Individual Project: Chip-8 Interpreter

Thomas Tardi-Zuch | <https://github.com/tommytz/chip8>

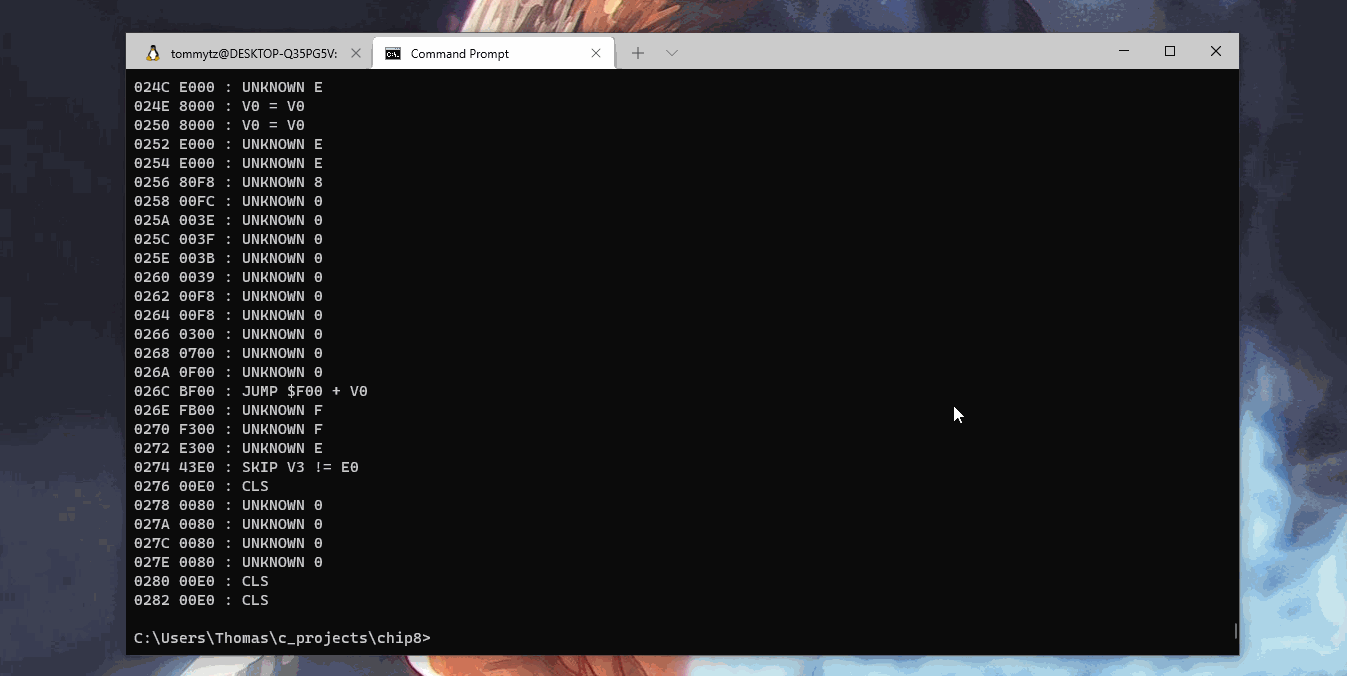
Chip-8 is an interpreted programming language used to program video games for old 8-bit microcomputer like the COSMAC VIP and the Telmac 1800. It is not a true emulator project, but still teaches many of the fundamentals required to move on to the emulation of real systems, such as the Intel 8080 for an arcade cabinet, or the NES console.

My objectives were to learn the basics of computer architecture, the C-programming language, and to develop an application that could run Chip-8 game files as a virtual machine.

# Tech stack and software tools

* Linux Mint environment, Gnu C Compiler (gcc), Gnu Debugger, Atom Text Editor
* Valgrind (Memory leak detection), Git CLI for version control
* MinGW-w64 for Windows compatible compilation.
* Simple Direct Media Layer (SDL/SDL2) for graphics rendering and I/O

# How the application was built

* A simple application called a disassembler was built to read Chip-8 files and develop my familiarity with the 35 operation codes (opcodes) in the Chip-8 instruction set. The Chip-8 files is read into a buffer and then the opcodes are fetched and decoded, printing out their function to the terminal. Bit shifting and bit-masking and bitwise operations are used to decode the opcode instructions. This feature would be reused in the full emulator.
* The state of the Chip-8 virtual machine was represented by a C struct called Chip8State that contains the main memory, registers, and graphics memory. A stack, a key-state array, and the delay and sound timers were also contained.
* Functions were developed for each of the 35 opcodes. The functions were fetched and executed through a switch-statement using the instruction type as the control expression. Each of the functions would operate on the state of the virtual machine (for example, adding the value of one register to another).
* Graphics were rendered to the screen using the SDL2 library. An SDL renderer and SDL texture were created, and the texture used the virtual machine’s graphics memory as a bitmap for the pixels to be rendered to the screen (with only two values, black and white) on an SDL window. SDL was also used for the key-scancodes to register input for the 16 control keys.

# How to run the application

Instructions on compiling and running the application are in the GitHub repo inside “Compiler instructions.txt”.

# Testing the application

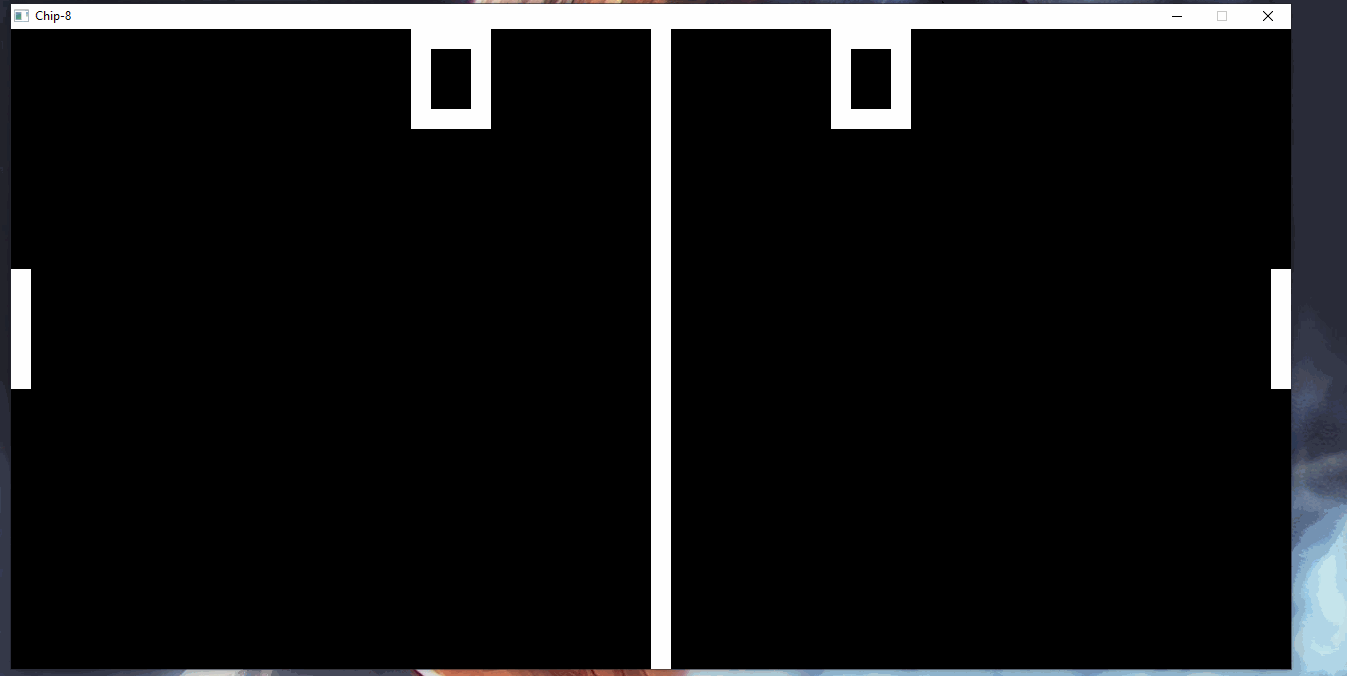
The testing part of development was made simpler due to unit-tests having been created by previous emulation and Chip-8 developers (there are still people developing modern Chip-8 games!). For example, a unit test may call an opcode, and then check its expected result, drawing whether the test has been passed or not.

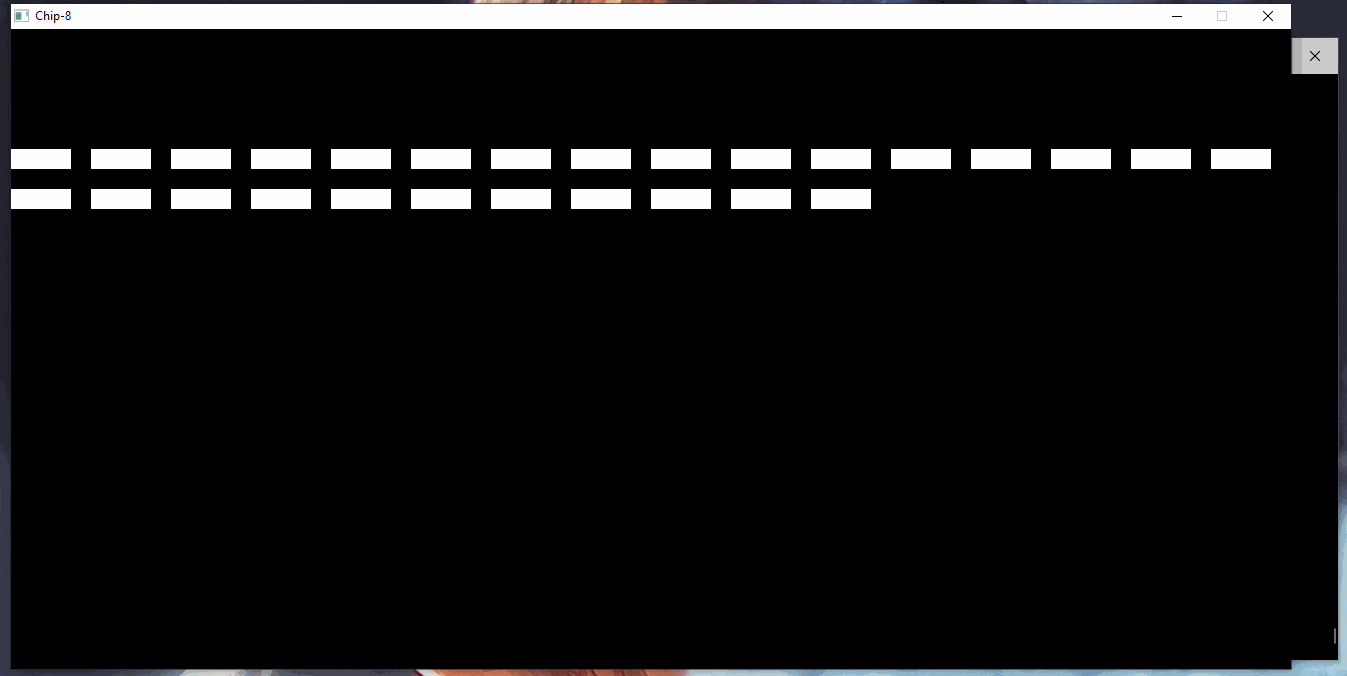
The current opcode and state of the virtual machine is also printed during every cycle and can be read to determine faulty operations. The disassembler can also be used to read the instructions of a file and determine what should happen (or in one case I ran into, read the sprite data that should have been rendered to the screen and write it out by hand).

The validity of the Chip-8 files can be double checked using a hex-editor (such as <https://hexed.it/>) to make sure the disassembler is not misreading the files.

# Tangible results

The application plays games and renders them to screen correctly!





There are some games that have problems, but this may be due to compatibility issues (there are a few versions of the Chip-8 interpreter that have slightly different opcodes, and it is not clear which version a game is developed for).

There is screen flicker during animations, but this is due to the way the pixels are updated using XOR operations (faithful to the original system). I attempted to smooth this with double buffering but did not get far. I was also not able to figure out how to generate a “BEEP” noise for the sound timer.

Before I was able to start my development, I had to gain a lot of background knowledge. At first, I was unable to read even the technical references and tutorials for developing a Chip-8 virtual machine. After all, I had never heard of a register, or different word lengths and the differences between signed and unsigned integers.

In the process of obtaining the background knowledge required, I learned about:

* Logic gates (NAND, NOT, MUX, etc), Arithmetic Logic Units and Central Processing Units
* Registers, memory, the stack, state
* Binary and hexadecimal, sized and signed integers, assembly languages
* Binary arithmetic, bitwise operations, bit shifting and bit masking

# Resources used as reference

* The Elements of Computing Systems: Building a Modern Computer from First Principles by Noam Nisan & Shimon Schocken (<https://www.nand2tetris.org/>)
* <https://github.com/mattmikolay/chip-8/wiki>
* <http://devernay.free.fr/hacks/chip8/C8TECH10.HTM>
* <https://tobiasvl.github.io/blog/write-a-chip-8-emulator/>
* <http://emulator101.com/>