nesdoug

Sprite Collision, and Controllers

OK, I kind of copied the idea from Shiru's example code. Controller 1 controls the yellow sprite. Controller 2 controls the blue sprite. The background color changes when a collision is detected.

CONTROLLER READS. I changed this. Normally, with neslib, you have to pass the controller read to a variable, and do a seperate function to get new button presses (trigger mode). I feel it would save zero-page RAM if you could access those internal variables directly.

So normally you would…

pad1 = pad_poll(0); // reads controller 1

I changed it so you do this…

pad_poll(0); // reads controller 1

…then to access the read value, you use the PAD_STATE variable. And to use the new button pressed value, you use the PAD_STATET variable. Example.

if(PAD_STATE & PAD_LEFT){ }

would do something if LEFT is pressed on controller 1.

if(PAD_STATET & PAD_LEFT){ }

would do something if LEFT is pressed this frame, but not the last frame. A new press.

You should read the controller at the beginning of each frame. (Or not, if your game logic takes 2 frames to complete, and you want a consistent controller value across both frames.).

For 2 player, you need to make sure you read the 2nd controller.

pad_poll(1); // reads controller 2

For the sprite collision, I wrote some ASM code, that expects 2 pointers to structs who's first 4 bytes are ordered like this…

struct BoxGuy {

unsigned char X;

unsigned char Y;

unsigned char width;

unsigned char height;};

It returns 0 if no collision, and 1 if collision. It's slightly buggy at the edges of the screen. This is the function…

unsigned char CheckCollision(struct BoxGuy* one, struct BoxGuy* two);

NOTE – it could have been written in C. Something like this…

```
unsigned char CheckCollision (struct BoxGuy* one, struct BoxGuy* two){
  if (((one->X + one->width) > two->X) &&
  ((two->X + two->width) > one->X) &&
  ((one->Y + one->height) > two->Y) &&
  ((two->Y + two->height) > one->Y)){
     return 1;
  }
  else {
     return 0;
  }
}
```

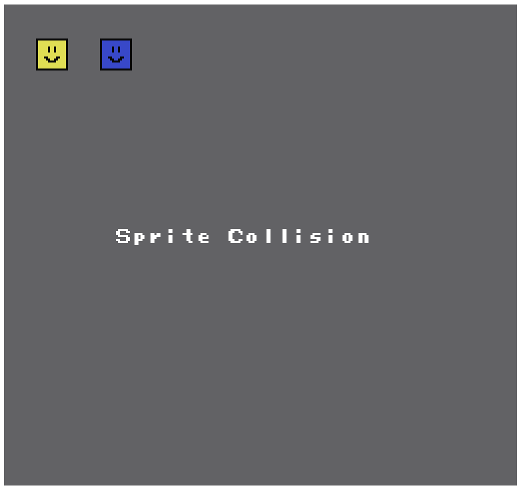
But, I tested that, and it runs twice as slow as the ASM version (478 cycles vs. 255 cycles). I'm assuming that a much more complex game will need to do lots of sprite collision checks. Hmm. Maybe I should make this even more efficient?

So, anyway, if collision == true, change the background color, else, change it back. You can change 1 color with this function.

pal_col(unsigned char index,unsigned char color);

pal_col(0, 0x30); // 0 is the first color in the palette array, 0x30 is white.

Here's the code.



http://dl.dropboxusercontent.com/s/qdkz26l9n3rpx6y/lesson25.zip
(http://dl.dropboxusercontent.com/s/qdkz26l9n3rpx6y/lesson25.zip)

Interesting side note. Notice how the yellow box is always in front of the blue box. That is because the yellow box was loaded first into the buffer. It has a higher priority. If the blue box was loaded first, it would be on top.

August 9, 2017August 10, 2017 dougfraker