nesdoug

11. Scrolling

The $2005 register controls the bg scrolling. The first write for horizontal and the second write for vertical. Writes to $2005 flip-flops back and forth x, y, x ,y. Reading from $2002 will reset the flip-flop back to x if you're not sure where you're at.

99% of NES games only have enough PPU memory to fill 2 screens (nametables). There are addresses for 4, so 2 will be mirrors of the other 2. For emulated games, we have to describe the cart layout in the iNES header. Bytes 6-7 describe the mapper. The low bit of byte 6 describes the mirroring. 0 = horizontal mirroring = scrolling up and down. 1 = vertical mirroring = scrolling right and left.

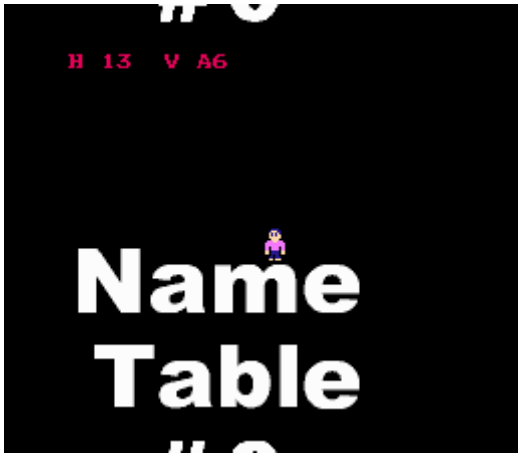
Gauntlet is special, it has 4 screen scrolling. That required an extra 2k RAM chip on the cartridge. MMC3 games can switch mirroring from horizontal to vertical. But, most games only have one mirroring mode, and most have only 2 nametables to use.

In this first example, it will be setup with vertical mirroring (Note the startup code reset.s sets the mirroring to vertical in the header). Moving the U/D/L/R on the controller will scroll the background around. The red letters are the H scroll value and the V scroll value. (I had to make them sprites so they wouldn't move with the background.) I highly recommend using FCEUX and having the Nametable Viewer on while moving around.

It looks like this…

Nametable #0 // Nametable #1

Nametable #0 // Nametable #1



As soon as you cross over above 255 (ff) on the H scroll, I have it switch the Nametable assigned to the $2000 register (PPU_CTRL). At that point you can continue scrolling right another 255.

My tool chain here was: make letters in Photoshop, index to 4 colors, cut and paste into YY-CHR, save. Open chr with NES Screen Tool. Draw the backgrounds. Save as .rle (compressed, as a c header file). Include them in the C code, and load them at the start. Movement now is changing the scroll position (rather than moving the sprites around).

```
void move_logic(void) {
 if ((joypad1 & RIGHT) != 0){
   state = Going_Right;
   ++Horiz_scroll;
   if (Horiz_scroll == 0)
      ++Nametable;
 }
 if ((joypad1 & LEFT) != 0){
   state = Going_Left;
   --Horiz_scroll;
   if (Horiz_scroll == 0xff)
      ++Nametable;
 }
 Nametable = Nametable & 1; // keep it 1 or 0
```

```
if ((joypad1 & DOWN) != 0){
   state = Going_Down;
   ++Vert_scroll;
   if (Vert_scroll == 0xf0)
      Vert_scroll = 0;
 }
 if ((joypad1 & UP) != 0){
   state = Going_Up;
   --Vert_scroll;
   if (Vert_scroll == 0xff)
      Vert_scroll = 0xef;
  }
 }
```

And, here is how my ‘every_frame’ function looks…

```
void every_frame(void) {
OAM_ADDRESS = 0;
OAM_DMA = 2; // push all the sprite data from the ram at 200-2ff to the sprite memory
PPU_CTRL = (0x90 + Nametable); // screen is on, NMI on
PPU_MASK = 0x1e;
SCROLL = Horiz_scroll;
```

```
SCROLL = Vert_scroll;  // setting the new scroll position
Get_Input();
}
```

————————————————————-

In this second example, it is set up for horizontal mirroring. (Note the startup code reset.s sets the mirroring to horizontal in the header) Now it looks like this…

Nametable #0 // Nametable #0

Nametable #2 // Nametable #2

You might notice the max V scroll is $ef. The screen is only 240 pixels high, so I have it jump from ef (239) to zero in the next move.

The only difference in the code is how the nametable is adjusted to write to the $2000 (PPU_CTRL) register. Rather than flopping between 0-1, it flops 0 or 2…

```
PPU_CTRL = (0x90 + (Nametable << 1));
```

Here's the source code…

http://dl.dropboxusercontent.com/s/g5vnwnzn7q1pa9j/lesson9.zip (http://dl.dropboxusercontent.com/s/g5vnwnzn7q1pa9j/lesson9.zip)

November 29, 2015April 15, 2017 dougfraker