

# Chocolate Chip

A well-documented, informative, and simple CHIP-8 Emulator

Clay Buxton

**Repository** <https://github.com/clbx/ChocolateChip>

## I. ABSTRACT

CHIP-8 is an interpreted machine language built in the 1970's for 8-bit micro computers. Today the only way to run a CHIP-8 program is through emulation. The goal of this project was to accurately emulate the CHIP-8 architecture. The CHIP-8 emulator is generally regarded as a simple emulation project and a good first step into the world of emulation. Chocolate Chip was designed to be easily readable, documented, and informative emulator, in addition to properly running a long lost architecture.

## II. INTRODUCTION

ChocolateChip is written in C++ and primarily focused on giving the user the entire picture of what is happening inside the machine opposed to just simply emulating it. The CHIP-8 architecture is much simpler to emulate than many other architectures due to it's small amount of opcodes and easy to understand. Chocolate Chip has a status screen that displays the last few opcodes executed with a line reading out what they do and their effect on the system, the status of the registers, the position in memory being pointed to, and the status of the stack. This is all thoroughly documented and tested in code as well created a stable and informative emulator

## III. INSTALLATION

### Requirements

- SFML For graphics
- fmt For properly formatting the logger
- gTest if you want to run the unit tests

### Installation

- Install the dependencies for your system
- Clone the repository  

```
git clone https://github.com/clbx/ChocolateChip
```
- Run `make`

## IV. REFERENCES

- Cowgod's Chip 8 Reference  
<http://devernay.free.fr/hacks/chip8/C8TECH10.HTM>
- Multigesture How to build an emulator  
<http://www.multigesture.net/articles/how-to-write-an-emulator-chip-8-interpreter/>
- Wikipedia's CHIP-8 Page  
<https://en.wikipedia.org/wiki/CHIP-8>

## V. METHODOLOGY

Chocolate Chip is divided into 3 classes, the Emulator, the Application and the Logger. The application portion displays the graphics and receives input from the user. The emulator takes an instruction from the program file, and the logger records current status for the user.

The Application portion shows a window of the currently running program and process user input (the 0-9 and A-F "keys")



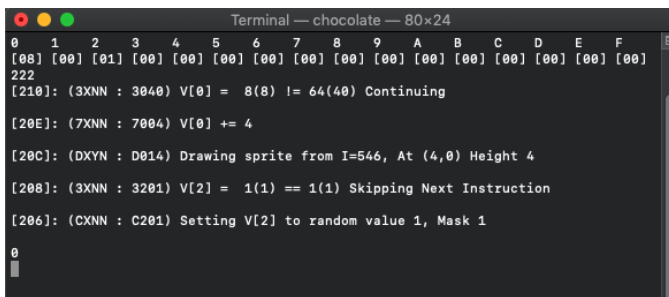
Fig. 1. The application window shown running the maze generator

This is done using the SFML graphics library, on every tick of the emulator the application redraws the screen based on the status of a pixel buffer. However this will eventually be improved in the future since only 2 opcodes actually affect graphics and redrawing on every tick is inefficient

The emulator takes and processes an instruction. In the CHIP-8 architecture, the code memory is put

in the same place as the usable memory, so reading a program consists of loading a ROM into memory and executing opcodes found at the first memory address and then moving on. The emulator then processes each opcode and moves on to the next instruction

The logger outputs information to the user about the last few executed opcodes and the current status of the registers, address pointer, and program counter. This was a feature I wanted to focus on this project so anyone can see what the emulator is doing at any time. This gives a simple way to peering into what is happening behind the scenes



```

Terminal — chocolate — 80x24
0 1 2 3 4 5 6 7 8 9 A B C D E F
[00] [00] [01] [00] [00] [00] [00] [00] [00] [00] [00] [00] [00] [00] [00] [00]
222
[210]: (3XNN : 3040) V[0] = 8(8) != 64(40) Continuing
[20E]: (7XNN : 7004) V[0] += 4
[20C]: (DXYN : D014) Drawing sprite from I=546, At (4,0) Height 4
[208]: (3XNN : 3201) V[2] = 1(1) == 1(1) Skipping Next Instruction
[206]: (CXNN : C201) Setting V[2] to random value 1, Mask 1
0

```

Fig. 2. The logger window that shows information about the emulator

## VI. RESULTS

As of right now, April 14, 2019, I have 24 opcodes implemented, including graphics. This is enough to run some basic programs, but the project won't be considered complete until they all are. The logger is functional, but isn't well organized or show anything about the stack. This is planned to be included. Many of the functions do not have any kind of logging and more are undocumented. There are also currently no unit tests written. These are all features that are planned to be implemented soon.