Tilemaps:
Identifying, Modifying and it's Applications

Giorgio di Chirico -
Mystery and Melancholy of a Street. 1914

Knirt @ FUT / 2008 ~ 2009

Version History

Version 0.5 – March 04, 2008

Version 1.0 – April 30, 2009

Version 1.0 (EN) – August 7, 2009

Contact

E-mail – knirtsoftweb@gmail.com

FUT's Profile – http://www.romhacking.trd.br/index.php?action=profile;u=843

RHWiki - http://wiki.romhacking.trd.br/index.php?title=Usu%C3%A1rio:Knirt

RH.net - http://www.romhacking.net/forum/index.php?action=profile;u=6115

Legal Info

Index

- Introducion.....04
- What Are Tilemaps?.....05
- Applications on Romhacking.....06
- Identifying the ocurrence of Tilemaps.....07
- Modifying Tilemaps
  - First Method – Repeated Fonts, aided with a table.....10
  - Second Method – Repeated Fonts, without a table.....14
  - Third Method – Game Logo's.....15
- Appendix.....20

Introduction

Romhacking is a hobby that, in the vision of many of it's addepts, wishes to modify the data on a certain ROM, may this modification lead to some hack, or to a translation. In any of the cases, a decent romhacker always looks after perfection on your work. The graphical modification, sometimes, means more than just modifying some tiles on the editor of your preference. Not rarely, the same graphic is used a lot of times on the same image or screen, as a way of economizing memory, following some instructions given by the ROM, assembling a tilemap, and the simple alteration of this map can finish with some headaches and problems that, otherwise, were impossible to solve. The methods of modifying these maps, the techniques applied in the identification of a tilemap and it's applications will be the focuses of this document.

I wish, to every one of you reading this, that you can read and understand this document, and more importantly, conclude your project. Any questions regarding this document can be forwarded to my e-mail, I'll be happy to help as I can.

What Are Tilemaps?

In understandable ways, tilemaps are a reference to the emulator (or the console, for instance), that tells it how the tiles should be arranged on the screen. Let's take the NES to give some real examples. NES video output is a 256x224 pixels matriz (in NTSC mode). What does that means? That the NES will exhibit 256 pixels in height by 224 pixels in lenght, on the screen. Each tile is 8x8, so we can make some simple math to discover how many tiles we can fit on the screen. In this case, 256/8=32 and 224/8=28. So, the NTSC NES video output is made by 32x28 tiles. The tilemaps works as the reference as how these 896 tiles should be filled. If you, Romhacker, can undertand what the game plans to put on these locations (even without understanding HOW he plans to put it there), you will be able to modify this content.

If you want to succeed in the Tilemaps pathway, you must be familiar with some terms. One of the most important of them is the “Pattern Tables”. The console can load a certain number of tiles in it's internal memory (in the NES case, in it's PPU). They usually pick up blocks of tiles (in the NES, 16x16 tiles), and these blocks are named Pattern Tables. The NES can load up to 2 Pattern Tables at once. This Tables are located on the ROM, and when they are needed, they are moved onto the PPU. Once they aren't needed, they are replaced with another Pattern Table and it goes on. On platforming games, it's common to see one Pattern Table containing the backgrounds and the other containing character sprites, and in cutscenes, usually one of the Tables if the font, and the other is the cutscenes itself. Knowing the first byte of a Pattern Table is essential to create a Character Table, that will be very useful on our First Technique of Tilemaps Manipulation. But this technique will be teached later on.

Applications on Romhacking

The use of tilemaps is very common in graphics that are repeated a lot of times in the game, or graphics that, after a modification, are bigger than the original, requiring the shifting of some tiles from the actual selection. In some cases, where the tilemaps are used to end with the issue of font repetition (as will be shown later), it can be easily managed with the aid of a table file. Otherwise, in some situations where the tilemaps must be arranged to increase a game logo, for an example, the work is made on a very rudimentar way, without any specific software to help you, just some tools like PPU Viewer, a graphical editor and a hex edtor. To do that, we need to get note of all the hex values of a specific Pattern Table, and then, modify these values manually on a hex editor. But this will be explained later on.

Identifying the Ocurrence of Tilemaps

To idenfity and consequently modify the tilemaps, mainly two things are necessary:

- That the graphics are visible on the ROM;
- That the Romhacker knows how to use the hex search, on your preferred hex editor.

Depending on the case, another pre-requisite that will help you is being able to create a table file without using a external software, that means, only using a simple text editor, as the table may be out of order.

To make the teaching easier, I'll be using two games as examples. The first one is the NES game "Kabuki - Quantum Fighter", and I will teach setp-by-setp how to modify it's main protagonist ID card, something that, without tilemaps, would be near impossible. Furthermore, we'll be modifying the NES game “The 3-D Battles of the World Runner” main logo, and in the process, teaching on how to add your custom logo (ORLY?). Firstly, let's see the ID card mentioned early, as it is shown in the game. See Picture 01.

Picture 01 – Scott's ID Card

After some search on our graphic editor, we find the ID card located on the offset #0003D210H. We'll use Tile Layer Pro as a reference. See Picture 02.

Picture 02 – ID Card on it's Pattern Table

Well, let's modify the ID card as we would with any other graphic. I was in process of translating the game to the Portuguese, and NAME should be translacted to NOME. So I just exchanged the A for the O. Check Picture 03.

Picture 03 – ID's card Pattern Table after the modification

Let's check what happens in the game when we make this modification. See Picture 04.

Picture 04 – ID card in the game, after modification

Sex: Mole?! Oge?! These words make no sense in the Portuguese, and afteralls, how it happened?! Well, the answer is simple. That A we saw on Tile Layer Pro wasn't used as reference only for the word NAME, but also for AGE and MALE. That means we can't simply modify the ID card if it was a simple graphic, but instead, we must think of an way to edit it's tilemap, and that's where this document comes in. I'll introduce the First Technique for solving this problem now.

First Method – Repeated Fonts, aided with a Table

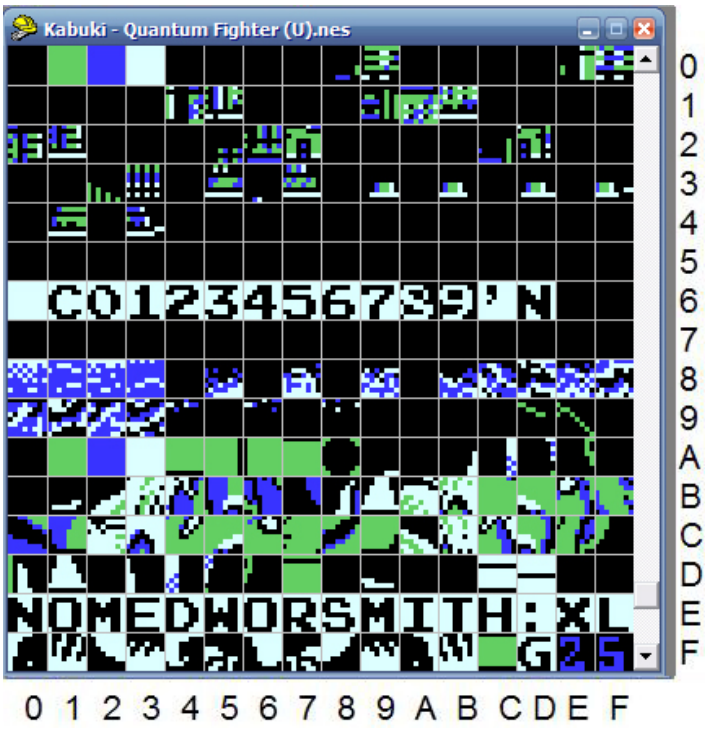
Making a relative search on the ROM, we find that it's character table corresponds to A=8A. Knowing that, we'll be able to center the Pattern Tables for the entire ROM in the Tile Layer Pro. On the program, let's find the game font.

Picture 05 – Pattern Table of the font, mapped

Look to the Picture 05. Notice the markings out of the Tile Layer Pro screen. The right side markings are referent to the first byte, and the lower side markings are referent to the second byte. Following this logic, if you look to the position 8A, you will find that it's over the letter A. Once we can make the markings match with the table, we will have the Pattern Table centered. This procedure is very important in the tilemaps manipulation, as the PPU ALWAYS load the Pattern Table centered. Whenever I mention centering and such, remember this procedure. Just to remember, if we press Page Up and Page Down on Tile Layer Pro, we can move between the Pattern Tables (if one of them have been previously centered). Usually, NES Pattern Tables start with four tiles filled with different colors, that represent the Pattern Table Palette.

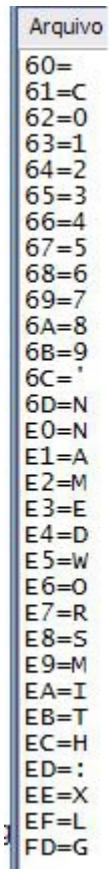
It's important to note that, if you reach the end or the beginning of the ROM, the Pattern Table will probably decentralize.

In our case, pressing Page Up once already takes us to our desired Pattern Table. It's shown on Picture 06. Using the markings, we can easily find the hex values of ALL THE LETTERS used in the ID card. We can't forget either that E1=A, and that it must be restored to the original value so we can see the image as it was originally. Another interesting detail is that the Pattern Table that contains the ID card has the four first tiles filled with the palette colours, confirming our previous affirmation. If we would modify the palette, we already would have a reference, these four tiles ;)



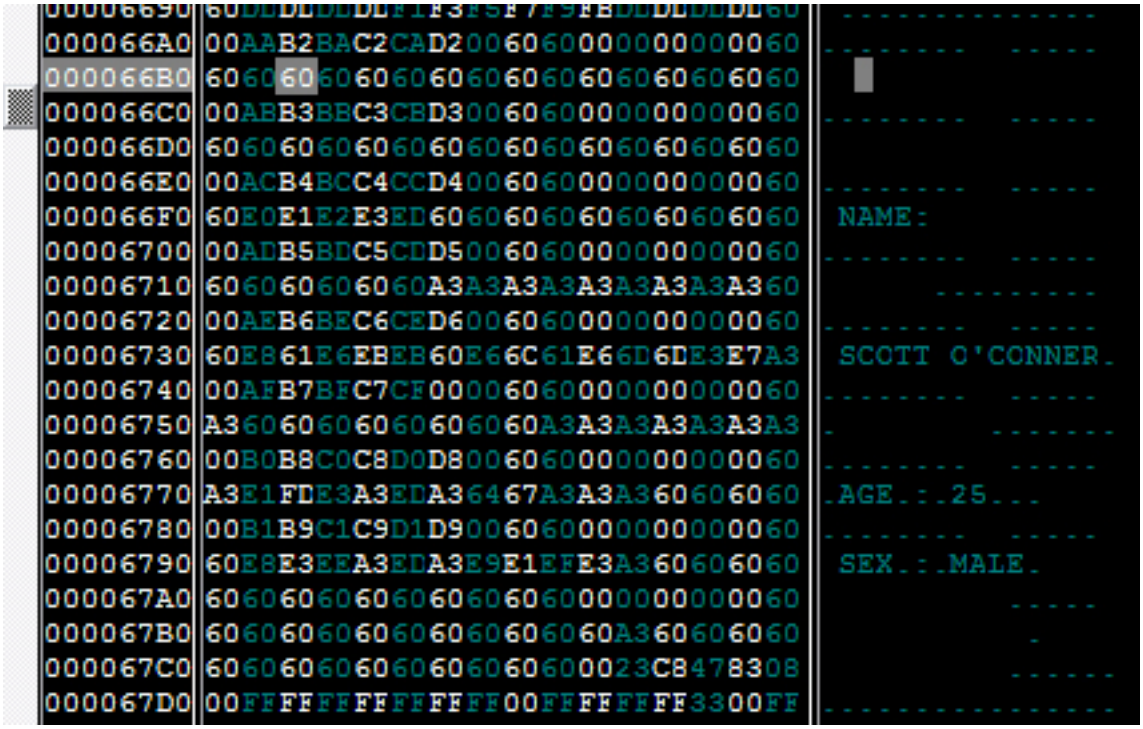
Picture 06 – ID card Pattern Table, after the conscient modification

From here on, the graphical editor won't be needed anymore. The remaining work will be accomplished using the Hex Editor and in the text editor (to make the table). Talking about that table, let's visualize how it'll look on the Picture 07.



Picture 07 – Character's Table

Loading that table on your Hex Editor, and making a simple search for the string NAME, we will find exactly where the text of the ID card are located, as seen in the Picture 08. You'll also find the other info relative to the ID card. From here on, you can modify all this info directly on the Hex Editor and check how the modifications will work in the game, by running it. If the ID card is sucessfully edited, I must say congratulations, as you probably understood the basic working mode of the Tilemaps!



Picture 08 – Final Considerations About the First Method

Some final considerations can be made abour Picture 08. Notice how the words AGE and SEX are aligned with two lines between one and another? In the game, doesn't them appear exactly one over the another? That happens due to the screen size of the NES, that fits 32 tiles of lenght, while the hex editor just fits 16 tiles. Another thing: do you notice the values just below NAME, over SCOTT, and going on? They are not shown in the hex editor, as they are not in the table, but if you kept up with the process until now, you will be able to tell me that they are correspondent to the Scott's ID photo. The values over NAME probably are the tiles used to assemble the eagle in the ID card.

Second Method – Repeated Fonts, without a table

The idea of editing without using a table is exactly like the idea of using a table editing, however, depends more on memorization capacity, fast-thinking, and the need to make an edit on-the-fly without spending time creating tables and so on. To illustrate the method, we'll use pictures and some methods of the previous example.

We saw that using the relative search upon the content of an image may not return favorable results (as happened when we searched NAME on the ID, we would never find the correct character table using relative search, due to the characters not being ordered in this table in special, as we checked after seeing it's Pattern Table). Therefore, we must repeat the procedures of centralization and localization of the Pattern Table concerned. Once we can get on screen the contents of the Picture 06 (with the exception of tile E1, which should be an A instead of an O), we can advance in our method.

What happens in this method is that, with Tile Layer Pro open only as a reference, we can go straight in the hex editor and do a conventional search for E0 E1 E2 E3, and the search result will be most likely the place of occurrence of NAME in Scott's ID card! Changing the tile E1 A to O, we can check whether the occurrence found is what we had imagined or not.

What happens is that, just doing this, the problem still occurs to OGE and MOLE. But that should not stop us, because we know the place where they are (a few lines below the occurrence of NAME), and we know which bytes must be modified. Thus, just below the E0 E1 E2 E3 we must find E1 FD E3 (which will be changed to EA E4 E1 E4 E3 – to free up more spaçe, shift the AGE to the right using null bytes, as 00 or 7D) and E2 E1 EF E3. If you followed the method so far, you've noticed that we realized all the changes without loading a new table, and a simple test will tell us if we had success this way!

If this method did not work, do not feel discouraged. It's analogous to the first method, then try the first one or two times more, and then return to this one! This method is only a little more practical and fast, but the final result is the same! It's important to practice it so you can become familiar with the third method quickly.

Third Method –

Game Logo's

After the presentation of the first two methods, not only had a broad and comprehensive concept and the manipulation of Tilemaps, but also had some examples of the applications. However, anyone unsuspecting can look to these examples and underestimate the true power of the Tilemaps. This third method is very similar to the first two, which involves the basic idea of them with some additions - a table will not be useful now (bearing in mind that we are not dealing with texts), so, the practice of the second method is fundamental for the sucessful execution of the third method. Thus, we'll use bytes that weren't used before (like the sequence of black bytes of Picture 06, line 7) to assemble pictures bigger than the original ones.

For this, we'll use the game "The 3-D Battles of the World Runner" as an example, whose title-screen is a big logo, with many things written, but which can not be encoded in a table (by not following a rule of 1 byte - 1 character). Here you can really see how changing a tilemap may increase, and much, the quality of a translation, especially of a title-screen, making the translation instantly more attractive. It applies equally to hacks and such.

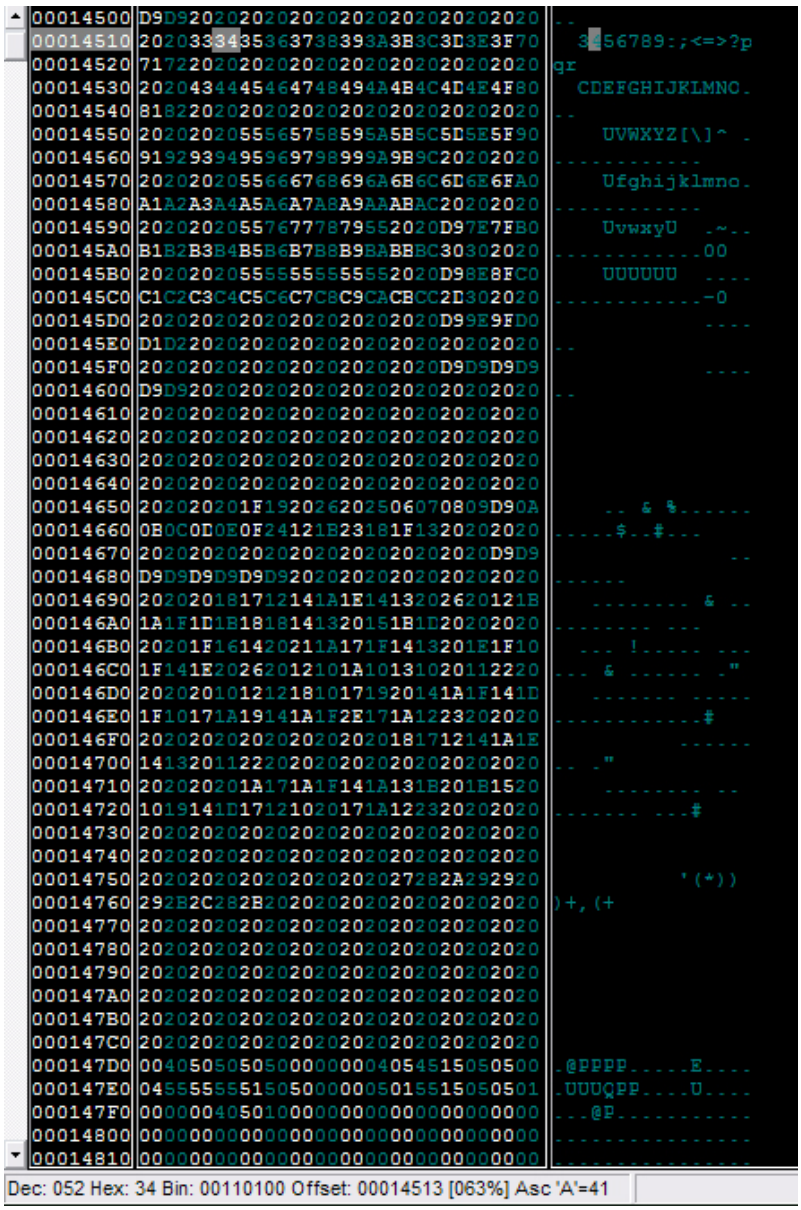
Picture 09 – Title-Screen Pattern Table

Firstly, let's give a look at the Picture 09. I've already centered the Pattern Table in this picture, the one that contains the elements of the title-screen. In your case, you can simply go to the offset #00014810H. We can draw two interesting conclusions of this image. The first is that the title-screen is completely disorganized, and it'll require an absurd amount of work, if we use the previous methods. The second conclusion is that those black bytes to the left of “World” may be useful.

While we can map the entire title-screen, then move it conveniently to the corner, remap it in the ROM (swap the old bytes for new bytes) and then edit it graphically, this work wouldn't be compensatory. Instead, we can simply modify the title-screen using the frame manipulation techniques of the Tile Layer Pro, and if it lacks space, we can fill the empty bytes with things that are interesting for us in the moment.

But since this document is aimed only to teach the techniques of manipulation of Tilemaps, not to really modify the game title-screen, let's take the liberty to customize our game. First, we map the title-screen.

Looking in the Picture 09, we'll go to the hex editor and search for the string 34 35 36 37 38 39 3A 3B, and see if we'll find something. Touché, in the offset #00014513H, we have the title-screen, according to the Picture 10. Notice that after the 3F we have a 70, which is completely understandable, if we check the Pattern Table as reference. Also see that, when the game wants to fill a portion of the screen with black tiles, it uses the byte 20. That shows us that we can modify the bytes located left of “World” without any complication. And that's what we'll do now. Once you understand the way these bytes are being modified, you'll see that you can use them to fit custom logo's that are bigger than the original ones.



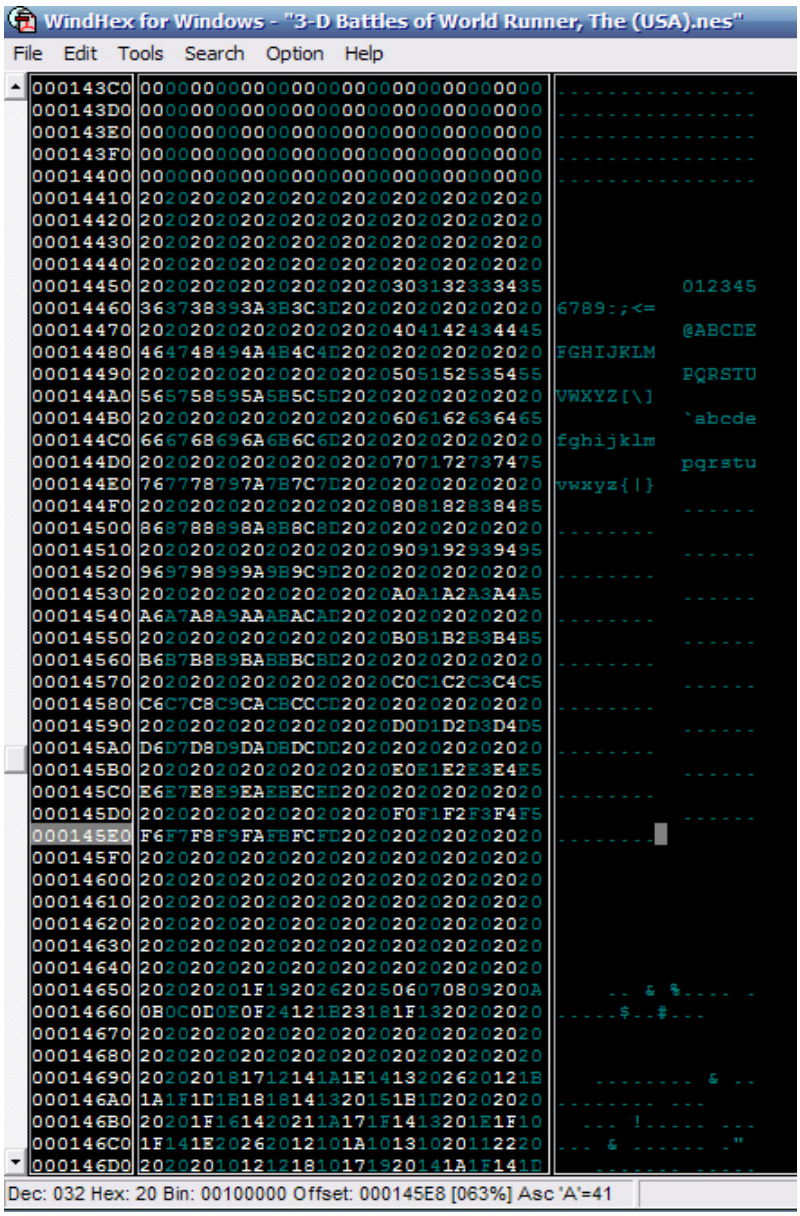
Picture 10 – Title-Screen Located in the Hex Editor

For that, I'll put my custom logo on a random position of the Pattern Table. Well, not so random, taking into account that I'm overwriting only bytes that were used for the old logo. Then, I'll remap the tilemap to show only it, instead of the old logo. See the custom logo (and it's consequent mapping) on the Picture 11. Any similarity is not coincidental. To learn how to add custom logos to your game, read also some articles about graphical editing using Tile Molester (recommended).

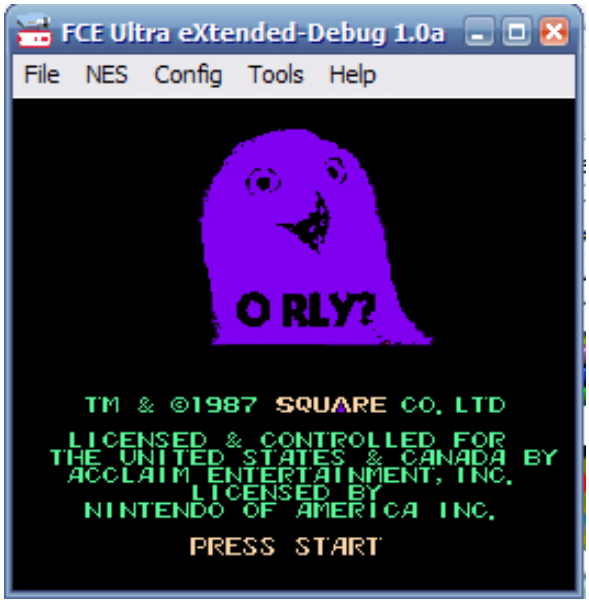
Picture 11 – Custom Logo already inserted into the ROM

If we run the game now, we'll see that the title-screen is all distorted. How we can solve this? We'll use the Pattern Table as reference and edit all the bytes that used to map the old logo. See on Picture 12 how the hex editor will look like after all the tilemaps modification, and finally, the final result in the Picture 13, taken from FCEUXD.

As you noticed, the possibilites of modification are almost infinite. I've chosen one custom logo just as a joke, but using your own custom images, you can make any title-screen you want! Just remember, never get discouraged, just explore your ROM a little more and you'll surely find anything you haven't noticed before!



Picture 12 – Title-screen remap on the hex editor



Picture 13 – Title-screen with it's logo completely modified

Appendix

All the images used on this document were captured by the author, therefore, are under the same license that rules this work. The only exception applies is the cover image, that was taken from the art website http://www.abcgallery.com/C/chirico/chirico.html, respecting it's own Creative Commons license.

Thanks

I must give special thanks to my parents, as without them, I would be nothing. Another person that deserves special recognition is Sandro Dutra aka “Odin”, the one who gave me the necessary info so I could proceed in the study of Tilemaps and NES in general, including 6502 architecture. This document wouldn't exist if he didn't helped me. Also, Israel Crisanto aka “Fallen_Soul” deserver recognition, because he was the one who gave me administrator powers on the RomhackingWiki, what motivated me to write this article about Tilemaps (that later became this document). And a huge thanks to everyone at the Brazilian Unified Translation Boards, aka FUT, that are always eager to help our problems and keep the brazilian romhacking scene always united and strong.

And of course, thanks for you, reader, that had the pacience to read all this document. I hope that it will be useful for you, and please, give me your feedback.

My regards, Fernando Laranja Palhares aka “Knirt”