Raspberry Pi Texting Game:

November 16th – December 12th, 2019



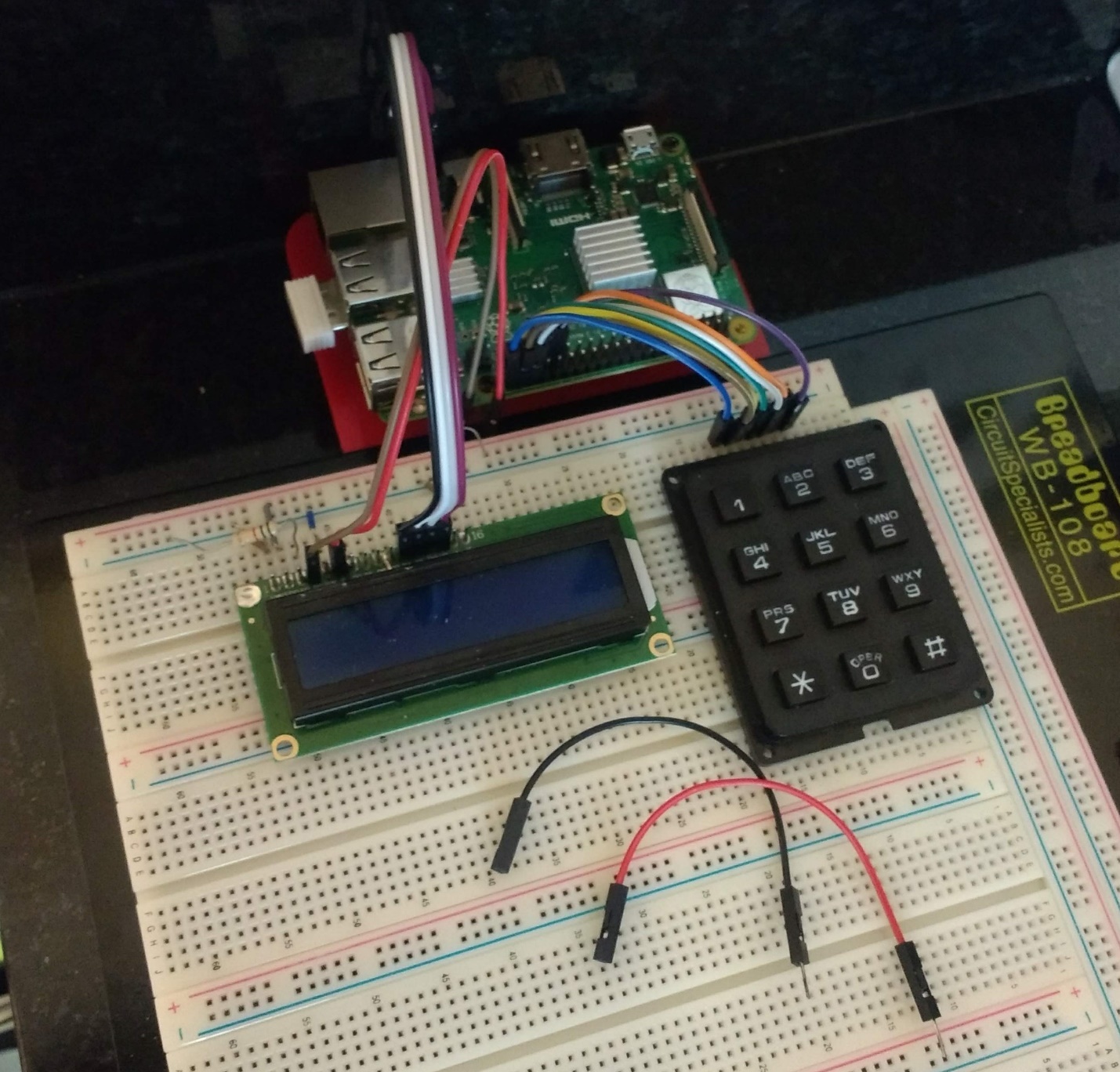
**Task:** I’ve always loved old video games, but after visiting a retro arcade with my brother on a long weekend, I realized I especially liked arcade games based around mundane, non-violent tasks. This arcade had a game I had never seen called “Timber,” in which two lumberjacks compete to chop down more trees in a forest. I had been looking for a Raspberry Pi project that used a phone keypad since I had one in my possession and it looked fun to work with, so I decided to make my own arcade game based around T9 texting, with “graphics” displayed on a small LCD character display.

**Process:**

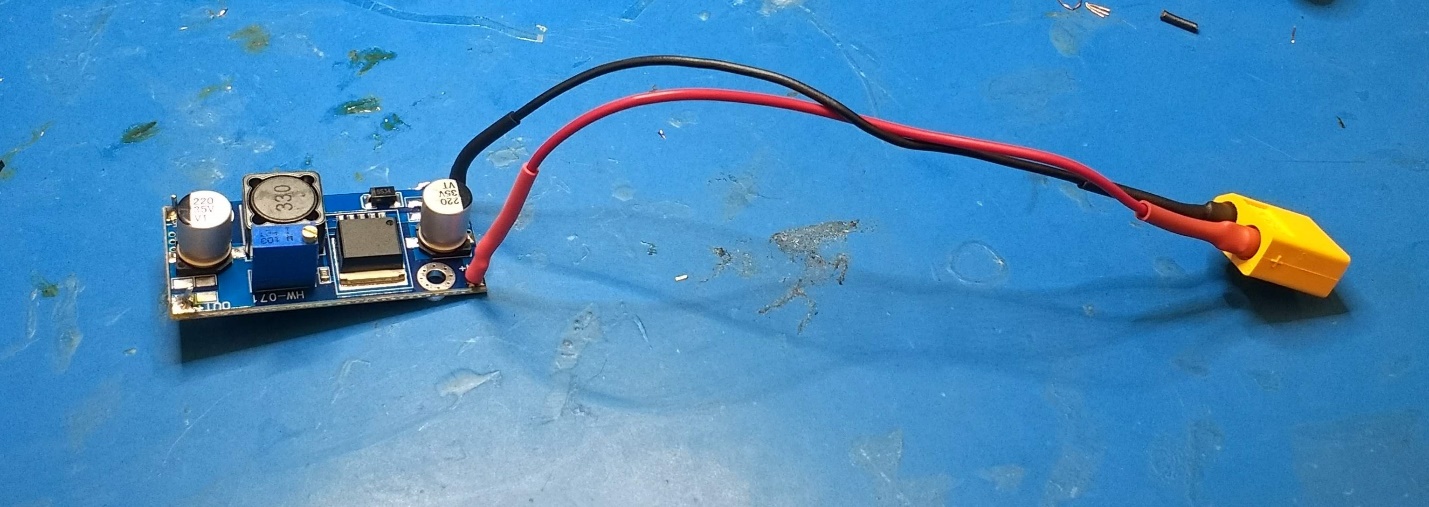
Hardware:

The main hardware I used was a Raspberry Pi 3 B+, a 16x2 Hitachi LCD display, and a 3x4 multiplexing phone keypad. I initially struggled to find the keypad pinout, but after scouring all over I finally found an Amazon user who found the pinout by hand after having my same problem. (This contributed to a negative review, which I’d agree with)

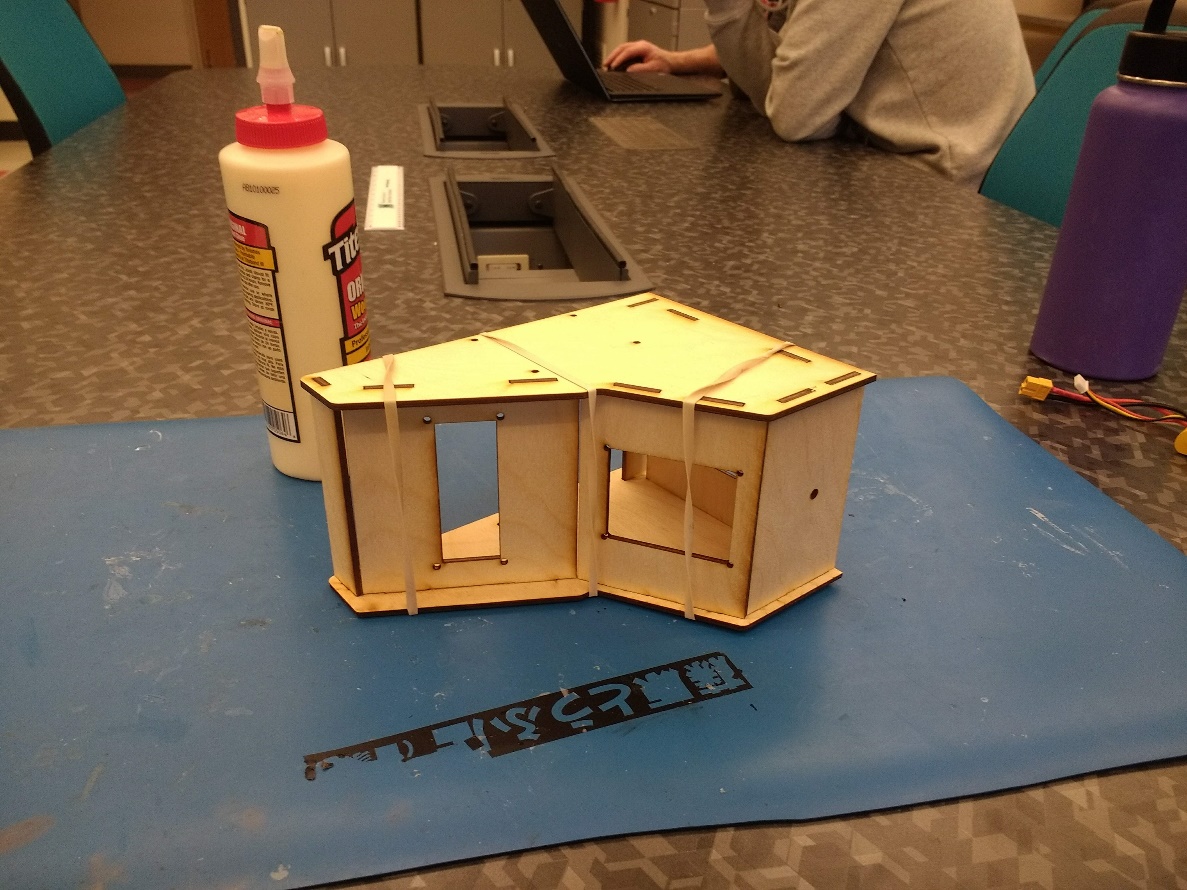
After wiring the screen and keypad to the RPi on a breadboard, I tested them using the RPLCD library and a Github user’s custom Keypad class, with both working quite well.



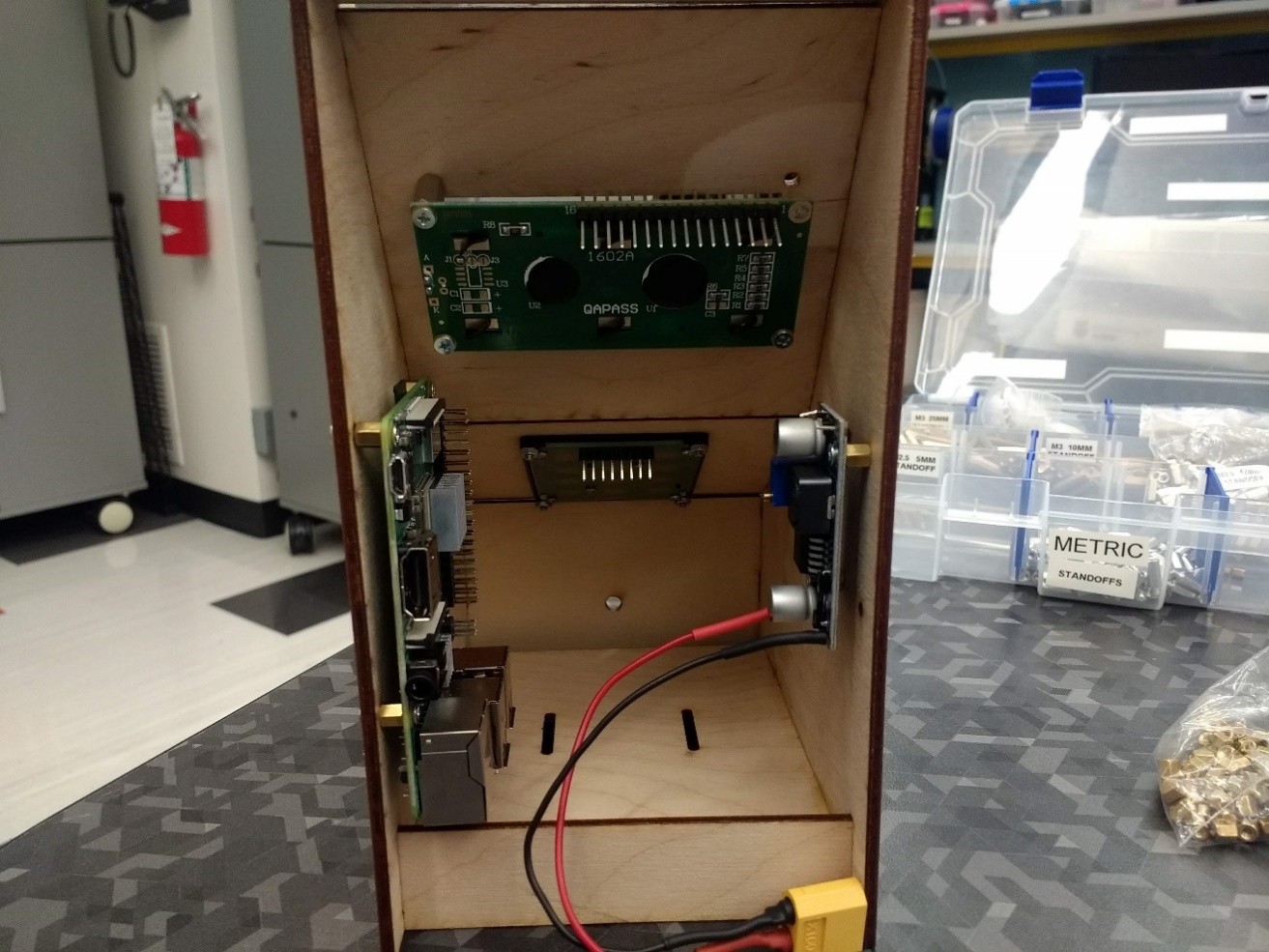
Due to the mobile nature of the project, I decided to implement a battery, using an 11 volt RC car battery wired to a DC buck converter. Though unconventional, Seattle U’s labs have plenty of RC car batteries on hand, making the setup simple to implement. The buck converter needed pins and a battery cable soldered to it, and an on-board potentiometer had to be set to output an RPi-friendly 5 volts, which I did with a DC power supply and multimeter.



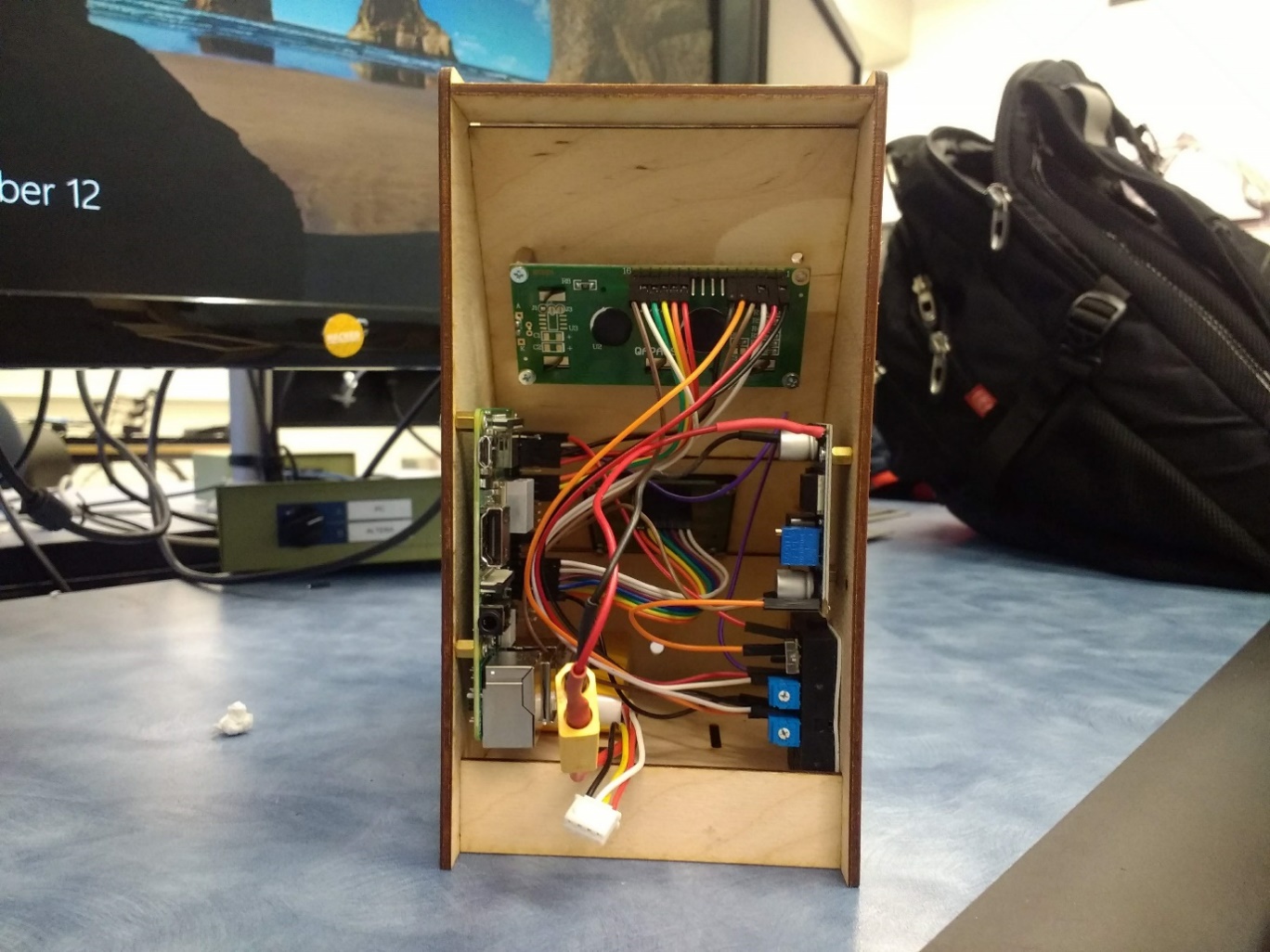
For the cabinet, one of the lab professors helped me draft and laser-cut a custom wooden enclosure, which I then glued together.



After the enclosure dried, I mounted all the hardware inside, with everything going on standoffs except the keypad.



Next came the part I had been dreading: the wiring. The LCD screen required potentiometers to set the brightness and contrast, so I stuck a small breadboard underneath the buck converter to hold those. I also added a small switch to bridge the buck converter and RPi, so the game could be shut off without unplugging the battery. The remaining breadboard terminals were used as 3.3v and Ground hubs, since the hardware required more than the RPi’s two outputs. The buck converter output went to 5v and Ground on the RPi, with all other hardware connections going to GPIO pins.



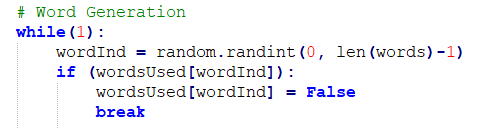
Finally, I crossed my fingers and tested the screen and keypad again, with a few connections needing connection before everything worked perfectly.

Software:

For the code, I decided to start with a T9 texting program and work outwards from there. To begin, I created a dictionary of the keypad buttons, with each value being a list of the characters accessed by hitting the button. The center of the program is a loop that waits infinitely for a key to be struck, saving the key’s digit when it happens. The dictionary can be used to print the appropriate character, with the key digit as the first index and the amount of times the key has been hit consecutively as the second. To see if a keypress is consecutive, each digit is saved to a lastvalue parameter: if a new press matches the lastvalue, the key is consecutive and the second index is incremented. Since the LCD screen can only print one character at a time, I set the cursor to begin printing at a certain location, and increment to the right any time a new key is hit. While waiting for a keypress, the lastvalue is reset to nothing after a constant amount of time, in case the next letter comes from the same key as the previous one (as in “nom”). For backspaces, I set an arbitrary backspace key (I used pound) and added logic that would move the cursor backwards and print a blank space if the key was hit. This took care of all the typing functionality.

Next, I determined the rules of the game. I decided to make it a simple typing challenge: a word is printed on the screen and the player types it out, with the reward being a point and a new word. The player has a set amount of time to type as many words as possible, with the game ending immediately as soon as the timer runs out. The timer can be extended by typing words flawlessly (no backspaces), with every three perfect words in a row adding a second for each word (six perfect words: six seconds). Breaking the streak brings it to zero, adding suspense for higher streaks. With the typing aspect completed, this left the word generation, score, timer and streak system left to implement.

For the word generation, I created a list of words, and had a loop create a second list of Boolean Falses for each of the words in the first. To select a word, a random index is drawn within a while loop, and if the second list entry for that index is false, the word is loaded and the loop breaks. That entry is then set to true, to avoid repeat words.



To compare what the user has typed to the drawn word, I created a guess list the same length as the word, with each typed letter assigned to the appropriate list spot. After a letter is added, the list is converted to a string and compared to the word, with a new word being selected and the score being incremented if they match. If the user’s guess exceeds the length of the word, the additional letters are ignored, streamlining the list management.

The timer was the trickiest aspect to implement, as I wanted to print the remaining time to the screen and the screen’s cursor can only be in one spot at a time. While I initially tried multithreading, I realized the program spent most of its time waiting for a keypress, so I just took a time.time reference point at the start of the program and printed the seconds between that and the current time.time while waiting for a keypress. The entire program is contained within a while loop that ends when that difference exceeds the time limit, which is then increased if the player gets a streak.

With the bulk of the program completed, I spent time testing for bugs and polishing, adding an attract mode with control explanations. Everything worked as expected.



**Outcome:** This project was excellent practice in hardware experimentation, wire management, and designing a program with many intricate pieces working in tandem. The software design section is rather sparse compared to the actual process; my code went through many iterations before resembling the final version, making the legibility extremely important as it grew larger and more complex. I’m very satisfied with the final version, though I would like to add a high score system.