [$C6A6]      A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C69C C9 05       CMP #$05                A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C69E F0 06       BEQ $06    [$C6A6]      A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6A0 C9 07       CMP #$07                A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6A2 F0 02       BEQ $02    [$C6A6]      A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6A4 A2 00       LDX #$00                A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6A6 A5 8A       LDA $8A    [$00:008A]   A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6A8 29 40       AND #$40                A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6AA F0 01       BEQ $01    [$C6AD]      A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6AC E8          INX                     A:0000 X:0000 Y:0000 P:EnvMXdIzc
$02/C6AD 8E B3 0A    STX $0AB3  [$00:0AB3]   A:0000 X:0000 Y:0000 P:EnvMXdIzc
```

Change this to

```
$02/C694 A6 8A       LDX $8A
    C696 BF C0 F8 07 LDA $07F8C0,X
    C69A AA          TAX
    C69B 80 10       BRA to following location.
    .... EA          NOP
$02/C6AD 8E B3 0A    STX $0AB3
```

Code change to load Map 2:

```
$1B/ECA2 A5 8A       LDA $8A    [$00:008A]   A:0000 X:0000 Y:0000 P:EnvmXdIzc
$1B/ECA4 29 40 00    AND #$0040              A:0000 X:0000 Y:0000 P:EnvmXdIzc
$1B/ECA7 F0 02       BEQ $02    [$ECAB]      A:0000 X:0000 Y:0000 P:EnvmXdIzc
$1B/ECA9 E8          INX                     A:0000 X:0000 Y:0000 P:EnvmXdIzc
$1B/ECAA E8          INX                     A:0000 X:0000 Y:0000 P:EnvmXdIzc
```

Change this to

```
$1B/ECA2 A6 8A       LDX $8A
    ECA4 BF C0 F9 07 LDA $07F9C0,X
    ECA8 AA          TAX             # no need to AND #$00FF because X register will chop it off.
    ECA9 EA          NOP
    ECAA EA          NOP
```

Hack: Start up Patch
Author: Euclid
Information: Set state of the Save RAM after entering name.
Screenshots:

screen **screen**

Rom: ALTTP (US), without header
Code Addresses:

ASM:

Definitely one of the things i didn't document/ Lost documentation for PW:

In PW Rom:

```
$0C/DC58 A6 00       LDX $00    [$00:0000]   A:0000 X:0A3C Y:0078 P:eNvmxdizC
$0C/DC5A 20 00 EC    JSR $EC00  [$0C:EC00]   A:0000 X:0A00 Y:0078 P:envmxdizC
```

The JSR indicates the hook into the current code straight after clicking END on the name selection screen.

The same principle can apply here to use the same spot.

The PW code compares the name with something, in this case we don't really care so the following code does not apply:

```
$0C/EC00 E2 20       SEP #$20               A:0000 X:0A00 Y:0078 P:envmxdizC
$0C/EC02 8B          PHB                    A:0000 X:0A00 Y:0078 P:envMxdizC
$0C/EC03 A9 0C       LDA #$0C               A:0000 X:0A00 Y:0078 P:envMxdizC
$0C/EC05 48          PHA                    A:000C X:0A00 Y:0078 P:envMxdizC
$0C/EC06 AB          PLB                    A:000C X:0A00 Y:0078 P:envMxdizC
$0C/EC07 C2 20       REP #$20               A:000C X:0A00 Y:0078 P:envMxdizC
$0C/EC09 A0 00 00    LDY #$0000             A:000C X:0A00 Y:0078 P:envmxdizC
$0C/EC0C BF D9 03 70 LDA $7003D9,x[$70:0DD9] A:000C X:0A00 Y:0000 P:envmxdizC
$0C/EC10 D9 32 EC    CMP $EC32,y[$0C:EC32]  A:0023 X:0A00 Y:0000 P:envmxdizC
$0C/EC13 D0 10       BNE $10    [$EC25]     A:0023 X:0A00 Y:0000 P:envmxdizC
```

```
$0C/EC15 E8          INX                     A:0023 X:0A00 Y:0000 P:envmxdizC
$0C/EC16 E8          INX                     A:0023 X:0A01 Y:0000 P:envmxdizC
$0C/EC17 C8          INY                     A:0023 X:0A02 Y:0000 P:envmxdizC
$0C/EC18 C8          INY                     A:0023 X:0A02 Y:0001 P:envmxdizC
$0C/EC19 C0 0C 00    CPY #$000C              A:0023 X:0A02 Y:0002 P:envmxdizC
$0C/EC1C D0 EE       BNE $EE    [$EC0C]      A:0023 X:0A02 Y:0002 P:eNvmxdizc
$0C/EC1E A9 08 00    LDA #$0008              A:00A9 X:0A0C Y:000C P:envmxdizC
$0C/EC21 A6 00       LDX $00    [$00:0000]   A:0008 X:0A0C Y:000C P:envmxdizC
$0C/EC23 9F 58 03 70 STA $700358,x[$70:0D58] A:0008 X:0A00 Y:000C P:envmxdizC
$0C/EC27 A0 00 00    LDY #$0000              A:0008 X:0A00 Y:000C P:envmxdizC
$0C/EC2A A6 00       LDX $00    [$00:0000]   A:0008 X:0A00 Y:0000 P:envmxdizC
$0C/EC2C E2 20       SEP #$20                A:0008 X:0A00 Y:0000 P:envmxdizC
$0C/EC2E AB          PLB                     A:0008 X:0A00 Y:0000 P:envMxdizC
$0C/EC2F C2 20       REP #$20                A:0008 X:0A00 Y:0000 P:envMxdizC
$0C/EC31 60          RTS                     A:0008 X:0A00 Y:0000 P:envmxdizC
```

So the above code will need to be modified as follows:

@ **67FB1** (because the same spot in PW is used....)

// for the correct values to put in I used a PW save near the end:

```
A9 03 33    LDA #$3303
9F C5 03 70 STA $7003C5,x
A9 05 01    LDA #$0105
9F C7 03 70 STA $7003C7,x
```

// sword 1 and shield 1

```
A9 01 01    LDA #$0101
9F 59 03 70 STA $700359,x


A0 00 00    LDY #$0000
60          RTS                     A:0008 X:0A00 Y:0000 P:envmxdizC
```

The Hook:

@ **65C5A**

```
$0C/DC5A 20 B1 FF    JSR $FFB1  [$0C:EC00]
```

Overwrites

```
$0C/DC5A A0 00 00    LDY #$0000
```

Now to that annoying message box

The main control is the follow line:

```
$02/81CD AF C5 F3 7E LDA $7EF3C5[$7E:F3C5]   A:4300 X:0040 Y:00FF P:envMXdiZc
$02/81D1 C9 02       CMP #$02                A:4303 X:0040 Y:00FF P:envMXdizc
$02/81D3 90 33       BCC $33    [$8208]      A:4303 X:0040 Y:00FF P:envMXdizC
```

If it branches - Treat as start - so start an X location
If it doesn't branch, the box pops up.

Change `CMP #$02` to `CMP #$04` - which covers 99% of the case.

With the above patch - this gets the user to start in the church... can't seem to figure out how to change the room number to be 260.

Hack: Antisword shooter
Author: Euclid
Information: 4way shooter -> anti sword shooter

You can adjust the speed! There's 2 numbers you can play with.

Value 1:
Rom Offset 0xEC8C7: (original rom should have 20, after this patch should be 0C)
Must be bigger than value 2 and not be 0.

Value 2:
Rom Offset 0xEC876: (original rom should have 18, after this patch should be 05)
Must be smaller than value 1 and not be 0.

The difference between the 2 values is as follows:
Value 1 is a timer, which is set when you start swinging your sword, the timer counts down from the value set to 0. When the timer is the same value as Value 2, the shooting comes out. The timer has to be counted down to 0 prior to the timer gets reset.

For fun, try setting value 1 to 02 and value 2 to 01, it comes out like crazy and lags the game too.

Screenshots:

screen **screen**

Rom: ALTTP (US), without header
Code Addresses:

ASM:

Technical stuff someone, some day may need and read:

Didn't have notes for this one in PW so i have to dig it up again.

2 hacks to be done here:

1. the direction to shoot the shooter - AT you rather than 4 directions.
2. SPEED of the shooter.

1 calculates which WAY to shoot the shooter,

```
$1D/C887 B9 65 C8    LDA $C865,y[$1D:C865]   A:FF44 X:000D Y:0000 P:envMXdiZC
$1D/C88A 9D 50 0D    STA $0D50,x[$1D:0D5D]   A:FF28 X:000D Y:0000 P:envMXdizC
$1D/C88D B9 63 C8    LDA $C863,y[$1D:C863]   A:FF28 X:000D Y:0000 P:envMXdizC
$1D/C890 9D 40 0D    STA $0D40,x[$1D:0D4D]   A:FF00 X:000D Y:0000 P:envMXdiZC
```

In PW, the first 2 bytes are replaced by a **BRA #$0A**, which skips the above code
This makes the shooter to shoot AT you rather than doing the rounding (becomes the 4 way shooter)

- saving the result into the sprites memory

```
$1D/C893 BD 10 0D    LDA $0D10,x[$1D:0D1D]   A:FF00 X:000D Y:0000 P:envMXdiZC
$1D/C896 18          CLC                     A:FFB4 X:000D Y:0000 P:eNvMXdizC
$1D/C897 79 53 C8    ADC $C853,y[$1D:C853]   A:FFB4 X:000D Y:0000 P:eNvMXdizc
$1D/C89A 9D 10 0D    STA $0D10,x[$1D:0D1D]   A:FFC0 X:000D Y:0000 P:eNvMXdizc
$1D/C89D BD 30 0D    LDA $0D30,x[$1D:0D3D]   A:FFC0 X:000D Y:0000 P:eNvMXdizc
$1D/C8A0 79 57 C8    ADC $C857,y[$1D:C857]   A:FF04 X:000D Y:0000 P:envMXdizc
$1D/C8A3 9D 30 0D    STA $0D30,x[$1D:0D3D]   A:FF04 X:000D Y:0000 P:envMXdizc
$1D/C8A6 BD 00 0D    LDA $0D00,x[$1D:0D0D]   A:FF04 X:000D Y:0000 P:envMXdizc
$1D/C8A9 18          CLC                     A:FFC4 X:000D Y:0000 P:eNvMXdizc
$1D/C8AA 79 5B C8    ADC $C85B,y[$1D:C85B]   A:FFC4 X:000D Y:0000 P:eNvMXdizc
$1D/C8AD 9D 00 0D    STA $0D00,x[$1D:0D0D]   A:FFC4 X:000D Y:0000 P:eNvMXdizc
$1D/C8B0 BD 20 0D    LDA $0D20,x[$1D:0D2D]   A:FFC4 X:000D Y:0000 P:eNvMXdizc
$1D/C8B3 79 5F C8    ADC $C85F,y[$1D:C85F]   A:FF06 X:000D Y:0000 P:envMXdizc
$1D/C8B6 9D 20 0D    STA $0D20,x[$1D:0D2D]   A:FF06 X:000D Y:0000 P:envMXdizc
$1D/C8B9 FA          PLX                     A:FF06 X:000D Y:0000 P:envMXdizc
```

2 the SPEED of the shooting

Think event counter, see the LDA #$20 down there? it gets saved into the sprite's RAM

```
$1D/C8BF BD 20 0F    LDA $0F20,x[$1D:0F20]   A:0001 X:0000 Y:0000 P:envMXdizC
$1D/C8C2 C5 EE       CMP $EE    [$00:00EE]   A:0000 X:0000 Y:0000 P:envMXdiZC
$1D/C8C4 D0 05       BNE $05    [$C8CB]      A:0000 X:0000 Y:0000 P:envMXdiZC
$1D/C8C6 A9 20       LDA #$20                A:0000 X:0000 Y:0000 P:envMXdiZC
$1D/C8C8 9D F0 0D    STA $0DF0,x[$1D:0DF0]   A:0020 X:0000 Y:0000 P:envMXdizC
```

and on every "tick" the value is decremented, when it is decremented to a specific RAM value (#$18 below), the event occurs.

```
$1D/C870 BD F0 0D    LDA $0DF0,x[$1D:0DF0]   A:0000 X:0000 Y:0000 P:envMXdiZC
$1D/C873 F0 46       BEQ $46    [$C8BB]      A:0000 X:0000 Y:0000 P:envMXdiZC
$1D/C875 C9 18       CMP #$18                A:001F X:0000 Y:0000 P:envMXdizC
$1D/C877 D0 41       BNE $41    [$C8BA]      A:001F X:0000 Y:0000 P:envMXdizC
-- event (shooting) occurs when this code does not branch --
$1D/C8BA 60          RTS                     A:001F X:0000 Y:0000 P:envMXdizC
```

In Parallel Worlds (so long ago, i wish i documented this before),
the timer is set to #$0C (from #$20), the listener gets kicked off at #$05 (from #$18)

So going by the above:

In the original game, when you swing the sword:
timer set to #$20
8 ticks later
shooting occurs #$18
24 ticks later before timer gets reset, so this acts as "cooldown" time.

In Parallel Worlds, when you swing the sword:
timer set to #$0C
7 ticks later

shooting occurs #$05
5 ticks later before timer gets reset, so this acts as "cooldown" time.

I do not think you can set both values to be the same, so the cruellest shooter will have the timer at 02 and shooting occur at 01.

Parallel Worlds - Random shooter interval

Euclid 6/04/2012

The goal
Copy and paste the random shooter interval code from PW.

Should just be the following snippet code asm i have in my stash of notes:

```
.code
07:FA50:

LDA $0DF0,x  ; check timer of shooter till next shot
BEQ start    ; if its 0 then jump to "find next shoot" routine
DEC          ;
BNE else     ; if it's 1 then go down into the SEC to shoot. (yes it shoots at 1 not 0)
SEC          ; else go to the CLC code and return.
RTL

start:
LDA $0E80,x  ; load recipe's own timer
ADC $1A      ; throw the frame counter AKA RNG into the mix
ADC $0D00,x  ; throw low byte of x/y positions into the mix
ADC $0D10,x
AND #$FC     ; take high 5 bits to be the next shoot interval
             ; can take more/less bits for more/less shooting
             ; or lower bits for more shooting
STA $0DF0,x  ; store as the next shoot interval countdown value

else:
CLC          ; CLC to make it not shoot now, then return.
RTL
.end


.ORG $1DC831
JSL.l $07FA50    ; 5 bytes to fill LDA something DEC something.
NOP              ;
.dc.b $90,$1A    ; hardcoded BCC (because stupid assembler won't let me overwrite
                 ; BCC $1A here since $1A refers to nowhere.
.end
```

No sword beam unless lvl 4 sword

$07/9C8A C9 02 CMP #$02
change to
$07/9C8A C9 02 CMP #$04

4th sword = master sword to pull out

hex changes

$09/87C8 C0 01 CPY #$01
to CPY #03
$09/8884 -> CMP #$03

Intro rooms

02C8F9 - Zelda prison.
02C8FD (14AFD) - Agahnim casting spells room.

Kills annoying telepathic message

$07/F498 - change from SEC to CLC (0x18)

SFX KILLER:

```
$0D/DD99 A9 11      LDA #$11
$0D/DD9B 8D 2F 01    STA $012F
```

$12F is the sfx
Change all above bytes to NOP (EA)

Ponds

Note the item numbers (for the LDA instructions) are the same as the item numbers in HM's chests.

```
$06/C8C6 to $06/C948

$06/C8C6 FE 80 0D    INC $0D80,x[$00:0D80]
$06/C8C9 AF CA F3 7E LDA $7EF3CA[$7E:F3CA]
$06/C8CD D0 35      BNE $35    [$C904]

$06/C8CF BD C0 0D    LDA $0DC0,x[$00:0DC0]

$06/C8D2 C9 0C      CMP #$0C              ; CHECK boomerang
$06/C8D4 D0 0C      BNE $0C    [$C8E2]
$06/C8D6 A9 2A      LDA #$2A              ; GIVE red boomerang
$06/C8D8 9D C0 0D    STA $0DC0,x[$00:0DC0]
$06/C8DB A9 01      LDA #$01              ; MSG 142 + lda number
$06/C8DD 9D B0 0E    STA $0EB0,x[$00:0EB0]
$06/C8E0 80 5E      BRA $5E    [$C940]

$06/C8E2 C9 04      CMP #$04              ; CHECK regular Shield
$06/C8E4 D0 0C      BNE $0C    [$C8F2]
$06/C8E6 A9 05      LDA #$05              ; GIVE red shield
$06/C8E8 9D C0 0D    STA $0DC0,x[$00:0DC0]
$06/C8EB A9 02      LDA #$02
$06/C8ED 9D B0 0E    STA $0EB0,x[$00:0EB0]
$06/C8F0 80 4E      BRA $4E    [$C940]

$06/C8F2 C9 16      CMP #$16              ; CHECK bottle
$06/C8F4 D0 0C      BNE $0C    [$C902]
$06/C8F6 A9 2D      LDA #$2D              ; GIVE magic potion
$06/C8F8 9D C0 0D    STA $0DC0,x[$00:0DC0]
$06/C8FB A9 03      LDA #$03
$06/C8FD 9D B0 0E    STA $0EB0,x[$00:0EB0]
$06/C900 80 3E      BRA $3E    [$C940]

$06/C902 80 45      BRA $45    [$C949]    ; not useful?
```

```
$06/C904 BD C0 0D    LDA $0DC0,x[$00:0DC0]
$06/C907 C9 3A      CMP #$3A              ; CHECK bow and arrows
$06/C909 D0 13      BNE $13    [$C91E]
$06/C90B A9 3B      LDA #$3B              ; GIVE silver bow and arrows
$06/C90D 9D C0 0D    STA $0DC0,x[$00:0DC0]
$06/C910 A9 04      LDA #$04
$06/C912 9D B0 0E    STA $0EB0,x[$00:0EB0]
$06/C915 A9 4F      LDA #$4F
$06/C917 A0 01      LDY #$01
$06/C919 22 19 E2 05 JSL $05E219[$05:E219]  ; message 14F
$06/C91D 60          RTS

$06/C91E C9 02      CMP #$02              ; CHECK sword 3
$06/C920 D0 0C      BNE $0C    [$C92E]
$06/C922 A9 03      LDA #$03              ; GIVE sword 4
$06/C924 9D C0 0D    STA $0DC0,x[$00:0DC0]
$06/C927 A9 05      LDA #$05
$06/C929 9D B0 0E    STA $0EB0,x[$00:0EB0]
$06/C92C 80 12      BRA $12    [$C940]

$06/C92E C9 16      CMP #$16              ; CHECK bottle
$06/C930 D0 0C      BNE $0C    [$C93E]
$06/C932 A9 2B      LDA #$2B              ; GIVE magic potion
$06/C934 9D C0 0D    STA $0DC0,x[$00:0DC0]
$06/C937 A9 03      LDA #$03
$06/C939 9D B0 0E    STA $0EB0,x[$00:0EB0]
$06/C93C 80 02      BRA $02    [$C940]

$06/C93E 80 09      BRA $09    [$C949]    ; not useful?

$06/C940 A9 8C      LDA #$8C
$06/C942 A0 00      LDY #$00              ; message 8C (generic "good" item message)
$06/C944 22 19 E2 05 JSL $05E219[$05:E219]
$06/C948 60          RTS
```

Remove warp effect after using mirror

Prevent the STA 7B from happening via NOPing it.

```
$07/A97E A5 8A      LDA $8A    [$00:008A]  A:0209 X:0026 Y:0013 P:eNvMXdizc
$07/A980 29 40      AND #$40                A:0244 X:0026 Y:0013 P:envMXdizc
$07/A982 85 7B      STA $7B    [$00:007B]  A:0240 X:0026 Y:0013 P:envMXdizc
$07/A984 F0 14      BEQ $14    [$A99A]      A:0240 X:0026 Y:0013 P:envMXdizc
```

No heart drop from boss

```
; FSNASM module by Euclid - 16/10/2006
; - makes boss drop nothing no matter what happens
;  but makes crystal drop if the room is tagged with clear level room header
; - side effects:
;  - boss drop nothing, always,
;  - boss will never revive itself even with kill boss again header

.ORG $9EE4F
LDA #$DA      ; give out 5 rupees just in case this patch fails
.end

.code
somewhere:
LDA $0403    ; the code
ORA #$80
STA $0403
```

```
JSL $1DF65F  ; replacement code
RTL
.end


.ORG $9EE53
JSL somewhere ; overwrote a JSL
.end
```

Damage taken by Link

This loads the damage for an enemy/item/object/whatever which causes you to get hurt.

```
$06/F400 B9 27 F4 LDA $F427,y[$06: F436]
$06/F403 8D 73 03 STA $0373 [$06: 0373]
```

so for those who don't understand this, 06:F427 of the memory, which works out to be 0x37627 hex in the rom, is the start of the damage table.

For that example above, i used that jumpy flower looking enemy in the dark world (it's the easiest to access for me), and the rom addr for it is 0x37636, so if you had a negative value (<= 0x80), that'll mean you'll gain health from the hit.

Using the above values, I'll write a simple guide to how to use that debugger to find the offset you want.

1. first get the debugger, load and play the game.
2. go somewhere close to the enemy, go the tools->state inspector.
3. in the 65816 tab (the first one), click breakpoint and type "06F400" and press add (this adds a breakpoint which stops when that instruction is executed, i.e. when you get hit it'll stop), then click ok.
4. now press run and get hurt by that enemy which you want to know the value of.
5. a box should pop up (if it doesn't, that might mean the enemy is somewhat special and probably have to find the data by other means)
6. in that box, you should see that "06F400 lda $F427,y" in the top line, now look at the box on the right, with A, X, Y, etc values (they're the register values) besides Y, you should see a hex number, note that.
7. use that hex number and add it to 0x37627 in a hex calculator (windows calculator has hex addition)
8. there you go, the result.
9. repeat from 4 if you want to know the value for another enemy.

I think it's precise enough for any asm newbie to follow.

Enemy health data

Code

```
$0D/B81C: <--- sprite data loading routine.
... some loading code
$0D/B829 B9 73 B1 LDA $B173,y[$0D: B195] //load enemy health
$0D/B82C 9D 50 0E STA $0E50,x[$0D: 0E5D] //store in "temp array"
... some loading code
```

the array size is 0x100, so you probably can't have >512 sprites in one screen (not likely anyway)

Sprite health data - 0x6B373 - 0x6B64C rom addrs.

for this test i used that flower enemy again, and it's 0x6B395.

Can't really write a "how to find an enemy's health" on this one, it's... well.. takes too long

okay just very brief, basically the same as before but instead of putting 06F400 you put 0DB829, and this triggers when the enemy is close to entering the screen (so like, for overworld enemy, make a screen with nobody in it, enter it, then before you move close to the enemy you want, set the breakpoint.)

And of course, you'll be adding to 0x6B373 with y instead.

36F33 - Damage ptr of weapons - 4 bytes (also determines which enemies you can kill with, and which enemies will be one-hit-ko)
6BAF1 - Damage table of weapons (including items like boomerang, which is just 0)

6BAFA - sword 1 damage
6BB02 - sword 1 spin damage/sword 2 damage
6BB0A - sword 2 spin damage/sword 3 damage
6BB12 - sword 3 spin damage/sword 4 damage
6BB1A - sword 4 spin damage

try not to put anything >=0x80 in those, because that way you won't be able to kill the enemy.

That damage routine translates to:

Code
```
{
temp = enemyId & 0xF (use only last 4 bits)
temp = temp << 1 + temp
Y = temp + armor strength
damage offset = 0x37627 + Y
}
```

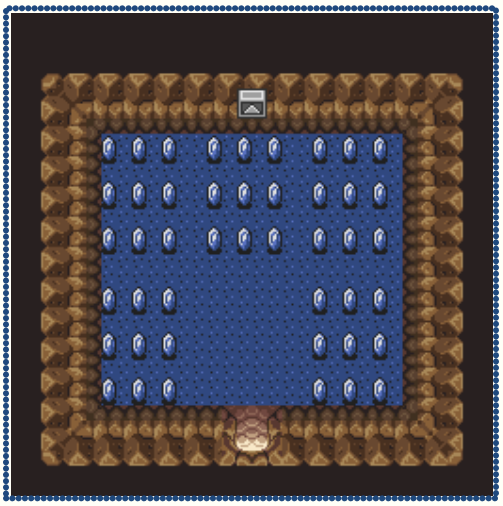
so an enemy id of 0x11 and 0x21 will produce the same result... And you'll be limited on using *high* amount values due to the fact that the armor strength (0,1,2) will not add up correctly if enemy id is like 1 and 0.

ok time for the "Real" enemy damage data (I'm sure this is it)

```
#just before here, you've seen it load the enemy health
#load enemy damage data.
$0D/B841 B9 66 B2 LDA $B266,y[$0D: B288]
# NOTE: this is adding 0x22, and *coincidentally* seph3's HM also says that jumping enemy
is 22, that just saved us tons of work because seph3's done it for us
$0D/B844 9D D2 0C STA $0CD2,x[$0D: 0CDF]
```

so this means, there's a table of damages, and there's a table of enemy damage pointers which points to positions in that table.

The Chris Hoolihan room



Yep the big secret, it took a while... but while trying to play around with code to stop the... err what was it again.... that annoying beginning message which pops up once in a while, I found this.

The room itself is a debug room for holes, surely everyone knows.

When you fall down a hole, the game checks for the X and Y positions and using some very specific calculations

Formula = ((((X & 0xFFF8) / 8) - 0xC0) & 0x7E) + ((((Y & 0xFFF8) - 0x600) & 0x3F0) * 8)

It finds out some number and it compares that number with a bunch of numbers which contains where the holes supposed to be. When it doesn't find the exact match, it'll go and load the Chris Hoolihan room.

The code for loading hole routine is at **$1B/B865** (if you're interested).

The game made an Assumption to the places where you fall down a hole, that's why sometimes you can pull that glitch off (start in sanctuary, run to Hyrule castle hole or even to the village hole).

The assumption they put on the player is they assume the **X** and **Y** positions are always correct when you exit a room.

Surely that's a valid assumption but you need to know when you exit a room, you tend to be placed a bit below where the exit supposed to be (as link kind of "walks out"), that little bit of difference is what caused some people to be able to access the room (since the **X** and **Y** is still pointed at the "white dots" in HM, if you look at it in the editor, and Link surely isn't at that pos when he comes out of the room)

When is that little difference fixed?
1. when you get hit
2. when you move, not run, i mean move up (so Link stays in the middle of the screen, instead of just a lil bit below it)
3. when you bounce off something when running

There's heaps of other factors as well which fixes that little bit of difference.

For the technical bunch, all those above event causes a change in a scroll value **$E8**, that's usually the culprit when it comes to getting to that room, if you really want to get in, you can try setting **$E8** to some weird value like **55** before you jump down a hole (yes the screen will look weird after you set it), you'll definitely be at the room since link is no longer in the middle of the screen.

Hope this helps for the hackers which wants to do something about the room.

Bunny Link Fix

Fix: Fixing the glitch where a bunny Link will always come back as normal Link after dying in any dungeon. (Specifically a problem in the Dark World.)

```
lorom

; defines!

!WorldIndicator = $7EF3CA
!HasMoonPearl = $7EF357

; end of defines

org $07F1BA

; STZ $02E0 -> NOP NOP NOP
; Doing this prevents Link's graphic set from going back to his normal form. Instead,
; He'll respawn with the graphic set he had last.

       NOP : NOP : NOP

org $0BFFE8

; LDA #$00; STA $5D -> JSL to our hook
; I USED JSL $07FFF0

       JSL $07FF00

org $07FF00

  LDA !WorldIndicator
  AND #$40 ; EXAMINE WHETHER WE ARE IN THE DARK WORLD OR NOT.

  BNE BRANCH_IN_THE_DARK_WORLD

BRANCH_HAS_A_MOON_PEARL:

  LDA #$00    ; WE'RE IN THE LIGHT WORLD AND LINK SHOULD COME BACK NORMALLY
  STA $5D

  RTL

BRANCH_IN_THE_DARK_WORLD:

  LDA !HasMoonPearl ; We're in the dark world, but do we have a moon pearl?

  BNE BRANCH_HAS_A_MOON_PEARL;

  LDA #$17
  STA $5D           ; Enter Permabunny mode

  LDA #$01
  STA $02E0         ; Change his tile set to that of a bunny

  JSL $0ED6DD; Initiate the bunny transformation
```

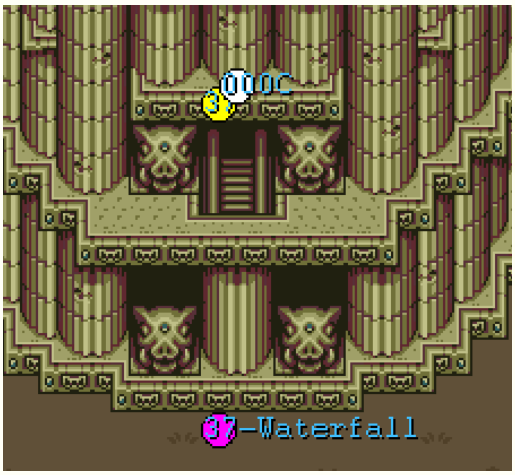
```
  RTL

org $09F528

  ; Changes the annoying LDA #$00 : STA $7EF3CA to NOPs
  ; The original code was dumb. If for some reason you die in the dark world
  ; you should be warped into the light world? How much sense does that make?

  NOP #$6

; End of patch
```

Moving the Weathervane / Bird sequence

```
; Originally meant for the Shards Of Might hack by Omega & SePH.

!wvArea = $30;
!wvSram = $7EF280+!wvArea;

org $07A425

      dw $0030     ; Use Area $0030 with the weather vane.

org $07A42C

      dw $0CB8     ; Lower limit to Y coordinates that trigger the weathervane.

org $07A431

      dw $0CF8     ; Upper limit to Y coordinates that trigger the weathervane.

org $07A438

      dw $02E0     ; Lower limit to X coordinates that trigger the weathervane.

org $07A43D

      dw $0320     ; Upper limit to X coordinates that trigger the weathervane.

org $098D65

      db $0C ; Upper byte of pieces' Y coords.

org $098CED

      db $F0, $F0, $F0, $F0, $F0, $F0, $F0, $F0, $F0, $F0, $F0, $F0

      ; These are the lower bytes of the weathervane pieces
      ; (or shards) Y coordinates.
      ; In the original they weren't set all the same, but here they
      ; are for simplicity
      ; I haven't messed with the lower bytes of the X coordinates.

org $098D72

      db $03 ; Upper byte of pieces' X coords.

org $1BC226

      dw $065E     ; Changing the place where replacement tiles are drawn
```

```
org $1BC232

       dw $0662      ; Change where the weathervane's propeller gets overwritten.

org $1BC243

       dw $06E0      ; And also change the upper left 16x16 tile as well.

org $098DC1

       LDA #$0CD0    ; Sets the Birds initial coordinates.
                     ; Important to make it work right.
       STA $00

       LDA #$0300
       STA $02

org $07A449

       LDA #$37    ; insurance plan for bug that keep showing up.
                   ; Makes sure the weather vane explodes

org $1BC255

       LDA !wvSram : ORA #$20 : STA !wvSram
```

Moving Ganon's Tower and its entrance animation




```
; Originally meant for the Shards Of Might hack by Omega & SePH.


!newTowerArea = $44 ; this is the area to change it to
!newSramAddr = $7EF280+!newTowerArea

org $1AF5C3

       CMP #!newTowerArea

org $099B99

       LDA !newSramAddr

; Enable lightning and tower glowing
```

```
org $0EF587

       LDA $8A : CMP #!newTowerArea

org $0EF622

       LDA $8A : CMP #!newTowerArea

org $0EF62C

       LDA !newSramAddr

; Disable it after the tower opens

org $0EFAA4
{
       ; Creating a new overlay handler (It just shifts the graphics
       ; over to the left side of the screen
       ; This routine is what is drawn when you reenter the screen
       ;(not necessarily the same as what you see when Ganon's tower is opened)
       ; We have to make sure they are the same for coherence

     LDA #$0E96 : STA $7E241E : INC A : STA $7E2420

     LDA #$0E9C : STA $7E249E : STA $7E251E : INC A : STA $7E24A0 : STA $7E2520

     LDA #$0E9A : STA $7E259E : INC A : STA $7E25A0

     RTS
}

       ; The next section is a series of alternations to routines.
       ; The (*) symbols indicate code that was changed, everything else is
       ; copied from the rom. Basically I had to shift all the tile writes.
       ; To the left side of the screen. Ganon's Tower used to be on the right
       ; side of a wide screen
       ; Now that it's in the center of a small screen, we need to adjust those
       ; coordinates.

org $1BD1B1

       LDX #$059E; *

org $1BD17A

       LDX #$051E; *
       LDA #$0E9C;

       JSL $1BC97C;

       LDX #$0520; *
       LDA #$0E9D;

       JSR $C9DE;

       LDX #$059E; *
       LDA #$0EA0;

       JSR $C9DE;

       LDA #$0EA1;
       LDX #$05A0; *

org $1BD15A

       LDX #$051E; *
       LDA #$0E9A;

       JSL $1BC97C;
```

```
       LDX #$0520; *

org $1BD134

       LDX #$049E; *
       LDA #$0E9C;

       JSL $1BC97C;

       LDX #$04A0; *

org $1BD114

       LDX #$049E; *
       LDA #$0E9A;

       JSL $1BC97C;

       LDX #$04A0; *

org $1BD0EB

       LDX #$041E; *
       LDA #$0E9C;

       JSL $1BC97C;

       LDX #$0420; *
       LDA #$0E97;

       JSR $C9DE;

       LDX #$049E; *

org $1BD0A0

       LDX #$041E; *
       LDA #$0E92;

       JSL $1BC97C;

       LDX #$0420; *
       LDA #$0E93;

       JSR $C9DE;

       LDX #$049E; *
       LDA #$0E94;

       JSR $C9DE;

       LDA #$0E94;

       JSR $C9DE;

       LDX #$051E; *
```

Bosshearts v1.1

```
lorom

; Apply this patch to a HEADERLESS Zelda 3 ROM!

;*******************************************************
; Bosshearts v1.1 - By Reshaper256
;
;(specifically made for use in GameMakr24's Quest for Calatia)
;
; Original Release:   05/22/2005 - v1.0
;
; Version History:   05/27/2005 - v1.1
;
; Purpose:    To make it so every boss which holds a pendant or crystal
;              drops a heart piece instead of a heart container. When the
;              heart piece is collected, the pendant/crystal then falls into
;         the room.
;
; Directions:  To apply this patch you will need an SNES assembler. I
;              recommend Xkas, which can be found at
;              <http://byuu.cinnamonpirate.com> under utilities.
;
; Contact:    Email me at <jajomart@gmail.com> if you have any questions or
;              bugs to report.
;
;
;*******************************************************
; Bosshearts Part 1 - Heart Containers --> Heart Pieces
;
;*******************************************************

org $05A079

   db $EB    ; Changes the item dropped by the Armos Knights boss from a
      ; heart container to a heart piece.

org $05A538

     db $EB   ; Changes the item dropped by the Lanmolas boss from a heart
      ; container to a heart piece.

org $09EE50

   db $EB   ; Changes the item dropped by the Moldorm, King Helmasaur,
      ; Arrghus, Mothula, Blind the Thief, Kholdstare, Vitreous,
      ; and Trinexx bosses from a heart container to a heart piece.
      ; So yeah, this covers all the rest of the bosses that drop
      ; heart containers - I checked. :P
```

```
;*********************************************************
; Bosshearts Part 2 - Still Drop the Treasure
;
;*********************************************************

org $05F006

   db $C0, $A0   ; This change is important, because the game will
         ; not drop the pendants/crystals if don't tell the
         ; game to set the high bit of $0403 in RAM.
         ;
         ; You'd be stuck... but not anymore, cuz now they
         ; include the high bit so the treasure still drops.

org $08CA99

   NOP #$7    ; This code originally conflicted with some heart
         ; pieces setting bit 6 in $0403 when you picked
         ; them up, because for some reason the game would
         ; check that bit and decide it shouldn't drop the
         ; treasure.  I couldn't find the reason why the game
         ; originally made this check, so I just NOPed out
         ; the problem.  If anyone can find a bug this
         ; creates, please report it.


;*********************************************************
; Bosshearts Part 3 - Minibosses Don't Drop Heart Pieces
;
;*********************************************************

org $06C0A3

   JSL $07FD00   ; JSL to my hook to check if we're in a miniboss
         ; room, and kill the heart piece if we are.

org $07FD00

   PHP      ; Push the processor status onto the stack.

   REP #$20

   PHX      ; Push X onto the stack.

   LDX #$04   ; Want to add more minibosses to the game?  Subtract
         ; 1 from the total number of miniboss rooms in the
         ; game, multiply by 2, and stick that value here in
         ; the place of the #$04.  Then, go down to the
         ; MINIBOSSTABLE and add their room numbers to it.

BRANCH_MINIBOSSCHECK:

   LDA $A0         ; Load the current room number.

   CMP MINIBOSSTABLE, X   ; Check against current value in the
            ; MINIBOSSTABLE

   BEQ BRANCH_ROOMMATCH    ; Was it the same value?

   DEX : DEX      ; No, so get ready to check the next value
            ; on the list...
```

```
   BPL BRANCH_MINIBOSSCHECK   ; But was that the last room number
               ; we need to check against? If no,
               ; back to the beginning of the loop!
               ; If yes, we're done checking -
               ; we're not in a miniboss room!

   PLX      ; Pull X back off the stack.

   PLP      ; Pull the processor status back off the stack.

   JSL $05F018   ; JSL to the normal heart piece routine...

   RTL      ; And return from everything, we're done!

BRANCH_ROOMMATCH:    ; We're in a miniboss room...

   PLX      ; Pull X back off the stack

   PLP      ; Pull the processor status back off the stack

   STZ $0DD0, X   ; ...So we kill off the heart piece being dropped
         ; from the boss before the heart piece even
         ; appears...

   RTL      ; ...And return from everything, we're done!

MINIBOSSTABLE:

   dw $001C, $006C, $004D   ; These are just the room numbers of the
            ; minibosses. You can edit them, or even add
            ; more - just be sure to change the LDX #$04
            ; above also, as the instructions beside it
            ; explain.


; End of 'Bosshearts' patch
```

Directions: Pay attention to whether you are using the version for a headered or headerless ROM (I prefer headerless, but meh...)

To apply the ASM patch you will need an SNES assembler. I recommend Xkas, which can be found at under utilities.

If that scares you, you can just apply the IPS patch, with Lunar IPS. You can find it about anywhere. (Google!)

Contact: Email me at if you have any questions or bugs to report.

Known Issues:

- The heart piece seems to fly higher in the air than the heart container did before falling back to the ground. O_o

- There seems to be a slight delay in how long it takes the pendant or crystal to fall down after exiting the 'you just stepped on a

heart piece' message - hey, at least it falls.

- If you receive your 'fourth' heart piece, Link holds a full heart over his head. The pendant/crystal will appear, but not fall, and it will have heart container GFX until Link puts the full heart container he was holding away. Then the treasure falls down. Apparently they share the same GFX location. You won't even see how that the treasure looks like a heart unless this happens at the end of the Moldorm boss fight, because otherwise it will be out of sight at the top of the screen when its GFX is the same as the heart container in Link's hand.

- The pendants and crystals don't seem to be as easy to pick up after applying this patch. The collision isn't as forgiving, but you can still pick them up if you walk directly into them like you would any other item.

- The heart pieces don't have the 'standing in shallow water' GFX when they're in shallow water.

Edit Log:

v1.1 - I realized that it was possible to have the values in part 2 of the patch set both the original bits of $0403 and the high bit as well. This caused a problem with the pendant/crystal drop sequence, which I NOPed out. I also added a routine that fixes the problem with minibosses dropping heart pieces.

MinibossCheck v1.1

```
lorom

; Apply this patch to a HEADERLESS Zelda 3 ROM!

;*******************************************************
; MinibossCheck v1.1 - By Reshaper256
;
;(specifically made for micksplacep)
;
; Original Release:   06/19/2005 - v1.0
;
; Version History:   06/22/2005 - v1.1
;
; Purpose:    To make it so a hacker can place minibosses in any room
;              and define it in the list of rooms at the end of this patch
;              so that those bosses won't drop heart containers.
;
;         (This patch comes with the original miniboss rooms from an
;         unedited version of Zelda 3 already in the list of miniboss
;         rooms.  Please scroll down and edit the list if you need
;              to change them.  Define as many rooms as you want.)
```

```
;
; Directions:  To apply this patch you will need an SNES assembler. I
;              recommend Xkas, which can be found at
;              <http://byuu.cinnamonpirate.com> under utilities.
;
; Contact:    Email me at <jajomart@gmail.com> if you have any questions or
;              bugs to report.
;
;
;*******************************************************
; MinibossCheck Part 1 - NOPing the 'Ganon's Tower' Check
;
;*******************************************************

org $05EF47

   NOP #$B      ; This check would conflict with the one I'm
         ; putting in.  This is the code that caused
         ; minibosses in Ganon's Tower to not drop
         ; heart containers in the original game. Now
         ; it's gone, so just deal with it. :P


;*******************************************************
; MinibossCheck Part 2 - Defining Minibosses by Rooms
;
;*******************************************************

org $06C099

   JSL $07FD00   ; JSL to my hook to check if we're in a miniboss
         ; room, and kill the heart container if we are.

org $07FD00

   PHP      ; Push the processor status onto the stack.

   REP #$20

   PHX      ; Push X onto the stack.

   LDX #$04   ; Want more/less minibosses in the game?  Subtract   *IMPORTANT - READ IF
YOU NEED TO EDIT ROOMS*
         ; 1 from the total number of miniboss rooms in the
         ; game, multiply by 2, and stick that value here in
         ; the place of the #$04.  Then, go down to the
         ; MINIBOSSTABLE and add their room numbers to it.
         ;
         ; EX. We have 3 rooms in the list, so...
         ;
         ;    3 - 1 = 2
         ;    2 x 2 = 4
         ;   4 =  #$04 (convert to hex)
         ;
         ; EX. But what if you have 10 rooms in your list?
         ;
         ;   10 - 1 =  9
         ;    9 x 2 = 18
         ;   18 =  #$12 (convert to hex)
         ;
         ; EX. And what if you have 1 room in your list?
         ;
```

```
        ;   1 - 1 = 0
        ;   0 x 2 = 0
        ;   0 =  #$00 (convert to hex)

BRANCH_MINIBOSSCHECK:

   LDA $A0         ; Load the current room number.

   CMP MINIBOSSTABLE, X   ; Check against current value in the
            ; MINIBOSSTABLE

   BEQ BRANCH_ROOMMATCH    ; Was it the same value?

   DEX : DEX      ; No, so get ready to check the next value
            ; on the list...

   BPL BRANCH_MINIBOSSCHECK   ; But was that the last room number
               ; we need to check against? If no,
               ; back to the beginning of the loop!
               ; If yes, we're done checking -
               ; we're not in a miniboss room!

   PLX      ; Pull X back off the stack.

   PLP      ; Pull the processor status back off the stack.

   JSL $05EF3F   ; JSL to the normal heart container routine...

   RTL      ; And return from everything, we're done!

BRANCH_ROOMMATCH:    ; We're in a miniboss room...

   PLX      ; Pull X back off the stack

   PLP      ; Pull the processor status back off the stack

   STZ $0DD0, X   ; ...So we kill off the heart container being dropped
         ; from the boss before the heart container even
         ; appears...

   STZ $0FFC   ; ...Make sure to turn the menus back on...

   RTL      ; ...And return from everything, we're done!

MINIBOSSTABLE:

   dw $001C, $006C, $004D   ; These are just the room numbers of the   *IMPORTANT - READ
IF YOU NEED TO EDIT ROOMS*
            ; minibosses. You can edit them, or have more
            ; or less than 3 - just be sure to change the
            ; LDX #$04 above also, *as the instructions
            ; beside it explain* or the game won't know
            ; how many rooms are in the list to start with.
            ;
            ; You can find the room numbers in Hyrule
            ; Magic, but you'll have to convert them to
            ; hex.  Just use the windows calculator.

; End of 'MinibossCheck' patch
```

Edit Log:

v1.1 - After being informed that after defeating a miniboss a player could be locked out of all the menus, I began looking for a a way to fix it. Euclid informed me that the Kill Boss Again tag (HM name) was required for this not to happen, so I looked into the code to determine how it did this. What it did was store a zero to $0FFC, and this caused the menus to work again. I just added this to my own code, so that we don't even have to mess with the Kill Boss Again tag anymore.

- ASM Hacks by XaserLE -

Overworlds overlays ASM

In the following zip file is an asm-patch, xkas.exe and all you need to try it. Only download it, extract, place a copy of your Zelda3(U).smc in this directory and then execute the overlay.bat. Use ZSNES to play and load state 0.

You can see the overlay when MasterSword is pulled, but not in the forest, in area 02 instead. If you go in the forest (Area 00), there is rain and if you go down there is a new blank-out and the forest-overlay (Area 0A).

Only the rain sound is missing because this can be done in Hyrule magic.

```
;this is for getting the overlays work in certain areas, i use bank 0x3F for the whole
code
;WRITTEN:    by XaserLE
;THANKS TO: -MathOnNapkins
;          -an unknown author of a rain activation code that helped a little
;TODO:
```

```
;         -missing: silent rain sound in dungeons. for that we need to save a rain
indicator in free ram and hook into the dungeon load routine
;         -rain doesn't work with overworld warping (teleporter/mirror)

;header
lorom

;THIS IS FOR KEEPING AN OVERLAY APPEARED WHEN GOING FROM ONE AREA TO ANOTHER WITHOUT THE
BLANK OUT
ORG $02AC18      ; go to the code that makes sure the overlay keeps appeared in
transition from overworld area to overworld area
BRA $00          ; overwrites an "BCS $03" that won't keep the overlay appeared if we are
not in the beginning mode
            ; only at the beginning an overlay keeps appeared when going from one area to
another without a blank out

;THIS IS THE OVERLAY-CODE
            ; IMPORTANT: If the overlay is drawn correctly depends on the GFX# in this
area (look at Hyrule magic).
            ;       Rain seems to work various GFX#'s, but clouds for example on 33, 47
and maybe more.

ORG $02AFA3      ; go to beginning of the overlay code (command 293F00)

JSL $3F9180      ; overwrite this with a long jump to expanded space
BRA $62          ; after long jump we return and branch after the piece of code that loads
the rain overlay (command A29F00)

ORG $3F9180      ; go to expanded space

LDA $7EF3C5    ; load game state
AND #$00FF      ;
CMP #$0002      ; test for beginning
BCS $04          ; if not beginning, jump to next test
LDX #$009F      ;
RTL            ;

LDA $8A        ; load actual area

CMP #$0000      ;
BNE $04         ;
LDX #$009F      ;
RTL            ;

CMP #$0002      ;
BNE $04         ;
LDX #$009E      ;
RTL            ;

LDA $8A        ; load actual area again

;for paste©
;CMP #$0003      ;
;BNE $04         ;
;LDX #$0095      ;
;RTL            ;

CMP #$0003      ;
BNE $04         ;
LDX #$0095      ;
RTL            ;
```

```
CMP #$0005      ;
BNE $04         ;
LDX #$0095      ;
RTL            ;

CMP #$0007      ;
BNE $04         ;
LDX #$0095      ;
RTL            ;

CMP #$000A      ;
BNE $04         ;
LDX #$009D      ;
RTL            ;

CMP #$0040      ;
BNE $04         ;
LDX #$009D      ;
RTL            ;

CMP #$0043      ;
BNE $04         ;
LDX #$009C      ;
RTL            ;

CMP #$0045      ;
BNE $04         ;
LDX #$009C      ;
RTL            ;

CMP #$0047      ;
BNE $04         ;
LDX #$009C      ;
RTL            ;

;dark world evil swamp, so test for evil-swamp-dungeon-is-opened is needed
CMP #$0070      ;
BNE $0D         ;
LDA $7EF2F0    ; test if evil swamp dungeon already opened
AND #$0020      ;
BNE $03          ; if yes, don't load the rain overlay
LDX #$009F      ;
RTL            ;

LDA $8A        ; load actual area again

CMP #$005B      ;
BNE $04         ;
LDX #$0096      ;
RTL            ;

;no match
RTL            ;

;Between the above and the next code there should be enough space to test every area or
to test for self-defined events
;in some areas. If you still need more, you need to move it more forward.

;THIS IS THE BLANK-OUT-CODE

ORG $02AADB      ; go to beginning of the overlay code (command 293F)
```

```
JSL $3F9700      ; overwrite this with a long jump to expanded space
CMP #$01      ; in the new routine i will set the accumulator to 0x01 to indicate that we
used the blank out, otherwise 0x00
BEQ $02         ; so if we want to use the blank out, branch to this routine in the
original code
BRA $0F         ; we don't want to use the blank out, jump to the routine after the RTS
in the original code

ORG $3F9700      ; go to expanded space
            ; IMPORTANT: accumulator holds the area that we come from, NOT the actual,
the game did this
            ; if want a blank out in an area, you need this when going in AND out

;test where we COME FROM

;for paste©
;CMP #$40      ;
;BNE $03         ;
;LDA #$01      ;
;RTL

CMP #$00      ; test if we come from this area
BNE $03         ; if not, jump to next area
LDA #$01      ; set blank out indicator
RTL

CMP #$02      ; test if we come from this area
BNE $03         ; if not, jump to next area
LDA #$01      ; set blank out indicator
RTL

CMP #$0A      ; test if we go in this area
BNE $03         ; if not, jump to next area
LDA #$01      ; set blank out indicator
RTL

CMP #$40      ;
BNE $03         ;
LDA #$01      ;
RTL

;test where we GO TO
LDA $8A         ; load ACTUAL area

;for paste©
;CMP #$40      ;
;BNE $03         ;
;LDA #$01      ;
;RTL

CMP #$00      ; test if we come from this area
BNE $03         ; if not, jump to next area
LDA #$01      ; set blank out indicator
RTL

CMP #$02      ; test if we come from this area
BNE $03         ; if not, jump to next area
LDA #$01      ; set blank out indicator
RTL

CMP #$0A      ; test if we go in this area
BNE $03         ; if not, jump to next area
```

```
LDA #$01      ; set blank out indicator
RTL

CMP #$40      ;
BNE $03         ;
LDA #$01      ;
RTL

;no match
LDA #$00      ; unset blank out indicator
RTL

;Between the above and the next code there should be enough space to test every area or
to test for self-defined events
;in some areas. If you still need more, you need to move it more forward.

;THIS IS THE RAIN-ANIMATION-CODE

;-------------------------------------------------------------------------------------
-------------------
;first of all a little helper: if you don't want this flash lights during the rain,
uncomment the next lines
;ORG $02A4E7
;BEQ $14
;ORG $02A4F7
;BEQ $04
;ORG $02A506
;BRA $02
;if want to set back to normal, here is the original code, uncomment it and out-comment
the modified above
;ORG $02A4E7
;BEQ $1D
;ORG $02A4F7
;BEQ $0D
;ORG $02A506
;LDA #$32
;-------------------------------------------------------------------------------------
-------------------

ORG $02A4CD      ; go to beginning of the rain animation code (command A58A)

JSL $3F9C00      ; overwrite this with a long jump to expanded space
CMP #$01      ; in the new routine i will set the accumulator to 0x01 to indicate that we
used the rain animation, otherwise 0x00
BEQ $0E         ; so if we want to use the rain animation, branch to this routine in the
original code
BRA $55         ; we don't want to use the rain animation, jump to the RTL in the
original code

ORG $3F9C00      ; go to expanded space

LDA $7EF3C5    ; load game state
CMP #$02       ; test for beginning
BCS $03          ; if not beginning, jump to next test
LDA #$01      ; set rain animation indicator
RTL

LDA $8A        ; load ACTUAL area

;for paste©
;CMP #$00      ; test if we are in this area
;BNE $03         ; if not, jump to next area
```

```
;LDA #$01      ; set rain animation indicator
;RTL

CMP #$00      ; test if we are in this area
BNE $03         ; if not, jump to next area
LDA #$01      ; set rain animation indicator
RTL

;dark world evil swamp, so test for evil-swamp-dungeon-is-opened is needed
CMP #$70
BNE $0E
LDA $7EF2F0      ; test if evil swamp dungeon already opened
AND #$20        ; if yes, don't animate the rain
BNE $03
LDA #$01      ; set rain animation indicator
RTL
LDA #$00      ; unset rain animation indicator
RTL

LDA $8A        ; load ACTUAL area again

;no match
LDA #$00      ; unset rain animation indicator
RTL

;Between the above and the next code there should be enough space to test every area or
to test for self-defined events
;in some areas. If you still need more, you need to move it more forward.

;THIS IS THE MOUNTAIN-FLASHLIGHTS-CODE

ORG $0EF587      ; go to beginning of the flashlights code (command A58A)

JSL $3FA100      ; overwrite this with a long jump to expanded space
CMP #$01      ; in the new routine i will set the accumulator to 0x01 to indicate that we
used the flashlights, otherwise 0x00
BEQ $06         ; so if we want to use the flashlights, branch to this routine in the
original code
BRA $02         ; we don't want to use the flashlights, jump to the branch to the RTL in
the original code

ORG $3FA100      ; go to expanded space

LDA $8A        ; load ACTUAL area

;for paste©
;CMP #$45      ;
;BNE $03         ;
;LDA #$01      ;
;RTL

CMP #$43      ; test if we are in this area
BNE $03         ; if not, jump to next area
LDA #$01      ; set flashlights indicator
RTL

CMP #$45      ;
BNE $03         ;
LDA #$01      ;
RTL

CMP #$47      ;
```

```
BNE $03          ;
LDA #$01      ;
RTL

;no match
LDA #$00      ; unset flashlights indicator
RTL
```


Changing the default speed values for different conditions

Convert to hex and send in: various info u can change in hex

There's a table in **Bank 7** (**$07:E227)** that has all speed values for the different conditions (Normal, stairs, **slow-ICE?, slower- ICE?,when you have an object on top of your head,** running with boots etc) in it. Not only can you make Link faster with that (try a value of **#$70**), you can also make him moonwalk using whack values.

The table is located at hex address **3E227-3E244**.

3E227 - Horizontal and vertical walking speed (Default = 18)
3E228 - Diagonal walking speed (Default = 10)
3E229 - Stairs walking speed (Default = 0A)
3E22A - ?????????????????????? (Default = 18)
3E22B - ?????????????????????? (Default = 10)
3E22C - ?????????????????????? (Default = 08)

3E22D - ?????????????????????? (Default = 08)
3E22E - ?????????????????????? (Default = 04)
3E22F - ?????????????????????? (Default = 0C)
3E230 - ?????????????????????? (Default = 10)
3E231 - ?????????????????????? (Default = 09)
3E232 - ?????????????????????? (Default = 19)
3E233 - walking through heavy grass speed (also shallow water) (Default = 14)
3E234 - walking diagonally through heavy grass speed (also shallow water) (Default = 0D)
3E235 - ?????????????????????? (Default = 10)
3E236 - ?????????????????????? (Default = 08)
3E237 - Pegasus boots speed (Default = 40)
3E238 - ?????????????????????? (Default = 2A)
3E239 - ??????????????????????
3E23A - ??????????????????????
3E23B - ??????????????????????
3E23C - ??????????????????????
3E23D - ??????????????????????
3E23E - ??????????????????????
3E23F - ??????????????????????
3E240 - ??????????????????????
3E241 - ??????????????????????
3E242 - ??????????????????????
3E243 - ??????????????????????
3E244 - ??????????????????????

The bouncing distance changes as well if you edit the table so be careful while editing the values. Jump distances ought not to be changed. The game's not designed to allow that. You will inevitably end up in a wall or on top of something illegal.

Another problem is that if you make Link too fast, he tends to run through walls, because the collision detection does not prevent him from doing so.

The routine that controls how he moves is not too far away from that table. It's located at hex address **3E245**.

In a small hack that I've done, I've deleted an LSR out of a sequence of 4 LSR A's. That had the effect of doubling his speed. Remove another one and you get craziness!

Changing the hardcoded grass transparent color

The grass color is hardcoded and different depending on the current environment.
No matter how long you're going to search for the palette in the editor, you won't find it.

Let's look at the code:

```
ldx #$2669

;this writes to $7e:c500.

$2669 is the color value.

;this is where the normal light overworld color is loaded: SNESLO $0B/FEA9
```

```
$05:FEA9
$06:00A9

x05FEA9
x0600A9    A2 69 26
84 136 86

2918E0

#548856
4A784A

2DC856
585830
```

Just change it to any color you like.

Here's the code snippet for that:

```
$0B/FEA8 A2 69 26 LDX #$2669 A:002C X:0000 Y:00FF P:envmxdizC
$0B/FEAB A5 8A LDA $8A [$00:008A] A:002C X:2669 Y:00FF P:envmxdizC
$0B/FEAD 29 40 00 AND #$0040 A:002C X:2669 Y:00FF P:envmxdizC
$0B/FEB0 F0 03 BEQ $03 [$FEB5] A:0000 X:2669 Y:00FF P:envmxdiZC
$0B/FEB5 8A TXA A:0000 X:2669 Y:00FF P:envmxdiZC
$0B/FEB6 8F 00 C5 7E STA $7EC500[$7E:C500] A:2669 X:2669 Y:00FF P:envmxdizC
```

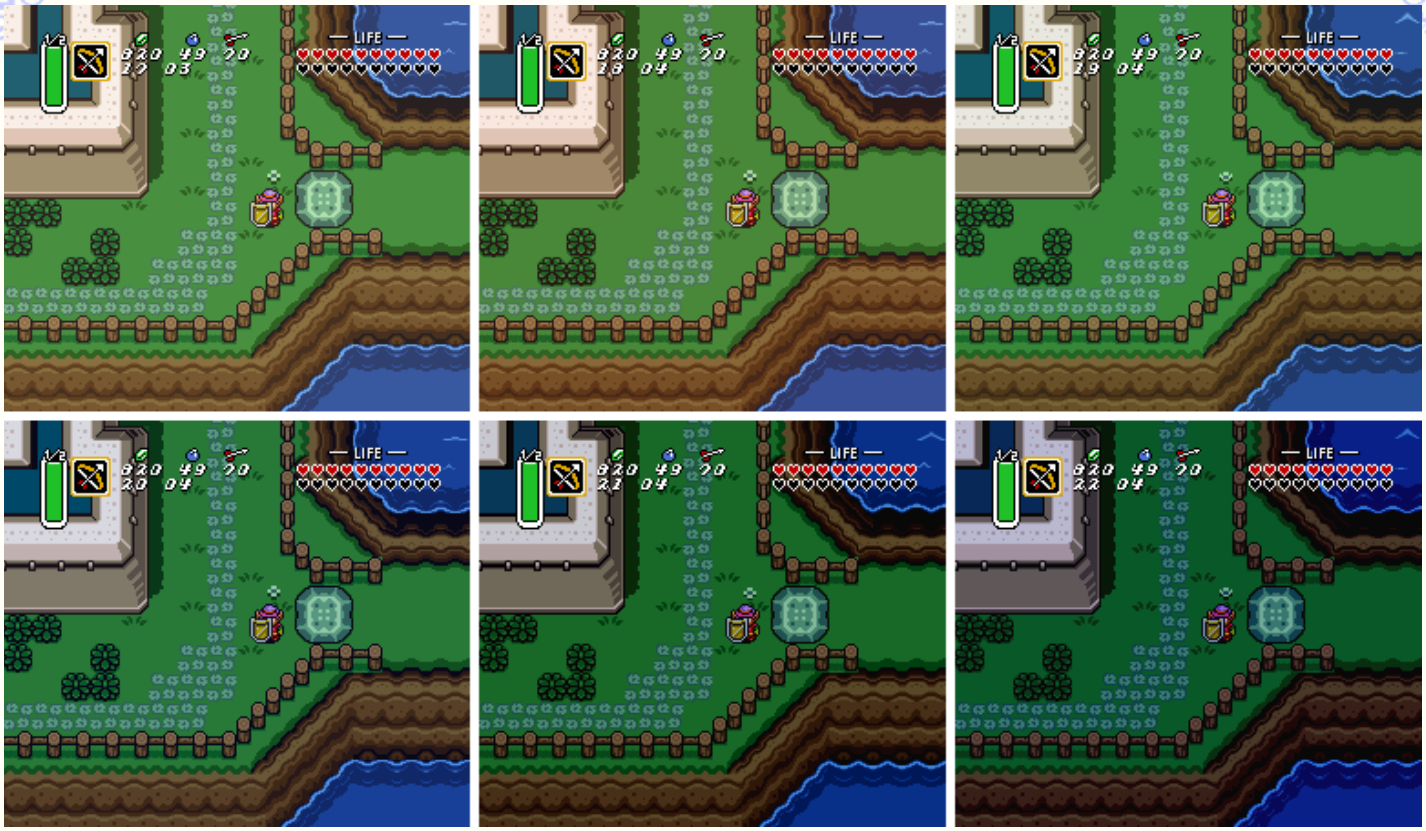
Added by Euclid

Dark World uses #$2A32
Light World uses #$2628

- ASM Hacks by JaSp -

Zelda3 Time + Day/Night System v2.1

This patch adds a time control (hours/minutes) to "Zelda 3 : A link to the Past", with a Day/Night handler using this new time feature.
A time indicator has been added to the status bar, showing the hours and minutes. The format is as 24-hour clock that means from 00:00 to 23:59.
The Day/Night works as follow : on each hour, the palette is updated and changed according to 3 tables telling how many units to subtract from the original palette color.
The color units to subtract are editable within the ASM source file.
The system is currently set so that 12 real-life minutes correspond to 24 in-game hours, according that the game runs at 60 fps (1 minute in the game = 0.5 real second).
This is also editable within the ASM patch.



This archive contains :
==========
*readme_z3time system.txt this readme file
*z3_time+daynight_source_v21.asm ***source asm file*** for use with an assembler on an ***"unheadered"*** Zelda3 ROM
*Zelda3 Time&DayNight_v21.ips ***ips patch*** to apply to a ***"headered"*** Zelda3 ROM
*ZeldaGFX.bin graphics file for use with ***zcompress*** containing the ***"Time-"*** graphics for HUD

Last update : September, 8 2006

History:
---
v2.1:
*fixed overworld map screens palette switching
v2.0:
*Major improvement: complete system re-write. The night effect works on the palettes now, and it works in all situations : layers, mosaic, HDMA warping, sub-areas...
v1.1:
*fixed the pause and save dialog box problems, time no longer increases when a dialog box is opened
v1.0:
*first release

```
lorom

;/!\ Patch designed for a HEADERLESS Zelda3 US ROM /!\
;-------------------------------------------------
; ZELDA 3 - Time + Day/Night handling code v2.1
;  by JaSp
```

```
;
;Last update: September, 8 2006
;
; This patch just adds a timer and a day/night
; handling system.
; Take a look at the readme for further details.
;
; If you have any questions or bug reports :
; jaspile(at)hotmail.com
;
;Thanks to
;  MathOnNapkins' "ZeldaFlow" document & Help
;  Euclid's status bar explanations & Help
;  d4s' Help
;  Geiger's Snes9x Debugger
;  FuSoYa's zcompress utility
;-----------------------------------------------


;-----------------------------------------------
;---------------[ Time system ]----------------
;-----------------------------------------------

; tiles locations on HUD (status bar)
!hud_min_low = $7EC79E
!hud_min_high = $7EC79C
!hud_hours_low = $7EC798
!hud_hours_high = $7EC796
!hud_template = $0DFF07

org !hud_template

      db $10,$24,$11,$24,$12,$24,$90,$24,$90,$24,$13,$24,$90,$24,$90,$24     ;HUD
Template(adjusts timer's color)

org $068361

      jsl $1CFF30       ;originally JSL $09B06E, executed every frame

org $1CFF30

      jsr counter_preroutine
      ldx #$00
debut:
      ldy #$00
      lda $7EE000,x
debut2:
      cmp #$0A
      bmi draw
      sbc #$0A
      iny
      bra debut2

draw:
      adc #$90
      cpx #$01
      beq minutes_low
      sta !hud_hours_low
      bra 04

minutes_low:
      sta !hud_min_low
```

```
      tya
      clc
      adc #$90
      cpx #$01
      beq minutes_high
      sta !hud_hours_high
      bra 04

minutes_high:
      sta !hud_min_high
      inx
      cpx #$02
      bmi debut
      jsl $09B06E
      rtl

;--------------------------------

counter_preroutine:
      lda $10                 ;checks current event in game
      cmp #$07          ;dungeon/building?
      beq counter_increasing
      cmp #$09          ;overworld?
      beq overworld
      cmp #$0B
      beq overworld           ;sub-area ? (under the bridge; zora domain...)
      cmp #$0E          ;dialog box?
      beq dialog
      rts

overworld:
      lda $11
      cmp #$23          ;hdma transfer? (warping)
      bne mosaic
mosaic:
      cmp #$0D          ;mosaic ?
      bmi counter_increasing
      rts

dialog:
      lda $11                 ;which kind of dialog? (to prevent the counter from
increasing if save menu or item menu openned)
      cmp #$02          ;NPC/signs speech
      beq counter_increasing
      rts

counter_increasing:
      lda $1A
      and #$1F          ;change value (1,3,5,7,F,1F,3F,7F,FF) to have different time
speed, #$3F is almost 1 sec = 1 game minute
      beq increase_minutes
end:
      rts

increase_minutes:
      lda $7EE001
      inc a
      sta $7EE001
      cmp #$3C          ; minutes = #60 ?
      bpl increase_hours
      rts
```

```
increase_hours:
      lda #$00
      sta $7EE001
      lda $7EE000
      inc a
      sta $7EE000
      cmp #$18          ; hours = #24 ?
      bpl reset_hours

      lda $1B                 ;check indoors/outdoors
      beq outdoors0
      rts
outdoors0:
      jsl rom_to_buff         ;update buffer palette
      jsl buff_to_eff         ;update effective palette
      lda $8C
      cmp #$9F          ;rain layer ?
      beq skip_bg_updt0
      jsl $0BFE70       ;update background color
      bra inc_hours_end

skip_bg_updt0:                ;prevent the sub layer from disappearing ($1D zeroed)
      jsl $0BFE72
inc_hours_end:
      rts

reset_hours:
      lda #$00
      sta $7EE000

      lda $1B                 ;check indoors/outdoors
      beq outdoors1
      rts
outdoors1:
      jsl rom_to_buff
      jsl buff_to_eff
      lda $8C
      cmp #$9F          ;rain layer ?
      beq skip_bg_updt1
      jsl $0BFE70       ;update background color
      bra reset_end

skip_bg_updt1:                ;prevent the sub layer from disappearing ($1D zeroed)
      jsl $0BFE72
reset_end:
      rts
;-----------------------------------------------
;----[ Day / Night system * palette effect ]----
;-----------------------------------------------

!blue_value = $7EE010
!green_value = $7EE012
!red_value = $7EE014

!temp_value = $7EE016
!pal_color = $7EE018
!x_reg = $08


org $02FF70       ; free space on bank $02

buff_to_eff:
```

```
      jsr $C769   ; $02:C65F -> palette buffer to effective routine
      rtl
rom_to_buff:
      jsr $AAF4   ; $02:AAF4 -> change buffer palette of trees,houses,rivers,etc.
      jsr $C692   ; $02:C692 -> rom to palette buffer for other colors
      rtl

; part of rom pal to buffer routine
;$1B/EF61 9F 00 C3 7E STA $7EC300,x[$7E:C422]
;$1B/EF3D 9F 00 C3 7E STA $7EC300,x[$7E:C412]
;$1B/EF84 9F 00 C3 7E STA $7EC300,x[$7E:C4B2]


org $1BEF3D
      jsl new_palette_load
org $1BEF61
      jsl new_palette_load
org $1BEF84
      jsl new_palette_load



org $0EEE25 ; free space

new_palette_load:

      sta !pal_color

      cpx #$0041
      bpl title_check
      sta $7EC300,x
      rtl
title_check:
      lda $10
      and #$00FF
      cmp #$0002  ; title or file select screen ?
      bpl outin_check
      lda !pal_color
      sta $7EC300,x
      rtl
outin_check:
      lda $1B
      and #$00FF
      beq outdoors2
      lda !pal_color
      sta $7EC300,x
      rtl

outdoors2:
      phx
      jsl color_sub_effect
      plx
      sta $7EC300,x
      rtl

;--------------------------------

color_sub_effect:
      lda $7EE000       ; lda #hours
      and #$00FF
      clc
      adc $7EE000       ; #hours * 2
```

```
      and #$00FF
      tax


do_blue:
      LDA !pal_color
      AND #$7C00
      STA !blue_value
      SEC
      SBC blue_table,x  ; substract amount to blue field based on a table
      STA !temp_value
      AND #$7C00        ; mask out everything except the blue bits
      CMP !temp_value         ; overflow ?
      BEQ no_blue_sign_change
blue_sign_change:
      LDA #$0400        ; lda smallest blue value
no_blue_sign_change:
      STA !blue_value

do_green:
      LDA !pal_color
      AND #$03E0
      STA !green_value
      SEC
      SBC green_table,x ; substract amount to blue field based on a table
      STA !temp_value
      AND #$03E0        ; mask out everything except the green bits
      CMP !temp_value         ; overflow ?
      BEQ no_green_sign_change
green_sign_change:
      LDA #$0020        ; lda smallest green value
      no_green_sign_change:
      STA !green_value

do_red:
      LDA !pal_color
      AND #$001F
      STA !red_value
      SEC
      SBC red_table,x         ; substract amount to red field based on a table
      STA !temp_value
      AND #$001F        ; mask out everything except the red bits
      CMP !temp_value         ; overflow ?
      BEQ no_red_sign_change
red_sign_change:
      LDA #$0001        ; lda smallest red value
no_red_sign_change:
      STA !red_value

      LDA !blue_value
      ORA !green_value
      ORA !red_value

      rtl


; color_sub_tables : 24 * 2 bytes each = 48 bytes (2 bytes = 1 color sub for each hour)


blue_table:
      dw $1000, $1000, $1000, $1000, $1000, $1000, $1000, $0800, $0000, $0000, $0000,
$0000, $0000, $0000, $0000, $0000, $0000, $0400, $0800, $0800, $0800, $1000, $1000, $1000
```

```
green_table:
      dw $0100, $0100, $0100, $0100, $0100, $00C0, $0080, $0040, $0000, $0000, $0000,
$0000, $0000, $0000, $0000, $0000, $0000, $0020, $0040, $0080, $00C0, $0100, $0100, $0100

red_table:
      dw $0008, $0008, $0008, $0008, $0008, $0006, $0004, $0002, $0000, $0000, $0000,
$0000, $0000, $0000, $0000, $0000, $0000, $0000, $0000, $0002, $0004, $0006, $0008, $0008


background_fix:
      beq no_effect           ;branch if A=#$0000 (transparent bg)
      jsl color_sub_effect
no_effect:
      sta $7EC500
      sta $7EC300
      sta $7EC540
      sta $7EC340
      rtl


subareas_fix:
      sta !pal_color
      phx
      jsl color_sub_effect
      plx
      STA $7EC300
      STA $7EC340
      rtl


thunder_fix:
      STA $7EC570,x
      LDA $F507,y
      sta !pal_color
      phx
      jsl color_sub_effect
      plx
      STA $7EC590,x           ; 17 bytes instead of 7
      LDA $F515,y
      sta !pal_color
      phx
      jsl color_sub_effect
      plx
      STA $7EC5E0,x           ; 17 bytes instead of 7
      LDA $F523,y
      sta !pal_color
      phx
      jsl color_sub_effect
      plx
      STA $7EC5F0,x           ; 17 bytes instead of 7
      rts


gloves_fix:
      sta !pal_color
      lda $1B
      and #$00FF
      beq outdoors3
      lda !pal_color
      STA $7EC4FA
      rtl
```

```
outdoors3:
      phx
      jsl color_sub_effect
      plx
      STA $7EC4FA
      rtl

map_screen_pal_tint:
      sta !pal_color
      phx
      jsl color_sub_effect
      plx
      sta $7EC500,x
      rtl



; $0BFE70 -> background color loading routine
;Background color write fix - 16 bytes
;$0B/FEB6 8F 00 C5 7E STA $7EC500
;$0B/FEBA 8F 00 C3 7E STA $7EC300
;$0B/FEBE 8F 40 C5 7E STA $7EC540
;$0B/FEC2 8F 40 C3 7E STA $7EC340

org $0BFEB6

      sta !pal_color
      jsl background_fix
      nop #8

; Subareas background color fix (under the bridge; zora...)
;$0E/D601 8F 00 C3 7E STA $7EC300[$7E:C300]
;$0E/D605 8F 40 C3 7E STA $7EC340[$7E:C340]

org $0ED601

      jsl subareas_fix

;--------------------------------

; Death Mountain thunder routine:
;$0E/F5F1 B9 EB F4    LDA $F4EB,y[$0E:F4EB]
;$0E/F5F4 9F 60 C5 7E STA $7EC560,x[$7E:C562]
;$0E/F5F8 B9 F9 F4    LDA $F4F9,y[$0E:F4F9]
;$0E/F5FB 9F 70 C5 7E STA $7EC570,x[$7E:C572]
;$0E/F5FF B9 07 F5    LDA $F507,y[$0E:F507]
;$0E/F602 9F 90 C5 7E STA $7EC590,x[$7E:C592]
;$0E/F606 B9 15 F5    LDA $F515,y[$0E:F515]
;$0E/F609 9F E0 C5 7E STA $7EC5E0,x[$7E:C5E2]
;$0E/F60D B9 23 F5    LDA $F523,y[$0E:F523]
;$0E/F610 9F F0 C5 7E STA $7EC5F0,x[$7E:C5F2]
;
; 35 bytes



org $0EF5F1

      LDA $F4EB,y
      sta !pal_color
      phx
```

```
      jsl color_sub_effect
      plx
      STA $7EC560,x
      LDA $F4F9,y
      sta !pal_color
      phx
      jsl color_sub_effect
      plx
      jsr thunder_fix         ; (jsr since it's from bank $0E)
      nop #2

;--------------------------------

; Gloves color loading routine
;$1B/EE1B C2 30       REP #$30
;$1B/EE1D AF 54 F3 7E LDA $7EF354[$7E:F354]
;$1B/EE21 29 FF 00    AND #$00FF
;$1B/EE24 F0 0F       BEQ $0F    [$EE35]
;$1B/EE26 3A          DEC A
;$1B/EE27 0A          ASL A
;$1B/EE28 AA          TAX
;$1B/EE29 BF F5 ED 1B LDA $1BEDF5,x[$1B:EDF7]
;$1B/EE2D 8F FA C4 7E STA $7EC4FA[$7E:C4FA]
;$1B/EE31 8F FA C6 7E STA $7EC6FA[$7E:C6FA]
;$1B/EE35 E2 30       SEP #$30
;$1B/EE37 E6 15       INC $15    [$00:0015]
;$1B/EE39 6B          RTL

org $1BEE2D

      jsl gloves_fix


;$0A/BA5A 9F 00 C5 7E STA $7EC500,x[$7E:C5FE]

org $0ABA5A

      jsl map_screen_pal_tint


;$0E/D745 9F 00 C3 7E STA $7EC300,x[$7E:C4B2]
;$0E/D749 9F 00 C5 7E STA $7EC500,x[$7E:C6B2]

org $0ED745

      jsl map_screen_pal_tint
      sta $7EC300,x
```

How to use it for my custom asm hacks ?
==========

RAM Addresses :
Hours $7EE000
Minutes $7EE001

How does the ASM source file work ?
==========

This file is for use with an assembler, I used xkas (http://byuu.cinnamonpirate.com/?page=utils) so it is fully compatible with this assembler.
The code is quite commented, though you may only want to just change the counter_increasing value (to have

faster/slower minutes) or the color_sub_tables.
Here is a little help for the units to substract with the tables :

val BLUE GREEN RED
---
00 0000 0000 0000
01 0400 0020 0001
02 0800 0040 0002
03 0C00 0060 0003
04 1000 0080 0004
05 1400 00A0 0005
06 1800 00C0 0006
07 1C00 00E0 0007
08 2000 0100 0008
09 2400 0120 0009
0A 2800 0140 000A
0B 2C00 0160 000B
0C 3000 0180 000C
0D 3400 01A0 000D
0E 3800 01C0 000E
0F 3C00 01E0 000F
10 4000 0200 0010
11 4400 0220 0011
12 4800 0240 0012
13 4C00 0260 0013
14 5000 0280 0014
15 5400 02A0 0015
16 5800 02C0 0016
17 5C00 02E0 0017
18 6000 0300 0018
19 6400 0320 0019
1A 6800 0340 001A
1B 6C00 0360 001B
1C 7000 0380 001C
1D 7400 03A0 001D
1E 7800 03C0 001E
1F 7C00 03E0 001F

I prefer to work with hex editors
==========
Time speed control is at 0xE8198 (headered ROM), current value is 1F.
Change it to a value as 1,3,5,7,F,1F,3F,7F,FF to keep it increasing periodically.
Color tables :
Blue is at 0x770EA
Green is at 0x7711A
Red is at 0x7714A
Two bytes per color (low byte, high byte*), 24 colors per table. Use the table above for help.
* example : 34 12 in hex will be $1234 in the game.

Contact Informations
==========

E-mail/MSN : jaspile@hotmail.com (be sure to mention a related object for your mail)
Homepage (currently only for my SMB3 hack) : http://klikechange.free.fr/smb3-sud/

You are allowed to share this patch or link it anywhere, as long as the archive isn't modified and that you don't say you made it.
You are allowed to release a modified version of the asm source if you don't claim the whole code as your own.
When used in hack, credit is recommended though not necessary.

Hack: Pot destroyed by L2-L4 Sword
Authors: Spane, Conn
Information: Destroys pots with Sword L2-L4
Rom: ALTTP (US), without header
Code Addresses:
$0x77B70 - 0x77B9B: code to implement feature

IPS URL: http://bszelda.zeldalegends.net/stuff/Con/pott_l2sword.zip

Screenshots:

ASM:

```
; This is a ASM FrontEnd Code for Zelda ALTTP (US, no header) to destroy pots with L2 -L4 sword.

lorom

org $01dabd

   jmp.l $0EFB70
   nop
   nop
   nop
   nop

org $0EFB70

   LDA $0301
   AND #$0002 ; check if hammer is used
```

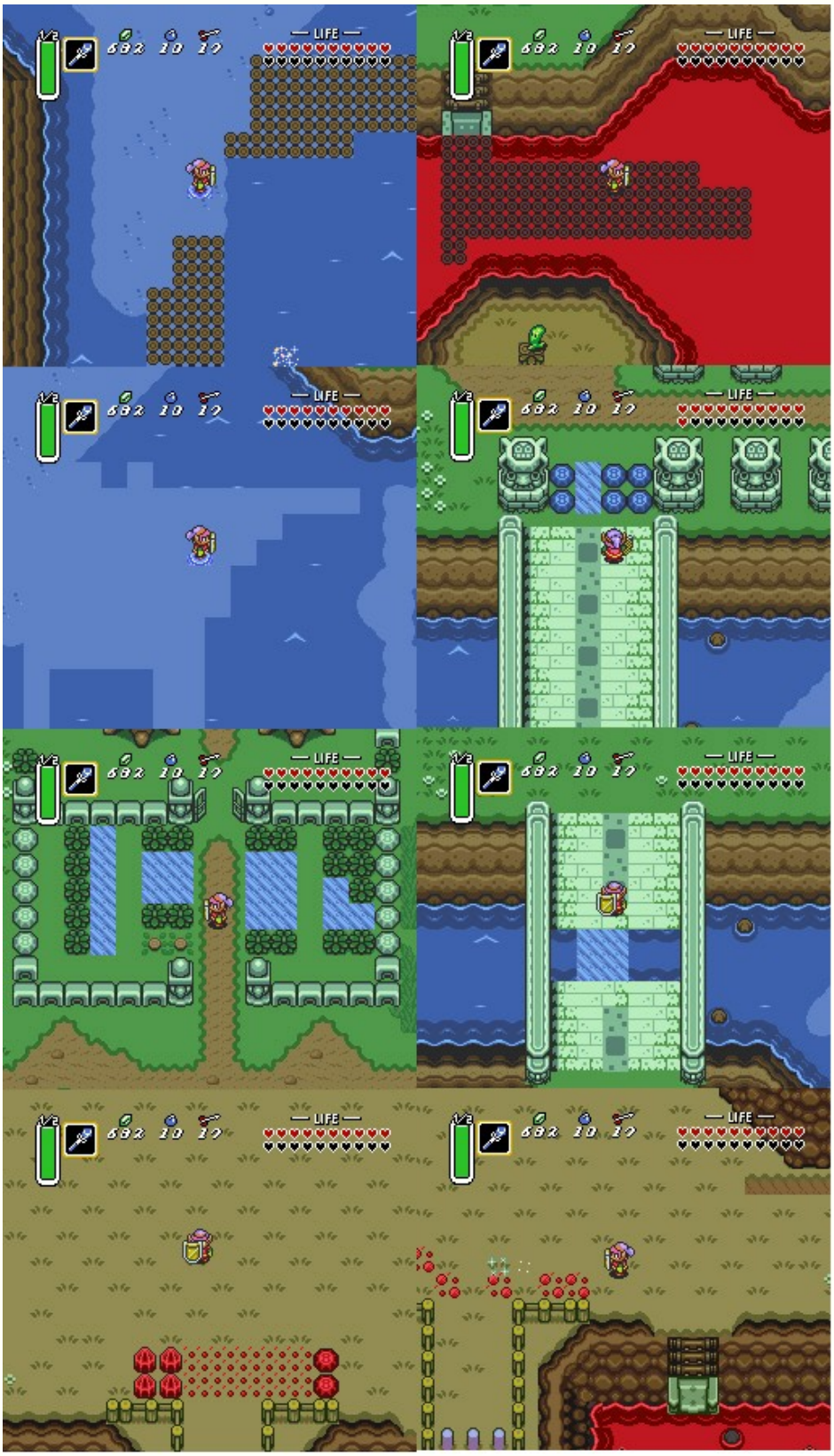
```
BEQ $04
JMP.l $01DAC5
LDA $0354
AND #$0027  ; check if sword is used
BNE $04
JMP.l $01DAB6
LDA $7EF359
AND #$00FF
CMP #$0001  ; check sword level
BNE $04
JMP.l $01DAB6
JMP.l $01DAC5
```

Hack: Ice rod freezes water
Authors: Potentialing, PuzzleDude, Conn
Information: you get icy water to walk on - works only in 16x16 screens (8-bit need still to be adjusted)
Rom: ALTTP (US), without header
Code Addresses: 0x77ba0 - 0x77d8f (after pot with sword destroy above)
Screenshots:

Icerod freezing water final:

This is the normal code for freezing deep water!
The effect was implemented in the overworlds only (simply to many unfixable bugs if indoors), and even here you might have a lot of trouble to avoid game paradoxes.

Variants of Icerod ASM:



List:
1 Ground rod
2 Ground rod (lava variant)
3 Shallow water rod
4 Water rock
5 Freeze bushes
6 Broken bridge
7 Lava rock
8 Lava ground

We are dealing with the same code, but the Icerod affects different things: from making ground and shallow water, to freeze new objects, such as water rocks, lava rocks and freezing lava ground.

Most recommended (because of zero possibility of bugs and zero possibility for game paradoxes): Water rock, Freeze bushes, Lava rock, Lava ground. With these 4 variants almost no bugs or paradoxes are possible, since you affect certain objects only.

If you however freeze all water, you can come to some paradoxes (original Icerod effect, ground rod, shallow water rod, broken bridge).

Dungeon Incomplete:

This is a try to get ice in dungeons, but I gave up on it due to several bugs:
- Link swims through ice
- Link jumps from ice into walls
- Makes Water Dungeon riddle to pull lever in light world obsolete since you must place a water layer on bg1 next to the transludent bg2 water layer!

ASM:

```
; This ASM was written by Conn
; This is a ASM FrontEnd Code for Zelda ALTTP (US, no header) to change tiles with icerod.
; Main idea is use icerod to freeze water to ice.


lorom


org $088a5d ; jsl to main code
jsl $0efba0


org $0efba0 ; main code
STA $03E4,x ; load native value
TAY
LDA $008C ; check if you're on overworld
BNE $01
RTL
LDA $03A3 ; disable other flying object icing water (boomerang, sword beam)
CMP #$06
BEQ $01
RTL
CPX #$04 ; check if ice shot #1 only is used (disable 2nd shot to ice)
BEQ $01
RTL
LDA $03E8 ; check if ice shot is on water tiles
CMP #$08
BEQ $08
LDA $03E8 ; check if ice shot is on native unused, edited blocks
CMP #$03
BEQ $01
RTL
LDA $0303 ; double check if really ice shot is used
CMP #$06
BEQ $01
```

```
RTL
LDA $0304
CMP #$06
BEQ $01
RTL
TXA
STA $7ED004 ; store native x value into ram to regain after code
LDA $4212 ; wait for vblank to enable dma transfer
AND #$80
BEQ $F9
REP #$30
LDA $2116
STA $7ED005 ; store native value to regain later
LDA $00 ; calculation procedure to get correct x,y coordinates for new tile
SEC
SBC $0708
AND $070A
ASL A
ASL A
ASL A
STA $06
LDA $02
SEC
SBC $070C
AND $070E
ORA $06
TAX
LDA #$00B7
STA $7E2000,x ; store new 16x16 ice tile into ram (property of tile!)
CLC
STZ $02 ; calculation procedure to get 8x8 vram map address (look of tile)
TXA
AND #$003F
CMP #$0020
BCC $05
LDA #$0400
STA $02
TXA
AND #$0FFF
CMP #$0800
BCC $07
LDA $02
ADC #$07FF
STA $02
TXA
AND #$001F
ADC $02
STA $02
TXA
AND #$0780
LSR A
ADC $02
STA $2116 ; store vram address for upper tile part (8x16) to $2116
STA $7ED007
LDA #$1D83 ; load new ice tiles
STA $7ED000
STA $7ED002
JSR $FD00 ; jsr to dma vram transfer for upper ice tile part
REP #$30
LDA $7ED007 ; regain vram address
ADC #$0020 ; add 20 for lower part (8x16) and store to $2116
STA $2116
JSR $FD00 ; jsr to dma vram transfer for lower ice tile part
LDA $7ED005 ; regain native register value
STA $2116
SEP #$30
LDA $7ED004 ; regain native x-value
TAX
RTL
```

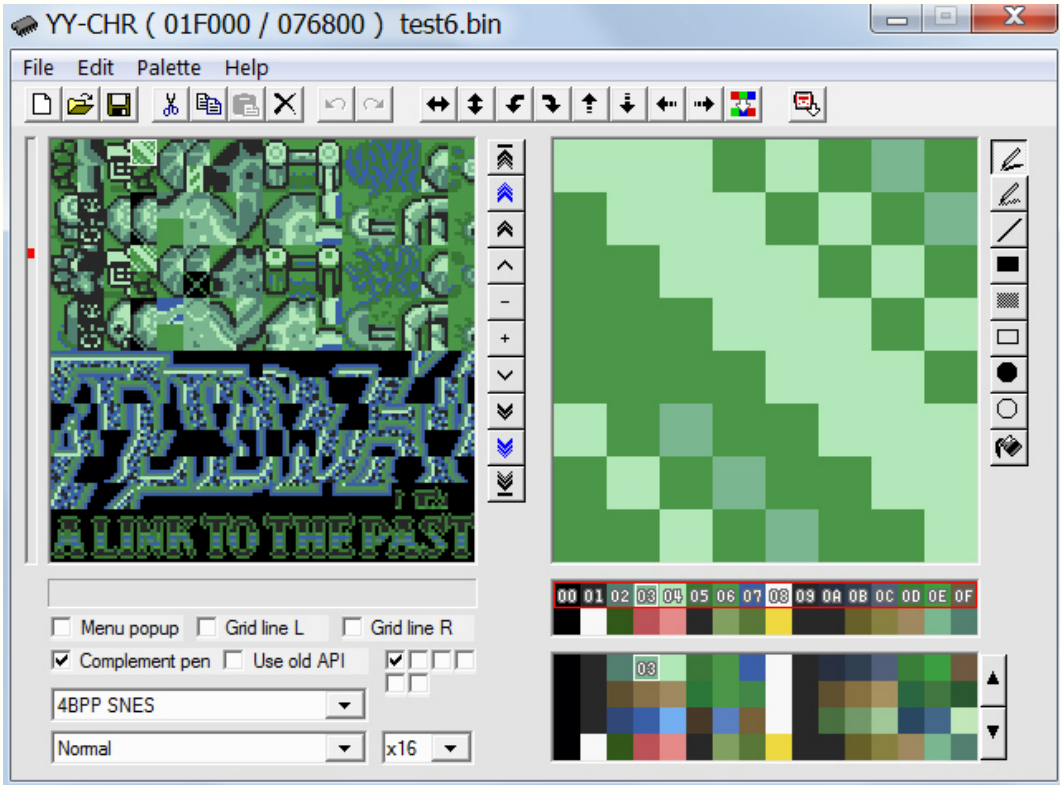
```
org $0efd00 ; vram dma transfer
LDA #$007E ; load origin of bytes to transfer (7E/d000)
STA $4304
LDA #$D000
STA $4302
SEP #$30
LDA #$18 ; bus
STA $4301
LDA #$04 ; transfer 4 bytes
STA $4305
LDA #$01
STA $4300
STA $420B ; make dma transfer
RTS


; bug fix to not swim through tiles but jump onto them

org $07dc9e
jsl $0efc80
nop

org $0efc80
LDA $0A
TSB $0343
TSB $0348
RTL

; bug fix to stop gliding on shallow water when leaving ice tile

org $07dd1b
jsl $0efc90
nop

org $0efc90
LDA $0A
TSB $0359
LDA $0350
CMP #$0100
BNE $03
STZ $034A
RTL


org $0e95dc ; get a 0e written here (first byte) to enable gliding on new tiles
ASL $5757

org $0f85b8 ; get new tile values (83 1d) written 4 times here
STA $1D,s
STA $1D,s
STA $1D,s
STA $1D,s
```

GFX AND HYRULE MAGIC EDITS ONCE THE CODE IS INSERTED

This document was written by PuzzleDude

GFX EDIT

After the "Icerod freezing water" code (all 4 parts!) is inserted via hex editing, one tile must be edited in the gfx using zcompress and yy-chr. Look at the included JPG picture.



In yy-chr go to offset 1F000. This is the place of two main gfx tilesets. These are neutral, containing bush, rock, etc. Change the top of the second bush into a small ice tile (for light and dark world gfx set).

The brightest colour in use is colour 04 in yy-chr, the main colour is 06, while the darkest is 03.

IMPORTANT!! (HYRULE MAGIC EDITS)

If you choose to implement the "Icerod freezing water" effect you might encounter some problems in the game, which are solvable by editing the overworld in Hyrule Magic, so that it fits this new effect.

Further global problems of freezing water in overworlds if no flippers:

-False collision with diagonal waves,
-Can not have a deep water screen transition (freezing deep water and transit, will make you fall in deep water with no flippers and no possibility of further freezing on the other screen),
-Can freeze rocks on water, if one 16x16 tile is half type 08, half type 02, can walk out of screen, since these rocks are a natural barrier,
-Can not jump on ice horizontally if one horizontal is iced,
-Icing to close to end of water, but leaving one line uniced, can make you jump of the ice into a wall,
-Can be stuck on the island with no green magic left.

Possible solutions:

-False collision with diagonal waves.
Change their tile type to 1 (solid).

-Can not have a deep water screen transition
I suggest bordering screen frames with shallow water in areas, which you can access without flippers.

-Can freeze rocks on water, if one 16x16 tile is half type 08, half type 02, can walk out of screen, since these rocks are a natural barrier.
If you need a barrier, you must make a full 16x16 tile, which consist completely out of a rock on the water (this tile can not change to ice).

-Can not jump on ice horizontally if one horizontal is iced.
Jump out vertically.

-Icing to close to end of water, but leaving one line uniced, can make you jump of the ice into a wall.
In most cases you are automatically bounced back into the water, despite standing partially on the wall.

-Can be stuck on the island with no green magic left.
Get the flute before the Icerod so you can escape. Get the flippers before going into dark world (no bird here).

ICE ROD FREEZING WATER, in HEX

The ASM for this effect was written by Conn. This document was written by PuzzleDude, by analysing the previously written ASM.

With this code you can freeze water using the icerod! The code is very adoptable and can be used for various other effects (like freezing new objects on the ground, freezing lava, or lava objects on the ground etc).

Rom must NOT have a header!

Explanation
3DC9E
A5 0A 0C 48 03 --> 22 80 FC 0E EA

this means go to address 3DC9E in hex and change the old bytes
A5 0A 0C 48 03 to new values, which are 22 80 FC 0E EA.

START

1.) JSL routines

3DC9E
A5 0A 0C 48 03 --> 22 80 FC 0E EA

3DD1B
A5 0A 0C 59 03 --> 22 90 FC 0E EA

40A5D
9D E4 03 A8 --> 22 A0 FB 0E
(this is the pointer to the new main code at 77BA0)

2.) New tile attributes

715DC
51 --> 0E

The new tile (which will be loaded when shooting water with the ice rod, can have these attributes):
00 - normal ground
01 - solid for Link and his weapons
02 - solid for Link only
08 - deep water
09 - shallow water
0E - slippery (default for ice tile)!

other values are not significant

3.) MAIN code

77BA0 (old values are blank FF bytes)
9D E4 03 A8 AD 8C 00 D0 01 6B AD A3 03 C9 06 F0 01 6B E0 04 F0 01 6B AD E8 03 C9 08 F0 08 AD E8 03 C9 03 F0 01 6B AD 03 03 C9 06 F0 01 6B AD 04 03 C9 06 F0 01 6B 8A 8F 04 D0 7E AD 12 42 29 80 F0 F9 C2 30 AD 16 21 8F 05 D0 7E A5 00 38 ED 08 07 2D 0A 07 0A 0A 0A 85 06 A5 02 38 ED 0C 07 2D 0E 07 05 06 AA A9 B7 00 9F 00 20 7E 18 64 02 8A 29 3F 00 C9 20 00 90 05 A9 00 04 85 02 8A 29 FF 0F C9 00 08 90 07 A5 02 69 FF 07 85 02 8A 29 1F 00 65 02 85 02 8A 29 80 07 4A 65 02 8D 16 21 8F 07 D0 7E A9 83 1D 8F 00 D0 7E 8F 02 D0 7E 20 00 FD C2 30 AF 07 D0 7E 69 20 00 8D 16 21 20 00 FD AF 05 D0 7E 8D 16 21 E2 30 AF 04 D0 7E AA 6B FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF A5 0A 0C 43 03 0C 48 03 6B FF FF FF FF FF FF FF A5 0A 0C 59 03 AD 50 03 C9 00 01 D0 03 9C 4A 03 6B FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF A9 7E 00 8D 04 43 A9 00 D0 8D 02 43 E2 30 A9 18 8D 01 43 A9 04 8D 05 43 A9 01 8D 00 43 8D 0B 42 60 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

4.) Tile reloading properties

785B8
DB 10 DC 10 EB 10 EC 10 --> 83 1D 83 1D 83 1D 83 1D

83 is the tile piece (not wise to change this, since the best possible tile was choosen, namely the upper part of the brown bush, so you can not have brown bushes in the game, so you should choose normal rocks in the areas with brown floor = mountain areas). The upper part of the brown bush is also neutral gfx tileset (do not choose the non-neutral tilesets, or already used tiles).

1D is the palette 7 for water (recomended), since ice rod produces an ice trail compatible with this pal.

The reloading tile values are also present in the main code! Change 77C44 accordingly (current is 83 1D, for tile and pal).

The difference between values in the main code and at 785B8, is the tile being loaded directly (at the use of rod) and the tile being reloaded after pressing the map or transiting in a big area and returning to ice (with no screen transit).

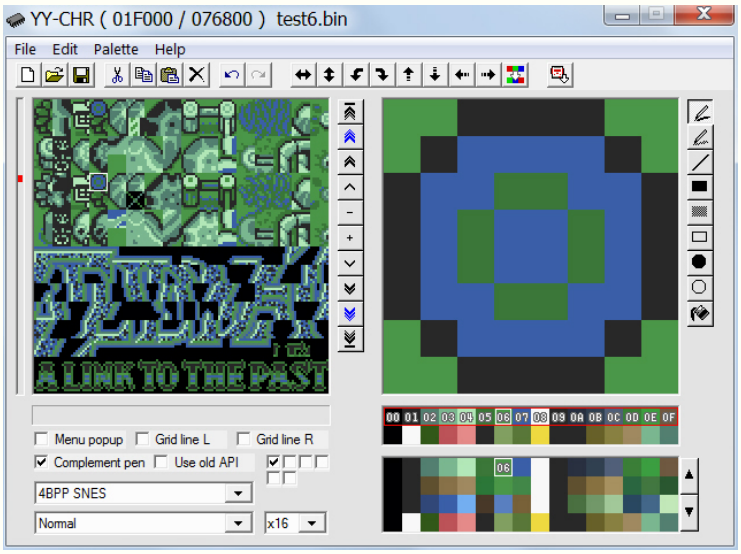
END

To implement the **Ground rod**, we can slightly change the "Icerod freezing water" ASM.

At new tile attributes, we will change the type to 00 (normal ground). It was 0E (slippery) before.

715DC
0E --> 00

The rest is a GFX change of the freeze tile into a ground tile, using zcompress and yy-chr (see picture).

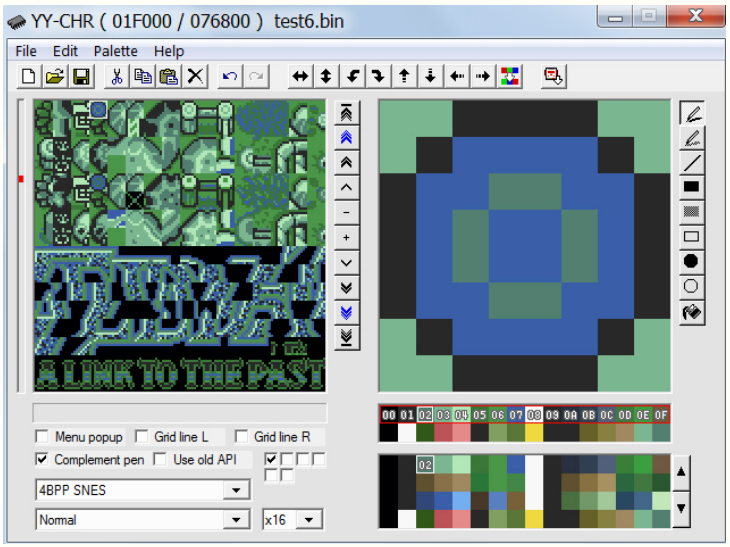


Because of the pal-7 which will be autoloaded, choose the colour 06 for background (shallow water colour), 01 will be the black frame of the ground object, 07 is the brown colour and 05 the dark brown (see picture, but ignore! the colours as presented in yy-chr, where you load custom palettes).

The **lava variant of Ground rod** is similar to normal Ground rod, it just changes some colours in yy-chr and global palette in Hyrule Magic.

Tile attribute remains 00.

715DC
0E --> 00



Because of the pal-7 which will be autoloaded, choose the colour 03 for background (deep water/lava colour), 01 will be the black frame of the ground object, 07 is the grey colour, and 02 the dark red (see picture, but ignore! the colours as presented in yy-chr, where you load custom palettes).

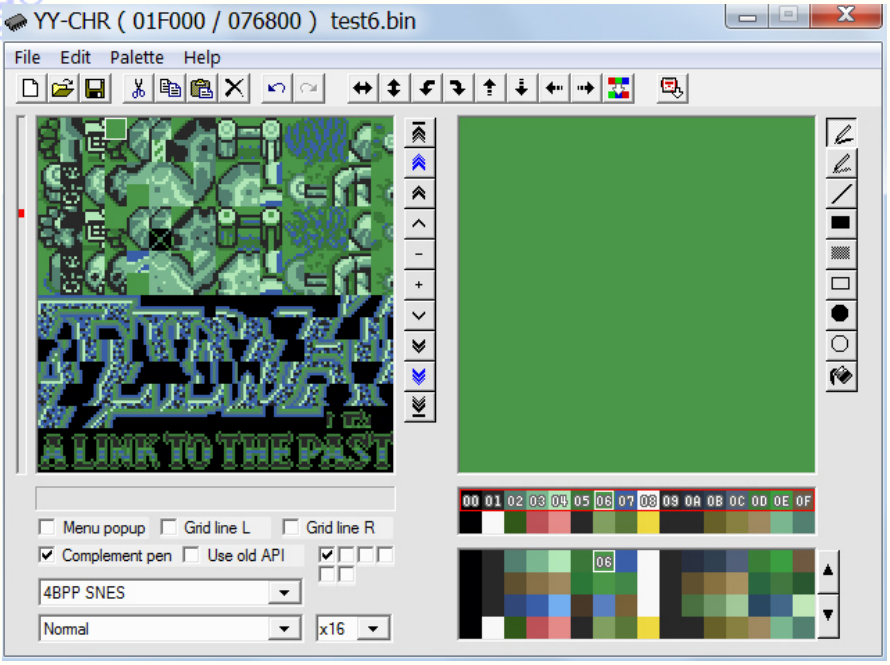
In the end we will change the global palette in Hyrule Magic to 23 (global lava palette).

To implement the **Shallow water rod**, we can slightly change the "Icerod freezing water" ASM.

At new tile attributes, we will change the type to 09 (shallow water). It was 0E (slippery) before.

715DC
0E --> 09

The rest is a GFX change of the freeze tile into a shallow water tile, using zcompress and yy-chr (see picture).



Because of the pal-7 which will be autoloaded, we will choose the colour 06 in yy-chr and blank out the entire tile (for light and dark world), so that the shallow water will be loaded.

All the codes (described so far) are great ASM achievements, but from the practical point of view very questionable. Because you can affect all water everywhere, this can easily lead to all sorts of bugs or game paradoxes.

Thus we can use this code in a different way.

Let's remember these two addresses:
77BBB and 77BC2

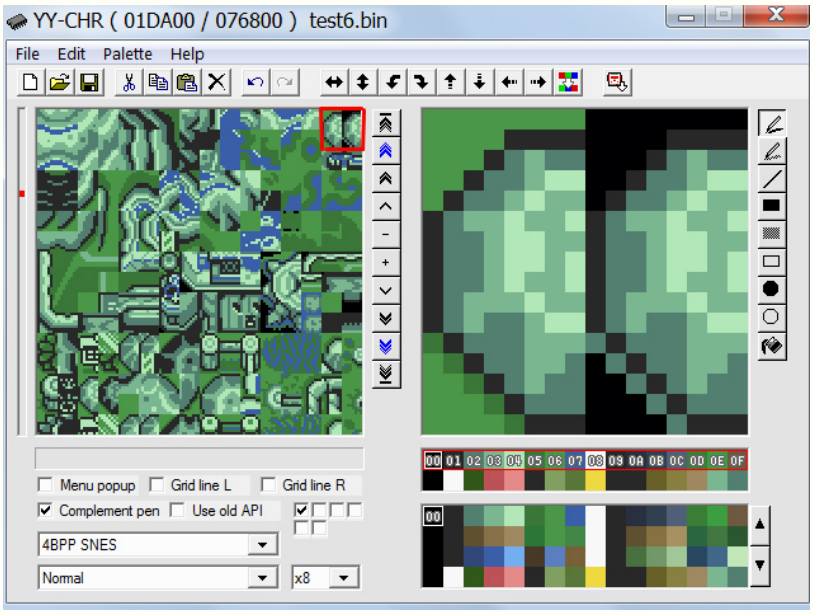
If we put both values to 03, nothing can be frozen, except the type 03 objects (which are not used anywhere). This gives us the opportunity to make a brand new item.

PuzzleDude presents... , **a Water Rock** (brand new object in Zelda3)! Let's take the original gfx for normal white rock for now and change the palette to blue = 7 (in Hyrule Magic).

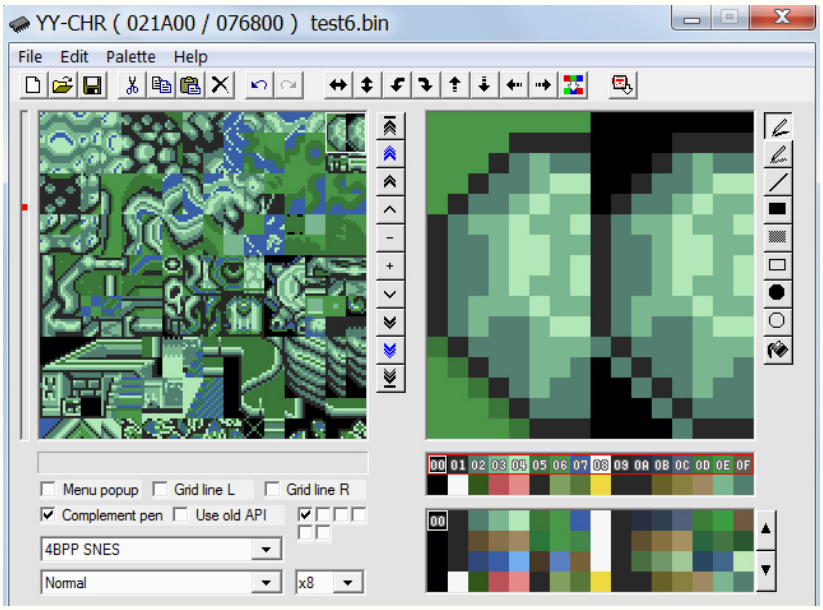
We must find an unused location for 2 tiles on neutral gfx. Fortunately 4 such tiles exist (in the overworld-wall tileset). Let's draw them in yy-chr and then give them a type 03 (later in HM).

We can have 2 versions, rock standing in shallow water (not recommended since the barrier is difficult to make in water), or on normal ground.

For the version standing in shallow water, the entire white rock can be copied, for the version standing on ground, the shadow will use a different colour (02 in yy-chr), while the green is achieved with colour 00 in yy-chr, but the frame remains 01 (note, 01 and 00 seem alike in yy-chr, but they are not).



Do the same for dark world wall-tileset!



Note, you must Not put this object in the long grass, only on plain ground type 00, or shallow water type 09, it can also handle collision with type 08 (deep water).

Let's remember these two addresses from the water rock
77BBB and 77BC2

If we put both values to 50, we can **freeze bushes**. Note, the bush must not be in long grass.

Because we have 2 places, we can freeze 2 types of objects in one game (planty enough). Here some variants:

77BBB and 77BC2:
-50 and 03 (can freeze bushes and water rocks),
-08 and 03 (can freeze deep water and water rocks),
-08 and 50 (can freeze deep water and bushes),
-09 and 08 (can freeze shallow and deep water).
etc

You can freeze any object, by selecting his tile type above.

Nothing is changed regarding the code of Icerod, simply make edits in Hyrule Magic to make a broken bridge. You can pass it with flippers or icerod.

-NOTE! No ground sprites (such as soldiers etc) must exist in such an area! or they will walk on water.
-Note, You can pass such a bridge with flippers or icerod,
-Note, You can not turn off freezing water (since it is normal water where the bridge is broken).

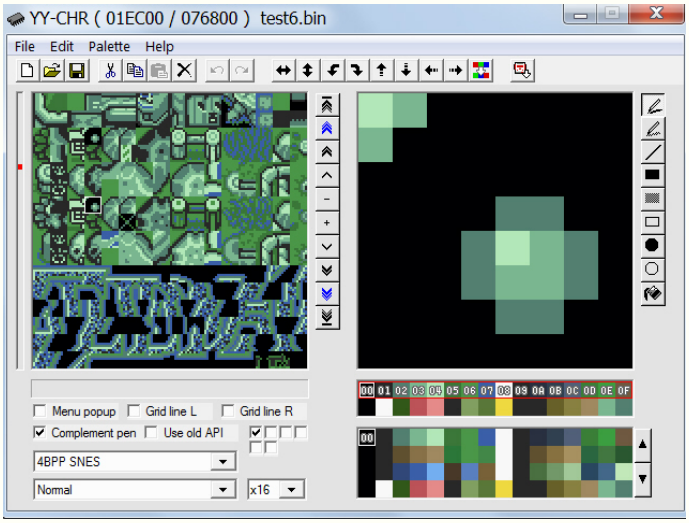
Lava rock is similar to Water rock. I've chosen 2 gfx variants here (normal white rock gfx and the lava rock from Parallel Words). You can have both in the game (it is best if they are both on solid ground, putting either in shallow lava is not a good idea; it can be done, but then you lose one type of rock, since only 4 tiles are empty).

The global pal of lava area is pal-23.

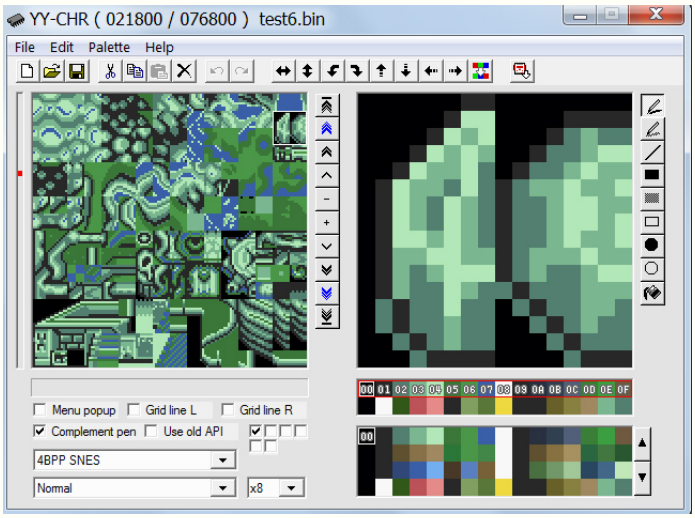
First we need to change the reloaded tile properties, since the ice tile is not compatible with pal-23.

At 715DC, choose 0D = hurting floor, since the result of a broken lava rock will hurt you (heat). You can also choose 00 here, if you don't wish to be hurt by this heat ground.

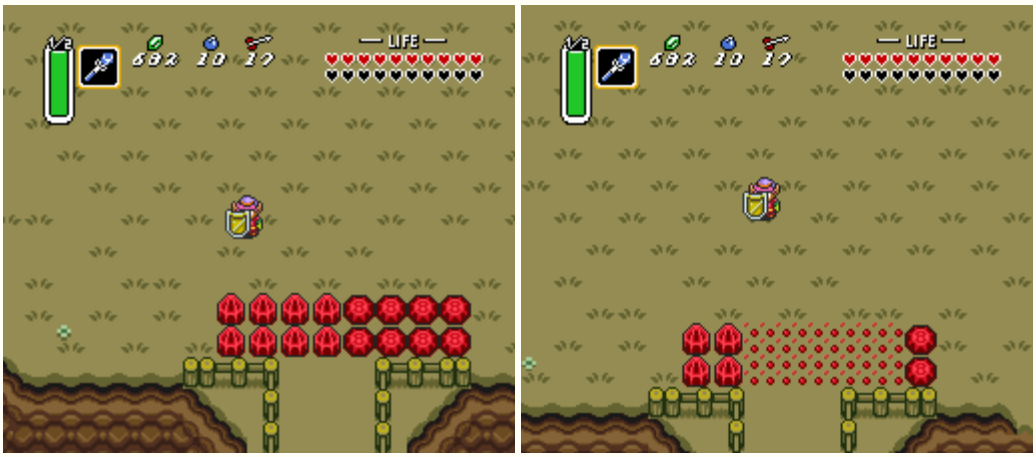
Next are the gfx changes (see JPG). We need to change the reloaded tile (previously ice) into a heat ground; choosing colours 04 (bright red), 03 (semi red), 02 (dark red), 00 (background). The gfx is also PW related.



Now we need to draw 2 types of rocks in the empty space (4 tiles) in the wall-tileset of dark world wall in yy-chr (see JPG).



Other properties are the same as in Water rock (so only type 03 can be affected by Icerod, addresses 77BBB, 77BC2, and all 4 new tiles are put to type 03 in Hyrule Magic later).



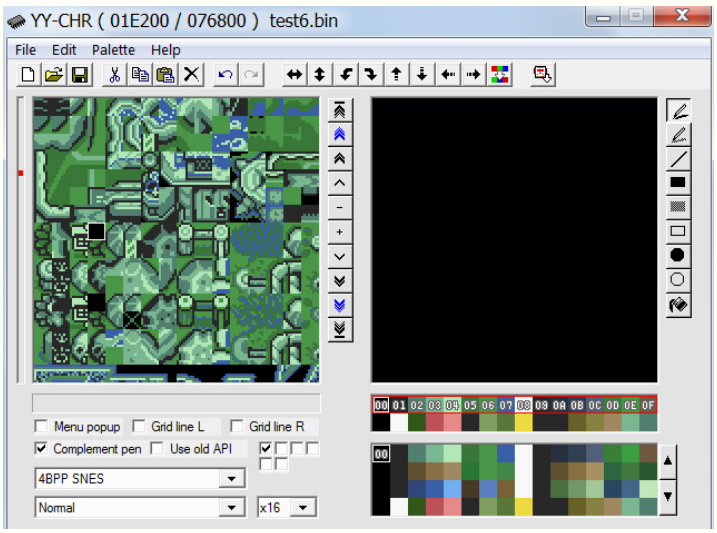
Note: Lava rock and Water rock cannot be in the same game, because one is using ice as result of freezing, the other one the hurting floor. All resulting floors can only have one tile type in one game.

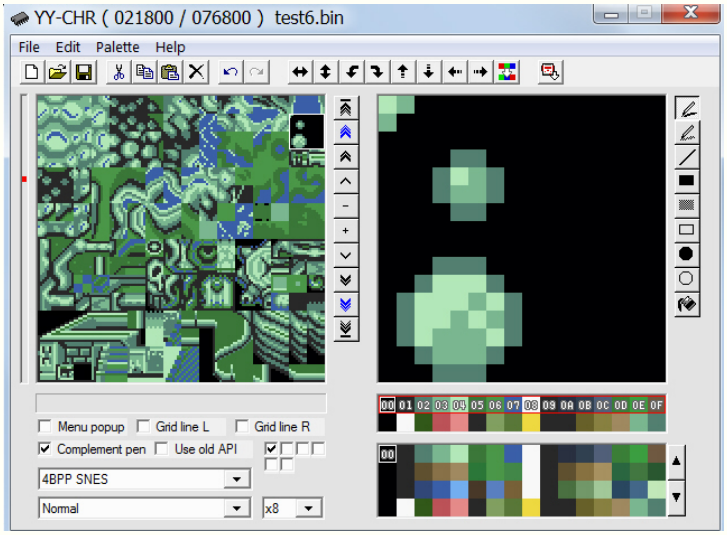
Lava ground is a variant of lava rock.

At 715DC, choose 00 = normal floor, since the result of a frozen lava ground is normal ground.

At 77BBB and 77BC2 we need 0D (which is 13 in dec), which is hurting floor. This indicates we can now freeze hurting lava ground.

Next are the gfx changes (see JPG). We need to change the reloaded tile into blank or normal floor (colour 00).





The previously drawn lava rock will now be changed into lava ground (can take 2 different tiles). These tiles will be put to type 13 (which is 0D in hex) in Hyrule magic later.

NOTE: Lava rock effect and Lava ground effect can Not coexist in one game (this is logic actually; you cannotfreeze lava rocks and lava ground at the same time, since the reload tiles of these two have different tile types and a different gfx to reload).

We could, however, make both lava rocks and lava ground disappear completely to normal ground.

Dungeon Incomplete

This code only works if you set a water layer on bg1 (normal tranlucent water layer in dungeons is bg2!)

This asm
- calculates correct dungeon coordinates
- removes bg2 layer after iceshot

Incomplete asm due to following bugs in dungeons

- Link jumps from new ice layers into wall
- Link swims through ice

ingame problems:
- swamp dungeon secret spoiled since you need a water layer on bg 1, the riddle to pull the lever in

Lightworld will be obsolete
- will only work if water is on upper floor of caves. (If you go down a ladder you're on bg2)

ASM:

```
; This ASM was written by Conn
; This is a ASM FrontEnd Code for Zelda ALTTP (US, no header) to change tiles with icerod.
; Main idea is use icerod to freeze water to ice.
; incomplete dungeon asm


lorom


org $088a5d ; jsl to main code
jsl $0efba0


org $0efba0 ; main code
STA $03E4,x ; load native value
TAY
;LDA $008C Disabled: check if you're on overworld
;BNE $01
;RTL
LDA $03A3 ; disable other flying object icing water (boomerang, sword beam)
CMP #$06
BEQ $01
RTL
CPX #$04 ; check if ice shot #1 only is used (disable 2nd shot to ice)
BEQ $01
RTL
LDA $03E8 ; check if ice shot is on water tiles
CMP #$08
BEQ $08
LDA $03E8 ; check if ice shot is on native unused, edited blocks
CMP #$03
BEQ $01
RTL
LDA $0303 ; double check if really ice shot is used
CMP #$06
BEQ $01
RTL
LDA $0304
CMP #$06
BEQ $01
RTL
TXA
STA $7ED004 ; store native x value into ram to regain after code
LDA $4212 ; wait for vblank to enable dma transfer
AND #$80
BEQ $F9
REP #$30
LDA $2116
STA $7ED005 ; store native value to regain later

;Dungeon extra
LDA $008B  ; check whether you're in dungeon
BNE $03
JMP $FCB0 ; jump to dungeon extra 1

LDA $00 ; in case you're on overworld: calculation procedure to get correct x,y coordinates for new
tile
SEC
SBC $0708
AND $070A
ASL A
ASL A
```

```
ASL A
STA $06
LDA $02
SEC
SBC $070C
AND $070E
ORA $06
TAX
LDA #$00B7
STA $7E2000,x ; store new 16x16 ice tile into ram (property of tile!)
CLC ; calculation procedure to get 8x8 vram map address (look of tile)
STZ $02
TXA
AND #$003F
CMP #$0020
BCC $05
LDA #$0400
STA $02
TXA
AND #$0FFF
CMP #$0800
BCC $07
LDA $02
ADC #$07FF
STA $02
TXA
AND #$001F
ADC $02
STA $02
TXA
AND #$0780
LSR A
ADC $02
STA $2116 ; store vram address for upper tile part (8x16) to $2116
STA $7ED007
LDA #$1D83 ; load new ice tiles
STA $7ED000
STA $7ED002
JSR $FD00 ; jsr to dma vram transfer for upper ice tile part
REP #$30
LDA $7ED007 ; regain vram address
ADC #$0020 ; add 20 for lower part (8x16) and store to $2116
STA $2116
JSR $FD00 ; jsr to dma vram transfer for lower ice tile part


JSR $FD30; jsr to dungeon extra 2

LDA $7ED005 ; regain native register value
STA $2116
SEP #$30
LDA $7ED004 ; regain native x-value
TAX
RTL


org $0efcb0; dungeon extra 1 - tile coordinates
LDA $00 ; calculation of tile coordinates
AND #$01F8
ASL A
ASL A
ASL A
CLC
ADC $04
CLC
ADC $05
TAX
AND #$F000
STA $7ED009
```

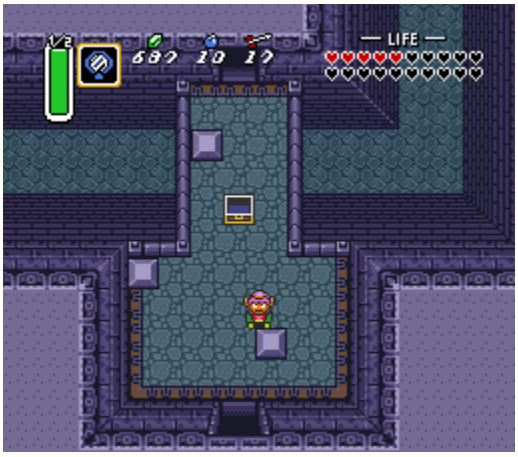
```
LDA #$0F0F
STA $7F2000,x ; tiles on bg1
STA $7F2040,x
STA $7F3000,x ; tiles on bg2
STA $7F3020,x
JMP $FC0E  ; jump to calculation procedure to get 8x8 vram map address (look of tile)



org $0efd00 ; vram dma transfer
LDA #$007E ; load origin of bytes to transfer (7E/d000)
STA $4304
LDA #$D000
STA $4302
SEP #$30
LDA #$18 ; bus
STA $4301
LDA #$04 ; transfer 4 bytes
STA $4305
LDA #$01
STA $4300
STA $420B ; make dma transfer
RTS


; bug fix to not swim through tiles but jump onto them

org $07dc9e
jsl $0efc80
nop

org $0efc80
LDA $0A
TSB $0343
TSB $0348
RTL

; bug fix to stop gliding on shallow water when leaving ice tile

org $07dd1b
jsl $0efc90
nop

org $0efc90
LDA $0A
TSB $0359
LDA $0350
CMP #$0100
BNE $03
STZ $034A
RTL


org $0efd30 ;dungeon extra 2 - remove bg2 layer
REP #$30
LDA #$0000
LDA $008B  ; check whether your in dungeon
BEQ $01
RTS
LDA #$01EB  ; load transparency tiles
STA $7ED000
STA $7ED002
CLC
LDA $7ED007
ADC #$1000
STA $2116
JSR $FD00  ; store transparency on bg2
REP #$30
CLC
```

```
LDA $7ED007
ADC #$1020
STA $2116
JSR $FD00
RTS


org $0e95dc ; get a 0e written here (first byte) to enable gliding on new tiles
ASL $5757

org $0f85b8 ; get new tile values (83 1d) written 4 times here
STA $1D,s
STA $1D,s
STA $1D,s
STA $1D,s
```

Hack: Blocks moveable multiple times
Authors: PuzzleDude, Conn
Information: the blocks can be moved unlimited (or until you hit an obstacle)
Rom: ALTTP (US), without header
pc w/o header ALTTP (US)
Code Addresses: 0x77e00 - 0x77e1c (after pot with sword destroy and icerod freezing water above)

IPS url: http://bszelda.zeldalegends.net/stuff/Con/moveblock.zip

Screenshot:



ASM:

```
; This ASM was written by Conn
; This is a ASM FrontEnd Code for Zelda ALTTP (US, no header) to move blocks multiple times.

lorom

org $01d87d
NOP
NOP
JSL $0EFE00 ; (below the icerod code to not overwrite it)

org $0efe00
CMP #$2727    ;check if block is already moved
BEQ $0F          ; if yes branch to after first RTL
```

```
LDA $7F2000,x ; load block value (e.g., 70, 71, 72...)
STA $7ED010  ; store to intermediate (unused) 7E/D000 is icerod
LDA $00          ; set old block value to 00 to indicate that it is moved
STA $7F2000,x
RTL
LDA $7ED010 ; branch here if 27 - load intermediate value
STA $7F2000,x  ; store intermediate instead of 27
RTL
```

Hack: Transfer sprites
Authors: Conn, PuzzleDude
Information: Transfer the data for Indoor sprites to another bank to have more room for them. This also means more room for outdoor sprites, since if left in place, they can extend into the former indoor sprites data-area.
Rom: ALTTP (US), without header
pc w/o header ALTTP (US)
Code Addresses: main code at $14C296 (can be put elsewhere)
Screenshot:

More information:

Transfer the indoor sprites

Goal: transfer the data for Indoor sprites to another bank to have more room for them. This also means more room for outdoor sprites, since if left in place, they can extend into the former indoor sprites data-area.

0.) Expand the rom to 2MB (go to 100000 and add another 100000 bytes).

1.) Go to 4D62E and select block 1671 in hex. Copy the selection first, then fill selection with FF bytes. Go to 14D62E and paste/rewrite.

2.) At 04C296:
A8 B9 2E D6 -->22 96 C2 29 (pointer)

3.) At 14C296:

E2 30 A9 29 48 AB C2 30 AD 8E 04 0A A8 B9 2E D6 6B (new code)

4th byte (29) controls the global bank.
15th and 16th byte (2E D6) control the local address.
Together is 2E D6 29 = 29D62E (snes) = 14D62E (pc)

All secondary pointers are local (so pointers + data have 8000 bytes of space, before it was 1671).

Written by Conn

To make it work:
at 04C296: A8 B9 2E D6 --> 22 96 C2 29
at 14C296: E2 30 A9 29 48 AB C2 30 AD 8E 04 0A A8 B9 2E D6 6B

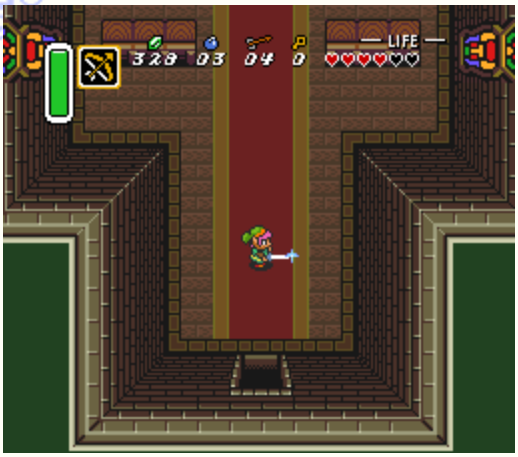
Explanation:
e2 30: 1 byte read
lda 29, pha, plb (you know)
c2 30: re-enable 2 byte read
ad 8e 04 - lda $048e, 0a ASL A regain native value in accumulator (rewritten in your hack to 29), located at 4c292
a8: TAY transfer A to Y
b9 2e d6: LDA $D62E,y

After sprites are loaded: E2 30 A9 09 48 AB C2 30 6B
check before if it is 1 or 2 byte read, in case it is 2 byte read you need the e2 30 and then the c2 30, if it already is 1 byte read, then dismiss it.

By the way, the table you point to must be located at 14/d62e (the code now points there). (You could however also rewrite the pointer opcode b9 2e d6 to b9 00 d0 and paste your table beginning at 14/d000)

But for the pointer code itself it is of course not needed to place it at at 14/c296 (it is somewhere in the middle). You can make the jsl anywhere where it is free space (keeps an order in your rom). E.g. take a free ff ff area as I did with the icerod code or place it to 14/c200 or 14/c000 or whatever.

Hack: Cannot move when holding sword
Authors: Conn, PuzzleDude
Information: When you hold your sword to do the spin attack, you cannot move (this avoids visual bugs like going backwards or sideward into the staircase).
Rom: ALTTP (US), without header
pc w/o header ALTTP (US)
Code Addresses: main code at $0009C2
Screenshot:



ASM:

```
org $0003d9
jsr $89c2 ; jsr to 0x0009c2 (unused region)

org $0009c2

LDA $4219 ; reload joypad opcode deleted for jsr
CLC
SBC #$80 ; exclude normal movement <80
BCC $07
SBC #$0B ; exclude movement >8a
BCS $03
LDA #$80 ; freeze if sword is charging
RTS
LDA $4219 ; regain value if normal movement
RTS
```

HEX:

No move (hex changes)

0003D9 (pointer)
AD 19 42 --> 20 C2 89

0009C2 (main code), old values were FF
AD 19 42 18 E9 80 90 07 E9 0B B0 03 A9 80 60 AD 19 42 60

Hack: Play custom music
Authors: Conn, PuzzleDude
Information: Load (play) custom overworld music, without the reseting when changing screens. (Original Alttp always resets it to Overworld music when exiting most entrances).

Rom: ALTTP (US), without header
pc w/o header ALTTP (US)
Code Addresses: main code at $11A200, new music table at $11A300.

ZIP url: https://www.dropbox.com/s/04lgxiheviak28g/Play%20custom%20music%20%28final%29.zip

Screenshot:

ASM:

```
;CUSTOM MUSIC, ASM BY CONN


;Goal: load custom overworld music, without the reseting when changing screens. (Original Alttp
always resets it to Overworld music).


lorom

org $008100
jsr $89d5  ;pc: 00/09d5 (after your sword charge no move)

org $0089d5
jsl $23a200
RTS

org $23a200
TAX ; transfer native value to X
SBC #$0F ; check if theme is in range 00-0F (overworld themes)
BCC $04 ; continue if yes
STZ $012C ; if no, set 012c to 00 and return to game
RTL
LDA $008c ; check overworld
cmp #$00
BNE $0b
LDA $008a
cmp #$00
BNE $04
STZ $012C ; if you are in dungeon/cave, set 012c to 00 and return to game
RTL
TXA ; regain native value X transfer to A
; now a comparison starts of themes you want to have replaced, this is overworld 02, lost woods 05,
kakariko 07, dark world 09, skull woods 0d. This is necessary to not replace title screen,
introduction, minigame, character selection, rabbit, rain, which is not on the map!
CMP #$02 ; if overworld theme is about to play jump to $A231 and replace, if not check if it is 05
and so forth.
BNE $03
JMP $A243
CMP #$05
BNE $03
JMP $A243
CMP #$07
BNE $03
JMP $A243
```

```
CMP #$09
BNE $03
JMP $A243
CMP #$0D
BNE $03
JMP $A243
STZ $012C ; if it is another theme (introduction, title screen -> return to game)
RTL
LDA $012C ; if theme is already loaded return to game (address is loaded twice)
CMP $0131
BNE $04
STZ $012C
RTL
LDX $040A ; load screen number to X
LDA $23A300,x ; select theme from table, in dependence of screen in X
STA $012C ; play theme
STA $2140
STA $0132
RTL
```

;The table at 11/a300 contains all 07 (kakariko). You simply need to replace these number by the number you like to have played.

;The table beginning at 01/4343 needs to be an exact copy of the table 11/a300 to avoid a music reset everytime you switch screens.

ADDITIONAL FIX

AD 8C 00 C9 00 D0 0B
AD 8A 00 C9 00 D0 04
9C 2C 01
6B

Load address 008C, compare to 00, if not 00 (so if 01= on overworld), branch 0B forward to code,
the same for the other, but branch only 4 bytes;
but an old 9C 2C 01 if it is 00 (so if indoors, remain as it was).

This fix implements the code only if you are in the overworld (but not indoors). It must be so.

Conn wrote:
I made a check 11/a209: ad 8c 00 c9 00 and ad 8a 00 9c 00. If both values (7e/008c and 7e/008a are 00) you are indoors.

CUSTOM MUSIC, in HEX

Original ASM (code) written by Conn

Goal: load custom overworld music, without the reseting when exiting any entrance. (Original Alttp always resets it to Overworld music).

Expand the rom to 2MB (go to 100000 and add another 100000 bytes, with the value 00).

100
(jsr routine)
9C 2C 01 --> 20 D5 89

9D5
(jsl routine after no-move while holding sword code)
FF FF FF FF FF --> 22 00 A2 23 60

14343
(Alttp music part1 with no ambient)
05 05 02 02 02 02 02 02 05 05 02 02 02 02 02 02 02 02 02 02 02 02 07 07 02 02 02 02 02 02 07 07 07 02 02 02 02
02 07 07 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02

14383
(Alttp music part2, must be a copy of part1)
05 05 02 02 02 02 02 02 05 05 02 02 02 02 02 02 02 02 02 02 02 02 07 07 02 02 02 02 02 02 07 07 07 02 02 02 02
02 07 07 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02

143C3
(Alttp music part3, first block 40 in hex must be a copy of part1 and part2)
05 05 02 02 02 02 02 02 05 05 02 02 02 02 02 02 02 02 02 02 02 02 07 07 02 02 02 02 02 02 07 07 07 02 02 02 02
02 07 07 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 0D 0D 09 0D 0D 0D 0D 0D 0D 0D 09 0D 0D
0D 0D 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09
09 09 09 09 09 09 09 09 09 09 09 09 05 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02

11A200
(new main code)

AA E9 0F 90 04 9C 2C 01 6B AD 8C 00 C9 00 D0 0B AD 8A 00 C9 00 D0 04 9C 2C 01 6B 8A C9 02 D0 03 4C 43 A2 C9 05 D0 03 4C 43 A2 C9 07 D0 03 4C 43 A2 C9 09 D0 03 4C 43 A2 C9 0D D0 03 4C 43 A2 9C 2C 01 6B AD 2C 01 CD 31 01 D0 04 9C 2C 01 6B AE 0A 04 BF 00 A3 23 8D 2C 01 8D 40 21 8D 32 01 6B

11A300
(new table to control area music, copy of part3)
(the default values are from Alttp, no ambient)
05 05 02 02 02 02 02 02 05 05 02 02 02 02 02 02 02 02 02 02 02 02 02 07 07 02 02 02 02 02 02 07 07 07 02 02 02 02 02 07 07 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 0D 0D 09 0D 0D 0D 0D 0D 0D 0D 09 0D 0D 0D 0D 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 09 05 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02

Remove balast bytes above 11A39F.

Note
at 11A300 all overworld music is controlled (from area 00 to 9F). Once this is set, do this:

1.) select block 40 in hex of the new table, and copy it to 14343 and 14383.

2.) select block A0 in hex of the new table (all), and copy it to 143C3.

With this the music autoreload after each screen transit is disabled! Music will reset only if 2 adjacent areas have different music, but not if music is the same in 2 adjacent areas.

Music song values in hex:

-00 No music

Overworld:
-01 Triforce + title screen
-02 Overworld
-03 Rain
-04 Rabbit
-05 Lost Woods
-06 Introduction
-07 Kakariko Village
-08 Portal SFX
-09 Dark World
-0A Master Sword
-0B Name select screen
-0C Guard summoned
-0D Skull Woods
-0E Minigame

-0F Title screen only

Indoors:
-10 Hyrule Castle
-11 Light World dungeon
-12 Cave
-13 Medallion/crystal acquired
-14 Sanctuary
-15 Boss
-16 Dark World dungeon
-17 Fortune Teller
-18 Cave (appears to be identical to -12)
-19 Zelda rescued in prison
-1A Sage rescued in crystal
-1B Fairy spring
-1C Ganon's theme only
-1D Ganon appears in Agahnim's shadow
-1E Face-to-face + Ganon's theme
-1F Ganon battle

Ending:
theme 20 triforce talk to you in alttp
theme 21 holding triforce+fanfare which shows happy hyrule
theme 22 credits

Big area problem:

one big area is 4 small ones, so 4 bytes must change:
lets say the area is xx

code is
xx, xx+1, xx+8, xx+9

+8 is one horizontal unit (of 8 small areas) to come to the lower part of the big area.

So xx is upper left, xx+1 is upper right, xx+8 is lower left, xx+9 is lower right to cover the big area.

A big area at position 00 is the same as 4 small areas at possitions 00, 01, 08, 09. --> big area 00 = small area 00 + small area 01 + small area 08 + small area 09. And the table corresponds to small areas.

So if area 00, you must change the bytes at areas:
00, 01, 08, 09
and put them all on 07 for music Town for instance, or 05 for forest music or 02 for overworld music etc.

Benefit of the new code and exceptions:

Can play any overworld song (town and forest as well), without reseting it to overworld music once you exit any entrance!
In Alttp it always resets to overworld music (regardless of the actual music set, which makes a paradox if the town or forest music is set in the area).

Exception are entrances 2C and 3C, see the other file (special entrances) for that.

Custom music HEX, Special entrances

Special entrances are 2C, 3C both connected and
39, 46, 11, 67 (all connected).

Conn wrote:

addresses for entrances 2C and 3C
(default is in the forest)
PC: 1043E: A2 05 (before sword draw) music part1
PC: 10448: A2 02 (after sword draw) music part2

So forest music in part1, but overworld music in part2 when exited from 2C or 3C.
---

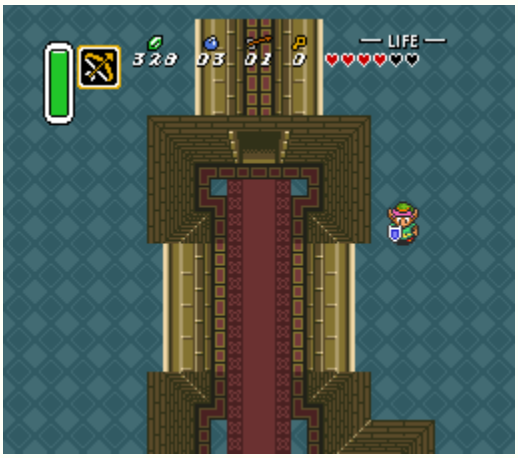
addresses for entrances 39, 46, 11, 67
(default is in the village)
PC: 10418: A2 07 (before agahnim) music part1 and part2
PC: 10422: A2 02 (after agahnim) music part3

So village music in part1 and part2, but overworld music in part3 when exited from 39, 46, 11, 67.

!
But all is covered by the table! (new code) except
PC: 1043E: A2 05 (before sword draw).
Ajust manually what 2C and 3C should play.

So you need to adjust the byte at upper address (05 by default = forest music) to set the correct music when comming out of these entrances.

Hack: Lens of Truth
Author: XaserLE
Information: This is for making BG2 disappear when pressing X and R together (parallel worlds x button secret). But only if you have the Mirror in the inventory (suitable to revert it into Lens of Truth).
Rom: ALTTP (US), without header
pc w/o header ALTTP (US)
Code Addresses: main code at $1FA600
Screenshot:



ASM:

```
;WRITTEN: by XaserLE
;THANKS TO: -MathOnNapkins' Zelda Doc's
; -wiiqwertyuiop for his Zelda Disassembly

; Layer Flags: xxxsabcd (i count BG from 1 to 4 - MathOnNapkins RAM-Map counts from 0 to 3)
; s - Sprite layer enabled
; a - (BG4 enabled)??
; b - BG3 enabled
; c - BG2 enabled
; d - (BG1 enabled)??

; NOTE BY Puzzledude, remove the 3 lines, which have "added" on the end, to disable bg2 all the
time (no Mirror needed)


;header
lorom

ORG $0288FD ; go to the code that branches behin the dungeon map load if player didn't press X
BRA $1C ; make it always branch, so map isn't loaded anymore

ORG $068365 ; go to a originally JSL that is executed every frame
JSL $3FA600 ; overwrite it (originally JSL $099F91)

ORG $03FA600 ; go to expanded space to write our routine

LDA $1B ; load data that tells us whether we are in a building or not
AND #$01 ; are we in a building?
BEQ $1A ; if not, don't use the x-button-secret, edited from 12 to 1A
LDA $F2 ; load unfiltered joypad 1 register (AXLR|????)
CMP #$50 ; X and R buttons pressed?
BNE $06 ; if not, go to enable BG2
LDA $7EF353 ; load ram mirror slot, added
CMP #$02 ; compare to value 02 (have mirror is 02, no mirror is 00), added
BNE $06 ; if not value 02, go to enable BG2, added
LDA $1C ; load layer flags
AND #$FD ; disable BG2 (0xFD = 11111101)
```

```
BRA $04 ; go to store layer flags
LDA $1C ; load layer flags
ORA #$02 ; enable BG2 (0x02 = 00000010)
STA $1C ; store layer flags

JSL $099F91 ; at first execute original code

RTL
```

HEX:

Lens of Truth HEX

This is for making BG2 disappear when pressing X and R together (parallel worlds x button secret). But only if you have the Mirror in the inventory (suitable to revert it into Lens of Truth).

108FD
F0 --> 80

30366
91 9F 09 -->00 A6 3F

new code (rom must be expanded to 2MB)
at 1FA600
A5 1B 29 01 F0 1A A5 F2 C9 50 D0 06 AF 53 F3 7E C9 02 D0 06 A5 1C 29 FD 80 04 A5 1C 09 02 85 1C 22 91 9F 09 6B

Diferent key combinations = change the 10th byte in the main string to the following:

byte-buttons pressed
80 - A (not recommended)
70 - X+R+L
60 - X+L
50 - X+R (default), command C9 50.
40 - X
30 - L+R
20 - L
10 - R

Hack: Direction Change while running with Pegasus Boots
Author: Conn, AST creator team ^^
Information: This is a stolen and adapted code from AST, it will make you change direction while running with your pegasus boots. You need to keep the run button pressed, because stopping by direction change will not work anymore (of course)
Rom: ALTTP (US), without header
pc w/o header ALTTP (US)
Code Addresses: main code at $0x3ff40 (works also with PW)

Version 2: Keep running during screen transitions
Zip: http://bszelda.zeldalegends.net/stuff/Con/pegasus_upgrade.zip

Screenshot:

More information:

Code was written by Conn (wrong... those guys who created AST ;)

For ALTTP (native and hacks) US, no header

This code will make 2 changes on the pegasus boots:
- you need the button keep pressed to run
- you can change direction while running

(1) code hijack at
03/911d: a5 f0 29 0f -> 4c 40 ff ea

(2) at 03/ff40
added adjusted code from AST

(Address 03/FF40 was chosen to make it also compatible with Parallel Worlds)

Hack: Close gap in Euclids 24 item-menu, by enabling an extra item (shovel)
Authors: Conn
Information: Get shovel in menu instead of empty space (patch changes empty space used previously for bottle content to get shovel there), so you can have and use the flute and shovel at the same time. You only need to place the shovel in a nice chest via HM or hex editing (not implemented by this patch since this is your choice).
Rom: Parallel Worlds-prepatched ALTTP (can be adapted to your menu if you use the 24 items menu hack from Euclid)

Zip: http://bszelda.zeldalegends.net/stuff/Con/pw_shovel.zip

Screenshots:

More information:

Parallel Worlds add-on patch by Conn

Get shovel in menu instead of empty space (patch changes empty space used for bottle content to get shovel there)

Bottle content is saved to unused 7E/D100, and 7E/D101 is used to check whether bottle is equipped

Apply on Parallel Worlds-prepatched ALTTP.

Still needs to be done:
- The current version (1.1) does not have a shovel, you must place it in a chest via Hyrule Magic
- Hide some treasures to dig up ;)

ips for rom without header (Parallel_Worlds_shovel.ips)
or for rom with header (Parallel_Worlds_shovel_header.ips)

(1) shovel gfx in menu

06/e46f: 51 -> 11
06/e517: a9 5c 2c 8d 58 14 8d 5a 14 8d 98 14 8d 9a 14 -> ea ea ea ea ea ea ea ea ea ea ea ea ea ea
06/ebb0: ad 02 02 -> 20 a0 ea (hijack code show letters "shovel" in menu)
06/eaa0:
ad 02 02
29 ff 00
c9 10 00 ; check if position 10 (bottle)
d0 04
a9 18 00
aa ; transfer pointer 18 for shovel to X
ad 02 02
60

06/fb60: ea 85 02 -> 20 c0 ea set pointers to show shovel as Y item in box when playing
06/eac0:

```
85 02
e2 30
af 01 d1 7e ;check if bottle is equipped
c9 00
d0 06  ; if not branch
e0 10
d0 02
a2 0d
c2 30
60
```

(2)load/store bottle content to 7E/d100

```
06/de0c: af f4 f3 7e -> af 00 d1 7e
06/E59f: 8f 4f f3 7e -> 8f 00 d1 7e
06/ebbb: af 4f f3 7e -> af 00 d1 7e
06/fae3: af 4f f3 7e -> af 00 d1 7e
```

(3) make bottle selectable item

```
06/de3d: d0 -> 80
06/de9d: d0 -> 80
06/E588:  d0 -> 80
```

(4) check if 7e/0202 is bottle or shovel, save 01 to unused 7e/d101 if bottle, 00 if shovel

06/faeb: a9 10 00 -> 20 20 ea (hijack code to unused 06/ea20, executed from menu to game if $0202 > 15 -> bottle)

```
06/ea20:
a9 01 00
8f 01 d1 7e ; store 01 to 7e/d101 to keep saved that bottle and not shovel is chosen
a9 10 00    ; store 10 to $0202
60
```

06/e5db: af 4f f3 7e 18 69 14-> 20 50 ea ea ea ea ea (hijack code to 06/ea50, executed from game to menu only when $0202 is 10)

```
06/ea50:
af 01 d1 7e
c9 01       ;check if bottle is in Y button-box (equipped)
f0 03
a9 10
60          ;if not, return
a9 00
8f 01 d1 7e  ; reset 7e/d101
af 00 d1 7e  ; load bottle number
18
69 14       ; add 14 to get to correct bottle in menu
60
```

06/fada: 9b aa 80 14 -> 4c 70 ea ea (hijack code to 06/ea70; executed from menu to game to put correct Y item to use)
06/ea70:
9b
aa
af 01 d1 7e ; check if bottle is chosen
c9 01 00
f0 0b ; if yes branch
af 4f f3 7e ; if not load shovel
29 ff 00
bb
4c fa fa
af 00 d1 7e ; load bottle number
29 ff 00
aa
bf 5b f3 7e ; load bottle content
29 ff 00
bb
4c fa fa

(5) use of items:

04/850E: 4c -> 4f (to store shovel at 7e/f34f) when you get it

03/A166: AF 4F F3 7E -> af 00 d1 7e (enable bottle use (remap))

pointer: 03/9afa 5b a1 -> 13 a3

03/A313: AF 4C F3 7E -> 5c e0 ea 0d (hijack code to distinguish between bottle, flute and shovel)
06/eae0:
af 01 d1 7e ; check if bottle is in Y
c9 00
F0 04 ; branch if not
5c 5b a1 07; jump to bottle pointer (overwritten at pointer above)
ad 02 02
c9 0d ; check if flute
d0 08 ; if not branch
af 4c f3 7e; load flute
5c 1d a3 07; return to fluteload
af 4f f3 7e; check status of 7e/f34e
c9 00
d0 04
5c 12 a3 07; if no value return
c9 01
d0 04
5c 2c a3 07; if shovel run execution
5c 1d a3 07; if any other value return to flute load

(6) further changes

(don't know when these are executed, but safe is safe, otherwise you may get a shovel with a bottle!)
06/de29: 8f 4f f3 7E -> 8f 00 d1 7e
06/e168: 8f 4f f3 7E -> 8f 00 d1 7e (think this is run when you receive a bottle)

Hack: Random and special treasures to dig up
Author: Conn
Information: This patch will give you the ability to dig up random treasures and you will be able to hide dig-up sprites like Heart Container pieces!
Rom: ALTTP (US), without header
Main code 0x77e30, which is unused space in native ALTTP - but not in Parallel Worlds. If you want to use it on PW, please shift the code to unused space there!

Zip: http://bszelda.zeldalegends.net/stuff/Con/random_special_treasure.zip

Inside you find merged patch with random rupees gain and the ability to insert specific treasures (tested and confirmed so far only for Heart Container pieces) to hide anywhere except big screens. Inside a zip you will also find the old patch that only allows to dig up random treasures.

Since it is not possible to set other items like boomerang to dig up, I made a small tutorial how to replace the flute by another item. This however is restricted to screen 2A where you natively find the flute.

Screenshots:
Random rupees:

Special treasure collect:

Flute replacement:

_____

Waiting for the dawn

Hack: Kholdstare Melting Shell Restoration
Author: MathOnNapkins
Information: Fixes a bug in the palette manipulation logic of Kholdstare's shell to restore a dormant fade effect. The fade is supposed to occur when the shell is defeated and is meant to simulate melting, probably.
Rom: ALTTP (US), without header

IPS: http://tcrf.net/images/e/eb/Kholdstare_shell.ips

Screenshot:

_____
Waiting for the dawn

~ Chapter VII - Additional knowledge

1) RAM Addresses

by MathOnNapkins

Version 6 (Last Updated 11/29/2011)

Quick Guide:
Memory addresses are listed in the following format:

$ADDR[**0xSIZE**]

Both **ADDR** and **SIZE** are expressed as hexadecimal values. Yes, I am aware that the '**$**' and '**0x**' prefixes both indicate hexadecimal. This is merely a convention.

Some arrays and addresses might conflict with one another, but such conflicts are often context dependent. That is, they don't actually conflict during gameplay because they are not used at the same time.

==========

Start of bank 0x7E and direct page memory

==========

Note that these addresses are only one byte and you can assume that they are **$7E00XX** where **XX** are the byte addresses you see below.

$00[**0x10**] - Mainly used as work registers. Storage of addresses and values.

$10[**0x01**] - Main Module index
$11**[**0x01**] - Submodule Index (See **$B0)

$12[**0x01**] - NMI Boolean.
If set to **zero**, the game will wait at a certain loop until NMI executes. The NMI interrupt generally sets it to **one** after it's done so that the game can continue doing things other than updating the screen.

$13[**0x01**] - Screen settings

$14[0x01] - Graphics Flag - value based flag, that if nonzero, causes the tile map to update
$15[0x01] - Graphics Flag - update CGRAM (0x200 bytes)
$16[0x01] - Graphics Flag - update HUD portion of BG3 tile map from $7EC700 (0x14A bytes)
$17[0x01] - Graphics Flag -
$18[0x01] - Graphics Flag

$19[0x01] - Graphics flag.
When nonzero, will trigger a transfer from $7FXXXX to vram address $YY00
XXXX is specified by variable $0118
YY is specified by this variable

$1A[0x01] - Frame Counter

$1B[0x01] - Flag is set to 1 if you are in a building. Set to 0 otherwise (credit: Euclid)

$1C[0x01] - Main Screen Designation ($212C):

xxxsabcd

s - Sprite layer enabled
a - BG3 enabled
b - BG2 enabled
c - BG1 enabled
d - BG0 enabled

$1D[0x01] - Sub Screen Designation ($212D)

$1E[0x01] - Window Mask Activation ($212E)
$1F[0x01] - Subscreen Window Mask Activation ($212F)

$20[0x02] - Link's Y-Coordinate (mirrored at $0FC4)
$22[0x02] - Link's X-Coordinate (mirrored at $0FC2)
$22[0x01] - (*alternate*) Used as a step counter for the attract mode.

$24[0x02] - 0xFFFF usually, but if Link is elevated off the ground it is considered to be his Z coordinate
i.e. it's his height off of the ground

$26[0x01] - Link's push state (These of course can be combined):

right = 0x1,
left = 0x2,
down = 0x4,
up = 0x8,
nothing = 0.

$26[0x01] - (*alternate*) Indicates the current 2bpp graphic to display in attract mode.

$27[0x01] - Link's Recoil for vertical collisions
$27[0x01] - (*alternate*) Flag indicating whether to display a new 2bpp graphic in attract mode.

$28[**0x01**] - Link's Recoil for horizontal collisions
$28[**0x01**] - (*alternate*) Aghanim's base X coordinate relative to the screen. (history mode)

$29[**0x01**] - vertical resistance
$29[**0x01**] - (*alternate*) Aghanim's base Y coordinate relative to the screen. (history mode)

$2A[**0x01**] - horizontal resistance

$2B[**0x02**] - ????

$2D[**0x01**] - ????

$2E[**0x01**] - animation steps.. seems to cycle from **0** to **5** then repeats

$2F[**0x01**] - The direction Link is facing:

0 - up,
2 - down,
4 - left,
6 - right.

$30[**0x01**] - When Link is moving down or up, this is the signed number of pixels that his Y coordinate will change by
$31[**0x01**] - Same as **$30** except it's for X coordinates

$32[**0x02**] - seems to be used in some subsection of **Bank 07**, not sure which one though

$34[**0x04**] - ???? These seem to be free ram

$38[**0x02**] - seems to be set some of the time when going up diagonal walls.
It's a bit field for tiles type **0x10** through **0x13**

$3A[**0x01**] - **Bit 7**: the B button is held down or sword is out.
Bit 6: the Y button is down.
Bit 0: Initial B button press

$3B[**0x01**] - **Bit 7**: the A button is down.
Bit 4: ???

$3C[**0x01**] - Lower Nibble: How many frames the B button has been held, approximately.
Upper nibble set to 9 on spin attack release.

$3D[**0x01**] - A delay timer for the spin attack. Used between shifts to make the animation flow with the flash effect.
Also used for delays between different graphics when swinging your sword.

$40[**0x01**] - Y coordinate related variable
$41[**0x01**] - X coordinate related variable

$42[**0x01**] - set to **0x0F** during preoverworld
$43[**0x01**] - "

$44[0x01] - set to 0x80 during preoverworld
$45[0x01] - "

$46[0x01] - A countdown timer that incapacitates Link when damaged or in recoil state.
If nonzero, no movement input is recorded for Link.

$47[0x01] - Set when damaging enemies, unsure of exact usage yet.

$48[0x01] - If set, when you push A Link will grab at the air.

$49[0x01] - This address is written to make Link move in any given direction. Indoors it is cleared every frame.
Outdoors it is not cleared every frame so watch out.

$4B[0x01] - Link's visibility status. If set to 0xC, Link will disappear.

$4C[0x01] - Counter that decreases every frame when the Cape is in used. Starts at 4 and counts down to 0.
When it reaches 0, your magic meter is decremented based on whether you have full or 1/2 magic.
There's a table in Bank 07 that determines this

$4D[0x01] - An Auxiliary Link handler.
As far as I know:

0x00 - ground state (normal)
0x01 - the recoil status
0x02 - jumping in and out of water?
0x04 - swimming state.

$4E[0x01] - ????

$4F[0x01] - Index for creating the dashing sound effect. If frozen to a single value, no sound occurs.

$50[0x01] - A flag indicating whether a change of the direction Link is facing is possible.
For example, when the B button is held down with a sword.

0 - can change,
non zero - can't change.

$51[0x01] - ????

$53[0x01] - ????

$55[0x01] - Cape flag, when set, makes you invisible and invincible. You can also go through objects such as bungies.

$56[0x01] - Link's graphic status:

1 - bunny link,
0 - real link.

$57[0x01] - Modifier for Link's movement speed.

0 - normal,
0x01 to 0x0F - slow,

0x10 and up - fast.
Negative values actually reverse your direction!

$58[**0x01**] - Bitfield describing interactions with stairs tiles.

uuuussss
s - Stair tiles
u - free ram

If this masked with **0x07** equals **0x07**, Link moves slowly, like he's on a small staircase
$02C0 also needs this variable to be **nonzero** to trigger

$59[**0x01**] - ????

$5A[**0x01**] - ????

$5B[**0x01**] - **0**: indicates nothing
1: Link is dangerously near the edge of a pit
2: Link is falling

$5D[**0x01**] - Link Handler or "State"
0x0 - ground state
0x1 - falling into a hole
0x2 - recoil from hitting wall / enemies
0x3 - spin attacking
0x4 - swimming
0x5 - Turtle Rock platforms
0x6 - recoil again (other movement)
0x7 - hit by Aghanim's bug zapper
0x8 - using ether medallion
0x9 - using bombos medallion
0xA - using quake medallion
0xB - ???
0xC - ???
0xD - ???
0xE - ???
0xF - ???
0x10 - ???
0x11 - falling off a ledge
0x12 - used when coming out of a dash by pressing a direction other than the dash direction
0x13 - hookshot
0x14 - magic mirror
0x15 - holding up an item
0x16 - asleep in his bed
0x17 - permabunny
0x18 - stuck under a heavy rock
0x19 - Receiving Ether Medallion
0x1A - Receiving Bombos Medallion
0x1B - Opening Desert Palace
0x1C - temporary bunny

0x1D - Rolling back from Gargoyle gate or PullForRupees object
0x1E - The actual spin attack motion.

$5E[0x01] - Speed setting for Link. The different values this can be set to index into a table that sets his real speed. Some common values:

0x00 - Normal walking speed
0x02 - Walking (or dashing) on stairs
0x10 - Dashing

$60[0x01] - During story mode, used as a counter for one set of jump tables.

$64[0x02] - 0x1000 if $EE = 1,
0x2000 if $EE = 0.

$66[0x01] - Indicates the last direction Link moved towards.
Value wise:

0 - Up,
1 - Down,
2 - Left,
3 - Right

$67[0x01] - Indicates which direction Link is walking (even if not going anywhere)
bitwise: 0000abcd.

a - Up,
b - Down,
c - Left,
d - Right

$68[0x01] - ????

$69[0x01] - ???

$6A[0x01] - ???

$6B[0x01] - moving up against a \ wall: 0x1A
moving right against a \ wall: 0x25
moving down against a \ wall: 0x15
moving left against a \ wall: 0x2A

moving up against a / wall: 0x19
moving left against a / wall: 0x26
moving right against a / wall: 0x29
moving down against a / wall: 0x16

$6C[0x01] - Indicates whether you are standing in a doorway

0 - not standing in a doorway.
1 - standing in a vertical doorway
2 - standing in a horizontal door way

$6D[0x01] - When you are moving against a diagonal wall and you are deadlocked, i.e. you are pressing against it directly, but aren't going anywhere, this will contain the value that **$6B**[0x01] would have.

$6E[0x01] - (archiving but not sure if this is correct) Related to certain tile behaviours (see tile type 2 I think) nearby
moving against a \ wall from below: **0**
moving against a \ wall from above: **2**
moving against a / wall from below: **4**
moving against a / wall from above: **6**

$6F[0x03] - As of now, I consider this to be free ram

$72[0x01] - ????

$73[0x01] - has value **0x02** when master sword beam is active...

$74[0x02] - ??? Related to moving water?

$76[0x02] - When link interacts with certain tile types, the index of that tile gets stored here.

$78[0x01] - possibly used in the context of chests.

$79[0x01] - Controls whether to do a spin attack or not (*Found by PARCCC from gshi*)

$7B

$82[0x02] - ????

$84[0x02] - not sure...
$86[0x02] - not sure...
$88[0x02] - not sure...

$8A[0x02] - Overworld Screen Index

$8C[0x01] - Overlay index (credit: Euclid)
$8D[0x01] - ???

$90[0x02] - Points to current position in the low OAM buffer (**the first 512 bytes**)
$92[0x02] - Points to current position in the high OAM table buffer (**later 32 bytes**)
$94[0x01] - Screen Mode Register (**$2105**)
$95[0x01] - Mosaic Settings (**$2106**)
$96[0x01] - Window Masks for Backgrounds 1 and 2 (**$2123**)
$97[0x01] - Window Masks for Backgrounds 3 and 4 (**$2124**)
$98[0x01] - Window Masks for Obj and Color Add/Subtraction Layers (**$2125**)
$99[0x01] - Enable Fixed Color +/- (**$2130**)
$9A[0x01] - Enable +/- per layer (**$2131**)
$9B[0x01] - HDMA channels to write to (**$420C**)

$9C[0x01] - writes to **$2132**. Red fixed color component
$9D[0x01] - writes to **$2132**. Green fixed color component
$9E[0x01] - writes to **$2132**. Blue fixed color component

$9F[0x01] - free ram?

$A0[0x02] - The index used for loading a dungeon room. There are 296 rooms all in all. (mirrored in other variables).
$A2[0x02] - Points to the previous dungeon room.
$A4[0x02] - Indicates the current floor Link is on in a dungeon.

$A6[0x01] - Set to **0** or **2**, but it depends upon the dungeon room's layout and the quadrant it was entered from.
$A7[0x01] - ""

$A8[0x01] - Composite of dungeon room layout info and quadrant info that gets updated periodically

000cccba

ccc - layout that the room uses (0 to 7 obviously)
b - ORed in value of **$AA**
a - ORed in value of **$A9**

$A9[0x01] - **0** if you are on the left half of the room.
1 if you are on the right half.

$AA[0x01] - **2** if you are the lower half of the room.
0 if you are on the upper half.

$AB[0x02] - free ram?

$AD[0x01] - ??? collision?

$AE[0x01] - In dungeons, holds the Tag1 Value. (*see Hyrule Magic*)
$AF[0x01] - In dungeons, holds the Tag2 Value. (*see Hyrule Magic*)

$B0[0x01] - Sub-submodule index. (Submodules of the **$11** submodule index.)
$B1[0x01] - free ram?

$B2[0x02] - Width indicator for drawing dungeon objects
$B4[0x02] - Height indicator for drawing dungeon objects

$B7[0x03] - Used as storage during dungeon loading for a 3-bit pointer to be indirectly accessed.

$BA[0x02] - Often used as a position into a buffer of data during dungeon loading.

$BD[0x0?] - ??? height for initializing sprites?
Address recorded during tile interactions? (**Bank 07**)

$BF[0x??] - Used during the dungeon

$C8[0x01] - (in menus) keeps track of what part of a menu you are in.
For example,

0 - 4 on the select game screen,
0 - 2 for each save game,
3 & 4 are copy and erase game.

(in ending module) 16 bit timer used for stepping through each ending sequence.

$E0[**0x02**] - BG1 horizontal scroll register (**$210F**)
$E2[**0x02**] - BG2 horizontal scroll register (**$210D**)
$E4[**0x02**] - BG3 Horizontal Scroll Register (**$2111**)
$E6[**0x02**] - BG1 Vertical scroll register (**$2110**)
$E8[**0x02**] - BG2 Vertical scroll register (**$210E**)
$EA[**0x02**] - BG3 Vertical Scroll Register (**$2112**)

$EC[**0x02**] - Tilemap location calculation mask. Is only ever set to **0xFFF8** or **0x01F8**

$EE[**0x01**] - In dungeons, **0** Means you're on the upper level.
1 Means you're on a lower level. Important for interaction with different tile types.

$EF[**0x01**] - Room Transitioning Value (bitwise)

bit 0 - Toggles between BG0 and BG1. One example: Sanctuary and Hyrule Castle. (see door type up-11)
bit 1 - Transition between Sewer and Hyrule Castle. Xors the dungeon index by **0x02**.

$F0**[**0x01**] - Unfiltered Joypad 1 Register: Same as **$F4, except it preserves buttons that were being pressed in the previous frame.

$F1**[**0x01**] - Unfiltered Joypad 2 Register: Same as **$F5, except it preserves buttons that were being pressed in the previous frame. Note: Input from joypad 2 is not read unless you do some ASM hacking.

$F2**[**0x01**] - Unfiltered Joypad 1 Register: Same as **$F6, except it preserves buttons that were being pressed in the previous frame.

$F3**[**0x01**] - Unfiltered Joypad 2 Register: Same as **$F7, except it preserves buttons that were being pressed in the previous frame. Note: Input from joypad 2 is not read unless you do some asm hacking

$F4[**0x01**] - Filtered Joypad 1 Register: [BYST | udlr].
Lower case represents the cardinal directions, T = start. S = select.

$F5[**0x01**] - Filtered Joypad 2 Register: [BYST | udlr].
Lower case represents the cardinal directions, T = start. S = select.
Note: Input from joypad 2 is not read unless you do some asm hacking

$F6[**0x01**] - Filtered Joypad Register [AXLR | ????]
LR: The shoulder buttons. ? = unknown inputs

$F7[**0x01**] - Filtered Joypad Register [AXLR | ????]
LR: The shoulder buttons. ? = unknown inputs.

Note: input from joypad 2 is not read unless you do some asm hacking.

$F8[**0x01**] - ???

$FC[**0x02**] - .... Overrides for dungeon room transitions? (Seen used with big bombable walls)

$FF[**0x01**] - Vertical IRQ Trigger (this is the vertical scanline that will trigger the IRQ)

==========

End of direct page memory and start of mirrored bank 0x7E memory locations) = Page 0x01

==========

This is beyond direct page and you can assume that all addresses here are $7E**XXXX** until you reach bank $7F.

$0100[**0x02**] - Numerical index that controls the graphics that are blitted for Link during VRAM. See DMA Variables for additional info.

$0102[**0x02**] - See DMA Variables
$0104[**0x02**] - See DMA Variables
$0106[**0x01**] - free ram?
$0107[**0x02**] - See DMA Variables

$0109[**0x01**] - See DMA Variables

$010A[**0x01**] - Set to **nonzero** when Link incurs death.
Used if the player saved and continued (or just continued) after dying,
indicating that the loading process will be slightly different.

$010B[**0x01**] - free ram?

$010C[**0x02**] - Temporary storage for a module number. Example: If we're in Overworld mode (0x9) and have to display a textbox (module 0xE), 0x9 gets saved to this location on a temporary basis. Once the textbox disappears this module will be resumed.

$010E[**0x02**] - gives the entrance index of the current dungeon

$0110[**0x02**] - In the context of loading dungeon rooms, contains the index of the room (see $A0) multiplied by 3. Allows for indexing into 24bit pointer address tables.

$0112[**0x02**] - Apparently a flag indicating the bombos medallion is falling. Stops the Menu from being dropped down too. It seems to work as a flag for any general extended animation that is currently in progress.

$0114[**0x01**] - Value of the type of tile Link is currently standing on.

$0116[**0x02**] - Used with routine $008CB0 to transfer 0x800 bytes from $7E1000 to a variable location. (determined by this memory location)

$0118[**0x02**] - Local portion of an address used to transfer data from $7FXXXX to vram whenever variable $19 is **nonzero**.

$011A[**0x02**] - BG1 Y Offset. Gets applied to **$0124**. Can be used in quakes/shaking
$011C[**0x02**] - BG1 X Offset. Gets applied to **$0120**

These are extra buffers for the scroll variables in addition to the **$E0**,...,**$EA** registers earlier
$011E[**0x02**] - BG1 Horizontal Scroll Register (**$210F**)
$0120[**0x02**] - BG0 Horizontal Scroll Register (**$210D**)
$0122[**0x02**] - BG1 Vertical Scroll Register (**$2110**)
$0124[**0x02**] - BG0 Vertical Scroll Register (**$210E**)

$0126[**0x01**] - Seems to be used during dungeon screen transitions as some sort of counter.

$0128[0x01] - ???? Seen as **0xFF** during the history sequence. When set to **0xFF**, the IRQ routine will disable the IRQ routine the next time it executes by writing **0x81** to **$4200**.

$012A[0x01] - Seems to be a flag for the crystal sequence scripting. As long as this flag is set, the sequence progresses. Also used for the Triforce sequences. Triggers a subsection of the NMI routine.

$012C - Music control
- **0x01** = Triforce opening
- **0x02** = light world
- **0x03** = legend theme
- **0x04** = bunny link
- **0x05** = lost woods
- **0x06** = legend theme
- **0x07** = kakkariko village
- **0x08** = mirror warp
- **0x09** = dark world
- **0x0A** = restoring the master sword
- **0x0B** = faerie theme
- **0x0C** = chase theme
- **0x0D** = dark world (skull woods)
- **0x0E** = game theme (Overworld only?)
- **0x10** = hyrule castle
- **0x13** = fanfare
- **0x15** = boss theme
- **0x16** = dark world dungeon
- **0x17** = fortune teller
- **0x18** = caves
- **0x19** = sentiment of hope
- **0x1A** = crystal theme
- **0x1B** = faerie theme w/ arpeggio
- **0x1C** = fear & anxiety
- **0x1D** = Agahnim unleashed
- **0x1E** = surprise!
- **0x1F** = Ganondorf the Thief
- **0x20** = nothing
- **0x21** = Agahnim unleashed
- **0x22** = surprise!
- **0x23** = Ganondorf the Thief
- **0xF1** = fade out
- **0xF2** = half volume
- **0xF3** = full volume
- **0xFF** = Load a new set of music.

$012D[0x01] - Ambient Sound effects
- **0x05** = Silencio
- **0x15** = Door to Triforce room opens

$012E[0x01] - Sound Effects 1
- **0x00** = none (no change)
- **0x01** = small sword swing 1 (Fighter Sword)

0x02 = small sword swing 2 (Fighter Sword)
0x03 = medium sword swing 1 (Master Sword and up)
0x04 = fierce sword swing 2 (Master Sword and up)
0x05 = object clinking against the wall
0x06 = object clinking Link's shield or a hollow door when poked
0x07 = shooting arrow (or red Goriyas shooting fireballs)
0x08 = arrow hitting wall
0x09 = really quiet sound
0x0A = hookshot cranking
0x0B = door shutting
0x0C = loud thud / heavy door shutting
0x0D = magic powder noise
0x0E = fire rod being fired
0x0F = ice rod being fired
0x10 = hammer being used?
0x11 = hammer pounding down stake
0x12 = really familiar but can't place it exactly
0x13 = playing the flute
0x14 = Link picking something small up?
0x15 = Weird zapping noise
0x16 = Link walking up staircase 1
0x17 = Link walking up staircase 2 (finishing on next floor)
0x18 = Link walking down staircase 1
0x19 = Link walking down staircase 2 (finishing on next floor)
0x1A = Link walking through grass?
0x1B = sounds like a faint splash or thud
0x1C = Link walking in shallow water
0x1D = picking an object up
0x1E = some sort of hissing (walking through grass maybe?)
0x1F = object smashing to bits
0x20 = item falling into a pit
0x21 = sounds like a plop on wood, can't remember where it gets used
0x22 = loud thunderous noise
0x23 = pegasus boots slippy sound
0x24 = Link making a splash in deep water (but having to come back out)
0x25 = Link walking through swampy water
0x26 = Link taking damage
0x27 = Link passing out
0x28 = item falling onto shallow water
0x29 = rupees refill sound
0x2A = whiffy sound
0x2B = low life warning beep
0x2C = pulling master sword out
0x2D = taking magic power damage from enemy
0x2E = Seems like a sound related to Ganon's Tower opening
0x2F = Seems like a sound related to Ganon's Tower opening

0x30 = chicken getting owned
0x31 = big faerie healing your wounds
0x32 = sounds familiar but can't place it. Whooshing noise
0x33 = cheesy noise that happens when Agahnim makes chicks disappear
0x34 = cheesy noise that happens when Agahnim makes chicks disappear

0x35 = faint rumbling noise
0x36 = faint rumbling noise
0x37 = spin attack has powered up
0x38 = swimming noise
0x39 = thunderous noise while Ganon's tower opens
0x3A = some kind of clinky hit
0x3B = sounds like magic powder but unfamiliar
0x3C = Error noise
0x3D = something big falling into water? (Argghus?)
0x3E = playing the flute (quieter)
0x3F = magic powder

$012F[0x01] - Sound Effects 2
0x00 = none (no change)
0x01 = master sword beam
0x02 = unintelligible switch noise
0x03 = ????? yet another loud thud
0x04 = Nutcase soldier getting pissed off
0x05 = shooting a fireball (Lynel)
0x06 = ?????
0x07 = giant metal balls coming out of nodes
0x08 = normal enemy taking a big hit
0x09 = normal enemy dying
0x0A = collecting rupee
0x0B = collecting single heart
0x0C = text scrolling flute noise
0x0D = single heart filling in on HUD
0x0E = sound of a chest being opened
0x0F = you got an item sound 1
0x10 = switching to map sound effect
0x11 = menu screen going down
0x12 = menu screen going up
0x13 = throwing an item
0x14 = clink (not sure what it's used for)
0x15 = loud thud 1
0x16 = loud thud 2
0x17 = rats
0x18 = dragging something
0x19 = zora shooting fireball
0x1A = minor puzzle solved
0x1B = major puzzle solved
0x1C = doing minor damage to enemy
0x1D = taking magic power damage from enemy
0x1E = keese flitting around
0x1F = Link squealing like a bitch when he falls into a hole

0x20 = switch menu item
0x21 = boss taking damage
0x22 = boss dying
0x23 = spin attack sound
0x24 = switching between different mode 7 map perspectives
0x25 = triggering a switch

0x26 = Aghanim's lightning
0x27 = Agahnim powering up an attack
0x28 = Agahnim / Ganon teleporting
0x29 = Agahnim shooting a beam
0x2A = tougher enemy taking damage
0x2B = Link getting electrocuted
0x2C = bees buzzing
0x2D = Major achievement completed (e.g. getting a piece of heart)
0x2E = Major item obtained (e.g. Pendant or Heart Container)
0x2F = obtained small key
0x30 = blop popping out of ground
0x31 = Moldorm's weird track like noise
0x32 = bouncing off a bungie
0x33 = short unknown jingle
0x34 = ...
0x37 = Neat sound... consider using as a confirmation (though make sure it's not buggy)
0x38 = ...
0x39 = ...
0x3C = Obtained mid-level item (e.g. bombs from a chest)
0x3D = ...
0x3F = More minor fanfares

$0130 - ????
$0131 - ????

$0132[0x01] - Buffer for playing songs. Put a value here to try to play a song. (Will try to write to $012C)

$0134[0x02] - VRAM target address for animated tiles. Usually $3B00 or $3C00.
Remember that if you were to look this stuff up in Geiger's debugger the byte addresses would actually be $7600 and $7800. SNES VRAM addresses are expressed as words addresses internally, you just have to deal with it unfortunately.

$0136[0x01] - Flag that toggles when different sets of musical tracks are loaded.
I think the two track sets are normal outdoor and the ending tracks.

$0137[0xC9] - Normal (Non-IRQ) Stack

==========

Page 0x02

==========

$0200[0x01] - Sub-submodule index for mode E.
$0201[0x01] - Seems to be never referenced
$0202[0x01] - currently selected item (credit: Euclid)

$0203[0x01] - *Module 0x0E.0x01* references it but never seems to use it
$0204[0x01] - *Module 0x0E.0x01* references it but never seems to use it
$0205[0x01] - *Module 0x0E.0x01* uses it in the creation and destruction of the bottle submenu as a progress indicator
$0206[0x01] - *Module 0x0E.0x01* increments it as a frame counter similar to $1A, but it never seems to get used (debug probably)
$0207[0x01] - *Module 0x0E.0x01* uses it as a timer for the flashing item selector circle

$0208[0x01] - Countdown timer that controls when the next animation of hearts being filled in occurs.
(The rotating animation)

$0209[0x01] - Index related to **$0208**. When **$0208** counts down to **zero**,
$0209 is incremented, then reassigned *modulo 4* (logical and by **0x03**)
Determines the graphics used in each step of the heart refill animation. (4 steps)
Once **$0209** reaches **4** (*which is 0 modulo 4*), **$020A** is set to **zero** to indicate that
the animation for this particular heart is finished.

$020A[0x01] - Flag that indicates whether a heart refill animation is taking place. **Nonzero** if that is the case.
$020B[0x01] - Seems to be a debug value for *Module **0x0E**.**0x01***
$020C[0x01] - oh hells naw
$020D[0x01] - Used in some submodule of module **0x0E** in **Bank 0A**
$020E[0x01] - Floor index for the dungeon map.
Floor 1F is the basic floor with 0x00. 1B is 0xFF. 2F is 0x01. You get the idea.
$020F[0x01] - Referenced in Module **0x0E** in **Bank 0A** but doesn't seem to be used.
$0210[0x01] - Mostly referenced in Module **0x0E** in **Bank 0x0A** with various specific usages I've yet to document

$0211[0x02] - ???? Shows up in **Bank 0x0A** only. Indexes **$0217**...

$0213[0x02] - Shows up in **Bank 0x0A** only.

$0215[0x02] - Relevant to dungeon map mode sprites...

$0217[0x02] - Dungeon map related too?

$0219[0x02] - Specifically used as the target address in VRAM for the HUD portion of the BG3 tile map.
For the entirety of the game this is supposed to stay as **0x6040** (**0xE080 in byte addressing**)
$021D[0x04] - Assigned the value **0x007F4841**... doesn't seem to be referenced anywhere else though
$0221[0x02] - Assigned a value of **0xFFFF** (**-1**) in **Bank 0x0C**, never seems to be referenced though.
$0223[0x02] - There's a few **STZ $0223**'s that pop up in the rom but other than that it's not clear if it was ever used.
$0225[0x0B] - free ram?

$0230[0x50] - Tentatively considered to be free ram at this time

$0280[0x0A] - special object OAM priority (full byte boolean)
If **nonzero**, use highest priority, **otherwise** use default priority.

$028A[0x0A] - Unknown special object variable
$0294[0x0A] - Movement related for special objects
$029E[0x0A] - Timers for special objects like Pendants (or altitude)
$02A8[0x0A] - Unknown special object variable

$02B2[0x0E] - free ram

$02C0[0x02] - bitfield possibly related to inroom staircases.
If this variable masked with **0x0070 is nonzero**, Link moves as though he's on an in-room south staircase
If this variable masked with **0x0007 is nonzero**, Link moves as though he's on an in-room north staircase

$02C2[0x01] - The value of **$5F** gets cached here sometimes, unclear what **$5F** is for though right now.

$02C3[**0x01**] - ????
$02C4[**0x01**] - ????

$02C5[**0x01**] - Related to **$46** (damage countdown timer)

$02C6[**0x01**] - ????

$02C7[**0x01**] - Probably countdown timer associated with **$29**

$02C8[**0x01**] - ????

$02C9[**0x01**] - Very limited usage only in **Bank 0x07**.

$02CA[**0x01**] - ????

$02CB[**0x01**] - Very limited usage only in Bank **0x07**.

$02CC[**0x01**] - Limited usage in **Bank 0x07** and **0x0D**.

$02CD[**0x02**] - Counter for the beginning of the game where Zelda telepathically contacts you over and over again.

$02CF[**0x01**] - It would appear that this is the index of the current tagalong state (out of 20 / 0x14 states)
Exact usage is not clear, though.

$02D0[**0x01**] - ????
$02D1[**0x01**] - Cache location for **$02D3**
$02D2[**0x01**] - ????
$02D3[**0x01**] - ????
$02D4[**0x01**] - Apparently a debug variable, doesn't appear to be used in the real game

$02D5[**0x01**] - free ram

$02D6[**0x01**] - ???? relates to tagalongs
$02D7[**0x01**] - ????
$02D8[**0x01**] - The index of the item you receive when you open a chest or pick up an item off the ground, or otherwise obtain the item (like Link getting the sword from his Uncle)

$02D9[**0x01**] - Related to **$02D8**, haven't worked out the details b/c it wasn't necessary yet.

$02DA[**0x01**] - Flag indicating whether Link is in the pose used to hold an item or not.

0 - no extra pose,
2 - holding up item with two hands pose (e.g. triforce),
everything else - holding up item with one hand pose

$02DB[**0x01**] - Flag that deals with Link warping with the magic mirror.... but not exactly sure what it means.

$02DC[**0x02**] - Mirror of Link's X coordinate which is used in **Bank 07**'s movement routines
$02DE[**0x02**] - Mirror of Link's Y coordinate which is used in **Bank 07**'s movement routines

$02E0[**0x01**] - Flag for Link's graphics set. (Mirrored at **$56**)

0 - Normal Link,
1 - Bunny Link.

$02E1[**0x01**] - Link is transforming? (Poofing in a cloud to transform into something else.)

$02E2[**0x01**] - Timer for when Link transforms between Link and Bunny modes.

$02E3[**0x01**] - ????

$02E4[**0x01**] - Flag that if set will not allow Link to move. Requires further research as to its generalized usage.
Also... Link cannot bring down the menu if this is **nonzero**.
Additionally, Euclid says that the value **0x1A** is written here after you kill Ganon. (And he's right)

$02E5[**0x01**] - Bitfield for chest interaction

uuuucccc

c - touching chest
u - free ram

$02E6[**0x01**] - free ram, but could be used as a bitfield for expanded tile types

$02E7[**0x01**] - Bitfield for big key locks and gravestone interactions

bbbbgggg

b - big key lock
g - gravestone

$02E8[**0x01**] - Bitfield for spike / cactus and barrier tile interactions?

bbbbssss

s - spike blocks / cactus tiles
b - orange / blue barrier tiles that are up

$02E9[**0x01**] - Item Receipt Method

0 - Receiving item from an NPC or message
1 - Receiving item from a chest
2 - Receiving item that was spawned in the game by a sprite
3 - Receiving item that was spawned by a special object.

$02EA[**0x02**] - Set to the tile type

$02EC[**0x01**] - seems to be a flag for (Link's) collision with bombs. Maybe other uses

$02ED[**0x01**] - If nonzero, it indicates that Link is near the tile
that triggers the Desert Palace opening with the Book of Mudora

$02EE[**0x01**] - Bitfield for spike floor and ????? tile interactions

ssssuuud

d - desert palace entrance trigger
s - spike floor tiles
u - not tested

$02EF[**0x01**] - Bitfield for dashable (breakable with dash) and ????? tile interactions

dddd????

d - dashable
u - not tested

$02F1[**0x01**] - ??? related to dashing. Set to **0x40** at start, counts down to **0x20**

$02F2[**0x01**] - Something to do with Kiki the monkey?

$02F4[**0x01**] - Only use is for caching the current value of **$0314** in some instances
$02F5[**0x01**] - **0** - Not on a Somaria Platform.
2 - On a Somaria Platform.

$02F6[**0x02**] - Bitfield for interaction with Blue Rupee, Grabbable, and Key Door tiles

rrrrrrrr kkkkgggg

k - Key Door tiles
g - Tiles grabbable by Hookshot
r - Blue Rupee tiles

$02F8[**0x01**] - Flag used to make Link make a noise when he gets done bouncing after a wall he's dashed into. Thus, it only has any use in relation to dashing.

$02F9[**0x01**] - best guess so far: zero if your tagalong is transforming, nonzero otherwise

$02FA[**0x01**] - Flag that is set if you are near a moveable statue (close enough to grab it)

$02FB[**0x05**] - free ram

==========

Page 0x03

==========

$0300[**0x01**] - Link's state changes? Happens when using boomerang.

$0301[**0x01**] - When **non zero**, Link has something in his hand, poised to strike. (Ice Rod, Hammer)
Bit 6 being set indicates that magic powder is being sprinkled

$0302[**0x01**] - ????

$0303[**0x01**] - In conjunction with the above variable when set to **0x13**, matching the above variable, the cape transformation is complete.

0x01 - Bombs
0x02 - Boomerang
0x03 - Arrows
0x04 - Hammer
0x05 - Fire Rod
0x06 - Ice Rod
0x07 - Bug catching net
0x08 - Flute
0x09 - Torch
0x0A - Magic Powder
0x0B - Bottle
0x0C - Book of Mudora
0x0D - Cane of Byrna
0x0E - Hookshot
0x0F - Bombos Medallion
0x10 - Ether Medallion
0x12 - Quake Medallion
0x13 - Cape
0x14 - Magic Mirror

$0304[**0x01**] - related to the magic cape transformation (a flag probably)

$0305[**0x01**] - (Potentially) Debug variable only seen in **Bank 07**.
If not equal to **0x01**, it will cause **$1E** and **$1F** to not be zeroed out every frame

$0308[**0x01**] - **Bit 7** is set when Link is carrying something.
Bit 1 set when Link is praying?

$0309[**0x01**] - **0**: nothing.
1: picking up something.
2: throwing something or halfway done picking up something

$030A[**0x01**] - Step counter used with **$030B**. Also, **$030A-B** seem to be used for the opening of the desert palace
$030B[**0x01**] - Animation timer for throwing and picking up items like rocks or signs
$030C[**0x01**] - ??? free ram?

$0310[**0x02**] - The Y velocity of a moving floor (Mothula's room)
$0312[**0x02**] - The X velocity of a moving floor (Mothula's room)

$0314[**0x01**] - The index (the X in **$0DD0** for example) of the sprite that Link is "touching"

$0315[**0x01**] - Seems to be a flag that is set to 0 if Link is not moving, and 1 if he is moving.
However it doesn't seem to get reset to zero.

$0316[**0x02**] - ???? bunny stuffs?

$0318[0x02] - related to moving floor
$031A[0x02] - related to moving floor

$031C[0x01] - tells us the actual graphic/state to use on the given step of a spin attack

$031D[0x01] - step counter for the spin attack
$031E[0x01] - used as an offset for a table to retrieve values for $031C. The offset comes in increments of four, depending on which direction Link is facing when he begins to spin. This makes sense, given that he always spins the same direction, and allows for reusability between the different directions, each one being a sub set of the full sequence.

$031F[0x01] - Countdown timer that when it's set causes Link's sprite to blink, i.e. flash on and off

$0320[0x02] - Bitfield for interaction with moving floor tiles

uuuuuuuu uuuummmm

m - Moving floor
u - free ram

$0322[0x01] - ????

$0323[0x01] - Mirror of $2F, which is an indicator for which direction you are facing.
Only used in the rendering of Link's OAM data in Bank 0x0D

$0324[0x01] - A flag telling a medallion spell it's okay to proceed with the effect.
If set to 1, the effect will wait until it is set to 0 to activate.

$0325[0x01] - free ram, with the caveat that it is cleared in a few locations in Bank 0x08

$0326[0x02] - ???? collision related?
$0328[0x02] - ???? collision related?

$032A[0x01] - seems to be nonzero when you hit a button to swim faster. zero otherwise

$032B[0x02] - ???? collision related?
$032D[0x02] - ???? collision related?

$032F[0x02] - ???? collision related?
$0331[0x02] - ???? collision related?

$0333[0x01] - Stores the tile type that the lantern fire or fire rod shot is currently interacting with.

$0334[0x02] - ???? collision related?
$0336[0x02] - ???? collision related?

$0338[0x02] - ???? collision related?
$033A[0x02] - ???? collision related?

$033C[0x02] - ???? collision related?
$033E[0x02] - ???? collision related?

$0340[**0x01**] - ????

$0341[**0x02**] - Bitfield for interaction with deep water and ??? tiles

uuuuuuuu ffffwwww

f - ??? (waterfall maybe?)
u - free ram
w - deep water

$0343[**0x02**] - Bitfield for interaction with normal tiles

uuuuuuuu uuuunnnn

n - normal tile
u - free ram

$0345[**0x01**] - Set to **1** when we are in water. **0** otherwise. Note this does not mean we are in shallow water. This means we are submerged.

$0346[**0x02**] - Exclusively used in **Bank 0x0D** for the purposes of drawing Link. This value gets bitwise ORed in to supply the palette bits of Link's sprite. It's only intended to take on one of two values:

0x000E, or **0x0000**.

0x0E00 indicates that palette 7 is being used, and
0x0000 indicates that palette 0 is being used.

Sometimes Link's palette swaps so that Link won't turn translucent when color addition is active. An example of this would be in the Flute Boy's meadow when he disappears. See **$0ABD** for more info.

$0348[**0x02**] - Bitfield for interaction with icy floor tiles

uuuuuuuu jjjjiiii

i - icy tile 1
j - icy tile 2 (distinction is not quite understood right now)
u - free ram

$034A[**0x01**] - Flag indicating whether Link is moving or not. (I think)

$034B[**0x01**] - Debug variable, as it is never read, and only written to in the equipment screen code

$034C[**0x02**] - Bitfield for the top of water staircase tile interactions

uuuuuuuu uuuussss

s - water staircase
u - free ram

$034E[**0x01**] - Definitely related to Cane of Somaria and how Link is displayed when on a Somaria platform

$034F[0x01] - If **nonzero**, Link does the harder stroke while swimming. This is triggered by pressing the A, B, or Y buttons while swimming, but ends shortly. It has no bearing on his swimming speed, though.

$0350[0x01] - free ram, though it would need to be reclaimed from the game engine as it's currently used in several places as an apparent debug variable

Water/Grass drawing variables:

$0351[0x01] - Value that if set to **1**, draws the water ripples around Link's body while standing in water. The drawing, of course, uses sprites.
If the value is **2**, then a patch of tall grass is drawn around Link instead.
Any value other than 2 (besides 0) produces the water effect.

$0352[0x02] - Used exclusively during writing Link's OAM data (**Bank 0x0D**) as an offset into the OAM buffer (**$0800**)

$0354[0x01] - Used exclusively during writing Link's OAM data for currently unknown purposes

$0355[0x01] - Secondary water grass timer. Increments when **$0356** reaches 9. Starts at 0 and reaches 3.

$0356[0x01] - Primary water/grass timer. Starts at zero and resets at 9.

$0357[0x02] - Bitfield for interaction with thick Grass / warp tiles

uuuuuuuu wwwwgggg

g - bits are for thick grass tiles
w - bits are for warp tiles (blue on OW, orange in dungeons)
u - free ram

$0359[0x02] - Bitfield for interaction with shallow water tiles

uuuuuuuu uuuuwwww

w - water tiles
u - free ram

$035B[0x02] - Bitfield for interaction with destruction aftermath tiles (bushes, rockpiles, etc)

uuuuuuuu uuuuaaaa

a - aftermath tiles
b - free ram

$035D[0x02] - Used exclusively during writing Link's OAM data (bank 0x0D). Bitwise ORed in for some of the OAM data.

$035F[0x01] - Seems to be a flag indicating that the boomerang is in play. It shows up in many places so far
Not even sure why it needs this flag... maybe hackish coding by Nintendo?

$0360[**0x01**] - A flag that when nonzero causes Link to be electrocuted when touching an enemy.
This seems counterintuitive to me, but whatever.

$0361[**0x01**] - free ram

$0362[**0x01**] - ????

$0364[**0x0?**] - ???? (relates to Link's sprites's object priority somehow?)

$0366[**0x02**] - Flag stating that Link is about to read something (assuming he's facing north)
Used with telepath tiles (dungeon), and signs (overworld)

$0368[**0x0?**] - ????

$036A[**0x01**] - ????

$036C[**0x01**] - Action index when interacting with tiles, like pots, rocks, or chests.

0 - ???
1 - Picks up a pot or bush.
2 - Starts dashing
3 - Grabs a wall
4 - Reads a sign.
5 - Opens a chest.
6 - ???
7 - ???

$036D[**0x01**] - Detection bitfield for vertical ledge tiles

dddduuuu

- d bits are for ledge tiles facing down
- u bits are for ledge tiles facing up

$036E[**0x01**] - Detection bitfield for horizontal ledge tiles and up + horizontal ledge tiles

ddddhhhh

- h bits are for ledge tiles facing left or right
- d bits are for ledge tiles facing up + left or up + right

$036F[**0x01**] - Detection bitfield for down + horizontal ledge tiles

uuuudddd

- d bits are for ledge tiles facing down + left or down + right
- free ram

$0370[**0x01**] - Bitfield for interaction with unknown tile types

bbbbaaaa

a - type **0x4C** and **0x4D** tiles (Overworld only)
b - type **0x4E** and **0x4F** tiles (Overworld only)

$0371[**0x01**] - Countdown timer for frames it will take Link to become tired pushing against something solid.
Once counted down, his appearance will look flushed and like he's dragging ass.
Resets once you stop pushing or moving.

$0372[**0x01**] - Flag indicating whether Link will bounce off if he touches a wall.

$0373[**0x01**] - Putting a **non-zero** value here indicates how much life to subtract from Link.
(quick reference: **0x08 = one heart**)

$0374[**0x01**] - Countdown timer for when Link is about to dash. When it reaches **0** he starts dashing. starts as **0x1D**

$0375[**0x01**] - This is the timer that is used to count down how long it takes before Link can jump off a ledge.
It is typically set to **19** (**0x13**) frames, though I don't believe it decrements every frame.

$0376[**0x01**] - **bit 0**: Link is grabbing a wall.
$0377[**0x01**] - ????

$0379[**0x01**] - ???

$037A[**0x01**] - ??? related to picking up bombs (see **Bank 07**)

$037B[**0x01**] - ???? related to cape transformation

$037D[**0x01**] - ???? related to link's bedspread

$037F[**0x01**] - Walk through walls and anytime warping in outdoors.

$036C - Indicates which type action Link is engaging in via the A button. Opening chest..., pulling, dashing, etc.

$0379 - Flag, if set, A button isn't read.
$037A - Puts Link in various positions, 1 - shovel, 2 - praying, etc... cane of somaria

$037B[**0x0?**] - If **nonzero**, disables Link's ability to receive hits from sprites. (But not pits)

$037E[**0x01**] - **0** - Hookshot is not dragging Link anywhere.
1 - hookshot is dragging Link somewhere

$037F[**0x01**] - Apparently if **zero**, all tile attributes are **$00** (nothing)

$0380[**0x02**] - (Possibly of size 5 instead)

$0385[**0x0A**] - special effect ???

$0394[**0x0A**] - special effect ???

$039F[**0x0A**] - Special Effect Timers

$03A4[**0x0?**] - Referenced in receive item initializer

$03B1[0x05] - countdown timer for special objects
(not that there are only 5 of these whereas there's usually 10 slots for special objects)

$03B6[0x04] - 16-bit values relating to bomb doors
$03BA[0x04] - 16-bit values relating to bomb doors
$03BE[0x02] - 8-bit values relating to bomb doors

$03C0[0x02] - Reserved for rock debris special object (0x08)
$03C2[0x02] - Reserved for rock debris special object (0x08)
$03C4[0x01] - Reserved for rock debris special object (0x08) and maybe bombs too.

$03C5[0x05] - Special Object ram.
$03CA[0x05] - Special Object ram.

$03CF[0x03] - Special Object ram.
$03D2[0x03] - Special Object ram.

$03D5[0x06] - Special Object ram.
$03DB[0x06] - Special Object ram.

$03E1[0x03] - Special Object ram.

$03E4[0x02] - tiles that bombs are touching on the floor

$03E9[0x01] - Flag that seems to set when moving gravestones are in play and puzzle sound is playing.

$03EA[0x0A] - special effect ???

$03EF[0x01] - Normally zero. If set to nonzero, it forces Link to the pose where he is holding his sword up.
One example of where this is used is right after Ganon is defeated.

$03F0[0x01] - ????

$03F3[0x01] - ????

$03F4[0x01] - Seems related to Cane of Somaria somehow

$03F5[0x02] - The timer for Link's tempbunny state.
When it counts down he returns to his normal state.
When Link is hit it always falls to zero.
Is always set to 0x100 when a yellow hunter (transformer) hits him.
If Link is not in normal mode, however, it will have no effect on him.
The value is given in frames, so if the value written is 0x80, you will be a bunny for 128 frames

$03F7[0x01] - Flag indicating whether the "poof" needs to occur for Link to transform into the tempbunny.

$03F8[0x0?] - Flag set if you are near a "pull for rupees" sprite

$03FA[0x02] - Relates to Link's OAM routine in Bank 0D somehow.
Appears to be the 9th bit of the X coordinate of some sprite(s).

$03FC[0x01] - ????

$03FD[0x01] - ???? related to bombs seemingly

$03FE[0x02] - free ram.

==========

Page 0x04

==========

For a break down on how room data gets saved, see the SRAM Document.

$0400

$0401 - Tops four bits: In a given room, each bit corresponds to a door being opened.
If set, it has been opened by some means (bomb, key, etc.)

$0402[0x01] - Certainly related to **$0403**, but contains other information I haven't looked at yet.

$0403[0x01] - Contains room information, such as whether the boss in this room has been defeated.
Loaded on every room load according to map information that is stored as you play the game.

Bit 0: Chest 1
Bit 1: Chest 2
Bit 2: Chest 3
Bit 3: Chest 4
Bit 4: Chest 5
Bit 5: Chest 6 / A second Key. Having 2 keys and 6 chests will cause conflicts here.
Bit 6: A key has been obtained in this room.
Bit 7: Heart Piece has been obtained in this room.

$0405[0x01] - This never seems to be written (free ram?)

$0406[0x02] - free ram

$0408[0x02] - ????

$040A**[0x01] - Area Index for the Overworld. Although there are **$C0 different possible values, the larger maps seem to take up more of these values so that coordinate addresses can be more easily calculated, and scrolling eliminated at times.

$040B[0x01] - free ram

$040C[0x02] - Map index for dungeons. If it = **0xFF**, that means there is no map for that area.
There is a data table of values, probably depends on the entrance you go in how this gets selected.
Notably used in Ganon's Tower for minibosses. Note that the value held here is the dungeon's intuitive index multiplied by 2. Hence this value is always even.

$040E[0x01] - Contains the layout and starting quadrant info (from dungeon header)

$040F[0x0?] - free ram.

$0410[0x02] - -------- ----udlr Screen transition direction bitfield

When the overworld is transitioning screens, only one of the "udlr" bits will be set.

$0412[**0x02**] - Index used during screen transitions to gradually, over the course of several frames, transmit data to vram. See variables **$19** and **$0118**

$0414[**0x02**] - BG2 properties in Hyrule Magic

Detailed description of CGADDSUB properties per type:

Note that both parallax modes are the same from the register standpoint
- **0** - "Off" - Main: BG2, BG3, Obj; Sub: None; +/-: background
- **1** - "Parallaxing" - Main: BG2, BG3, Obj; Sub: BG1; +/-: background
- **2** - "Dark Room" - Main: BG2, BG3, Obj; Sub: BG1, BG1; +/-: background
- **3** - "On top" - Main: BG1, BG2, BG2, Obj; Sub: None; +/-: background
- **4** - "Translucent" - Main: BG2, BG3, Obj; Sub: BG1; +/-: (1/2 +) back. BG1
- **5** - "Parallaxing 2" - Main: BG2, BG3, Obj; Sub: BG1; +/-: background
- **6** - "Normal" - Main: BG2, BG3, Obj; Sub: BG1; +/-: background
- **7** - "Addition" - Main: BG2, BG3, Obj; Sub: BG1; +/-: (full +) back. BG1

$0416[**0x02**] - ------------udlr Screen transition direction bitfield 2
When the overworld is transitioning screens, only one of the "udlr" bits will be set.

$0418[**0x02**] - **0** - Overworld screen transition up
1 - Overworld screen transition down
2 - Overworld screen transition left
3 - Overworld screen transition right

Values greater than 3 should not show up

$041A[**0x02**] - vvvvvvvv vvvvvvid

d - disable horizontal and vertical
i - inverts the direction of the floor movement, specified by **$0310** / **$0312**
v - if any of these bits are set, the floor movement will be vertical. Otherwise it will be horizontal

$041C[**0x02**] - ??? referenced in trick hidden walls

The value of this is used by various "Effect" settings in the dungeons

$041E[**0x02**] - ???

$0422[**0x02**] - X offset of the moving floor. (Also used to position the crystal maidens during the 3D sequence)
$0424[**0x02**] - Y offset of the moving floor. ""

$0426[**0x02**] - free ram

$0428**[**0x01**] - Mirror of **$AD, which is "Collision" in Hyrule Magic. This is an independent flag that determines how scrolling of the BGs occurs in the dungeon submodule.

$0429[**0x01**] - free ram?

$042A[**0x02**] - Used with Hidden walls? Maybe other uses. See **Bank $01**

$042C[0x02] - Types that use this index: moveable blocks, pots and other liftable objects, breakable floors, and moles. Collectively the limit for these types of objects is 16 per room.

Notes about cracked floors so I don't go insane!
*** The breakable floor that is first in the objectlist is the the one that will open up. ***
(*Do ctrl-B on a cracked floor among others in Hyrule Magic if you don't believe me*).
Okay... so why does only one cracked floor open up?
It doesn't make any sense right? The tile type is 62 for all cracked floors post load.

$042E[0x02] - Index of torches in the room (since blocks are loaded first this value gets updated after the fact.)

$0430[0x02] - Flag that is **nonzero** when Link has triggered a floor switch and is still standing on it.
When he walks away from it, this flag will reset to **zero**.

$0432[0x02] - number of star shaped switches in a room

$0434[0x02] - free ram?

$0436[0x01] - ????

$0437[0x01] - ????

$0438[0x02] - number of in-floor inter-room up-north staircases ..... (**1.2.0x2D**)
$043A[0x02] - number of in-floor inter-room down-south staircases ..... (**1.2.0x2E**, **1.2.0x2F**)
$043C[0x02] - number of in-room up-north staircases ..... (**1.2.0x30**)
$043E[0x02] - number of in-room up-north staircases ..... (**1.2.0x31**)
$0440[0x02] - number of inter-pseudo-bg up-north staircases ..... (**1.2.0x32**)
$0442[0x02] - number of in-room up-north staircases ..... (**1_2.0x33**) (*for use in water rooms*)
$0444[0x02] - number of activated water ladders ..... (**1.2.0x35**)
$0446[0x02] - number of water ladders ..... (**1.2.0x36**)

$0448[0x02] - This is tracking variable for dungeon objects with the catch that
it is only used in rooms that have water objects (*not 100% sure of this*.)
But it seems to be the activator that converts normal inter-bg type staircases
to ones that let you jump into water, like in the Swamp Palace.

$044A[0x02] - Something to do with in-room staircases, but I'm otherwise clueless

$044E[0x02] - number of different kind of doors? (*related to **$0460** I think*)

$0450[0x02] - number of Type **0x14** (*10 decimal in HM*) doors. Also see **$06D0**

$0452[0x02] - ???? seen used with bombable walls (*the big ones*) but otherwise, not sure

$0454[0x02] - Seems to indicate the state of a large wall being bombed open. Ranges from **0x01 to 0x15**
The numerical value probably is used to pick which section of the wall to bomb open.

$0456[0x02] - Used by doors system? Not sure how exactly.

$0458[0x02] - **0** - In a dark room, **1** - you're in a dark room and have the lantern

$045A[0x01] - Seems to be the number of torches that are lit in a dark room.
Torch objects during load can be set to be permanently lit, so this can affect how the room's lighting behaves.

$045B[0x01] - ????

$045C[0x02] - ????

$045E[0x02] - free ram

$0460[0x02] - Number of Locked doors? (*range 0 - 3?)* Number of doors that have been loaded in the room?

$0462[0x02] - ??? related to *submodule 12 of modules 7 and 9*
With staircases, indicates a value of **0 to 7** of which staircase it is.

$0466[0x02] - Seems to be related to trap doors or switches somehow...

$0468[0x02] - Flag that is set when trap doors are down.

$046A[0x02] - Similar to **$0490**, this is Floor 2 in Hyrule Magic

$046C[0x01] - "Collision" in Hyrule Magic.

$046D[0x05] - free ram

$0470[0x02] -

$0472[0x02] - ???? related to watergate special routine

$0474[0x02] - Limited use between Bank **0x01** and **0x07**.

$0476[0x02] - Pseudo bg level.
Indicates which "level" you are on, either BG1 or BG2. BG1 is considered 1 in many cases. However, there is no need for BG1 necessarily. When Link can interact with BG1, this value should match **$00EE**, I think.
This mostly applies to staircases in rooms that only use one BG to interact with.

$0478[0x02] - ????

$047A[0x02] - ????

$047C[0x02] - Used with doors.

; checked first?

$047E[0x02] - number of wall up-north spiral staircases ..... (**1.2.0x38**)
$0480[0x02] - number of wall down-north spiral staircases ..... (**1.2.0x39**)
$0482[0x02] - number of wall up-north spiral staircases ..... (**1.2.0x3A**)
$0484[0x02] - number of wall down-north spiral staircases ..... (**1.2.0x3B**)

$0486[0x02] - ????
$0488[0x02] - ????

$048A[0x02] - Relates to the plane variables (**$063C-$0640**)

$048C[0x02] - ????

$048E[0x02] - Room index in dungeons?

$0490[0x02] - In a dungeon room, provides the type of filler tiles (*Floor 1 in Hyrule Magic*)

$0492[0x02] - ????
$0494[0x02] - ????

$0496[0x02] - number of chests in the room
$0498[0x02] - number of big key lock blocks in the room

$049A[0x02] - number of **1.3.0x1B** objects
$049C[0x02] - number of **1.3.0x1C** objects
$049E[0x02] - number of **1.3.0x1D** objects

$04A0[0x01] - Countup timer that stops at 0xC0. While running, the floor that Link is on in a dungeon is displayed.
$04A1[0x01] - free ram, probably
$04A2[0x02] - number of in-wall inter-room up-north straight staircases ..... (**1.3.0x1E**, **0x26**)
$04A4[0x02] - number of in-wall inter-room up-south straight staircases ..... (**1.3.0x20**, **0x28**)
$04A6[0x02] - number of in-wall inter-room down-north straight staircases ..... (**1.3.0x1F**, **0x27**)
$04A8[0x02] - number of in-wall inter-room down-south straight staircases ..... (**1.3.0x21**, **0x29**)

$04AA[0x02] - (conflict) Flag, that if **nonzero**, tells us to load using a starting point entrance value rather than a normal entrance value.

$04AA[0x01] - This is used when you die and choose to save & continue
It is the dungeon entrance to put Link into.
This variable is only used if you die in a dungeon, and is set to the last dungeon entrance you went into.

$04AB[0x01] - free ram

$04AC[0x02] - When most tiles are changed in a particular overworld screen this value is incremented by 2 and **$7EF800** and **$7EFA00** are updated to contain the address of the modification and the tile (map16) used replace it. This only comes into play when a warp between the DW or the LW fails and you have to warp back.

$04AE[0x01] - number of in-room up-south staircases ..... (**1.3.0x33**) (*for use in water rooms*)

$04B0[0x02] - ???? Relates to object type **1.1.0x35**
Which, by the way, seems to be unused... and buggy.

$04B2[0x01] - Related to shovel, but not entirely clear yet how.

$04B4[0x01] - see overworld module submodule 0
$04B5[0x01] - ????

$04B6[0x02] - Position in tile attribute buffer where trigger tile was hit?

$04B7[0x01] - ????

$04B8[0x02] - Flag that indicates you are close to a big key door, and the text message saying you don't have a key has already triggered. It resets when you move away from the door.
Also works outdoors when a door tells you you can't enter with stuff following you.

$04BA[0x02] - Index of overlay to load in a dungeon room (in response to an event, typically)

$04BC[0x02] - Seems to be a toggle between two different states of holes in a room. (For rooms that can switch back and forth?)

$04BE[0x01] - Countdown timer for trinexx red dragon head palette transitioning
$04BF[0x01] - Countdown timer for trinexx blue dragon head palette transitioning

$04C0[0x01] - Relates to trinexx red dragon head palette transition state
$04C1[0x01] - Relates to trinexx blue dragon head palette transition state

$04C2[0x01] - Relates to giving of critical items after boss fights...

$04C3[0x01] - free ram

$04C4[0x01] - Number of credits left for opening minigame chests **0xFF** if entirely disabled

$04C5[0x01] - State dealing with Ganon's fight = **2** - you can hit him,
1 - he's translucent,
0 - he's invisible.

$04C6[0x01] - Trigger for special animations
0 - nothing
1 - Dark Palace entrance
2 - Skull Woods entrance (when it burns)
3 - Misery Mire entrance
4 - Turtle Rock entrance
5 - Ganon's Tower entrance

$04C7[0x01] - Appears to serve as a barrier variable for tag related logic in Dungeon rooms when set.
Also prevents traveling on staircases and other such things.

$04C8[0x02] - Only used in peg puzzles as an index of which or how many peg tiles have been hammered down
In the case of the Light World Death Mountain peg puzzle it's an index of how many successful steps have been completed, and is actually twice the number of pegs that have been hammered
In the case of the Dark World peg puzzle near the ruined Smithy house it's just the number of pegs hammered so far

$04CA[0x01] - When you are low on life, this is used as a timer
It loops between **$1F** down to **$0**. When it reaches **zero**, the low life beep happens

$04CB[0x15] - free ram.

$04F0[0x10] - Timers for Torches (**byte entries**). When lit, starts with a value of **0xFF** and counts down to **0**.
Setting it before the torch is lit is a bad idea - it will not cause a torch to light, nor will it brighten the room. Note that this range indicates we can have up to 16 torches in an area.

==========

Page 0x05

==========

$0500[0x20] - replacement tile attributes for moveable/liftable/poundable objects.
$0520[0x20] - Object's position in the object data itself (multiplied by 3)
$0540[0x20] - Object's tile map position
$0560[0x20] - replacement tile map value (upper left 8x8 tile)
$0580[0x20] - replacement tile map value (lower left 8x8 tile)
$05A0[0x20] - replacement tile map value (upper right 8x8 tile)
$05C0[0x20] - replacement tile map value (lower right 8x8 tile)

$05E0[0x10] - see routine **$3ED3F**
$05F0[0x10] - ""

$05FC[0x02] - Two byte array. Each one contains the index of a dungeon object that could potentially be changed. Examples include pots and moveable blocks.

==========

Page 0x06

==========

$0600[0x02] - ??? These four are Y coordinate related
$0602[0x02] - ??? "
$0604[0x02] - ??? "
$0606[0x02] - ??? "

$0608[0x02] - ???? These four are X coordinate related
$060A[0x02] - ????
$060C[0x02] - ????
$060E[0x02] - ????

$0610[0x02] - ??? Up room transition scroll target
$0612[0x02] - ??? Down ""
$0614[0x02] - ??? Left ""
$0616[0x02] - ??? Right ""

Camera Variables:

$0618[0x02] - Y coordinate of the scrolling camera. Probably the lower bound for scrolling.
$061A[0x02] - Y coordinate of the upper bounds of scrolling.
$061C[0x02] - X coordinate of the lower bounds of scrolling.
$061E[0x02] - X coordinate of the upper bounds of scrolling.

Ending Sequence Variables:

$0620[0x02] - ????
$0622[0x02] - ????

$0624[0x02] - ???
$0628[0x02] - ???

$062A[**0x02**] - ???

$062C[**0x02**] - In loading dungeons, contains the upper even byte of the BG0 H scroll reg.
$062E[**0x02**] - In loading dungeons, contains the upper even byte of the BG0 V scroll reg.
note: *upper even byte refers to the operation (argument & 0xFE00)*

BG3 V-IRQ Values:

$0630[**0x01**] - During V-IRQ or H-IRQ (not sure if this game uses H-IRQ), this are the values of the BG3 Hscroll Register.
$0631[**0x01**] - This is the upper byte of the previous address.

$0632[**0x03**] - free ram

$0635[**0x01**] - ????

$0636[**0x01**] - Something to do with the Overworld map.

$0637[**0x01**] - During Attract Mode, is a timer for the Mode 7 zoom in sequence.
$0638[**0x02**] - Mirror of $211F (M7X)
$063A[**0x02**] - Mirror of $2120 (M7Y)

$063C[**0x01**] - Hole / Teleporter plane
$063D[**0x01**] - Staircase 1 plane
$063E[**0x01**] - Staircase 2 plane
$063F[**0x01**] - Staircase 3 / door plane
$0640[**0x01**] - Staircase 4 / door plane

$0641[**0x01**] - Flag that is nonzero when a moveable block that triggers something is pushed in a dungeon room.
$0642[**0x01**] - apparently a flag for indicating a state change in water puzzle rooms. (And hidden wall rooms?)

$0643[**0x03**] - free ram?

$0646[**0x01**] - **Educated guess**: *this is a flag that is nonzero when a cane of Somaria block is on top of a switch that needs to be weighed down.*

$0647[**0x01**] - Has to do with mosaic and its effect in dungeons?

$0648[**0x28**] - free ram

$0670[**0x10**] - presumably all related to spotlight hdma or the setup of it
$0670[**0x02**] -
$0672[**0x02**] - free ram
$0674[**0x02**] -
$0676[**0x02**] -
$0678[**0x02**] -
$067A[**0x02**] -
$067C[**0x02**] -
$067E[**0x02**] -

$0680[**0x10**] - hurp
$0682[**0x02**] -
$0684[**0x02**] - ???

$068C[**0x02**] - Top 4 bits hold information about which doors have been opened

$068E[**0x02**] - ???? related to trap doors and if they are open ; possibly bomb doors too?
Update: *module 0x07.0x4 probably uses this to know which key or big key door to open*

$0690[**0x02**] - ???? related to trap doors and if they are open

$0692[**0x02**] - (Overworld) Contains the resultant map16 value of the most recently modified map16 tile.
(i.e. when picking up a bush/rock)
(Dungeon) Relates to doors being opened or closed

$0694[**0x02**] - ????

$0696[**0x02**] - Entrance value. If 0x0000 indicates no doorway on OW.
Values < 0x8000 indicate tile map coordinates for a wooden doorway.
0xFFFF indicates there is no doorway and you will come out facing the opposite direction (see north exit of bottle house in Kakkariko). Possibly also used for pits? Values >= 0x8000 indicate a special type of door + coordinates ranging from 0x0000 to 0x1FFF, found by **bitwise AND** with the value and 0x1FFF.
(i.e. if(b >= 0x8000) { tilemapAddr = b & 0x1FFF ; type = special; }
This is also the thing HM seems to have so much trouble editing.

$0698[**0x02**] - When lifting a big rock, this is the starting address of where the hole graphic will get stored to

$069A[**0x01**] - ????

$069E[**0x01**] - ????
$069F[**0x01**] - ????

$06A0[**0x0?**] - special object slots for stars type **1.2.0x1F**

$06B0[**0x08**] - special object slots for type **1.3.0x1E**,**0x1F**,**0x20**,**0x21**
and **1.2.0x2D**,**0x2E**,**0x2F**,**0x38**,**0x39**,**0x3A**,**0x3B** objects

$06B8[**0x08**] - special object slots for type **1.3.0x1B** and **1.2.0x30**,**0x31**,**0x32**,**0x33**,**0x35**,**0x36**

$06C0[**0x10**] - special door slots for type **0x16** doors

$06D0[**0x08**] - special door slots for type **0x14** doors

$06E0[**0x0C**] - Stores tile map positions of chests, big key chests, and big key locks
The array is 12 bytes long, and you can only have 6 chests in one room.
(Because of the layout of save game ram and associated code)

Note: If the top bit is set, it's a Big Key lock, otherwise it's one of the chest types.

$06EC[**0x0C**] - special object slots for **1.3.0x1C**,**0x1D**,**0x33**

$06F8[**0x08**] - free ram

==========

$0700[0x02] - Generally is equal to the area number you are in currently in times two.
Only bottom byte is used, and consists of the following pattern:

yyyzxxx0

y - obtained by masking Link's Y coordinate ($20) with 0x1E00, shifting left three times
x - obtained by masking Link's X coordinate ($22) with 0x1E00, and bitwise ORing it with the the y bits.
z - this is the overlap of the x and y bits described above.

$0702[0x06] - free ram

$0708[0x02] - ???? I'm too lazy to document these 4 values, but essentially they're masks that determine
$070A[0x02] - ???? the size of each overworld "block"
$070C[0x02] - ????
$070E[0x02] - ????

$0710[0x01] - Global flag for graphics routines. Detailed usage undocumented as of yet.

$0712[0x02] - If nonzero, seems to indicate that it's a smaller overworld map

$0714[0x02] - Cache of previous dimension setting (from $0712)

$0716[0x01] - Forms right and bottom bounding value for where scroll activates for player?

$0718[0x08] - Free ram

$0720[0x02] - Flag that if raised means to move to the next line. When text scrolls, this flag also raises but text remains on the 3rd line unless some command forces it to another line.
(e.g. [0x01], [2] in the monologue editor.)

$0722[0x02] - Used to indicate which line the VWF is generating text on. (values = 0, 2, or 4)

$0724[0x02] - Used to step through $7EC230[0xC0?] in VWF text generation (start values are 0, 0x40, or 0x80)

$0726[0x02] - Base position in $7F0000[0x7E0]. Only has 3 possible values (0, 0x02A0, and 0x0540).
These correspond to the current line the game is generating text on. It is held constant while an individual line is rendering.

$0728[0xD8] - Free ram

==========

Page 0x08, 0x09 and 0x0A

==========

OAM Basic 512 byte table:

$0800[0x200] - OAM data. This is blitted to VRAM every frame via DMA.

How OAM works:

Byte1: X coordinate on screen in pixels. This is the lower 8 bits. See Extended OAM table below
Byte2: Y coordinate on screen in pixels.
Byte3: Character number to use. This is the lower 8 bits. See Byte 4
Byte4: vhoopppc (source: Qwertie's guide)

v - vertical flip
h - horizontal flip
p - priority bits
c - the 9th (and most significant) bit of the character number for this sprite.

Extended OAM 32 byte table:

$0A00[**0x20**] - Each byte contains information for 4 sprites (in the same order as the normal OAM table. For each sprite:

bit0 : size toggle bit. (this can mean 8x8 or 16x16, or 8x8 or 32x32, etc.
bit1 : 9th (and most significant) bit of the X coordinate.

$0A20[**0x80**] - Apparently contains bits of data to combine and write to $0A00 to $0A1F later. Thus the individual properties are stored in this array, and ORed in later to form the data that will get blitted to VRAM.

Apparently Palette Related, but details lacking:

$0AA0 - ???

$0AA1[**0x01**] - Main Tile Theme index.
Altering this has far more effect than $0AA2.
There are 0x25 (37) main BG tile themes.

$0AA2[**0x01**] - Auxiliary Tile Theme index.
There are 0x52 (82) auxiliary BG tile themes.

$0AA3[**0x01**] - Sprite Graphics index. Note that when in a dungeon, this value is the number found in the dungeon header, plus $40. Overworlds do not have this complication.

$0AA4[**0x01**] - Misc. Sprite Graphics index. Valid values are: 0x01, 0x0A, or 0x0B.
0x01 - Light world Overworld
0x0A - Dungeons
0x0B - Dark World Overworld

All other values are so far assumed to decompress the wrong type of graphics.
I.e., not all the graphics in the rom are de facto compressed. Some are just stored in a proprietary 3bpp format.

$0AA5[**0x01**] - free ram

$0AA6[**0x02**] - Unused? (but referenced in a few places). It's location would indicate that at one point it would have been a variable used to configure graphics.

$0AA8[**0x02**] - ***Note:*** *typically only the high byte (**$0AA9**) of this variable is modified.*
By design this variable is only ever set to two values: **0x0000** and **0x0200**.
It's used during palette loading to select between the auxiliary palette buffer (**$7EC300**) and

the main palette buffer ($7EC500) as targets to write colors to.
Only really used in Bank $1B

$0AAA[0x01] - Confusing variable, relates to $0FC6 somehow

$0AAB[0x01] - Free ram?

$0AAC[0x01] - Used to load SP-0 (first half) Usually inanimate object sprites. Valid values are 0x00 to 0x0B
$0AAD[0x01] - Used to load SP-5 (first half) Usually for sprites specific to an area / room. Valid values are 0x00 to 0x17
$0AAE[0x01] - Used to load SP-6 (first half) Usually for sprites specific to an area / room. Valid values are 0x00 to 0x17

$0AAF[0x01] - Free ram

$0AB0[0x01] - So far only seen referenced in an unreferenced palette routine.
Thus, can be considered Free RAM for now.

$0AB1[0x01] - Used to load SP-6 (second half) Used for the palette for throwable objects

$0AB2[0x01] - Used to select the first palette in CGRAM (the screen's background is its first color)
As far as I know, this value is only 0 or 1.
1 seems to be for special modes like the ending sequence.
The first palette is by default in LTTP used for the HUD.

$0AB3[0x01] - Selects BP-2 through BP-6 (first halves)
0 - Light World
1 - Dark World
2 - Light World Death Mountain
3 - Dark World Death Mountain
4 - Used during history mode
5 - Seems to be used during initialization

$0AB4[0x01] - Selects BP-2 through BP-4 (second halves). Values 0x00 to 0x13 are valid.
Only used during loading the overworld and the title screen.
$0AB5[0x01] - Selects BP-5 through BP-7 (second halves). Values 0x00 to 0x13 are valid.
Only used during loading the overworld and the title screen.

$0AB6[0x01] - Controls all 6 of the 4bpp background palettes (in dungeon mode) (BP-2 through BP-7)
The 2bpp background palette controller is $0AB2 above.

$0AB7[0x01] - While only referenced as being cached to $7EC20C, it is seemingly unused in practice

$0AB8[0x01] - Selects BP-7 (first half). Only valid for values 0x00 to 0x0D

$0AB9[0x04] - Free ram

$0ABD[0x01] - Used in order to swap palettes under certain special circumstances.
Apparently related almost entirely to the flute boy ghost and the ponds of wishing.
When zero, doesn't induce any behaviour change, but when nonzero, it will cause
SP-0 and SP-7 (full) to swap and SP-5 and SP-3 (second halves) to swap.

$0ABE[0x02] - Free ram

$0ABF[0x01] - Set to zero when an event is initialized, and will be set to 1 the next time a change of overworld area occurs. *This is used to trigger the magic powder showing up in the Witch's hut, as well as the finishing of your sword being tempered. Also used to make sure you didn't cheat during the heart piece maze game in Kakkariko.*

DMA Variables: To see these in action, look up routine $9E0 in the banks files. These are all Word values.

$0AC0[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored here is grabbed from a table indexed by address $0107.
$0AC2[0x02] - Also a ROM Address for DMA transfers, usually is 0x180 higher than $0AC0

$0AC4[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored here is grabbed from a table indexed by address $0108.
$0AC6[0x02] - Also a ROM Address for DMA transfers, usually is 0xC0 higher than $0AC4

$0AC8[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored here is grabbed from a table indexed by address $0109.
$0ACA[0x02] - Also a ROM Address for DMA transfers, is a fixed distance from $0AC8 which is also determined using a table indexed by $0109.

$0ACC[0x02] - ROM Address for certain DMA transfers from Bank $10.
The value stored here is grabbed from a table indexed by address $0100
$0ACE[0x02] - Also a ROM Address for DMA transfers, usually is 0x200 higher than $0ACC

$0AD0[0x02] - ROM Address for certain DMA transfers from Bank $10.
The value stored here is grabbed from a table indexed by address $0100
$0AD2[0x02] - Also a ROM Address for DMA transfers, usually is 0x200 higher than $0AD0

$0AD4[0x02] - ROM Address for certain DMA transfers from Bank $10.
The value stored here is grabbed from a table indexed by address $0102

$0AD6[0x02] - ROM Address for certain DMA transfers from Bank $10.
The value stored here is grabbed from a table indexed by address $0104

$0AD8[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored here is grabbed from a table indexed by address $02C3
$0ADA[0x02] - Also a ROM Address for DMA transfers, usually is 0x100 higher than $0AD0

$0ADC[0x01] - ROM Address for certain DMA transfers from Bank $7E.
The value stored here is determined like this: the value at $7EC00F + the fixed value 0xA680.

$0ADE[0x02] - free ram

$0AE0[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored here is determined like this: the value in the table at $0085DE indexed by $7EC015 + the fixed value 0xB280.
$0AE2[0x02] - Also a ROM Address for DMA transfers, usually is 0x100 higher than $0AE0

$0AE4[0x04] - free ram

$0AE8[0x02] - Used as an offset for $0AEC.

$0AEA[0x02] - Used as an offset for $0AF0
$0AEC[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored is determined like this: the value at $0AE8 + the fixed value 0xB940
$0AEE[0x02] - Also a ROM Address for DMA transfers, usually is 0x200 higher than $0AEC

$0AF0[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored is determined like this: the value at $0AEA + the fixed value 0xB940
$0AF2[0x02] - Also a ROM Address for DMA transfers, usually is 0x200 higher than $0AF0

$0AF4[0x02] - Used as an offset for $0AF0

$0AF6[0x02] - ROM Address for certain DMA transfers from Bank $7E.
The value stored is determined like this: the value at $0AF4 + the fixed value 0xB540
$0AF8[0x02] - Also a ROM Address for DMA transfers, usually is 0x200 higher than $0AF6

$0AFA[0x06] - free ram

==========

Pages 0x0B and 0x0C

==========

Overlord Model (Most if not all arrays are 8 bytes in length)

$0B00[0x08] - Overlord types for this room.

Overlord Catalog:

0x00 - Cannon Room
0x01 - Metal Ball Generator?
0x02 - Cannon Room
0x03 - Cannon Balls?
0x04 - Snake Trap
0x05 - Stalfos Trap
0x06 - ???
0x07 - Moving Floor
0x08 - Transformer (HM name) but I think it's the blobs that fall from the ceiling in the room right before the boss of Misery Mire.
0x09 - Wallmaster Overlord
0x0A - Falling tiles Overlord
0x0B - Falling tiles Overlord 2
0x0C - Falling tiles Overlord 3
0x0D - Falling tiles Overlord 4
0x0E - Falling tiles Overlord 5
0x0F - Falling tiles Overlord 6
0x10 - Fish producer (facing right) used in Swamp Palace
0x11 - Fish producer (facing left) used in Swamp Palace
0x12 - Fish producer (facing down) used in Swamp Palace
0x13 - Fish producer (facing up) used in Swamp Palace
0x14 - Tiles that rise out of floor overlord
0x15 - Wizzrobe spawner
0x16 - Small vermin that come out of bombed out cave walls
0x17 - Pot Trap

0x18 - Stalfos that Materialize (a la Eastern Palace)
0x19 - Armos Knights Handler
0x1A - ???

$0B08[0x08] - Overlord X coordinate lower byte
$0B10[0x08] - Overlord X coordinate upper byte
$0B18[0x08] - Overlord Y coordinate lower byte
$0B20[0x08] - Overlord Y coordinate upper byte
$0B28[0x08] - ????
$0B30[0x08] - Overlord countdown timer
$0B38[0x08] - ????
$0B40[0x08] - Overlord floor selector
$0B48[0x10] - Overlord's offset in the overworld sprite position buffer (see $7FDF80)

$0B58[0x10] - Timers for stunned enemies. Counts down from 0xFF.

$0B68[0x01] - Location to cache Link's or sprite's floor status.
It seems it later gets used with special objects in Bank 0x08.

$0B69[0x01] - Variable that tutorial soldiers and Blind use. Since it's not reinitialized when you save and quit, it can trigger a glitch where the Tutorial Soldiers will start saying things that belong to other characters or entities in the game. Spawned a Gamefaqs thread.

$0B6A[0x01] - A variety of sprites use this, including Blind, Statue Sentries, and the sprites that block the way into the Desert Palace.

$0B6B[0x10] - ???

$0B7B[0x01] - Flag indicating whether Link can move or not. Set to 1 to prevent him from moving.
$0B7C[0x02] - ???
$0B7E[0x01] - ???
$0B7F[0x01] - related to a routine in Bank $1E... (sprites 0xEF, etc.)

$0B80[0x08] - Seems to be a memory of the past 4 rooms you've visited (dungeon mode only)

$0B96[0x01] - Used by tutorial soldiers to cycle through messages (0x00 through 0x06, but has 0x0F added to it).
Also used by Blind the Thief when he's hit.
If after being Blind one saves and continues and starts a new game, there's a strong likelihood that the tutorial soldiers will say the wrong messages, if the value of this variable was more than 0x07 after beating Blind.

$0B89[0x10] - Object priority stuff for sprites?

$0B99[0x01] - Related to shooting gallery guy?
$0B9A[0x01] - Related to shooting gallery guy?

$0B9B[0x01] - ??? used with "dash item"?

$0B9C[0x02] - ??? Relates to dungeon secrets

$0B9E[0x01] - Relates to snake trap and "pot trap" overlords.

$0B9F[0x01] - free ram

$0BA0[0x10] - undocumented sprite variable (shows up a lot, though)

$0BB0[0x10] - For sprites that interact with special objects (arrows in particular), the special object will identify its type to the sprite via this location.

$0BC0[0x10] - contains the index of the sprite (i.e. its position in the $0E20[0x10] array only seems to be modified in the initial dungeon loading routine (room transitions don't appear to write here.)

$0BD0[0x10] - free ram

$0BE0[0x10] - prize pack for a sprite in the sprite object model (see below)

$0BF0[0x0A] - special effect something or other
$0BFA[0x0A] - Special Effect Y Coordinate Low Byte
$0C04[0x0A] - Special Effect X Coordinate Low Byte
$0C0E[0x0A] - Special Effect Y Coordinate High Byte
$0C18[0x0A] - Special Effect X Coordinate High Byte
$0C22[0x0A] - Helps trigger the Ether effect. (probably has more general usage)
$0C2C[0x0A] - ???
$0C36[0x0A] - With bombs... seems to be a fractional (partial pixel) part of Y movement
$0C40[0x0A] - ???
$0C4A[0x0A] - (may have multiple uses) byte array that tells us what special effects are happening.
c.f. 0x18 - ether effect. But sprites also seem to use it... weird.

$0C54[0x0A] - array that might contain info about steps for effects indicated in
$0C5E[0x0A] - Array that contains an item index to give to Link. e.g. 0x38 for the pendant of power.
$0C68[0x0A] - unknown
$0C72[0x0A] - special effect (only known application so far is bomb's direction when laid)
$0C7C[0x0A] - special effect floor selector (BG1 or BG0). Analogue for sprite objects would be $0F20[0x10]
$0C86[0x0A] - free ram?
$0C90[0x0A] - number of sprites the special effects uses * 4

The Sprite Object Model. All arrays are 16 bytes long since there are 16 sprites per room.

Note: also see $0BE0

$0C9A[0x10] - Room or Area number that the sprite has been loaded to. (If in a dungeon, only contains the lower byte)

$0CAA[0x10] - Bit 0: disabled???
Bit 1: ????
Bit 2: If set, makes sprite impervious to sword and hammer type attacks
Bit 3: if set, makes the sprite collide with less tiles than usual
Bit 4: ????
Bit 5: ????
Bit 6: Same as bit 7 in some contexts (zora in particular)
Bit 7: If set... creates some condition where it may or may not die

$0CBA[0x10] - If this is the following when the sprite dies, then:
0x00 = nothing happens.

0x01 = leaves a normal key.
0x03 = single green rupee.
anything else: Big Key

$0CCA[0x08] - Area that an overlord was loaded during

$0CD2[0x10] - How much damage a sprite can do (credit: Euclid)

$0CE2[0x10] - When the sprite is hit, this is written to with the amount of damage to subtract from the sprite's HP.

$0CF2[0x01] - Damage type determiner

$0CF3[0x01] - free ram

$0CF4[0x01] - Activates bomb or snake trap overlords when set to a nonzero value.

$0CF5[0x02] - Related to palace map submodule, but otherwise I dunno.

$0CF7[0x01] - More luck related crap?

$0CF8[0x01] - Used in Bank 07 as a temporary variable for picking a sound effect with appropriate panning

$0CF9[0x01] - Luck (credit: assassin17)

$0CFA[0x01] - Luck Kill Counter. When this reaches 10 luck will revert to normal.
That goes for Bad luck and good luck.

$0CFB[0x01] -
$0CFC[0x01] -

$0CFD[0x01] - Counter used to cause delay between rupee refill sound effects

$0CFE[0x02] - Something to do with frozen or carried sprites...?

==========

Pages 0x0D

==========

$0D00[0x10] - The lower byte of a sprite's Y - coordinate.
$0D10[0x10] - The lower byte of a sprite's X - coordinate.
$0D20[0x10] - The high byte of a sprite's Y - coordinate.
$0D30[0x10] - The high byte of a sprite's X - coordinate.

$0D40[0x10] - Y direction for a sprite. (velocity)
$0D50[0x10] - X direction for a sprite. (velocity)

$0D60[0x10] - Y "second derivative" to give a path a more rounded shape when needed.
$0D70[0x10] - X "second derivative" to give a path a more rounded shape when needed.

$0D80[0x10] - Controls whether the sprite has been spawned yet. 0 = no, not 0 = yes. Also used as an AI pointer

$0D90[0x10] - In some creatures, used as an index for determining $0DC0

$0DA0[0x10] - usage varies considerably for each sprite type

$0DB0[0x10] - hard to say at this point.
$0DC0[0x10] - Designates which graphics to use.

$0DD0[0x10] - An indicator to determine each sprite's state
0x00 = sprite is dead, totally inactive
0x01 = sprite is in a state of falling into a hole. timed by $0DF0
0x02 = sprite transforms into a puff of smoke, often producing an item
0x03 = ????
0x04 = ????
0x05 = falling into pit
0x06 = ????
0x08 = sprite is being spawn, initialization routine will be run then shift to state 09
0x09 = sprite is in normal, active mode
0x0A = sprite is being carried by Link
0x0B = sprite is frozen
0x0C or above = do not use as it will cause a crash

$0DE0[0x10] - A position counter for the statue sentry? May have other uses

$0DF0[0x10] - Delay Timers for certain behaviours
$0E00[0x10] - Auxiliary Delay Timer 1
$0E10[0x10] - Auxiliary Delay Timer 2
$0E20[0x10] - What type of sprite it is.

Sprite Catalog:

0x00 = Raven
0x01 = Vulture
0x02 = Flying Stalfos Head
0x03 = Unused (Don't use it, the sprite's ASM pointer is invalid. It will certainly crash the game.)
0x04 = Good Switch being pulled
0x05 = Some other sort of switch being pulled, but from above?
0x06 = Bad Switch
0x07 = switch again (facing up)
0x08 = Octorock
0x09 = Giant Moldorm (boss)
0x0A = Four Shooter Octorock
0x0B = Chicken / Chicken Transformed into Lady
0x0C = Octorock
0x0D = Normal Buzzblob / Morphed Buzzblob (tra la la... look for Sahashrala)
0x0E = Plants with big mouths
0x0F = Octobaloon (Probably the thing that explodes into 10 others)
0x10 = Small things from the exploder (exploder? wtf is that?)
0x11 = One Eyed Giants (bomb throwers) aka Hinox
0x12 = Moblin Spear Throwers
0x13 = Helmasaur?
0x14 = Gargoyle's Domain Gate
0x15 = Fire Faery
0x16 = Sahashrala / Aginah, sage of the desert
0x17 = Water Bubbles?

0x18 = Moldorm
0x19 = Poe
0x1A = Dwarf, Mallet, and the shrapnel from it hitting
0x1B = Arrow in wall?
0x1C = Moveable Statue
0x1D = Weathervane
0x1E = Crystal Switch
0x1F = Sick Kid with Bug Catching Net
0x20 = Bomb Slugs
0x21 = Push Switch (like in Swamp Palace)
0x22 = Darkworld Snakebasket
0x23 = Red Onoff
0x24 = Blue Onoff
0x25 = Tree you can talk to?
0x26 = Charging Octopi?
0x27 = Dead Rocks? (Gorons bleh)
0x28 = Shrub Guy who talks about Triforce / Other storytellers
0x29 = Blind Hideout attendant
0x2A = Sweeping Lady
0x2B = Bum under the bridge + smoke and other effects like the fire
0x2C = Lumberjack Bros.
0x2D = Telepathic stones?
0x2E = Flute Boy's Notes
0x2F = Heart Piece Race guy and girl
0x30 = Person? (HM name)
0x31 = Fortune Teller / Dwarf swordsmith
0x32 = ??? (something with a turning head)
0x33 = Pull the wall for rupees
0x34 = ScaredGirl2 (HM name)
0x35 = Innkeeper
0x36 = Witch / Cane of Byrna?
0x37 = Waterfall
0x38 = Arrow Target (e.g. Big Eye in Dark Palace)
0x39 = Middle Aged Guy in the desert
0x3A = Magic Powder Bat /The Lightning Bolt the bat hurls at you.
0x3B = Dash Item / such as Book of Mudora, keys
0x3C = Kid in village near the trough
0x3D = Signs? Chicken lady also showed up / Scared ladies outside houses.
0x3E = Rock Rupee Crabs
0x3F = Tutorial Soldiers from beginning of game
0x40 = Hyrule Castle Barrier to Agahnim's Tower
0x41 = Soldier
0x42 = Blue Soldier
0x43 = Red Spear Soldier
0x44 = Crazy Blue Killer Soldiers
0x45 = Crazy Red Spear Soldiers (And green ones in the village)
0x46 = Blue Archer Soldiers
0x47 = Green Archer Soldiers (in the bushes)
0x48 = Red Spear Soldiers (in special armor)
0x49 = Red Spear Soldiers (in the bushes)
0x4A = Red Bomb Soldiers
0x4B = Green Soldier Recruits (the idiots)

0x4C = Sand Monsters
0x4D = Flailing Bunnies on the ground
0x4E = Snakebasket
0x4F = Blobs?
0x50 = Metal Balls (in Eastern Palace)
0x51 = Armos
0x52 = Giant Zora
0x53 = Armos Knights Boss
0x54 = Lanmolas boss
0x55 = Zora / Fireballs (including the blue Agahnim fireballs)
0x56 = Walking Zora
0x57 = Desert Palace Barriers
0x58 = Sandcrab
0x59 = Birds (boids)
0x5A = Squirrels
0x5B = Energy Balls (that crawl along the wall)
0x5C = Wall crawling fireballs
0x5D = Roller (vertical moving)
0x5E = Roller (vertical moving)
0x5F = Roller
0x60 = Roller (horizontal moving)
0x61 = Statue Sentry
0x62 = Master Sword plus pendants and beams of light
0x63 = Sand Lion Pit
0x64 = Sand Lion
0x65 = Shooting Gallery guy
0x66 = Moving cannon ball shooters
0x67 = Moving cannon ball shooters
0x68 = Moving cannon ball shooters
0x69 = Moving cannon ball shooters
0x6A = Ball N' Chain Trooper
0x6B = Cannon Ball Shooting Soldier (unused in original = WTF?)
0x6C = Warp Vortex created by Magic Mirror
0x6D = Mouse
0x6E = Snakes (forgot the Zelda 1 name)
0x6F = Bats / Also one eyed bats
0x70 = Splitting Fireballs from Helmasaur King
0x71 = Leever
0x72 = Activator for the ponds (where you throw in items)
0x73 = Link's Uncle / Sage / Barrier that opens in the sanctuary
0x74 = Red Hat Boy who runs from you
0x75 = Bottle Vendor
0x76 = Princess Zelda
0x77 = Also Fire Faeries (seems like a different variety)
0x78 = Village Elder
0x79 = Good bee / normal bee
0x7A = Agahnim
0x7B = Agahnim energy blasts (not the duds)
0x7C = Boos? As in the Boos from SM3/SMW? That's crazy talk!
0x7D = 32*32 Pixel Yellow Spike Traps
0x7E = Swinging Fireball Chains
0x7F = Swinging Fireball Chains

0x80 = Wandering Fireball Chains
0x81 = Waterhoppers
0x82 = Swirling Fire Faeries (Eastern Palace)
0x83 = Rocklops
0x84 = Red Rocklops
0x85 = Yellow Stalfos (drops to the ground, dislodges head)
0x86 = Fire Breathing Dinos?
0x87 = Flames
0x88 = Mothula
0x89 = Mothula's beam
0x8A = Key holes? Spikes that move
0x8B = Gibdos
0x8C = Arghuss
0x8D = Arghuss spawn
0x8E = Chair Turtles you kill with hammers
0x8F = Blobs / Crazy Blobs via Magic powder or Quake Medallion
0x90 = Grabber things?
0x91 = Stalfos Knight
0x92 = Helmasaur King
0x93 = Bungie / Red Orb? (according to HM)
0x94 = Swimmers
0x95 = Eye laser
0x96 = Eye laser
0x97 = Eye laser
0x98 = Eye laser
0x99 = Penguin
0x9A = Moving Water Bubble (only in Swamp Palace)
0x9B = Wizzrobes
0x9C = Black sperm looking things
0x9D = Black sperm looking things
0x9E = The ostrich animal w/ the flute boy?
0x9F = Flute
0xA0 = Birds w/ the flute boy?
0xA1 = Ice man
0xA2 = Kholdstare
0xA3 = Another part of Kholdstare
0xA4 = Ice balls from above
0xA5 = Blue Horse Knight, and Lynel Fireballs
0xA6 = Red Horse Knight
0xA7 = Red Stalfos Skeleton
0xA8 = Bomber Flying Creatures from Darkworld
0xA9 = Bomber Flying Creatures from Darkworld
0xAA = Like Like (O_o yikes)
0xAB = Maiden (as in, the maidens in the crystals after you beat a boss)
0xAC = Apples
0xAD = Old Man on the Mountain
0xAE = Down Pipe
0xAF = Up Pipe
0xB0 = Right Pipe
0xB1 = Left Pipe
0xB2 = Good bee again?
0xB3 = Hylian Inscription (near Desert Palace). Also near Master Sword

0xB4 = Thief's chest (not the one that follows you, the one that you grab from the DW smithy house)
0xB5 = Bomb Salesman (elephant looking guy)
0xB6 = Kiki the monkey?
0xB7 = Maiden following you in Blind Dungeon
0xB8 = Monologue Testing Sprite (Debug Artifact)
0xB9 = Feuding Friends on Death Mountain
0xBA = Whirlpool
0xBB = Salesman / chestgame guy / 300 rupee giver guy / Chest game thief
0xBC = Drunk in the inn
0xBD = Vitreous (the large eyeball)
0xBE = Vitreous' smaller eyeballs
0xBF = Vitreous' lightning blast
0xC0 = Monster in Lake of Ill Omen / Quake Medallion
0xC1 = Agahnim teleporting Zelda to dark world
0xC2 = Boulders
0xC3 = Symbions 2 (vulnerable part)
0xC4 = Thief
0xC5 = Evil Fireball Spitters (THE FACES!!!)
0xC6 = Four Way Fireball Spitters (spit when you use your sword)
0xC7 = Fuzzy Stack
0xC8 = Big Healing Faeries / Faerie Dust
0xC9 = Ganon's Firebat (HM also says Tektite?)
0xCA = Chain Chomp
0xCB = Trinexx
0xCC = Another Part of Trinexx
0xCD = Another Part of Trinexx (again)
0xCE = Blind the Thief
0xCF = Swamp Worms (from Swamp of Evil)
0xD0 = Lynel (centaur like creature)
0xD1 = A yellow hunter
0xD2 = Flopping fish
0xD3 = Animated Skulls Creatures
0xD4 = Landmines
0xD5 = Digging Game Proprietor
0xD6 = Ganon! OMG
0xD7 = Copy of Ganon, except invincible?
0xD8 = Heart refill
0xD9 = Green Rupee
0xDA = Blue Rupee
0xDB = Red Rupee
0xDC = Bomb Refill (1)
0xDD = Bomb Refill (4)
0xDE = Bomb Refill (8)
0xDF = Small Magic Refill
0xE0 = Full Magic Refill
0xE1 = Arrow Refill (5)
0xE2 = Arrow Refill (10)
0xE3 = Faerie
0xE4 = Key
0xE5 = Big Key
0xE6 = Red Shield (after dropped by pickit)
0xE7 = Mushroom

0xE8 = Fake Master Sword
0xE9 = Magic Shop dude / His items, including the magic powder
0xEA = Full Heart Container
0xEB = Quarter Heart Container
0xEC = Bushes
0xED = Cane of Somaria Platform
0xEE = Movable Mantle (in Hyrule Castle)
0xEF = Cane of Somaria Platform (same as 0xED but this index is not used)
0xF0 = Cane of Somaria Platform (same as 0xED but this index is not used)
0xF1 = Cane of Somaria Platform (same as 0xED but this index is not used)
0xF2 = Medallion Tablet

$0E30[0x10] - Subtype designation 1
Is formed as follows: Take bits 5 and 6 from the sprite's Y coordinate byte and shift right twice.
Then take bits 5, 6, and 7 from the X coordinate byte and shift right five times. This produces 000yyxxx.
The bottom three bits cannot all be set at the same time or else we'll have an overlord instead.
So if that wasn't the case we'd have 32 possible subtypes, but in lieu of the extra rule,
7, 15, 23, and 31 cannot be used.
A safe way to use this would be to just not use the bottom bit. (allows 16 subtypes)

$0E40[0x10] - Bits 0-4: If zero, the sprite is invisible. Otherwise, visible.
(actually, it seems like this is the number of OAM sprite slots allocated to this sprite object)

Bit 5: Causes enemies to go towards the walls? strange...
Bit 6: No idea but the master sword ceremony sprites seem to use them....?
Bit 7: If set, enemy is harmless. Otherwise you take damage from contact.

$0E50[0x10] - Sprite HP

$0E60[0x10] - Bit 4: If set, draw a shadow for the sprite when doing OAM handling
Bit 6: if set, sprite is impervious to all attacks (also collisions?)
Bit 7: Add coordinate offsets for some sprites?

$0E70[0x10] - ????

$0E80[0x10] - Subtype designation 2 (varies from sprite to sprite though)
$0E90[0x10] - When a Pikit grabs something from you it gets stored here. Used wildly different between sprite types.

$0EA0[0x10] - ????

$0EB0[0x10] - ????
$0EC0[0x10] - ????

$0ED0[0x10] - ???? (Used with sprite 0xBE)

$0EE0[0x01] - Auxiliary delay timer 3

$0EF0[0x10] - abbbbbbb:
a - start death timer?
bbbbbbb - death timer?

$0F00[0x10] - Pause button for sprites apparently. If nonzero they don't do anything.

$0F10[0x10] - Auxiliary delay timer 4 (may be more proprietary than the others)

$0F20[0x10] - Floor selector. Tells us which floor each sprite is on (in multilevel rooms)

$0F30[0x10] - Seen in some code as a flip flop that gets multiplied by -1 (**XOR 0xFF** then adding 1)

$0F40[0x10] - Same as **$0F30**

$0F50[0x10] - Controls palette, tile flipping, and apparently tile set?

$0F60[0x10] - **bits 7-5**: ????
bits 0-4: hit box settings (uses a table to load offsets and form a hit box)

$0F70[0x10] - Height value (how far the enemy is from its shadow)

$0F80[0x10] - Auxiliary Timer 4

$0F90[0x10] - ???

$0FA0[0x01] - When a special effect is executing, its index is stored here (**0 to 0x0D**)
Also applies to sprites and perhaps other similar types of objects.
Would require more research to verify.

$0FA1[0x01] - Accumulator for generating random numbers. Each time a ranom number is generated, the current low byte of the h count (ppu) plus the frame index (**$1a**) is added to this variable and then stored back to this variable. The resultant value of this variable is the random number that ultimately gets returned.

$0FA2[0x03] - free ram

$0FA5[0x01] - Tile type for sprite/tile interactions (used on a temporary basis for special effects)

$0FA6[0x02] - free ram

$0FA8[0x01] - Screen relative X coordinate of a sprite (only lowest 8 bits)
$0FA9[0x01] - Screen relative Y coordinate of a sprite (only lowest 8 bits)

$0FAB - ???
$0FAC[0x01] - ????

$0FAD[0x01] - Temporary low byte of X coordinate for sprite or special object
$0FAE[0x01] - Temporary low byte of Y coordinate for sprite or special object

$0FB0[0x01] - Used to offset the high byte of pixel addresses in rooms. (X coord)
$0FB1[0x01] - Used to offset the high byte of pixel addresses in rooms. (Y coord)

$0FB2[0x01] - Unknown. c.f **$0314** though

$0FB3[0x01] - Corresponds to sort sprites in Hyrule Magic

$0FB4[0x01] - ????

$0FB5[0x01] - used in constructing special designation **$0E30**[]. Two most significant bits of the value.
also used in calculating sprite damage. **$0DD0**[] is stored here on a temporary basis.

$0FB6[0x01] - used in construction of special designation **$0E30**[]. Three least significant bits of the value.

$0FB8[0x02] - ????

$0FBA[0x02] - ????

$0FBF[0x01] - Holds base upper byte of current overworld area's coordinates

$0FC1[0x01] - Set to one during Desert Palace / Book of Mudora sequence. Maybe has uses in other sequences
Seems to freeze sprites. Could have uses in making a scripting system.

$0FC2[0x02] - Link's X-Coordinate (See **$22**)
$0FC4[0x02] - Link's Y-Coordinate (See **$20**)

$0FC6[0x01] - Only time this ever gets written with a nontrivial value is from **$0AAA**, which is also a
poorly understood variable.

$0FC7[0x01] - ?????

$0FCD[0x01] - Alert flag that activates "searching" enemies like soldiers. usually in response to a sound effect.

$0FCE[0x??] -

$0FD7[0x01] - The code that writes and reads this variable is not enabled. For this reason it is considered
a debug or "cheat" variable. When the value is odd, the game is frozen. The sprites that are on screen
will stay on screen.

$0FD8[0x02] - Cached 16-bit version of Link's X coordinate used in sprite logic calculations.
$0FDA[0x02] - Cached 16-bit version of Link's Y coordinate used in sprite logic calculations.
$0FDC[0x01] - **0 or 3**, when **3** counts down to **zero**. This is **nonzero** when a projectile hits a wall.
$0FDD[0x01] - ????

$0FE0[0x0C] - current position in each of the 6 different reserved areas of the OAM table
$0FE2 - occurs specifically in module 1A.

$0FEC[0x0C] -

$0FF8 - number of boss components left in a room? (e.g. Armos Knights)

$0FFA[0x01] - When a screen transition occurs in the overworld or indoors,
this is set to the value of the variable **$1B**
Sprites are cached and reloaded from adjacent rooms only indoors,
so this serves the purpose of letting the game know to not reload
cached sprites during a screen transition.

$0FFC[0x01] - If set, Link can't bring up his menu.

$0FFD[0x02] - ????
$0FFF[0x01] - Indicates whether you are in the light world or dark world (**0,1** respectively)

==========

Pages 0x10 to 0x17

==========

$1000[0x02] - For transfers that use variable **$14**, this indicates the current length written into the buffer starting at **$1002**

Seems to be addresses used for blitting graphics onto the screen after a room or area has already loaded.

$1002[0x02] - Big endian representation of the VRAM target address (word address)

$1004[0x02] - dma configuration

ssssssss wfttttttt

f - If set, DMA source address is fixed. Otherwise it auto increments.

s, t - One less(!) the 14-bit size of the DMA transfer, as expressed in bytes.
This value is composed of the 's' and 't' bits thusly:

00ttttttssssssss

w - If not set, vram target address increments on writes to $2118.
Otherwise, it increments on writes to $2119.
^ This documention is not 100% correct. I'll fix it when I have time.
Basically it determines whether or not to increment vram address by 1 or 32 words
after each adjustment (write to $2118 or $2119)

$1006[0x??] - First word of data for the transfer. Many more may follow.

Setup for making overlaid tiles appear (word addresses):

Notes on converting in game tile positions into VRAM positions:

In game layout: (u = unused?) uuddaaaa abcccccc
Translated into VRAM base address: uuuddbaa aaaccccc

$1100[0x02] - Target VRAM address for the transfer

$1102[0x01] - The value that will be written to VMAIN (**$2115**) before the transfer.
Typically this controls whether the DMA transfer is intended to write words
or alternating bytes to VRAM.

$1103[0x01] - Number of bytes to transfer to VRAM.
Note: *Putting **0** here would transfer **0x10000 bytes**, which could be really crazy.*

$1104[0x??] - Local portion of the source address for the DMA transfer. The source bank is forced to **zero** by the code.

$1100 - Contains two chunks of data for blitting an extra tile to the screen. The hex values indicate where each portion lies in the memory address.

0x8000 - indicates whether to use 2 byte or 64 byte increment when doing the DMA
0x4000 - ? not used?
0x3FFF - Contains the number of bytes to transfer

$1102[**0x02**] - VRAM target address for the tile setup.
$1104[**0x02**] - **$1183** - The bytes that get blitted to VRAM

$1980[**0x20**] - Low Byte: Door type (Take HM door type number, convert to hex, and multiply by 2)
High Byte: Door direction (**0** = up, **2** = down, **4** = right, **6** = left)

$19A0[**0x10**] - Door tilemap address

$19B0[**0x10**] - For doors that have two halves, this is the tilemap address for the other half

$19C0[**0x20**] - Door direction. **0** - up, **1** - down, **2** - left, **3** - right

$19E0[**0x02**] - current door index when loading doors
$19E2[**0x08**] - Seems to store the addresses of up to 4 different exit doors.

$19EA[**0x16**] - free ram

$1A00[**0x14**] - These **$1AXX** addresses seem to be related to tagalongs, that's the best I can tell so far
$1A14[**0x14**] -
$1A28[**0x14**] -
$1A3C[**0x14**] -
$1A50[**0x14**] - see **Bank 09**
$1A64[**0x14**] - Tagalong priority bits

$1A78[**0x??**] - ????

$1B00[**0x1C0**] - indirect HDMA pointers for the spotlights that open and close when leaving/entering doors

$1CC0[**0x10**] - free ram?

Text related variables (**$1CD0**[**0x20**])

[**0x02**] - When drawing the message box border, the value of **$1CD2** gets stored here during initialization. It is subsequently used as an index for forming DMA transfers that will copy the tilemap entries of the message box border during the NMI interrupt.;

$1CD2[**0x02**] - The VRAM position of upper left corner of the message box. This only has one of two values, and they are designed to not obscure Link's sprite.

$1CD4[**0x01**] - Second Level controller for *Module 0x0E.0x02*
$1CD5[**0x01**] - Maybe the current position the message is outputting to?

$1CD5[**0x01**] - Mirror of speed below... but... not sure why this is different

$1CD6[**0x01**] - Speed XX
$1CD7[**0x01**] - ???? related to [window 01] (*not used in game*)

$1CD8[**0x01**] - First Level controller for *Module 0x0E.0x02* (i.e. it controls the sub-submodule)
$1CD9[**0x02**] - Number of bytes into the **$7F1200** buffer for the processed characters of the dialogue.

$1CDB[**0x01**] - free ram

$1CDC[**0x01**] - This is supposedly some kind of color variable according to Hyrule Magic but I can't find any affect to using it. It's modified by the [Color XX] command in the monologue editor. This was included presumably before they implemented a VWF, and used single or double spaced characters? The reasoning being, with such a font you can set the colors for individual characters much more easily. With 2bpp font characters you don't have a lot of available variety for colors in a single tile anyways.

$1CDD[**0x02**] - Number of bytes into the source buffer for the dialogue message.
$1CDD[**0x02**] - *(alternate) - once the text begins being processed, this is instead the tile position for the output 0x00 is the first line, 0x28 is the second line, 0x50 is the third line*

$1CDF[**0x01**] - Used in [Scroll] command, not sure what else

$1CE0[**0x02**] - ???? referenced in [Wait XX]

$1CE2[**0x02**] - Is the template for the text message's tilemap. Each character has a tilemap value that differs only in CHR
[Color XX] can change the palette (out of 8) obviously.
Changing this value can also alter the h-flip and v-flip, as well as priority settings. Generally speaking you wouldn't want to do that unless you had a good reason.

$1CE4[**0x02**] - free ram

$1CE6[**0x02**] - Perhaps this is the graphical position of where the next character goes?

$1CE8[**0x01**] - For a multi selection box with two choices, this is either 0 (the top choice) or 1 (the bottom choice)

$1CE9[**0x01**] - This is a delay timer for the [WaitKey] command. Starts at **0x1C**, when it counts down to zero it starts looking for player input. (basically this is 1/3 of a second and is probably there to give the game more time to draw its text

$1CEA[**0x01**] - Determines how many times to scroll the text up by one pixel per frame
Only applies during the [Scroll] command and is set by the [ScrollSpd XX] command. However, some other commands may inadvertently affect this address. E.g. [Name], [Position XX]. As there is no logical connection I can only guess that this was a coding oversight.

$1CF0[**0x02**] - Dialogue Message Index. e.g. 0005 = "you can't enter with something following you"
This is the same numerical value that can be viewed in Hyrule Magic (except there they're in decimal.)

$1CF2[**0x02**] - Numeric parameter that can be passed to text messages. It goes as follows.
first byte: low nybble corresponds to [Number 00], high nybble to [Number 01]
second byte: low nybble is [Number 02], high nybble is [Number 03]
If you're confused see message number 004 in Hyrule Magic's dialogue editor.

$1CF4[**0x01**] - Used to save the value of **$1CE8** when the save and continue box comes up

$1CF5[**0x0B**] - free ram

$1D00[**0x10**] - Cache for **$0DD0**

$1D10[**0x10**] - Cache for **$0E20**

$1D20[**0x10**] - Cache for **$0D10**
$1D30[**0x10**] - Cache for **$0D30**
$1D40[**0x10**] - Cache for **$0D00**
$1D50[**0x10**] - Cache for **$0D20**

$1D60[**0x10**] - Cache for **$0DC0**

$1D70[**0x10**] - Cache for **$0D90**
$1D80[**0x10**] - Cache for **$0EB0**
$1D90[**0x10**] - Cache for **$0F50**
$1DA0[**0x10**] - Cache for **$0B89**
$1DB0[**0x10**] - Cache for **$0DE0**
$1DC0[**0x10**] - Cache for **$0E40**
$1DD0[**0x10**] - Cache for **$0F20**
$1DE0[**0x10**] - Cache for **$0D80**
$1DF0[**0x10**] - Cache for **$0E60**

$1E00[**0x01**] - Step counter during the Triforce sequence at the beginning.

$1E08[**0x01**] - see *module 0x14*
$1E09[**0x01**] - see *module 0x14*

$1E0A[**0x01**] - Seems to be a frame counter (loops at **0xFF**) used for various delays and timing of events during the Triforce or Crystal sequences

$1E10[**0x01**] -

$1E68[**0x98**] - free ram???

$1F00[**0x01**] - Flag that when set, tells IRQ to activate
$1F0A - Use to swap out the two different stacks. (the **$1F00** region and the **$1FF** region)
$1F0C - Related to IRQs?
$1F32-$1F3F - Contains data to use when coming out of the IRQ. Including a return address, the Data Bank Register, the Direct Page register (gets set to **$1F00**), and various other data. It took a while to figure this out, believe me.

==========

Start of unmirrored WRAM

==========

Overworld:
$7E2000[**0x2000**] - Map16 tile data for the overworld. Supports up to 1024x1024 pixels of tiles. ***Note:*** *this is handled somewhat differently than the Map8 data in the dungeons which supports up to 512x512.*

Dungeons:

$7E2000[**0x2000**] - when loading dungeon graphics, contains the (map8) tilemap for BG1.

$7E4000[**0x2000**] - when loading dungeon graphics, contains the (map8) tilemap for BG0.

A note about these tilemap representations:

These are the tilemaps but not exactly as they would linearly appear in VRAM.
These maps are stored as the whole 512x512 pixel screen, line by line.

The first $80 bytes contains the top line of the picture as it appears to us.
The second $80 bytes represents the second line of the picture as it appears to us, etc.

In VRAM, however, the tile indices are stored by quadrant of the screen. Thus, the algorithm to convert the address of a tile in this map to one in VRAM is:

```
int B = tilemap_address;
int Y = 0; // Y will be the resulting VRAM offset

if(B & 0x1000) // Is this tile in the lower two quadrants?
{
    // YES!

    Y += 0x1000;
    B ^= 0x1000; // Tells us to ignore this bit in the proceeding calculations.
}
if(B & 0x40) // Is this tile in a right hand quadrant?
{
    // YES!

    Y += 0x800;
    B ^= 0x040; // The ^ symbol is XOR, in case you didn't know.
}

Y += (B - ((B & 0xFF80) >> 1));
```

Keep in mind these aren't real addresses, just offsets into respective arrays in WRAM and VRAM. e.g. X or Y registers would be the offset in WRAM and your DMA controls registers would handle the problem for DMA.

```
WRAM  -> VRAM
$0    -> $0
$40   -> $800
$80   -> $40
$C0   -> $840
$100  -> $80
$140  -> $880
$180  -> $C0
$1C0  -> $8C0
$280-$2BF -> $140-$17F
...       -> ...
$1000 -> $1000
$1040 -> $1800
$1080 -> $1040
$10C0 -> $1840
etc.  -> etc.
```

$7E6000[**0x3000**] - Scratch space where decompressed data is stored temporarily

$7E9000[**0x2E00**] - Buffer for an assortment of tiles that need to change appearance

$7E9000[0x300] - 24 Sword tiles
$7E9300[0x180] - 12 Shield tiles
$7E9480[0x1C0] - 14 Ice / Fire Rod tiles
$7E9640[0x1C0] - 14 Hammer tiles
$7E9800[0xC0] - 6 Bow tiles
$7E98C0[0x100] - 8 Shovel tiles
$7E99C0[0x100] - 8 Tiles (Sleep 'Z's, Musical notes, two unknown tiles)
$7E9AC0[0x100] - 8 Hookshot tiles
$7E9BC0[0x380] - 28 Bug Net tiles
$7E9F40[0x1C0] - 14 Cane tiles (Byrna and Somaria)
$7EA100[0x80] - 4 Book of Mudora tiles
$7EA180[0x300] - 24 unused tiles (free ram)
$7EA480[0x200] - 16 Push Block tiles
$7EA680[0x400] - Animated BG tiles for step 0
$7EAA80[0x400] - Animated BG tiles for step 1
$7EAE80[0x400] - Animated BG tiles for step 2
$7EB280[0xC0] - Sprite tiles for rupee animation (3 steps)
$7EB340[0x200] - Tiles for the steps that animate the barrier tiles going up and down
$7EB540[0x400] - Sprite tiles for bird and thief's chest
$7EB940[0x400] - Tagalong graphics, room for 8 16x16 sprites (32 8x8 tiles)
$7EBD40[0x80] - Sprite tiles for receive item (4 8x8 tiles)
$7EBDC0[0x40] - Star tiles (part of animated tiles, consists of 2 8x8 tiles)

$7EBE00[0x200] - possibly free ram

Values related to Dungeon Headers:

$7EC000[0x01] - The dungeon header can have up to five destinations for you to travel to
$7EC001[0x01] - These five addresses ($7EC000-$7EC004) hold the room numbers of these
$7EC002[0x01] - destinations. Notice that you can only use this for rooms with numbers
$7EC003[0x01] - Less than 256.
$7EC004[0x01] -

$7EC005[0x01] - Tells us whether to do a "lights out" before room transitioning.

$7EC006[0x01] - Mirror for $7EC005. Chains things along really. If the last room had the "lights out" property, then when I return to it it should also, right?

$7EC007[0x02] - Related to $7EC011 in that it helps control the mosaic level as well as the stages of color filtering during a mosaic transition (forest to normal ow, etc)

$7EC009[0x02] - 0 when darkening the screen, 2 when lightening the screen (during color filtering)

$7EC00B[0x02] - The target level of color filtering and mosaic for $7EC007 during mosaic screen transitions

$7EC00D[0x02] - Countdown timer that, when it reaches zero, changes the set of tiles used in BG tile animation. Normally starts at 9 and changes at 0, but according to the code can start at 0x17 if certain overlays are used. It haven't been verified that those overlay numbers actually are used in the game, though.

$7EC00F[0x02] - Determines which frame of animation to use for animated tiles. Range: $0 - $800, ($400 increment)

$7EC011[0x01] - Mosaic "level" in screen transitions transitions.

$7EC013[0x02] - Rupee sprite tile animation timer? (countdown)

$7EC017[0x01] - ???? Linked to $045A seems to determine fixed color +/-

$7EC018[0x01] - free ram

$7EC019[0x06] - Array of palette filter settings that get temporarily stored to **$7EC007** for Agahnim's sprite.
$7EC01F[0x06] - Array of palette filter settings that get temporarily stored to **$7EC007** for Agahnim's sprite.

$7EC025[0xDB] - free ram

$7EC100[0x02] - Mirror of **$040A**

$7EC102[0x02] - Mirror of **$1C**

$7EC104**[0x02] - Mirror of **$E8
$7EC106**[0x02] - Mirror of **$E2

$7EC108[0x02] - Mirror of **$20**
$7EC10A[0x02] - Mirror of **$22**

$7EC10C[0x02] - Mirror of **$8A**
$7EC10E[0x02] - Mirror of **$84**

$7EC110[0x02] - Mirror of **$0618**
$7EC112[0x02] - Mirror of **$061C**
$7EC114[0x02] - Mirror of **$0600**
$7EC116[0x02] - Mirror of **$0602**
$7EC118[0x02] - Mirror of **$0604**
$7EC11A[0x02] - Mirror of **$0606**
$7EC11C[0x02] - Mirror of **$0610**
$7EC11E[0x02] - Mirror of **$0612**
$7EC120[0x02] - Mirror of **$0614**
$7EC122[0x02] - Mirror of **$0616**

$7EC124[0x01] - Mirror of **$0AA0**
$7EC125[0x01] - Mirror of **$0AA1**
$7EC126[0x01] - Mirror of **$0AA2**
$7EC127[0x01] - Mirror of **$0AA3**

$7EC128[0x02] - free ram?

$7EC12A[0x02] - Mirror of **$0624**
$7EC12C[0x02] - Mirror of **$0626**
$7EC12E[0x02] - Mirror of **$0628**
$7EC130[0x02] - Mirror of **$062A**

$7EC132[0x0E] - free ram?

$7EC140[0x02] - Mirror of **$040A**, alternate overworld area number

$7EC142[0x01] - Mirror of $1C
$7EC143[0x01] - Mirror of $1D

$7EC144[0x02] - Mirror of $00E8, BG1 V scroll value
$7EC146[0x02] - Mirror of $00E2, BG1 H scroll value

$7EC148[0x02] - Mirror of $0020, Link's Y coordinate
$7EC14A[0x02] - Mirror of $0022, Link's X coordinate

$7EC14C[0x02] - Mirror of $008A, overworld area number
$7EC14E[0x02] - Mirror of $0084, ????

$7EC150[0x02] - Mirror of $0618, Camera's Y coordinate lower bound
$7EC152[0x02] - Mirror of $061C, Camera's X coordinate lower bound
$7EC154[0x02] - Mirror of $0600
$7EC156[0x02] - Mirror of $0602
$7EC158[0x02] - Mirror of $0604
$7EC15A[0x02] - Mirror of $0606
$7EC15C[0x02] - Mirror of $0610
$7EC15E[0x02] - Mirror of $0612
$7EC160[0x02] - Mirror of $0614
$7EC162[0x02] - Mirror of $0616

$7EC164[0x01] - Mirror of $0AA0
$7EC165[0x01] - Mirror of $0AA1, (blockset for dungeon entrance)
$7EC166[0x01] - Mirror of $0AA2
$7EC167[0x01] - Mirror of $0AA3

$7EC168[0x02] - ???? See pre dungeon mode
$7EC16A[0x02] - Mirror of $0624
$7EC16C[0x02] - Mirror of $0626
$7EC16E[0x02] - Mirror of $0628
$7EC170[0x02] - Mirror of $062A
$7EC172[0x02] - Orange/blue barrier state
$7EC174[0x02] - Mirror of $86
$7EC176[0x02] - Mirror of $88
$7EC178[0x08] - free ram?

$7EC180[0x02] - Mirror of $E2
$7EC182[0x02] - Mirror of $E8
$7EC184[0x02] - Mirror of $20
$7EC186[0x02] - Mirror of $22

$7EC188[0x02] - Mirror of $0600
$7EC18A[0x02] - Mirror of $0604
$7EC18C[0x02] - Mirror of $0608
$7EC18E[0x02] - Mirror of $060C
$7EC190[0x02] - Mirror of $0610
$7EC192[0x02] - Mirror of $0612
$7EC194[0x02] - Mirror of $0614
$7EC196[0x02] - Mirror of $0616

$7EC198[0x02] - Mirror of $0618
$7EC19A[0x02] - Mirror of $061C

$7EC19C[0x02] - Mirror of $A6
$7EC19E[0x02] - Mirror of $A9

$7EC1A0[0x06] - free ram?

$7EC1A6[0x02] - Mirror of $2F
$7EC1A8[0x02] - Mirror of $0476
$7EC1AA[0x02] - Mirror of $A4

$7EC1AC[0x54] - free ram?

$7EC200[0x02] - Mirror of $E0
$7EC202[0x02] - Mirror of $E2
$7EC204[0x02] - Mirror of $E6
$7EC206[0x02] - Mirror of $E8

$7EC208[0x01] - Mirror of $0414
$7EC20A[0x01] - Mirror of $0AB6
$7EC20B[0x01] - Mirror of $0AB8
$7EC20C[0x01] - Mirror of $0AB7
$7EC20D[0x01] - free ram?
$7EC20E[0x01] - Mirror of $0AA1
$7EC20F[0x01] - Mirror of $0AA3
$7EC210[0x01] - Mirror of $0AA2

$7EC211[0x01] - Mirror of $1C
$7EC212[0x01] - Mirror of $1D

$7EC213[0x02] - Mirror of $8A
$7EC215[0x02] - Mirror of $84
$7EC217[0x02] - Mirror of $88
$7EC219[0x02] - Mirror of $86

$7EC21B[0x02] - Mirror of $0418
$7EC21D[0x02] - Mirror of $0410
$7EC21F[0x01] - Mirror of $0416

$7EC221[0x02] - Mirror of $011A or $7EC007
$7EC223[0x02] - Mirror of $011C or $7EC009

$7EC225[0x01] - Mirror of $99
$7EC226[0x01] - Mirror of $9A

$7EC227[0x01] - Mirror of $0130
$7EC228[0x01] - Mirror of $0131

$7EC229[0x01] - Mirror of $9B

$7EC22A[0x05] - free ram

$7EC230[**0xC0? (best guess)**] - related to the VWF?

; These are set when new graphics are loaded using **$0AA2**
$7EC2F8[**0x01**] - Cached value for subset 0 of secondary bg graphics
$7EC2F9[**0x01**] - Cached value for subset 1 of secondary bg graphics
$7EC2FA[**0x01**] - Cached value for subset 2 of secondary bg graphics
$7EC2FB[**0x01**] - Cached value for subset 3 of secondary bg graphics

; These are set when new graphics are loaded using **$0AA3**
$7EC2FC[**0x01**] - Sprite Graphics Subset 0
$7EC2FD[**0x01**] - Sprite Graphics Subset 1
$7EC2FE[**0x01**] - Sprite Graphics Subset 2
$7EC2FF[**0x01**] - Sprite Graphics Subset 3

$7EC300[**0x200**] - Auxiliary palette buffer. Probably used for manipulations of the palette that would be cause graphical glitches if applied every frame.

$7EC500[**0x200**] - Main palette buffer (**512 bytes**)
Palette Data that will get written to CGRAM (**512 bytes**) whenever **$7E0015** is **nonzero**.

The following is a list of tile types found in each palette slot.

Extra Info:

BP-<number> means background ***palette <number>***
SP-<number> means ***sprite palette <number>***

The "first half" indicates the first 8 colors of a 4bpp palete, and the "second half" indicates the latter 8 colors.

Note: *however, that the first color of the "first half" palettes are never actually displayed b/c the hardware treats them as transparent.*

HUD-<number>- indicates a 4 color palette used with the 2bpp heads up display graphics. *<number> ranges from **0 to 7***.
Only the last 3 colors are displayed as the first color in each HUD palette is considered transparent.

Palettes are listed in order of appearance:

HUD-0 - Arrow icon and blank item box.
HUD-1 - Numbers, hearts, the "Life" icon, and the Floor Indicator.
HUD-2 - Key icon, outline of magic bar and item box, the two dashes on either side of the "Life" icon and the border of the dialogue window.
HUD-3 - Bomb icon
HUD-4 - ????
HUD-5 - Pegasus boots on menu window
HUD-6 - Dialogue text,
HUD-7 - Magic meter (green) and rupee icon

BP-2 (first half) - Overworld: Green / Tan ground, most of the rock and related colors on Death Mountain
Dungeons: Misc things in dungeons, like edges of pits, parts of doors, boundaries for doors,

boundaries of torches

BP-2 (second half) - Overworld: Houses, urns, tops of wooden gates,
Dungeons: Walls / doors in dungeons, houses on overworld

BP-3 (first half) - Overworld: Secondary ground color, borders of certain paths, like in desert.
Dungeons: Torches in dungeons, pots, statues, orange switch blocks

BP-3 (second half) - Overworld: Statues, particular parts of rocks, paths and outlines of stone paths
Dungeons: Primary floor colors

BP-4 (first half) - Overworld: particular parts of rocks
Dungeons: doorway statue in dungeons

BP-4 (second half) - Overworld: Green trees, cacti, woods seen from high above on Death Mountain
Dungeons: parallax floor below in dungeons, dungeon wall border.

BP-5 (first half) - Animated tiles in overworld, beds, chairs tables in indoors

BP-5 (second half) - Overworld: Pathway in Kakkariko and shrub hedges, overlays using color addition (fog in forest, lava in dark world)
Dungeons: Dungeon entrance floor

BP-6 (first half) - Overworld: Bushes, fences, grass, and the ground bushes and grass leave behind, signs
Dungeons: Chests, candles in fortune teller huts, blue switch blocks

BP-6 (second half) - Overworld: Tower of Hera, some parts above doors, master sword platform, small trees, red trees, Sahasralah's house.
Dungeons: Rug in smithy house, BG1 floor

BP-7 (first half) - Pyramid of Power, Dark Palace, Blue House roofs, warp tiles / cloud tiles.
Seems to be an important palette.
Dungeon: and tiles related to chests tiles in dungeons

BP-7 (second half) - Overworld: cloud tiles (some), rope bridges
Dungeons: Ceilings

SP-0 (first half) - Sanctuary and Hyrule Castle Mantle, old guy at the bar in Kakkariko's body, weird head looking things at the entrance to the Desert Palace, bombos and ether tablets
SP-0 (second half) - heavy rocks
SP-1 (first half) - apples from trees, part of master sword beam, grass around your legs, off color bushes
SP-1 (second half) - red rupees, small hearts, red potion in shops, some shadows, link's bow
SP-2 (first half) - numerous special objects (dash dush from boots, sparkles, death / transformation poof), warp, whirlpool
SP-2 (second half) - blue rupees
SP-3 (first half) - red soldiers, bees,
SP-3 (second half) - usually set to all dark grey colors, but can be swapped with SP-5 (second half)
SP-4 (first half) - faeries, blue soldiers, chickens, green soldier shields, crows, lumberjack saw
SP-4 (second half) - green rupees, green enemy bombs, some shadows, magic decanters
SP-5 (first half) - palette determined by $0AAD, so varies by area / room
SP-5 (second half) - Link's Sword (first three colors) and Shield (last four colors)
SP-6 (first half) - palette determined by $0AAE, so varies by area / room

SP-6 (second half) - throwable items like bushes, pots, rocks, signs,
and the pieces all of those shatter into skulls that lurk in the ground under bushes
SP-7 (full) - Link's body palette, including his glove color

$7EC700[**0x14A**] - The HUD tile indices buffer. (**330 bytes**)

$7EC7F2[**0x04**] - floor indicator (*top two tiles*)
$7EC782[**0x04**] - floor indicator (*bottom two tiles*)

$7EC84A[**0x1FB6**] - seemingly free ram (*nearly 8K!*)

$7EE800[**0x800**] - it's cleared out but I don't know what it's used for

$7EF000[**0x500**] - Save Game Memory, which gets mapped to a slot in SRAM when you save your game. SRAM slots are at **$70**:**0000**, **$70**:**0500**, **$70**:**0A00**. They are also mirrored in the next three slots. See the sram documentation for more details.

$7EF500[**0x80**] - Possibly free ram

$7EF580[**0x280**] - (dungeon) For items under pots, this keeps track of which ones have been revealed in a given room (array probably resets on reentrance to the dungeon or mirror usage)

$7EF800[**0x100**] - Stores the overworld tilemap address for a tile that has been modified.
Seems to support up to **0x80** different tiles.
At first I thought this was for knowing what to reinstate when the screen scrolls on larger overworld areas, but that doesn't seem to be the case. The index into this table is **$04AC.**

$7EF900[**0x40**] - free ram?

$7EF940[**0x200**] - (dungeon) moveable block data (persistent across rooms and only regenerates upon dungeon reentry or mirror). In the original game, only **0x18C** of these bytes are filled with actual block data.

Each block entry consists of 2 16-bit values:

The first is the room that the block exists in.

The second is the tilemap address of the block.
If the tilemap address is from 0x0000 to 0x1FFF, then it's on BG2
If it's from 0x2000 to 0x3FFF, the block is on BG1.
This value is always anded with 0x3FFF, forcing the block to one BG or the other.

$7EFA00[**0x200**] - (overworld) Note that this clashes with the block data

$7EFB40[**0x180**] - (dungeon) torch data (persistent across rooms)
Each torch entry consists of 2 16-bit values:

The first word is the room that the torch exists in.

The second

$7EFCC0[**0x80**] - Each byte is the sprite graphics set to use for each overworld area.

$7EFD40[**0x80**] - Each byte is the sprite palette set to use for each overworld area.

$7EFDC0[**0x40**] - free ram?

$7EFE00[**0x200**] - CHR to Tile Attribute table. Each byte tells the game how Link interacts with the each CHR type.

After the tilemaps are created from loading objects, the game then uses this table to translate CHR values into behavior types, and creates behavior tables for BG0 and BG1.

As an example, let's say the CHR value for a particular tile in the tilemap was 0x04. In this lookup table, that would be the 5th byte (since it starts at 0), and you'd find the behavior type for that CHR to be 0x02. CHR values 0x140 0x1BF are loaded with behaviors that vary depending on the current value of $0AA2, the secondary graphics index.

See $**7F2000** and **$7F3000** for details on tile attributes in dungeons.

Also see routine **$0717D9** in the Banks files (if you have them) and **$71659** in Zelda3_ROM.log.

==========

Start of bank 0x7F

==========

$7F0000[**0x7E0**] - tile index buffer for text

$7F0000[**0x850**] - no idea

$7F07E0[**???**] - ????

$7F1200[**0x800**] - text buffer for character data, using pointers loaded from the table at $7F71C0

$7F2000[**0x2000**] - at some point carries the tile map data for the rain overlay.

$7F2000[**0x1000**] - (Dungeons) BG2 tile attribute table - (after the level/room has loaded), tile information for the room/map.

In particular tells the game how to handle each 8x8 tile. For example, a chest will have the value 0x58 (or something similar) in 4 different locations, interlaced of course.

List of Tile types:

0x00 - normal?
0x01 - collide
0x02 - ???
0x03 - ???? what the hell is this?

0x08 - swim / deep water
0x09 - shallow water (not swimmable)

0x0A - ????
0x0B - ????

0x0C - moving floor (Mothula's room)
0x0D - spike floor (hurts)
0x0E - ice floor
0x0F - more ice floor?

0x1C - top of water staircase
0x1D - in room staircase
0x1E - in room staircase
0x1F - in room staircase

0x20 - Pit / Hole tiles

0x21 - ????

0x22 - stairs that slow you down

0x23 - Lower half of trigger tile (Object 1.1.0x35 also uses it, but it seems like a broken mess)
0x24 - Upper half of trigger tile

0x26 - Boundary tile for In-floor inter-room staircases
0x27 - white statues (dungeons) / fences (overworld)

0x28 - Ledge leading up
0x29 - Ledge leading down
0x2A - Ledge leading left
0x2B - Ledge leading right
0x2C - Ledge leading up + left
0x2D - Ledge leading down + left
0x2E - Ledge leading up + right
0x2F - Ledge leading down + right

0x30 - Up Staircase to room 1 of 5
0x31 - Up Staircase to room 2 (")
0x32 - Up Staircase to room 3 (")
0x33 - Up Staircase to room 4 (")

0x34 - Down Staircase to room 1 of 5
0x35 - Down Staircase to room 2 (")
0x36 - Down Staircase to room 3 (")
0x37 - Down Staircase to room 4 (")

0x38 - Boundary tile for straight up inter-room staircases
0x39 - Boundary tile for sraightt up inter-room staircases

0x3A - ????
0x3B - star tiles that change up the floor

0x3D - inter-floor staircases?
0x3E - inter-floor staircases?
0x3F - inter-floor staircases?

0x40 - Thick Grass (and smashed in moles?)

0x42 - Gravestone

0x44 - Spike Block (dungeon) / Cactus (overworld)

0x48 - (overworld) Normal blank ground
0x4A - ????

0x4B - Orange Warp Tile (Dungeons) / Blue Warp Tile (Overworld)

0x4E - Mountain rock tile found in only a few select areas
0x4F - Mountaon rock tile found in only a few select areas

0x50 - (overworld) bush
0x51 - (overworld) off color bush
0x52 - (overworld) small light rock
0x53 - (overworld) small heavy rock
0x54 - (overworld) sign
0x55 - (overworld) large light rock
0x56 - (overworld) large heavy rock
0x57 - (overworld) rock pile

0x58 - chest 0
0x59 - chest 1
0x5A - chest 2
0x5B - chest 3
0x5C - chest 4
0x5D - chest 5

0x5E - upward staircase tile

0x60 - ????

0x62 - bombable cracked floor

0x63 - minigame chests?

0x66 - blue / orange block that is down
0x67 - blue / orange block that is up

0x68 - conveyor belt
0x69 - conveyor belt
0x6A - conveyor belt
0x6B - conveyor belt

0x70 - pot or bush (or mole?)
0x71 - pot or bush
0x72 - pot or bush
0x73 - pot or bush
0x74 - pot or bush
0x75 - pot or bush

0x76 - pot or bush
0x77 - pot or bush
0x78 - pot or bush
0x79 - pot or bush
0x7A - pot or bush
0x7B - pot or bush
0x7C - pot or bush
0x7D - pot or bush
0x7E - pot or bush
0x7F - pot or bush

0x80 - open door?

0x8E - send player to overworld? (well supported so far >_>)

0x90 - screen transition with BG toggle (BG0 <-> BG1)
0x91 - screen transition with BG toggle (BG0 <-> BG1)
0x92 - screen transition with BG toggle (BG0 <-> BG1)
0x93 - screen transition with BG toggle (BG0 <-> BG1)
0x94 - screen transition with BG toggle (BG0 <-> BG1)
0x95 - screen transition with BG toggle (BG0 <-> BG1)
0x96 - screen transition with BG toggle (BG0 <-> BG1)
0x97 - screen transition with BG toggle (BG0 <-> BG1)

0xA0 - screen transition with dungeon toggle
0xA1 - screen transition with dungeon toggle
0xA2 - screen transition with dungeon toggle
0xA3 - screen transition with dungeon toggle
0xA4 - screen transition with dungeon toggle
0xA5 - screen transition with dungeon toggle

0xB0 - Cane of Somaria line (up/down)
0xB1 - Cane of Somaria line (left/right)
0xB2 - ???
0xB6 - Cane of Somaria line node (question mark shaped)

0xC0 - Torch 0x00
0xC1 - Torch 0x01
0xC2 - Torch 0x02
0xC3 - Torch 0x03
0xC4 - Torch 0x04
0xC5 - Torch 0x05
0xC6 - Torch 0x06
0xC7 - Torch 0x07
0xC8 - Torch 0x08
0xC9 - Torch 0x09
0xCA - Torch 0x0A
0xCB - Torch 0x0B
0xCC - Torch 0x0C
0xCD - Torch 0x0D
0xCE - Torch 0x0E
0xCF - Torch 0x0F

0x0D0 - ????

0xF0 - Key door 1
0xF1 - Key door 2
0xF2 - Key door 3
0xF3 - Key door 4
0xF4 - ????

$7F3000[0x1000] - Tile Attribute table for BG1. Same as $7F2000 but for a different background.

$7F4000 - enemy related
$7F5000[0x800] - free ram
$7F5800[?] - used?

$7F5B00[0xA0] - Array of sound settings for overworld areas.
The top 4 bits determine the ambient sound effect, which is written to $012D
The bottom 4 bits determine the song number to play when the area is loaded ($012C)
This array is preloaded with differing values depending on which stage of the game you're in.
The stages are determined by $7EF3C5 (See $3C5 in Zelda_SRM.log)

$7F5BA0[0x60] - free ram

$7F6000[0x1000] - enemy damage related

$7F7000[0x1C0] - Used to generate the hdma table that the intro and outro spotlight effect uses

$7F71C0[0x4A7] - text / dialogue pointers (all of them!). Each one is a 3 byte long pointer.

$7F7667[0x6719] - free ram

$7FDD80[0x200] - Serves as a temporary buffer for storing a copy of the current palette buffer.
Sometimes the contents of $7EC500 gets stored here, and other times it's $7EC300
being preserved. The general idea here is that the game hopes to eventually
(optionally) restore the palette to this set of colors later.

$7FDE00[0x180?] - ???

$7FDF80[0x280] - (indoors) I think this is used to describe which rooms have had their sprites loaded
(while in a dungeon)
$7FDF80[0x1000] - (outdoors) Memory region that indicates where sprites are in the currently loaded overworld map
(which map16 tile they're on)

$7FEF80[0x200] - death status for the overworld sprites?

$7FF180[0x?] - ???

$7FE200 -

Special animation memory region:

$7FF800[0x1E] - type of special animation

0x01 - Fireball chain... fireballs? Is the whole thing made up of these?
0x02 - Mothula beam?
0x03 - Falling tile? Or the effect after the tile has falling of it disappearing?
0x04 - Something to do with eye lasers in the walls
0x05 - one sparkling point produced when you freeze an enemy and as while it's frozen periodically
0x06 - Black sperm looking things related...
0x07 - Something to do with Kholdstare, maybe the ice that falls from the ceiling?
0x08 - Fireball sprite from zora or evil face things. Maybe the trails of those instead?
0x09 - Something to do with Vitreous... maybe his lightning
0x0A - bush clippings scattering (also used for grass)
0x0B - Swimmer splashes?
0x0C - Trinexx related...
0x0D - Invalid, don't use this animation as it will certainly crash the game. (It's a null pointer)
0x0E - Trinexx related...
0x0F - Blind related...

0x10 - Something to do with Trinexx... perhaps fire or ice blasts.
Also related to Ganon's firebats that spawn fireballs.
I think it's pretty clear it's those fireballs

0x11 - Spawned from special animation 0x0F...
0x12 - Sprite falling into hole?
0x13 - The Ganon bat smashing into the pyramid of power
0x14 - Red hat man's dash puffs?
0x15 - Arghus splashes?
0x16 - pot shattering

$7FF81E[0x1E] - special animation Y coord. low byte
$7FF83C[0x1E] - special animation X coord. low byte
$7FF85A[0x1E] - special animation Y coord. high byte
$7FF878[0x1E] - special animation X coord. high byte
$7FF896[0x1E] - special animation unknown 1
$7FF8B4[0x1E] - special animation unknown 2
$7FF8D2[0x1E] - special animation unknown 3
$7FF8F0[0x1E] - special animation unknown 4
$7FF90E[0x1E] - special animation countdown timer
$7FF92C[0x1E] - special animation floor designator
$7FF94A[0x1E] - special animation unknown 5
$7FF968[0x1E] - special animation unknown 6
$7FF986[0x1E] - special animation unknown 7
$7FF9A4[0x1E] - special animation unknown 8

$7FF9C2[0x10] - Sprite tile interaction value. Acquires its value from $0FA5.

$7FF9D2[0x2C] - Free ram?

$7FF9FE[0x1E] - special animation unknown B

Holdable objects

$7FFA1C[0x10] - Array that seems to handle the objects you hold above your head?
$7FFA2C[0x10] -
$7FFA3C[0x10] -
$7FFA4C[0x10] -
$7FFA5C[0x10] -
$7FFA6C[0x10] -
$7FFA7C[0x10] -
$7FFA8C[0x10] -
$7FFA9C[0x10] -
$7FFAAC[0x10] -
$7FFABC[0x10] -
$7FFACC[0x10] -
$7FFADC[0x10] -
$7FFAEC[0x10] - free ram?
$7FFAFC[0x10] - free ram?
$7FFB0C[0x10] - free ram?
$7FFB1C[0x10] - used, but unknown
$7FFBDC[0x10] - used, but unknown

; most of these seem to be referenced for Helmasaur King's tail (maybe other bosses too)

$7FFC00[??] - unknown
$7FFC80[??] - unknown

$7FFC9C[0x10?] - used, but unknown

$7FFD00[??] - unknown

$7FFD5C[0x01] - apparently Ganon related
$7FFD68[0x01] - apparently Ganon related

$7FFD80[??] - unknown

$7FFE00[0x80] - statue sentry uses if for something
$7FFE80[0x80] - statue sentry uses it for something
$7FFF00[0x80] - statue sentry uses it for something
$7FFF80[0x80] - free ram?

----------End of variables----------

Routines:

$333
While this routine appears to be complicated and technical, it really serves two purposes.

Depending upon the entry point to the routine, two 16 bit values will be written to VRAM. The value at $00 goes to $0000-$1FFF and the one in $02 goes to $6000-$67FF. That's it. The values are written non-incrementally, so it's just those two values getting written over and over again.

$07C0
Zeroes out the first **$2000 bytes** of WRAM.
Checks the checksums on your save files to see if they are valid. Erases them if not.

$00082E

$0888
Loads SPC with data at specified address

$0901
Sets up address for **$8888**
Address = $198000 => $0C8000

$093D
Initializes the Screen

$094A
Saves your game.

$07C0
Sets up address for **$8888**
Address = $1A9EF5 => $D1EF5

$12A1
Upon starting this routine, inspect the 8-bit value at the long address [$00], Y
If it is positive, that value is stored at $04, Y is incremented, and the next value is stored at $03. Y increments. If the next value at [$00], Y is negative (AND 0x80 tells us this) then A will end up as 0x1, and 0x0 is the value was positive. This is stored at $07. The very same value is read in but this time we AND with 0x40, STA $05, and LSR A three times. The AND left a nonzero result, we will have A = 0x80, and if not, A = 0x00. Ultimately either are ORed with 0x01 so we have A = 0x81 or 0x01. The 0x01 tells us data will be being transferred in Mode 2 (a la register $4310). Data from the source is written to $2118.

Summary: 1st Value read: XXXX XXXX (If negative, the SR exits) -> $04
2nd Value read: XXXX XXXX -> $03
3rd Value read: AXXX XXXX (A = 0:

$D81B
If NOPed, you will not be able to pick up some types of pots in dungeons.

$10054
The subroutine is used to verify that all the save files are uncorrupted. Each save file has one mirrored slot 0xF00 bytes offset from the original. If the first file is corrupt and the mirror is fine, it will copy the mirror to the original and use it. Basically it checks if the 0x500 bytes in the slot add up to 0x5A5A. When save files are saved a checksum is calculated to make sure this constraint is met.

$17EBB
I'm going to generalize this subroutine because the data extraction method is cumbersome to figure out at a moment's glance. Sub $175F5 is used to get a sequence of codes, and after each code follows data to be put in memory at $7F4000, Y. Update: After some serious thought, I've begun to think of this as a decompression routine, possibly for OAM data.

The way the data is handled depends on the three most significant bits of the code. The number of bytes to write is determined by the five least significant bits of the code plus one. Let this number be called R.

[XXX | XXXXX]
(Code), (R - 1)

Example:
[010 | 10001]
Code = 010
R = 10001 + 1 = 10010 = 18d (d for decimal)

Codes:
[000] : Write R bytes after the code. For instance, 03 11 12 13 14 would have you write 11, 12, 13, and 14 in succession into memory.

[001] : Write the one byte after the code R times. 23 15 would make you write 15 four times into memory.

[100], [110], [101] : After the code is a 16-bit index. This means copy memory from $7F4000 + (the Index). You will copy R bytes of course

[010] : This code is used for repeatedly storing a 16-bit number, rather than the [001] case. Over all R bytes will still be written.

[011] : Whatever value is picked after the code will be incremented (R - 1) times and written R times. 34 01 for example would write 01 02 03 04 05

$6FE77-$6FFC0 - Template for the status bar. That is, set of tile indices.
When the game begins these are mapped to **$7EC700-$7EC849**

$3E245 Controls Link's movement, i.e. his speeds and different movement events, such as swimming and dashing.

2) ROM Addresses

by MathOnNapkins

!!! These addresses are understood to be from a
!!! ROM with no 0x200 byte header at the beginning
!!! Please take this into consideration

Continuguous Mappings:

$0 - $1395, **size = 1395h**
$14813 - $1787E, **size = 281Eh**

==========

BANK 00

==========

$0 - $489 - Code:

$48A - $493, **size = Ah**, **1 word each**.

Offsets for reading from SRAM. Based on an index stored at **$70:1FFE**, it will load the value **$0000**, **$0500**, **$0A00**, or **(possibly) $0F00**. The last value seems sketchy because that is an offset for the mirror of the first save slot.

DMA offsets for **Bank $7E** (usage undetermined as of yet. Probably pointers to font tiles):

$494 - $5FB, size = 168h, **1 word each**.

$494 - $49B: Used in DMA transfers as local addresses in **Bank $7E**. See **$0AD8** in the RAM addresses.
$49C - $4AB: Used in DMA transfers as local addresses in **Bank $7E**. See **$0AC0** and **$0AC2** in the RAM addresses.
$4AC - $4B1: Used in DMA transfers as local addresses in **Bank $7E**. See **$0AC4** and **$0AC6** in the RAM addresses.
$4B2 - $5B1: Used in DMA transfers as local addresses in **Bank $7E**. See **$0AC8** in the RAM addresses.
$5B2 - $5D1: Used in DMA transfers as local addresses in **Bank $7E**. See **$0ACA** in the RAM addresses.
$5D2 - $5DD: Stored to **$7EC013**. The value stored is indexed by **$7EC015**.
$5DE - $5FB: Used in DMA transfers as local addresses in **Bank $7E**. See **$0AE0** and **$0AE2** in the RAM addresses.

$5FC - $E4B - Code:

$E4C - $E53, **size = 9h**

????

$E54 - $110E Code:

$110F - $113E, **size = 30h**

????

$137B - $1395, **size = 1Ch**

????

$1396 - $15F3, **size = 25Eh**, **1 word each**.

Pointers to ROM Addresses used in DMA transfers. While these are local addresses,
The DMA transfers generally use **Bank $10**. See **$0ACC** and **$0ACE** in the RAM addresses

$15F4 - $1851, **size = 25Eh**, **1 word each**.

Pointers to ROM Addresses used in DMA transfers. While these are local addresses,
The DMA transfers generally use **Bank $10**. See **$0AD0** and **$0AD2** in the RAM addresses

$1852 - $1887, **size = 36h**, **1 word each**.

Pointers to ROM Addresses used in DMA transfers. While these are local addresses,
The DMA transfers generally use **Bank $10**. See **$0AD4** and **$0AD6** in the RAM addresses

$1888 - $18AA, **size = 23h**, **1 byte each**.

Upper byte of a series of VRAM Target addresses, that are used depending on the situation.
Table is indexed by **$7E0116**

$18AB-$18BF NULL

$18C0 - $18DF

$1B52 - $

$4FF3-$528F, **size = 29Dh** - indexed (long) table to compressed graphics packets

Each of these packets is treated differently: i.e. **$0A** = Dungeon sprite

$5B57 - $????

?????

$6073 - graphics data? Appears to come in 8 byte chunks.

$7C9C -

==========

==========

$C6FC, size = Dh - Falling Item Data

Use the entrance number of the dungeon, divided by two to obtain an index into this array.
The possible values are 00 - nothing, 01 - pendant of courage, 02 - pendant of widsom,
03 - pendant of power, or 06 - crystal. 1 byte per dungeon value. See data in $48D90 as it is
related to this table.

****

$DB69 - $DDE8 Dungeon Secrets Data

size = 280h

This is a pointer table that uses references local addresses assumed to be in bank $01.
Since there are 320 rooms in the game there are naturally 320 different pointer entries
here. In the game's original state, however, the rooms past room 295 just contain null data.

Secrets Contents:

Data comes in 3 byte chunks and is terminated by an instance of $FFFF

First two bytes: A word that denotes the VRAM address that the item is found at.
This identifies the item as belonging to that part of the screen.

Third byte: identifies the type of item that is hiding there. e.g. $08 is a key

****

$E96E - $EB65 Array of chest information in 3 byte chunks. 168 entries in all (0xA8)

First word - Room number it is located in. MSB determines whether it is a big chest or not. (big chest if set)
Last byte - index of the type of item contained in the chest.

Note: For rooms with two keys, 5 is the chest limit, technically. You will have conflicts in what you can grab.
Normal rooms will have a limit of 6 chests, though it might be possible to create a workaround.

I'm probably going to move this array out to extended areas so we can have more chests.

****

==========

==========

****

$12475-$1252C - Code

****

$1252D-$1253B

****

$1253C-$125EB - Code

****

$125EC-$129C3

$125EC-$1262B - (0x40 entries, 1 byte each) - actual OW location to use for conglomerated OW areas
$1262C-$12633 - (0x04 entries, 2 bytes each) - adjustment for $84
$12634-$12833 - (0x100 entries, 2 bytes each) -
$12834-$1283B - (0x04 entries, 2 bytes each)
$1283C-$12843 - (0x04 entries, 2 bytes each)
$12844-$12883 - (0x40 entries, 1 byte each) - Size of overworld areas (0x00 = 512x512 pixel map, 0x20 = 1024x1024 pixel map) All other values are not considered.
$12884-$128C3 - (0x40 entries, 1 byte each) - ???? Gets written to $0717, whatever that does.

Note that the following two arrays overlap in the middle (by a range of 0x80 entries)

$12884-$128C3 - (0x80 entries, 1 byte each) - Y / X?
$128C4-$129C3 - (0x80 entries, 1 byte each) - Y / X? fill in later

****

$129C4-$12CD9 - Code

!**********

$12CDA-$12D49

size = 70h

1 word each

Tile Type listings for use in drawing dungeons

****

$12D4A-$135AB Code

****

$135AC-$135DB - (0x30 entries, 1 byte each) unknown data indexed by $7E00A8

****

$135DC-$139CB - Code

****

$139CC-$139DB - unknown data

****

$139DC-$13B87 - Code

****

$13B88-$13B8F - Bitmasks for the routine below it

****

$13B90-$13DBF - Code

****

$13DC0-$13DC7 - Data used for setting $0614-$061B

****

****

Normal Entrances: 0h - 84h
Pointers to rooms for each entrance. So entrance number one points to the first word value in this array.

$14813 - $1491C (word values)

size = 10Ah

Recall, room numbers are 16 bits, typically ranging from 0-295. Expansion will hopefully bump that up to 320 total rooms (0-319)

!**********

Scroll Edges - unsure about what these do exactly

$1491D-$14D44

size = 428h

Each entry is 8 bytes long.

Format: 1 byte each. Corresponds to hyrule magic values in the "More" aka Entrance Properties dialog box.

HU, FU, HD, FD, HL, FL, HR, FR

!**********

Normal Entrance Y Scroll

$14D45 - $14E4E

size = 10Ah

2 bytes each

!**********

Normal entrance X Scroll

$14E4F - $14F58

size = 10Ah

2 bytes each

!**********

Normal entrance Y Coordinates

; This is where Link starts off at in the dungeon when he enters

$14F59 - $15062

size = 10Ah

2 bytes each

!**********

Normal entrance X Coordinates

$15063 - $1516C

size = 10Ah

2 bytes each

!**********

Normal entrance X Camera Coordinates

$1516D - $15276

Lower bounds for scrolling (upper bounds = this plus 2)

size = 10Ah

2 bytes each

!**********

Normal entrance Y Camera Coordinates

$15277 - $15380

Lower bounds for scrolling (upper bounds = this plus 2)

size = 10Ah

2 bytes each

!**********

Normal entrance Blocksets

$15381 - $15405

size = 85h

1 byte each

!**********

Normal entrance Floor values

This tells us what kind of properties the floor has.
If 1, then it has floors that you can fall through to the next level
If 0, I think it means it can be fallen down to
If -1, has no pits
If -2, ????

$15406 - $1548A

size = 85h

1 Byte each

!**********

Normal entrance Dungeon Values

Dungeons actually in the game are numbered like so 0, 2, 4, ..., 18, 1A
1C and 1E are unused (and hence could potentially be added.) Notice they are all even numbers.
In asm routines these values are divided by two sometimes to provide an index into other arrays.
That's why. The definition of a "dungeon" is something with a key, compass, and map.

$1548B - $1550F

size = 85h

1 Byte each

FF = -1 denotes it doesn't belong to a dungeon.

!**********

Normal Entrance Doorway Type Data

This is used only when you exit the given room. For example, let's say you enter through a skull doorway in Skullwoods. But that's not really a door way, it's just a set of tiles that are no different from a hole you might create after picking up a big rock. But if you entered from a building door, the game has to know whether to put the door frame when you exit.

$15510 - $15594

1 Byte each

size = 85h

0 = no doorway when exiting
1 = draws a door frame upon exiting
2 = ???? HM says vertical doorframe. But in the original game, no doorway is vertical. It appears the designers may have intended doorways where you enter from the left or right... fits in well with my idea to implement that.

And technically, I believe it should be the other way around. 1 should be vertical and 2 should be horizontal.

!**********

Normal Entrance Ladder and BG Settings

$15595 - $15619

size = 85h

1 byte each, with the following 2 bit layout

xxxaxxxb

x - unimportant
a - if set, then Link enters on the lower level (BG2)
b - if set... not sure what happens exactly. Check the game engine to be sure.

!**********

Normal Entrance Horizontal and Vertical Scroll Properties

$1561A - $1569E

size = 85h

For these, I guess if the flags in the appropriate places are set, then the room will be able to scroll in those directions.

1 byte each, with the following layout:

xxaxxxbx

a - Horizontal flag
b - Vertical flag

!**********

$1569F - $15723

size = 85h

Normal Entrance Scroll Quandrant

1 byte each

has four different expected values
0h, 2h, 10h, 12h

Whether the HM interpretation of this is correct, I need to check.

!**********

$15724 - $1582D

size = 10Ah

Normal Entrance EXIT Door location

1 word each. Denotes, um X and Y coordinates for overworld?

!**********

$1582E - $158B2

size = 85h

Normal Entrance Dungeon Music

1 byte each

FF - same

See ZeldaFlow for more information on values under $012C

!**********

$158B3-$15B6D Code

!**********

$15B6E - $15B7B

size = Eh

Starting Location Entrances: 85h - 8Bh. Note there are 7 Starting locations all in all.

1 word each

Pointers to rooms for each starting location entrance.

!**********

$15B7C - $15BB3

size = 38h

Starting Location Scroll Ranges (still not sure how they work totally)

Each entry is 8 bytes long.

Format: 1 byte each. Corresponds to hyrule magic values in the "More" aka Entrance Properties dialog box.

HU, FU, HD, FD, HL, FL, HR, FR

!**********

$15BB4 - $15BC1

size = Eh

Starting Location X Scroll Data

1 word each

!**********

$15BC2 - $15BCF

size = Eh

Starting Location Y Scroll Data

1 word each

!**********

$15BD0 - $15BDD

size = Eh

Starting Location Link X Coordinate

1 word each

!**********

$15BDE - $15BEB

size = Eh

Starting Location Link Y Coordinate

1 word each

!**********

$15BEC - $15BF9

size = Eh

Starting Location Camera Y Coordinate

1 word each

!**********

$15BFA - $15C07

size = Eh

Starting Location Camera X Coordinate

1 word each

!**********

$15C08 - $15C0E

size = 7h

Starting Location Blockset Type

1 byte each

see Normal Entrance description

!**********

$15C0F - $15C15

size = 7h

Starting Location Floor Type

1 byte each

see Normal Entrance description

!**********

$15C16 - $15C1C

size = 7h

Starting Location Dungeon Designation

1 byte each

see Normal Entrance description

!**********

$15C1D - $15C23

size = 7h

Starting Location Ladder and BG Settings

1 byte each, with the following 2 bit layout

xxxaxxxb

x - unimportant
a - if set, then Link enters on the lower level (BG2)
b - if set... not sure what happens exactly. Check the game engine to be sure.

!**********

$15C24 - $15C2A

size = 85h

Starting Location Horizontal and Vertical Scroll Properties

For these, I guess if the flags in the appropriate places are set, then the room will be able to scroll in those directions.

1 byte each, with the following layout:

xxaxxxbx

a - Horizontal flag
b - Vertical flag

!**********

$15C2B - $15C31

Starting Location Doorway Type Data

This is used only when you exit the given room. For example, let's say you enter through a skull doorway in Skullwoods. But that's not really a door way, it's just a set of tiles that are no different from a hole you might create after picking up a big rock. But if you entered from a building door, the game has to know whether to put the door frame when you exit.

1 Byte each

0 = no doorway when exiting
1 = draws a door frame upon exiting
2 = ???? HM says vertical doorframe. But in the original game, no doorway is vertical. It appears the designers may have intended doorways where you enter from the left or right... fits in well with my idea to implement that.

And technically, I believe it should be the other way around. 1 should be vertical and 2 should be horizontal.

!**********

$15C32 - $15C3F

Starting Location Overworld Exit Location

1 word each. Denotes, um X and Y coordinates for overworld?

!**********

$15C40 - $15C4D

Starting Location Entrance Values

1 word each. Since the starting locations have to reference an entrance, this is necessary.

!**********

$15C4E - $15C54

Starting Location Music Values

1 byte each. See ZeldaFlow.rtf for more info.

!**********

$15C55 - $15D89 Code

!**********

$15D8A-$164A2 - Exit Data (Note that some of these are fake "rooms" used in the ending sequence.)

This is not the complete listing of exit data that HM gives access to in its dialog. See _____

$15D8A-$15E27 - (0x4F entries, 2 bytes each) - Rooms that exit to overworld Areas ("Room" in HM)
$15E28-$15E76 - (0x4F entries, 1 byte each) - Overworld area that the exit leads to. ("Map" in HM)
$15E77-$15F14 - (0x4F entries, 2 bytes each) - VRAM locations to place Link at. Gets fed to $7E0084 (???? in HM)
$15F15-$15FB2 - (0x4F entries, 2 bytes each) - Y Scroll Data
$15FB3-$16050 - (0x4F entries, 2 bytes each) - X Scroll Data
$16051-$160EE - (0x4F entries, 2 bytes each) - Link's Y Coordinate
$160EF-$1618C - (0x4F entries, 2 bytes each) - Link's X Coordinate
$1618D-$1622A - (0x4F entries, 2 bytes each) - Camera Y Coordinate
$1622B-$162C8 - (0x4F entries, 2 bytes each) - Camera X Coordinate
$162C9-$16317 - (0x4F entries, 1 byte each) - I assume these are sequencing variables to tell the camera where to go? Writes to **$0624** and **$0625** (Ukn1 in HM)
$16318-$16366 - (0x4F entries, 1 byte each) - I assume these are sequencing variables to tell the camera where to go? Writes to **$0628** and **$0629** (Ukn2 in HM)

$16367-$16404 - (0x4F entries, 2 bytes each) - Door type setting 1. Writes **to $0696** (See Zelda_RAM.log for in-depth description)

0x0000: no change
bmmmmmmm mmmmmmmm: (bit pattern)
b - if set, it's a bombable exit, otherwise it's a wooden exit.
m - (15-bit number) map16 address of the exit

$16405-$164A2 - (0x4F entries, 2 bytes each) - Door type setting 2. Write to $0698.

0x0000: no change
smmmmmmm mmmmmmmm: (bit pattern)
s - if set, it's a palace exit, otherwise it's a sanctuary exit.
m - (15-bit number) map16 address of the exit

!**********

$164A3 - $166E0 Code

!**********

$166E1 - $167E0

Size = 100h

thingamajig with scroll ranges but for ending sequence too tired right now to do.

0x20 bytes per section. 1 byte per entry in each section.

The sections are interlaced and are written like so:

Section -> Address
$166E1, Y -> $0600
$16701, Y -> $0602
$16721, Y -> $0604

$16741, Y -> $0606
$16761, Y -> $0610
$167A1, Y -> $0612
$16781, Y -> $0614
$167C1, Y -> $0616

!**********

$167E1 - $16800?

??????

1 word each

Gets written to **$0708**

!**********

$16801-$16850 Exit Data (the ones that warp between overworld areas)
$16801-$16810 - Direction Link is facing when entering
$16811-$16820 - Sprite graphics index for the new area.
$16821-$16830 - Graphics index for the new area
$16831-$16840 -

!**********

$16851 - $16AE4 Code

!**********

$16AE5 - $16C38

Size = 154h

Ending Sequence (again?)
Each entry in all arrays is 1 word long.

$16AE5 - $16B06

Ending Sequence Overworld Areas to load.

$16B07 - $16B28

Ending Sequence VRAM locations for link.

$16B29 - $16B4A

Ending Sequence Y Scroll Data

$16B4B - $16B6C

Ending Sequence X Scroll Data

$16B6D - $16B8E

Ending Sequence Link Y Coordinate

$16B8F - $16BB0

Ending Sequence Link X Coordinate

$16BB1 - $16BD2

Ending Sequence Camera Y Coordinate

$16BD3 - $16BF4

Ending Sequence Camera X Coordinate

$16BF5 - $16C16

Ending Sequence ?????

Gets stored to **$0624**

$16C17 - $16C38

Ending Sequence ?????

Gets stored to **$0628**

!**********

$16C39 - $16CF7 Code

!**********

$16CF8 - $16D07

Size = 10h

Overworld Whirlpool Locations

1 word each

!**********

$16D08 - $16DC4 Code

!**********

$16DC5 - $16EC4

Overworld Bombable Wall Locations

1 word each for each overworld area

I guess array contains info on which part of the tile map to look to draw the "opened" door overlay and all that. So, once you bomb a wall, the 2nd bit of the $7EF280, X array is set, and it will remember to use this array to draw
that overlay.

!**********

$16EC5 - $1700C Code

!**********

$1700D - $17030 Jump Tables

!**********

$17031 - $171FB Code

!**********

$171FC - $1720D Jump Table

!**********

$1720E - $17252 Code

!**********

$17253 - $17272 Jump Table

!**********

$17273 - $1787E Code

****

$1787F - $17884

****

$17885 - $17888

size = 4h

Tile Position Offset, probably used in decompression routine.

1 word each, 2 entries.

!**********

$17889 - $1788C

#$0000. Used for decompression and graphics routines. Don't ask.
#$0020. same

!**********

$1788D - $1794C

Size = C0h

Flags for loading tiles or graphics. Not sure on the exact usage.

1 byte each.

Although there are only about 82h areas in the overworld, the rest of this array is zeroes so I can't make heads or tails of it. So we're going to assume they planned more outdoor areas. for now (C0h)

****

$1794D - $17D0C

Size = 3C0h

Overworld Map Data

$1794D-$17B2C - Compressed Map32 Data (the high bytes of each word of map32 data)
$17B2D-$17D0C - Compressed Map32 Data (the low bytes of each word of map32 data)

referenced at locations 0x1759E and 0x175C9 in the rom by means of a LDA $....., X

!**********

$17D0D - $17F6D Code

****

$18000-$1B3FF - Map32 to Map16 conversion values (upper left corners)
$1B400-$1E7FF - Map32 to Map16 conversion values (upper right corners)

See **$20000** as well.

!**********

$1E800-$1EB04? - Compressed data that will be written to **$7F6000**.

****

$20000-$233FF - Map32 to map16 conversion values (lower left corners)
$23400-$267EF - Map32 to Map16 conversion values (lower right corners)

See **$18000** as well.

****

$26CC0-$26F2E - Pointers to dungeon overlays + data (e.g. pits that appear)

****

$26F2F-$271CC - Pointers to dungeon layouts + data

****

$271CD-$271DD - extra 3-byte object data that seems in some form another to relate to the watergate
potentially related to other phenomena

****

$271DE-$27369 - push block data

4 bytes per entry

1st word: Room that the block exists in.
2nd word: Position of the block (expressed as tilemap coordinate)

****

$2736A-$27489 - (inter room) torch data

Consists of variable length segments consisting of:

1st word: Room that the torches belong in

After this is a series of words, each one being a tilemap address for a torch.

This list of torches is terminated by the word value 0xFFFF

****

$2748A-$27501 - initial equipment and misc. values for save game files

When a new save game file is created, this 0x3A byte array gets copied
to data starting at the offset ($700340 + (0x500 * save file slot number)

It actually contains two different initial equipment setups. The first 0x3C bytes
are used with a cheat code of sorts meant to help play test the game by the programmers, beta testers, etc.
It gives you nearly all equipment and puts you at a place in the game just after saving Zelda.

The second 0x3C bytes is used for the usual expected initial equipment setup that players
all experience.

Routine $65A4D uses this data array.

****

$27502-$27FEF - Array of pointers to header information. Each header is 14 bytes long.

$27502-$27781 - 2 byte pointers to headers, 1 per room

There are pointers for 320 rooms

$27781-$27FEF - Actual header data

Header contents:

byte 0: aaab bbcd
the a bits are transformed into 0000 0aaa and stored to $0414 ("BG2" in Hyrule Magic)
the b bits are transformed into 0000 0bbb and stored to $046C ("Collision" in Hyrule Magic)
the c bit is unused
the d bit is stored to $7EC005 (If set, use a lights out routine in the room transition)

byte 1: aabb bbbb
the a bits are unused
the b bits are transformed into bbbb bb00, thus making them a multiple of 4.
This value is used to load 4 different palettes for the dungeon, and corresponds to,
you guessed it, Palette # in Hyrule Magic!

The resulting index is used to load values for $0AB6, $0AAC, $0AAD, and $0AAE

byte 2: gets stored to $0AA2 (GFX # in Hyrule Magic)

byte 3: value + #$40 gets stored to $0AA3 (Sprite GFX # in Hyrule Magic)

byte 4: gets stored to $00AD ("Effect" in Hyrule Magic)

byte 5: gets stored to $00AE ("Tag1" in Hyrule Magic)

byte 6: gets stored to $00AF ("Tag2" in Hyrule Magic)

; These are the planes to use for bytes 9 through D. This determines which
; BG you appear on, and possibly more.

byte 7: aabb ccdd
the a bits are transformed into 0000 00aa and stored to $063F
the b bits are transformed into 0000 00bb and stored to $063E
the c bits are transformed into 0000 00cc and stored to $063D
the d bits are transformed into 0000 00dd and stored to $063C

; Note, the only safe values for a plane seem to be 0,1, or 2. Hyrule Magic
appears to violate this rule by letting you put 3 down, but nothing higher.

byte 8: aaaa aabb
the a bits are unused
the b bits are transformed into 0000 00bb and stored to $0640

byte 9: stored to $7EC000 These are all room numbers that you could possibly exit to.
byte A: stored to $7EC001
byte B: stored to $7EC002
byte C: stored to $7EC003
byte D: stored to $7EC004

****

Euclid says:
actually here's complete list of spells, i haven't manage to find out what the others are for yet.
3B287 - fire/ice rod
3B28A - 3 spells
3B28D - magic powder
3B290 -
3B293 - Red Staff (block making one)
3B296 -
3B299 - torch
3B29C -
3B29F - Blue Staff (start using)
Euclid™ says:
they're 3 bytes each, first one for normal, second one for 1/ 2 magic third for 1/4 magic

****

$41543-$4154D - Bomb timers for each explosion state

1 byte each. There are 0x0C different bomb states

****

$484E8-$4857F - Memory locations to write to when a chest is opened.
e.g. When you get a tempered sword, the game has to know to write to $7EF359.
So sword chests are mapped in such a way to write to that location.

There are $4C => 76 entries in this array, but obviously some are repeated locations b/c there is more than one type of boomerang, more than one type of sword, etc...

$48580-$????? - Byte values to write to the locations described in $484E8, X above. e.g. a chest containing the golden sword would point to a value of 4 somewhere in this array.

****

$48B90-$48B96 - Table of Item Values to give. When items fall they use a number from 0-6 as a base index.

size = 7h

That value is used as an index into this table, which has the following values in the original rom:

0 -> $10 - Ether Medallion
1 -> $37 - Pendant of Courage
2 -> $39 - Pendant of Power
3 -> $38 - Pendant of Courage

4 -> $26 - LiarHeart (according to HM).
5 -> $0F - Bombos Medallion
6 -> $20 - Crystal

****

Weathervane stuff

Note: A word about overworld coordinates. The coordinate for 0,0 is at the top left corner
OF THE WHOLE OVERWORLD (this applies to areas 0 through 0x7F, not the others). Just because you are looking at something in hyrule magic, doesn't mean
that top left corner of that map section is (0,0). For example, look at the bird's coordinates below.
Looking at it in Hyrule Magic, you'd assume that they are roughly the same, but area #$18 happens to be on the left edge
of the map, whereas it is about 2-3 whole screens down in terms of Y position. Hence the bird's
Y coordinate is about 4 times the size of it's X coordinate.

$48CD5-$48D10 - Initial position data for the weathervane piece sprites (12 of them)

$48CD5 ???
$48CE1 ???
$48CED - 12 bytes. Lower Y coordinate for all 12 pieces
$48CF9 - 12 bytes. Lower X coordinate for all 12 pieces
$48D05 ???

$48D65 - 1 byte. Upper Y coordinate byte for all 12 pieces. Normally #$07
$48D72 - 1 byte. Upper X coordinate byte for all 12 pieces. Normally #$02

Caution: edits to these must be exact and careful, because they are embedded in code. Or rather,
they are actually part of the code itself.

$48DC2 16bit Y coordinate for the bird to initialize to. Normally #$0788
$48DC7 16bit X coordinate for the bird to initialize to. Normally #$0200

$3A425 - 1 byte. Overworld area to use with the weathervane. Normally this reads #$18. Make sure to use hex.
$3A42C - 2 bytes. Overworld Y coordinate lower boundary for triggering the weather vane. Normally reads #$0760
$3A431 - 2 bytes. Overworld Y coordinate upper boundary " ... Normally reads #$07E0
$3A438 - 2 bytes. " X " lower " " ... Normally reads #$01CF
$3A43D - 2 bytes. " X " upper boundary " ... Normally reads #$0230

^So basically that defines a square that is $60 by $80 pixels, that serves as a trigger location,
given the right conditions. Edit these to your heart's content.

****

$4C635-$4C6F4 - Overworld widths of areas?

size = 0xC0 bytes, one per Overworld area

values of either 0x04 or 0x02

****

"Beginning" corresponds to values of $7EF3C5 of 0 or 1.
"First Part" corresponds to a value of $7EF3C5 at 2.
"Second Part" corresponds to values of $7EF3C5 at 3.

$4C881-$4C900 - Array of pointers to OVERWORLD sprite and overlord information, for "Beginning" mode.

It has information for ONLY areas $0-$3F. (Notice that means light world, and hence why Dark World enemies use light world enemies
if you warp there during the beginning. They're reading the wrong data. In fact, it's Light world "First Part" data that it
is inadvertently reading.

The sprite / overlord information comes in 3 byte clusters. The game reads until it spots a $FF byte. (The $FF after the last cluster)
Since the overworld utilizes a sprite map, there can be many more than 16 sprites in memory in one overworld area. They are killed off
and created as they come into view.

First Byte: Y coordinate, which will be converted into pixel units later.

Second Byte: X coordinate, which will be converted into pixel units later.

Third Byte: Sprite or Overlord type. If a sprite, this will get loaded into a slot at $0E20, X. If an Overlord, will get loaded to $0B00, X.

This operates a bit differently from Dungeon overlords and sprites. Any value greater than or equal to $F3 will be considered an Overlord. $F3 is then chopped off of that value and used as the Overlord index.

$4C901-$4CA20 - Array of pointers to Overworld sprite and overlord information, for "First Part" mode.

Basically the same as the array above, but for a later point in the game.

$4CA21-$4CB40 - Array of pointers to Overworld sprite and overlord information, for "Second Part" mode.

Basically the same as the array above, but for a later point in the game.

****

$4D62E-$4D92E - Array of pointers to DUNGEON sprite and overlord information. Pointers are local, and each room has a listing of sprites. They are in order of rooms.

size = 300h

Each entry is 2 bytes long.

NOTE: Interesting to note that this pointer list seems to have space for $180 rooms, whereas the built in limit is $128. That's the first I've seen of an allocated array in the rom being larger than necessary. O_o.

Layout of these listings:
First Byte: Stored to $0FB3. Not sure what the purpose of that is yet.

After the first byte, you will see 3 byte clusters that break down thus:

```
First Byte: 7  6  5  4  3  2  1  0
            |  |  |  |  |  |  |  |
            |  |  |  \--+--+--+--+-- Y coordinate (in pixels from the top, starting at 0) of the sprite divided by 16.
            |  |  |
            |  \--+-------------- If these are set, they will be used to generate a subtype, stored to $0E30, X.
            |
            \--+--+------------ If set, the sprite is on BG2, if not it's on BG1

Second Byte: 7  6  5  4  3  2  1  0
             |  |  |  |  |  |  |  |
             |  |  |  \--+--+--+--+-- X coordinate (in pixels from the far left, starting at 0) of the sprite divided by 16.
             |  |  |
             \--+--+------------ If all these bits are set then this is an Overlord, otherwise it's a normal sprite.
                                 If only some of them are set they can be used to generate a subtype, stored at $0E30, X
```

Third Byte: Sprite or Overlord type. If a sprite, this will get loaded into a slot at $0E20, X. If an Overlord, will get loaded to $0B00, X.

The array is terminated with a byte of $FF.

****

Sprite Statistics. All of the following arrays are $F3 bytes long (as expected, since there are $F3 different sprite types).

$6B080, X - ??? Gets stored to $0E40, X.
$6B173, X - Initial HP for all sprites. Gets stored to $0E50, X
$6B266, X - Amount of damage the sprite can cause to Link.
$6B359, X - Possibly a main palette index. Stored to $0E60, X
$6B4CC, X -
$6B8F1[0x???]

****

$70000-$70FFF - BG3 VWF graphics (uncompressed)

View this area in a tile editor and you'll see the text characters for the VWF.

****

$74703-$747C6 - Pointers for dictionary entries. (Start point and upper bound)

****

$75460 - location of palette indices (grouped in 4's)
These get mapped to **$0AB6**, **$0AAC**, **$0AAD**, and **$0AAE** respectively.

****

$78000 - Map16 to Map8 conversions (comes in groups of 8 bytes, 4 Map8 tiles each)

****

$7902A - ??? Gets mapped to **$7EFF40** to **$7EFFBF**

****

$80000-$86FFF - 4bpp graphics sets for all of Link's animations

****

$87000-$8B7FF - 3bpp graphics sets (uncompressed)

****

3bpp?
$C0D64 - location of compressed graphics for action sprites, (shields and shovel)

2bpp
$C2F0D - location of compressed graphics for the menu screen,
and overlaps with graphics for the dungeon map screen. (includes text for items, and the map icon)

2bpp
$C3520 - location of compressed graphics for the menu screen (bug catching net, medallions, etc)

2bpp
$C3953 - location of compressed graphics for the menu screen (magic mirror, book of mudora, etc)

****

$DB800-$DB85F - Hole Data

$DB800-$DB825 - (0x13 entries, 2 bytes each) modified (less 0x400) map16 coordinates for each hole
$DB826-$DB84B - (0x13 entries, 2 bytes each) corresponding area numbers for each hole
$DB84C-$DB85F - (0x13 entries, 1 byte each) corresponding entrance numbers

!**********

$DB860-$DB8BE Code

!**********

$DB8BF-$DBBF3 Overworld accessible entrances (other entrances are accessible via other methods, like game startup)

$DB8BF (0x2C entries, 2 bytes each) - valid map8 (CHR) values for entrances (left side)
$DB917 (0x2C entries, 2 bytes each) - valid map8 (CHR) values for entrances (right side)
$DB96F (0x81 entries, 2 bytes each) - area numbers for each entrance
$DBA71 (0x81 entries, 2 bytes each) - map16 coordinates for each entrance
$DBB73 (0x81 entries, 1 byte each) - entrance numbers (ranging from 0x00 to 0x84)

!**********

$DBBF4-$DBF4B Code

****

$DBF4C-$DBF63 Data (unknown type)

****

$DBF64-$DC2F8 Code

!**********

$DC2F9-$DC8A3 - Overworld Secrets (Item) Data

$DC2F9-$DC3F8 - Local pointer table (2 bytes, 0x80 entries) to the data for each Area
$DC3F9-$DC89B - The actual data, each entry being 3 bytes.
The first two bytes are the map16 coordinate for the secret.
The last byte is the value of the secret. Valid values are 0x00 to 0x16, and 0x80, 0x82, 0x84, 0x86, 0x88
$DC89C-$DC8A3 (4 entries, 2 bytes each) - replacement map16 values. E.g. a hole secret would have a hole map16 as the replacement

Note: Areas 0x80 and above don't have items.

****

$DC8A4-$?????

****

$E0000-$E7F29 - Text / Dialogue Data

(see also **$76E20** for a bit more text data in this same format)

Some quick notes: the text box is 0x15 tiles (horiz.) by 0x06 tiles (vert.)
Each tile consists of 0x10 bytes, as they are 2bpp.

Characters:

$0 to $19: 'A' through 'Z'
$1A to $33: 'a' through 'z'
$34 to $3D: '0' through '9'
$3E: '!'
$3F: '?'
$40: '-'
$41: '.'
$42: ','
$43: '...'
$44: |> (arrow pointing right)
$45: '('
$46: ')'
$47: Ahnk (used in Hylian script)

$48: three waves (used in Hylian script)
$49: snake? (used in Hylian script)
$4A: picture of Link's head in the kidnapper signs (left half)
$4B: picture of Link's head in the kidnapper signs (right half)
$4C: "" (left orientation)
$4D: small arrow pointing up
$4E: small arrow pointing down
$4F: small arrow poitning left
$50: small arrow pointing right
$51: '" (apostrophe)
$52: heart piece upper left filled (just left side)
$53: heart piece empty (just right side)
$54: heart piece left filled (just left side)
$55: heart piece 3/4 filled (just left side)
$56: heart piece upper right filled (just right side)
$57: heart piece all filled (just left side)
$58: heart piece all filled (just right side)
$59: space (as in)
$5A: <| (arrow pointing left)
$5B: 'A' in bold. (indicates the A button)
$5C: 'B' in bold. (indicates the B button)
$5D: 'X' in bold. (indicates the X button)
$5E: 'Y' in bold. (indicates the Y button)
$5F: alternate "I" or "I"? (apparently not used)
$60: alternate "!" (apparently not used)
$61: upside down "!" (apparently not used)
$62 to $65: apparently tab characters or space characters? (apparently not used)
$66: strange red and white '.' (apparently not used)

Commands:

$67 [NextPic] command
$68 [Choose] command
$69 [Item] command (for waterfall of wishing)
$6A [Name] command (insert's player's name)
$6B [Window XX] command (takes next byte as argument)
$6C [Number XX] command (takes next byte as argument)
$6D [Position XX] command (takes next byte as argument)
$6E [ScrollSpd XX] command (takes next byte as argument)
$6F [SelChng] command
$70 [Crash] command (obviously that's probably not what it's intended to do but that's what HM lists it as)
$71 [Choose2] command
$72 [Choose3] command
$73 [Scroll] command
$74 [1] command (aka [Line1])
$75 [2] command (aka [Line2])
$76 [3] command (aka [Line3])
$77 [Color XX] command (takes next byte as argument)
$78 [Wait XX] command (takes next byte as argument)
$79 [Sound XX] command (takes next byte as argument)
$7A [Speed XX] command (takes next byte as argument)

(this will be the Black Magic syntax for the following three codes. simple eh? Do NOT use these. Ever)

$7B [Command 7B]
$7C [Command 7C]
$7D [Command 7D]

$7E [Waitkey] command
$7F stop command (ends the whole message)

$80 signals to switch to the second set of text data (much saller)

$81 - $87 unused (don't use these!)
$88 - ??? Dictionary encoding bytes. Represents longer strings

****

$F4EBC - Message IDs for Maidens

-- fill in later.

****

$F8000, **size = 3C0h** (**140h entries each 3 bytes in length**)

- Location of type, layout, and object information for each Dungeon room.
It's a table of 24 bit pointers with an entry for each room. (all 320 of them ;))

Each long snes cpu address in the table points to a structure with the following layout:

byte layout:

byte0: aaaa bbbb. The a bits are transformed to aaaa0000 and select the type of
empty space to fill in. Hyrule Magic calls this 'Floor 1' (Gets stored to **$7E0490)**

The b bits are transformed to bbbb0000 and are the what
Hyrule Magic calls 'Floor 2' (Gets stored to **$7E046A)**

byte1: aaab bbcd The a bits are unused and should not be used
The b bits determine the room's layout type, ranging from 0 to 7.
The c and d bits are unknown, but I have a feeling it's related to $AA and $A9

After that the bytes come in 3 or 2 byte object structures, used by routine **$01:88E4**.
Objects are loaded until an object with value **0xFFFF** occurs.
If a value **0xFFF0** is loaded, the game will start loading Type 2 objects
And will not go back to loading Type 1 objects until it is time to load
the next layer. (Layer as in HM, not to be confused with the SNES' Backgrounds.
A value of **0xFFFF** will also terminate the loading of Type 2 objects.
The routine immediately terminates if that happens during the loading of either object type.

Type 1 Object structure: (3 bytes)

Third Byte: Routine to use. - **If this byte is >= 0xF8 and < 0xFC**, then it is a subtype 3 object.
- **If the index is >= FC**, it is a subtype 2 object. If not, it is a subtype 1 object.

Subtype 1 Objects ----------

First and Second Byte: High Byte Low Byte
yyyy yycc xxxx xxaa

The a bits are stored to $B2
The c bits are stored to $B4
The x and y bits are transformed into: 000y yyyy yxxx xxx0
This is a tilemap address that indexes into $7E2000 and / or $7E4000

Use the third byte * 2 as an index into the table at $8200
This is the routine that is used to draw and otherwise handle the object.
Subtype 1 objects have a maximum width and height of 4. width and height are measured in terms of 32 x 32 pixels. (<-- last part is questionable)

Subtype 2 Objects ----------

Subtype 2 objects are those with an index >= 0xFC

1st, 2nd, & 3rd bytes: Bank Byte High Byte Low Byte
ffdd dddd eeee cccc aaaa aabb

The a bits are unused, but after all they are the marker for this type of object subtype.

The b, c, e, and f bits are transformed into a VRAM tilemap address:

000c cccf fbbe eee0

Might I add this is one messed up format?

The d bits are used as an index into the table at $8470. Since such indicies are going to be even, the d bits are transformed into: 0000 0000 0ddd ddd0

Subtype 3 Objects ----------

Similar to Subtype 1, with a few small exceptions.

The vram address is calculated the same way. However, $B2 and $B4 are not used as length or width dimensions here. The routine that is used is determined as follows:

Take the original index (times two) that a Subtype 1 would have used. AND that with 0x000E. Then shift left 3 times to produce 0000 0000 0eee 0000. Then, OR in $B2 and $B4 and shift left once, so the final result is:
0000 0000 eeea abb0.

Also, this value indexes into **$85F0** instead of **$8200**.

Type 2 Object Structure: (2 bytes)

High Byte Low Byte

cccc cccc bbbb ddaa

The a bits form a 2-bit value (0000 0aa0) that determines the routine to use for the object. In Hyrule Magic, corresponds to the "direction" of the door.

The b bits are transformed into 000b bbb0 and stored to **$02** -> X.
Corresponds to "Pos" of door objects in the Hyrule Magic. Note that these range from 0x00 to 0x16 (always even) which if you halve those values is 0 - 11 in decimal. This is easily verifiable in Hyrule Magic.

The c bits are shifted into the lower byte and stored to **$04** -> A and $0A. This is later used to grab the tiles used to draw the door and the area below it.
In Hyrule Magic, corresponds to "type". Note the type is 1/2 of the number listed here. This is because to avoid using an ASL A command, the c bits are always even.

The d bits are unused.

Next I'll go into the nitty gritty of the various types (the value of **$04**). Again note that we'll only be dealing with even values b/c that's what you'll see in the code. To convert between here [as well as the code] and Hyrule Magic, take the hex value here and divide by two. Convert to decimal and that's your Hyrule Magic "type."

Types:

0x00 - Basic door. Index = **$0460**
0x02 - Normal door?
0x04 - ???
0x06 - ???
0x08 - Waterfall door (only used in Swamp palace; in one room at that!)
0x0A - ???
0x0C - Trap door (probably other types but this seems to be most common)

0x12 - Adds a property to some doors allowing you to exit to the overworld
(this is accomplished by writing to the tile attribute map)
0x14 - Transition to dark room?
0x16 - Toggles the target BG Link will emerge on. e.g. if Link starts on BG0 in the next room he'll be on BG1.

0x20 - Locked door specifically for BG0.
0x22 - "
0x24 - Locked door for either BG0 or BG1
0x26 - "
0x30 - Large exploded pathway resulting from a switch being pulled (unusual to have as a door as it's huge)
0x32 - Sword activated door (e.g. Agahnim's room with the curtain door you have to slash)
0x46 - warp door?

****

$F83C0 - Loaded when the header is loaded for each dungeon room. These appear to be the direct offset to where

the door objects appear. The normal objects and the door objects are seperated by the byte sequence $FFF0. Apparently
the offsets in this set are just there for convenience?

3) Special Effects

Guide to the code of Zelda 3 (A Link to the Past) Special Effects Mechanics
by MathOnNapkins

Special Effect 0x07 - Bombs

$0280[**0x0A**] - ??? timer?
$028A[**0x0A**] - ???
$0294[**0x0A**] - gravity indicator
$029E[**0x0A**] - the bomb's elevation from the ground in pixels
$02A8[**0x0A**] - fraction part of the bomb's elevation

$0380[**0x04?**] - carrying status (probably also includes cane of somaria blocks)

0 means not being carried. **1 to 3** are various stages of the bomb being picked up by Link
3 means it is in Link's hands ready to be thrown

$0385,**X** - **nonzero** if the bomb is flying through the air (in motion)
$039F,**X** - countdown timer

This timer starts at **0xA0** and counts down to **0**. At a certain point it causes the bomb to flash
When it reaches **0** it initiates the explosion sequence

$03B1[**0x0A?**] - when Link is picking up a bomb, this is a delay timer for the animation frames

$03C0[**0x02**] used to indicate that this slot is filled with a bomb (or rock fall)
$03C2[**0x08**] - ???
$03CA[**0x0A**] - like **$0C7C**, this is a floor selector for the bomb, but mirrors the less used **$0476**
$03D5[**0x0A**] - transition flag indicating that the bomb needs to switch floors
$03E1[**0x02?**] - ???
$03E4[**0x02**] - tile type the bomb interacts with
$03EA[**0x0A**] - gah
$03F4[**0x0A**] ???
$03FD - the only thing that seems to set this is the bird (and only indoors oddly enough)

$0BFA[**0x0A**] - Y coordinate low byte
$0C04[**0x0A**] - X coordinate low byte
$0C0E[**0x0A**] - Y coordinate high byte
$0C18[**0x0A**] - X coordinate high byte
$0C22[**0x0A**] - Y velocity

$0C2C[0x0A] - X velocity
$0C36[0x0A] - ???
$0C40[0x0A] - ???
$0C54[0x0A] not used?
$0C5E[0x0A] graphical state of the bomb (from 0 to 0x0B each step progresses from normal to fully exploded)
$0C68[0x0A] - 0 or 2?
$0C72[0x0A] - 0x10 if bomb has touched the floor this frame
$0C7C[0x0A] - floor selector
$0C86[0x0A] - the starting point in the OAM buffer for the sprites of the special effect
$0C90[0x0A] - number of sprites the special effects needs * 4 (since each OAM entry is 4 bytes)

Alternates between 0x10 and the direction Link was facing when he laid the bomb ($7E0074)
Someone please explain why this is needed :(

Special Effect 0x22 - ReceiveItem

Necessary components that could be encoded in the sprite object data:

-How the object was received
-What to do when the timer is about to expire
-What to do when the timer expires (finishing move?)
-Whether to do an animation sequence
-Text message with perhaps additional logic for individual types of objects
-Whether to trigger the boss victory mode (debatable whether this should even part of the object)
-Additional customized options (e.g. moon pearl transforming Link instantaneously)
-Sound effects to play either when opened or when the object's counter is at the expiration state

4) SRAM (*.srm) hacking guide

Date: 12/01/2006
Version 5. Minor revisions.

Date: 11/23/2005
Version 4. Nearly Totally correct and complete. It remains to be proven that certain areas are unused.

MathOnNapkins' Zelda III SRAM (*.srm) hacking guide. .srm is the save file type used with Snes9x.

This FAQ is based off of save slot 1. There is an offset of $500 bytes for each slot. Thus,
First Game starts at $000, Second at $500, and the Third at $A00. Also note that
each slot is "mirrored," a technique that makes two copies of your save data in the same file.

Slot 1: $000; Mirror: $F00
Slot 2: $500; Mirror: $1400
Slot 3: $A00; Mirror: $1900

It suffices to edit the main copy. Note each copy has its own checksum.

-FOR ROM HACKERS-

Note that since each save file and mirror is 0x500 bytes and the length of the .srm file is 0x2000 bytes,
we have addresses $1E00-$1FFF free for our own fiendish purposes. But with that comes responsbility. Since the game engine only loads
what it needs, you will have to make sure your memory is initialized to what you want it to. Don't assume it will be zero or some other value
on startup of the machine. The three major emulators even differ on what they put there by default. (Sleuth, Zsnes, and Snes9x)

Take into consideration if you want a permanent stat to be added in, you should probably index it in such a way that you have copies
for all three save files. Note, the game for some odd reason uses address $701FFF on a temporary basis. Annoyed the hell out of me when
a romhack I was doing conflicted with that location. I thought the game didn't use anything past $1DFF at that time. Just be aware of that
one. You can use it probably, but don't save permanent data to it, like a game timer ;).

If you are going to take advantage of this extra space remember to use long writes to bank $70,
or intelligently manipulate the data bank register if you have to.

-END ROM HACKER BLOCK-

==========

Crash course in reading the FAQ

Hexadecimal Numbers: Numbers written with the $ symbol before them denote hexadecimal numbers, or numbers in base 16. For example, $521 denotes 5*(16^2) + 2*(16^1) + 1*(16^0) = 1313 in decimal. Numbers without a $ can normally be assumed to be decimals.

Hexadecimal numbers, sometimes called just hex numbers, were made to be nicer representations of
binary numbers - numbers in base two. All numbers in binary are represented with ones and zeroes.

Bit ordering in bytes: a "Byte" contains 8 bits, and they are numbered from right to left as follows-

7 6 5 4 3 2 1 0

For example, a byte with bits 2 and 6 set would look like 01000100 in binary.
$77 is the hexadecimal way to represent this number.

Each bit position corresponds to a value in base 2.

Decimal Hex

Bit	Value	Value
0	1	$1
1	2	$2
2	4	$4
3	8	$8
4	16	$10
5	32	$20
6	64	$40
7	128	$80

Some binary numbers are larger, such as 16,24,32,or even 64 bit numbers. In this case, bits of higher order correspond to greater powers of two. Ex: Bit 8 of a 16-bit number corresponds to 256. 16-bit numbers are often called "Words" as opposed to "Bytes".

****CAUTION*****

The Zelda sram relies upon an inverse checksum. That is, if you add to any byte in the .srm file, you must subtract from that save game's checksum to maintain the balance of things. Otherwise, when the game is started, Zelda.smc will detect that all the bytes in the file don't add up to the checksum, and your file will be deleted. If you know how to NOP (make ineffective) the subroutine that does this than you can work a lot easier. That is beyond the scope of this document and is a topic in 65816 assembly language, i.e. the code for the the SNES processor.

****CAUTION*****

Beginning of Address information. These offsets directly correspond to $7E:F--- when your particular save file is being played. When the game is finished it writes the information into bank $70 in the corresponding slot + offsets presented here. E.G. if you're playing the second save file, information from $7E:F000-$7E:F4FF will get saved from to $70:0500-$70:09FF, and mirrored to $70:1400-$70:18FF.

Room information: There are 296 ($128 in hex) rooms in Zelda3. Hyrule magic cannot expand the number of rooms, and I probably wouldn't want to expand them just because rearranging this map information in SRAM would painful, among other things.

Each room has one word of information devoted to it, so that means addresses $0-$24F contain the information for our dungeon levels.

$000 - $24F : Data for Rooms (two bytes per room)

High Byte	Low Byte
d d d d b k ck cr	c c c c q q q q

c - chest, big key chest, or big key lock. Any combination of them totalling to 6 is valid.
q - quadrants visited:
k - key or item (such as a 300 rupee gift)

d - door opened (either unlocked, bombed or other means)
r - special rupee tiles, whether they've been obtained or not.
b - boss battle won

qqqq corresponds to 4321, so if quadrants 4 and 1 have been "seen" by Link, then qqqq will look like 1001. The quadrants are laid out like so in each room:

```
--------------
|    |    |
| 4  | 3  |
|    |    |
|------------|
|    |    |
| 2  | 1  |
|    |    |
--------------
```

Breakdown of where the data comes from in work RAM: (we'll reprint the diagram)

```
High Byte                Low Byte
d d d d b k ck cr       c c c c q q q q
| | | | | | | |       | | | | | | | |
| | | | | | | |       | | | | \-+-+-+--- Derived from $7E0408
| | | | | | | |       | | | |
| | | | \-+-+--+--------+-+-+-+---------- Derived from ($7E0402 >> 4) in 16 bit mode
| | | |
\-+-+-+------------------------------- Derived from $7E0400
```

See routine $13947 for more details.

$250-$27F Nothing, but if we could expand the game's data a bit, we could easily use these for extra dungeon rooms.
That would bring the grand total to 320 (decimal), which would be $140 in hex.
Notice that each room has two bytes, so that would extend up to $27F.
This makes sense given this area is completely unused.
Nintendo probably just didn't end up using the extra rooms and thus never put any data in them. But it appears they planned for extras if they needed them ;).
I credit Euclid (the romhacker, not the mathematician) with encouraging me to accept this.

$280-$2FF Overworld Event Information

(one byte per area)

?ho???s?

o - If set, the area will draw its designated overlay when you enter it.

e.g. If you're triggered misery mire then the entrance shows up there the next time you enter that area.
You can view overlays in the latest release of Hyrule Magic (v0.963).
Other overlays include stairs leading underground, and the removal of the weathervane after it has exploded.

s - If set, the area will draw its secondary overlay when you enter it.

h - If set, heart piece has been collected in this area already.
Also used for a handful of other sprites.

? - unknown and probably unused.

$300-$33F Nothing again, but it also has potential to be used for more overworld data. That would allow for $C0 (192 decimal) areas over all.
(meaning 64 in addition.) Note areas $80 and $82 are used but don't save data the way the other areas do. (i.e. not here)
I believe they wanted more overworld areas, and planned to use this area.

--- Item Information ---

Bow: $340. 0 - nothing. 1 - bow w/ no arrows. 2 - bow w/ arrows. 3 - silver arrows

Boomerang: $341. 0 - nothing. 1 - blue boomerang. 2 - red boomerang.

Hookshot: $342. 0 - nothing. 1 - hookshot.

Bombs: $343. How many bombs you have. Can exceed 50, up to $FF

Mushroom: $344: 0 - nothing. 1 - Mushroom. 2 - Magic Powder

Fire Rod: $345: 0 - nothing. 1 - Fire Rod.

Ice Rod: $346: 0 - nothing. 1 - Ice Rod.

Bombos Medallion: $347: 0 - nothing. 1 - Bombos Medallion.

Ether Medallion: $348: 0 - nothing. 1 - Ether Medallion.

Quake Medallion: $349: 0 - nothing. 1 - Quake Medallion.

Torch: $34A. 0 - nothing. 1 - torch.

Hammer: $34B. 0 - nothing. 1 - magic hammer.

Flute: $34C. 0 - nothing. 1 - shovel. 2 - flute, no bird. 3 - flue, bird activated.

Bug Catching Net: $34D. 0 - nothing. 1 - bug catching net.

Book of Mudora: $34E. 0 - nothing. 1 - Book of Mudora

Bottles: $34F: 0 - nothing. 1 - has bottles.

Cane of Somaria: $350. 0 - nothing. 1 - cane of somaria.

Cane of Byrna: $351. 0 - nothing. 1 - cane of byrna.

Magic Cape: $352. 0 - nothing. 1 - magic cape.

Magic Mirror: $353. 0 - nothing. 1 - scroll looking thing that works like mirror.
2 - mirror with correct graphic.

Gloves: $354. 0 - normal strength. 1 - Power Gloves. 2 - Titan's Mitt

Boots: $355. 0 - nothing. 1 - boots. *Just having the boots isn't enough to dash.
You must have the ability flag corresponding to run set as well.
See $379.

Flippers: $356. 0 - nothing. 1 - flippers. Having this allows you to swim, but doesn't make the swim ability text show up by itself. See $379
Unlike the boots, the ability is granted, as long as you have this item.

Moon Pearl: $357. 0 - nothing. 1 - moon pearl.

???? $358. Unused? Beginning to think it really is unused.
Not even referenced in the rom by any means that I could see.

Sword: $359.
0-No sword
1-Fighter Sword
2-Master Sword
3-Tempered Sword
4-Golden Sword

See Side Note 2

Shield: $35A.

0-No shield
1-Blue Shield
2-Hero's Shield
3-Mirror Shield

See Side Note 2

Armor: $35B.
0-Green Jerkin
1-Blue Mail
2-Red Mail

See Side Note 2

Bottle: $35C-F
0-No bottle
1-Mushroom (no use)
2-Empty bottle
3-Red Potion
4-Green Potion
5-Blue Potion
6-Fairy
7-Bee
8-Good Bee

Rupees: $360-1 (Goal)
$362-3 (Actual).

This number can be set above the 999 limit.
It is a word, so you can have lots of rupees. (Up to about 65000)
One word is capacity, The other is your current supply.

THE COMPASS, BIG KEY, AND DUNGEON MAP WORD LOCATIONS ARE ALL MAPPED OUT THE SAME WAY, AS YOU WILL OBSERVE.

Note: The unused areas would correspond to dungeon numbers $1C and $1E in the variable $7E040C, but alas there are no dungeons
in the game with those values.

If you see "(doesn't exist)" it means this feature was not used in this dungeon in the original, in case you were wondering.

Compass1: $364. bit 0: Unused
bit 1: Unused
bit 2: Compass of Ganon's Tower

bit 3: Compass of Turtle Rock
bit 4: Compass of Gargoyle's domain
bit 5: Compass of Tower of Hera
bit 6: Compass of Ice Palace
bit 7: Compass of Skull Woods.

Compass2: $365 bit 0: Compass of Misery Mire.
bit 1: Compass of Dark Palace
bit 2: Compass of Swamp Palace
bit 3: Compass of Hyrule Castle 2 (doesn't exist)
bit 4: Compass of Desert Palace
bit 5: Compass of Eastern Palace
bit 6: Compass of Hyrule Castle (doesn't exist)
bit 7: Compass of Sewer Passage (doesn't exist)

BigKey1: $366. bit 0: Unused
bit 1: Unused
bit 2: Big Key of Ganon's Tower
bit 3: Big Key of Turtle Rock
bit 4: Big Key of Gargoyle's domain
bit 5: Big Key of Tower of Hera
bit 6: Big Key of Ice Palace
bit 7: Big Key of Skull Woods.

BigKey2: $367. bit 0: Big Key of Misery Mire.
bit 1: Big Key of Dark Palace
bit 2: Big Key of Swamp Palace
bit 3: Big Key of Hyrule Castle 2 (doesn't exist)
bit 4: Big Key of Desert Palace
bit 5: Big Key of Eastern Palace
bit 6: Big Key of Hyrule Castle
bit 7: Big Key of Sewer Passage (doesn't exist)

Dungeon map1: $368. bit 0: Unused
bit 1: Unused
bit 2: Map of Ganon's Tower
bit 3: Map of Turtle Rock
bit 4: Map of Gargoyle's domain
bit 5: Map of Tower of Hera
bit 6: Map of Ice Palace
bit 7: Map of Skull Woods.

Dungeon map2: $369 bit 0: Map of Misery Mire
bit 1: Map of Dark Palace
bit 2: Map of Swamp Palace

bit 3: Map of Hyrule Castle 2 (doesn't exist)
bit 4: Map of Desert Palace
bit 5: Map of Eastern Palace
bit 6: Map of Hyrule Castle
bit 7: Map of Sewer Passage (doesn't exist)

Wishing Pond Rupee Count $36A. Number of rupees in the pond

Heart pieces collected: $36B. Number of heart pieces (out of four) you have earned

Health: $36C. Goal (capacity) Health. Each increment of $08 is worth one heart. $04 is a half heart. The max is generally $A0. The game is coded to not accept health values beyond this.

$36D. Actual Health. Same as above, but this reflects your current health status rather than potential.

Magic Power: $36E. Magic power ranges from 0 to $80 (128). Each small bottle refills $10. Setting Magic above $80 causes the magic meter to glitch and you can't use special items.

Keys: $36F. Number of Keys you have in the dungeon you are currently in. You can earn keys on the overworld but they don't do anything. If you're in a non-keyed dungeon it will generally read $FF.

Bomb Upgrades: $370. Number of upgrades your bomb capacity has received. Behavior varies after a while. Will probably need recoding to be consistent.

Arrow Upgrades: $371. Number of upgrades your arrow capacity has received. Same as above, more or less.

Hearts filler: $372. Write to this location to fill in a set number of hearts. Make sure to write in a multiple of $08. Otherwise, you will end up filling the whole life meter.

Magic filler: $373. Write to this location how much magic power you want filled up. The maximum effective value is $80.

Pendants: $374. Bit 0: Courage
Bit 1: Wisdom
Bit 2: Power

Bomb filler: $375. Write to this location to add X bombs to your arsenal. It will not exceed your

maximum, as defined with $370

Arrow filler: $376. Write to this location to add X arrows to your arsenal. It will not exceed your maximum, as defined with $371.

Arrows: $377. Can exceed 70.

???? $378. ????

Ability Flags: $379. Bit 0:
Bit 1: Swim
Bit 2: Run / Dash
Bit 3: Pull
Bit 4: ----
Bit 5: Talk
Bit 6: Read
Bit 7: ----

Crystals: $37A. Bit 0: Misery Mire
Bit 1: Dark Palace
Bit 2: Ice Palace
Bit 3: Turtle Rock
Bit 4: Swamp Palace
Bit 5: Gargoyle's Domain
Bit 6: Skull Woods

Magic usage $37B. $0: normal consumption
$1: 1/2 consumption
$2: 1/4 consumption

Keys earned per dungeon:
$37C: Sewer Passage
$37D: Hyrule Castle
$37E: Eastern Palace
$37F: Desert Palace
$380: Hyrule Castle 2
$381: Swamp Palace
$382: Dark Palace
$383: Misery Mire
$384: Skull Woods
$385: Ice Palace
$386: Tower of Hera
$387: Gargoyle's Domain
$388: Turtle Rock
$389: Ganon's Tower

$38A: ??? possibly unused. (Were they planning two extra dungeons perhaps?)
$38B: ??? possibly unused.

--- Game Event Information ---

Progress Indicator (value, not bitwise)
$3C5: $0: Unset, Will put Link in his bed state at the beginning of the game. (Also can't use sword or shield)
$1: You have a sword and start in the castle on start up.
$2: Indicates you have completed the first Hyrule Castle dungeon.
$3: Indicates you have beaten Agahnim and are now searching for crystals.
$4 and above: meaningless. Though, you could write code using them to expand the event system perhaps.

$3C6: Progress Flags (bitwise)
0: Set after your Uncle gives you his gear in the secret passage. Prevents him from showing up there again.
1: Indicates that you've touched the dying priest in Sanctuary?
2: Set after you bring Zelda to sanctuary?
3: Unused? (98% certainty)
4: Set after Link's Uncle leaves your house. It's used to prevent him from respawning there.
5: Set after you obtain the Book of Mudora (this is a guess)
6: Set after you bring the dwarf back to his partner????
7: Unused? (98% certainty)

Map Icons Indicator 2 (value, not bitwise)
$3C7: $0: ????
$1: ????
$2: ????
$3: The Three Pendants
$4: Master Sword in Lost Woods
$5: Agahnim (skull icon at Hyrule Castle)
$6: Just crystal 1 shown (Sahasrala's idea)
$7: All crystals shown
$8: Agahnim (skull icon at Ganon's Tower)

All values beyond 8 are invalid, it seems.

$3C8: Abbreviations:
LH = Link's House;
SA = Sanctuary;
MC = Mountain Cave;
PP = Pyramid of Power in DW;

0: Start the game in Link's house always
1: SA
5: LH or SA or MC

Progress Indicator 3 (bitwise)
$3C9: 0: If set, means the bum gave you his bottle already.
1: If set, means that the salesman in the village sold you a bottle already
2:
3: Flute Boy (DW) has been arborated
4: Thief's Chest has been opened by the middle aged guy
5: After you save the Smithy's partner, this bit gets set.
6:
7: Means Smithies have your sword. Once they give it back it's no longer set. (so nonpermanent)

Lightworld / Darkworld
$3CA: 0:
1:
2:
3:
4:
5:
6: If set, we're in dark world. Other wise, in light world.
7:

Unused?
$3CB: ?????

Tagalong Indicator (who is following you, if anyone?) (value based)
$3CC: $0: no one
$1: Princess Zelda
$2: ???
$3: ???
$4: Old Man?
$5: Zelda (invisible) bitching at you about coming to rescue her
$6: Blind masquerading as a Maiden
$7: Missing Dwarf (smithyfrog as I call him) in DW
$8: Missing Dwarf in LW
$9: Middle Aged Guy w/ Sign
$A: Kiki the monkey
$B: ???
$C: Thief's chest
$D: Super Bomb
$E: After a boss fight, this invisible tagalong is present
$F and beyond: don't use. will crash the game

; following values are related to the old man on the mountain sequence but it's not quite clear how
$3CD: ????

$3CE: ????

$3CF: ????

$3D0: ????

$3D1: ????

$3D2:

$3D3: Set to 0 normally. Set to $80 if a Super Bomb is going off.

$3D4-$3D8: ????

Player's Name
$3D9-$3E4: See appendix for listing of character codes. Note each of the six letters is represented by a 16-bit number.

Validity (Checksum) of the File:
$3E5-$3E6: There is a subroutine in the ROM that checks to make sure this value is 0x55AA.
(Note the reverse byte order in the actual SRAM.)
If you alter this your file is automatically tagged for deletion at startup.
The game is designed to delete it, it's not a Super NES feature or anything.
In short, Don't mess with it. Unused game slots have the value 0x0000 here,
and you will too if you mess with it - resulting in your save file(s) being wiped
out.

$3E7-$402 (?) Deaths totals for each dungeon. Each number is 16 bit. Thanks to Euclid for helping verify this!
$3E7: Sewers
$3E9: Hyrule Castle
$3EB: Eastern Palace
$3ED: Desert Palace
$3EF: Hyrule Castle 2
$3F1: Swamp Palace
$3F3: Dark Palace
$3F5: Misery Mire
$3F7: Skull Woods
$3F9: Ice Palace
$3FB: Tower of Hera
$3FD: Gargoyle's Domain
$3FF: Turtle Rock
$401: Ganon's Tower

Life/Save Counter:
$403-4. Counts the number of times your saved or died in the game, before you beat it.

PostGame Death Counter:
$405-6. When you start the game this is written to with the value -1 (0xFFFF). On the game select screen, it will only display a number if this is not 0xFFFF. The max displayable number is 999. When you beat the game, the number of times you died gets recorded here.

Presumed to be unused
$407-$4FD: ????

Inverse Checksum: $4FE-F. If you add numbers to the file, you need to subtract from this location. See Side Note 3 for more information.

==========

Appendix:

Side Note 2: Items other than the standard equipment can be equipped. For instance, it's possible to equip the compass as a sword. By some miracle, the items you end up with often work the same way as their appropriate counterparts. Sometimes they are far superior to the normal items, and sometimes they just suck. At least one item makes your armor invincible! Just experiment a little. Beware, the palletes will not be standard.

Side Note 3: A tutorial on inverse checksums. Let's say I add to location $3EE. Now E = 14 in the decimal system. (Note we are looking at the last digit. "E" that is.) Therefore $3EE is even.

If I add a value to a memory location with an even address,
I must subtract from the even address of the inverse checksum. Example: Suppose I add $4 to $305. $305 is odd, so I SUBTRACT $4 from the odd checksum byte: $4FF. If I subtracted from $305, I must add to $4FF. This maintains the "balance" of the file and keeps it from being erased.

Now this will work for slight changes in the checksum, but it takes a bit of insight to recognize that the checksum is really a 16-bit number, not just two 8-bit numbers functioning separately.

(edit later)

Character Codes:

Alpha-numeric

00-A 01-B 02-C 03-D 04-E 05-F 06=G 07-H 08-I^ 09-J 0A-K 0B-L 0C-M 0D-N OE-O OF-P
10-??
20-Q 21-R 22-S 23-T 24-U 25-V 26-W 27-X 28-Y 29-Z 2A-a 2B-b-2C-c 2D-d 2E-e 2F-f
40-g 41-h 42-k 43-j 44-i 45-l 46-m 47-n 48-o 49-p 4A-q 4B-r 4C-s 4D-t 4E-u 4F-v

60-w 61-x 62-y 63-z 64-0 65-1 66-2 67-3 68-4 69-5 6A-6 6B-7 6C-8 6D-9 6E-"?" 6F-"!"

80-"-" 81-"." 82-"," 85-"(" 86-")"

Special characters (not normally accessible. This is by far an incomplete listing)

A0-small right arrow A1-"'" (apostrophe) A2-HPiece again A3-"empty right hand heart cont."
A4-see A7 A5-Same as A7 A6-"Quarter Heart piece, top right corner."
A7-"Heart piece,left half" A8-"Heart piece, right half" A9-blank AA-"left arrow"
SNES Button Alphabet: AB-A AC-B AD-X AE-Y
AF-I
B1-blank^

^This code is not the canon encoding of this character. ex. AF is the proper "I". 08 is not.

5) VRAM Documentation

Zelda 3 VRAM Documentation by MathOnNapkins

V0.2

Last Updated 12/17/2006

****

****

Module 7 (Dungeon) and Module 9 (Overworld)

$0000-$1FFF BG2 Tilemap

$2000-$3FFF BG1 Tilemap

$4000-$7FFF 4bpp Tiles for BG0 and BG1 [They share :)]

$8000-$BFFF Sprite Graphics Memory

> Sprite graphics can be seen as divided in half. Areas that have 0's as sprite indexes will not overwrite the graphics from the previous screen. This can cause inconsistent behavior though, so be careful.
>
> ---
> $8000-$87FF Reserved Sprite Memory (Slot 0)
>
> This region contains Link's body's current sprites, as well as various other essential sprites that can't easily be taken out of memory. e.g. rupees, the sword and shield, etc.

$8800-$9FFF Locale Specific Graphics (Slots 1 - 3)

"Locale" in this sense can be one of three things: Dark World, Light World, or inside a Dungeon. This will determine what the pots and bushes you pick up look like, (rocks as well)

$A000-$BFFF Room / Area Specific Graphics (Slots 4 - 7)

Every overworld area and dungeon room has a graphics number that tells the game which sprite graphics to load dynamically.

$C000-$DFFF BG3 Tilemap

$E000-$FFFF 2bpp Tiles for BG2 [This is the Heads up Display and Menu, etc]

$F800-$FFFF - used for text message data

- note that graphics packs 6A, 69, and 68 are used with srctypeone for this

****

6) Dungeon objects database

by MathOnNapkins

star tile class objects

1.2.1F active star tile

- **$0432** as tracking variable
- **$06A0**, **X** as tile address

staircase class objects

1.2.2D in-floor up north staircase

- **$0438** as tracking variable
- **$06B0**, **X** as tile address

1.3.1E

- **$04A2** as tracking variable
- **$06B0**, **X** as tile address

1.3.20

- $04A2 as tracking variable
- $06B0, X as tile address

1.3.21

- $04A8 as tracking variable (shared with 1.3.29)
- $06B0, X as tile address

1.3.26 straight up north staircase

- $04A2 as tracking variable
- $06B0, X as tile address

1.3.27 straight down north staircase

-
- $06B0, X as tile address

1.3.28 straight up south staircase

- $04A2 as tracking variable
- $06B0, X as tile address

1.3.29 straight down south staircase

- $04A8 as tracking variable (shared with **1.3.21**)
- $06B0, X as tile address

1.2.38

- $047E as tracking variable
- $06B0, X as tile address

1.2.39

- $0480 as tracking variable
- $06B0, X as tile address

1.2.3A

- $0482 as tracking variable
- $06B0, X as tile address

1.2.3B

- $0484 as tracking variable
- $06B0, X as tile address

theory: There is an order in which staircase objects must be stored in order to work 100% correctly.

reasoning: depending on which staircase object is invoked, the starting position for a variable number of additional staircase type objects gets updated. See diagram:

$0438 - updated by 1 routine

$047E - updated by 2 routines

$0482 - updated by 3 routines
$04A2 - updated by 5 routines
$04A4 - updated by 7 routines

$043A - updated by 9 routines

$0480 - updated by 10 routines
$0484 - updated by 11 routines
$04A6 - updated by 13 routines
$04A8 - updated by 15 routines

As we can see, $04A8 would correspond to objects that must come last in the stream, and $0438 corresponds to objects that must come first in the stream

// ==

ladder class objects

1.2.30 ????? staircase (unused in original game)

- $043C as tracking variable
- $06B8, X as tile address

1.2.31 inter-bg north staircase

- $043E as tracking variable
- $06B8, X as tile address

1.2.32 inter-pseudo-bg staircase

- $0440 as tracking variable
- $06B8, X as tile address

1.2.33 inter-bg north staircase (water room)

- $0442 as tracking variable
- $06B8, X as tile address

1.2.35 water ladder

- $0444 as tracking variable
- $06B8, X as tile address

1.2.36 inactive water ladder

- $0446 as tracking variable
- $06B8, X as tile address

1.3.1B in-room up-south staircase

- $049A as tracking variable
- $06B8, X as tile address

1.3.1C in-room up-north staircase

- $049C as tracking variable
- $06EC, X as tile address

1.3.1D in-room up-south staircase

- $049E as tracking variable
- $06EC, X as tile address

in-room staircases and ladders class objects

$043C - updated by 1 routine
$043E - updated by 1 routine
$0440 - updated by 1 routine
$0442 - updated by 1 routine
$0444 - updated by 2 routines
$0446 - updated by 3 routines
$049A - updated by 1 routine

// ==========

2.vertical.0 standard door?

- door is completely on BG2

2.any.1 inset version of 2.any.0

- Sets priority bits of tilemap entries in a 7 x 4 region leading from the bottom of the door to the edge of the screen.

- Since this door type insets tile priority, doesn't that mean it can only be used in the positions farther from the edge of the room? (Otherwise causing memory writing problems...?

- door is completely on BG1

2.down.2 cave exit door (inset)

- why is this different from 2.down.8?

2.any.3 unused

- perhaps this is a debug property / feature. should be investigated
- it gets a fair amount of attention in the code for each door direction

2.up.4 waterfall door

- probably hides itself via priority bits in tilemap?

2.down.5 large palace exit door (BG2)

- door is completely on BG2
- only manifests as anything special in down direction

2.down.6 large palace exit door (BG1)

- door is completely on BG1
- only manifests as anything special in down direction

2.down.7 cave exit door

- no comment

2.down.8 cave exit door (inset)

- no comment

2.vertical.9 exit door

- Triggers a transition to the overworld
  How that transition occurs depends on whether the room number is <= 255. If greater than that, Link simply comes out at the location he came in, regardless of what door he exits from.

2.any.10 palace toggle property

- Use it to modify other doors to give them this property

2.any.11 floor toggle door

- Use it to modify other doors to give them this property

2.any.12 double sided trap door

- apparently must be triggered by something to open

2.up.13 invisible door?

- only used in Turtle Rock
- door is completely on BG1

2.any.14 locked door that opens into normal door

- versatile, locked on both sides (naturally)

2.up.15 big key door that opens into normal door

2.up.16 locked door that obscures staircase type
2.up.17 locked door that obscures staircase type

- door is completely on BG1

2.any.18 unused

- might do something, but not seen in original game

2.up.19 locked door that produces staircase?

- seems like it doesn't draw the top of the door (to help interface with staircases.)
- only used once in the game.
- door is completely on BG2

2.up.20 bomable vermin door
2.left.20 bombable vermin door

- opens when bombed but makes small vermin enemies come out (somehow)

2.down.21 bombable exit door

- notable in that it turns into exit door when bombed open

2.any.22 unused

- might do something, but not seen in original game

2.up.23 bombable wall

- seems to work in a variety of positions

2.up.24 blow-up-able wall

- rarely used
- door starts off as a wall but a trigger blasts it open

2.up.25 hidden door that opens with sword slash

- rarely used
- door is completely on BG1

2.any.26 unused

- might do something, but not seen in original game

2.any.27 BG1 only right side trap door
2.any.28 BG1 only left side trap door

- can have two-part form
- door is completely on BG1

2.any.29 unused
2.any.30 unused
2.any.31 unused

- might do something, but not seen in original game

2.any.32 top on BG1 door

- can have two-part form
- top of door is on BG1, rest is on BG2

2.any.33 unused

- might do something, but not seen in actual game

2.left.34
2.up.34 trap door triggered by event?

- can have two-part form
- seen in positions 9, 10, and 11
- opened by sprite logic perhaps?

2.horizontal.35 arbitrary room link door?

- designed to specially link to other rooms via dungeon header? (guess)
- seen in positions 3 and 9
- top of door is on BG2, rest is on BG1

2.left.36 one side trap door

- two-part door
- only seen in position 11
- top of right-facing door is on BG2, rest is on BG2

2.left.37 one sided trap door

- two-part door
- trap door on left, normal (or somewhat normal on right?)
- only seen in position 11
- top of right-facing door is on BG2, rest is on BG1

- Trap on left, normal on right

7) Decompression/Compression codes

Decompression Code

Commented by Peekin - Feburary 22nd, 2001

```
; 65816 SNES Disassembler   v2.0a (C)opyright 1994 by John Corey
; Begin: $00e7a3  End: $00e852
; Hirom: No   Quiet: No   Comments: 2  DCB: No   Symbols: No   65816: No
00e7a3 20 43 e8      JSR $e843      ; get next byte
```

```
00e7a6 c9 ff         CMP #$ff       ; end of compressed stream if code = 0FFh
00e7a8 d0 03         BNE $e7ad      ; continue if any other code
00e7aa e2 10         SEP #$10       ; Index (8 bit)
00e7ac 60            RTS            ; end of compression routine
; get code and length (code is upper 3 bits, length is lower 5)
00e7ad 85 cd         STA $cd        ; save byte
00e7af 29 e0         AND #$e0       ; get code (upper 3 bits)
00e7b1 c9 e0         CMP #$e0       ; special code for longer run count
00e7b3 f0 0a         BEQ $e7bf
00e7b5 48            PHA            ; save code
00e7b6 a5 cd         LDA $cd        ; retrieve byte
00e7b8 c2 20         REP #$20       ; Accum (16 bit)
00e7ba 29 1f 00      AND #$001f     ; mask byte to get length (lower 5 bits)
00e7bd 80 12         BRA $e7d1
; long run count
00e7bf a5 cd         LDA $cd        ; retrieve byte
00e7c1 0a            ASL            ; shift byte left 3 times for new code
00e7c2 0a            ASL
00e7c3 0a            ASL
00e7c4 29 e0         AND #$e0       ; get new code (upper 3 bits)
00e7c6 48            PHA            ; save code
00e7c7 a5 cd         LDA $cd        ; retrieve original byte
00e7c9 29 03         AND #$03       ; select lowest two bits for count
00e7cb eb            XBA            ; save count's two msb in AH (*256)
00e7cc 20 43 e8      JSR $e843      ; read next byte for a total 10bit count
00e7cf c2 20         REP #$20         ; Accum (16 bit)
; depending on if the code was E0, the count will either be 0-63 or 0-1023
00e7d1 1a            INC            ; count++
00e7d2 85 cb         STA $cb        ; store count
00e7d4 e2 20         SEP #$20       ; Accum (8 bit)
00e7d6 68            PLA            ; retrieve code
00e7d7 f0 16         BEQ $e7ef      ; transfer bytes from source
00e7d9 30 4a         BMI $e825      ; transfer bytes from output buffer
00e7db 0a            ASL
00e7dc 10 20         BPL $e7fe      ; repeat single byte
00e7de 0a            ASL
00e7df 10 2a         BPL $e80b      ; repeat two alternating bytes
; repeat single incrementing byte??
00e7e1 20 43 e8      JSR $e843      ; read single byte
00e7e4 a6 cb         LDX $cb        ; load count
00e7e6 97 00         STA [$00],Y    ; write byte
00e7e8 1a            INC            ; increment byte value??
00e7e9 c8            INY            ; destination ptr++
00e7ea ca            DEX            ; count--
00e7eb d0 f9         BNE $e7e6      ; loop while count <> 0
00e7ed 80 b4         BRA $e7a3      ; go to top of loop for next code
; transfer bytes directly
00e7ef 20 43 e8      JSR $e843      ; read next byte to transfer
00e7f2 97 00         STA [$00],Y    ; write byte
00e7f4 c8            INY            ; destination ptr++
00e7f5 a6 cb         LDX $cb        ; reload count (since ReadByte changed it)
00e7f7 ca            DEX            ; count--
00e7f8 86 cb         STX $cb        ; store count
00e7fa d0 f3         BNE $e7ef      ; loop while count <> 0
00e7fc 80 a5         BRA $e7a3      ; go to top of loop for next code
; repeat single byte
00e7fe 20 43 e8      JSR $e843      ; read single byte to repeat
00e801 a6 cb         LDX $cb        ; load count
00e803 97 00         STA [$00],Y    ; write byte
00e805 c8            INY            ; destination ptr++
00e806 ca            DEX            ; count--
00e807 d0 fa         BNE $e803      ; loop while count <> 0
```

```
00e809 80 98         BRA $e7a3      ; go to top of loop for next code
; repeat run of alternating even/odd bytes
00e80b 20 43 e8      JSR $e843      ; read first byte
00e80e eb            XBA            ; save first byte into AH
00e80f 20 43 e8      JSR $e843      ; read second byte
00e812 a6 cb         LDX $cb        ; load count
00e814 eb            XBA            ; swap first byte with second
00e815 97 00         STA [$00],Y    ; write first byte
00e817 c8            INY            ; destination ptr++
00e818 ca            DEX            ; count--
00e819 f0 07         BEQ $e822      ; exit loop if count = 0
00e81b eb            XBA            ; swap first byte with second
00e81c 97 00         STA [$00],Y    ; write second byte
00e81e c8            INY            ; destination ptr++
00e81f ca            DEX            ; count--
00e820 d0 f2         BNE $e814      ; loop while count <> 0
00e822 4c a3 e7      JMP $e7a3      ; go to top of loop for next code
; copy run of bytes already in output buffer to end
00e825 20 43 e8      JSR $e843      ; read low byte ptr
00e828 eb            XBA
00e829 20 43 e8      JSR $e843      ; read high byte ptr
00e82c eb            XBA
00e82d aa            TAX            ; copy buffer source to X
00e82e 5a            PHY            ; save destination ptr
00e82f 9b            TXY            ; move buffer source to Y for indexing
00e830 b7 00         LDA [$00],Y    ; read existing buffer byte
00e832 bb            TYX            ; copy back to X, why??
00e833 7a            PLY            ; retrieve destination ptr
00e834 97 00         STA [$00],Y    ; write byte
00e836 c8            INY            ; destination ptr++
00e837 e8            INX            ; buffer source++
00e838 c2 20         REP #$20       ; Accum (16 bit)
00e83a c6 cb         DEC $cb        ; count--
00e83c e2 20         SEP #$20       ; Accum (8 bit)
00e83e d0 ee         BNE $e82e      ; loop while count <> 0
00e840 4c a3 e7      JMP $e7a3      ; go to top of loop for next code
; read next byte
00e843 a7 c8         LDA [$c8]      ; read single byte from ROM
00e845 a6 c8         LDX $c8        ; load source ptr
00e847 e8            INX            ; source ptr++
00e848 d0 05         BNE $e84f      ; if not beyond end of bank
00e84a a2 00 80      LDX #$8000     ; wrap source to beginning of next bank
00e84d e6 ca         INC $ca        ; increment to next source bank
00e84f 86 c8         STX $c8        ; store source ptr
00e851 60            RTS            ; end of read byte
00e852 ff ff ff ff   SBC $ffffff,X  ; those familiar separating FF's
```

Compression Codes

Code	Address	Description
`000.....`	`00e7ef`	transfer bytes from source transfer Count bytes after code to buffer
`001.....`	`00e7fe`	repeat single byte read next byte and repeat Count times
`010.....`	`00e80b`	repeat two alternating bytes read next two bytes and alternately repeat
`011.....`	`00e7e1`	repeat single incrementing byte

Code	Address	Description
		read next byte and repeat incrementing each time long count
111.....	00e7bf	bits 2-4 become 5-7 for the new code. bottom 2 bits become top two bits of count. read next byte for lower 8 bits of count +1.
1xx.....	00e825	transfer bytes from output buffer read next two bytes for buffer source pointer (low/high)
11111111	00e7aa	end of compressed stream

Examples

Transfer bytes from source

```
03 12 34 56          -> 12 34 56
```

Repeat single byte

```
23 12                -> 12 12 12
```

Repeat two alternating bytes

```
45 12 34             -> 12 34 12 34 12
```

Repeat single incrementing byte

```
63 12                -> 12 13 14
```

Long count (followed by source transfer of 302h bytes)

```
E3 01 12 34 56 ..    -> 12 34 56 78 90 12 34 ...
```

Transfer byte from output buffer (starting at offset 4)

```
83 00 04             -> 90 12 34
```

Revised on January 10, 2011 08:30:27 by Matthew Callis

8) Tile locations

<ZST>	2000h	64x64:word 16x16	Overworld Field (One large piece)
		64x64:word 8x8	Underworld Tilemap (One large piece)
	4000h	64x64:word 8x8	Second story of (If applicable)
		128x?:word	Overworld Pre-Graphics Field?
	12000h	64x64:byte 8x8	Underworld Field
	13000h	64x64:byte 8x8	Second story of underworld (If applicable)
<ROM>	54927h	byte 8x8	Overworld Map (When X is pressed)

Underworld simply refers to anything not above ground, caves, castles, dungeons, and the insides of homes.

Modifying the field at 2000h while in the Overworld changes both the tile looks and physical properties. Modifying any tiles at 2000h while in a dungeon only changes the look; the attributes remain the same. Change them at 12000h to change its properties, perhaps to remove a wall or change where a door takes you.
<ROM> 492032 150000h Base of Overworld tile table

Note that only the Overworld uses a tile table (since it is so large with varying landscapes), not the dungeons or caves. They instead store the exact units as they would appear in VRAM. Also, they use an 8x8 playfield rather than one 16x16.

9) Zelda 3 Source Code

Exists in two versions! One commented by MathOnNapkins and another one by Wiiqwertyuiop! Although both files are rather large in size so it would be probably too much of a burden to include these in this very own faq so instead I'll link to them:

MathOnNapkins Zelda3 source code: **link zfsdsdfsd latest version October 2012**

wiiqwertyuiop zelda 3 disassembly : **link fdssdfsfs latest version in 2012**

10) Debugging Features

Full Inventory/Walk Through Walls
Use Pro Action Replay code **0083F8EA**, then create a file with a name beginning with **B** in the **first save slot** to enable the following debugging features:

- You start with 15 hearts, 255 rupees, 50 bombs, and 50 arrows in the Light World.
- You have a complete inventory. Your four Bottles are filled with a Fairy, and one each of the Red, Green, and Blue Potions. Additionally, you have the Pegasus Boots, Flippers, Titan's Mitt, Moon Pearl, Fighter's Sword, and Blue Shield.
- Press **B** on Controller 2 to enable free-movement mode, which will allow Link to freely pass through obstacles. It will also allow you to use the Magic Mirror anywhere on the overworld, even to warp from the Light World to the Dark World (which is not normally possible).
- Press **R** on Controller 2 to remove the selected item or weapon while the status sub-screen is on.

Full Restore/Equipment Upgrade
Use the PAR code **0683C1EA**, then create a file named **BAGE** (JP: ンルルル) in the **first save slot** to enable the following debugging features:

- Press **R** on Controller 1 to permanently cut magic consumption in half.
- Hold **R** and press **B** on Controller 1 to enable free-movement mode, which will allow Link to freely pass through obstacles. It will also allow you to use the Magic Mirror anywhere on the overworld, even to warp from the Light World to the Dark World.
- Hold **R** and press **A** on Controller 1 to upgrade your sword, armor, and shield by one step. Once the sword is upgraded to Level 4, pressing the **A** button again will reset all three items to Level 1.

- Hold **R** and press **Y** on Controller 1 to max out your life, magic, arrows, bombs, and keys, as well as add 255 Rupees.
- Hold **R** and press **Select** on Controller 1 to access to the Triforce after the fight against Ganon. Works for three save slots and with any name.

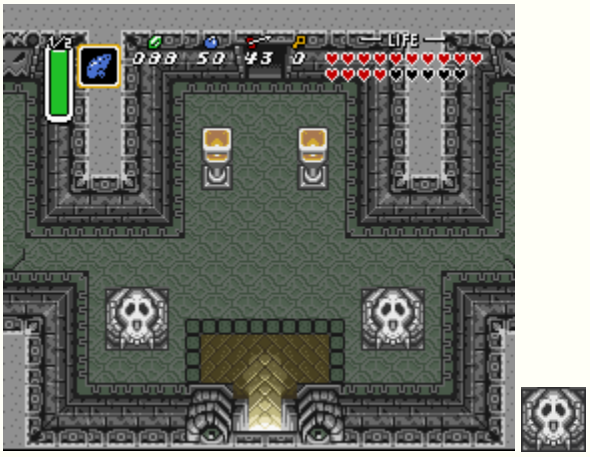
Frame Advance

PAR code **00803A00** enables frame advance. Press **L** on Controller 1 to freeze the game; while frozen, press **R** to advance one frame. Press **L** again to resume normal play.

11) Unused Graphics

a) Dungeon Features

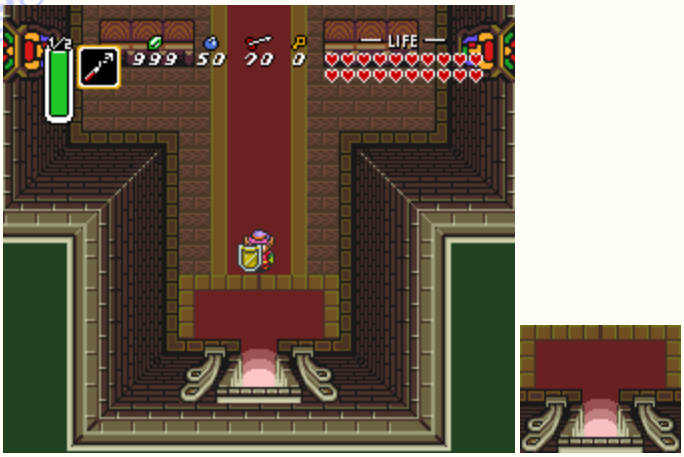
Skull Statue



The Skull Woods dungeon tileset contains an unused large skull statue object. This can be seen in a tile viewer, but it's also possible to cause the game to load them into other dungeons via glitching. This tile goes unused in the SNES port, but it *is* used in the GBA remake for the Palace of the Four Sword (only accessible by beating *Four Swords*).

The screenshot shows the skull tile loaded into the Dark Palace dungeon, replacing the Rocklops statues in the first room. An explanation of how to perform this glitch can be found here.

Sanctuary Entrance



Every dungeon entrance has a specially decorated entrance doorway, the only exception being the Sanctuary. There *is* an entrance doorway for the Sanctuary, but it can't be seen by normal means. It can be seen by either moving through certain walls, or via Hyrule Magic's map editor.

Keyhole

Unused graphics for a keyhole object. Could have been used in any number of places.

b) Items

Meat

Stored with the graphics for the large and small magic refill decanters are sprites for a large and small piece of meat on a bone, looking extremely similar to the Bait from *The Legend of Zelda*. Whether it would have worked in the same manner is unclear. It takes up the space that graphics for the fish appear normally, but is loaded in indoor areas, so it may have simply been some kind of object for use in houses. Perhaps it was an alternate health-restoring item found indoors, much like the Apples that appear from trees outside.
This may have been originally planned for enemies to drop large and small meat chunks for health refills, with the larger one restoring more life hearts than the smaller one. In the final game, enemies drop hearts instead of meat, and they only ever refill one life heart at a time.

Magical Clock

An unused stopwatch object! This would presumably work like the Magical Clock item in *Zelda 1*, freezing any onscreen enemies. It appears alongside graphics such as Rupees, so it was probably intended to be dropped by enemies in the same manner as the original game. It was presumably removed because the Quake Medallion performs a similar effect.

Sword Text

剣

Japanese text for "ken/tsurugi", which means "sword", and is fittingly found near the sword graphics in memory.

Letter

The Letter (called てがみ (tegami) in the Japanese version, but untranslated and blank in the English version) is another item making a return from *Zelda 1*. As in that game, it uses the same sprite as the Map. It can actually be added to the inventory in all versions of the game; it occupies the spot of the Magic Mirror, suggesting that you needed the Letter to acquire the Mirror at some point in development, in the same way you need the Shovel to acquire the Flute, which then takes the Shovel's spot in the inventory. It's unknown exactly why the letter was canned.

To add it to your inventory, use Action Replay code **7EF35301**. In both versions, the item acts exactly like the Magic Mirror when used.

c) Unused graphical effects

Kholdstare's Shell "Melting"

A close inspection of Kholdstare's logic indicates that its shell was intended to gradually fade out after being defeated with fire-based weapons. Normally, it just abruptly disappears - several frames after being destroyed. The sudden disappearance of the shell is not aesthetically pleasing, so it's not surprising that this transitioning effect was implemented. Unfortunately, a coding bug accidentally disabled the effect. A fairly simple patch has been created to restore this effect, a link to which can be found above.

As for an explanation of how it was accidentally disabled, it turns out that the palette that was supposed to be manipulated to achieve the fadeout was not correctly invoked. Instead, a neighboring palette was selected for fading. Regrettably, there were no graphics visible in that scene that also used the palette being faded. Thus, this bug understandably slipped through play-testing due to a lack of obvious side effects, visual or otherwise. This bug is known to be present in all SNES releases of the game. Interestingly enough, the shell's fade effect was restored when the game was ported to the Game Boy Advance.

link to a valid link!

Download Kholdstare Melting Shell Restoration Patch (U)
File: Kholdstare_shell.ips (28 KB)
Current version: 1.0

d) NPC sprites

Duck Sprites

The duck has additional frames of animation that are never used - one of his wings up really high and one of him leaning back slightly, which looks like it could have been used for picking Link up and dropping him off. However, the duck never assumes either of these poses at any time.

Sad Blob

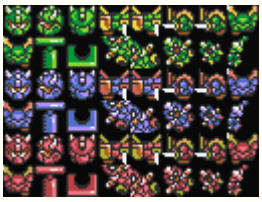
While this pink blob does appear in the game, he never changes from his standard, happy expression. These graphics can only be found in the Japanese version.

Ending Character

This unused NPC appears alongside the graphics for "The End", which would imply that he was originally meant to appear in the ending sequence somewhere. It's possible he was a Kakariko villager, or potentially a Dark World resident who had returned to normal. He appears to be jumping, which would make sense if he was celebrating in the ending.

e) Enemy sprites

Unused Soldier Helmet



An alternate helmet for a soldier enemy is present alongside the body graphics for the short blade-wielding regular Soldier, those met early in the game. The other helmets used in-game appear alongside their respective body graphics, so this implies that the weak, less intelligent soldiers originally used this graphic for their helmets.

Why it was removed is unknown, but possibly because it looks more threatening than it should for such weak enemies, or because it is the only helmet that makes it clear there is a human body still within the armor and Nintendo didn't like the idea of Link hurting brainwashed humans (while the game implies that the soldiers were merely brainwashed, other

material such as the comic and manga have suggested that the enemy soldiers are simply living armor, which might explain why they seem to outnumber the rest of Hyrule's population ten-to-one).

Dark World Bat

An odd, unused Dark World enemy. Some kind of bat thing. Its programming doesn't seem to exist in the game anymore, and its graphics can only be found in the Japanese versions of the game.

Apparently, it would have shot fireballs at Link.

f) Miscellaneous

Faces

A "mean" face and a "happy" face that can be found stored with the menu graphics.

Bomb Shop Sign Women

Link and the sprite with index **0x3d** have had a strained relationship over the years. It normally manifests as a white-haired lady pacing back and forth in front of her house in Kakkariko Village. If Link gets too close, she'll call out for soldiers to come arrest him and bolt inside her home, locking the door.

If this same sprite is spawned indoors, however, she's a different woman altogether. In fact, she's downright *weird*. This sprite will turn to face the player like many other NPCs in the game, but the only thing she seems to want to talk about is that the Bomb Shop is somewhere West of Link's current position. This mildly suggests that the Bomb Shop was originally in the Light World; or perhaps that there were multiple Bomb Shops.

One could speculate that this sprite's indoor logic was used in the production phase to test out various text messages, or that perhaps the Bomb Shop had multiple rooms. Maybe it was intended at some point to have an indoor sprite direct Link towards the Bomb Shop within a network of rooms.

The red-haired woman (sprite index **0x34**) in the village, that also spends her time looking for people to narc on, behaves identically to the older woman when placed indoors. As the white-haired woman retains her appearance indoors, so too does the red-haired woman. This is due to the fact that that these two sprites share the same logic, except for the subroutines called to render them to the screen.

I can't believe how different you are when we're alone, unless the chicken turned into a lady counts as a person.

Maybe you, me, and that older lady could get together and have a classy time discussing the Bomb Shop over drinks?

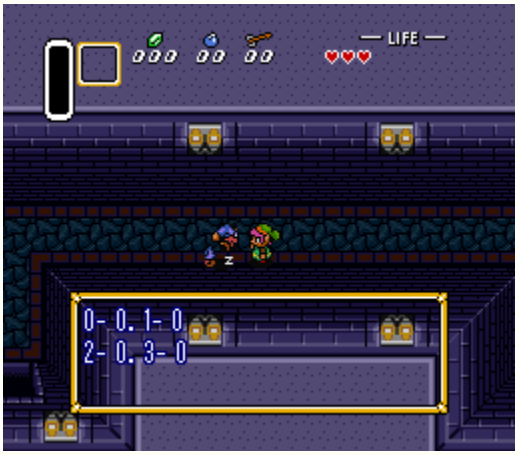
Dialogue Tester Sprite

The sprite entity with index **0xb8** appears to be a leftover debug feature for testing the game's dialogue messages. While it was previously thought that the sprite had some sort of menu for selecting the next message to be displayed, this is a misconception due to the fact that the templates for some of the game menus are found at some of the lowest message indices.

Without other logic driving the sequencing of those menu templates, they have no effect.
The sprite initializes its message index to **0x0000** and will increment this index after each message is displayed, which also causes its physical orientation to cycle to another cardinal direction. Each message is triggered automatically when the player gets close enough to the sprite, so the A button is not used to interact in this situation. This effectively prevents the player from passing through the sprite.

There is no bounds checking performed on the message index. Therefore, if the player reads until all of the valid message indices are exhausted, the game will crash, as the next message index will reference data that is not valid for the dialogue system.

When spawned in most indoor rooms, the sprite doesn't look properly constructed. It was later discovered that the Priest ['Sage' in the North American release] sprite and the dialogue tester call the same subroutine to be drawn to the screen. The color of the sprite's garb and skin tone is different from that of the priest, however.



The dialogue tester as it appears in a room without the

proper graphics configuration.

The dialogue tester shown in a room with a compatible graphics configuration.

12) Unused Enemies

Cannon Trooper

This Hyrulean soldier with a portable cannon is never actually used in the game. He's fully coded and functional, and would fit quite well in areas like Agahnim's Tower.

13) Chris Houlihan room

The Chris Houlihan room is used as an error handler if you fall into a hole and the game cannot find a proper destination. It is a single cave room and contains a telepathic tile, as well as 225 Rupees. If you exit the cave, you will be warped to the front of Link's House, regardless of which world you were in before entering the room.

There is a bug in the game which can lead you to this room, related to screen transitions using the Pegasus Boots: go to the area to the left of Hyrule Castle and go up. After the transition has ended, drop a bomb in front of you and wait until it explodes. It will hurt you, and push you against the bottom of the screen. Now, charge up the Pegasus Boots, and turn to the bottom while charging up, so you will immediately move down to the next screen. Now, go to the hole at Hyrule Castle which leads to the secret passage and fall into it, and you will appear in the Chris Houlihan room.

Chris Houlihan is a kid who participated at a *Nintendo Power* event and received the honor to appear in a future Zelda game. Note that the GBA remake fixed this glitch, so there's no known way to get into this room, and even if you could there's no longer any mention of Chris.

Japanese:「ここは、秘密の部屋だよ~ん。みんなにはないしょだよ~ん。」(Koko wa, himitsu no heya da yoon. Minna ni hanaisho da yoon. (This is a secret room~ It's a secret to everyone~))
English: My name is Chris Houlihan. This is my top secret room. Keep it between us, OK?
French : C'est ma pièce la plus secrète. Que cela reste entre nous, ok ?
German : Dies ist mein ganz geheimes Zimmer. Das bleibt aber unter uns, ja?

GBA Version

In the GBA version, the room is slightly different. The telepathic tile and the blue floor no longer exist. Only the rupees remained in place. If it is not possible to access in the room, it still has the ID **82**.

The only possible method to access in this room is to use this code when entering a door or falling into a hole:

Version	Code
All Versions	03002C4C 82

14) Regional Differences

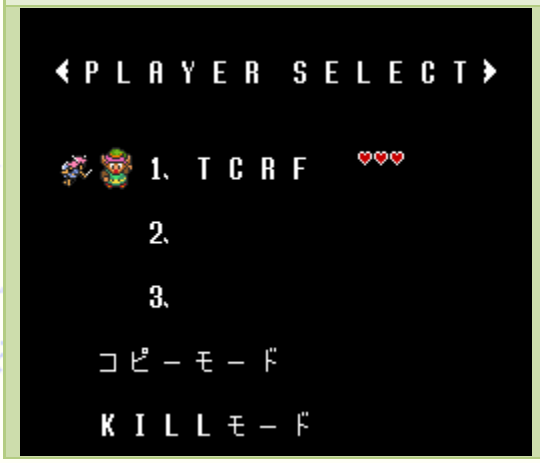
Title Screen

Japanese	American

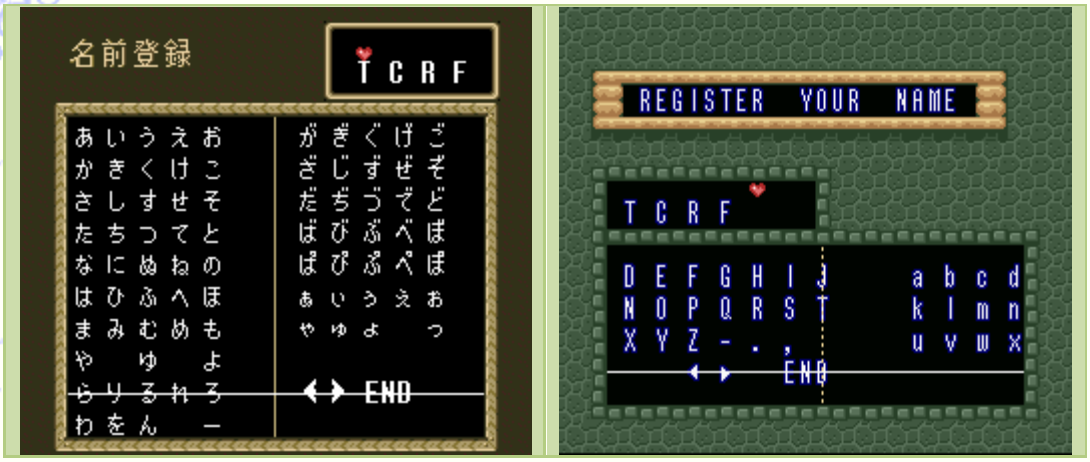
The Japanese title screen has neither the sword nor the castle background seen in the American title screen. In the Japanese version, you can go straight to the File Select by pressing **Start**. In the American version, this option is only offered to you after using "Save & Quit" - if you start the game normally, you have to wait until the title screen actually displays the title before you can go to the File Select.

File Select

Japanese	American
	

The Japanese File Select has a stark black-and-white contrast not unlike that of *Zelda 1*. The American version uses some nice graphics there.

Japanese	American



File names in the Japanese versions can only be four characters long. In the American version, this limit has been generously raised to six.

Hylian Script

Japanese	American
	

In the Japanese version, the hieroglyphic font is much larger and more detailed than in America. All the text boxes were also made a little wider for the US version.

Eastern Palace

Japanese	American
	

This tile in the Eastern Palace was changed to remove a religious reference.

Wanted Signs

Japanese	American
おたずね者"File Name" ゼルダ姫を城よりさらった犯人 みつけたら大声でしらせよ！	WANTED! This is the criminal who kidnapped Zelda. Call a soldier if you see him!

In the Japanese version, the Wanted Signs of Link in Kakariko Village displays the File Name you have chosen next to the portrait.

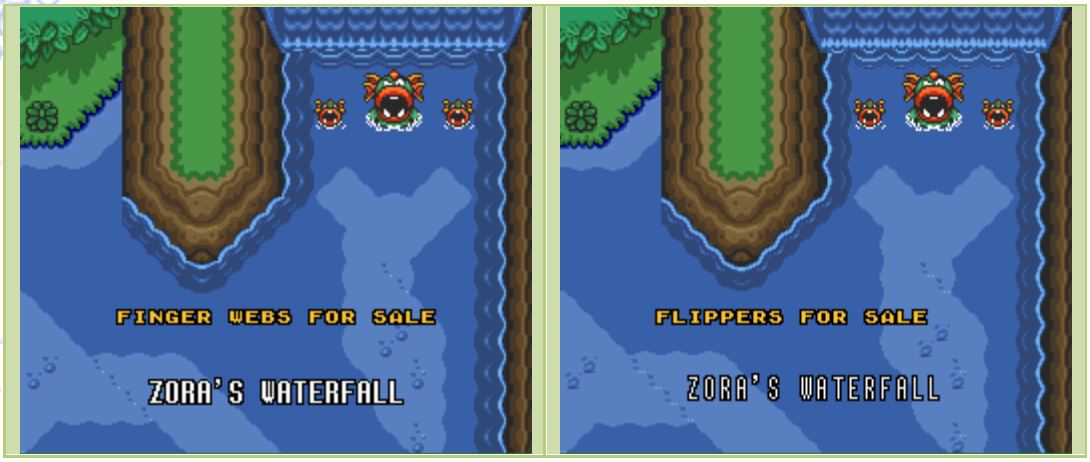
Ending

Some lines of the ending sequence were altered between releases:

Japanese	American
	

This was changed to remove another religious reference.

Japanese	American



A typical case of Engrish. Fixed up in the American version. However, they forgot to re-center the text when doing so, (this can be seen in the other text revisions as well but not as clear as in this example).

Japanese	American
	

Changed since the Ocarina is called Flute in the American version (why? we'll never know).

Japanese	American
GANNON'S TOWER	GANON'S TOWER

Credits

An extra script credit for the localization effort was added to the English version.

Japanese	American
	

More Engrish fix-up in the Quest History.

Also, all text in the game was for some reason drawn a few pixels higher in frame in the English version, so the Quest History stats came out looking a little bit akward as a result of this, as the regular text isn't correctly aligned with the smaller secondary text that is otherwise only present in the ending of the game.

15) Revisional Differences

Glitches in Japanese v1.0

- After getting the Pegasus Boots, by pressing **Y + A** at once you can dash while holding out an item. The item works like that, too: by using the Shovel, for example, you can dig an entire row of holes. [1]
- In the Dark World, there's a ledge on Death Mountain that connects two parts of Turtle Rock. By using the Magic Mirror on the above ledge, you could warp on top of the wall, then jump down to the Turtle Rock ledge and skip a large portion of the dungeon. In later versions, you can still warp on top of the wall, but you won't be able to jump down. [2]

- In any area with movable blocks where you can't use the Magic Mirror (like the watergate room in the Light World), push the movable block and try to use the Magic Mirror at the same time, and the block will disappear completely. [3]

Glitches in Japanese v1.1

- In the Tower of Hera, on floor 3F, there's a hole at the very right that's next to the wall. Drop down the right side of it to end up on 2F, inside the wall. Jump off to the right and you're "under the floor", from where you can just run straight to the ending! The hole was moved to the left in later versions, to fix this bug. [4]
- In the Dark World, killing yourself inside a shop or house will cause Link to reappear on the Pyramid of Power with no music. Switching the screens afterwards causes the Light World music to play, rather than the Dark World music. [5]

(Source: darkeye14)

Glitch in Japanese v1.0 and v1.1

- If you make a regular slash with your sword against bombable walls, you'll hear the same sound effect as when you cut bushes, this of course didn't make much sense and was corrected in v1.2. (Not to be confused with the "hollow" sound effect you get when you "thrust" your sword against those same walls, which was already in place.)

Game Boy Advance Port

The Game Boy Advance version came bundled with Four Swords. Achieving some goals in that game would unlock some extras:

- A sidequest involving the third lumberjack, requiring you to catch some special enemies with a special nest item. A new sword move is unlocked this way.
- A new dungeon with rematches with previous bosses, and a new boss battle. It exploits some of the additions of the GBA version, like the ability to dive underwater, and has new enemies. It uses the eighth unused tileset.

The file select screen has been dummied, and replaced with another one common to the two games in this port. The engine was redone, and as such the game is far more stable: the Chris Houlihan room no longer bears this name and is considerably more difficult to access. The translation was corrected at some points too.

The Ice dungeon has been altered to remove a difficult puzzle involving pushing a block across several floors: this way, two rooms from the SNES version were made inaccessible, yet still coded.

~ Chapter VIII - Hacks database

By the time I started hacking this game, there were no hack released.. Then a few months ago, the Z3 hacking community started to grow much more... As of now, some hackers show off their works, some even released demos; this section will keep a up to date database of the hacks in progress, as well as information, pictures and demo links (If released), the WIP (work in progress) hacks and the discontinued, not updated in a long time hacks...

Legend:

Hack: *Name of the hack here*
Author(s): *People who made the hack*
Information: *The type of hack, basic information*
Screenshots: *Self-Explanatory*
Latest version: *Complete, demo or N/A*
URL: *Where to get it / website URL*

---

1) W.I.P. hacks

Hack: Zelda 3 Challenge ~ Quest for Calatia
Authors: GameMakr24
Information: Completely new dungeons, overworlds, gfx
Release date: ???
Screenshots:

Latest version: N/A
URL: http://www.questforcalatia.net/Zelda3C/index.html

Hack: The Legend of Zelda - Renascence of Evil
Authors: Aeranima
Information: Completely new dungeons, overworlds, gfx
Release date: ???
Screenshots:

Latest version: N/A
URL: N/A

Hack: The Legend of Zelda - Parallel Universes
Authors: SePH, Euclid, PuzzleDude, LinksDarkArrows
Information: Completely new dungeons, overworlds, gfx
Release date: 31 December 20XX
Screenshots:

Latest version: N/A
URL: http://parallelsequel.wordpress.com/

Hack: The Legend of Zelda - Gates of Time

Authors: PuzzleDude, SePH

Information: Completely new dungeons, overworlds, gfx, PuzzleDude is extending the gameplay

Release date: ???

Screenshots:

Latest version: N/A

Hack: The Legend of Zelda - Prophet of Chaos
Authors: Spane, XaserLE
Information: Completely new dungeons, overworlds, gfx
Release date: ???
Screenshots:

Latest version: N/A
URL: ???

2) Iced hack?! ~ *I have a feeling it's still in the works in some secret lab... somewhere... that's just my feeling!*

Hack: The Legend of Zelda - Dark Prophecy
Authors: Omega45889 (*Overworlds, story*), Dude Man (*Dungeons*), MathOnNapkins (*ASM*),
Bit-Blade (*Link Sprite, New graphics ~ created from scratch*), SePH (*titlescreen/overworld graphics insertion*)
Information: Completely new dungeons, overworlds, gfx
Release date: ???
Screenshots:

Latest version: N/A
URL: N/A

3) Hacks with unknown status

Hack: The Legend of Zelda - Dream Scepter
Authors: DiscoPeach
Information: Completely new dungeons, overworlds
Release date: ???
Screenshots:

Latest version: N/A
URL: http://acmlm.kafuka.org/board/thread.php?id=4339

Hack: Lyra Islands Project
Authors: Zeniea
Information: Continuation of an older abandoned hack
Release date: ???
Screenshots: N/A
Latest version: N/A
URL: http://zeniea.com/zelda/LyraIslands/

Hack: Untitled Zelda 3 Hack
Authors: Vampire

Information: Unknown
Release date: ???
Screenshots:

Latest version: N/A

Hack: The Puppeteer (Ruins of Rockvan 2)
Authors: Torin

Information: Unknown
Release date: ???
Screenshots:

Latest version: Demo
URL: http://acmlm.kafuka.org/uploader/get.php?id=3174

4) Complete hacks

Hack: The Legend of Zelda - Parallel Worlds

Authors: Euclid, SePH

Information: Completely new dungeons, overworlds, gfx, with new music

Release date: 31 December 2006

Screenshots:

Latest version: 1.1

URL: http://acmlm.kafuka.org/uploader/get.php?id=4369

Hack: The Legend of Zelda - Goddess of Wisdom
Authors: PuzzleDude, SePH, Omega45889, Dude Man, bregegrahf1
Information: Completely new dungeons, overworlds, gfx, extended version of Shards of Might.
Release date: 16 August 2010
Screenshots:

Latest version: 3.0
Preview: http://www.youtube.com/watch?v=aFqOjl_tKGo
Download: http://acmlm.kafuka.org/uploader/get.php?id=4369

Hack: The Legend of Zelda - PuzzleDude's Quest

Authors: PuzzleDude

Information: Completely new dungeons, overworlds

Release date: 01 September 2011

Screenshots:

Latest version: 1.0

URL: http://acmlm.kafuka.org/uploader/get.php?id=4369

5) Remodelled hacks

Hack: The Legend of Zelda - Parallel Remodel
Authors: PuzzleDude, Euclid, SePH
Information: Improvement, PuzzleDude remade Euclid's dungeons to be easier
Release date: 29 July 2012
Screenshots:

Latest version: 1.1
URL: http://acmlm.kafuka.org/uploader/get.php?id=4369

6) Master Quest hacks

Hack: The Legend of Zelda - Link to the Past Master Quest
Authors: Moulinoski
Information: The dungeons are now harder. Bosses are also harder.
Release date: 18 February 2007
Screenshots:

Latest version: 1.2
URL: http://www.romhacking.net/hacks/170/

Hack: The Legend of Zelda - Link to the Past Master Quest

Authors: Jamesbrad277

Information: All dungeons are rearranged with tougher enemies, just like the Master Quest version of Ocarina of Time.

Release date: 30 September 2009

Screenshots:

Latest version: 0.9

URL: http://www.romhacking.net/hacks/601/

7) Minor hacks

Hack: Minish Cap Link into LTTP
Authors: ghillie
Information: Sprite hack
Release date: N/A
Screenshots:

Latest version: N/A
URL: http://acmlm.kafuka.org/board/thread.php?id=7321

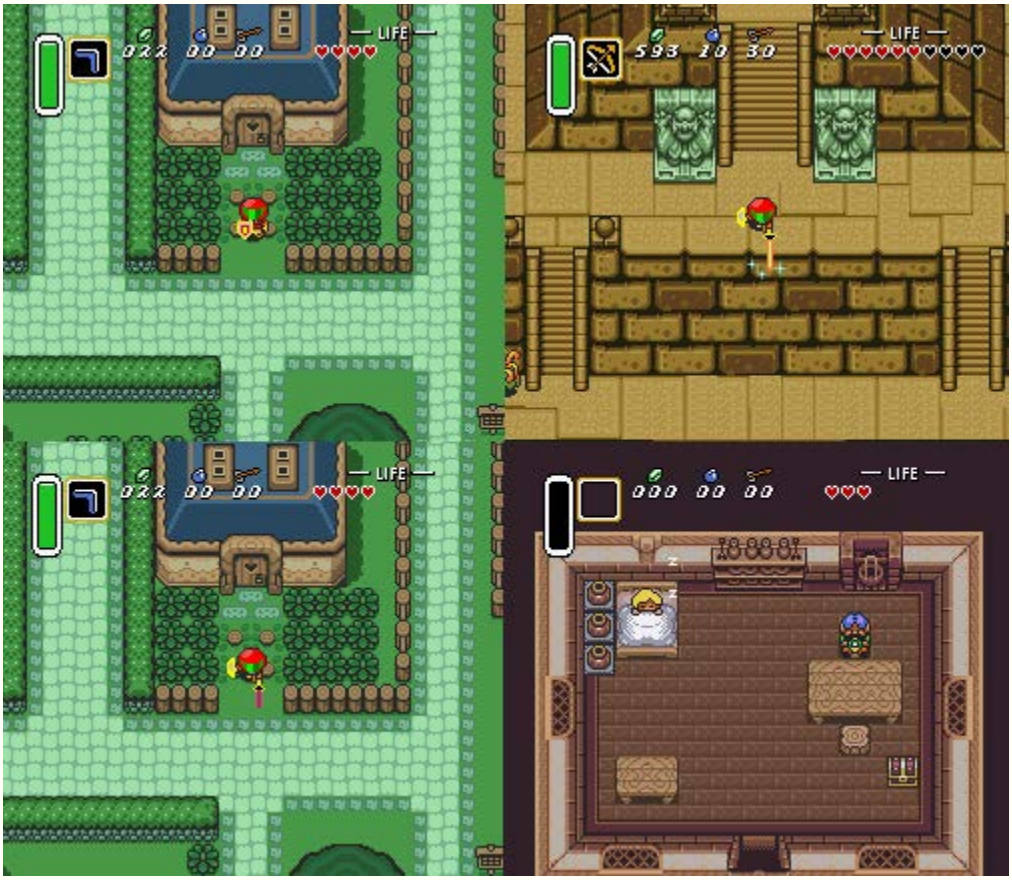
Hack: Play as Princess Zelda In ALTTP
Authors: «•CRIMSON•»
Information: Sprite hack
Release date: January 09 2012
Screenshots:

Latest version: 1.0
URL: http://acmlm.kafuka.org/board/thread.php?id=7321

Hack: Derpy Link Sprites
Authors: Botchos
Information: Sprite hack
Release date: August 11 2012
Screenshots:

Latest version: 1.1
URL: http://www.deviantart.com/download/306595881/derpy_link_sprites_v1_1_by_botchos-d52jew9.zip

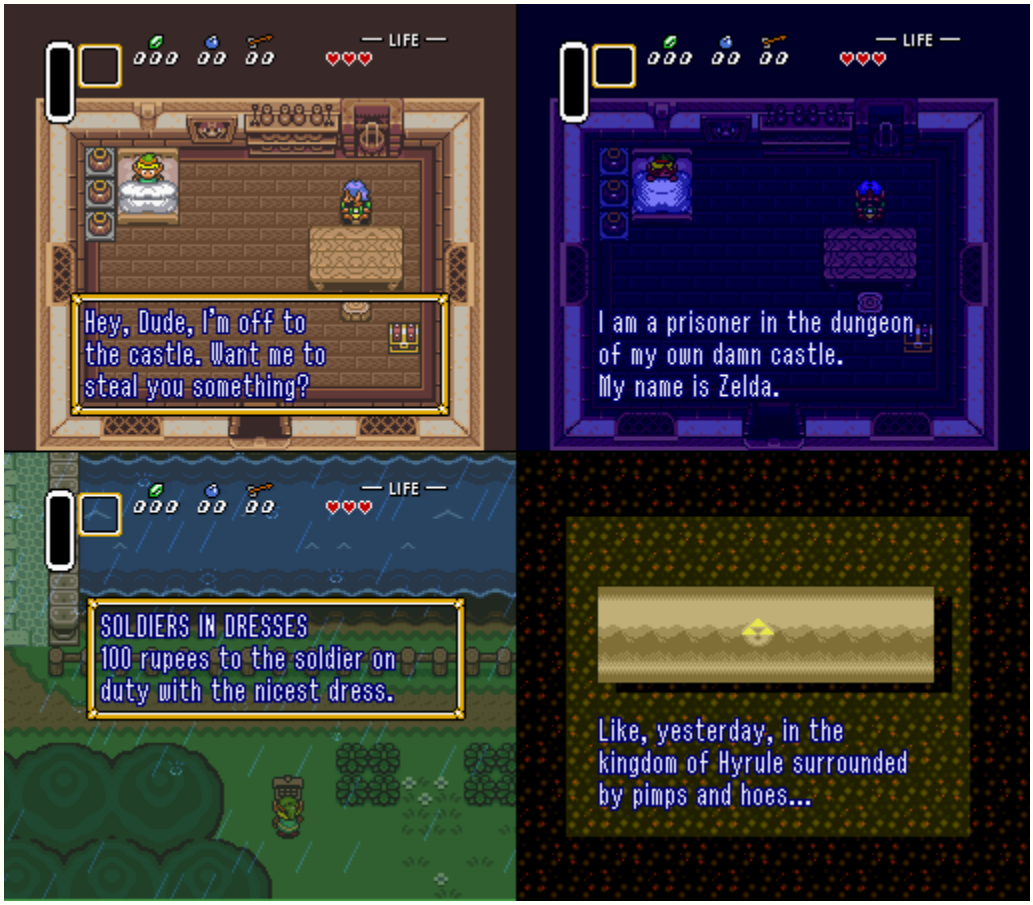
Hack: The Legend of Samus
Authors: Megamancpx
Information: Sprite hack
Release date: ???
Screenshots:



Latest version: 1.0
URL: http://www.zophar.net/download_file/4374

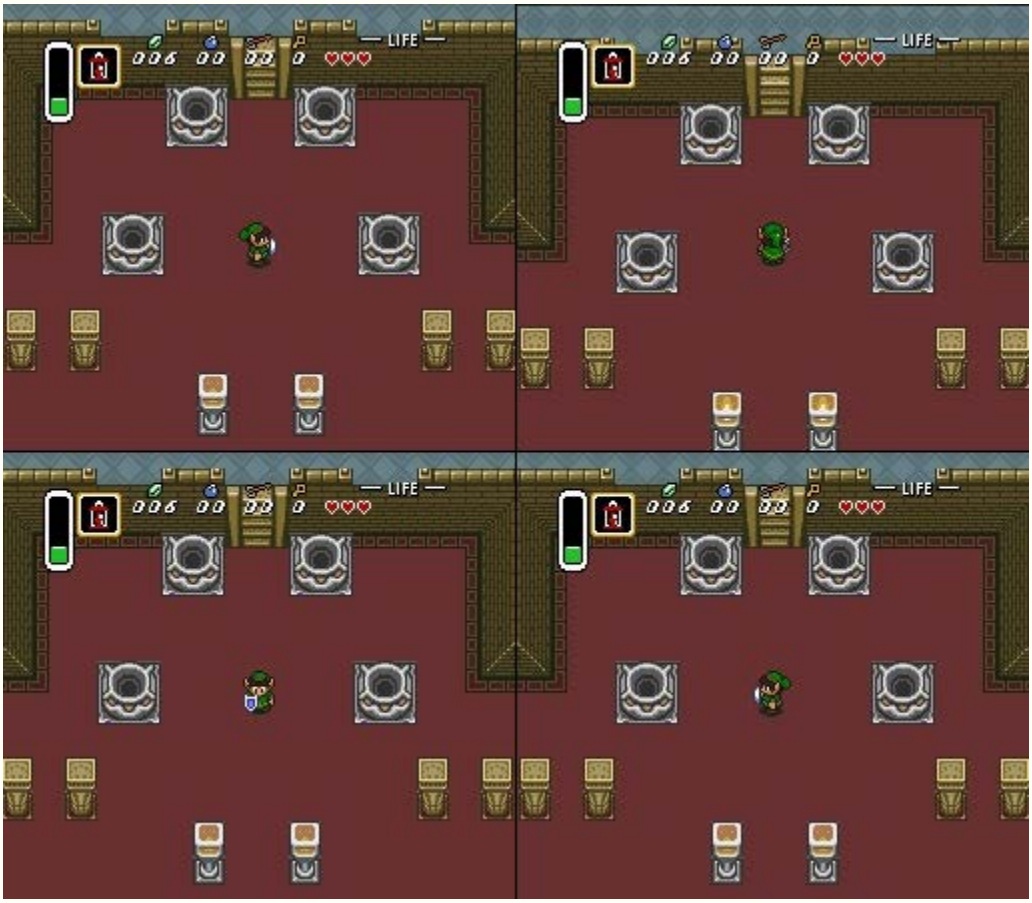
Hack: Link's Sex Change
Authors: Beneficii
Information: Sprite hack
Release date: 20 November 2004
Screenshots: N/A
Latest version: 1.0
URL: https://www.dropbox.com/s/4qbeoegwr6s7gnd/altpgirl.zip

Hack: Zelda: A Swear To The P*st
Authors: Meliz
Information: Spoof hack, text hack
Release date: 08 January 2008
Screenshots:



Latest version: 1.0
URL: http://www.romhacking.net/hacks/358/

Hack: Link's Pink Be Gone
Authors: Vacent
Information: Palette hack
Release date: 14 August 2012
Screenshots:



Latest version: 1.0
URL: http://www.romhacking.net/hacks/961/

Hack: Zelda 3 - Alttp Minor Fix
Authors: Eternaldragoonx
Information: Palette hack
Release date: N/A
Screenshots:

Latest version: N/A
URL: http://acmlm.kafuka.org/archive2/thread.php?id=4019

8) Discontinued/Cancelled hacks

Hack: The Legend of Iced Hyrule
Authors: richyawyingtmv
Information: Completely new dungeons, overworlds, with new music
Release date: ???
Screenshots:

Latest version: Demo, abandoned a long time ago

URL: http://acmlm.kafuka.org/uploader/get.php?id=3293

Hack: The Legend of Zelda - Dodongos Gold
Authors: NEONswift
Information: Abandoned a long time ago
Release date: ???
Screenshots:

Latest version: Demo
URL: https://www.dropbox.com/s/9m8ws58vd2cng2i/zdgDemo.ips

Hack: The Legend of Zelda - Omega
Authors: Omega45889
Information: Completely new dungeons, overworlds
Release date: 19 July 2004
Screenshots:

Latest version: 1.0
URL: http://www.romhacking.net/hacks/613/

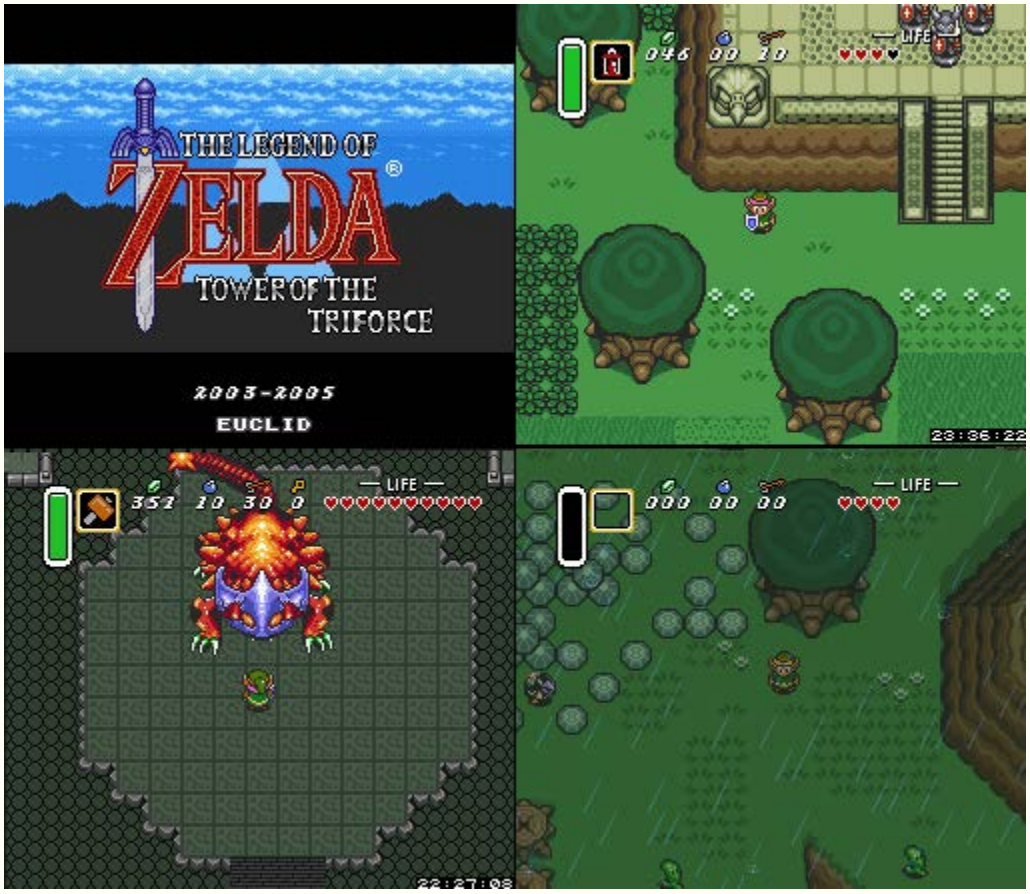
Hack: The Legend of Zelda - Tower of the Triforce

Authors: Euclid

Information: Euclid's original hack before I joined and it became Parallel Worlds

Release date: ???

Screenshots:



Latest version: Demo

URL: http://acmlm.kafuka.org/uploader/get.php?id=3250

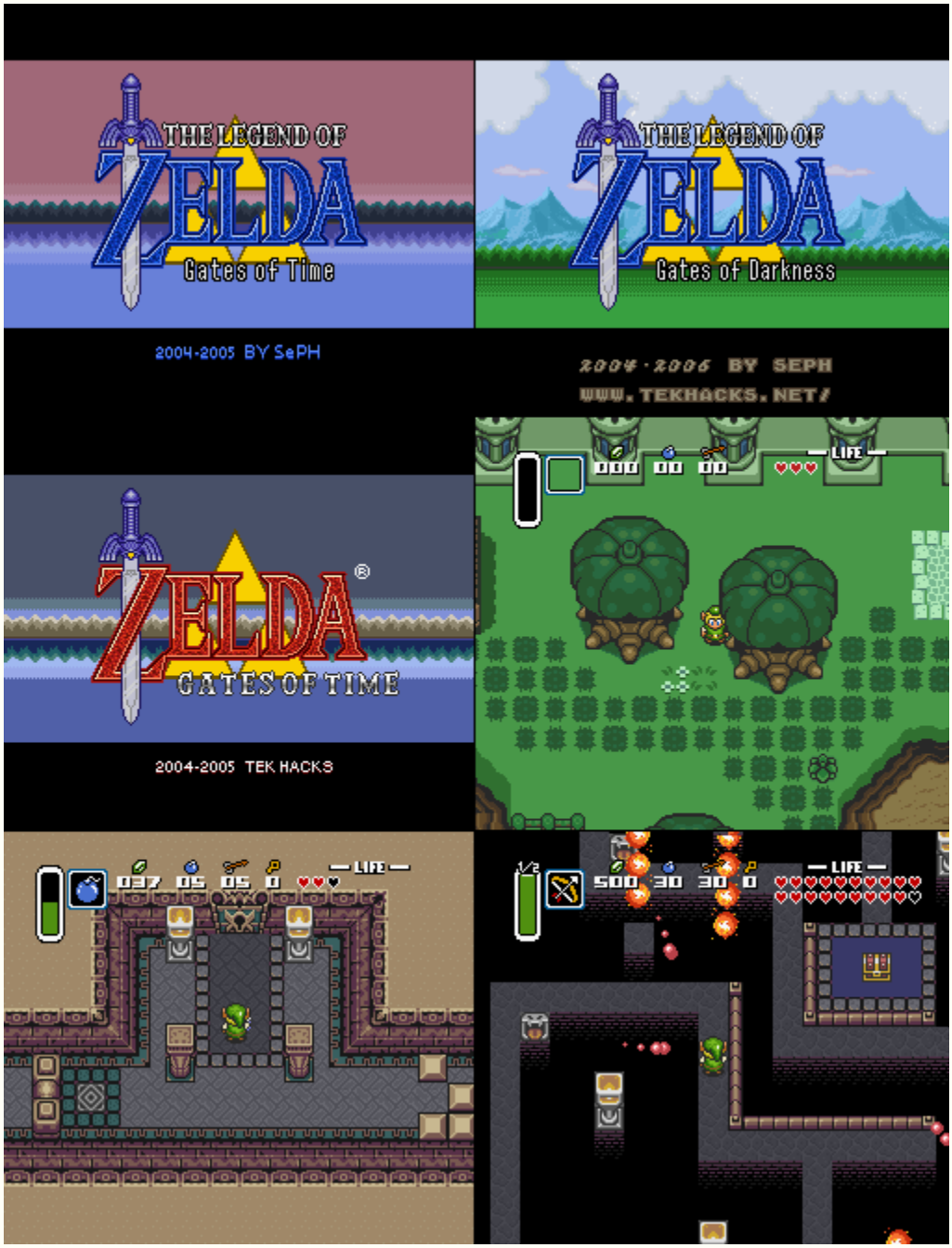
Hack: The Legend of Zelda - Gates of Time/Darkness

Authors: SePH

Information: Abandoned a long time ago, picked up again as a project by PuzzleDude

Release date: 11 May 2010

Screenshots:



Latest version: Demo

URL: http://acmlm.kafuka.org/uploader/get.php?id=3146

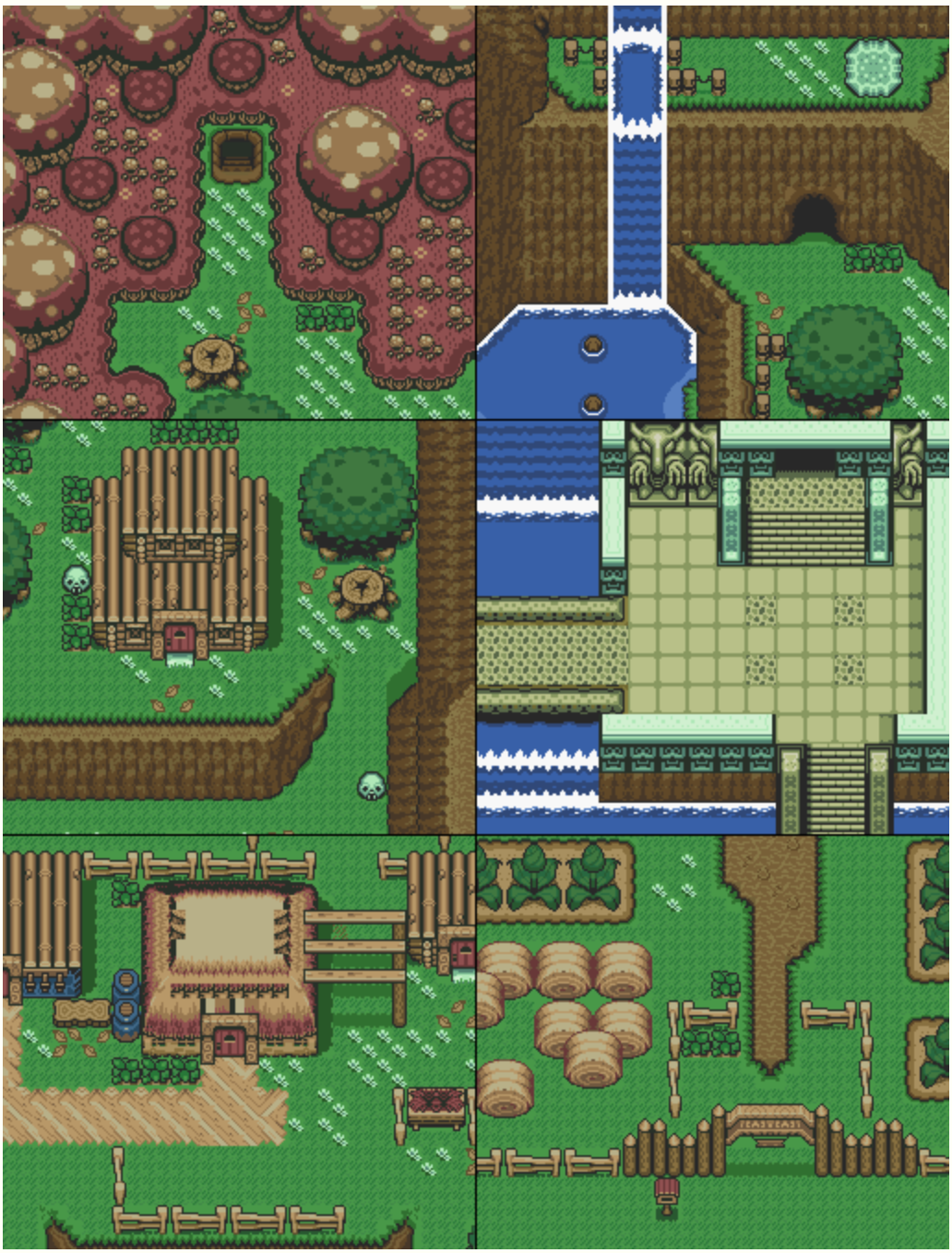
Hack: The Legend of Zelda - Lyra Islands
Authors: SePH, ghillie
Information: Abandoned a long time ago, picked up again as a project by Zeniea
Release date: 11 May 2010
Screenshots:

Latest version: Demo
URL: http://acmlm.kafuka.org/uploader/get.php?id=3147

Hack: The Legend of Zelda - Shards of Might
Authors: Omega45889, SePH, Dude Man, MathOnNapkins
Information: Abandoned a long time ago, completed as a project by PuzzleDude
Release date: July 08 2009
Screenshots:

Latest version: Demo
URL: http://acmlm.kafuka.org/uploader/get.php?id=1956

Hack: Legends of Hyrule
Authors: NEONswift
Information: Abandoned a long time ago
Release date: N/A
Screenshots:



Latest version: Demo

URL: https://www.dropbox.com/s/znf72ii6dgprh8q/Legends-Of-Hyrule.zip

Hack: The Legend Of Zelda - A Link To The Past 2
Authors: Aeranima
Information: His project before being renamed Renascence of Evil
Release date: N/A
Screenshots:

Latest version: N/A
URL: http://zfgc.com/forum/index.php?topic=37231

Hack: Ruins of Rockvan
Authors: Torin
Information: Completely new dungeons, overworlds
Release date: 12 April 2009
Screenshots:

Latest version: 4.0
URL: http://acmlm.kafuka.org/uploader/get.php?id=2949

Hack: Bruce Campbell Vs Ganon
Authors: KGP4death
Information: Spoof hack
Release date: 31 October 2009
Screenshots:

Latest version: 1.1
URL: http://www.romhacking.net/hacks/607/

Hack: The Legend of Zelda - A Link To The Future
Authors: bregegrahf1
Information: Story told in multiple chapters, chapter three extends the gameplay of Shards of Might a little
Release date: ???
Screenshots:

Latest version: ???
URL: http://zelda-a-link-to-futur.blogspot.ca/

~ Epilogue

1) Version history

Version 1.0 - (??-??-13)

Initial Release.

2) Legal

The **Hyrule Magic**, **ZCompress** and **YY-CHR** programs and all the other tools described in this FAQ (hereafter referred to as "ZTools") are not official or supported by Nintendo or any other commercial entity.

The ZTools are freeware thus they can be distributed freely on following conditions:
1) This document is for use with the ZTools and both the document and ZTools are not modifiable in any way.
2) The ZTools are not distributed with or as part of any ROM image in any format, and
3) No goods, services, or money can be charged for the ZTools in any form, nor may it be included in conjunction with any other offer or monetary exchange.

The ZTools are provided AS IS, and you use them at your own risk. Anyone mentioned in this document will not be held liable for any damages, direct or otherwise, arising from theirs use or presence.

Hyrule Magic: http://www.dragoneyestudios.net/index.php?page=utilities&id=6
Mirror: http://www.zophar.net/utilities/download/hmagic.zip
ZCompress: http://fusoya.cg-games.net/zelda/index.html
YY-CHR: http://www.briansemu.com/yymarioed/

3) Credits ---NEEDS UPDATING!!!!!

Editor made by **Sephiroth3**
Beta-Testing by **Jonwil**, **Yarx** and **GameMakr24**.

FAQ written by **Orochimaru** (aka **SePH**),
- Data gathering across the boards, compiling and correcting.
- For most lists (Overworld & Dungeons Sprites GFX# and such things...)

Big Thanks goes to:

Euclid
- For his Hyrule Add-ons Editor.
- For providing numerous valuable data.
- For answering so many questions.
- For some pictures.

Omega45889
- For the Forest Background Easy Hex Edit.
- For some help on the damage table (Hyrule Add-ons).

NEONswift
- For starting the original Zelda 3 FAQs Thread at TEKHacks.net.
- For giving me inspiration to start this FAQ.
- For answering so many questions.
- For some pictures.

MathOnNapkins
- For his RAM document.
- Chrono Trigger/Final Fantasy Style Chests

Sephiroth3
- For the original help file contents.
- For the data of the X scroll, Y scroll and the exit door for entrances.
- For the Treasure Chest Additional Data offsets.
- For creating the best editor out there.

Jonwil
- For the Item Pond Data offsets.
- For the Shop Data offsets.

dxEDGE
- For the Dungeon GFX# List.
- And some notes.

d4s
- For his WLA DX Assembler Notes.
- For some CPU addresses.
- For Dungeon GFX#218-219 Clarifications.

ghillie
- For his YY-CHR Tutorial.

Dude Man
- For his Dungeon Tips.

Commando125
- For the Blocks Types list.

Chickenlump
- For how to show the correct GFX palettes in YY-CHR.
- For supplying hex offsets from the old Acmlm's board.

Jaspile
- For finding various hex data using Evil Peer's Snes9x Tracer.

solid-tbone
- For his HEX lessons for beginners tutorial.

FuSoYa
- For the help on using Zcompress.
- For the Legal layout.

Weasel
- For his GameGenie Codes to HEX Converting Guide.

Peekin
- For the Tile Locations.
- For the Decompression/Compression Codes.

Unknown
- For finding the Outdoor Layout Location.
> And everyone that I forgot.

"Fun fact: Orochimaru = SePH!" ~MathOnNapkins