nesdoug

Introduction

Hello everyone.

I'm Doug (dougeff). I've been programming games for play on the NES since 2015, welcome to my blog. I intend to write a how-to make your own NES game tutorial, mostly because I want to encourage more people to make NES games.

My blog will be unique, because I will explain how to write it entirely in C, so the majority of programmers will be able to make a game quickly and without having to learn 6502 Assembly. As far as I know, there are no other tutorials for cc65 (not counting the example games over at Shiru's site.)

Keep in mind — I'm an idiot. With no degree in Computer Science, and I've never written a blog before. For more detailed information, I suggest going to the wiki,

http://wiki.nesdev.com/ (http://wiki.nesdev.com/)

(More information about the NES, not about my idiocy, for that you'd have to visit the forum) 😉

I will try to make the learning process as easy as possible, using the simplest examples I can think of. I recommend, though, that you also start with the easiest possible game ideas you can do. Everyone starts off thinking he's going to reinvent Zelda. Don't. The smallest games take 2-3 months to finish. Zelda would take 2-3 years…just enough time to get bored or distracted or busy with more important things, and your Zelda never gets completed. Stick with the Pacman sized games, at least for the first few projects.

(Just so you know… I will sometimes use $ to indicate hexadecimal numbers, and sometimes 0x.)

———————————————

Before we start anything, I think we should talk about the memory map of the NES system.

The NES has 2 separate memory maps. The CPU references memory from 0-$ffff (65535 in decimal).

The first $800 is the internal RAM

$6000-7fff some games use this for SRAM (to save the game) or extra Work RAM.

$8000-ffff is where the game ROM is mapped. More advanced mappers can use more than 32k ROM, but it is generally mapped to the $8000-ffff area of the CPU map.

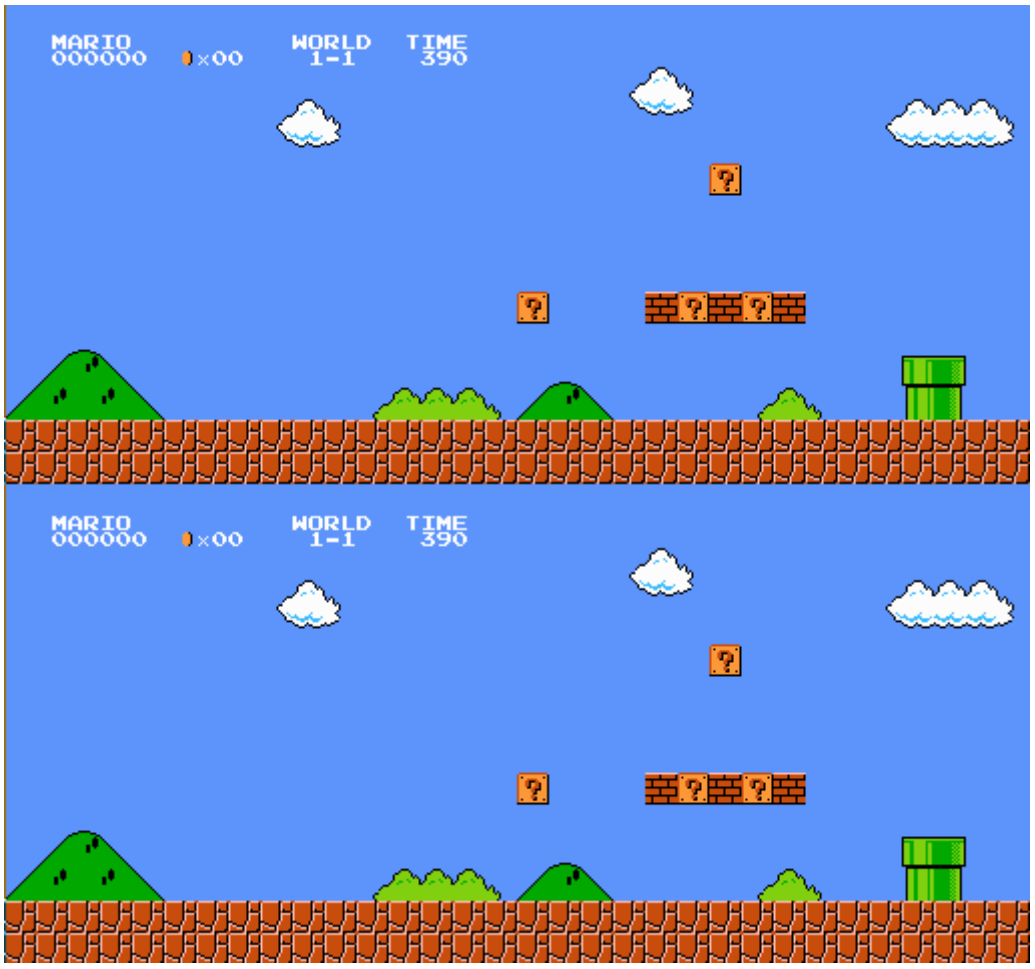
We need to make sure that addresses $fffc-d point to the start of the program.

Here's more information on the CPU map…

http://wiki.nesdev.com/w/index.php/CPU_memory_map
(http://wiki.nesdev.com/w/index.php/CPU_memory_map)

And the other memory map is for the PPU, the chip that draws pictures on the tv screen.

It has addresses from 0-$3fff, but many of these are mirrors. It looks like there are 4 screens worth of addresses, but only half can be used (for 99.9% of games). If the game is set for horizontal scrolling (vertical mirroring) its map looks like this…



0-$1fff = where graphic tiles are stored

$2000-23ff = nametable 0

$2400-27ff = nametable 1

$2800-2bff = nametable 2

$2c00-2fff = nametable 3

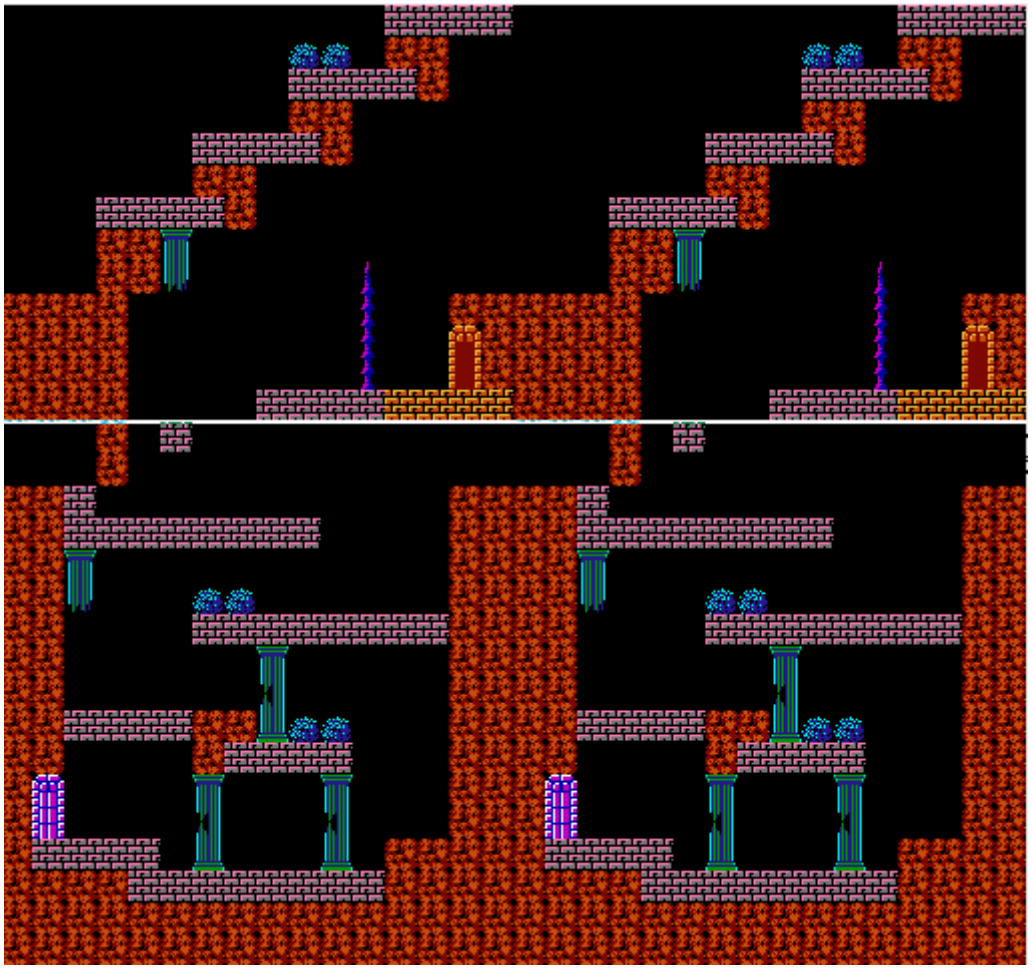
(but nametable 2 & 3 are only mirrors of 0 & 1)

$3f00-3f1f = color palette

(nametable = RAM addresses that reference a location on the screen. Essentially, a map of how the graphic tiles are arranged on the screen.)

Scrolling to the right will show part of nametable 0 and part of nametable 1, but scrolling down will just show a copy of nametable 0 below nametable 0.

And, if it's set for vertical scrolling (horizontal mirroring) it looks like this (with the right nametables only a copy of the left nametables)…



Scrolling to the right will show a copy of nametable 0 to the right of nametable 0, and scrolling down will show part of nametable 0 and part of nametable 2. You only can see 1/4 of this at any one time on the TV.

And the PPU has a seperate OAM memory (which holds data for Sprites) and has its own 256 bytes of memory (0-$ff). These will control how the movable characters appear on the screen.

Here's more information on the PPU map…

http://wiki.nesdev.com/w/index.php/PPU_memory_map
(http://wiki.nesdev.com/w/index.php/PPU_memory_map)

————————————————

Another thing you should know. There are 2 types of NES carts. Some have 2 ROM chips. 1 for PRG-ROM (executable code), and 1 for CHR-ROM (graphics). This type automatically maps the graphics to the PPU memory at 0-1fff. Making an image appear is as simple as writing a tile number on a nametable. I will exclusively be using the CHR-ROM style.

The other kind of carts have a PRG-ROM chip and instead of a CHR-ROM chip, have a $2000 byte CHR-RAM chip. The graphics are located somewhere in the PRG-ROM, and the program has to load them into the CHR-RAM (by writing to 0-1fff of the PPU) before they can be accessed. I won't be using any CHR-RAM examples.

Hope that's not too confusing. Now, let's make some games!

November 15, 2015March 21, 2018 dougfraker

2 thoughts on "Introduction"

1. **Abdul** says:
   February 9, 2016 at 11:39 pm Edit
   Thank you so much for this awesome resource, Doug

   Reply

2. **lolkiu64** says:
   March 4, 2018 at 9:08 am Edit
   I can't wait to get started

   Reply