{RIVIERADEV}

NES programming

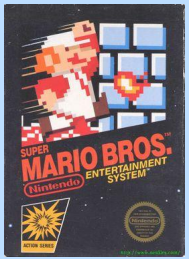
with neslib

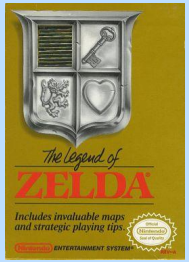
The NES

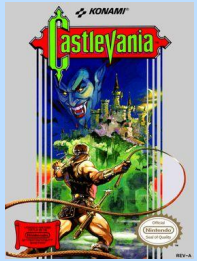
Released in 1983 in Japan (Famicom), discontinued in **2003**
Named “Nintendo Entertainment System” in USA (1985) and EUR (1987)

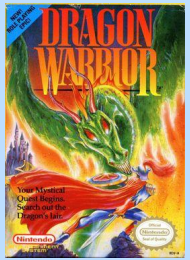
61 million units sold, 716 licensed games

Debuted some of the most famous franchises of all time.









Not you

Hardware

Processor: 8bits @1,79MHz (NTSC version)
RAM: 2kb
Video RAM: 2kb

ROM: 32kb for the program, 8kb for the graphics

Resolution: 256x224 (NTSC version)
52 colors palette
64 movable sprites (8x8 pixels, 3 colors each + transparency)
8 sprites per line max

Homebrew

New NES games are still being released in 2019

Twin Dragons (brokestudio.fr)

- 3 people (incl. 1 programmer)
- Coded in ASM
- 2 man-years worth of work
- Uses a special mapper to expand ROM size (switch banks between levels)

Goals

Code **simple** games without too much hassle:

- No assembler required
- No complicated stuff, just learn the bare minimum required about the machine
- Plain C code

We'll use a **subset of C**, and all the bad practices you dream to use at work

- No malloc()
- No function pointers
- Tons of global variables
- Macros
- Hardcoded values everywhere

3rd parties

CC65 - Compiler & linker

- git clone https://github.com/cc65/cc65.git
- make
- make install

YY-CHR - Sprite & palette editor

- http://www.geocities.jp/yy_6502/yychr/yy-chr20120407_en.zip
- It’s a windows exe, but works fine with Wine

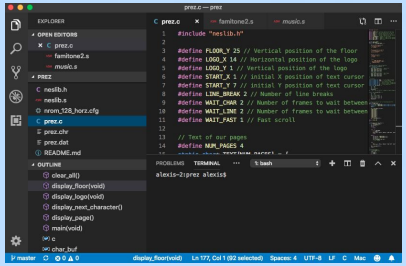
FCEUX - NES emulator

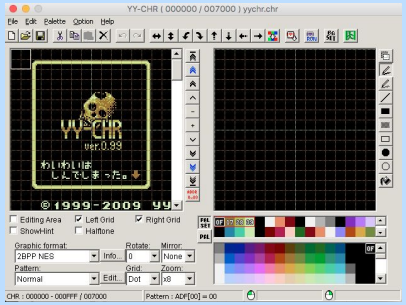
- http://www.fceux.com/web/download.html

Toolchain

RIVIERADEV

The hard stuff we don't want to code



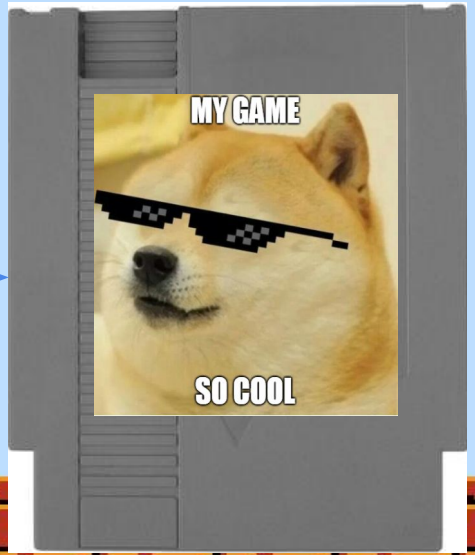


game.c

game.chr

neslib
famitone
cfg

cc65
ca65
ld65



Overview

- Initialization
  - Init display
  - Select palettes
  - Select sprite banks
  - Draw main background
- Main loop (one per frame)
  - Read controller state
  - Compute next state
  - Update background
  - Update sprites
  - Wait for end of frame

Colors

Palette has 64 colors... does it?
Only 52 are usable because of duplicates.
Some shades are really close to each other, almost indistinguishable.
Not a great palette, but it’s all we have!

Palettes

Each palette has 4 colors
But the first color is the same (background color, transparent)

Our program can use a maximum of 8 palettes:

- 4 for the background
- 4 for the foreground (sprites)

```
char *palettes = { 0x00, 0x01, ... }; // 32 bytes
pal_all(palettes);
```

Sprite data

256 sprites per bank
Each sprite is 8x8 pixels

2 sprite banks available:

- One for the background
- One for the foreground (sprites)
- The same bank may be used for both, why not...

```
bank_bg(0);
bank_sprite(1);
```

Background

Background data is stored in Video RAM (nametable)

- Sprite ID (0-255) for each 1x1 tile
- Palette info (0-3) for each 2x2 metatile (last 64 bytes)

neslib defines two constants `NAMETABLE_A` and `NAMETABLE_B`

3 methods:

- `vram_adr()` + `vram_write()`
- `vram_adr()` + `vram_fill()`
- `set_vram_update()` (easy way to do batch update)

Background

Background palettes are stored in the attribute table.
Last 64 bytes of NAMETABLE_A.

Full description here:
http://wiki.nesdev.com/w/index.php/PPU_attribute_tables

Each byte contains 4 x 2 bits. Use bitwise operators.

Example: selecting palette #1 for the bottom rows of the screen:

```
vram_adr(NAMETABLE_B-16);
vram_fill((1 << 6) | (1 << 4) | (1 << 2) | (1 << 0), 16);
```

Background

```
static unsigned char list[10] = {
    // Each frame we can update these 3 background sprites
    MSB(NTADR_A(2,2)),LSB(NTADR_A(2,2)),0,
    MSB(NTADR_A(3,2)),LSB(NTADR_A(3,2)),0,
    MSB(NTADR_A(4,2)),LSB(NTADR_A(4,2)),0,
    NT_UPD_EOF
};

set_vram_update(list);
```

Foreground

Each frame we need to define / update the position of sprites.

For the lazy: at each frame, start the sprite counter at 0.

For each sprite:

- `counter = oam_spr(x, y, sprite, palette | attr, counter)`
- Attributes (bit mask) `OAM_FLIP_V`, `OAM_FLIP_H`, `OAM_BEHIND`

After the last sprite is defined, make sure no leftover from previous frame

- `oam_rest(counter)`

Controls

8 inputs for each controller:

- 4 directions (up, down, left, right)
- Buttons A, B, Select, Start

Two ways to read: poll or trigger (to only capture the moment button is pressed)

```
state = pad_poll(0);
if (state & PAD_B)
  keep_running();
```

```
trig = pad_trigger(0)
if (trig & PAD_A)
  jump();
```

Random

If you need random numbers, neslib provides 2 functions ready to use.

One random number per frame is plenty enough.

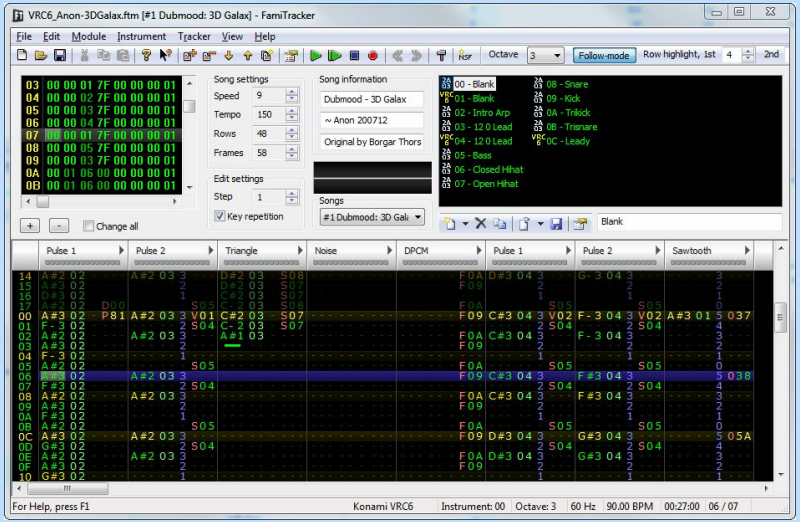
```
char r8 = rand8();
int r16 = rand16();
```

Sound

The NES has 5 audio channels:

- Square wave 1&2
- Triangle wave
- Noise
- PCM (samples) (what really?) (yes)

Famitracker: http://famitracker.com



Let’s code

Text slides and the RivieraDev logo:

```
git clone https://github.com/alexisbietti/rivieradev-nes-prez.git
```

Silly little game Toutou:

```
git clone https://github.com/alexisbietti/nes-toutou.git
```

Resources

http://wiki.nesdev.com

https://github.com/clbr/neslib

Thanks to shiru for neslib