nesdoug

8.Sprite Collisions

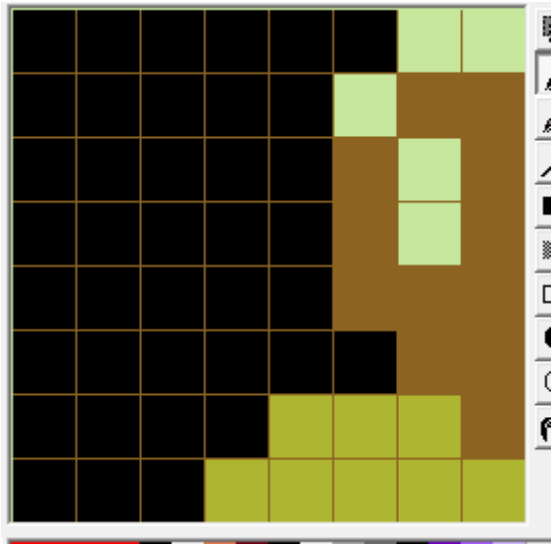
The easiest kind of collision detection is sprite vs. sprite. In this case I will be doing metasprites of 16×16 pixels. That seems to be a standard NES size.

"Are 2 things touching?" is a process comparisons. Is A(left side) less than the right of B(right side)? Is A(right side) more than B(left side)? Is A(top) less than B(bottom)? Is A(bottom) more than B(top)?

If all of these are true, then object A is touching object B. I prefer to define sprite positions by the top, left pixel of the top, left sprite. All other sides are offsets from that position. In my example it looks like this…

```
A_left_side_X = A_X + 3;
A_right_side_X = A_X + 12;
A_top_Y = A_Y;
A_bottom_Y = A_Y + 15;
//(etc for B)
if (A_left_side_X <= B_right_side_X &&
   A_right_side_X >= B_left_side_X &&
   A_top_Y <= B_bottom_Y && A_bottom_Y >= B_top_Y){do something}
```

Why is the left side X + 3? Because the left 3 pixels of our Sprite are blank…



However, because we are always using unsigned char variables, things > 255 will roll over to 0, making problems at the edges (when you walk off the edge of the screen). Here's a little hack to fix it. (so that if A_X = 250, + 12 won't = 6 (absurd) but 255 (wrong, but still more reasonable). It seems to work.

I suppose we could have made these variables 16-bit (int), but I think that would slow down our collision code (speed is especially important when we're checking collisions between 20+ objects on the screen). Or, another option would be to make sure sprites never go off the edge of the screen.

```
A_left_side_X = A_X + 3;
if (A_left_side_X < A_X) A_left_side_X = 255; //if overflow, set to max high
```

In our next example object B is auto-moved (with previously used code), and controller moves A. Every frame they touch, our scoreboard will go up by 1. Here's the code that adjusts the scoreboard (score5 = ones digit)…

```
if (score5 > 9){
 ++score4;
 score5 = 0;
}
if (score4 > 9){
 ++score3;
 score4 = 0;
}
if (score3 > 9){
 ++score2;
 score3 = 0;
}
if (score2 > 9){
 ++score1;
 score2 = 0;
}
if (score1 > 9){ //if overflow, rolls back to 0
 score1 = 0;
 score2 = 0;
 score3 = 0;
 score4 = 0;
 score5 = 0;
}
```

And, everytime the score is changed, it sets a flag to do this at the beginning of the next V-blank, to actually draw those number tiles onto the screen. We absolutely must only write to the PPU during V-blank, that's why we wait till the beginning of the next V-blank. Remember from before, that we have NMI's turned 'on' to know exactly when the V-blank has begun.

```
void PPU_Update (void) {
 PPU_ADDRESS = 0x20;
 PPU_ADDRESS = 0x8c;
 PPU_DATA = score1+1; //I made tile 0 = blank, tile 1 = "0", tile 2 = "1", etc
 PPU_DATA = score2+1; //so we have to add 1 to the digit to get the corresponding
 PPU_DATA = score3+1;
 PPU_DATA = score4+1;
 PPU_DATA = score5+1;
}
```

Here's the link…

http://dl.dropboxusercontent.com/s/dps1glbmy04onxx/lesson6.zip (http://dl.dropboxusercontent.com/s/dps1glbmy04onxx/lesson6.zip)

November 28, 2015April 15, 2017 dougfraker