ECE 445

Senior Design Laboratory

Final Paper

The Auto Board

Team #23
Rafal Czech
(rczech2@illinois.edu)
Kevin Villanueva
(kevinmv2@illinois.edu)
Nicholas Rappe
(nrappe2@illinois.edu)

TA: Dean Biskup (dbiskup2@illinois.edu)

December 8, 2021

Contents

Introduction

1.1 Problem and Solution Overview

People with physical disabilities can find difficulty when participating in family game night. Anyone with restrictive moving or difficulty of control can struggle to move the many pieces on some of the most famous board games like Checkers and “Chutes and Ladders” [1]. Those with disabilities can feel alienated when having to rely on others to assist in participating in games like these. In nursing homes alone, almost one quarter of the elderly 85 and older are disabled with many more having limited movement [2]. On average nursing homes have one staff member for every six to eight residents [3], so something as simple as managing a board game between residents can become a tedious task compared to their other responsibilities.

To solve this problem we introduce, “The Auto Board”, which is a voice controlled, automatic game board that, once the pieces have been set up, would allow for seamless game play without physical contact or assistance. The player’s voice commands will dictate how the board moves its pieces. This would allow for anyone to participate regardless of physical capability and feel as if they are playing for themselves rather than relying on someone else to make their moves. This can be most effective in settings such as nursing homes and hospitals where there is an increased frequency of disabled people and not enough caretakers to accommodate all their needs. While our product is mainly designed as a tool to help physically limited people play board games, it would also bring back the novelty of family game night as well as elevate the experience for all.

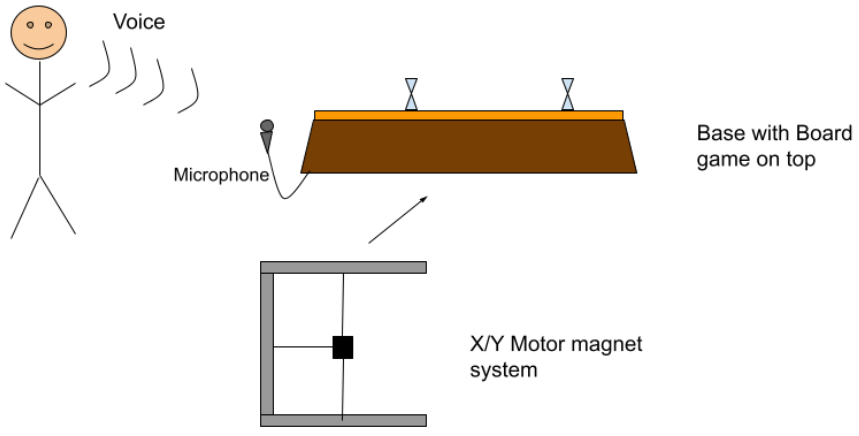

Figure 1 : Visual Aid for the Auto Board

Figure 2 : Top View of Chutes and Ladders

1.2 High Level Requirements

After the initial setup of the game and its pieces, users will be able to utilize voice command functionality to play the chosen game. They will not have to physically interact with the game board until the selected game is over, and the next one needs to be set up.

The product must be able to take the player's voice command in the input form “Move (Piece) at (location) to (location)” for all moves in the game as well as commands such as “roll the dice” or “spin the wheel”.

The final product must be able to support at least two games. Depending on which game the user selects, the product will be able to perform the legal moves associated with each.

Design

2.1 Block Diagram Overview

The project will require three high-level areas as outlined in Figure 1. These blocks are the following: voice, gameboard, and power. Each blue block in Figure 1 contains subblocks that represent the subsystems, logic, and physical components required to execute the desired

functionality of the project. The final block diagram is the result of eight individual revisions that consisted of adding and subtracting parts as the design process progressed.

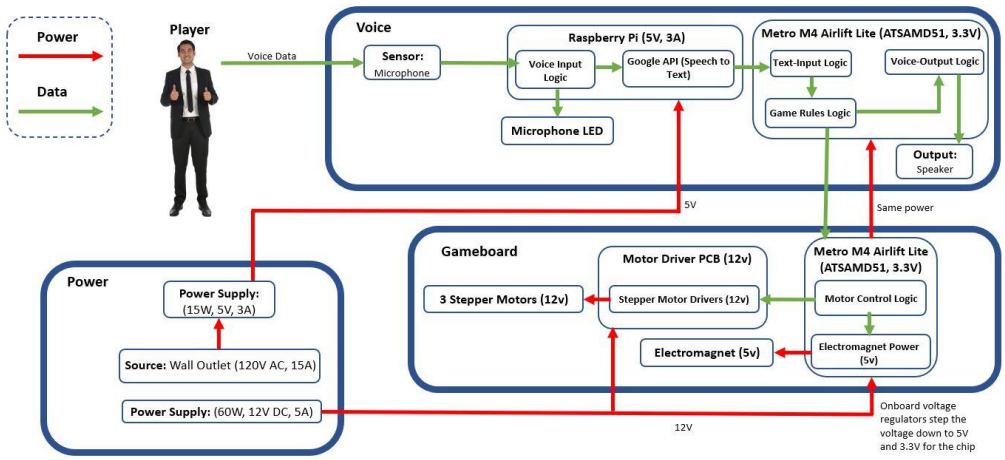

Figure 3 : Block Diagram of the Auto Board

2.2 Power Block

The “Power” block will be necessary to transfer adequate power to the various components in our system. A 60W PSU will be the main component required for stepping down the wall voltage to a 12V, 5A source. This power is then split up among the 12V and 5V voltage regulators which were hooked up to the Motor Driver PCB and Electromagnet respectively. The Metro M4 Development Board[4] had onboard power regulators so it took the full 12V input, then stepped it down to 5V for additional outputs. The Raspberry Pi[5] was powered individually by a Dell 15W PSU which delivered 5V, 3A from the wall. It was necessary to use this extra power adapter due to the singular USB-C power input on the Raspberry Pi board.

2.3 Voice Block

The “Voice” block will be necessary for taking in the player’s voice as a data input and converting it into text. This will be done by using the Wi-Fi accessible Google Speech-to-Text API[6]. The voice data will be recorded as a WAV file which is an uncompressed audio file. Voice data will then be converted into a base64 encoded string so that it is in the correct format to be transferred to the Google Speech-to-Text API. Once the API has finished processing the data it will be back over Wi-Fi to the Raspberry Pi 4 as a text version of the spoken command, which will be sent back to the ATSAMD51J19 Microcontroller[7] via a serial connection to determine where the game piece is moved to be moved to.

2.4 Gameboard Block

The “Gameboard” block contains all physical functionality required to move the motors and electromagnet. The Motor Driver PCB is contained within this block because it hosts the three Easy Drivers[8] which are responsible for sending the power and movement commands to each motor. The Metro M4 Airlift Lite is responsible for giving the motor drivers the movement commands which are passed along to the three motors.

Design Verification

3.1 Power Block

3.1.1 [Wall Outlet]

Since the goal of this project was to design a commercial product that could be used at home, it was imperative that the system relied on a wall outlet to be powered. Using a power supply found in the lab would have been unrealistic for our end user and would likely make it an unpopular product family game night.

3.1.2 [60W, 12V, 5A Power Supply]

This power supply was originally the only one that the project was designed to rely on. The intended design would have had the 12V, 5A input into the custom PCB and then have it stepped down to 12V, 5V, and 3.3V to be used where applicable. This power supply is able to provide sufficient current to the system at maximum load as outlined in Appendix A, Table 4. By using a digital multimeter, it was also confirmed that all voltage regulators on the custom PCB were outputting voltage within their rated 4% tolerance.

3.1.3 [15W, 5V, 3A Power Supply]

Once it was clear that there was a memory limitation issue on the custom PCB, a Raspberry Pi 4 was introduced into the design in order to handle the Google API calls, and needed an external power source. The USB-C power input was utilized by connecting a standard 15W Dell laptop charger to the Raspberry Pi board and having the other side plugged into a secondary wall outlet. Although this setup requires the end-user to take up two wall outlets to play the games, in the future these devices would be consolidated so that only one outlet is required.

3.2 Game Block

3.2.1 [Stepper Motors & Magnet]

The stepper motors are responsible for the movement of the electromagnet. The AutoBoard consists of a X-Y rail grid that contains one motor on the x-axis and two on the y-axis. When the motor steps it pulls a belt moving either the electromagnetic if its the x-axis motor or the entire x-axis rail if its the y-axis motors. In combination it is able to move the electromagnet anywhere in a 18in by 18in area as seen in Figure 4. To control when the electromagnet turns on or off a simple BJT circuit was constructed. As seen in Figure 5, a 3.3V signal from the microcontroller would turn on the electromagnet by dropping EM- to 0V and thus creating a 5V difference against its polarities.

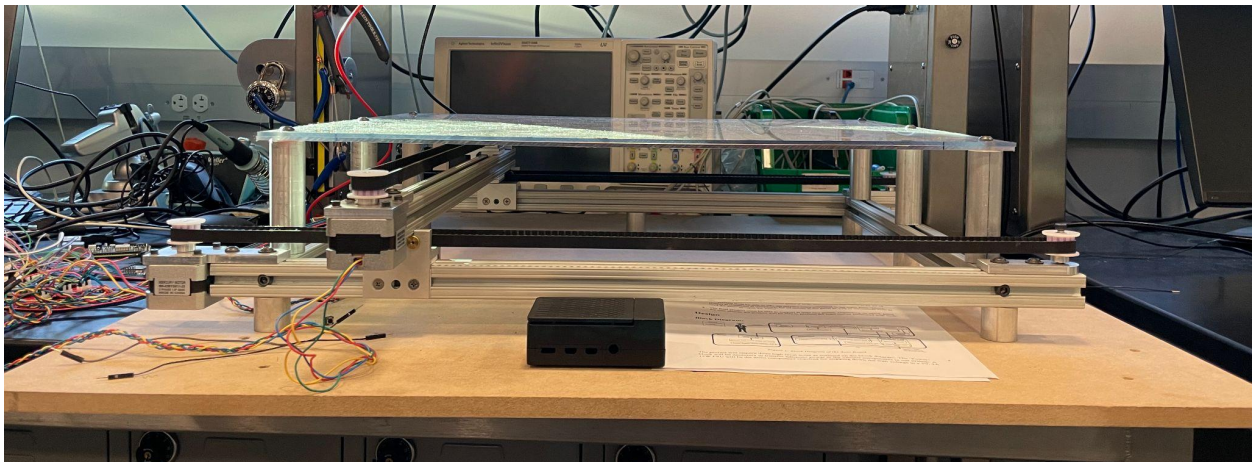

Figure 4 : Side View of Auto Board Motor System

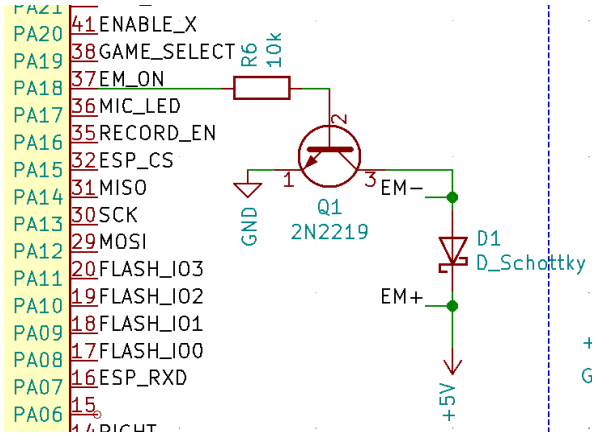

Figure 5 : Electromagnet circuit

3.2.2 [Motor driver PCB]

Two custom PCBs were designed and fabricated. The first contains most of the chips and components that are on the Metro M4 Airlift with additional parts to serve our needs. The second is our motor driver board. This was fabricated to effectively connect our motors, drivers and microcontroller together. The PCB interconnects all 12V and GND paths while providing the correct breakouts for motor coils and control pins. Since we have two motors on the y-axis, its control pins paths must be shared while still having separate coil breakouts.

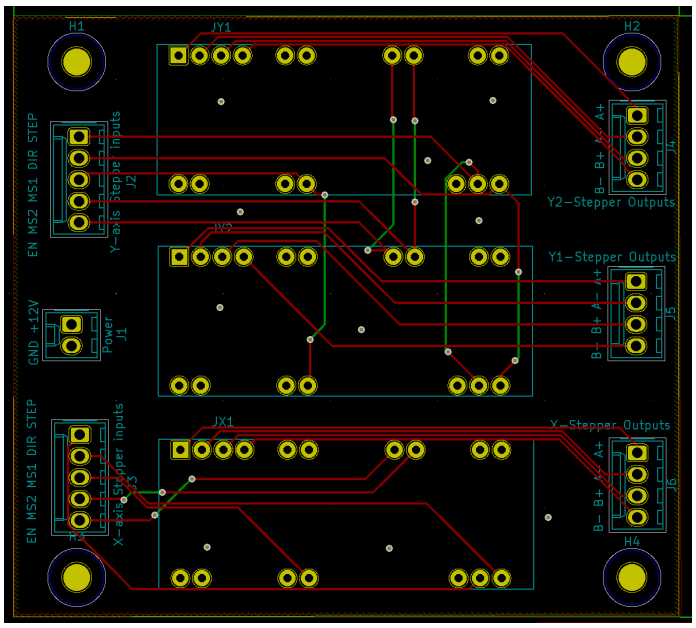

Figure 6: Secondary Motor Driver PCB layout

3.2.3 [Metro M4 Airlift Lite]

As mentioned previously, the motors and driver board are connected to the Metro M4 airlift. Once the Metro M4 has received a voice command from the Raspberry Pi it must process the command based on the current game that is being played which is determined by the current_game variable. For now we only have two games so that variable is either a 1 or a 0. The command is then checked to see if it is valid for the given game. If it isn't then the Metro M4 will send a signal to the Raspberry Pi to output the “invalid command” audio to the user. Otherwise the Metro M4 will proceed with the command. Commands are given in the format “Move Piece at Location to Location. We have a 2d array mapping each spot of the game board to certain x and y step coordinates based on the first spot being (0,0) (Appendix D Figure 9). The Game logic will determine if the piece we want to move is in fact at that location, then check if the location is available based on the 2d array and if so will proceed with moving the motors. It is here where all the motor control logic is. We created a function called “StraightLine” that is called by our game logic (Appendix D figure 10). Most of the logic is based on moving from location to location in terms of steps. Then given the current, starting, and end positions it can calculate the

step sequence of the motors to create a straight line between all spots. Factors that must be determined and represented in high/low voltages from its digital pins are when to step, direction, type of step, speed, and enable. As seen in Appendix A table 3, this motor logic is tested thoroughly as its calculations are vital for successful moves from the product.

3.3 Voice Block

3.3.1 [Microphone]

The microphone is connected to the Raspberry pi via USB. It is used to record the voice commands. We chose a microphone that has adjustable sensitivity so as to allow us to have our users stand farther away from the board and still be able to input commands. The raspberry pi was incharge of telling the microphone when to record and at what settings.

3.3.2 [Raspberry Pi 4]

The Raspberry Pi 4 is the main component of the Voice block. It acted as a secondary device that our main board could communicate with via serial communication. While not processing a command the raspberry pi sits in an infinite loop polling the serial communication to see if it receives a signal from the Metro M4. The signal sent is a string with the command that the Raspberry Pi should perform. It is a signal saying to record a voice command or to play a certain audio file from the speaker, in this case we are looking at the former (Appendix C Figure 1). Once it has received the record signal from the M4 it begins by recording audio from the microphone for five seconds. That audio is saved as a WAV file and is then converted into a Base64 text file so that it is in the format expected by the Speech to text API (Appendix C image 2). We then send a post request to the API with the audio file. The request consists of information about the audio file like the number of channels the frequency it was recorded in and the type of file it is saved as. We also include an array of strings containing all words and phrases that could be used in the commands the board can receive. This helps the api have a better understanding of what it is looking for and significantly improves the performance of the API (Appendix C, Figure 3). Once the Raspberry Pi has received a response from the API it then sends the command the API found to the Metro M4 via the same serial communication as before. Once finished the Raspberry Pi goes back to polling the serial communication waiting for the next command sent by the Metro M4.

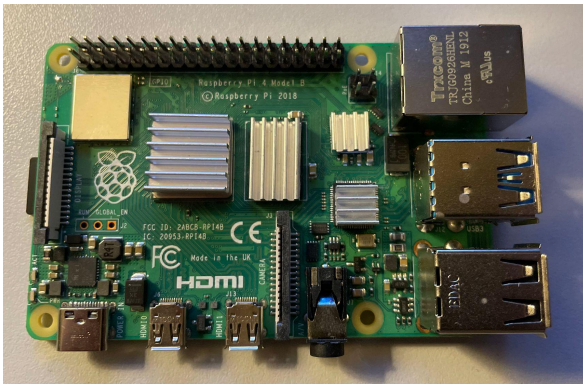

Figure 7 : Raspberry Pi 4

3.3.3 [Metro M4 Airlift Lite]

The Metro M4 is the main board that handles all of our game logic and our motor control. It also was in charge of determining when a command needed to be input based on a button. When this button is pressed, the M4 would send a signal to the Raspberry Pi 4 that it needed to record a command. Once the M4 sent this signal to the Raspberry Pi, the M4 would continuously wait for the raspberry pi to send back a response. Once it receives a response from the raspberry pi with the text command it tries to run the command.

Costs

4.1 Parts

Part	**Manufacturer**	**Retail Cost($)**
(3) Stepper Motors	Spark Fun	15.95x3=47.85
(3) Bipolar Stepper Motor Driver	Easy Driver	14.95x3 =44.85
(1) Electromagnet. 5v, 0.5A	Sourced From Lab	Sourced From Lab
(1) Adafruit Metro M4 Express AirLift (WiFi) - Lite	Microchip Technology	34.95
(1) 2MB SPI Flash in 8-Pin SOIC package	GigaDevice	1.25
(1) SEGGER J-Link EDU Mini - JTAG/SWD Debugger	Segger	19.95
(1) 3.5 mm Jack	Spark fun	3.95
(1) Microphone	FIFINE Metal Condenser Recording Microphone	29.99

(1) Speaker	Bose SoundLink Color II	Sourced From Peers
(1) ESP32 Wifi Chip	Espressif	8.95
(1) 60W PSU, 12V, 5A + Wall Outlet Cable	Mean well usa	21.95 + 5.95
(1) Barrel Jack (5.5 mm)	CUI Inc	1.25
(1) Power + Microphone LED Pack	Spark fun	3.30
(1) Voltage Regulator 5V	ST	0.95
(1) Voltage Regulator 12 V	ST	1.50
(1) Programmer Header	4UCON	1.50
(1) Voltage regulator 3.3V	ST	1.95
(1) USB type A connector	Amphenol ICC (FCI)	0.99
(1) 74AHC1G125 buffer	Diodes Incorporated	0.37
(1) NPN Transistor	USA Inc.	0.68
Total Cost of Parts		232.04

Table 1 : Part Costs

4.2 Labor

As electrical engineers we would reasonably make $40 per hour as that is around a $75,000 yearly salary which is close to what an average graduate from UIUC would make [9]. We anticipate that all teammates work about 6 hours per week throughout the semester to complete the product. A semester has approximately 15 weeks, therefore with a 2.5× modifier for exigent circumstances, the total labor for our team of three is 3 (people) x 15 (weeks) x 6(hrs/week) x 40($/hr) x 2.5 (multiplier) = $27000.

As for the ECEB Machine Shop [10], they estimate our product will take about 30 hours of work. With the estimated wage for a machinist being around 50 dollars an hour, the quoted machine Shop labor costs come out to (30hr) x(50 $/hr) = $1500.

Conclusion

5.1 Accomplishments

As commented on by the faculty of this course, this project has many simple subsystems but when all interconnected makes a complex final product. By having connected many of our

subsystems together to complete a wide variety of tasks, it is an accomplishment to be able to play a full game of checkers as well as chutes and ladders. The AutoBoard can successfully move an electromagnet to any spot on the board using our X-Y motor system and motor logic. It can frequently determine the command from a user's voice input and the product can accurately determine if the command is valid based on the game, situation and input. All of these individual successes are then connected to provide smooth, non-contact game play.

5.2 Challenges

5.2.1 PCB Design

Although our microcontroller connections went through two revisions and were checked by multiple people, three paths leading from the debugger footprint to the microcontroller were overlooked and forgotten. This led to straight wire soldering from the tiny microcontroller pins to the debugger holes. In the process, the wires were ripped off taking one of the microcontroller pins with it. This essentially rendered any controller from the PCB impossible and would result in development boards being used instead. As seen below in Figure 8, the correct amount of debugger pins were labeled, but the microcontroller's pins were never assigned to them.

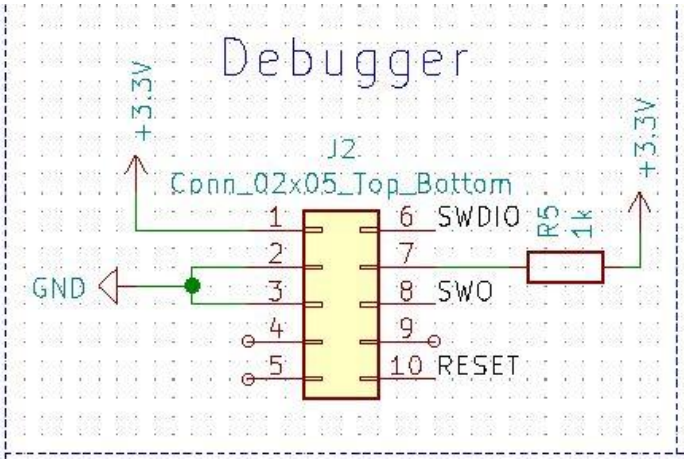

Figure 8 : Debugger Schematic

5.2.2 Memory Limitations

The ATSAMD51J19 microcontroller has roughly 128 kb of memory with only 124kb available for use. Circuit python allows for the recording of audio in a .WAV format. This format has a recording time to file size ratio of roughly one second to 100 kb of memory. This already is an issue as the product must be able to have and use two audio files for auditory feedback with the user. Using the Google Speech-to-Text API required sending post requests to a server with the

file being sent in a JSON format. The converted Json file increased its size by more than four times, making it impossible to even send a one second long .WAV file. A solution determined was using audio files that had been converted into a more compressed Audio Codec. The smallest file size attainable was with AMR (Adaptive Multi-Rate) [11]. Using AMR compression, five second length audio files only took 15kb in size. This proved to be a successful way to send files that were downloaded to the flash memory on the chip, but due to the lack of support regarding recording or converting audio files into this audio codec for circuitpython, AMR was abandoned. It was decided that it was better to solve this issue by using an external board for the audio processing that did not have the memory limitations that the microcontroller had.

5.2.3 Inconsistency with Margent Interactions

Throughout our production, it has seemed as though the interaction between our electromagnet and neodymium magnets were constantly changing. It was determined through the tolerance analysis near the beginning of the semester that one layer of felt was ideal for controlled movement of the neodymium magnet using our 5V, .5A electromagnet. However, at the end of the semester demo, three layers of felt were required. This provided many challenges during testing when identifying logic errors. It is theorized that the reason for such inconsistencies was due to the way the neodymium magnets were stored and the increased sag of the board. When stored, the neodymium magnets were magnetized to each other, compressing the felt attached. When all game pieces were on top of the board, it increased the sag leading to a decrease in distance from the magnets and the electromagnet.

5.3 Ethical Considerations

In the name of ethics, we recognize two potential problems with our product. First being our use of already trademarked games. The issue would arise that the making of any profits with the use of a trademark item would be illegal. This will not be a problem in the context of this class as we will not be making any money or using the product commercially, we vow to uphold the standards as outlined in point four of the IEEE Code of Ethics [12].

Additionally, players may have a problem with their voice data being used and stored. While our product will not store any of the speech data, there is the possibility of speech data being stored by an outside party's speech processing service/software. The exact details will be known to the patrons of the product. We will approach this in accordance with points five and seven of the IEEE Code of Ethics.

5.4 Future Work

Looking towards the future, there are three areas of improvement. First, would be to relook at that existing hardware. Reinforcing the board with additional pillars along all sides would reduce

sag and help any problems with inconsistent magnet interactions. It would also be worthwhile to research different materials that are rigid that could replace our top boards. Also creating a side bucket that could store all the removed pieces would help with the autonomy of the game. The second area is our current user interaction system. By adding more audio responses for certain aspects of the games, it would make using the product more intuitive. Some examples of possible additions are having “moving” or “player 1/2 turn”. The last future work recommendation is to continuously add more games to the product. Right now the product handles two games, but the more you add and the popularity of the added games, will increase demand.

References

[1] Hasbro, Inc. (1923). ”Chutes and Ladders”, [Online]. Available: https://shop.hasbro.com/en-us/product/chutes-and-ladders-game:1095F835-5056-9047-F548-2F4D0AEF4ACC [Accessed 29-Sept-2021]

[2] B. Burwell and B. Jackson, “The Disabled Elderly and Their Use of Long-Term Care,” aspe.hhs.gov, para. 14, Jun. 30, 1994. [Online]. Available: https://aspe.hhs.gov/reports/disabled-elderly-their-use-long-term-care. [Accessed 27-Sept -2021].

[3] L. Rickard, “The Assisted Living Golden Ratio: How Many Residents Can One Caregiver Serve?, para. 3, Mar. 5, 2015, [Online]. Available: https://myhealthspin.com/the-assisted-living-golden-ratio-how-many-residents-can-one-caregiver-serve/. [Accessed 27-Sept- 2021].

[4] Adafruit. “Adafruit Metro M4 Express AirLift (WiFi) - Lite”. [Online]. Available: https://www.adafruit.com/product/4000. [Accessed: 08-Dec-2021].

[5] Raspberry Pi. “Buy a Raspberry Pi 4 Model B- Raspberry Pi”. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ . [Accessed: 08-Dec-2021].

[6] Google Cloud. “Speech-to-text: Automatic Speech Recognition | Google Cloud,” Google. [Online]. Available: https://cloud.google.com/speech-to-text. [Accessed: 30-Sep-2021].

[7] “Dynamic product page | microchip technology”. [Online]. Available: https://www.microchip.com/en-us/product/ATSAMD51J19A. [Accessed: 04-Oct-2021].

[8] SparkFun. “EasyDriver - Stepper Motor Driver”. [Online]. Available: https://www.sparkfun.com/products/12779. [Accessed: 08-Dec-2021]

[9]“Salary Averages”, The Grainger College of Engineering Electrical and Computer Engineering [Online] https://ece.illinois.edu/admissions/why-ece/salary-averages. [Accessed: 29-Sept-2021]

[10] Electrical and Computer Engineering Department. ”Machine Shop”, [Online]. Available: https://ece.illinois.edu/directory/office/10. [Accessed 29-Sept-2021]

[11] “Adaptive multi-rate audio codec,” *Wikipedia*, 16-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/Adaptive_Multi-Rate_audio_codec. [Accessed: 08-Dec-2021]

[12] IEEE. (2016). ”IEEE Code of Ethics”, [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed 29-Sept-2021]

[13] DPA Microphones, “Facts about speech intelligibility: Human voice frequency range,” DPA, 03-Mar-2021. [Online]. Available: https://www.dpamicrophones.com/mic-university/facts-about-speech-intelligibility. [Accessed: 30-Sep-2021]

[14] J. Cochary, “Common Noise Levels,” Noise Awareness Day, 15-May-2021. [Online]. Available: https://noiseawareness.org/info-center/common-noise-levels/. [Accessed: 30-Sep-2021]

Appendix A - Requirements and Verifications Tables

1. Subsystem: Voice Input and Processing (__ / 30)

Requirement:	Verification:
1. The microphone must be able to detect frequencies that capture the range of the human voice (200Hz-5kHz) from up to 1 foot away [6]. The microphone must pick up voice in an environment of 30dB which is equivalent to a room of soft whispers [13]. 2. Must be able to detect a wake word at least 60% of the time to know that a command is coming and that it should begin recording. 3. Must be able to send the audio data to the Speech to Text API in the .wav format. 4. Must be able to correctly recognize voice commands in the returned string from the API at least 60% of the time. 5. Wi-Fi Unit must have connection over 802.11 band with less than 10% packet loss.	● Record samples from users at 1 foot away in a quiet room and playback the audio on a laptop to determine the quality of the recording. ● Test the wake word functionality by using a set of commands and observing how often the system initiates a recording. ● Try and make a call to the API, if the call returns an error the data is not in the correct format. ● Check each word in the string to see if it fits in the current games list of commands, remove any words that do not fit. Check if the final string makes a valid command. Observe the frequency of success over the course of multiple trials. ● Stable connection will be tested over a minute long span to see how volatile the connection is, and how much data is retained.

Table 1 : Voice Input and Processing Verifications

2. Subsystem: Voice Output Logic (__ / 5)

Requirement:	Verification:
1. If a voice input was not understood, the user will hear “Input Command”. 2. If a voice input was understood, and the move was completed, the user will hear “Move Complete”.	● Test the logic by inputting valid and invalid commands to see when the audio output is correctly describing the completed action.

Table 2 : Voice Output Logic Verifications

3. Subsystem: Board Game Logic (__ / 5) and Motor Control Logic (__ / 5)

Requirement:	**Verification:**
1. The MCL will be able to turn the electromagnet on/off when appropriate. 2. The MCL will be able to determine a valid path to move a piece on, so that it does not interfere with other pieces. 3. The MCL will be able to activate the motors to move the electromagnet along the valid path. 4. The board game rules logic subsystem will have to ensure that any move announced by a player is legal. If a move is found to be illegal, a new command will be required before a player's turn is ended.	● Attempt to turn the electromagnet on/off many times in succession to observe its reactivity. ● Test whether moving the electromagnet across adjacent spaces will have adverse effects on adjacent pieces. ● Set up a variety of "situations" in which there are only a few correct paths. Ensure that the system correctly identifies that path by observing its movement. ● When programming this logic unit, it will be necessary to have test cases that attempt to break the game's intended rules. ● Attempt to move pieces by several spaces that are impossible to reach in the game. ● Attempt to move pieces in 4 directions (forward, backward, left, and right) at different points on the board (corners, sides, edges, etc.) to ensure that pieces will stay on the intended track for the game.

Table 3 : Board Game and Motor Control Logic Verifications

4. Subsystem: Power Supply Unit (__ / 5)

Requirement:	**Verification:**
1. The PSU must provide up to 2.50A at between 11.2 - 12.6V DC continuously to the system. 2. Voltages from the outputs of each voltage regulator will stay within .1V to .15V range from its set voltage. a. 3.3V: 3.05-3.45V, b. 5V: 4.85V-5.15V, c. 12V: 11.85V-12.15V	● The maximum current needed by the system will be under 2.5A. By measuring the output current at maximum load with a multimeter, the PSU output can be verified to be sufficient. ● Measure the voltage output of the PSU to make sure that the DC voltage stays within the rated range. ● Probe around the voltage regulators with a multimeter to determine whether their voltage outputs stay within the rated range.

Table 4 : Power Supply Unit Verifications

Appendix B - Project Schematic

1. PCB Schematic

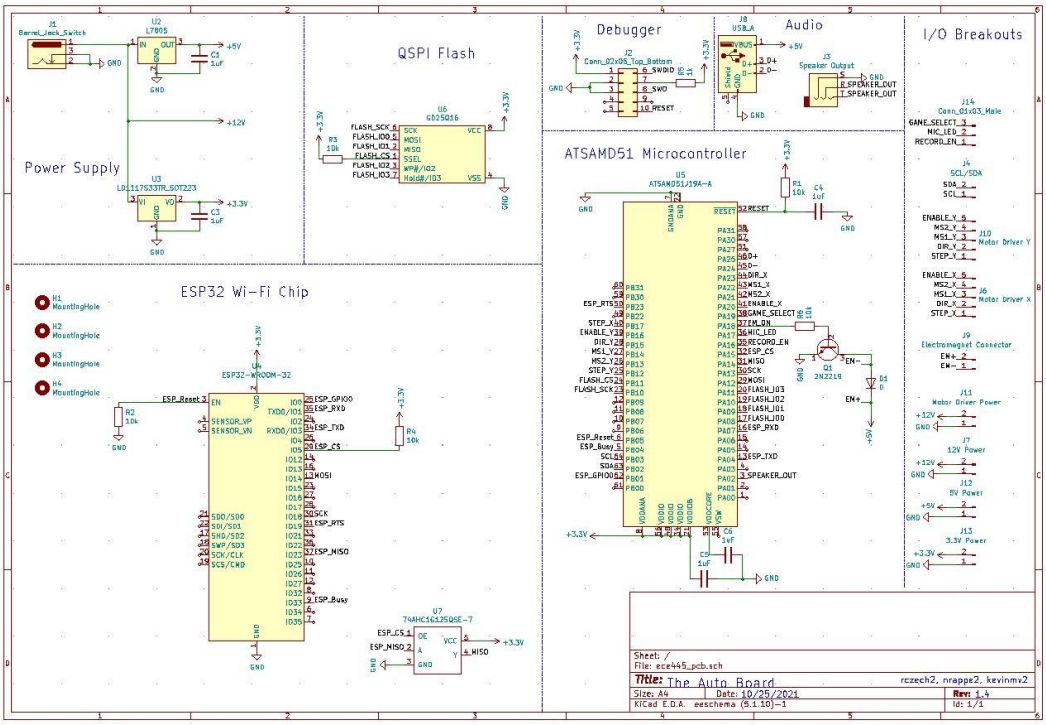

Figure 5 : Full PCB Schematic

Appendix C - Raspberry Pi Code

1. Serial Communication

```
ser = serial.Serial('/dev/ttyACM1', 9600)
ser.reset_input_buffer()
play_audio(-1)
while True:
    line = ser.readline()
    print(line.decode('utf-8').strip())
    if line == b'400\n':
        print(line)
        command = get_command()
        command = command.upper()
        print("pi says")
        print(command)

        command = command + "\n"
        ser.write(command.encode('utf-8'))
        print(command.encode('utf-8'))
        time.sleep(1)
    elif line == b'-1\n':
        play_audio(-1)
    elif line == b'0\n':
        play_audio(0)
    elif line == b'1\n':
        play_audio(1)
    elif line == b'2\n':
        play_audio(2)
    elif line == b'3\n':
        play_audio(3)
    elif line == b'4\n':
        play_audio(4)
    elif line == b'5\n':
        play_audio(5)
    elif line == b'6\n':
        play_audio(6)
```

Figure 6 : Serial Communication Code

2. Recording Audio

```
def get_command():
    form_1 = pyaudio.paInt16 # 16-bit resolution
    chans = 1 # 1 channel
    samp_rate = 48000 # 44.1kHz sampling rate
    chunk = 4096 # 2^12 samples for buffer
    record_secs = 7 # seconds to record
    dev_index = 3 # device index found by p.get_device_info_by_index(ii) is either 
    wav_output_filename = 'command1.wav' # name of .wav file

    audio = pyaudio.PyAudio() # create pyaudio instantiation


    # set up led
    LED_PIN = 17
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LED_PIN, GPIO.OUT)

# create pyaudio stream
    stream = audio.open(format = form_1,rate = samp_rate,channels = chans,
                        input_device_index = dev_index,input = True,
                        frames_per_buffer=chunk)
    print("recording")
    GPIO.output(LED_PIN, GPIO.HIGH)
    frames = []

# loop through stream and append audio chunks to frame array
    for ii in range(0,int((samp_rate/chunk)*record_secs)):
        data = stream.read(chunk)
        frames.append(data)

    print("finished recording")
    GPIO.output(LED_PIN, GPIO.LOW)

# stop the stream, close it, and terminate the pyaudio instantiation
    stream.stop_stream()
    stream.close()
    audio.terminate()

# save the audio frames as .wav file
    wavefile = wave.open(wav_output_filename,'wb')
    wavefile.setnchannels(chans)
    wavefile.setsampwidth(audio.get_sample_size(form_1))
    wavefile.setframerate(samp_rate)
    wavefile.writeframes(b''.join(frames))
    wavefile.close()
```

Figure 7 : Serial Communication Code

3. Speech to Text Api Call

```
f = open('command1.wav', 'rb')
file_input = f.read()
enc = base64.b64encode(file_input)
f.close()
POST_URL = "https://speech.googleapis.com/v1/speech:recognize/?key=AIzaSyDvqe4gC1kSlsn4k_2nJzWtZORk2HfXgeU"

json_data = {
    "config":{
    "encoding": "LINEAR16",
    "languageCode":"en-US",
    "sampleRateHertz": 48000,
    "audioChannelCount": 1,
    "enableSeparateRecognitionPerChannel": False,
    "speechContexts":[
        {
            "phrases":
    [  "move piece location to location", "undo move", "start game", "spin the wheel", "MOVE PIECE 5 spots forwared",
       "move piece 5 spots forward", "MOVE WHITE H8 TO G7", "MOVE BLACK D4 TO E5", "MOVE BLACK D4 TO F6",
       'MOVE', 'WHITE', 'BLACK', 'UNDO', 'START', 'GAME', 'SPIN', 'THE', 'WHEEL', 'BOY', 'GIRL', 'SPOTS', 'FORWARD', 'TO',
       'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8',
       'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8',
       'E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8',
       'G1', 'G2', 'G3', 'G4', 'G5', 'G6', 'G7', 'G8', 'H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7', 'H8'],
            "boost":15

            }]
},
"audio":{
    "content": enc
},

}

response = requests.post(POST_URL, json = json_data)
```

Figure 8 : Speech to Text Api Call

Appendix D - Metro M4 Code

1. Checkers Mappings

```
checkers_mappings_game = {   # I for Inferior(Lower) edge of board,  U for upper edge of board, R for right side of baor
'A0' :(0, -1), 'B0' :(1, -1), 'C0' :(2, -1), 'D0' :(3, -1), 'E0' :(4, -1), 'F0' :(5, -1), 'G0' :(6, -1), 'H0' :(7, -1),
'A1' : (0, 0), 'B1' : (1,0), 'C1' : (2,0), 'D1' : (3,0), 'E1' : (4, 0), 'F1' : (5, 0), 'G1' : (6, 0), 'H1' : (7,0),
'A2' : (0, 1), 'B2' : (1,1), 'C2' : (2,1), 'D2' : (3,1), 'E2' : (4, 1), 'F2' : (5, 1), 'G2' : (6, 1), 'H2' : (7,1),
'A3' : (0, 2), 'B3' : (1,2), 'C3' : (2,2), 'D3' : (3,2), 'E3' : (4, 2), 'F3' : (5, 2), 'G3' : (6, 2), 'H3' : (7,2),
'A4' : (0, 3), 'B4' : (1,3), 'C4' : (2,3), 'D4' : (3,3), 'E4' : (4, 3), 'F4' : (5, 3), 'G4' : (6, 3), 'H4' : (7,3),
'A5' : (0, 4), 'B5' : (1,4), 'C5' : (2,4), 'D5' : (3,4), 'E5' : (4, 4), 'F5' : (5, 4), 'G5' : (6, 4), 'H5' : (7,4),
'A6' : (0, 5), 'B6' : (1,5), 'C6' : (2,5), 'D6' : (3,5), 'E6' : (4, 5), 'F6' : (5, 5), 'G6' : (6, 5), 'H6' : (7,5),
'A7' : (0, 6), 'B7' : (1,6), 'C7' : (2,6), 'D7' : (3,6), 'E7' : (4, 6), 'F7' : (5, 6), 'G7' : (6, 6), 'H7' : (7,6),
'A8' : (0, 7), 'B8' : (1,7), 'C8' : (2,7), 'D8' : (3,7), 'E8' : (4, 7), 'F8' : (5, 7), 'G8' : (6, 7), 'H8' : (7,7),
'A9' : (0, 8), 'B9' : (1,8), 'C9' : (2,8), 'D9' : (3,8), 'E9' : (4, 8), 'F9' : (5, 8), 'G9' : (6, 8), 'H9' : (7,8)
}

boardlayout_current = [[' ', 'x', ' ', 'x', ' ', 'x', ' ', 'x'],
                ['x',' ', 'x', ' ', 'x', ' ', 'x', ' '],
                [' ', 'x', ' ', 'x', ' ', 'x', ' ', 'x'],
                [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
                [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
                ['o',' ', 'o', ' ', 'o', ' ', 'o', ' '],
                [' ', 'o', ' ', 'o', ' ', 'o', ' ', 'o'],
                ['o',' ', 'o', ' ', 'o', ' ', 'o', ' ']]

previousmove = []

magnet_location = "A1"
```

Figure 9 : Checkers Mappings

2. StraightLine function

```
for steps in range(abs(LargestJump)):
    cXStep = (abs(diffx1) * steps) / LargestJump
    fXStep = (abs(diffx1) * (steps + 1)) / LargestJump
    cYStep = (abs(diffy1) * steps) / LargestJump
    fYStep = (abs(diffy1) * (steps + 1)) / LargestJump

    if cXStep != fXStep and  cYStep != fYStep:
        if diffx1 > 0:
            step('x', 'forward')
        if diffx1 < 0:
            step('x','backward')
        if diffy1 > 0:
            step('y','forward')
        if diffy1 < 0:
            step('y','backward')
    elif cXStep != fXStep:
        if diffx1 > 0:
            step('x','forward')
        if diffx1 < 0:
            step('x','backward')
    else:
        if diffy1 > 0:
            step('y','forward')
        if diffy1 < 0:
            step('y', 'backward')
    #time.sleep(0.01)
# we have reached our start location turn magnet on to drag piece
MAG.value = True
#now same idea but from start to end instead of current to start
if abs(diffx2) >= abs(diffy2):
    LargestJump = diffx2
else:
    LargestJump = diffy2
```

Figure 10 : StraightLine Code Snippet