How Tile Graphics Are Stored In TSB (NES)

By averagetsbplayer, June 12, 2008 in Hacking Documentation

averagetsbplayer
Tecmo Legend
●●●●●●

Members
141
1,381 posts
Location: Madison, WI

Posted June 12, 2008

[i'm assuming that this guide would explain other NES games, but I haven't looked at other games, yet.]

Each tile in TSB is made up of an 8x8 grid for a total of 64 pixels. Each tile is made up of 16 bytes. A tile can be made up of 3 colors (When using a tile editor, we can see that there are four colors available per tile. However, one of the colors is transparent.).

Each of the 8 rows is stored as two bytes on the rom:

Bytes 1 and 9 make Row 1
Bytes 2 and 10 make Row 2
Bytes 3 and 11 make Row 3
Bytes 4 and 12 make Row 4
Bytes 5 and 13 make Row 5
Bytes 6 and 14 make Row 6
Bytes 7 and 15 make Row 7
Bytes 8 and 16 make Row 8

Color 1 is stored in the first 8 bytes, Color 2 is stored in the last 8 bytes (9-16), and Color 3 is stored as part of all 16 bytes.

Each column of the grid is represented by a hex value.

Column 1 is represented by 'x80'
Column 2 is represented by 'x40'
Column 3 is represented by 'x20'
Column 4 is represented by 'x10'
Column 5 is represented by 'x08'
Column 6 is represented by 'x04'
Column 7 is represented by 'x02'
Column 8 is represented by 'x01'

Of course, at this point, nothing has been made clear as to how this works. I'll try to explain.

Let's take this next graphic as an example:

```
1 - Color 1
2 - Color 2
3 - Color 3
```

```
[ 1 |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
```

In this preceding graphic, the top left corner has a '1' in it. This is showing that a pixel of Color 1 is stored in this tile. To determine what is stored in the 16 bytes that make this graphic, we use the preceding assumptions about how the data is stored.
The pixel is stored in row 1, so it is stored in Bytes 1 and 9.
The pixel is using color 1, so it is stored in one of the first 8 Bytes. (This means that Byte 1 will actually hold the data.)
The pixel is in column 1, so it is represented by 'x80'.
The resulting string of bytes is as follows:

```
80 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
```

Using the same rules as above, let's examine these next 4 examples.

```
[   |   |   |   |   |   |   |   ]        [   |   |   |   |   |   |   |
[ 1 |   |   |   |   |   |   |   ]        [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]        [ 1 |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]        [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]        [   |   |   |   |   |   |   |

[   |   |   |   |   |   |   |   ]        [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]        [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]        [   |   |   |   |   |   |   |
```

Here is the hex code that represents the 16 bytes corresponding to each of these tiles. The tile all the way on the left is labeled as '1' while the tile all the way on the right is labeled as '4'.

```
1. 00 80 00 00 00 00 00 00    00 00 00 00 00 00 00 00
2. 00 00 80 00 00 00 00 00    00 00 00 00 00 00 00 00
3. 40 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
4. 20 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
```

Using the same rules as above, let's examine these next 4 examples. Notice, the '1' has been replaced by '2' to signify the use of Color 2.

```
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[ 2 |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [ 2 |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
```

Here is the hex code that represents the 16 bytes corresponding to each of these tiles. The tile all the way on the left is labeled as '1' while the tile all the way on the right is labeled as '4'.

```
1. 00 00 00 00 00 00 00 00    00 80 00 00 00 00 00 00
2. 00 00 00 00 00 00 00 00    00 00 80 00 00 00 00 00
3. 00 00 00 00 00 00 00 00    40 00 00 00 00 00 00 00
4. 00 00 00 00 00 00 00 00    20 00 00 00 00 00 00 00
```

Using the same rules as above, let's examine these next 4 examples.

```
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]       [   |   |   |   |   |   | 2 |
[   |   |   |   |   |   |   | 1 ]       [   |   |   |   |   |   |   |
```

Here is the hex code that represents the 16 bytes corresponding to each of these tiles. The tile all the way on the left is labeled as '1' while the tile all the way on the right is labeled as '4'.

```
1. 00 00 00 00 00 00 00 01    00 00 00 00 00 00 00 00
2. 00 00 00 00 00 00 00 00    00 00 00 00 00 00 02 00
3. 00 00 00 00 08 00 00 00    00 00 00 00 00 00 00 00
4. 00 00 10 00 00 00 00 00    00 00 10 00 00 00 00 00
```

Of course, placing one pixel on the grid is completely useless in almost all cases. Let's move on to handling placing multiple pixels on the grid. Each row of the grid is handled by two bytes (Byte X and Byte X+8. So, Byte 1 and Byte 9, 2 and 10, 3 and 11, etc. are the pairings as listed previously. So, a pixel on Row 2 does not affect a pixel on Row 1. So, take for example the following displays:

```
[ 1 |   |   |   |   |   |   |   ]      [   |   |   |   |   |   |   |
[ 1 |   |   |   |   |   |   |   ]      [ 1 |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]      [ 1 |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]      [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]      [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]      [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]      [   |   |   |   |   |   |   |
[   |   |   |   |   |   |   |   ]      [   |   |   |   |   |   |   |
```

Here is the hex code that represents the 16 bytes corresponding to each of these tiles. The tile all the way on the left is labeled as '1' while the tile all the way on the right is labeled as '4'.

```
1. 80 80 00 00 00 00 00 00    00 00 00 00 00 00 00 00
2. 00 80 80 00 00 00 00 00    00 00 00 00 00 00 00 00
3. 00 40 00 40 00 00 00 00    00 00 00 00 00 00 00 00
4. 00 00 00 00 00 00 00 00    00 00 04 04 00 00 00 00
```

The more complex part of this process occurs when placing multiple pixels on the same row. The hex values associated with the columns that have the pixel are added to determine the hex value that is stored in the byte. So, let's assume we are looking solely at Row 1. Let's also put Color 1 in Column 1 and in Column 2. Since it's Row 1 and Color 1, we know that Byte 1 is going to be the location of the value. However, how do we determine the value? This is where we add the hex values associated with the columns. So, Column 1 and Column 2 are individually represented by 'x80' and 'x40'. If we add these values together, we get 'xC0'. So, Byte 1 is going to have a value of 'xCO' stored in it to represent that Column 1 and Column 2 have a pixel of Color 1 stored. Take the following as an example:

```
[ 1 | 1 |   | 1 |   | 1 |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
```

If we have this preceding situation, we can see that Bytes 1 and 9 could potentially have a value stored since Row 1 contains pixels. Since Color 1 is the only color stored in that row, we know that only Byte 1 will be affected. Adding the values of the columns that have the pixel, we can get the following:
'x80' + 'x40' + 'x10' + 'x04' = 'xD4'
So, the resulting string of bytes will be:

```
D4 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00
```

The same idea will be carried through with Color 2.
Now let's see the example where we are using both Color 1 and Color 2 in a given row:

```
[ 1 | 1 | 2 | 1 | 2 | 1 | 2 | 2 ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
```

So, for Byte 1 we add all the column values associated with Color 1:
'x80' + 'x40' + 'x10' + 'x04' = 'xD4'
So, for Byte 9 (Color 2), we add all the column values associated with Color 2:
'x20' + 'x08' + 'x02' + 'x01' = 'x2B'
The resulting string of hex values would be as follows:

```
D4 00 00 00 00 00 00 00    2B 00 00 00 00 00 00 00
```

If we are using Color 3, we need to show that in both the left byte (Byte 1) and the right Byte (Byte 9).
So, for a simpler example:

```
[   |   | 3 |   | 3 |   | 3 |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
```

Again, we are working with Bytes 1 and 9 since it is Row 1. Since it is Color 3, it is reflected in both bytes.
So, for Byte 1:
'x20' + 'x08' + 'x02' = 'x2A'
So, for Byte 9:
'x20' + 'x08' + 'x02' = 'x2A'
So, the resulting string of hex values will be:

```
2A 00 00 00 00 00 00 00    2A 00 00 00 00 00 00 00
```

The last step is to put all of this together. Let's assume the following:

```
[ 2 | 1 | 3 | 1 | 3 | 2 | 3 | 1 ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
[   |   |   |   |   |   |   |   ]
```

So, Byte 1 is going to add the column values for Color 1 and Color 3.
Byte 9 is going to add the column values for Color 2 and Color 3.
This means the following calculation for Byte 1:
'x40' + 'x20' + 'x10' + 'x08' + 'x02' + 'x01' = 'x7B'
This means the following calculation for Byte 9:
'x80' + 'x20' + 'x08' + 'x04' + 'x02' = 'xAE'
So, the hex string will be:

```
7B 00 00 00 00 00 00 00    AE 00 00 00 00 00 00 00
```

I know there are parts that are not going to be clear. I'll try to make this post more clear when I get chance. Until then, if you have any questions, post them and I'll try to answer them.

+ Quote 1

bruddog reacted to this

Madison Tecmo Tournament - Only 1/2 garbage since 2008
2018: Green Bay - Not Good; Madison - Sweet 16 | 2017: Green Bay - Not Good; Madison - t-5th | 2016: Madison - Sweet 16 | 2015: Green Bay - 2nd Place; Madison - Elite 8, Ohio - Not Good, Iowa - Not Good | 2014: Nebraska - 2nd Place; Madison - Sweet 16; Ryder Cup - Winner (Team Madison); Iowa - Winner | 2013: Nebraska - Elite 8; Madison - Round of 32; Ohio - Sweet 16; Iowa - Final 6 | 2012: Madison - Sweet 16; Ohio - Sweet 16 | 2011: Madison - Round of 32; Ohio - 2-3 in triple-elim | 2010: Madison - Elite 8; Ohio - Sweet 16; Chicago - Final 4 | 2009: Madison - Round of 32; Ohio - Elite 8 | 2008: Madison - Round of 32

Reply to this topic...

RECENTLY BROWSING 1 MEMBER

SBlueman