Mini Arcade Cabinet

Final Project Review

Team Number: sdmay23-26
Client: Brad Yenger
Advisers: Wymore, Mathew L [E CPE]
Team Members/Roles:

Bradley Yenger: Electrical Engineering
Mark Gores: Software Engineering
Jeffrey Marsh:Software Engineering
Liam Tureaud: Electrical Engineering
Alexander Glass: Software Engineering
David Helmick: Software Engineering

Team Email: sdmay23-26@iastate.edu
Team Website: https://sdmay23-26.sd.ece.iastate.edu/

Design Evolution

Cabinet

The box went through a few iterations before the design was finalized. There was a thought to have acrylic panels on the side that were able to be easily removed and to showcase the internals of the system. We wanted the LED matrix set back slightly within the machine and an acrylic panel over the top and in front of the monitor for protection as well. We did not have the time, know how, or resources to implement this design however so the idea for that was scrapped. Other smaller changes were the orientation of the USB attachments, an improved mount for the monitor, and how the LED matrix was attached to the top of the cabinet. These all were modified from the original 3D model for convenience or wire management.

LED Matrix Circuit

There have been numerous deviations from our original design in relation to EE 491. One design change came towards the end of last semester and was lightly touched in our final design document. That being the LED matrix circuit. We originally wanted to capture the 80's retro arcade aesthetic and having a flashing display is the epitome of that. Time and research was dedicated into finding what materials we would need and how we could make our own.

The original size of the matrix was 48x8. This would give us a large display with a nice resolution as the intent was to display messages across the board. When installing the matrix into the cabinet it would not fit as it was too long. The fix for this issue was to cut it down by 3 columns. Next the issue with it was that it was not able to display certain letters. We believe the issue to be a short circuit as letters like A and H are not able to be printed. This led us to change up the idea and do some random effects to keep the flashy feel.

The driver circuitry behind the LED matrix went through several iterations. The original intent was to make PCBs. This idea was scrapped as testing and soldering on perfboard took quite a while. It was also in our best interest as the PCB would cost us extra money and there is always the risk that the boards do not work. There was a lot of soldering and desoldering that took place as we were trying to find the optimal position for the 74HC595 chips and other components that were needed.

Power Supply Circuit

The original design called for a range of power requirements. These were projected to be a 3.3, 5, and 12 volt supply. This was built between the two semesters. This circuit worked for the original test using a multimeter. it was able to supply all the voltages required. This was put aside until the final constitution of the box. When we tried to power the speaker, we found a large problem. With just the speaker attached, the 12 volt supply was burning up. With testing, we found a range of problems. Both the voltage regulator and the wires used were not rated for the amperage that was required. This would need a complete redesign, ordering parts, and testing time which wasn't available. This circuit was replaced by a power strip in the final design. This circuit is shown in Appendix II G.

On/Off circuit

The circuit below was designed to briefly interrupt the power supply to the Raspberry Pi. After both the change to the power supply and the circuits unreliability to turn the Raspberry Pi, it was replaced and incorporated with the temperature sensor. After testing, the Raspberry Pi seems to have a power regulator that can keep the Pi on if power is briefly interrupted. This would stop the Pi from turning off, and the button became unreliable. The change was to let the microcontroller that was reading the temperature sensor also read the input of the button press. The microcontroller would then output a HIGH value to a relay which broke the circuit for the power strip. This circuit is shown in Appendix II H.

Controller Mapping

Deviations had to be made when there was no real viable way to receive controller inputs with the java version the Raspberry Pi was running. Had to receive and map the game controller inputs on an operating system level and make an interface that could communicate with the operating system program that would be running in the background to accurately update the mapping of inputs to the user's custom liking.

Icon Size

Game buttons are accompanied by an icon that describes them on the game select screen, to keep the design of the game select screen slick these icons should be 64x64 pixels. The original plan to enforce these size restrictions was to disallow the user from choosing an incorrectly sized icon when adding a game to the USB. It quickly became clear that forcing the user to resize their images would be too annoying to deal with. Instead we implemented a system that allows any image to be used as an icon and said image will be automatically resized when opening the select game screen on the PI.

Focus on the .jar file format

Our original assumption was that running games of many different types on the PI would be a simple task. This could not have been farther from the truth, writing the planned code that would run different types of game on the Pi would have been a simple task but the computing resources on the PI were much more restrictive than anticipated so it's unlikely that code would have worked. In order to simplify implementation we decided to focus more heavily on the jar format which worked natively on the PI. Given more time figuring out how to get more game types working would be a logical next step.

Minimal graphics

One feature we planned to implement was different graphical “themes” for the UI allowing the user to customize the look of the UI. We chose to not go forward with this idea because of the long load time associated with loading new graphics on the PI. After changing the theme it would take quite a while for the new graphics to load. Especially since you have already loaded all the screens of the original theme. There are definitely ways to get around this but we decided instead to just focus on making the theme we have look the best it can.

Functional and Nonfunctional Requirements

Functional:

- Users have the ability to play an assortment of retro games.
- Users have the ability to use any controller they choose.
- Users have the ability to upload any game that is a runnable jar file.
- Users have the ability to control the volume.

NonFunctional:

- Game will boot up in no longer than 5 secs
- Fan will kick on to cool the CPU to prevent throttling.
- I/O Delay will be kept to a minimum(not noticeable)

Relevant Standards

For wire routing and standard powers, we will mimic the JAMMA standard:

This is a set of basic standards for easy repairs for arcade cabinets. This allowed for old arcade cabinets to be repaired and reused by using standard parts and standard voltages for components

OSHA standards: Safety and preventive planning

ASAP*(Appliance standards awareness project) (https://appliance-standards.org/national)

*While we cant meet all standard under this section, we took inspiration to implement power saving (while not in use, save power)

Engineering Constraints

There were a collection of constraints and requirements we were to follow when creating our project.

Physical:

We were given a size constraint of a 2ft cube

Able to be portable and light enough to be carried

Wanted to have a retro aesthetic

User:

The user should be able to control audio volume

control scheme for the games and create custom controls have responsive controls (10ms input delay)

Software:

Clean UI

Able to run .jar files

Lag free gameplay

Able to read inputs from a controller

Material:

All devices contained must stay below 85 Celsius

Cabinet made out of a hardwood

Concerns

There are many concerns that arise when trying to create a functional device. The first concern in the creation of our project was to figure out which coding language would be used to construct the UI as we were using the Raspberry Pi. Our group knew Java well and the Pi was able to support that.

Another concern present was wire management and making sure each subsystem was working properly. The ability of having our wires neat and organized leads to each system being powered properly and secured to their respective locations. A power strip was added with a surge protector as well for added protection.

Stemming from the issues with messy cables, the issue of hot components or having them overheat is an ever present issue. This could lead to a fire and the total destruction of a subsystem or at worst, the cabinet as a whole. To prevent this issue and to cool our circuits to make sure they are running at peak efficiency, an automatic fan will kick on when the ambient temperature reaches a certain point.

There are a whole set of issues that could happen with the software of our system. Due to research and independent testing, the Raspberry Pi was able to run the needed software that was chosen for this project. Running into bugs and having the software crash is another issue that needs to be addressed. The software team did extensive testing to fix the issues within their code.

There were other issues that were quite low with the likelihood of them happening. We had ideas to prevent the breaking of certain components, like the monitor, by placing an acrylic panel over but this was never implemented. Another issue, that is serious but not so applicable to this scenario, would be the insertion of a malicious usb into the exposed ports in which the user can upload their games to the system. The fix for this would be a user set password on the ports. This was also never implemented.

	LikeliHood	Severity	Mitigation	severity after mitigation
Overheat/fire	moderate/low	major	Apply external cooling	low
Raspberry pi not able to run software	moderate/low	moderate	have other options to run software	low
Power being disconnected/surge	low	low	ensure wire and cords are properly stored neatly and surge protector is in place	low
software crashing	high/moderate	low	extensive testing and allow a reboot option for user	low
Malicious USB upload	low	major	Set password to access and upload files (not implemented)	low
Breaking of components	low	major	Acrylic panels (not implemented), proper wiring and correct power delivery	low

Implementation Details

Software:

The Raspberry Pi is running its default Pi operating system (OS) which is a linux- like OS. The Graphical user interface (GUI) is all written in Java Swing. The mapping of controllers utilizes a program called Qjoypad which is a program written in C. All controller inputs are mapped to keyboard strokes. A Swing GUI is implemented to interface with the Qjoypad layout files that allows for the user to easily rewrite the layout file in order to custom map their own controller inputs. Uploading games onto the Pi was made through a java swing application that allows the user to select what files,text, and images they would like to upload to the cabinet and then proceeds to build a configuration file and move the selected files to the USB drive. The usb drive is then inserted into the cabinet and the game is automatically loaded to the select game screen after the configuration file is read. The Diagram for the software systems can be referenced in Appendix II F.

Hardware:

The hardware team sat down and ironed out the subsystems that would need to be implemented within our design. In Appendix II E the block diagram shows how the circuits were connected. The purple rectangles are outputs and the orange rectangles are user inputs. The red triangle is an environmental input, which would tell the arduino the temperature in the cabinet. With power supplied, the electrical engineers were able to build the rest of the required systems.

Power Circuit

The first system created was the power circuit which would pull from a wall socket and power the Pi, arduino, speaker, monitor and connected systems. The final design ended up being a simple latch circuit, controlled by an arduino, wired to a 25 cent arcade style power button.

LED matrix

We created a 45x8 led matrix. The anodes of the leds were all soldered together to create the columns of the matrix. The cathodes were then soldering together to create the rows. This is shown in Appendix II, D. To drive the leds we then needed to create the circuity. This consisted of an arduino and 7, 74HC595 pin extenders.There were 6 pin extenders for the rows. They were in daisy chain configuration as shown in Appendix II, B. A resistor was used to limit the current into the leds. The first 74HC595 in the series had pin 11 connected to digital output 2 of the arduino, pin 12 of 74HC595 was connected to digital output 3. Finally pin 14 of the 74HC595 was connected to digital pin 4 of the arduino. We used 1 pin extender for the 8 rows, as it has 8 outputs. A resistor was used to limit the current going into the leds as well. Pins 11, 12, and 14 were wired to digital pins 5, 6, and 7 respectively. The diagram for the row driver can be found in Appendix II C. With some arduino code you are able to print various messages or do random flashy effects across the board.

On/Off circuit

Like a modern computer, we wanted to add a stylized power button that the user will press and hold for a brief moment before the system turns on. We decided to go with a button that looks like a coin slot, keeping true to the style of a retro arcade cabinet. This also can help older users understand the implication of the button. When you play an arcade game, you usually insert a quarter. This understanding will let new users know where to start to turn on the device. This button gets fed into an arduino. This arduino continuously checks the input of the button. If the button does get pressed, it has a delay for a second, then it will check for the button press again. This is implemented in the station where you just slightly bump into the button but you don't want it to turn on. If the button is pressed for both checks, it will output a HIGH value to a relay. This relay breaks the power from the power strip. Once the relay gets the HIGH value from the arduino, it closes the circuit and lets the power strip power all the other devices.

The button works in the same way when the user would like to turn the device off. It repeats the same double check for the button, and if both are met it will supply a LOW to the relay. This will break the circuit and turn off all the devices.

Temperature sensor and fan

The same arduino that controls the power button also controls the temperature sensor circuit. This implements a DS1822 temperature sensor, connected to an analog input on the arduino. When over a set temperature (65°F) the arduino will output a HIGH to a relay. This relay allows a 12 volt supply to power a fan. When the temperature drops below the set temperature, it will supply a LOW to the relay and break the circuit.

Testing

Unit testing - All portions were individually unit tested by the individual that created the portion, then an outsider who did not work on the unit also tested it to gain outside perspective and ensure units should be functioning as intended. Most hardware components were simulated using softwares such as TinkerCad and LTSpice. Then Breadboards were used to test the components for extra verification. After creation of PCBs or circuits they were then tested again before connecting with other systems.

Integration testing- A two week period was provided for all integration testing as we agreed that integration testing is where most bugs, faults, and failures come up. We began by testing pairs of units to ensure each pair worked properly together, then slowly tested larger groups of units together.

Regression testing - As faults began to arise we noticed we had to take out some features provided by some of the units (most notably .EXE files). The units that were changed had to be retested to ensure the changes did not cause any other faults or problems in other locations.

Appendix I - Operation Manual

Power and Start-up

-Two power cables to plug into an outlet (one for the arduino controlling the On/Off state, one for the power strip that everything else is connected to)

-Once powered, push in 25c coin slot button for a half second to power on the cabinet

-The system will boot up and automatically bring up the home screen UI

-Push the 25c coin slot button to power the cabinet off

Adjust Volume

-The speaker is located directly above the display

-Adjust the dial to where the volume is to the users preferred level

Default and Custom Controls

-The arcade cabinet has a set of default controls if you choose to use them, but there is a method to set custom controls to then map any controls you wish

-Navigate into settings

-Select "Customize Controls" to set the controls that you want to use to play the games

-Follow the directions on screen and input each key bind that you want to be mapped to either the joystick and buttons or a third party controller

-After inputting the keystrokes, make sure that the "Activate Custom Controls" option is selected

-ready to play with the new customized controls

Adding New Games to the Cabinet

-Download a game of your choice and images to use for the background and icon (images are optional).

-Run the .jar application on the USB drive and in the GUI select the files you just downloaded and enter any remaining information.

-Click save. If the program detects any issues, it will inform you.

-Insert the drive into a USB port on the lower, front of the cabinet.

-The arcade application will read the drive and insert a new instance of a game to the game list screen.

-Once selected, you can play the game that you have uploaded.

Playing Games

-Once at the main menu, select the "Play Games" options

-A list of games will pop up and display icons and descriptions associated with the specific game

-Scroll and select which game you want to play

-Once selected, a pop up screen will appear, running the game you selected.

-Have fun!

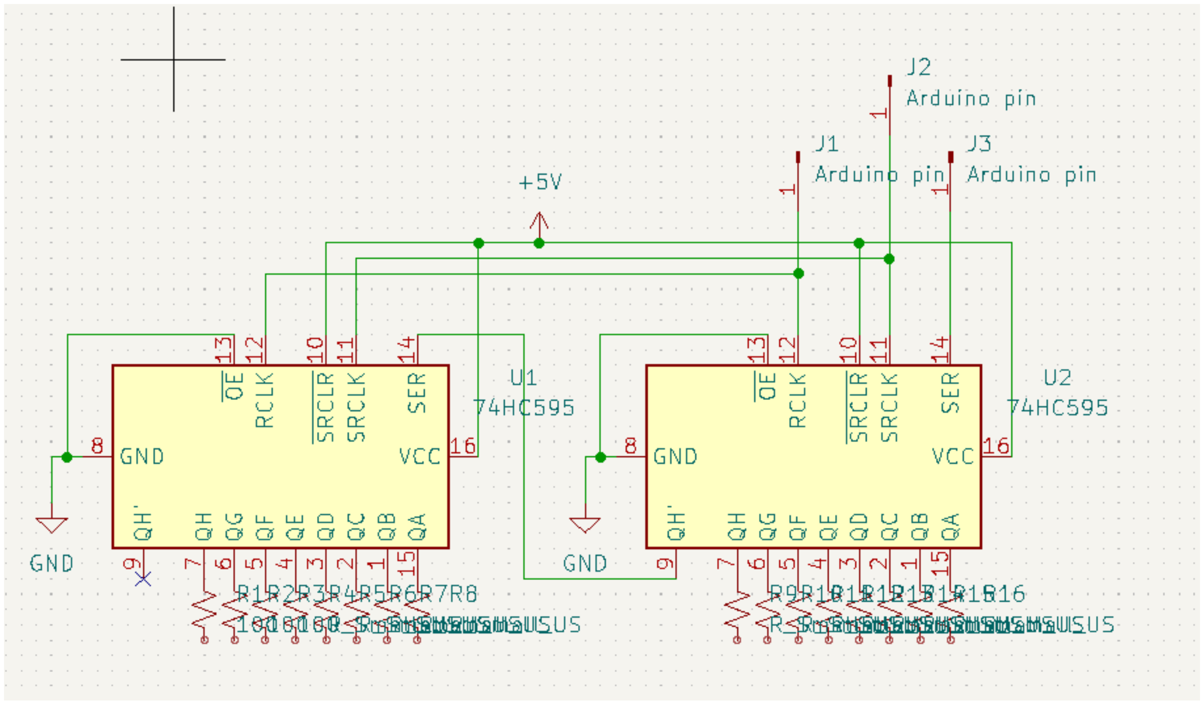
Appendix II - Alternative Versions

A. Software:

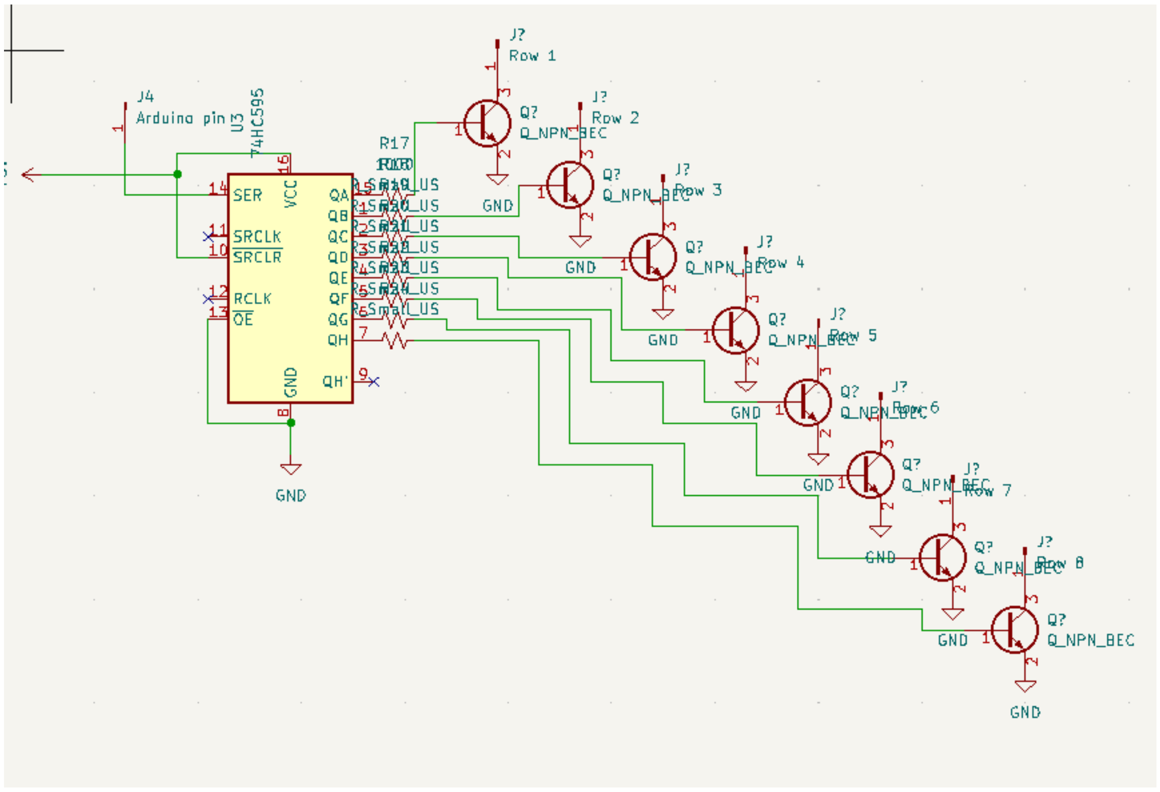
Original idea to receive controller inputs was to use a java library called JInput. This library (along with most other game controller inputs were depreciated and would not work with the version of java running on the PI. Had to change the idea to receive the controller inputs on an operating system level and made a java interface that was able to write to the layout file that the operating system program utilized.

Original idea was to have the ability to run windows .exe files to run games on the Raspberry Pi. This idea had to be changed because the Pis operating system is linux based and was very difficult to run windows programs. Wine was not a viable alternative for running the .exe files as it only supports x86 architectures and the Pi uses ARM.

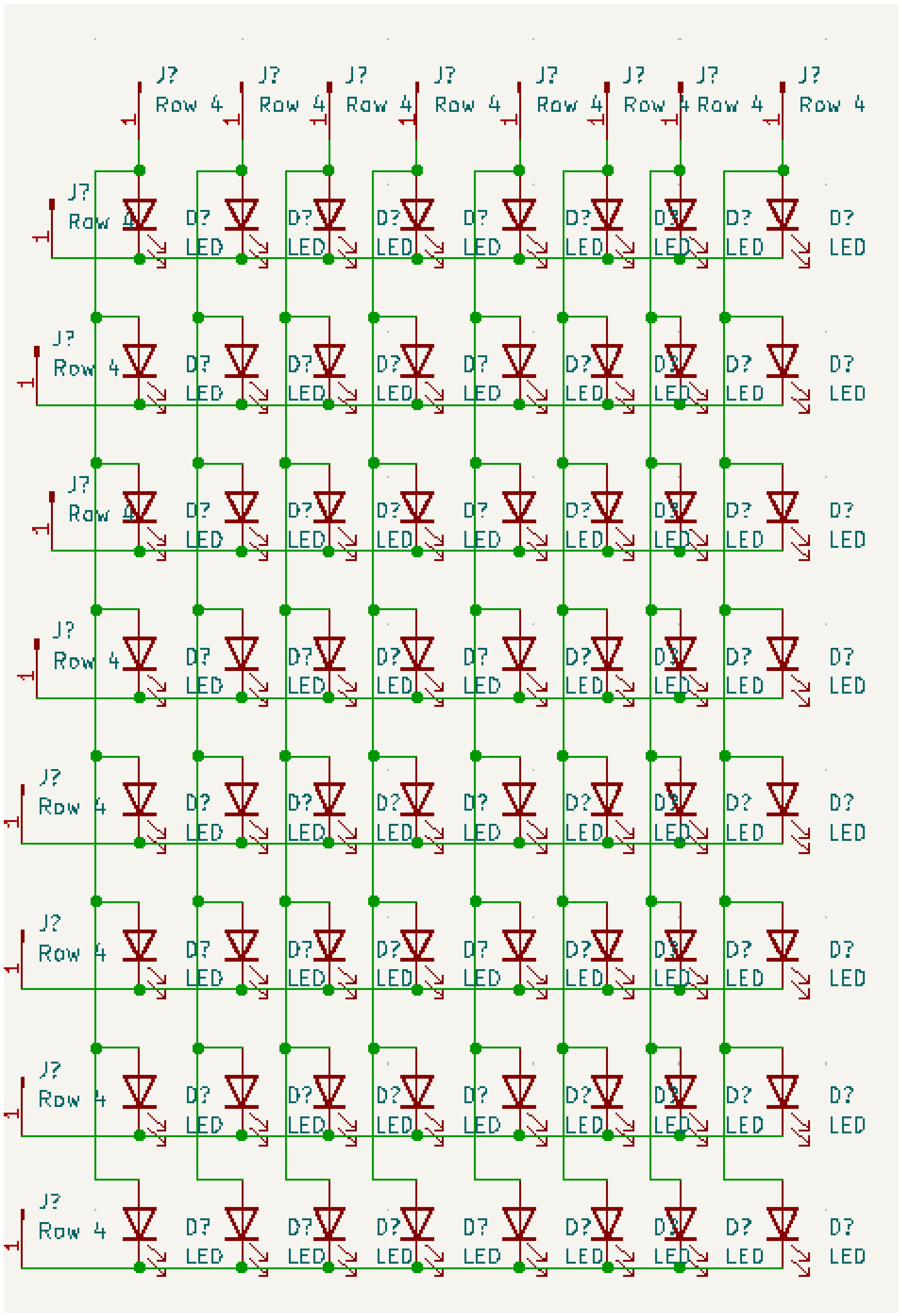
B. Column Driver Circuit



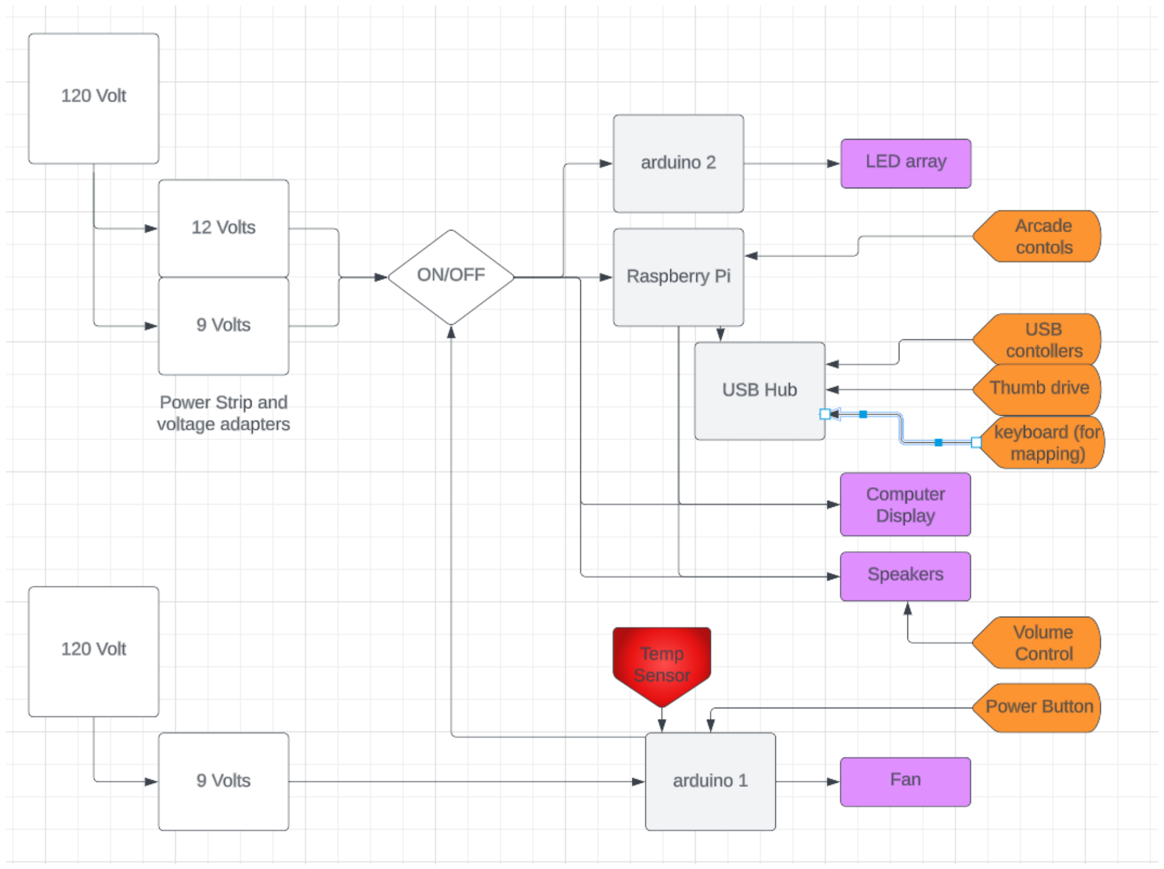
C. Row Driver Circuit



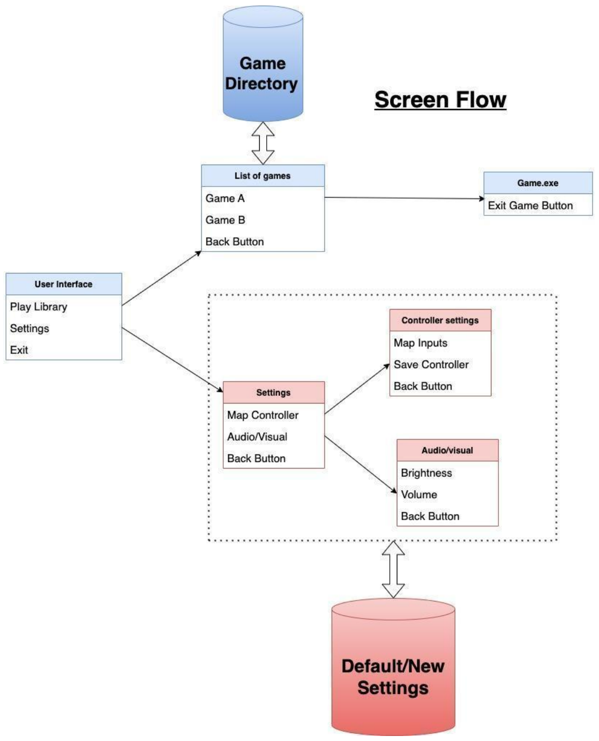
D. LED Matrix Circuit



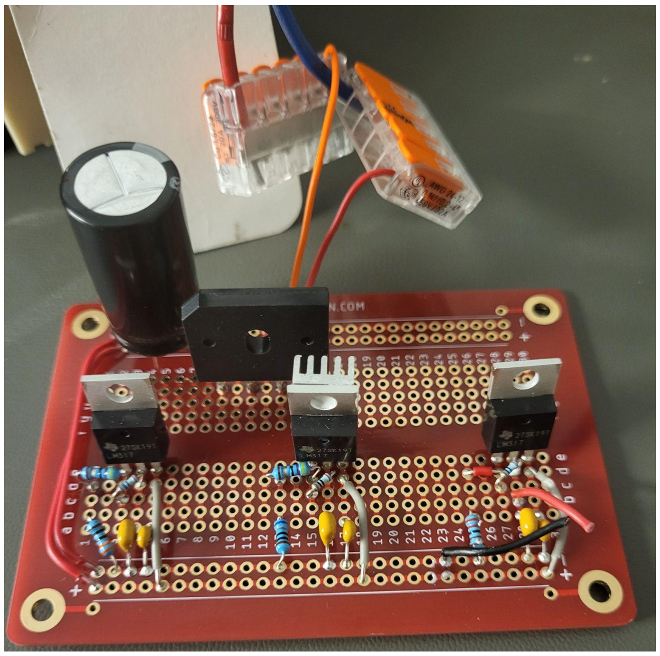
E. Hardware Diagram



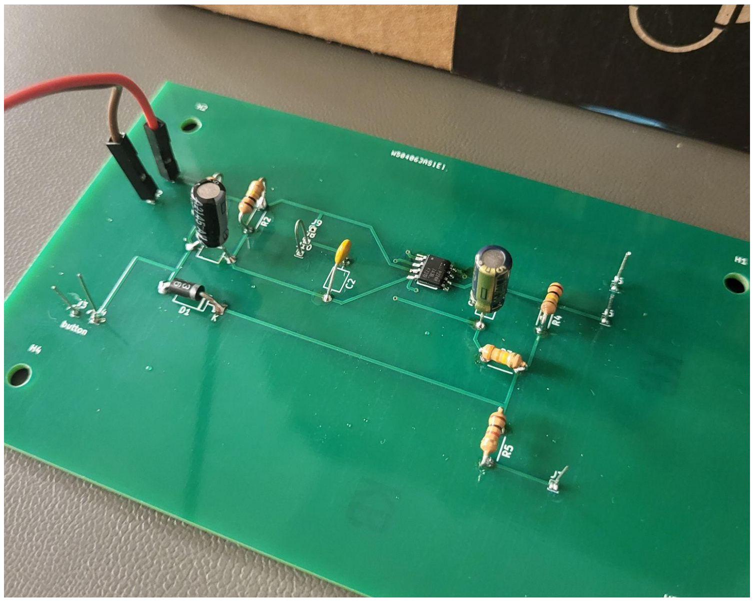
F. Software Diagram



G. Power Supply Circuit



H. On/Off Circuit



Appendix III - Other Considerations

Maybe put skills that were needed or classes that helped? Maybe something you learned.

A. **Skills needed**

- PCB / Circuit Design
- Soldering
- Carpentry
- 3D CAD Design
- Coding in various languages for various applications

B. **Applicable Courses from Iowa State University Curriculum**

EE	CprE	SE/COMS	Others
303	281	227	ENGL 314
333	288	228	
230		309	
		319	
		317	