EMUL-8

A CHIP-8 Emulator

Final Documentation

Team 15

Aleksandar Kostic, James Mitsumasu,

Sean Aspinall, Sean Maas, Zhen Wang

Project Summary

Originally, our project’s goal was to emulate the Nintendo Game Boy on the Raspberry Pi by recreating hardware components of the console - the CPU, memory, and graphics - entirely using software. However, due to being overambitious (further discussed in the key problems area section of this document), we were forced to downscale and switch to a less complex emulator.

Our project’s goal is now to emulate the CHIP-8 interpreter on the Raspberry Pi by recreating hardware components - the CPU, memory, and graphics - entirely using software. The emulator will execute CHIP-8 programs (i.e. video games such as Space Invaders) running on the Raspberry Pi hardware.

Welcome to EMUL-8.

Key Accomplishments

While being a complex project, approaching our methodology in an object-oriented way worked well as it allowed us to effectively modularize the main hardware components (e.g. the CPU, memory, GPU, and timer) into their own classes. As this was done early on in the project, it became easier to debug and find where certain bugs were originating. Modularization led to the encapsulation of our components. For example, we had a bug where the graphics were not showing and this was quickly determined to be a result of one of the registers being of the wrong type.

As a general principle, we were also keen on ensuring we had low coupling and high cohesion in our program. This was to ensure that code was easier to read, maintain, and extend as needed. Furthermore, we also did our best to adhere to the SOLID design principles.

As many of our group members were not familiar with emulators, online resources aided these members in learning more about the specific emulator, including the technical specifications which would be needed for this project. Furthermore, one of the particularly useful resources available was documentation on the CHIP-8 interpreter (e.g. http://devernay.free.fr/hacks/chip8/C8TECH10.HTM) which allowed us to understand the opcodes in a more meaningful way, allowing us to program the instructions correctly.

Having the documentation readily available to us allowed us to focus on determining effective and efficient design patterns that could be applicable to our implementation of the emulator. For example, we implemented the singleton design pattern when we created a logger to assist in debugging - this ultimately reduced debugging time by printing what was happening in the system internally. Finally, we immediately implemented the Model-View-Controller design pattern as this stood out to us due to the fact we were using an external library for the graphics (wxWidgets) which was updated by the GPU based on input by the CPU.

Key Problem Areas

An area that went wrong during the development was being overly ambitious on the topic of the project. There was a gross underestimation of time, resources, and knowledge associated with our initial undertaking of Game Boy emulation. During the various stages there were signs that Game Boy emulation was simply too advanced and complicated as many group members had no prior experience working with memory, user interfaces, and graphics.

However, due to overconfidence, and miscalculations in terms of time to study the resources for the Game Boy, it became clear that a change of direction was needed. That change resulted in a switch to the CHIP-8. The CHIP-8 emulator was selected to take advantage of the resources we have already built out and since it was easier to comprehend to those group members who had no prior knowledge of the subject matter.

One process that was not successful was taking action prior to planning. Even though there were some initial plans regarding how to approach the project, it was not hashed out entirely. It was only upon taking action and much further down into development that we began noticing these issues take centre-focus. A major issue is that we did not think of many design patterns to implement until after the project was completed and ready for a stable release. This was problematic because the team had to retrospectively consider design patterns and how to incorporate them into our product without causing too many issues in terms of bugs and time resources.

Indeed, during the planning phase, we considered to use state and factory design patterns to map our opcodes. However, we found that the state design for our opcodes implementation was not quite practical. One of the reasons is the enormous number of opcodes, which would make the program file split all over (i.e. we would have upwards of 40 classes for our opcodes alone). Ultimately, we decided that the cost-benefit of adding these design patterns for the opcode implementation would not be cost effective and thus discarded this idea.

Another issue that we encountered was creating measurable and deliverable user stories. There was a huge disconnect for our group in framing the user questions properly and this remained a challenge for most of the project’s development. The challenges presented by creating actionable user stories led to many debates about the direction and plausibility of certain acceptance cases for our group. One of the major issues we faced was that many of our users stories were not testable until the final product was completed - this posed an issue for the stage three deliverable. Knowing this, we worked in conjunction with our teaching assistant, Marilyn, who assisted and guided us in gaining a better understanding of both the product and direction in terms of the user stories.

As we developed the product there were challenges with git. Given the lack of familiarity with git there were issues with coordinating pushes and pulls among the members. There were a few instances where certain pieces of code were overwritten. However, once we got a system in place and members had time to read documentation pertaining to git, it was much easier to manage and understand git as a group. Overall git was an invaluable skill that needed to be learned and everyone managed to learn its basics.

The main problem of our current project is that a huge amount of time was contributed to our abandoned project, which was the Game Boy emulator. As a result, the time left for the CHIP-8 emulator was really narrow, which also resulted in poor decisions in terms of design patterns and their implementation. However, based on the research we did for the Game Boy, and some other platforms, the CHIP-8 emulator would not take too much time to implement relative to other emulators.

Lessons Learned

While great to be ambitious when doing a project like this, it is also important to be realistic. There are many areas to be realistic about our expectations such as knowledge, available resources, time, and experience. We've all definitely learned this throughout this project, and will be considerate of this fact in the future.

Furthermore, as most of us have not worked in a group environment like this many times, communication at times was not the best. However, as time went on, we were able to implement many communication strategies like ensuring all members checked their emails, messages, and Slack at least once a day and through the use of polls to ensure everyone had a chance to get involved.

Another best practice that we developed over time was to breakdown our project ad system as soon as possible. This allowed us to draw out our system before we started coding which ultimately reduced the headaches we would have had down the road.

We would definitely be interested in working in a large scale project like this in the future; however, we would prefer to choose a project that more group members were familiar with.