Smart Chess

The Move Assist and Cheat Resist Smart Chess Board

Donald Elrod
Texas A&M University

What is Smart Chess?

Game tracking software keeps track of the game progression

- The board will alert if invalid move is detected or if pieces are moved out of turn

LED indicators under each playable tile show valid moves for each piece

Design Considerations

Circuit Design

Circuit Fabrication

Program Design

Device Communication

Components

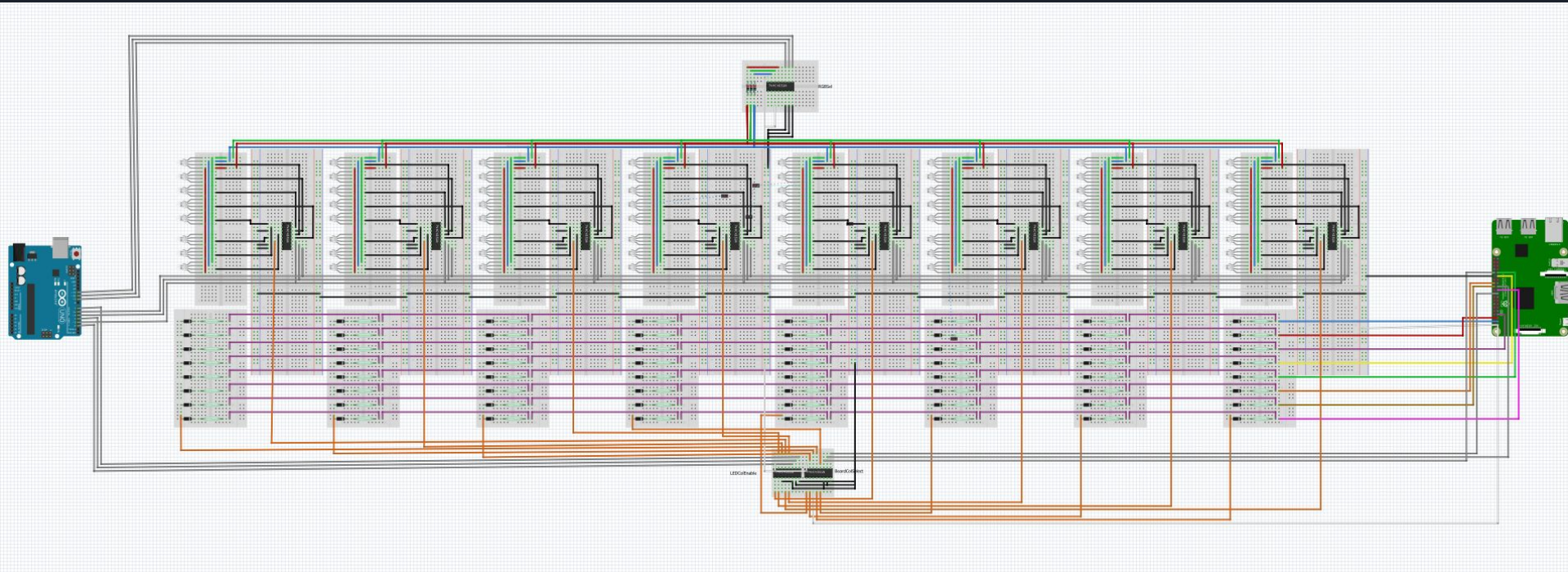
1 x	Raspberry Pi V3
1 x	Arduino Uno
64 x	reed switches (but they break easily so realistically ~80)
64 x	RGB or single color LEDs
11 x	8-to-1 multiplexers
A lot of	wire

Circuit Design

The circuit design consisted of 3 parts:

1. Creating the LED matrix and minimizing GPIO pins required to do so
2. Creating the reed switch matrix to keep track of where the pieces are on the board
3. Actually building the physical circuit

Circuit Design

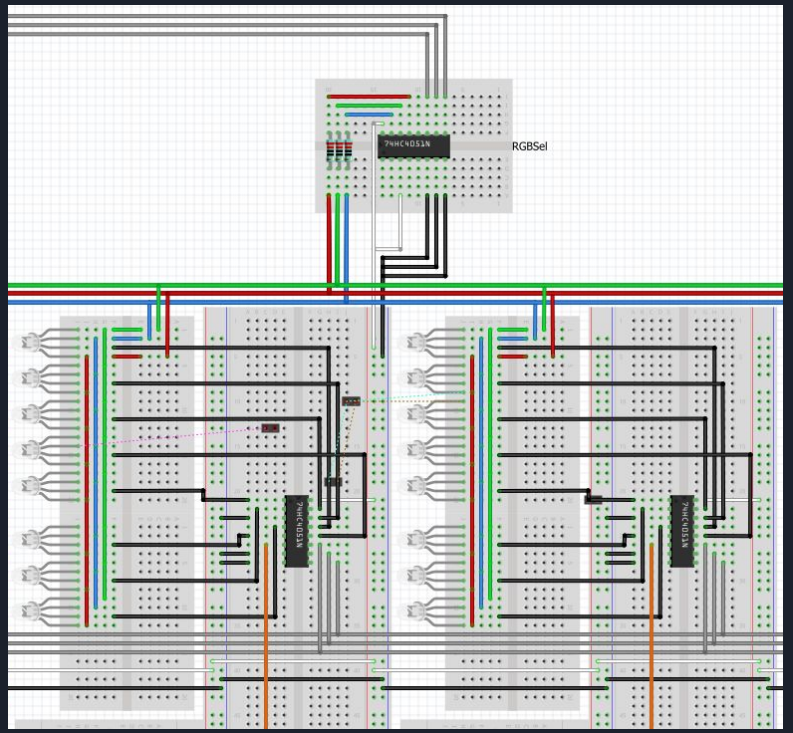


Circuit Design - Controlling the LEDs

The Arduino's job was to communicate with the LEDs, using a system of multiplexers

The top multiplexer controls what color the light should be, and the multiplexers below control which light turns on

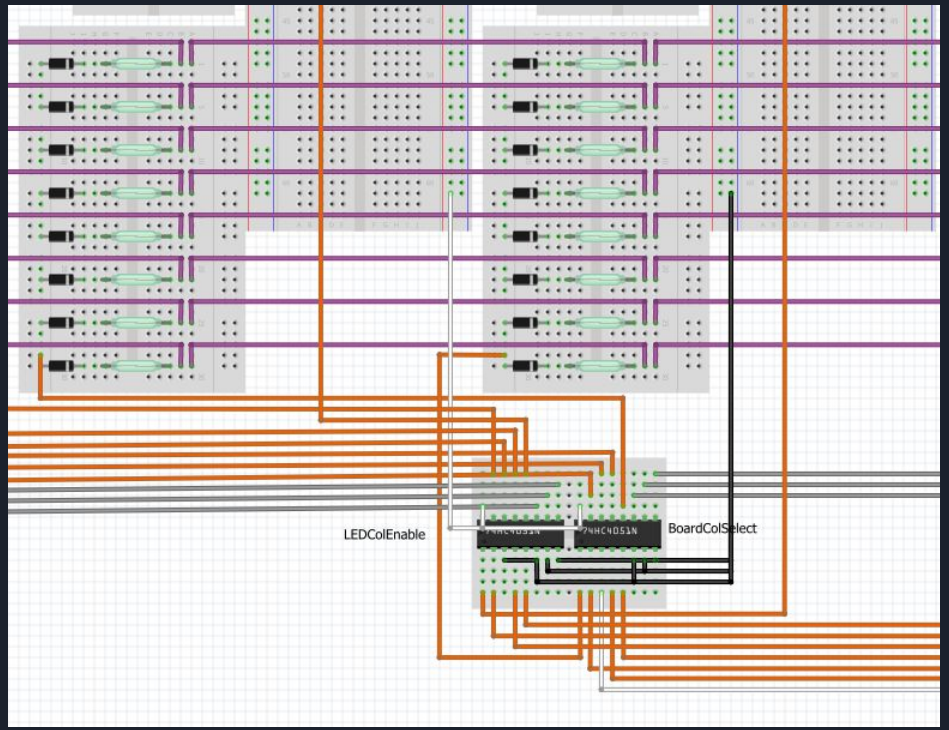
This allows each row of light to be toggled and colored individually



Circuit Design - Controlling Individual LEDs

The multiplexer on the left toggles the state of the multiplexers that control which rows are enabled, allowing for each LED on the board to be toggled individually

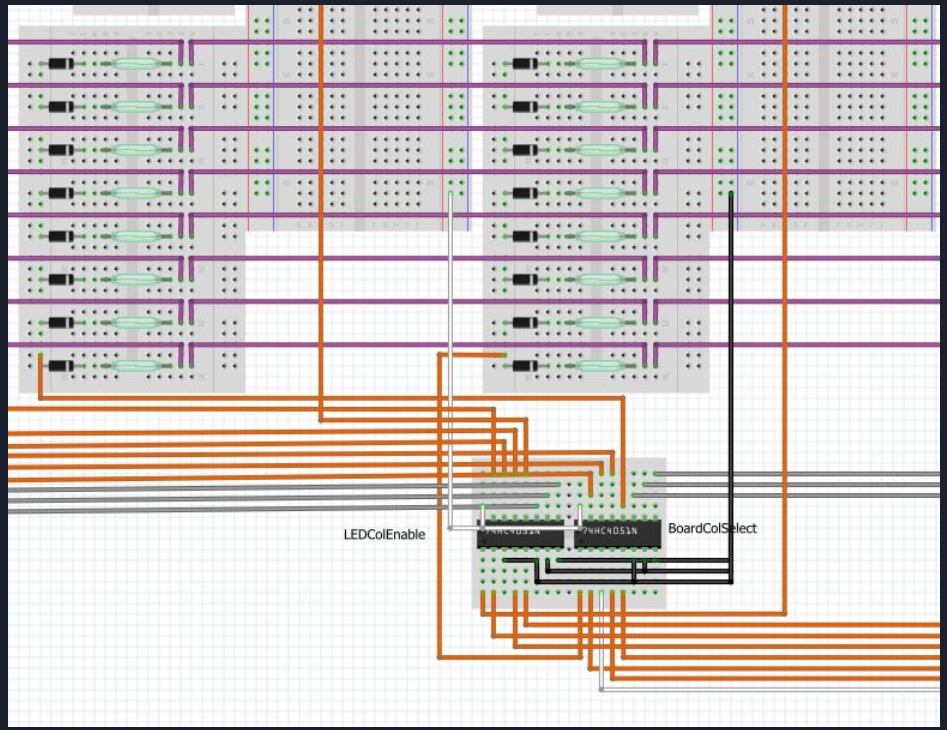
Using these layers of multiplexers, control of all 64 LEDs was possible with only 9 GPIO pins on the Arduino



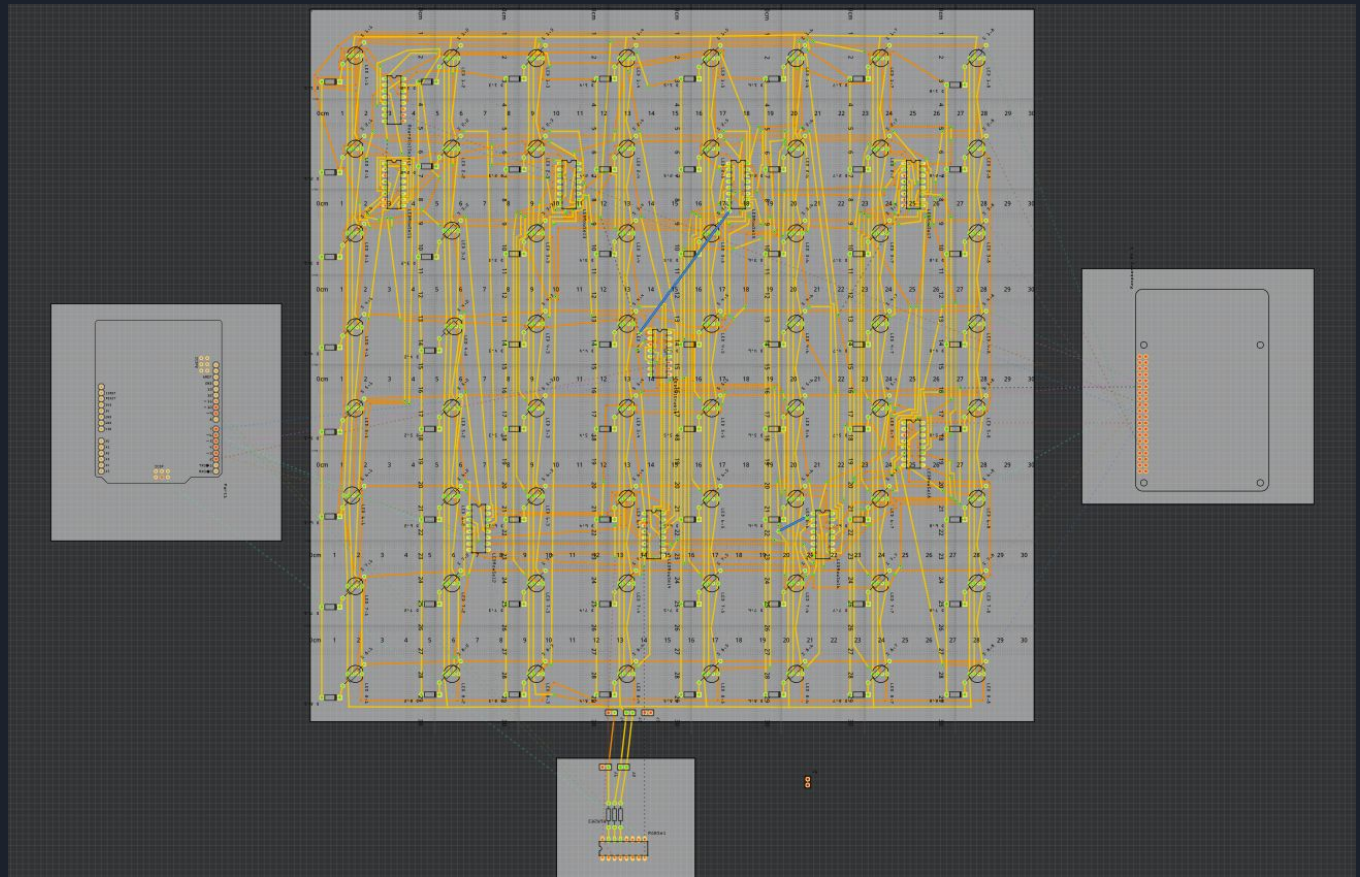
Circuit Design - Reading in Piece Positions

The multiplexer on the right is controlled by the Raspberry Pi, and allows for power to be selectively sent to the rows of reed switches, which activate when the magnets in the pieces are above them

This allows the python script on the Raspberry Pi to cycle through the columns of reed switches and read which locations are occupied by pieces



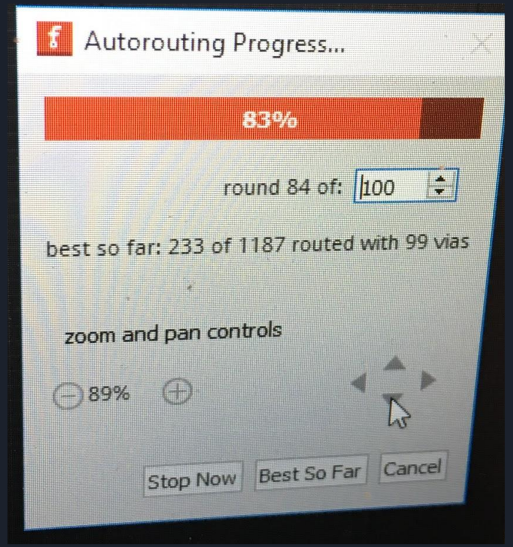
Circuit Fabrication - The Failed Attempt



Circuit Fabrication - The Failed Attempt

Lessons learned

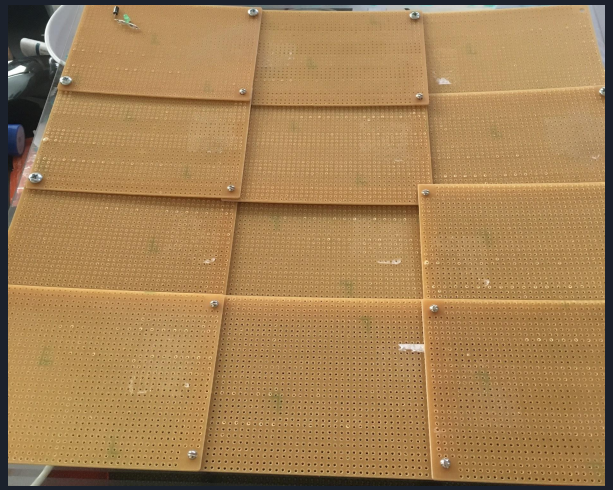
- Always check fabrication constraints before designing a large scale schematic
- Fritzing is a great program, but does not scale well and other alternatives should be used for designs of this scale



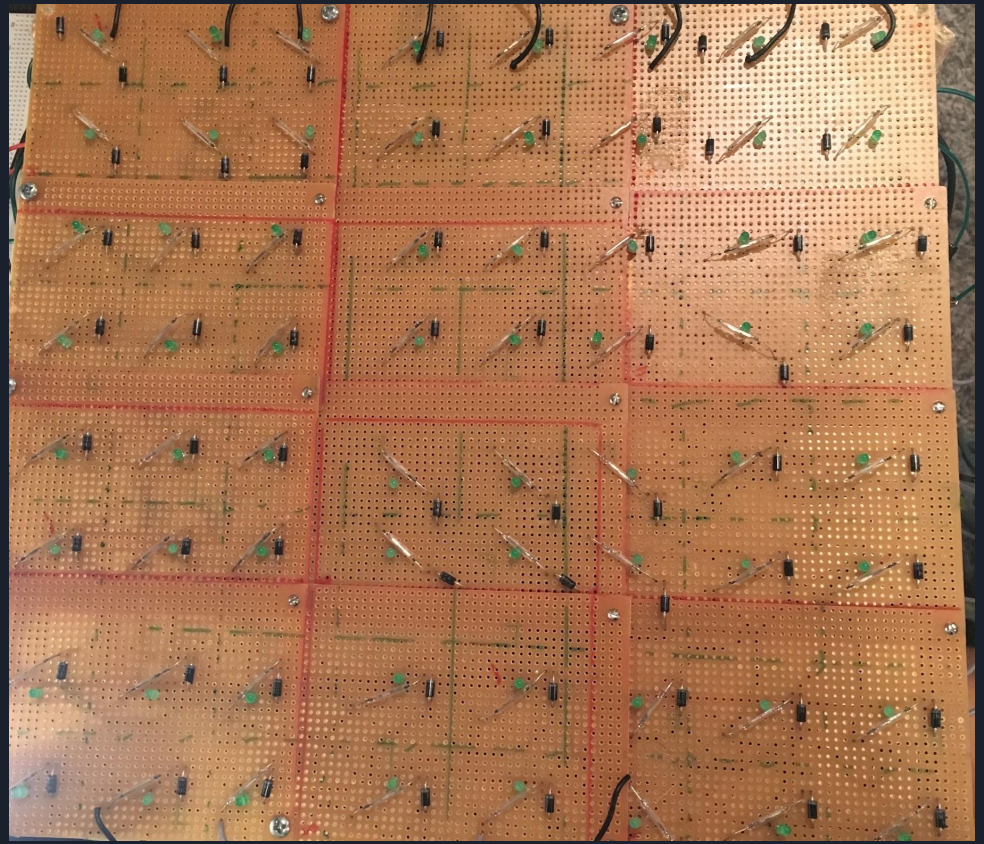
Circuit Fabrication - The Successful Attempt

Utilizing small breadboards from our local electronics supplier, EPO Houston, we created the board out of 12 separate 3” x 4” PCBs

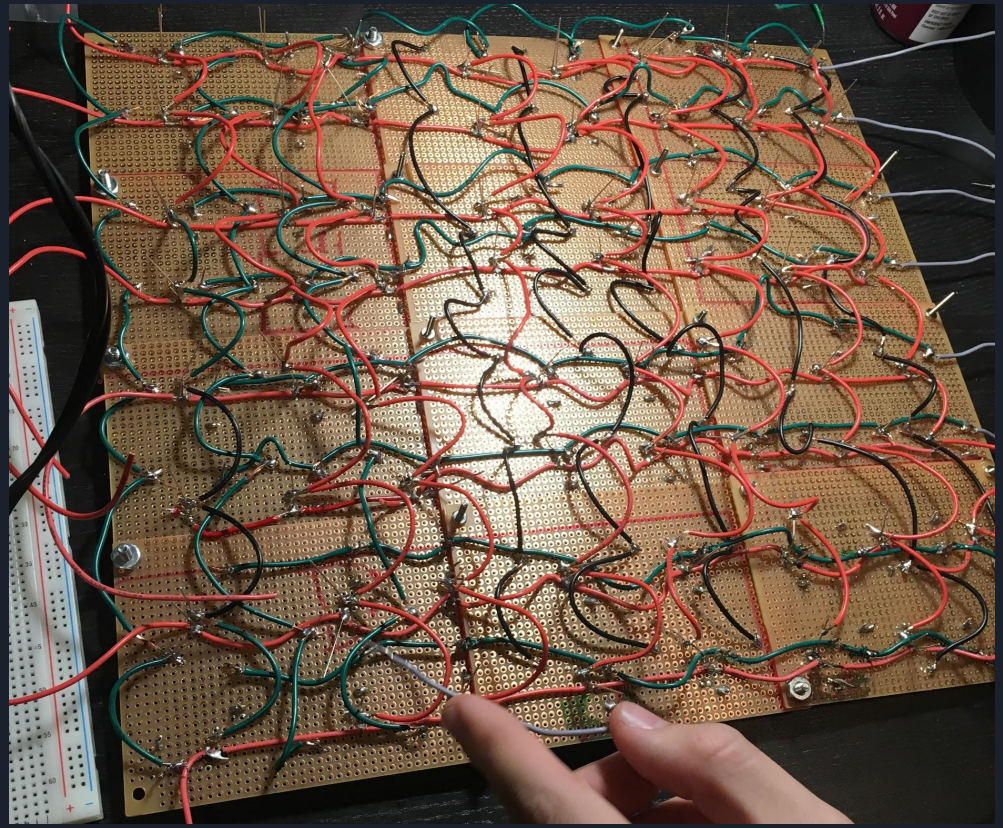
Due to not having a printed PCB, all components needed to be hand-wired and placed so that they were offset an even height from the board



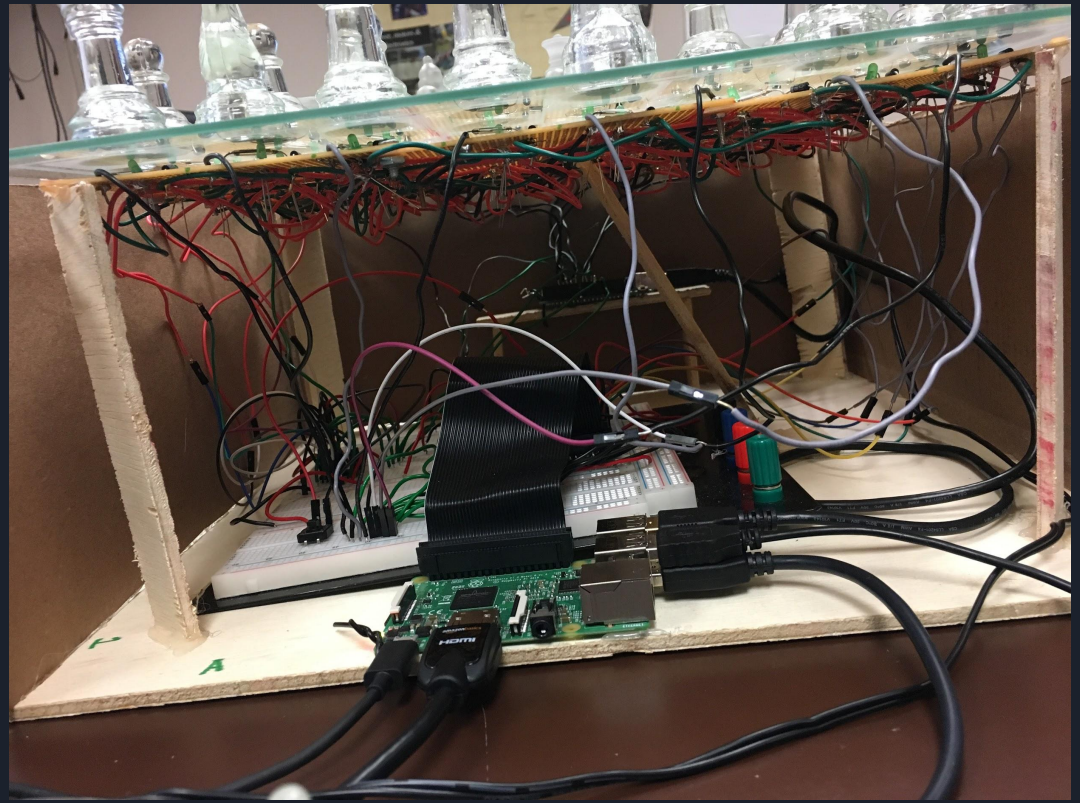
Circuit Fabrication - The Successful Attempt



Circuit Fabrication - The Successful Attempt



Circuit Fabrication - The Successful Attempt



Circuit Fabrication - The Successful Attempt

Program Design

The program design consisted of 3 parts:

1. Designing the logic for controlling the LEDs from the Arduino
    a. Best way to move through the matrix of LEDs: column by column or row by row
    b. Maximum frequency lights can/should be flashed
2. Chess board logic on the Raspberry Pi
3. Reading the state of the physical board from the Raspberry Pi

Program Design - Arduino LED Logic

In order to find the maximum rate to cycle through the LEDs, the order of multiplexer activation needed to be optimized

- We wanted to cycle the lights at a high enough frequency to make all possible moves appear as a solid light, which is why we used the Arduino in the first place
- The most efficient way to crawl through the board and light up individual lights is to select the column, and then cycle through the rows.
- We were unable to cycle through the LEDs fast enough to make them appear to be on simultaneously due to the delay in the multiplexers
- The solution used was instead blinking each LED for half a second

Program Design - Chess on Raspberry Pi

The chessboard logic was adapted from a python3 library called ***python-chess***

Interfacing with this library was key for keeping track of the game state and finding available moves

Program Design - Reading the Board State

The board state is read by cycling the GPIO pins to change the state of the multiplexers

The outputs of all 8 rows are routed to additional GPIO pins, allowing the board to read the state of each column individually

The board is polled continuously at a rate of 2 Hz, allowing for almost immediate board updates

```
#-------------------poll pins--------------
switch0 = gpiozero.Button(18, pull_up = False)
switch1 = gpiozero.Button(4, pull_up = False)
switch2 = gpiozero.Button(17, pull_up = False)
switch3 = gpiozero.Button(27, pull_up = False)
switch4 = gpiozero.Button(22, pull_up = False)
switch5 = gpiozero.Button(6, pull_up = False)
switch6 = gpiozero.Button(13, pull_up = False)
switch7 = gpiozero.Button(19, pull_up = False)

mux0 = gpiozero.LED(21)
mux1 = gpiozero.LED(20)
mux2 = gpiozero.LED(16)
```

Device Communication

Communication between the Raspberry Pi and Arduino was done over serial communication, using the standard Arduino Serial communication library and the python serial library

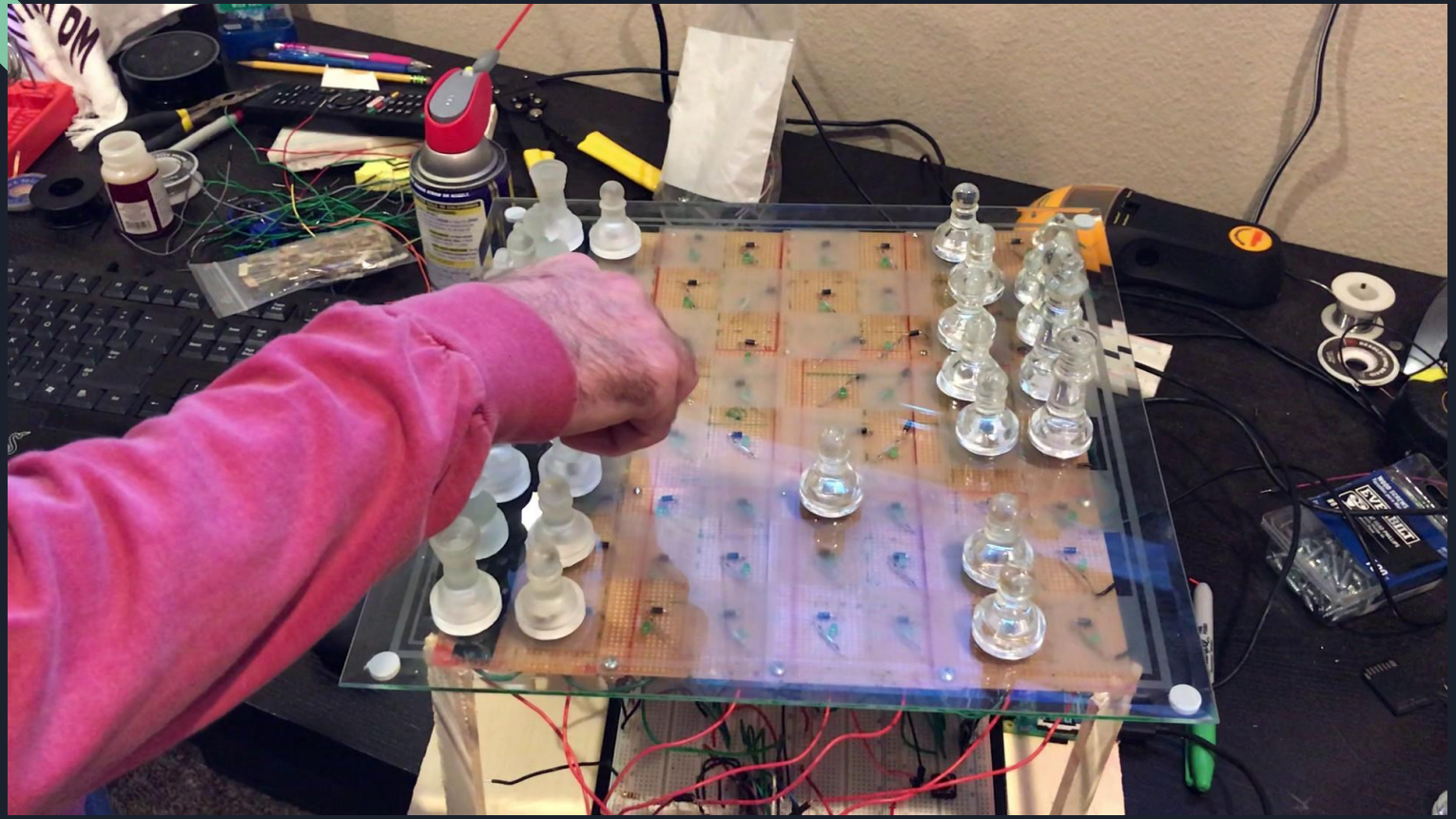
The Raspberry Pi sends the potential moves as a 64 byte string of locations to light up to the Arduino over the serial connection

```
#convert true false array to arduino string
def toArduinoString(board):
    s = ''

    for b in board:
        if b == True:
            s += '1'
        else:
            s += '0'

    return s
```

The Finished Product



Questions, Comments, Concerns?