<u>08/24/22:</u>

Potential project ideas:

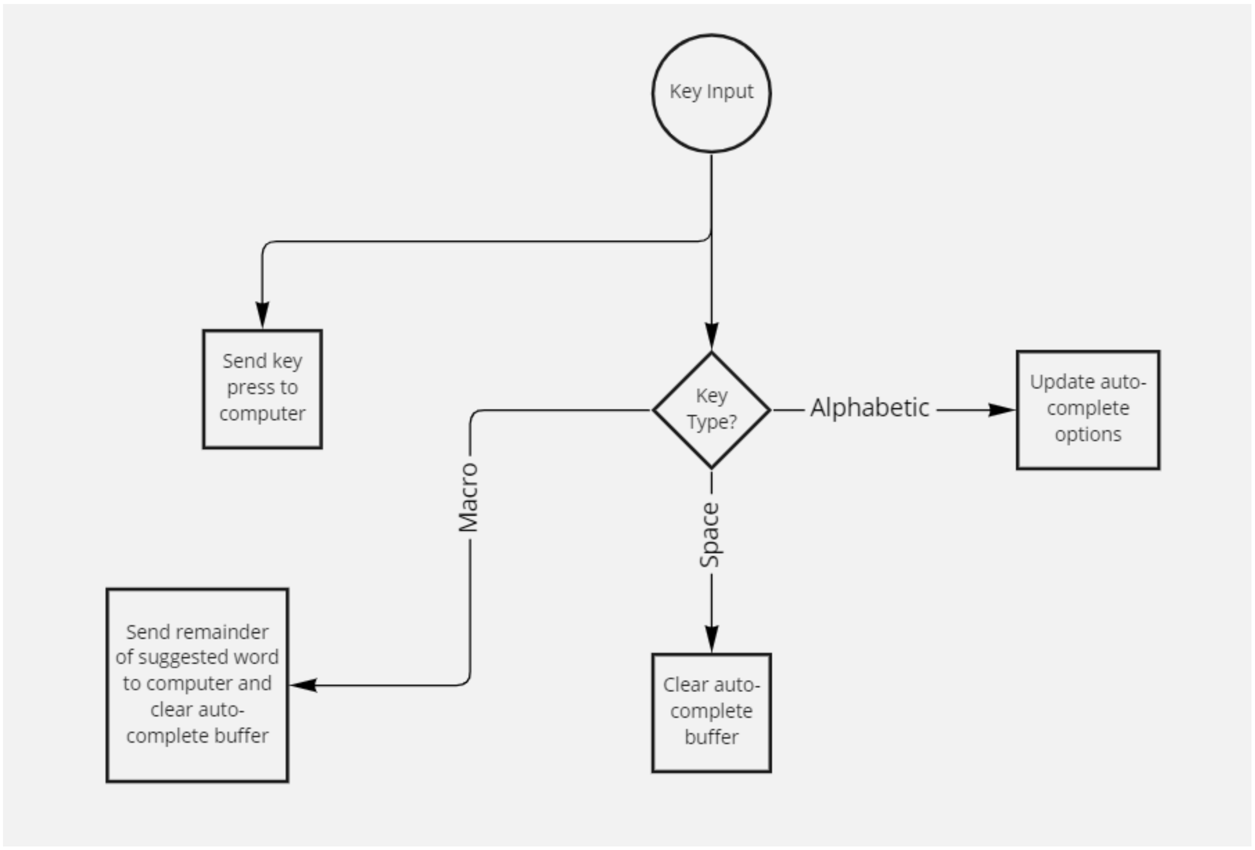
- Sheet music reading software/playback
- Mushroom fruiting chamber
- Aeroponics

<u>09/14/22:</u>

- We were planning to meet with representative from disabilities services, but the representative did not show up
- The group is planning to meet with somebody from a location in Bryan to discuss struggles with Parksinon's

<u>09/17/22:</u>

- Determined general flow of program, responsibility of various stages of implementation
    - Hardware responsible for detecting key presses, microprocessor decodes input from hardware to determine which key has been pressed and updates autocomplete recommendations accordingly


Key Input
Send key press to computer
Key Type?
Alphabetic
Update auto-complete options
Macro
Space
Send remainder of suggested word to computer and clear auto-complete buffer
Clear auto-complete buffer

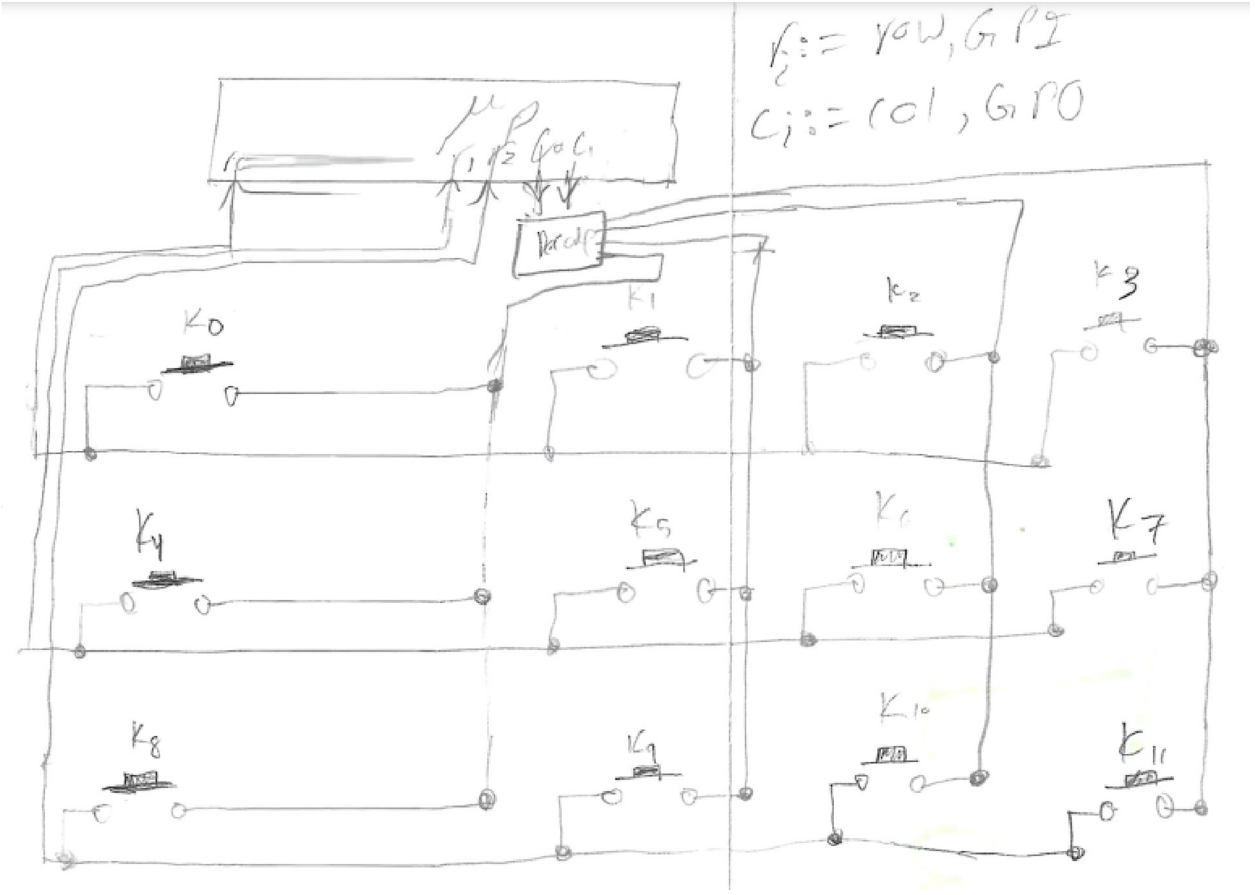
<u>09/21/22:</u>

- Jackson and I had an in-depth discussion about how our autocomplete recommendations should be generated and how our trie should be structured
    - It was mentioned that the trie should be generated during compilation

    - The trie should implement some form of priority amongst siblings
    - Nodes should contain the priority information, along with which letters can possibly follow the current letter, as well as whether or not the current node can be the terminal letter in a word
    - Decided that case-sensitivity was not something that we need to handle
- I discovered that we will need to aqcuire additional off-chip flash for our Raspberry Pi Pico in order to store our entire Trie and have enough storage for the rest of our implementation
    - Potential option: https://www.mouser.com/datasheet/2/698/adet_s_a0009751514_1-2956581.pdf

09/26/22:

- As a group, we began to formulate a posiible layout for our keyboard, as this is needed for me to finalize the circuitry design
    - Andrew began to come up with a concrete design
- I also began to research how scancodes are typically generated and handled, so we can begin to figure out how our computer will need to format key press information before sending it to a computer
    - We discussed having the microprocessor generate scancodes directly from the hardware input
- Lastly, I designed a miniature circuit to handle key press detection
    - It works by outputting a voltage at each column individually, then reading each row to check whether or not a HIGH signal was detected

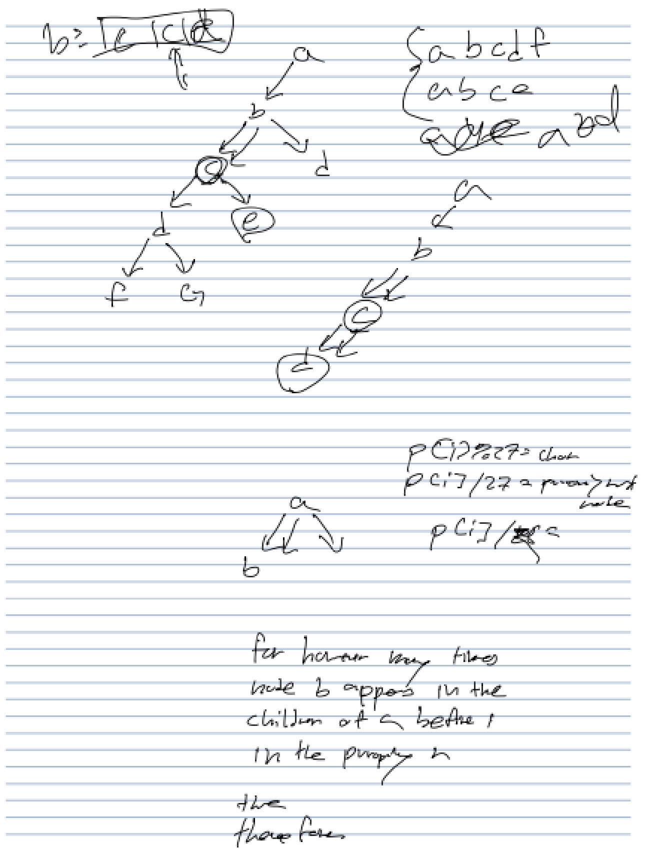


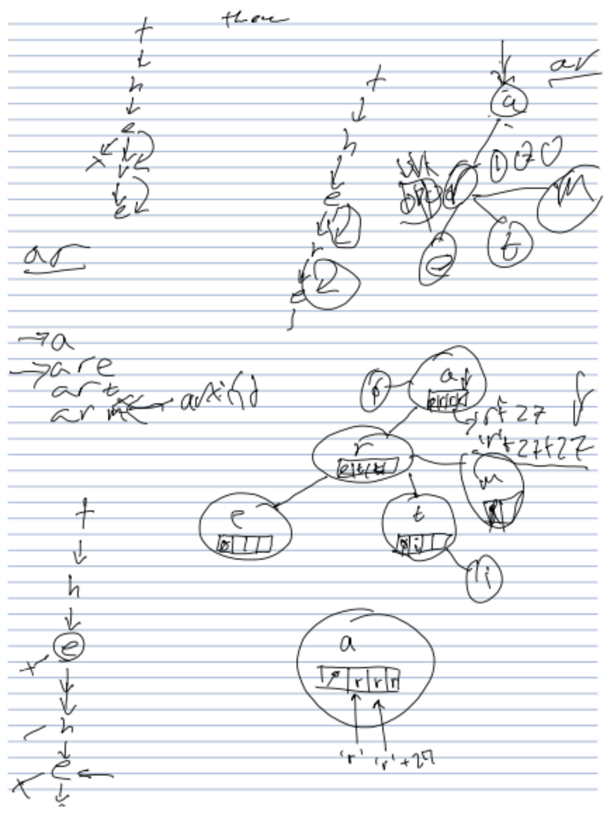
- The decoder will reduce the number of GPIO pins used to send signals to detect key presses
    - Given M rows and 2^N columns, only M + N pins will need to be used, instead of M + 2^N
- Additionally, debounce circuitry will likely need to be added to this design

<u>09/28/22:</u>

- Today, I mentioned to the team that I think we should slightly alter how we hande keyboard input. My proposed solution is to have a secondary microprocessor that polls the keyboard in order to read key presses, generates a scancode, and then sends an interrupt to the primary microprocessor
    - This design keeps the two functionalities separate, and it keeps the primary microprocessor from having to continuously poll for input
    - This would also make it easier to do software debouncing without significantly interfering with the execution of our predictive software
- Jackson brought up some potential issues with the way our algorithm makes autocomplete suggestions, so he and I came up with a better way to construct our trie and find autocomplete suggestions
    - Our new design gives more optimal suggestions to users, because it does not prevent paths involving the same child nodes

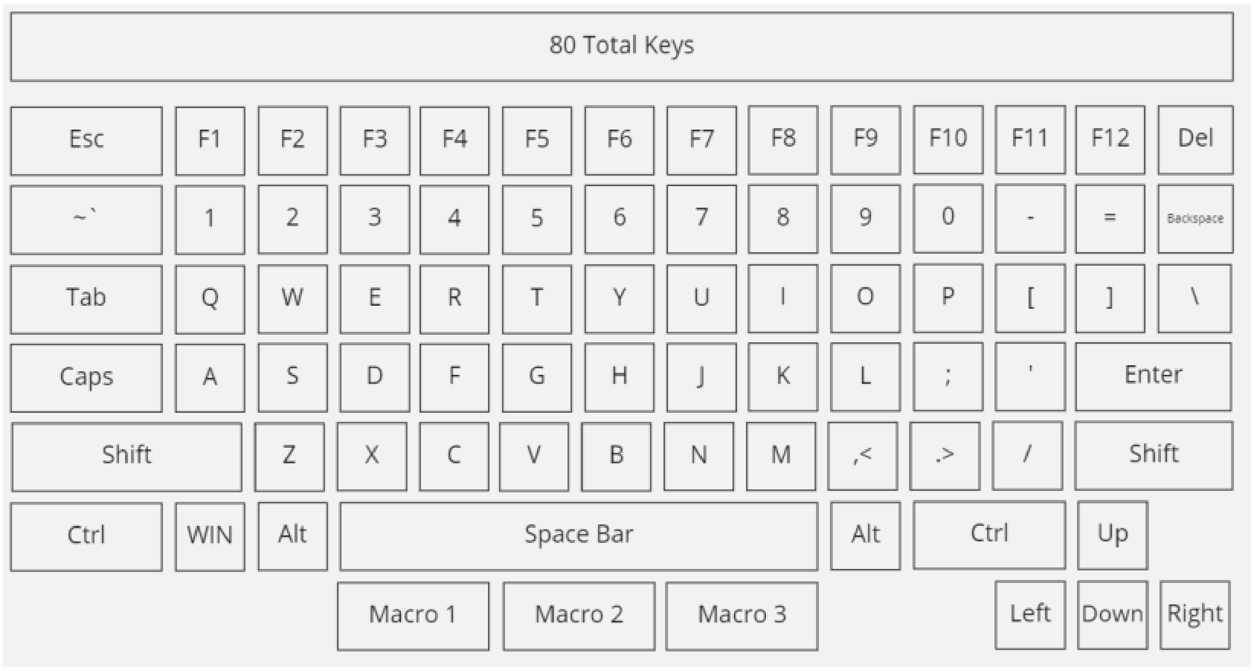
        - When the trie is constructed, the priority list is now allowed to contain duplicates, but each successive duplicate is now offset by 28 (the number of characters that each node must keep track of)
            - For this scheme, modulo is used to determine which child to visit next
            - Additionally this allows one to determine, through integer division, which priority slot of the current node's child to look at for the next suggestion (which grandchild to visit, given a child node)
            - This trie generation scheme now guarantees that the three most frequently used words beginning with the current substring are the words that are recommended
- Below are some drawings made by both Jackson and me while developing our new trie generation and traversal algorithms:





09/29/22:

- Andrew finalized our keyboard layout today, paring it down to 80 keys, per my suggestion
- 80 keys makes it easier cleanly map the layout to the keyboard matrix circuitry and reduce the number of GPIO pins used


80 Total Keys
Esc F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 Del
~` 1 2 3 4 5 6 7 8 9 0 - = Backspace
Tab Q W E R T Y U I O P [ ] \
Caps A S D F G H J K L ; ' Enter
Shift Z X C V B N M ,< .> / Shift
Ctrl WIN Alt Space Bar Alt Ctrl Up
Macro 1 Macro 2 Macro 3 Left Down Right

<u>10/03/22:</u>

- Today, the whole group discussed in greater depth how we would actually represent the key press information being sent from the keyboard
    - We developed a 2-byte encoding scheme, MENA Encoding Not ASCII (MENA), that uses the first 4 bits to encode whether ctrl, alt, shift, or Windows are pressed, and the rest to encode other keys
        - This format is beneficial because it allows us to easily determine whether or not a key press needs to be fed into the autocompletion unit, and it allows for easier generation of scancodes
        - We also discovered that there are multiple sets of scancodes in use, and tentatively decided on one to begin implementing
- I wrote a general outline for converting key presses into MENA values:

```
// Array to map key coordinate to associated MENA value
const unsigned short MENA_MAP[5][16] = {
    {/* MENA vals of row 0 keys */},
    {/* MENA vals of row 1 keys */},
    {/* MENA vals of row 2 keys */},
    {/* MENA vals of row 3 keys */},
    {/* MENA vals of row 4 keys */},
};

const unsigned short KEY_MASK = 0xff00;

const unsigned short CTRL = 0x8000;
const unsigned short ALT = 0x4000;
const unsigned short SHIFT = 0x2000;
const unsigned short WIN = 0x1000;

const unsigned short KEY = 0x00FF;

const unsigned int WIDTH = 16;
const unsigned int HEIGHT = 5;

void get_key_press() {
    unsigned int mena_val = 0;
    unsigned int key_press = 0;
    while (true) {
        // reset mena_val
        mena_val = 0;

        for (unsigned int col = 0; col < WIDTH; col++) {

            // Write HIGH to column <col> of scanning matrix
            gpio_write(col, HIGH);

            for (unsigned int row = 0; row < HEIGHT; row++) {

                // Check for key  press at (<row>, <col>)
                if (gpio_read(row, HIGH)) {

                    // Get associated MENA value
                    key_press = MENA_MAP[row][col];

                    // If non-special key is pressed, overwrite
                    // (intended to handle multiple key press
problem, but may not work)
                    if (is_non_special(key_press)) {

                        // Mask key values for overwrite
                        mena_val &= KEY_MASK;
                    }

                    // Bitwise or to update mena_val
                    mena_val |= key_press;
                }
            }

            // Set output to LOW if necessary
            gpio_clear(col);
        }

        // If key(s) is/are pressed, send interrupt w/ MENA value

        if (mena_val != 0) {

            generate_interrupt(mena_val);
        }

        // Add delay
        }
}
```

10/05/22:

- The group is currently experiencing shipping delays, which has caused us to be slightly behind on our hardware development

10/15/22:

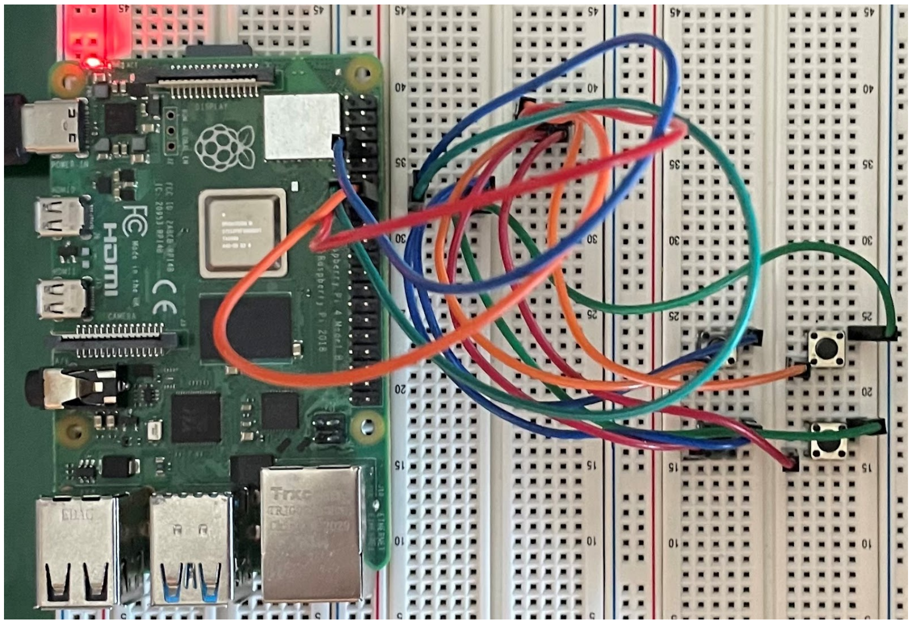
- Now that our parts have arrived, we have been trying to program the Raspberry Pi Pico in C, but have run into many issues trying to get the proper environment set up
    - Several group members met today to work on this issue with little success
- Additionally, our switches have a protrusion that prevents us from using them on a breadboard, and the shape of the pins made it difficult to connect to the female ends of any jumpers
    - To test the key scanning circuitry, we are using push buttons that fit into a breadboard

10/16/22:

- I got a working environment for programming the Pico in C, but am still having some trouble with programming the GPIO pins to accept inputs
- I also wired up a 2x2 prototype of our key scanning matrix. Since we were having issues

10/17/22:

- Jackson and I worked on programming the GPIO pins in C on a Pi 4 to get the 2x2 key scanning matrix operational and recorded a demo for our CDR
- There was an issue with the matrix reading in the wrong key in a specific case that we were not able to diagnose. This did not occur when we used Python on the Pi 4.



10/21/22:

- After struggling to get GPIO working fully in C, we decided to use MicroPython to program the Pico because there seems to be better support and it is easier to use
- The group finalized our decision to do our word suggestion through a Raspberry Pi 4

10/24/22:

- Today, I got a 2x4, then a 3x4 key scanning matrix fully operational on our Pico using MicroPython and began experimenting with sending an interrupt, along with data from the Pico
- Jackson was able to get the Raspberry Pi 4 properly generating scancodes and working as a functional keyboard when plugged in through USB

10/26/22:

- I got the Pico successfully generating an interrupt, along with data to another Pico through UART, but there were issues moving the recipient code to the Raspberry Pi 4, since there is not as much support for GPIO programming in C/C++
    - We initially had some issues with the recipient Pico reading in multiple characters at a time, but diagnosed this problem as a timing issue on the sending side

```
# Pico MicroPython Code

from machine import Pin, UART
import time
import sys

uart0 = UART(0, baudrate=9600, tx=Pin(12), rx=Pin(13))

interrupt = Pin(15, Pin.OUT, Pin.PULL_DOWN)

MENA_MAP = [
    ['h', 'e', 'l'],
    ['o' , 'w', 'r'],
    ['d', 'y', '!'],
    ['t', '\b \b', ' ']
]

WIDTH = 2
HEIGHT = 4

R0 = Pin(2, Pin.IN, Pin.PULL_DOWN)
R1 = Pin(3, Pin.IN, Pin.PULL_DOWN)
R2 = Pin(4, Pin.IN, Pin.PULL_DOWN)
R3 = Pin(5, Pin.IN, Pin.PULL_DOWN)

C0 = Pin(16, Pin.OUT)
C1 = Pin(0, Pin.OUT)
C2 = Pin(1, Pin.OUT)
```

```
cols = [C0, C1, C2]
rows = [R0, R1, R2, R3]


while True:
    mena_val = 0

    for i in range(len(cols)):
        cols[i].on()
        for j in range(len(rows)):
            c = ""
            if (rows[j].value()):
                interrupt.on()
                c = MENA_MAP[j][i]
                uart0.write(bytes(c, 'utf-8'))
                interrupt.off()
                time.sleep_ms(100)

            time.sleep_ms(10)
        cols[i].off()
```

- Jackson and I discussed experimenting with serial communication through USB between the Raspberry Pi 4 and the Pico

10/31/22:
- Jackson and I were able to get the Pico successfully sending data to the Pi 4, but now need to make the communication asynchronous. We had to switch from MicroPython to CircuitPython, but we may be able to switch back if a library that I found works properly.
- The major thing left for me to do is alter our Pico code to generate MENA codes, rather that characters, and send the generated code to the Pi for suggestion generation and scancode conversion

11/02/22:
- Andrew made a prototype PCB to match the circuit that I implemented on a breadboard. We were able to connect that to the Pico and test it using the key scanning code. With the exception of some loose connections, things seemed to be working well.
- We are having some trouble setting up GPIO interrupts in C++ on the Pi

11/14/22:
- We received our PCB and began testing today. We hit several roadblocks due to some errors in the PCB design, but they should be things that can be resolved. One of the

issues is that all of the unused pins are grounded, causing certain pins that should not be shorted to be shorted.
- There also seems to be an issue occurring with either the decoding circtuiry or the portion of code responsible for selecting which column to power, but further debugging will be required to determine which is the source of the issues we are seeing.

11/21/22:
- Brittany and I did some testing of the decoding circuitry and determined that that is not the source of the column selection issues that we noticed last week. After taking a closer look at the code, Jackson and I found a minor logical error that was causing a conditional to be mis-evaluated.
-

11/23/22:
- Today, I fully integrated each of the components of our keyboard, with the exception of the LCDs. Additionally, I was able to set up the Pi to automatically start our driver when it receives power, allowing the keyboard to start working once it is plugged in. It does take a little bit of time for the Pi to start, so this is something that may need to be fixed.
- I also came up with a better way to transmit MENA data from the Pico to the Pi. The Pi now runs a bash script that pulls from the serial port (/dev/ttyACM0) and pipes the data into the driver.
- Lastly, I found a bug that occurs when only a modifier key is pressed that results in a combination of keys being typed. This should be simple to fix once I meet with Jackson.

11/27/22:
- We met as a group today, and Jackson and I were able to resolve a lot of the bug we found. Additionally, we were able to integrate the LCDs, allowing us to fully make use of our auto-complete functionality.

11/30/22:
- While testing the keyboard, we noticed some latency issues. These were fixed in software by keeping track of key presses in a slightly more nuanced manner. The Pico now keeps a counter that tracks how long a key is held and can only send repeated key presses to the Pi after a set period of time. This handling enabled us to make the keyboard more responsive, greatly improving the observed latency.