Video Game Development with Microcontrollers, Emulators, and Snap! Code

CS4991 Capstone Report, 2024

Anisha Poudel
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ap6acf@virginia.edu

ABSTRACT
The Snap! code video game development guide created during my independent research during the 2023-2024 school year is designed primarily for students with little to no experience with arcade game development. The project consists of creating arcade games using emulators, microcontrollers, and Snap! code. The project was developed using an agile framework called Kanban. The development team had a two to three week iteration period during which members would work independently and reconvene to showcase what they had been working on, discuss updates, and address any questions. The guide aims to introduce audiences to a wide range of technical concepts including software and hardware development. As the project is still in the works, more chapters need to be created before publishing.

1. INTRODUCTION

The Arcade game creation guide consists of 4 sections: 1) Introduction, 2) Emulation, 3) Control Bar, 4) Microcomputers. The introduction covers the early history of arcade games and the key tools used in the project. The emulation chapter covers how to install the correct emulator to the user's PC. The control bar chapter reviews the central components of the Raspberry Pico microcontroller and how to design a control bar. The last chapter focuses on how to install an operating system on the Raspberry Pi and the tools needed.

Each chapter needs to be reviewed and revised as it is written. These reviews consisted of ensuring that the code is logical, instructions are compatible with varying operating systems, along with other revisions that adds clarity to readers.

2. BACKGROUND

Snap! is a drag and drop programming language, implemented with building blocks. The language was first introduced by developers at the University of California, Berkeley in 2011 (UC Berkeley Snap!, 2019). It also features first class list, procedures, and continuations. These capabilities make it ideal for an introductory language. Snap! runs on a user's browser, and is implemented using JavaScript. An emulator is a computer or program that imitates another computer or program.

The emulation capability can differ based on the type of the emulator, but the goal of each emulator is the same: replicating the experience of the original software or hardware. Emulator allows software and hardware to be used cross platform. Microcontrollers are simplified microcomputers functioning to govern specific operations in an embedded system. It consists of a central processing unit, memory, and peripherals. The Snap! code logic needs to be translated to a language that the

microcontroller can interpret, which is done by an emulator.

3. RELATED WORKS

In this project, Snap! is used to introduce arcade development concepts. Similarly, Khan et al. (2018) explores Snap! in an educational context, teaching children in developing countries AI programing. Their study illustrated how block-based programming makes topics as complex as AI accessible and engaging for young learners. The study found that students who had prior programming experience deemed AI programming straightforward, while those without took longer time to grasp these concepts. Their work highlights the importance of scaffolded learning environments to bridge learning gaps among students.

In the process of exploring the foundational aspects of microcontroller programming for arcade game development, the work of Kohli (2023) serves as an essential resource. This article functions as a comprehensive technical guide for microcontrollers, discussing various platforms such as Raspberry Pi and Ardunio, alongside their application in real world embedded systems. Kohli outlines critical programing tools and techniques, including IDEs, debuggers, and PVM, which are all integral to designing and optimizing game performance and functionality. This resource not only enriched my knowledge of hardware functionalities but also paralleled the project's use of emulators and microcontrollers to translate programming code into game logic, allowing for compatibility across different hardware systems.

4. PROGRAM DESIGN

The team met to discuss possible applications and coding languages to use in the instruction guide and how to display that information. The following subsections describe each chapter in the book and possible refinements.

4.1 Introduction

The readers are briefly introduced to the early history of microprocessors and their contribution to arcade game development. The operating system and hardware of today's computers significantly differ from early microcomputers. Emulators are so called because they emulate the operating system of early computers allowing instructions that were previously stored in read-only memory (ROM) to be now executed by contemporary computers. Microcomputers like Raspberry Pi combined with emulators allow for a new generation of open source video games inspired by classic video games.

4.2 Introduction to Emulators

For this section, I reviewed and tested the instructions on how to install MAME, one of the most popular arcade emulators, on various operating systems. This is where I ran into my first issue regarding compatibility with Windows and Macs OS. There had been many issues installing MAME on non-Windows operating systems. The team chose to switch to Ruffle, an open source flash emulator. This was specifically made to run on all modern operating systems, which is the scope of our project.
If any issues arise with further development, this is subject to change.

4.3 Creating an Arcade Control Bar

An arcade control bar can replace keyboard controls, and can even be customized to mimic retro video game remotes. This essentially creates a microcontroller that can be programmed to mimic key presses on a computer keyboard. This section taught users how to program their microcontroller to interpret signals from

their arcade controllers that can be translated to the computer.

Figure 1: Raspberry Pico Microcontroller

The first step was to create instructions to design the arcade controller.

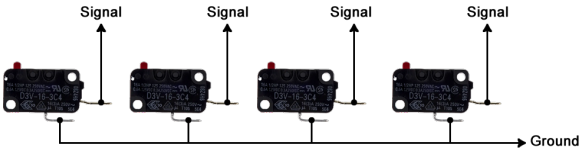

Figure 2: Button Signal Connection

Figure 2 illustrates how the switches on the arcade button should connect. Each signal is connected to one another through the ground input and connected to four different digital pins on the microcontroller as well.

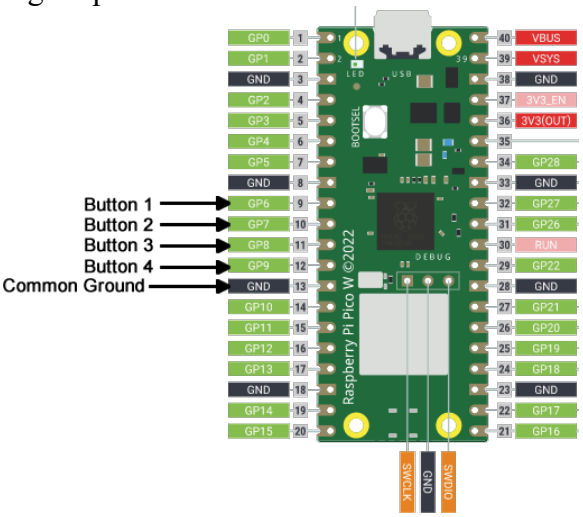

Figure 3: Circuit layout

Figure 3 is an example of ways signals can be connected to digital pins. In this case, four buttons are connected to digital input pins 6, 7, 8, and 9 on the microcontroller. Users do not need to make the same configuration with their digital pins as the example as long as they match the number of the physical pins on the microcontroller to the corresponding references in the computer that monitors these inputs. The four ground connections are connected to a ground input on the microcontroller. Additionally, the four microswitches on the joystick are connected to the input pins of the microcontroller in a similar configuration.

4.4 Programming the Microcontroller

The Raspberry Pico can be programmed to emulate keystrokes on a keyboard using MicroBlocks, a block-based coding language. To use MicroBlocks, users must first install firmware on the microcontroller. Firmware is a software that provides basic machine instructions that allow the hardware to function and communicate with other software running devices.

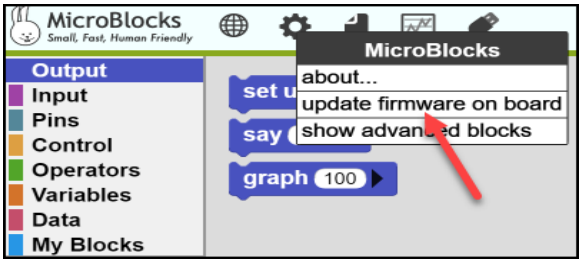

Figure 4: Firmware Setting Menu

Further instructions are given to readers to load *Keyboard and Mouse* libraries in MicroBlocks. In keyboard emulation, MicroBlocks sends the correlated keypress to the program running the arcade when the joystick is pushed up, down, left, or right, simulating a keystroke on a computer keyboard. Additionally, the hold key code block can emulate pressing and holding keys by entering the associated letter. In Figure 5, when the appointed button on the arcade control panel is pressed, the letter “S” is emulated:

Figure 5: Keyboard "S" Emulation

In the case of keys such as arrow keys, a numeric code is used in place of the letter, as seen in Figure 6:

Figure 6: Figure 5: Keyboard Up Arrow Emulation

4.5 An Introduction to Microcomputers

First, an operating system must be installed on the Raspberry Pi. To accomplish this, the user needs internet access, a microSD card, a microSD card reader, and the microcomputer. First step is to download the Raspberry Pi Disk Imager install file and select the appropriate operating system for the user's computer. Disk imagers are programs that copy an operating system from memory of one computer to another.

1. Micro SD Card Slot
2. USB-C Power Port
3. Micro HDMI Ports
4. USB 2.0 Ports
5. USB 3.0 Ports
6. Ethernet Port

Figure 7: Raspberry Pi 4 Microcomputer Labels

Figure 7 helps readers set up their microcomputers and connect appropriate ports. After the microcomputer is connected to a monitor by the HDMI port, it can be powered through the USB-C port. Shortly after, the operating system will be launched.

One benefit of using microcomputers for a project is being able to directly launch into a program. As seen in Figure 9, users can enter the following command to their terminal: *sudo nano/etc/xdg/lxsession/LXDE-pi/autostart*

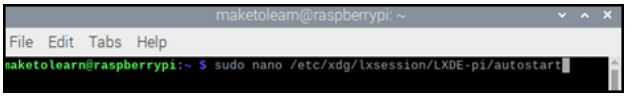

Figure 8: Terminal Input

If entered correctly, this should open the autostart file and allow you to edit it. The file should already contain several lines of text. To run Snap*!* program on start, move the cursor to beneath the other text and enter the commands below:

@chromium-browser --kiosk --disable-web-security --allow-file-access-from-files file:////home/[your username]/Snap/snap.html#run:file:////home/[your username]/Snap/[your program].xml

Once this is set up, the file can be launched.

5. RESULTS

Preliminary testing has shown that users can follow the manual to replicate the project outcome with ease regardless of experience level. However, future testing on all chapters remains pending. After comprehensive testing concludes for the remaining chapters, I anticipate the publishing process will begin.

6. CONCLUSION

This Snap! code video game development guide represents advancements in making game development more inclusive for all.

The project not only serves as an educational guide but intentionally caters to those who have little to no experience in video game development, different compared to many other development guides. This project has deepened my understanding of both programming and project management which has been invaluable to my professional growth.

7. FUTURE WORK

This guide needs more chapters to be completed. Multiple games are in the works for students to follow. Further testing will be conducted on these games and chapters to ensure compatibility within all devices. A handful of other tasks need to be finished by the end of 2024. Each chapter goes through multiple iterations in which the content is refined and tested. The team will continue to work together throughout the year to complete the guide.

R|EFERENCES

[1] UC Berkeley, Snap! (2019). Snap! Build Your Own Blocks. Snap! Build Your Own Blocks. Retrieved April 16, 2024, from https://snap.berkeley.edu/about

[2] Kahn, K., Megasari, R., Piantari, E., & Junaeti, E. (2018). AI Programming by Children using Snap! Block Programming in a Developing Country. European Conference on Technology Enhanced Learning.

[3] Kohli, V. (2023, April 12). Microcontroller Programming: Mastering the Foundation of Embedded Systems. Wevolver. Retrieved April 13, 2024, from https://www.wevolver.com/article/microcontroller-programming-mastering-the-foundation-of-embedded-systems