RomHacking 102

Font & Sprites

Index

Introduction & Tools:

The document will guide you to get and reinsert the sprites and font (for a gb/gbc) from scracth, using a GBC game as an example. A good advice is reading http://www.romhacking.net/start/ first.
However many of the technics, procedures and knowledge involved may be and are probably used in other game console/hardware.

Methods from 02 and 03 are interchangeable but to make the document easier to swallow and less repetitive it has been decided to use each with a different goal, with no preference in mind.

For this document/tutorial, you will be required to have and use the following tools.
Windhex, https://www.romhacking.net/utilities/291/
Game’s ROM: Kikansha Thomas - Sodor-tou no Nakamatachi (J) [C][!]
BGB emulator, https://bgb.bircd.org/
Tile molester, https://www.romhacking.net/utilities/109/

Preface

"It's dangerous to go alone! Take this."
The legend of Zelda, NES game (1986).
For those who don’t know it, in that game, this is spoken by an unnamed old man, met in the very first cave of the game and the player is given no further explanation on how to progress within the game world. The ROMhacking world is usually like that.

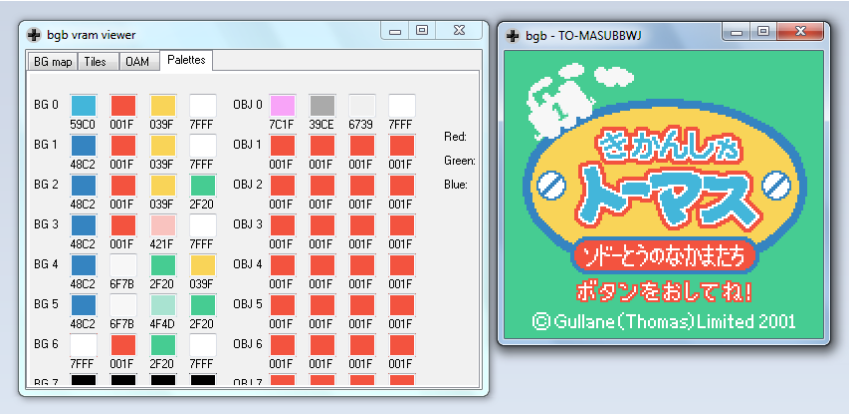
Step 01, Palette

First you'll need to find the palette offset, in this step, you will use a relative search approach for that.

[BGB has a window to show a graphics view on VRAM]

Open BGB and load the ROM, now when the title screen appears press right click in the mouse and select, other > vram viewer (or to the palette viewer in vba).
From the popped up window go to see the palette.

You will see something like this [1]



[1]

Here you can see various palletes from B0 to B7 (mind you, if you read more technical documents, palettes have different names in. And BGP/BCP would be more consistent with the register names).

That is because a graphics view on VRAM has all background and object pallets displayed in it. Besides, it shows what tile is using what pallete.

Which means the 4 color pallet with its hexes value in 2byte format are,
59C0 001F 039F 7FFF

The game boy console uses little endian so you need to swap the hex bytes of each color
[Go to my previous document, RomHacking 101, or read about the gameboy console if you want to know more about the technicals of that]

[Also, the individual palette bytes are loaded in back to front for each entry]

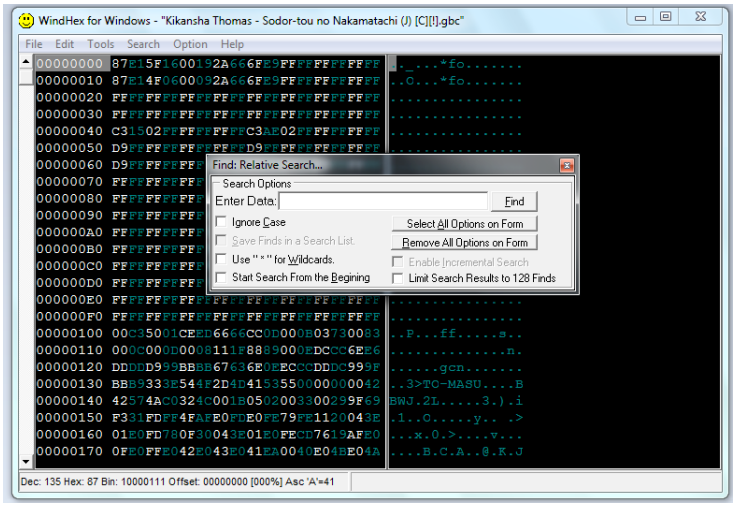
So, swaping bytes,

59C0 is C059,
001F is 1F00,
039F is 9F03,
7FFF is FF7F.

Which means you have to look for this string C059 1F00 9F03 FF7F in the ROM

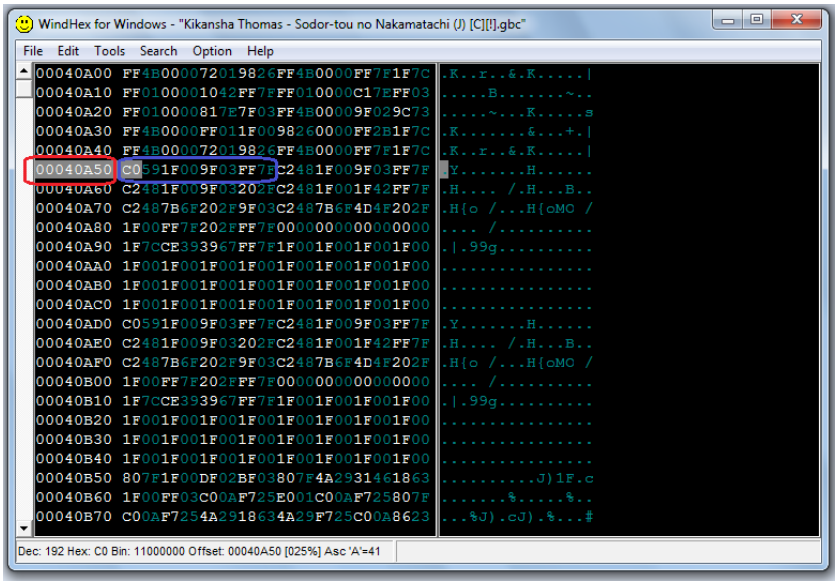
To do that, open windhex or the hex editor of your preference, load the rom:

Open File for Editing or ctrl+ o
Now go to Search>Hex Search or ctrl + h
This is to find the occurrences of one string inside the ROM file.
You will see now something this [2]



[2]

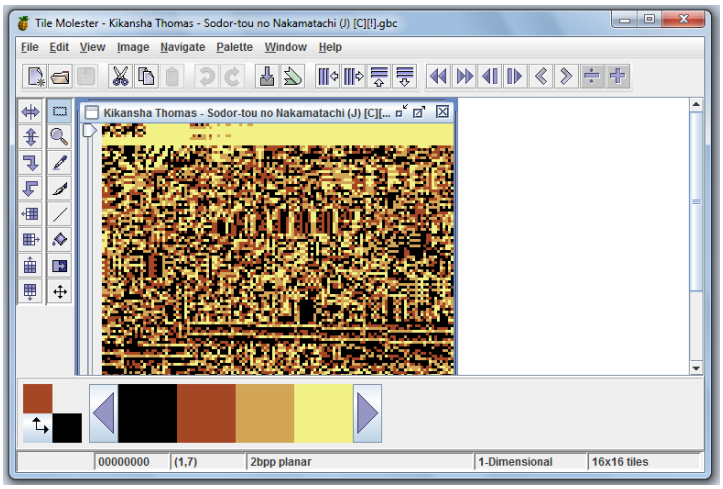
After the search, of C059 1F00 9F03 FF7F, the first result comes at the address 0x40A50, see [3]



[3]

With that value it's time to open Tile Molester and load the ROM (File>Open).

You may see something like this [4]. Don't be afraid, you are looking at the room in a graphical way, but that doesn't mean that all inside are graphics. [And even if they are, some may be compressed or the viewer is using the wrong displaying format]



[4]

Be careful to see the tiles from the same screen of the palette you are editing. Otherwise you will go crazy without noticeable changes in the game, and eventually making unexpected errors.
In Tile Molester, introduce the screen address, Navigate>Go to…
Offset: 0x40A50
Leave the other options as they are (“hex” and “absolute”)

If you’re using the same Tile Molester version from this document… You need to convert that value to Decimal. [I assume this is the developer’s way to avoid users to input the wrong values in either of the bars. There are Tile Molester mod versions that let you write both Hex and Dec].

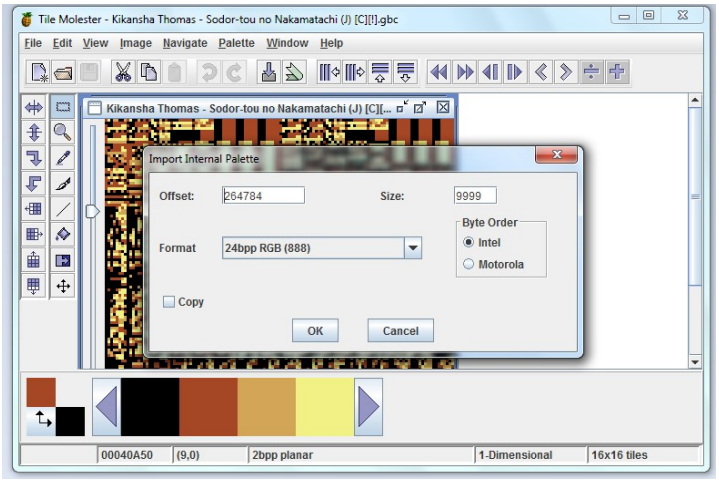
After the conversion, the offset 0x40A50 becomes 264784.

It’s now time to try the value and see if it’s the correct palette:
select Palette → Import From → This File…
Fill in the offset bar with the number you just calculated (264784), and set size to 9999 (this is to make sure it has enough space for the palette). Like this [5]

We assume Tile Molester put Format (24bpp) and byte order (intel) correctly. If not, you will have to search how the console of your project works and change those values.

Click OK, and the palette below should have changed. The palette shouldn’t change if the palette is not correct (or in the same bank at least). If it didn’t change, go back and repeat the process until you find the correct one.

[Despite being a wrong palette, in this case the color changed, that’s why a bit of in-game testing is always good as a sanity check]



[5]

You made it, you changed the palette Congrats!

Close any other Programs that may be Using your ROM. Otherwise, the save will fail due to the file being currently in use. And save your edits, File>Save as…
[Make sure you saved it properly]

After doing it, you can run your game and check the changes, if by any chance the palette was wrong or you want to make more edits, you just have to redo the same process again. Go and have fun changing games' palettes.

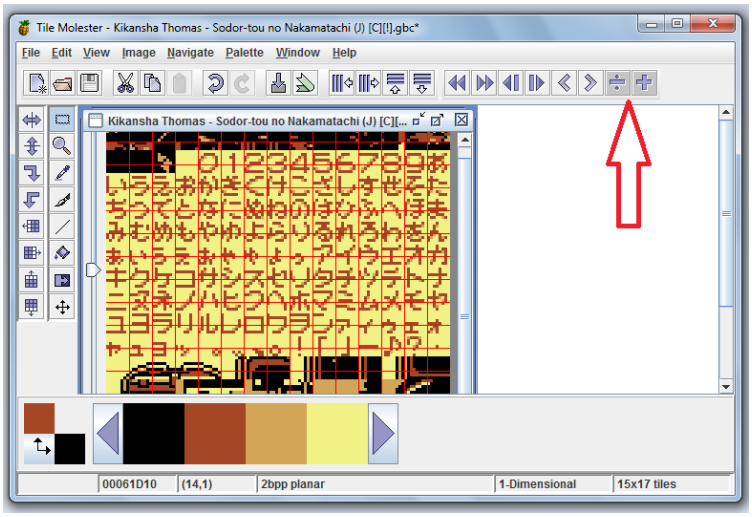
Extra note: As it was said before, this palette was wrong, or not used. For that reason, here are the two palettes of the main title screen of this game, so you can play-test with them,
0x53a49 = 342601
0x53a89 = 342665

Step 02, Font

Since this step uses the brute force method, you will need only your ROM and your tile viewer of choice, this example uses tile molester.

Open tile molester and load the ROM, File> Open, or ctrl+ o.
[For this step is adviced to activate the view>Tile grid, to find the right tile's offset].

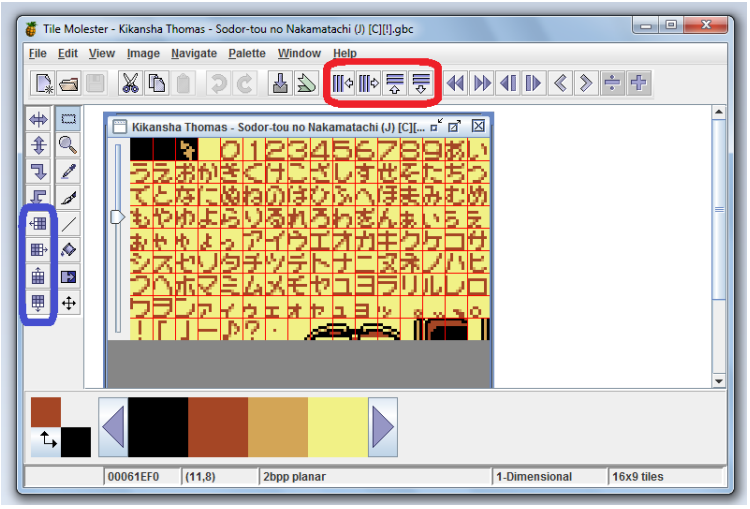
Just scroll you're way until the sprite comes out. In this case at address 61EF0. You can see it in [6].



[6]

You may need the right offset to see the letters properly, you can find it going to the next or the previous bytes in the ROM. (clicking in the buttons pointed from the red arrow in [6]).

You may want to have the font in a nice display [7].



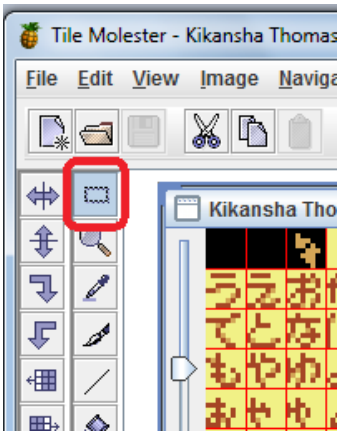
[7]

To do that go and use the red and blue sections in [7].

Sometimes, specially with sprites , you may need to check View>Row-Interlieved

Now that you have found the font in the ROM, is time to edit it. You can do it in the tile viewer (editor) or you can select the bitmap and export it to make the edits in your image editor of choice.

To export the file, select the font bitmap [8], go to Edit> copy to… and save it as a jpg/png/bmp file. [PNG and BMP are the recomended formats].



[8]

Once you've finished your new font (or you have chosen one from other sources), go to Edit> Paste From… and import your new font.

Make sure the previous bitmap has been overwritten. An "*" should appear indicating you've made changes on the file.

Close any other Programs that may be Using your ROM. Otherwise, the save will fail due to the file being currently in use. Now, save your edits, File>Save as…
[Make sure you saved it properly]

You can now run your modifyed ROM (game) and check the changes.

If you made any improper change to the palette, or you modified something you shouldn't, the file will be corrupted and the game will crash. [For the record, this is a method to check things too].

If everything went well, congratulations, you've changed the font of your game!
[If not, go back and check/redo the previous moves]

Step 03, Sprites

In this step we are going to learn how to take advantage of the VRAM (and VRAM graphics viewer) to find things with the debugger.

If you have previous experience or don't have a VRAM graphics viewer in your emulator, you may go right to the debugger's view part.

Mind you, some emulators show you the ROM/RAM only in a debugger's view (hex and dec values). BGB has a window to show the graphics from VRAM. We are taking advantage of that.

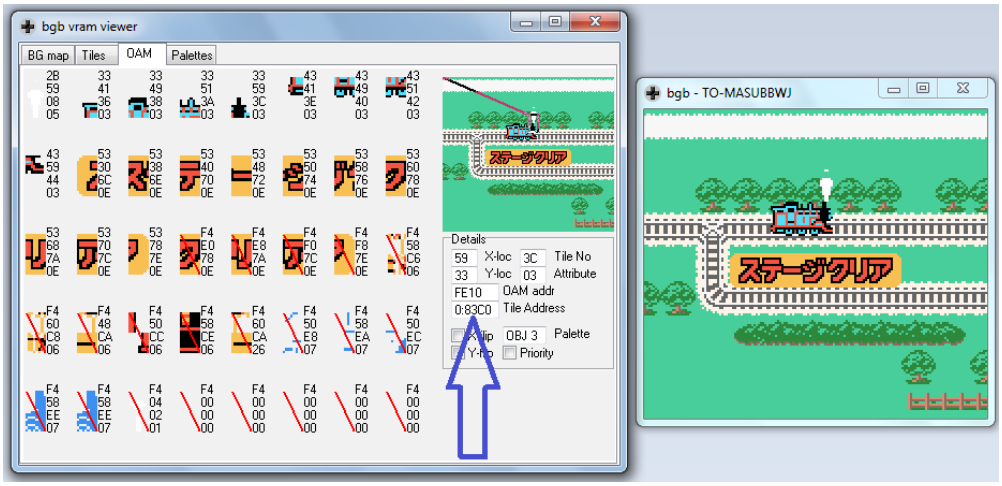
First thing you have to do is start the emulator and load the ROM.

Play, or load a save state, when the screen is displaying the assets you are trying to find.

The Reverse Engineering process starts once you are in that screen.

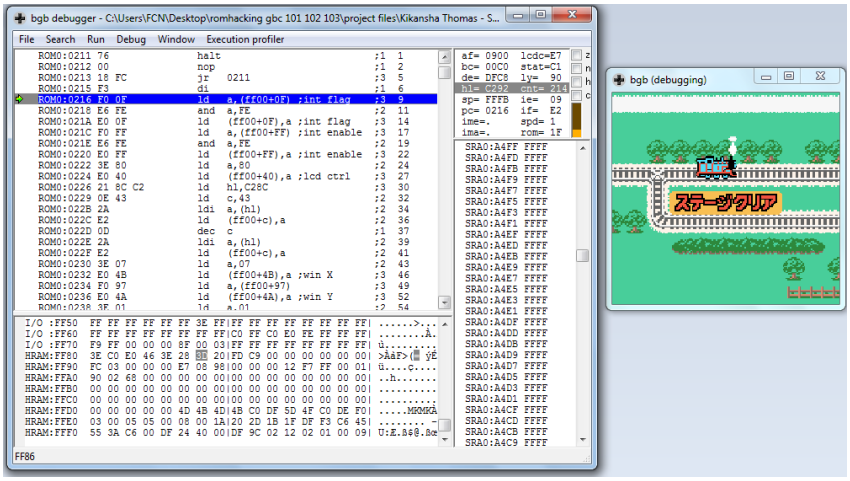
Reminder: this method uses volatile memory. Pause emulation in the desired screen, to avoid the Screen Refresh messing with your work.

You can find the tile address for the debugger's VRAM section, in the "Tiles" section or if it's a sprite asset, you can use the "OAM" section [9].
[To learn more about the OAM for the game boy, go to https://gbdev.io/pandocs/OAM.html].



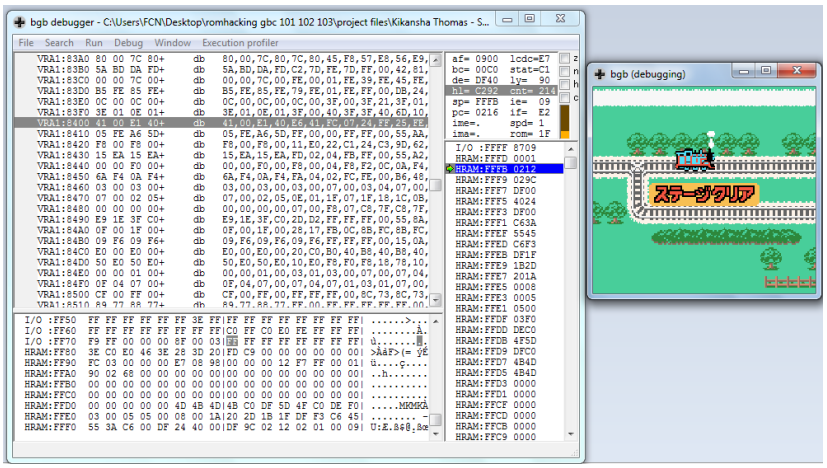
[9]

You may take a few consecutive tile addresses, to avoid false positives.
Note them down, and open the debugger. [10] [11].



[10]

If you use the search function to locate the bytes noted before, you will see the only bank showed in the emulator is the VRA1, but the tile address pointed to the VRA0 (remember the “0:” part), see [11].



[11]

This is because the GameBoy VRAM is divided into two banks, both mapped to the same part of CPU space with a bankswitch control somewhere.
[more about it in https://gbdev.io/pandocs/CGB_Registers.html]

From here on, the procedure becomes more in deep about technical knowledge of the ASM and the GB hardware, which you can read about in any of the previous links or by searching through the web.

The scope of this document was to make an introduction for those who wanted to get into romhacking or review the basics after long periods far from the scene. Due to that, this step needs to be halted here.

Extra note: Another approach with the debugger is to put a break point to see when the data for the first tile was written and then look to see where the data was being read from.

Step 04, Possible expected problems

Since GB 2BPP only allows for 3 colors + transparency, just like NES 2BPP, it uses two superimposed images to get 6 colors. [it's almost always sprites]
If you want to dump those, you'll be using 2 pics, one with each palette loaded.

Sprites are compressed, you'll first have to find the compress/uncompress routine in the ASM code. After that, you will have a couple of options:

> Stub out the routine, so it doesn't load the old compressed graphics (or any indeed).
>
> Let the decompression routine happen, but with your own compressed graphics.
>
> Let the compress graphics routine happen, then memcopy your uncompressed graphics after.

Further explanations of compression are out of the scope for this document, however you may need a verification step, for times when the data is compress, that is a corruption attack: "Overwrite the bytes in ROM that you found and see if it trashes the graphics tile".

Extra documentation

Theoretically you shouldn't need any other extra document to get what it is explain in this document, but for those who want to dive deep or get extra technical info here are (in the document author's opinion) some of the best documents to get you started.
[Although the length varies, they are not very long].

Displaying a 2BPP Tile, by Lin, http://www.romhacking.net/documents/457/
"A brief document explaining how 2BPP graphics (Gameboy, other systems) are displayed."

Console Graphics Document, by Klarth, http://www.romhacking.net/documents/21/
"Covering 11 of the most common console graphic formats", the bpp for gb/gbc among them.

Title Screen Hacking for the Gameboy with NO$GMB v1.3, by Catalyst
https://www.romhacking.net/documents/32/
This paper teaches you via hex edit, literally. You'll be using part of his methods, the main difference is that you won't be editing the sprites in a hex editor. (Written in 2000)
[I think editing graphics and palettes with graphical view it's preferred. If you prefer hex editing them, you're free to do it]

Everything You Always Wanted To Know About GAMEBOY, Pan of ATX- Document Updated by contributions from: Marat Fayzullin, Pascal Felber, Paul Robson, Martin Korth and Last update 17-Mar-98 by kOOPa https://www.romhacking.net/documents/309/

A Tile Molester guide, by felixwright
https://wiki.metroidconstruction.com/doku.php?id=general:guides:tile_molester_graphics_guide
Self-explanatory. [Also, I got some of my explanations from there].

And the last two. I saved them for the end because you'll have to dig a little through google to find them, however the effort is worth it.

"Editing fonts without the aid of a Graphics Editor! A rather useless document!" by Neil_

"An Introduction to Understanding/Hacking 8 bit Graphics" by prez

Additional notes

The 102 document ends here, if you found an error and want to report it please send a private message to Bunkai in the https://www.romhacking.net/forum/

If you have a question about the process you may do them in the forum or in the discord, using the proper channels for that. But I won't answer any private question about your projects.

A translation project should include: script dump, fonts and sprites edit, and script insertion.

This should be a series of documents to cover those 3, with an ordered naming (101, 102, 103) trilogy. Currently, this 102 document and the 101 documents exist, but not the 103 document. Since the text insertion document doesn't have a release date and may not be released never. Please don't ask about that.

PS: A style revision and consistency revision should be made, but the intention is/was to have the 3 documents or at least the first 2 out, so people can used them, and to include any bug fix in the review.

Credits

Guidance and explanations: 342, Bootleg Porygon, zlago, Abridgewater, 256, vinheim3
Document author: Bunkai

Special Thanks

To the RHDN discord server and the RHDN community in general for their wonderful support to this hobby with manuals, tools, explanations and so on.