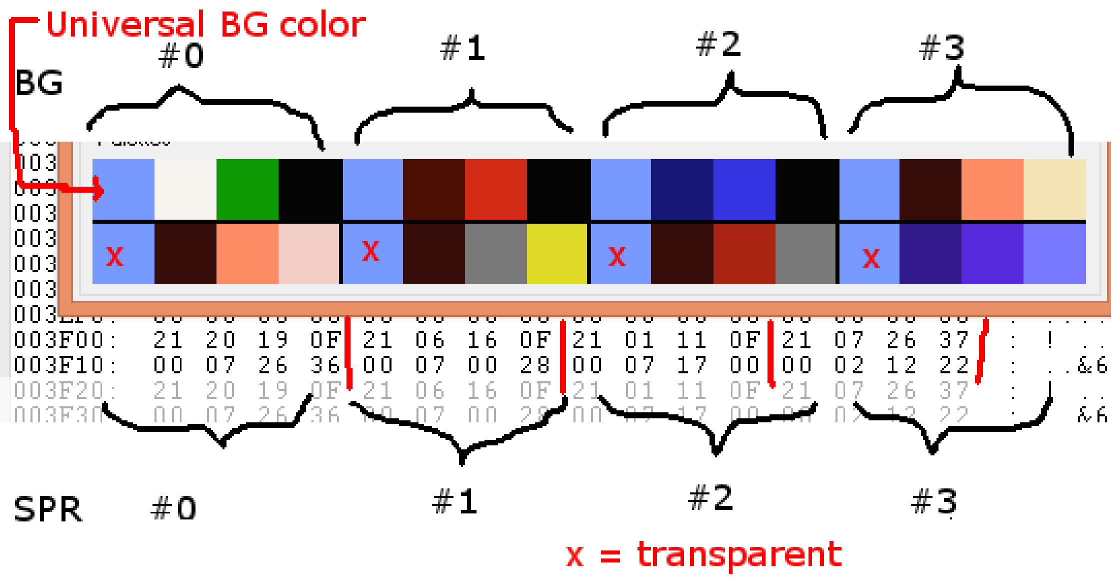
5. A little color

So far everything has been in black and white…let's add some color.

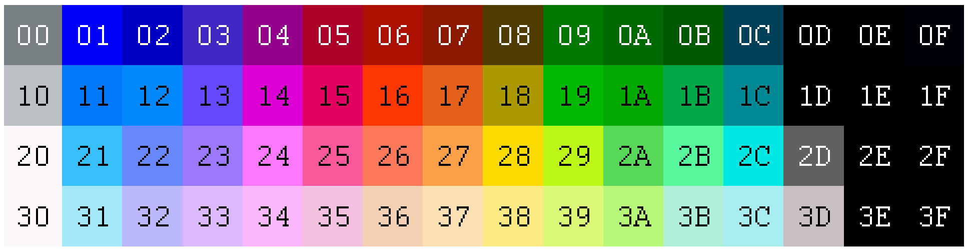
First, a bit about the NES color palette. There are 32 memory addresses reserved for color ($3f00-3f1f of the PPU). 16 for backgrounds ($3f00-3f0f) and 16 for sprites($3f10-3f1f). Divided into 4 sections of 4, and the first color of each is ignored, and is treated as transparent (for backgrounds, transparent = the default background color will show instead). So really each sprite can use 3 unique colors. And each background tile can use 3 unique colors + the default background color.

($3f00 = the default background color, although I could swear that I've changed the background color by writing to $3f10…hmm…the wiki says "Addresses $3F10/$3F14/$3F18/$3F1C are mirrors of $3F00/$3F04/$3F08/$3F0C"…so writing to 3f10 is equivalent to writing to $3f00.)



```
003F00: 21 20 19 0F 21 06 16 0F 21 01 11 0F 21 07 26 37 : !  ..
003F10: 00 07 26 36 00 07 00 28 00 07 17 00 00 02 12 22 : ..&6
003F20: 21 20 19 0F 21 06 16 0F 21 01 11 0F 21 07 26 37 : !  ..
003F30: 00 07 26 36 00 07 00 28 00 07 17 00 00 02 12 22 : ..&6
```

That makes 25 colors available on the screen at the same time, out of the 50ish possible colors to choose from.



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F

Notes: You can darken/tint the colors by setting various "color emphasis" register bits, but I tend to avoid that. And, a slight warning, don't use color $0d, it will cause problems with some tvs, just use the f colors for black.

Backgrounds are further limited, because you can only define a palette for a 16×16 section. (even though tiles are actually 8×8).The Nes Screen Tool is very good for seeing the attribute table limitations. Most games just build their backgrounds in 16×16 blocks (metatile = a 2×2 block of tiles). With a little imagination, we can make some backgrounds that aren't 100% square blocks.

Each background screen (nametable) has 64 bytes devoted to "attributes" …ie which palette to use. Nametable #0 = ppu addresses $2000-23ff. The attribute table is $23c0-23ff. Each byte in the attribute table covers a 32 pixel x 32 pixel area, and is divided into 4 parts. Each 2 bits chooses a color palette for a 16×16 part of the screen. It goes like this (bits vs screen position)

44332211 =

1, 2

3, 4

So, for example, if you want to change the bottom right area (a 16 x 16 block of background) to palette #1…you would change the high 2 bits of its attribute byte to 01. Such as 01xx xxx. If you wanted to change the top left area to palette #3, you would change the low 2 bits to 11. Such as xxxx xx11.

I made a few changes to the code from last time…for one, I changed the palette to be more colorful, and I added a few attribute table bits, to show the size of the smallest unit you can define (16×16). I also put those new bits in a separate file and used an #include, and put the variables in the zeropage with a #pragma (just to show how that can be done). Zeropage variables work faster than non-zeropage variables. (Note, cc65 will also use about 10-20 zeropage variables for various C library functions — leaving you 235 or so to use.)

```
#pragma bss-name(push, "ZEROPAGE")
blah variables here
```

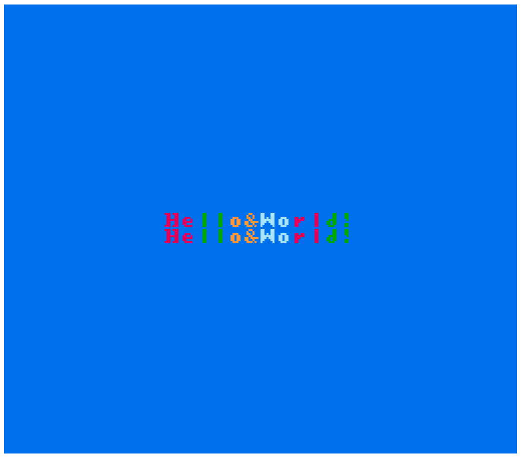
```
#include "PALETTE.c"
#include "CODE.c"
```

The code looks cleaner if you separate various parts.

```
const unsigned char PALETTE[]={
0x11, 0x00, 0x00, 0x31, // blues
0x00, 0x00, 0x00, 0x15, // red
0x00, 0x00, 0x00, 0x27, // yellow
0x00, 0x00, 0x00, 0x1a, // green
};
// note, 11 is the default background color = blue
// we are only writing the BG palettes (16 bytes total).
```

```
const unsigned char Attrib_Table[]={
0x44, // 0100 0100,
0xbb, // 1011 1011,
0x44, // 0100 0100,
0xbb}; // 1011 1011 };
// this is attribute table data,
// each 2 bits defines a color palette
// for a 16x16 box
```

```
 PPU_ADDRESS = 0x23;
 PPU_ADDRESS = 0xda;
 for( index = 0; index < sizeof(Attrib_Table); ++index ){
  PPU_DATA = Attrib_Table[index];
  } // this inserts the Attrib Table data into the PPU
```

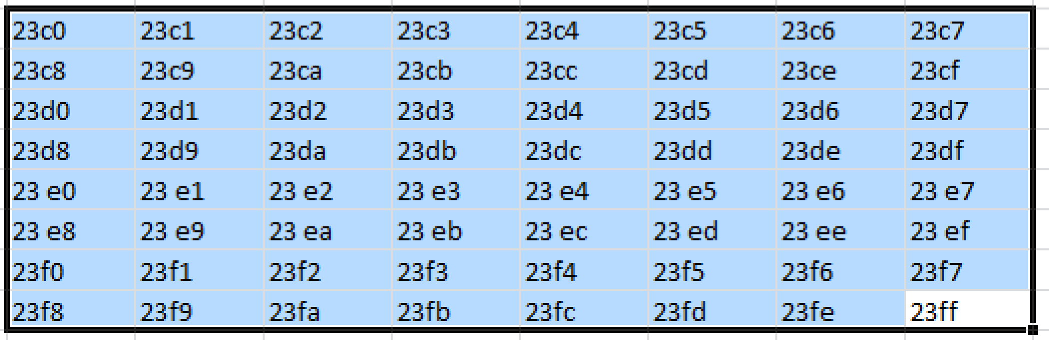


Here's the link to this code…

http://dl.dropboxusercontent.com/s/tqp6s3odgurieep/lesson3.zip (http://dl.dropboxusercontent.com/s/tqp6s3odgurieep/lesson3.zip)

Our next project will show how to draw sprites to the screen. See you next time.

If you're still confused about attribute tables, here's a grid of the entire attribute table and its position on the screen.



23c0	23c1	23c2	23c3	23c4	23c5	23c6	23c7
23c8	23c9	23ca	23cb	23cc	23cd	23ce	23cf
23d0	23d1	23d2	23d3	23d4	23d5	23d6	23d7
23d8	23d9	23da	23db	23dc	23dd	23de	23df
23 e0	23 e1	23 e2	23 e3	23 e4	23 e5	23 e6	23 e7
23 e8	23 e9	23 ea	23 eb	23 ec	23 ed	23 ee	23 ef
23f0	23f1	23f2	23f3	23f4	23f5	23f6	23f7
23f8	23f9	23fa	23fb	23fc	23fd	23fe	23ff

Since the screen is 256 pixels wide, each box represents a 32×32 block of the screen. Each byte is then further divided into 2 bit pieces representing 16×16 blocks of the screen, defining which palette that block gets.

November 19, 2015April 15, 2017 dougfraker

3 thoughts on "5. A little color"

1.
plin says:
February 9, 2017 at 1:26 am Edit
Hey thanks a lot for these, i prefer C over ASM so these tutorials have been very helpful! Ive been reading a bit about pattern tables, nametables, and attribute tables to really understand this lesson and also playing a bit with the code so my (really newbie) question is… is it necessary or useful to know how to modify the data in the pattern tables directly in the code? I understand you can use NES Screen Tool to make and edit tiles in pattern tables instead but is there any advantage in doing all of that in the code (which by the way seems a lot more difficult).

Reply

dougfraker says:
February 9, 2017 at 4:34 pm Edit
If your background changes, for example Mario breaks a brick and it disappears, you will need to know how to locate those 4 tiles and that 1 attribute byte. And, it is very complicated, yes.

Reply

2.
Dan says:
June 9, 2017 at 10:47 pm Edit
Here's a weird thing that I noticed, which is probably my own misunderstanding of something :), but I am curious if you've seen this before.

After I got the initial code working I was playing around, trying to manipulate the color palette, and none of the colors were changing no matter what I did. Finally, I commented out the palette loading function and built the rom which gave me a blank gray screen as expected. When I uncommented the function and rebuilt the rom again my color changes had taken effect.

Reply

Create a free website or blog at WordPress.com.