Introduction to Engineering Design with Professional Development 1

Final Report for
Chessboard for the Blind

Team: Group 1
Section 3
Instructors: Randall McFarlane
Glen Gross

Version 1.0
May 3, 2021

Prepared by
Heather Converse (2023, Mechanical Engineering)
Eileen Beres (2023, Mechanical Engineering & DIS)
Brooke Zatowski (2023, Aeronautical Engineering)
Nicolas Nigohosian (2023, Mechanical Engineering)
Julian Matthews (2023, Mechanical Engineering & Computer Science)
Carter Wynn (2023, Mechanical Engineering)

Executive Summary

The team decided to create a simulated blind chessboard for this project. In brainstorming ideas for the project, the team focused on coming up with ideas that would be challenging but achievable, interesting to work on, and fulfilled the project parameter of serving an underrepresented group. A blind chessboard was an effective solution that proved to have both anticipated and unanticipated challenges.

The team followed the engineering design process, from researching customer needs and benchmarking, translating these needs into technical specifications, concept selection, and eventually the final concept. The design process required diligence to the mission, effective communication between team members, and attention to detail to execute effectively.

The project was split into six subsystems: user interface, piece movement, game feedback, game functionality, piece detection, and power. Each team member worked on their subsystem to develop an engineering solution for their part. The team then integrated all six subsystems to formulate the entire system for the final concept. Communication was required between subsystems in order to ensure inputs and outputs between subsystems lined up with one another. Due to the online learning environment as a result of the coronavirus pandemic, the project was simulated using various simulation packages but not prototyped or built. Team members gained exposure to new simulation packages in the process of simulating their subsystems.

The team is proud to present the design process and final concept in this report. The developed chessboard for the blind includes an electromagnetic push release system for piece movement, vibrational in-game feedback, in-game move recording, a built-in computer opponent, and other features. It is a revolutionary new development in the blind chess world that is bound to excite blind chess players and expand interest in learning the game of chess for blind students.

Table of Contents

1 Introduction

Many popular games, like monopoly, are extremely visual. When adapted for the blind or visually impaired, these games are often converted to braille, which can be cumbersome and unintuitive. The blind can play chess without the need for braille. Unlike games like monopoly, there are many competitive chess tournaments worldwide, offering competition at many levels of experience. Chess has seen significant growth in popularity after the release of Netflix's The Queen's Gambit, yet blind chess players are still underrepresented when it comes to the experience of chess. With current chess boards on the market, pieces are easily knocked over, and the game is challenging to learn for beginners. Overall, there are few chess boards for the blind that go above and beyond basic features to provide a tailored experience for the senses of touch and sound. For these reasons, this project was worth pursuing because it focuses on a unique and niche market where there is much room for new ideas.

First, the customer requirements and concept generation processes will be detailed. From there, there will be an in-depth analysis of societal considerations surrounding the project and a close look at the design of each subsystem. To conclude, there will be an explanation of the results and accomplishments of the product design, and then appendices for additional information, diagrams, and code.

2 Project Objectives & Scope

In the 10-week allotted time, the team has worked to design and simulate an improved version of a chessboard for the blind and visually impaired. The minimal constraints on this project supported the team's decision to choose and create a product that supports people with disabilities who are often underrepresented in society.

Listed below are the Project Objectives of the Spring 2021 Semester

- Cohesive and positive team dynamic
- Execute all parts of the engineering design process
- Target needs specific to the customer
- A working chessboard that has both CPU and 2 player modes

The following subsections outline the mission statement, customer requirements, and technical specifications of the Chessboard for the Blind.

2.1 Mission Statement

This project aims to improve the experience of chess for the visually impaired by designing a chessboard with appropriate tactile features. Additional tournament play and learning features will make chess more accessible to students and competitive players. Game feedback built into the chessboard will make the game easier to play with only sound and touch. All

built-in modes of play will be designed to provide sufficient audio and physical feedback to minimize the difficulties experienced by the blind in the gameplay of chess.

2.2 Customer Requirements

The target customers for this project are blind chess players, with a specific focus on middle and high school-age students. In speaking to prospective customers, including a phone call with George Abbott, vice president of The Lighthouse for the Blind in Seattle, it was discovered that the subgroup of students was in particular need of an improved board to help them learn the game. He also noted the need for tournament standard boards equipped with aids to allow blind players to compete with more ease, particularly for middle and high school competitions. The blind is a demographic often overlooked in chess because visual impairment creates unique challenges for players that require special assistance in the design of game features. The project's goal was to improve upon chess boards currently available to blind chess players as well as increase access to the game of chess for the blind by incorporating a feature for training in our design for students and other learners.

Blind chess players have specific needs regarding tactility in the design of the chess game board since visual signifiers are of no use. Some of these needs include the ability to differentiate black and white pieces, squares on the board, and piece type. Tactile signifiers for these various game components must be obvious to the player but not distracting. Therefore one of the customer's needs is that pieces are of the same general shape. Another issue faced by blind chess players is pieces toppling over when they are feeling around the board. This is particularly common for the bulkier pieces, such as the king and queen. Therefore one of the key needs addressed in the project will be incorporating a system that improves the mechanism to hold pieces in place on the board while the player feels around.

The customers also request that the board allows them to compete in traditional chess tournament-style play. As a result, the board must be up to tournament standards regarding board size while providing appropriate aids to the blind player to assist in play. Going along with tournament standards, the customer will also need a no-touch clock system that times moves. As such, the clock must have an easy-to-use switching system that reads out when certain amounts of time have passed. The customer needs this clock to be integrated into the board as well.

Additionally, in tournament play, the players must record their moves. This can be difficult for blind players who normally have to voice record moves, which takes away concentration from the game and valuable playing time. As a result, one of our customer needs is the ability for the board to automatically take notes of moves and allow a list of moves to be extracted from the system upon game completion.

Finally, the customer needs the board to learn the game and improve upon current skills through training capabilities. As a result, one of the customer needs is a built-in trainer that includes a computer opponent that operates at several levels of difficulty for players to practice. This, paired with the ability to track moves during a training session using the move recorder function, will assist students in learning the game.

The respective summarized needs that have been discussed are listed below in Table 1 and have been translated into general requirements and technical specifications with target values.

Table 1: Summary of Overall Customer Requirements

Customer Need	**Requirement/ Technical Specification**	**Target Value / Range of Values**
Pieces Can't Be Knocked Over	Strong Magnets	Magnet Strength >= Weight of Piece
Distinguishable Pieces	Differentiable Piece Material for Black/White	Stainless steel & wood coated stainless steel
To compete	Up to Tournament standard	Each square for tournament play should be between 5 cm to 6 cm that is equivalent to 1.97 inches to 2.36 inches
Different computer difficulties	Easy, medium, hard, expert modes	1000, 2000, 3200 elo stockfish
*No touch clock**	Clock automatically stops when move is made	When reader detects valid move, clock stops, starts for other player
*Take quick notes of moves played**	Moves are recorded and extractable via usb or sd card	Board records PGN for match, saves on USB or SD card
*Integrated clock**	Clock timer is attached to the chessboard	Reads out each players time every 30s
*Pieces of same shape**	Square pieces (two different materials used for black and white) & each piece indicates the direction of play for that piece	3.5cm x 3.5cm x 3.5cm blocks

The team also spoke with the United States Blind Chess Association, avid chess players included in our target demographic, to further our customer needs inquiry. In speaking with players who are highly familiar with playing chess blind, the team received several pointers on our initial proposed design ideas. Their specific needs included: raised signifiers rather than indented signifiers on pieces and buttons for readability, a smaller board to allow them to quickly find pieces and squares, a mechanism for centering pieces on squares, a push-down release

system for an electromagnet system holding pieces in place to allow for piece movement, and the possibility of a mechanism for locating dropped pieces. The team chose to address all of these needs except for a mechanism for locating dropped pieces. It would be too difficult to attempt to implement this function with all of the other proposed functions given our time constraints. A summary of the customer needs and requirements added after consulting the USBCA is featured in Table 2.

Table 2: Summary of Customer Needs Gathered From USBCA

Customer Need	**Requirement/ Technical Specification**	**Target Value / Range of Values**
Relatively small board	Around 10in x 10in	Within 2in on either side of 10in square board
Pieces stay centered on square	Magnets in center of piece and square	Electromagnet centered on each square
Touch read out of squares	Speaker read out of square position with force applied to the square	Instantaneous readout w/ force applied to square
Raised signifiers on pieces	Rectangular prism shaped pieces with raised symbol on top face	Symbols raised by 2mm on top face of piece
Push release system for piece holding system	Electromagnet system that releases with application of force on square	Electromagnet releases with x force applied to square
Mechanism for finding lost pieces	Sound that plays when piece is separated from board	When piece is more than 3 ft from board, beeping sound plays

2.3 Technical Specifications

The customer requirements were translated into technical requirements and target values. For the magnetic piece attachment, the magnet strength must be greater than the weight of the piece for the piece not to be knocked over. For the board itself, each square should be around 2in x 2in, a tournament standard value. The overall board should be approximately 10in x 10in (later changed to 16in x 16in). The pieces should be square and uniform, around 3.5cm x 3.5cm, and made of wood and steel for differentiation. There should be 2mm raised signifiers on each piece to distinguish it. The computer should record all moves made and output them in PGN format via USB. The integrated clock should read out each player's time every 30 seconds.

For the gameplay, the electromagnets should be centered well under each square. They should react instantly to release that piece when the force of a push of a human finger is applied to the top of the piece. There should be at least three levels of difficulty in the computer opponent, which will be a 1000, 2000, and 3200 ELO rating of the stockfish free chess engine.

When a piece falls off the board and is more than 3ft from the board, it should play a beeping sound. The target values for the technical specifications are shown in the final column of Tables 1 and 2. However, the CPU difficulty modes and piece finding mechanisms were not implemented for this project. Later development of the subsystems also created new specifications and tests for analyzing the performance of each subsystem of the board.

3 Assessment of Relevant Existing Technologies

There are already chess boards for blind players on the market. The Braille Superstore sells a wooden chess set made specifically for blind and visually impaired players (The Blind Superstore, n.d.). The black squares are raised to help blind players track the diagonals. There is a circular peg on the bottom of each chess piece, which fits inside the circular hole located in the center of each square. The pegs also fit in the evenly spaced holes in the capture bay (a capture bay is where the chess pieces are stored once a player captures one of their opponent's pieces). Small spheres are located on top of each light-colored piece to distinguish their color from the dark-colored piece. The design is shown in Figure 1.

Figure 1: Wooden Chess Set by The Braille Superstore

There are several issues with this design. Blind chess players find it difficult to feel around the board without knocking the pieces out of their holes (Balata et al., 2020). New players may also have difficulty identifying the positions on the board without any braille markings to designate the rows and columns (rows and columns are referred to by numbers 1-8 and letters A-H). This design is also not compatible with tournament play because there is no integrated clock. A summary of this board's relation to the project is shown in row 1 of Table 3.

Chess Baron sells a similar wooden model that also utilizes raised dark squares and a tactile indicator to distinguish the color of each piece. The differences in this design include the shape of the indicator and the method of piece attachment. There are spikes on the top of each

dark-colored piece, and the pieces are secured to the board with magnets. The entire Chess Baron set is shown in Figure 2, and the design of the dark chess pieces is shown in Figure 3.

Figure 2: Full Chess Baron Set

Figure 3: Chess Baron Set, Black Pieces

This design solves one of the issues of the previous design but introduces more issues. The magnets attach the pieces to the board more securely than the peg and hole method. It lacks a capture bay, which makes it easier for blind players to lose their unused pieces. The spikes on the dark pieces are a safety hazard for young players. This board also does not resolve the position identification and tournament play issues generated by the previous design. A summary of this board's relation to the project is shown in row 2 of Table 3.

The final design the team researched is currently not for sale to the public. Duncan McKean modeled a chess set for the blind on his website (McKean, 2021). This board helped generate ideas for a new and improved chessboard concept. McKean's design can be seen in Figure 4.

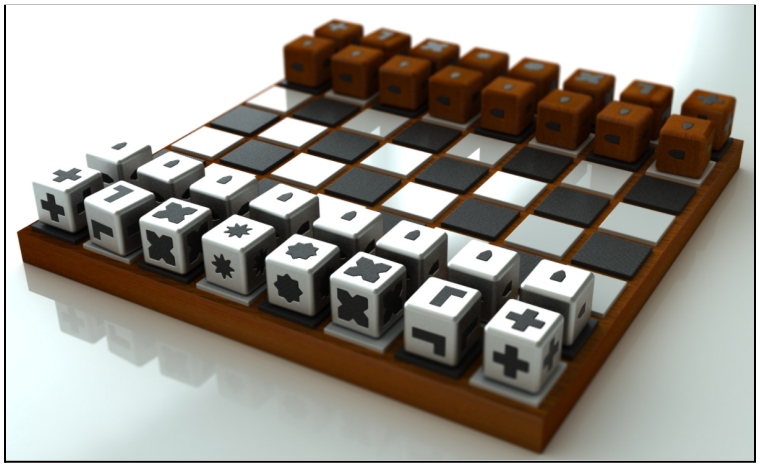


Figure 4: Duncan McKean Blind Chessboard Design

This model features same-sized square blocks for the game pieces instead of the original design for chess pieces. Each square piece has a design on it that represents the allowed move that particular piece can make, which is shown in Figure 5.

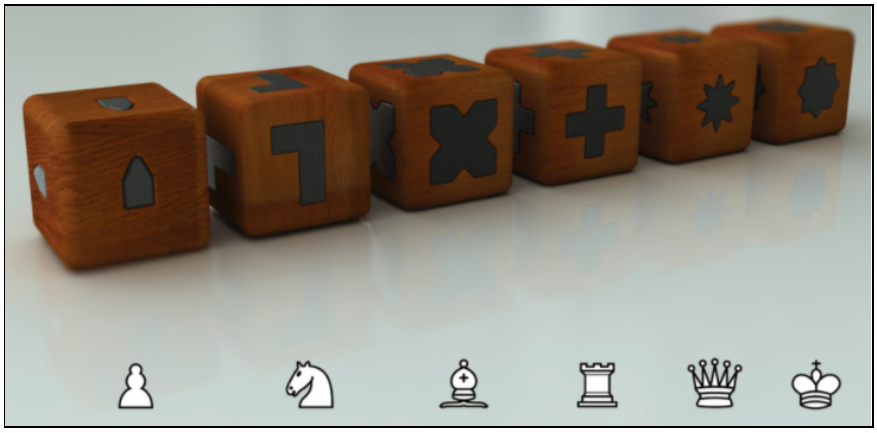


Figure 5: Modeled Black Pieces, Duncan Mckean Design

In addition to the different shape indicators, the weight varies for each cube which helps identify the piece being used. The white and black pieces are made of different materials so that they are easily distinguishable. The white pieces are made of matte steel and are cold to the touch, while the black pieces are made of hardwood and are warm to the touch (McKean, 2021). The board itself uses magnets to hold the pieces in place, so they do not accidentally knock over. The white tiles have weaker magnets than the black tiles to help differentiate between squares on the board.

While this design is unique, it still leaves various issues that need to be addressed. There is no integrated clock timer in this model, which is necessary for competition play. Users also must have another person to physically play with them because there is no option for a virtual computer opponent. Players who are new to chess will have a difficult time learning how to play as there is no learning mode. Finally, there is no capture bay to place the pieces after they are removed from the board, making it difficult for the player to keep track of pieces. Duncan McKean's model accommodates many needs of blind chess players, but it still lacks some of the key customer needs. A summary of this board's relation to the project is shown in row 3 of Table 3.

Table 3 : Competitive Benchmarking

Competitive Product	Title / Description	Relation to this project
Chess Set, Wooden (Tactile) by The Braille Superstore	Classic design for a chess set made for the visually impaired	1. Solves customer need to track the diagonals by raising the black squares 2. Attempts to addresses customer need of not knocking over the chess pieces by attaching them to the board via the peg and hole method 3. Solves distinguishable chess pieces customer requirement by having a

		wooden sphere on top of the white pieces
10 inch Chess Set for the Blind – Magnetic by Chess Baron	Wooden chess set made specifically for the visually impaired with magnetic chess pieces	1. Solves customer need to track the diagonals by raising the black squares 2. Attempts to addresses customer need of not knocking over the chess pieces by attaching them to the board via magnets 3. Solves distinguishable chess pieces customer requirement by having a spike on top of the dark pieces
Yanko Design by Duncan McKean	Chess set designed for the visually impaired (currently not for sale to the public)	1. Solves customer need to track the diagonals by having different strength magnets for the white and black squares 2. Attempts to addresses customer need of not knocking over the chess pieces by attaching them to the board via strong magnets 3. Solves distinguishable chess pieces customer requirement by making the light and dark pieces out of different materials with different weights and thermodynamic properties

The team referenced patents to brainstorm ideas for the concept generation phase of the design process. Patent number 5503400 was chess set construction (Silva,1996). This patent permitted chess to be played with the chessboard in a vertical position. To accomplish this, a magnet was placed in the center of each chess piece. This method of piece attachment was used in team 1's design. The patent chess pieces were also halved, which the team considered when designing the chessboard. Halving the pieces so that they lay horizontally on each chess square maintains the pieces' tactile identifiers while decreasing the height of the pieces. Decreasing the height of the pieces decreases the likelihood of knocking over the pieces. Figure 6 shows the design of the halved chess pieces in the patent. However, since the team decided not to use traditional chess piece shapes (chose to use square blocks instead) to include raised directions on the top of each piece, halving the pieces was no longer beneficial.

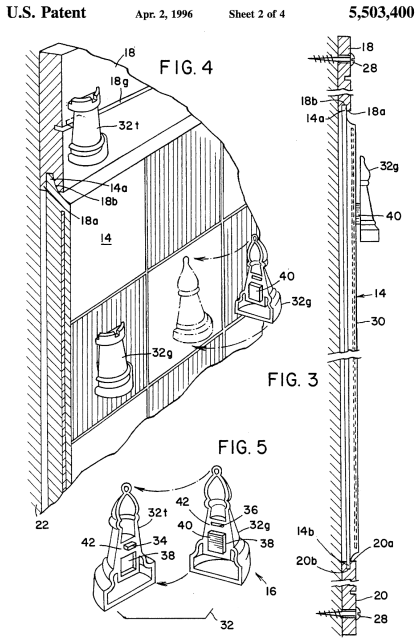


Figure 6: Illustrated Design of Halved Chess Pieces in Patent 5503400

Another patent the team referenced was patent number 5848788: Electro-magnetic game board (Hess, 1998). All of the chess pieces in this patent have a game piece magnet, and there are electromagnets underneath the board. The chess pieces flip in the air if the electromagnet and the game piece magnet are opposite polarities. After seeing this patent, the team decided to research electromagnets to secure the pieces to the board. The thought process was that if electromagnets are strong enough to flip chess pieces off of a chessboard, they should be strong enough to secure chess pieces to the chessboard if the polarities were changed. Heather decided to use electromagnets in her piece attachment subsystem. Table 4 summarizes the patents' relations to this project.

Table 4 : Patent Research for Related Technologies

Patent Number	Title / Description	Relation to this project
5503400	Chess set construction	1. Utilizes strong magnets in the chess pieces and in the chessboard to keep the chess pieces from falling off the board

		2. Halves chess pieces, which would help prevent the pieces from being knocked over
5848788	Electro-magnetic game board	Uses electromagnets to flip over chess pieces, so they could be strong enough to secure chess pieces to the board

4 Professional and Societal Considerations

As part of the design process, the team analyzed the impact the design had in multiple areas. The chessboard's impact was graded in six areas: public health and safety, global, cultural, societal, environmental, and economic. The results of the analysis are compiled in Table 5 - Engineering Solutions Impact.

Table 5 : Engineering Solutions Impact

Area of Impact	**Impact**	**Description of Impact**
Public Health and Safety	N	2.5 - minimal negative
Global	Y	5 - strong positive
Cultural	Y	4 - positive
Societal	Y	5 - strong positive
Environmental	Y	2 - negative
Economic	Y	5 - strong positive

The first area of impact the team looked at was public health and safety. A chessboard is not inherently dangerous, but the team's design, with its extensive use of electronic and small pieces, does present some issues. One issue is the use of stainless steel for the white chess pieces. The most common form of 304 stainless steel uses approximately eight percent nickel which can trigger an allergic reaction in about seventeen percent of women and about three percent of men. However, by using hypoallergenic stainless steel, this issue can be avoided entirely. Another possible safety issue is the electronics in the board. According to The U.S. Consumer Product Safety Commission, approximately 180 people are electrocuted in the United States per year by consumer products. Twenty percent of these electrocutions are due to wiring hazards. The wiring in our chess board is completely contained in the board's shell and therefore negates the risk of a

user electrocuting themselves. The blind chessboard also has potential negative impacts on health and safety due to the risk of small pieces. Young children and animals can accidentally swallow small pieces if left out, and they can pose a tripping hazard if accidentally dropped on the floor. In regards to swallowing pieces, according to Harvard Medical Center, "Complications can include tears in the esophagus (the tube that connects the mouth and stomach), movement of the object into the tissue of the esophagus, and infection." Furthermore, our pieces pose additional risks since they are magnetic. Harvard Medical School details, "If more than [magnet] is swallowed, they can stick together and erode through tissue" (H, 2019).

The product has the positive impact of taking the well-being of the blind into account in its creation. The chessboard aimed specifically at blind learners of chess, which provides blind students increased access to chess, an activity that challenges their brain and provides a fun hobby. According to Healthline, chess improves memory, increases intelligence, and elevates creativity, and can lead to better planning skills, among other benefits (Stanborough). The product's mental health benefits could bring to the blind population help to negate the potential negative health and safety aspects of our design. Overall the board's design has a minimal negative impact on public health and safety and can be considered to have no impact on this area.

The next area of impact the team analyzed was global impact. Around the globe, there are approximately 39 million blind people with minimal access to chess teaching tools. The team's design aims to fix this issue by building a chessboard from the ground up with the intent to be used to teach blind and visually impaired individuals how to play chess. Currently, the chessboard design does not meet all the tournament standards defined by the International Braille Chess Association. Specifically, the design is missing the hole and peg method to secure the chess piece. However, if the design is sold on the market and popular with blind chess players, then the rules could be changed. Overall the board's design targets a neglected part of the international chess community and therefore has a strong positive impact globally.

Cultural impact was the next area the team analyzed. Chess is a globally loved game that extends across many cultures. Opponents do not even need to speak the same language to compete against each other. By helping the blind learn chess, this design has the power to unite cultures through the game of chess. The uniting effect of chess and the design's ability to include blind individuals in the game means that the team's design has a positive impact in the cultural area.

The team then analyzed its design's impact on the societal area. There is a stigma in society that individuals in the blind community are less able than their peers. Due to this stigma, they are at a disadvantage regarding employment and advancement opportunities. By teaching blind individuals to play chess, a game perceived as reserved for the superiorly intelligent, the team's product could break the stigma, opening up job opportunities for the blind community. The design could also help the mental health of older blind individuals. Thirty percent of blind individuals in the United States are over 65 and therefore are at high risk of death from the COVID-19 virus (World Health Organization). As a safety precaution, the elderly have had to isolate themselves from friends and family members. This isolation can adversely affect their

mental health. The team's product provides the elderly a source of mental engagement that does not require human interaction. The user has the option to play in CPU mode, which allows them to play chess against the computer. With its benefits to the blind community, the team's design positively impacts society.

The next area of impact the team analyzed was environmental impact. The team's design of the chessboard contains many electronic components, including Arduinos, RFID chips, RFID readers, and circuitry. If manufactured, the product would add to the 50 million tons of e-waste generated every year. E-waste sites have a strong negative impact on the environment. For example, water in the soil of the Chinese e-waste site "Guiyu" contains lead 2,400 times higher than safe levels (*Tons,* The World Counts). The product increases the amount of e-waste, which increases the number of e-waste sites, which contaminates a greater percentage of our planet. Despite this, the team worked to design the board to be as sustainable as possible. If a component breaks on the board, it can be replaced with having to replace the entire board. It can still be used as a standard chessboard with some benefits for blind individuals if all electronics break. Stainless steel was chosen as the material used for the chess pieces, which is a hundred percent recyclable. These sustainable design elements do not entirely cancel out the e-waste, so this design has a negative environmental impact.

Finally, the team analyzed the economic impact of their design. Introducing this product to the market creates competition for existing manufacturers of blind chess boards. Increasing competition lowers prices for consumers, increases efficiency and productivity, and spurs innovation. Manufacturing this product would create numerous new job opportunities for the unemployed. Employed individuals use their wages to contribute to the economy in terms of the goods or services they buy. Finally, using stainless steel for the chess pieces is economical. Assuming the customers send the chess pieces back, recycling stainless steel lowers manufacturing costs. Alternatively, the company can sell the stainless steel as scrap metal. Selling scrap metal allows the company to earn back some of the initial investment they paid for the metal. This design has a strong positive impact economically by creating more competition in the blind chess board market and creating more manufacturing jobs.

5 System Concept Development and Selection

5.1 Concept Generation

The first step of the team's concept development process was concept generation. The team brainstormed different aspects of the project relating to the product's overlying features, functionality, aesthetics, and safety. With these aspects taken into consideration, a rough initial sketch was produced.

This sketch featured a few of the design aspects that the customers requested when interviewed by the team. An integrated clock timer was included for competitions, and buttons were placed on both sides to switch the clock easily. Electromagnets kept the pieces in place and

turned off for three seconds while the user made a move. The board read out the name of the tile that the user's piece was hovering over. This would allow the user to confirm they were putting the chess piece on the correct square before placing it down. RFID chips for piece detection and piece raising to track the last move made were also featured. Finally, the chess pieces had varying heights and weights. This design sketch is shown in Figure 7.

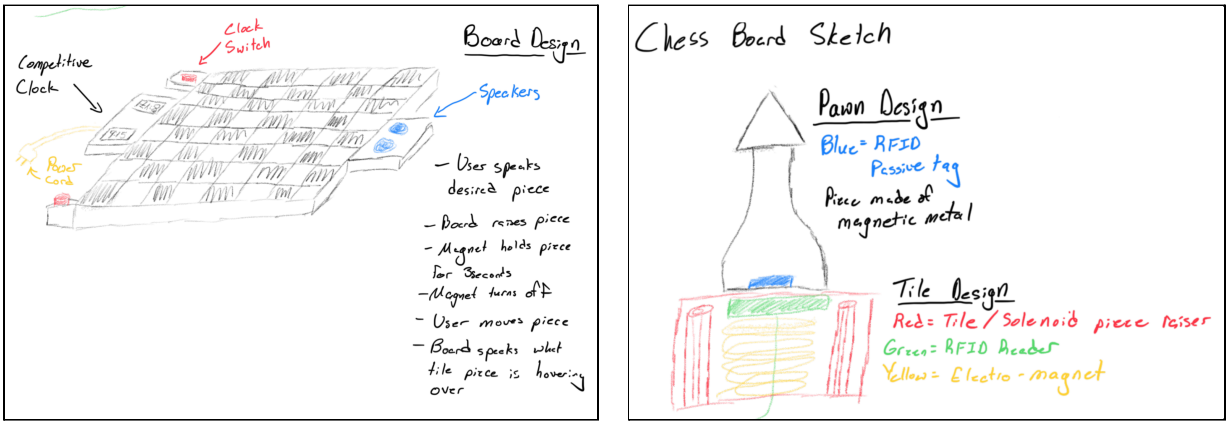


Figure 7: Initial Concept Sketches

A second rough sketch was drawn as part of the brainstorming process. This sketch implemented many design ideas similar to the first sketch but also introduced some new concepts. Shown above the board to the top left was the proposed RFID chip in each piece, the piece raising mechanism, and the electromagnet shown in pink. Concept pieces are shown at the top middle, with different materials for white and black, and engraved movement patterns for each piece. For the board itself, the left-hand side had buttons with braille for selecting computer opponent difficulties and learning or competitive mode. There was an integrated speaker on either side of the board to read out moves made by the players or the computer. For competitive play, this speaker generates a tone after a specific interval, such as 30 seconds, and reads each players' time at important intervals. Shown in the bottom right is the power cord for the board, mainly powering the electromagnets and solenoids. With all of these features, this design followed tournament standards while implementing a simple way for blind students to learn chess. The second concept sketch is shown in Figure 8.

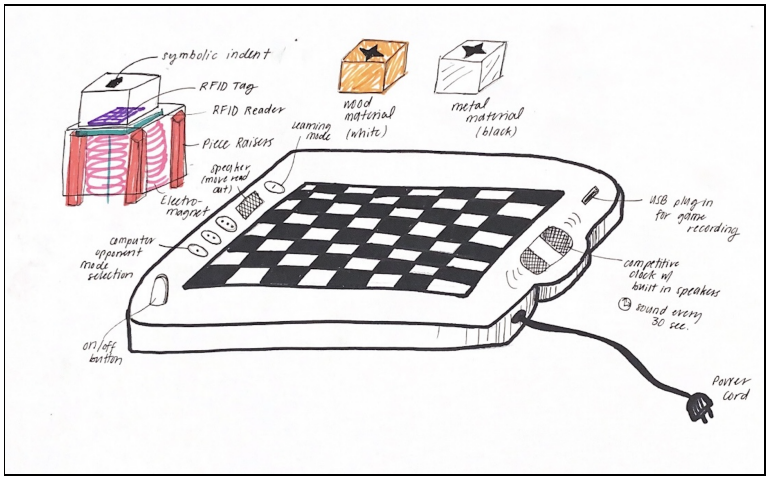


Figure 8: Second Concept Sketch

After considering these, the overall design problem was divided into six functional subsystems, explored with mind mapping, as shown in Figure 9.

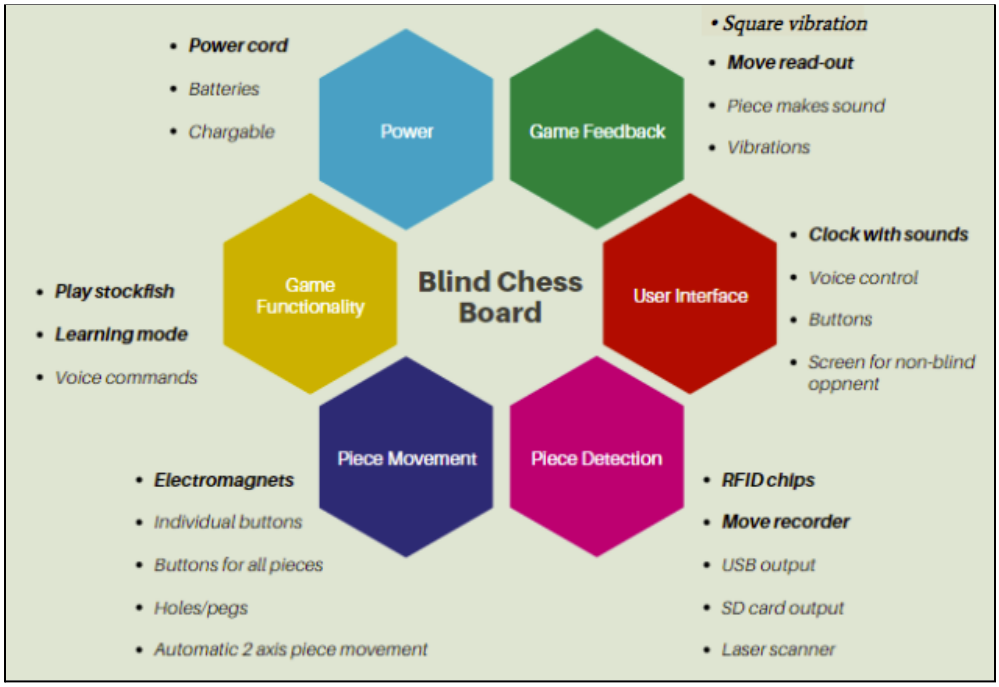


Figure 9: Mind Map

The bolded bullet points were the chosen design ideas which will be explained further in section 5.2, Concept Selection.

5.2 Concept Selection

After brainstorming, the team began the concept selection process. During this process, the team prioritized customer needs to design a product that best accommodated blind users. The

ideas were placed into a selection matrix where they were graded based on a variety of criteria. This included feasibility, adherence to customer needs, cost, and tactility. Each criterion was given a weight that corresponded with the importance of said criteria. Adherence to customer needs, requirements, and specifications was weighted the most since the goal of the product was to create a better blind chess experience for the customer. On the other hand, the cost was weighed the least since no physical prototype was created. Each design option was given a score from zero to five, five being the best, for each criterion. A weighted average was taken for each idea based on its scores and the weight of the criterion. The concept selection matrix is shown in Table 6.

weight:	0.2	0.4	0.1	0.3		
	Feasibility	**Adherence to Customer Needs**	**Cost**	**Tactility**	***TOTAL***	***WEIGHTED TOTAL***
Game Feedback						
Piece vibration	4	5	3	5	17	4.6
Move Read-out	4	5	5	4	18	4.5
Piece makes sound	3	2	3	3	11	2.6
Vibrations	1	2	2	4	9	2.4
User Interface						
Clock w/ sounds on interval	4	4	4	3	15	3.7
buttons with braille	5	5	4	5	19	4.9
Voice control	3	4	3	3	13	3.4
Screen for non-blind opponent	3	1	3	0	7	1.3
Piece and Board Structure						
Cube chess pieces	5	5	4	5	19	4.9
Different materials (black vs white)	4	4	4	5	17	4.3
Spikes on white, smooth black	4	3	4	4	15	3.6
Tactile dots on white	4	3	4	4	15	3.6
Metal rod on white	4	2	4	3	13	2.9
Piece Detection						
RFID Chips	4	5	3	5	17	4.6
Move Recorder	5	5	4	5	19	4.9
usb output	5	5	4	4	18	4.6
SD card output	4	5	5	3	17	4.2
laser scanner	2	4	2	3	11	3.1
move printer	3	3	2	4	12	3.2
Electromagnets	4	5	4	5	18	4.7
Piece Movement						
magnets, individual buttons	3	3	4	3	13	3.1
magnets, all piece buttons	4	4	4	5	17	4.3
Holes/pegs	5	4	4	4	17	4.2
automatic 2-axis piece mover	2	4	1	3	10	3
Voice control	3	3	3	2	11	2.7
locking piece attachment	4	4	4		12	2.8
Game Functionality						
play stockfish	4	5	4	4	17	4.4
Learning Mode	4	5	4	4	17	4.4
Voice Commands	3	4	4	4	15	3.8
Power						
Power Cord	5	3	5	4	17	3.9
Batteries	3	3	4	4	14	3.4
Chargable	2	4	3	4	13	3.5
Scoring: 0-5 (5 is the best)						

Table 6: Concept Selection Matrix

The ideas colored green were the highest-ranking ideas for each subsystem. Some of the subsystems show two selected design criteria because the team decided this would best satisfy customer needs. For the final design, piece vibration and move read-out scored the highest for the game feedback subsystem. For the user interface subsystem, an integrated timer clock with sounds for time intervals and corresponding buttons with braille scored the highest. This was important for customers who wanted the board to have tournament capability while being easy to navigate. The piece detection subsystem had the highest scoring in RFID chips, a move recorder, and USB output. For the piece movement subsystem, electromagnets for each square were decided on. For the game functionality subsystem, stockfish as a chess engine and the

incorporation of a learning mode scored the highest. The use of a power cord scored the highest for the power subsystem. For the overall piece and board structure, cube chess pieces differentiated by material were chosen. Users indicated that smaller, same-size pieces were easier to move.

5.3 Final Concept

A model of the components under each chess square is shown in Figure 10. The black box at the bottom of the figure is the RFID chip reader. The component above the chip reader is the electromagnet: a donut-shaped ferrite core wrapped in wire. Next is the vibration motor, which is small enough to fit in the inside diameter of the electromagnet. Above the vibration motor is the antenna (the coiled silver wire). The vibration motor is not directly located under the square, for it is required that the antenna of the RFID reader is as close to the chess pieces as possible. However, given that the antenna is less than the thickness of a credit card, the vibrations are strong enough to pass through the antenna to the chess piece.

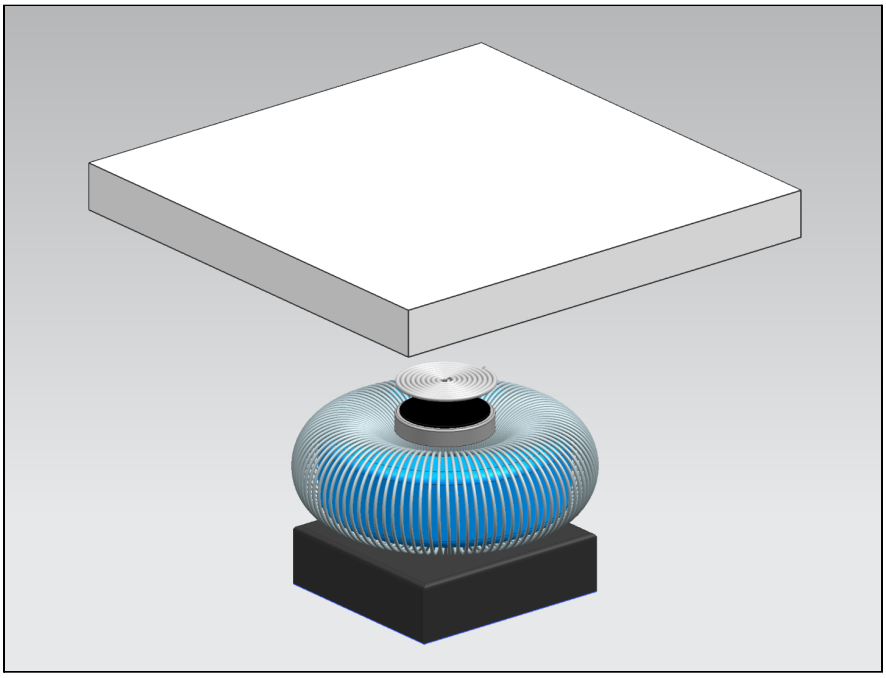


Figure 10: Modeled Components Under a Chess Square

The final models of the chessboard are shown below. Figure 11 shows the CAD model of the board structure in Rhino 5. As shown, the general shape of the board is rectangular with filleted edges to ensure safety for blind users in feeling around the board. The square game board is adjacent to the rectangular user interface board, which includes the speaker system and buttons to select game functions. The game board itself is designed to be 16 inches squared, and the entire board, including the user interface, is designed to be 20 inches squared. Figure 12 shows a

more detailed view of the user interface from a top view. There are two square speaker platforms on either side to ensure equal audio output to a player on either side of the board. In the center, buttons two, three, four, and five are stacked in a line. They are differentiated by small Braille-like raised bumps on the tops of each button. On either side of the rectangular user interface board, a larger button is used to switch play from one player to the other when using the built-in timing system.

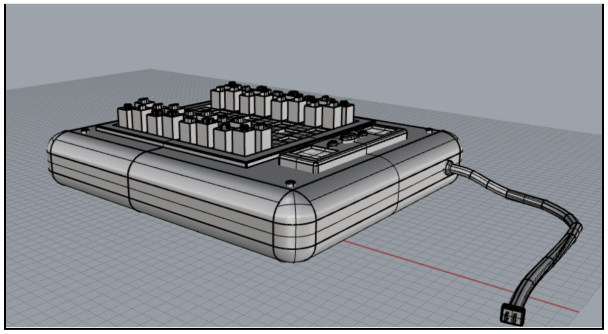


Figure 11: CAD Model of Board, Rhino 5

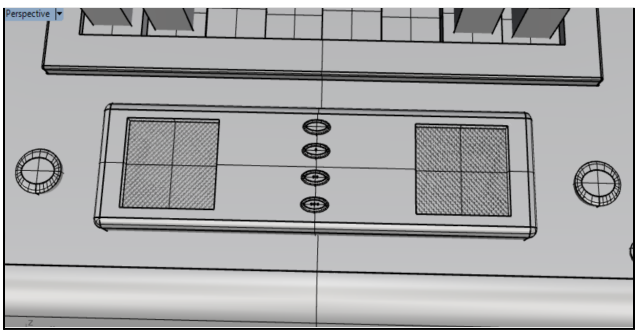


Figure 12: CAD Model of User Interface, Rhino 5

A top view of the final concept modeled in Rhino 5 is shown below in Figure 13. Raised symbols differentiating each of the pieces are shown on the top face of each piece. The symbols are based on the Duncan McKean design, but they have been raised rather than indented as suggested by the USBCA. The symbols are designed to direct the player of the possible directions each piece can move, which is a helpful aspect for a blind learner.

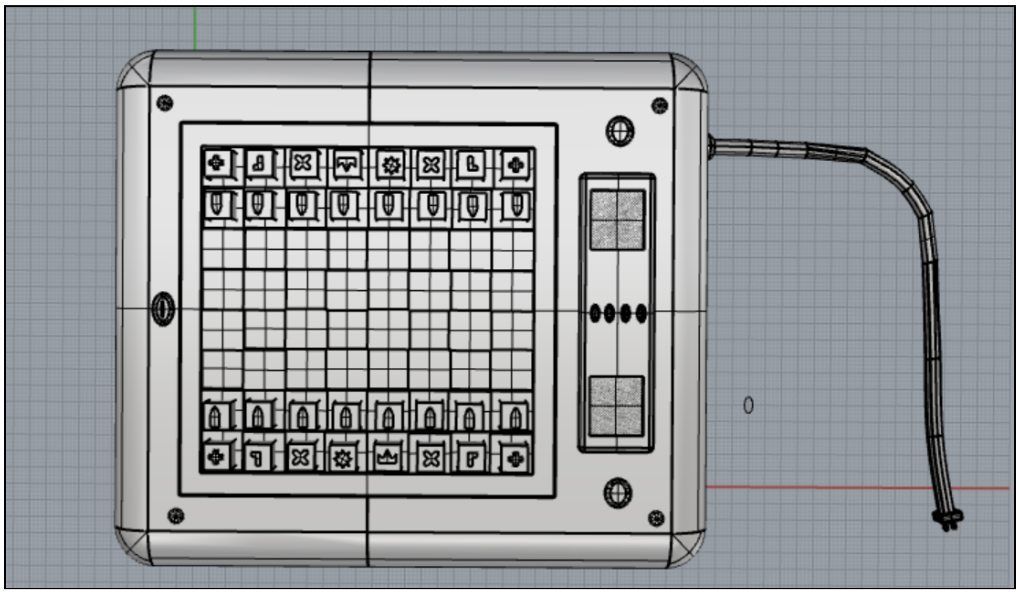


Figure 13: CAD Model of Board in Rhino 5, Top View

Figure 14 shows a top view of the overall board structure with appropriate textures, colors, and materials. Figure 15 provides a side view of it. The final model was created by importing the Rhino 5 CAD model into the Unreal Engine and applying materials, textures, and

lighting features to give a realistic image of the final concept. The board structure is constructed of hard plastic for durability and protection of the electrical components within the board. A black plastic mold protects the power cord for safety and insulation of the wires. The pieces are made of stainless steel, and the white pieces are overlaid with a wood coating to provide tactile differentiation. The black and white squares are also differentiated by placement at different heights. The white squares are raised above the black squares by 2mm. A close-up view of the pieces and squares is shown in Figure 16. Colors are used sparingly because they are unimportant for the blind user base. However, some simplistic coloring has been added for ease of use if a non-blind player that is unfamiliar with the tactile signifiers should use the board with a blind player. As shown, the team chose to color the white and black squares as well as the buttons.

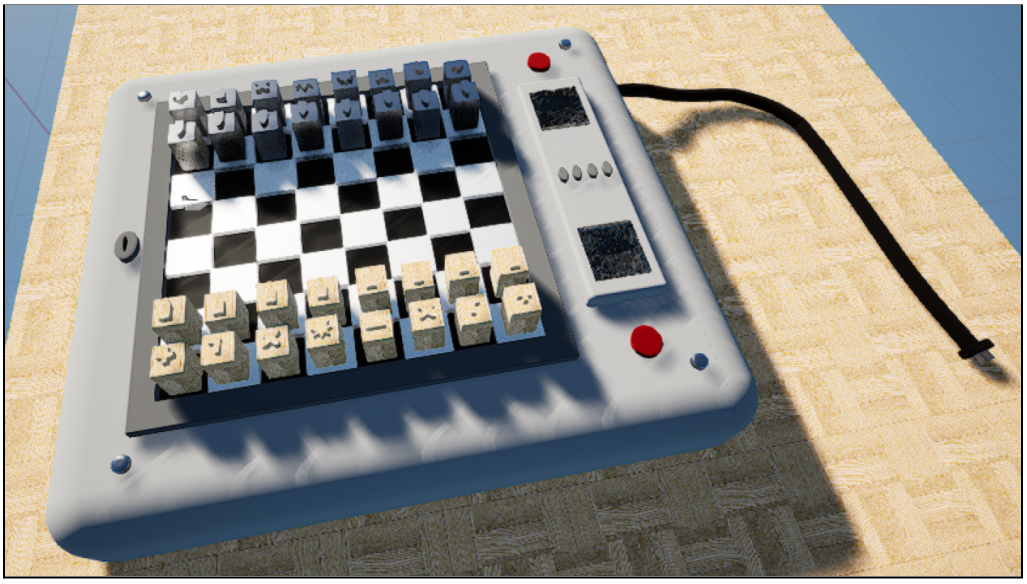


Figure 14: Final Rendered Board in Unreal Engine, Top View

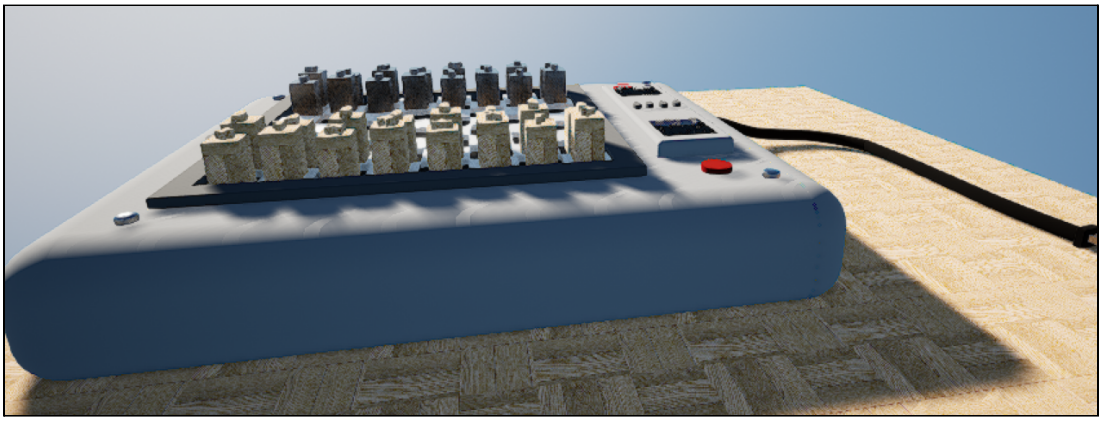


Figure 15: Final Rendered Design in Unreal Engine, Side View

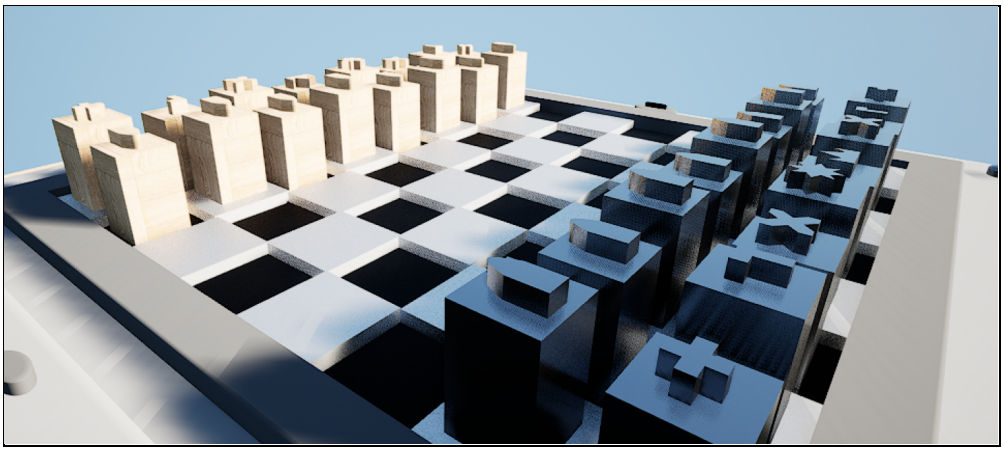


Figure 16: Close-up View of Pieces & Squares in Unreal Engine

6 Subsystem Analysis and Design

The goal of the project was to improve the gameplay of chess for the visually impaired. Our plan tackled this obstacle with our unique design of a responsive electromagnetic chessboard. Figure 17 shows how the technical design of the board was split into six subsystems. The user interface, constructed by Nico, manages the usage and functionality of the timer, buttons, and speaker. Under the jurisdiction of Carter, piece detection manages the RFID chips and converts the physical moves into data that can be manipulated and recorded by code. Piece movement, handled by Heather, controls the electromagnetic piece attachment and the release mechanism for when pieces need to be moved. Game functionality, by Julian, identifies legal move detection and computer chess engine moves for the computer to play against a player effectively. Power, managed by Eileen, distributes power to the other subsystems and models the electrical circuitry of the board. Game user feedback, designed by Brooke, manages the piece-vibrating mechanism that alerts the user of their opponent's last move.

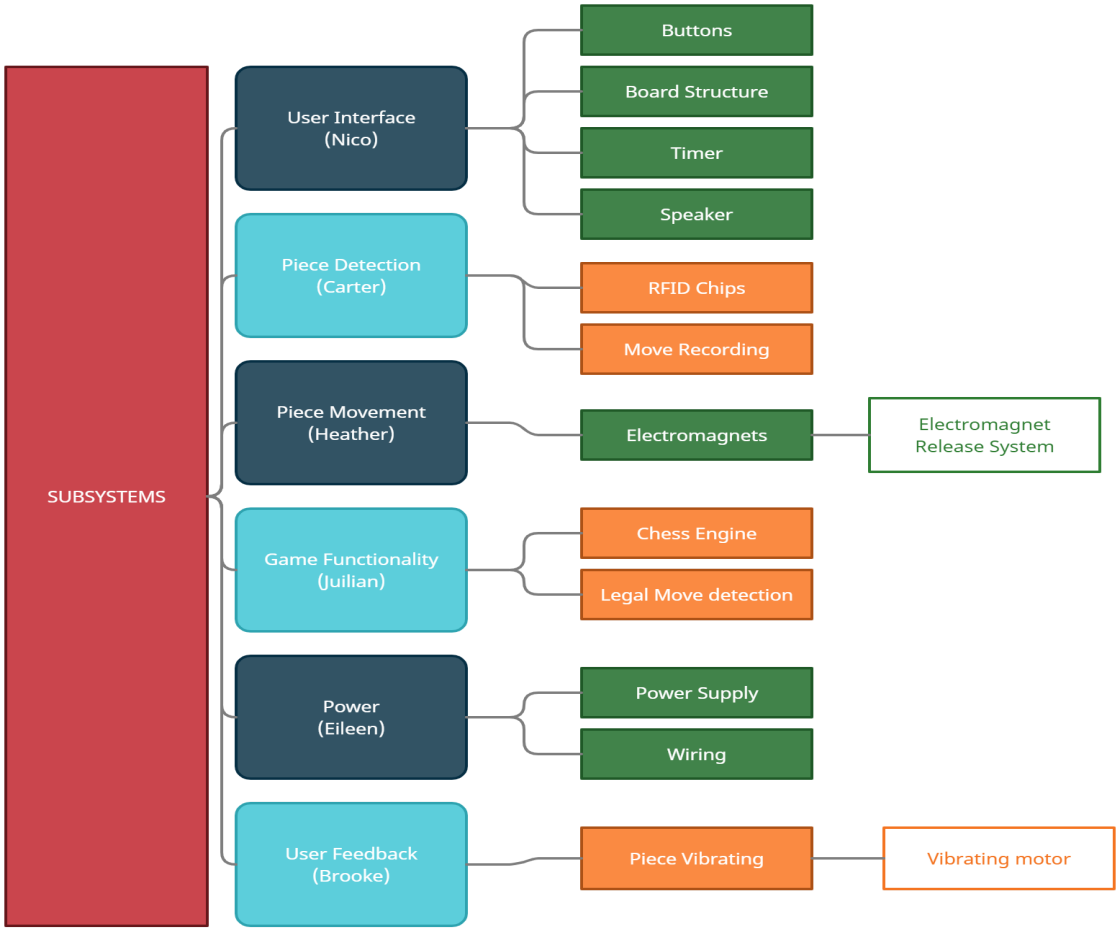


Figure 17: Subsystem Hierarchy Diagram

Though these subsystems are separated in the design process, they worked in conjunction to achieve the gameplay of chess for the blind both in tournament play and in CPU mode. All of these subsystems require inputs and outputs to or from other subsystems. For instance, the game feedback subsystem requires input from the game functionality subsystem, which is the code that determines which pieces need to be vibrating. This code requires input from the user interface subsystem as the mode selected by the user. This chain and others can be seen further in Figure 18, where the relationships between subsystems and their integration are illustrated. The lines connect the inputs and outputs of each subsystem, illustrating the flow of information. Specific tasks addressed by each subsystem were simulated through various packages of TinkerCad, C++, LTspice, and Python to validate each subsystem before integration.

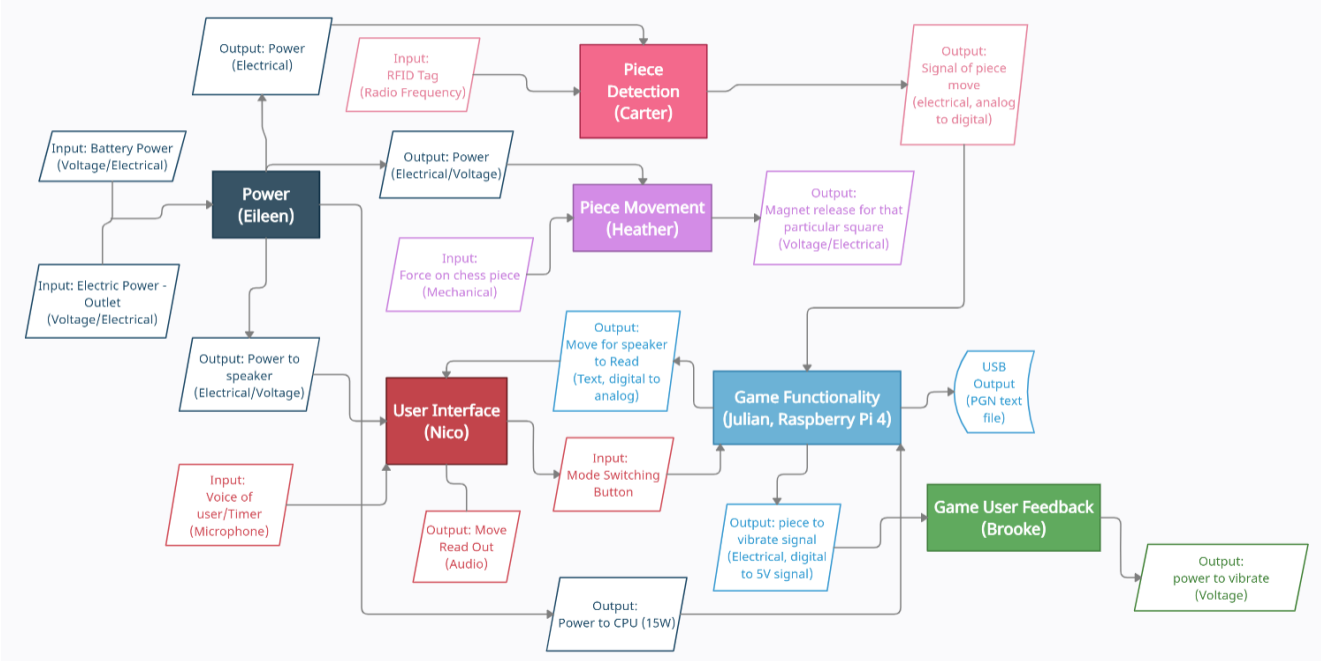


Figure 18: Subsystem Block Diagram

6.1 Subsystem 1 - User Interface

Prepared by: Nico Nigohosian

This project aimed to improve the experience of chess for the visually impaired by designing an appropriate interface that included tactile features, tournament play, and learning features. A model of this interface is shown in Figure 19. This project also aimed to make chess more accessible to students and competitive players. Features in the chessboard make the game easier to play with only sound and touch. Additionally, the chessboard has a built-in computer for training mode and tournament mode. All built-in modes of play are designed to provide sufficient audio, and physical feedback declared as necessities by members of the United States Blind Chess Association (USBCA) to minimize the difficulties experienced by the blind in the gameplay of chess.

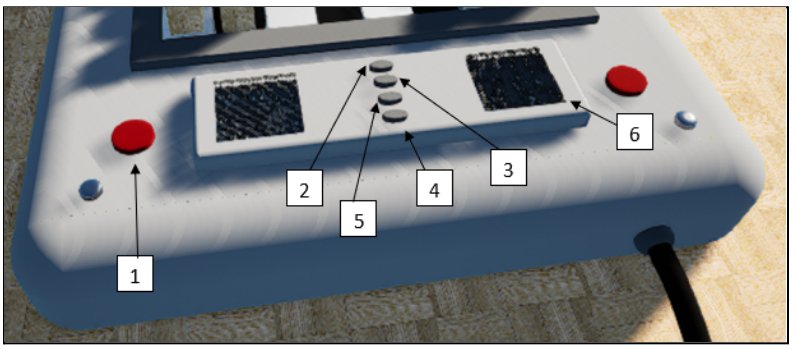


Figure 19: Chessboard Buttons

The user interface was created to support blind players with the proper guidance to play chess with ease. As seen in Figure 19, the user interface includes six buttons: switch, start, pause, Add 5 minutes, CPU/Player, and key describer/speaker. The user interface additionally included a speaker and internal timer. These features provided guidance through the audio outputs of the speaker. Players were directed on previous moves made, notifications of remaining time, functionality of buttons through the key describer, and any actions made during the game like pausing or starting the timer. This function also included switching from CPU to player mode or clicking buttons that started or paused the timer.

A demonstration of this subsystem was done through the simulation package of python code within the console spyder. The python code is included in Appendix J. In Figure 20, a condensed version of the code is shown. This imported the wX library and allowed the modification of a pop window. The timer created through the rest of my code appears with buttons and a clock that uses functions of the pyttsx3 library. The pyttsx3 library is used to transfer input signals to a variable for the speaker's audio outputs to get called on.

```
"""
Created on Thu Apr  8 21:05:02 2021

@author: Nigohn
"""

import time, threading, wx, os, sys
from subprocess import call
from os import system
import pyttsx3
import numpy as np
import random

#Creating a window for my program with a specified title and size
app = wx.App()
window = wx.Frame(None, title = "Nico's Chess Timer", size = (700,400))
panel = wx.Panel(window)
popup = wx.Frame(None, title = "Time's up!", size = (200, 150))
```

Figure 20: Python Code

When this code was called, it resulted in a pop-up window seen in Figure 21 that mimics the functions of the chessboard interface and the gameplay of a chess game.

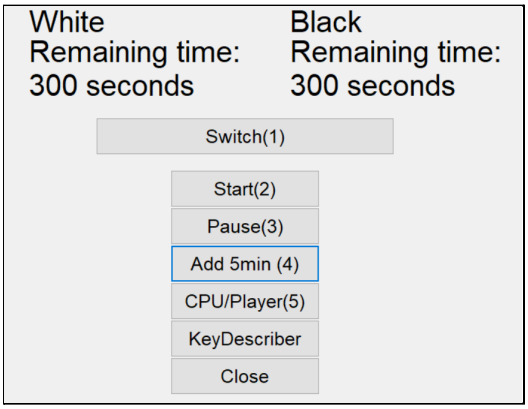


Figure 21: Chessboard Interface

The following test validated the demonstration of the subsystem: the User selected key describer function, which resulted in an audio output of "Button 1 is to switch to opponents timer alternating turns. Button 2 is to start the timer beginning the game. Button 3 is to pause this timer mid-game once started. Button 4 is to select your time interval of the game by inputting increments of 5 minutes. Button 5 is to choose between CPU or player opponent." This speaking function of the pysttsx3 library is demonstrated through the code seen in Figure 22. More complexity was added with creating the button and defining the button action to call a specific event.

```
def KeyDescriber(event):
    engine = pyttsx3.init()
    engine.setProperty('rate',170)
    engine.say(" Button 1 is to switch to opponents
    engine.runAndWait()
```

Figure 22: Speaking Engine

Whereas the second test was done, the input was a user selecting CPU or player mode. The output was the mode announced with the audio declaring "Player mode initiated" or "CPU mode initiated." Within the player mode of the system, a move is announced after each user clicks on each switch button. In CPU mode, after the white team switches the time to the opponent, the CPU is given 7 seconds to announce the counter move made by the CPU. This is shown in Figure 23. When the button is clicked, the audio output of the pyttsx3 engine again translates text input to audio output. The code for the switch function shows both the CPU and player function as logic of 1 and 0s through the variables "mode" and "turn," allowing them to toggle between each.

```
def switchTurn(event):
    global mode
    global turn
    global blackTime
    #pieces = (['knight'],['pawn'],['rook'],['bishop'])
    engine = pyttsx3.init()
    engine.setProperty('rate',99)

    if mode == 1:
        if turn ==0:
            engine.say(" Pawn position F2 moved to. F4")
            engine.runAndWait()
            turn = 1
        if turn == 1:
            time.sleep(5)
            blackTime = blackTime - 5.0
            engine.say(" Pawn position C7 moved to. C6")
            engine.runAndWait()
            turn = 0
    if mode ==0:

        if turn == 0:

            engine.say(" Pawn position F2 moved to. F4")
            engine.runAndWait()
            turn = 1

        else:
            engine.say(" Knight position B8 moved to. C6")
            engine.runAndWait()
            turn = 0
```

Figure 23: Switch Function(Mode & Turn)

In the third subsystem test, the user selected the 'Add 5 min' button, and the output is 300 seconds added to both clocks with an audio output of "Added 5 minutes to timer." In the fourth test, the user selected start, and the audio output was "Timer has begun, Game has started! May the best chess player win." The timer is executed and deducted by one every second. The code of these functions was also similar in design to the speaking event seen in Figure 22. The difference was that a time variable called the code instead of the clicking of a button. The fifth test entailed the user selecting the switch button. The result of this event was that the previous move by the player was announced with audio, and the opposite team's timer started to count down, also seen in Figure 23. The user interface was also created to notify players when there is a minute remaining left in the game and when there is no time left. This was done so when the input of the remaining time is equal to 60, the audio output is: "1 minute remaining of Black's (or white's) time." When the clock strikes zero, an audio output reads, "Blacks (or whites) time is up … Game Over." This is seen in Figure 24, which concluded the game of chess for the user.

```
def double_timer():
    global whiteTime, blackTime
    global whiteRemainingTime, blackRemainingTime
    global turn, paused


    while whiteTime > 0 and blackTime > 0:


        if turn == 0 and paused == 0:
            whiteTime = whiteTime - 1.0
            whiteRemainingTime.SetLabel("Remaining time: \n%d seconds" %whiteTime)
            timeFeed()

        elif turn == 1 and paused == 0:
            blackTime = blackTime - 1.0
            blackRemainingTime.SetLabel("Remaining time: \n%d seconds" %blackTime)
            timeFeed()

        time.sleep(1)




    if paused == 0:
        popup.Show(True)
        call(["osascript -e 'set volume output volume 50'"], shell = True)


        if whiteTime <= 0:
            pass
            engine = pyttsx3.init()
            engine.setProperty('rate',130)
            engine.say("Whites time is up..Game Over")
            engine.runAndWait()
        if blackTime <= 0:
            pass
            engine = pyttsx3.init()
            engine.setProperty('rate',130)
            engine.say("Blacks time is up..Game Over")
            engine.runAndWait()

def timeFeed():
    global whiteTime, blackTime
    global whiteRemainingTime, blackRemainingTime
    global turn, paused
    engine = pyttsx3.init()
    engine.setProperty('rate',170)
    if whiteTime > 59 and whiteTime < 60:
        engine.say(" 1 minute remaining of white's time")
        engine.runAndWait()
    if blackTime > 59 and blackTime < 60:
        engine.say(" 1 minute remaining of black's time")
        engine.runAndWait()

def switchTurn(event):
    global mode
```

Figure 24:Double Timer Engine

6.2 Subsystem 2 - Game Functionality

Prepared by: Julian Matthews

The purpose of the game functionality subsystem is to provide all necessary features of the game of chess itself, including legal move checking and move recording. Additionally, one of the customer requirements was a learning mode to make chess more accessible to new players. When in this mode, a user can play the computer and the computer move is read aloud. This

subsystem interacts closely with the user interface, piece detection, and game feedback subsystems.

To provide the legal move checking and chess engine, Stockfish 13 was chosen because it is open source, written in C++, and is the strongest chess engine in the world. A large majority of integrating this subsystem to the other subsystems of the chessboard was understanding and modifying the input parsing used by Stockfish. This is mostly located in the uci.cpp and uci.h (Universal Chess Interface) files of the open-source code. To add the chessboard functionality, 7 helper functions were added, and the main input parsing function UCI::loop was modified to accept additional input commands such as getpieceraise which outputs the piece to be vibrated on the chessboard. This function was renamed UCI::ChessboardLoop, as shown in Figure 25.

```
void UCI::ChessboardLoop(int argc, char* argv[]){
  // Waits for a command from stdin, parses it and does the appropriate operation.
  // Modified from UCI::loop in original code.

  // Object Setup
  Position pos;
  string token, cmd;
  string PGN;
  vector<string> PGN_vec;
  string PGN_command = "startpos moves ";
  StateListPtr states(new std::deque<StateInfo>(1));

  pos.set(StartFEN, false, &states->back(), Threads.main());

  // Parsing
  for (int i = 1; i < argc; ++i){
      cmd += std::string(argv[i]) + " ";
    }

  do {
    // Waiting for input
    if (argc == 1 && !getline(cin, cmd)){
            cmd = "quit";
      }

        istringstream is(cmd);

        token.clear();
        // Avoid a stale if getline() returns empty or blank line
        is >> skipws >> token;

// Operations based on inputs:
        if (token == "quit" || token == "stop"){
          // Quit
          Threads.stop = true;

        }else if (token == "printposition"){
          // Prints current position
          istringstream is 2(PGN command);
```

Figure 25: ChessboardLoop Function

The first input the code receives is the most recent move from the piece detection subsystem in PGN form such as "e2e4", describing the square the chess piece was moved to and from. An example is shown in Figure 26. The second input is when the user interface has been set to computer vs. player mode, the user interface can request the best computer move for the current board position. An example is shown in Figure 27.

This subsystem has four main outputs. The first is legal move checking, which is read aloud by the board's integrated speaker. If a move is illegal given all chess rules or is incorrectly formatted due to a system error the ChessboardLoop function outputs either "Invalid move format" or "Not a legal move". If the move is legal, the prints to the terminal the updated current position, as shown in Figure 26.

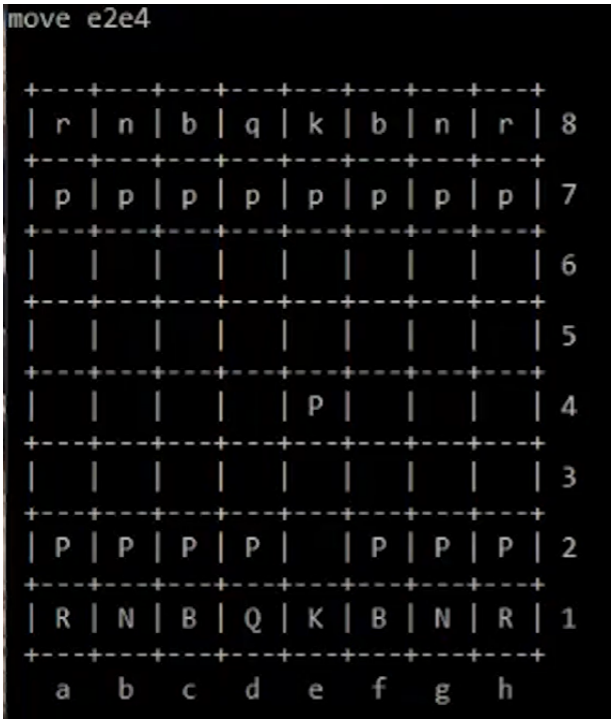


Figure 26: Legal Move Output

The next main output is the computer's best move. When the computer move is requested, Stockfish is given the current position and generates a depth 22 move (analyzes 22 moves into the future) in around 3-10 seconds depending on the computer. This is shown in Figure 27. Given less processing power, the depth can be lowered by 1 to 3 to meet the requirement of an under 5-second move search. Given that each level of depth takes exponentially longer than the previous level, depth 19 or 20 is significantly faster than 22. At around depth 20, chess engines have an elo of around 2900 (Ferreira, 2013). This is around the same strength as the current best chess player in the world. By decreasing depth or Stockfish UCI difficulty settings, different computer difficulty levels could be easily integrated into the chessboard. However, due to time constraints, this was not done.

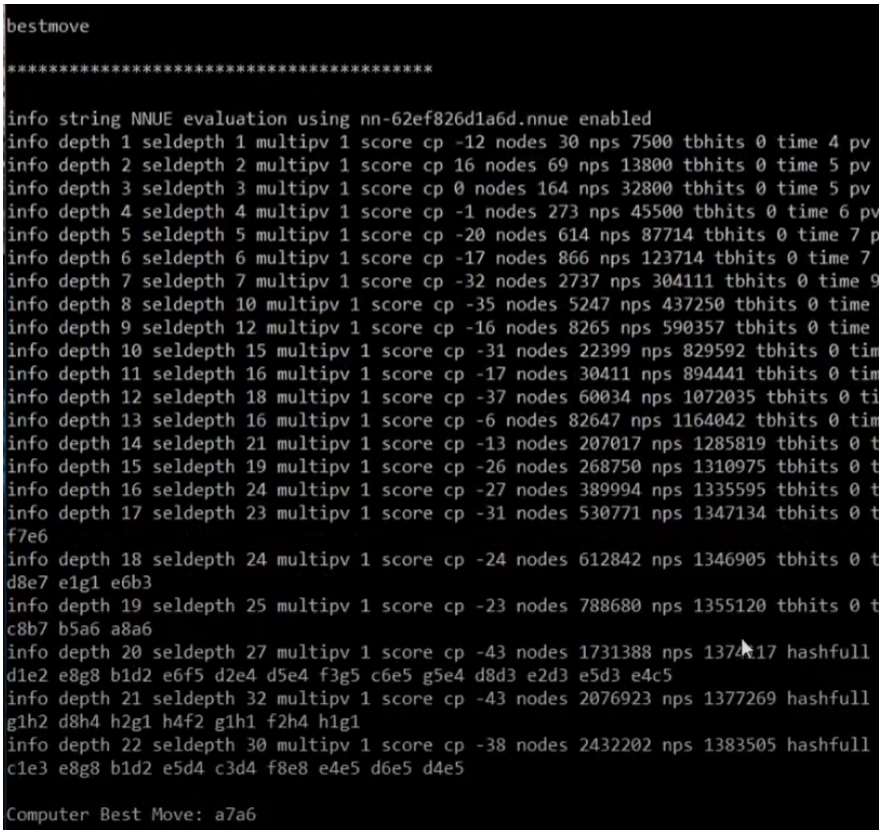


Figure 27: Depth 22 Move Search

The third main output is the square to be vibrated by the game feedback subsystem. The subsystem takes the last square from the PGN move format and converts this to both a decimal and binary number from 0-63 corresponding to the square on the board, as shown in Figure 28. The binary output is to simulate how the subsystem would operate on a Raspberry Pi, with each bit representing one of the GPIO (general-purpose input/output) pins.

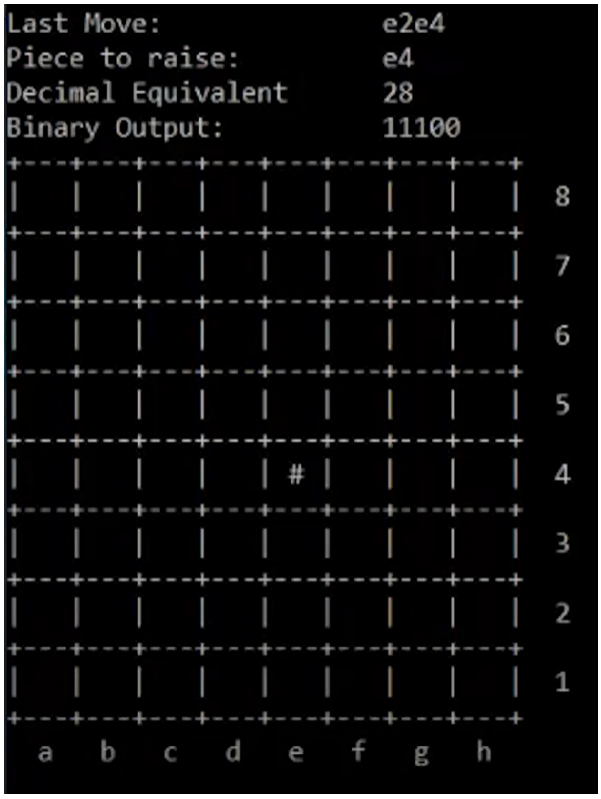


Figure 28: Piece Vibration Output

The final main output is the recorded moves for the whole game. Each time a legal move is made, it is stored in a vector. This vector can be requested at any time and will be formatted as text to be output via a USB port on the Raspberry Pi. Additionally, the current game position at all times is represented in Fen notation. In this notation, each slash represents a new row of the chessboard. The various characters represent the positions of the pieces, and the w at the end means it is white to move.

```
Fen: r1bqkbnr/1ppp1ppp/p1n5/1B2p3/4P3/5N2/PPPP1PPP/RNBQK2R w KQkq - 0 4
PGN vector: 1. e2e4 2. e7e5 3. g1f3 4. b8c6 5. f1b5 6. a7a6
```

Figure 29: PGN Vector

The two subsystem inputs, PGN formatted moves and computer best move requests are simulated as user command inputs. If the subsystems were fully implemented these commands could easily be replaced by interaction between the several programs for the chessboard. The parsing of these commands is done by the do operator in the ChessboardLoop function. This waits for the user to press enter on a command, and then stores it in a variable. The following if else statements check if the input is a viable command, and if so the corresponding sequence of functions and operations is executed. The main Stockfish functions that the custom written ChessboardLoop function uses are position and go. The position function takes in moves as an

input and sets up the position internally in Stockfish. The go function takes an integer depth as an input and runs the engine on the set position. The main purpose of the code written for the chessboard is making these two functions compatible with the inputs and outputs required by the connected subsystems.

The code written for this subsystem also includes seven helper functions. These functions come before the ChessboardLoop function and do operations such as converting a move to a decimal and binary number, as well as parsing PGN strings and printing out helpful chessboard visualizations. Two of these functions are shown in Figure 30.

```
//               Custom written and modified for RPI IED Spring 2021 Group 1 - Blind Chessboard
//             |-------------------------------------------------------------------------------|

int moveToNumber(string themove){
  //Converts a move such as "e2e4" to a number from 0 to 63 corresponding
  //to the 64 squares ordered bottom left to top right
  string secondsquare = themove.substr(2, 2);
  int index = toupper(secondsquare[0]) - 'A';
  return index + 8 * (stoi(secondsquare.substr(1,1))-1);
}

string printPieceRaising(string thesquare){
  // Prints visualization of the single square that is raised on the chessboard
  string output;
  std::string s = "+---+---+---+---+---+---+---+---+\n";
  vector<string> ranks{"8", "7", "6", "5", "4", "3", "2", "1"};
  vector<string> files{"a","b","c","d","e","f","g","h"};
  int i = 8;
  for (vector<string>::iterator it = ranks.begin(); it != ranks.end(); it++){
    output.append(s);

    for (vector<string>::iterator itr = files.begin(); itr != files.end(); itr++){
      output.append("| ");
      if (*it == thesquare.substr(1, 1) && *itr == thesquare.substr(0, 1)){
        output.append("#");
      }else{
        output.append(" ");
      }
      output.append(" ");
    }
    output.append("|");
    output.append("  ");
    output.append(to_string(i));
    output.append("\n");
    i--;
  }
  output.append(s);
  output.append("  a   b   c   d   e   f   g   h\n");
  return output;
}
```

Figure 30: Helper Functions

The concept development for this subsystem was heavily related to how the Stockfish code already worked. Initially, the idea was to use the already existing UCI in a separate program. However, this proved to be overcomplicating and instead, it was decided to modify the existing user interface code. The testing done for this subsystem was vigorous error checking of corner cases and large sample size inputs of moves. Additionally, by playing through games and entering many illegal and incorrectly formatted moves into the move command, many errors were found and fixed within the code. Each of the outputs was tested for many inputs as well.

For instance, the text to board position calculation such as "e4" to "11100" was checked by hand calculations.

6.3 Subsystem 3 - Game User Feedback

Prepared by: Brooke Zatowski

When the user plays chess in CPU mode, they play chess against the computer, which requires them to move the computer's chess pieces manually. The purpose of the game user feedback subsystem is to alert the user of the current position of the computer's chess piece and the position they have to move the chess piece to. This was done by vibrating the chess square the chess piece was currently on for ten seconds, then vibrating the chess square the chess piece was supposed to move to for ten seconds.

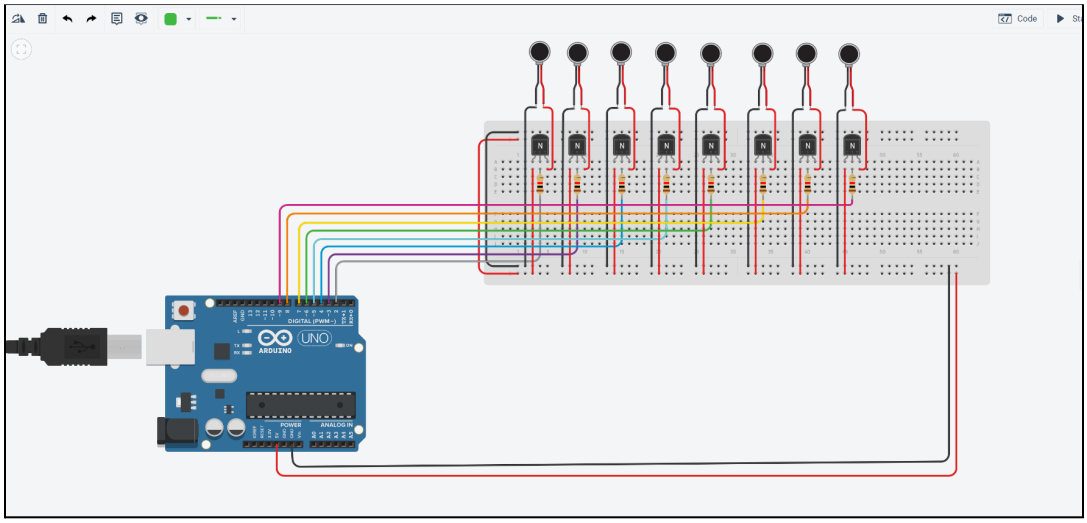


Figure 31: Arduino circuitry model

This was simulated by creating Arduino circuits in Tinkercad. Figure 31 shows the circuitry model for one row of the chessboard (the other seven rows of the chessboard have identical circuitry). The first row of the chessboard contains positions A1- H1. Eight vibration motors were wired to the 5V Arduino input pin and the ground pin through the breadboard. There are eight vibration motors because each row of the chessboard has eight squares. Each vibration motor was also connected to a transistor and a resistor wired to a specific GPIO pin (pins 2-9).

The input for this subsystem was a 6bit binary signal sent from the game functionality subsystem. There are 64 possible binary signals, for there are 64 positions on a chessboard. Code was written in C++ to assign each binary signal to a specific GPIO output pin and vibration motor. For example, GPIO output pin 2 was assigned to position A1 on the chessboard (the first

vibration motor from the left in Figure 31). Additional code was written to output the 4.18 V received from the power subsystem to the GPIO pin assigned to the binary input signal given. The code was written so that the vibration motor was on for ten seconds total (7 intervals of power on for one second and power off for 0.5 seconds). For example, when the circuit receives the 6bit binary signal B000010, which represents position B1 on the chessboard, the second vibration motor from the left in Figure 31 vibrates for 10 seconds. (Note, C++ requires a B to be placed before each signal to mark it as binary.) The code for the circuit can be found in Appendix 21.

6.4 Subsystem 4 - Piece Detection

Prepared by: Carter Wynn

The design of this chessboard combined a physical, tactile chessboard with the brain of an open-source chess engine. One major issue was bridging the gap between the physical game and the computer game. The chess engine needed to know where the physical chess pieces were to function properly. The piece detection subsystem determines the location of each chess piece after each move and provides that information to the chess engine in the game functionality subsystem.

To distinguish each chess piece it was decided to use RFID tags. Passive RFID tags are placed at the bottom of each chess piece. Then RFID readers are embedded into each square on the chessboard. This would allow for a computer code to determine the identity and location of all chess pieces. This choice created a slew of new problems that needed to be solved. Namely, how to power sixty-four RFID readers and how to read information from sixty-four different readers at the same time.

To tackle these problems, it was decided to power each RFID reader only for enough time to send its data. Each reader would then be connected to the same RS232 port on the internal Raspberry Pi. By individually powering each RFID reader, they all can be connected to the same input port on the Raspberry Pi as only one reader will be sending data at any given time. This solves the issue of powering each reader and reading the information from each RFID reader

The first part of the piece detection subsystem is an Arduino seven-bit binary. This Arduino counts from one to sixty-four in binary. TinkerCAD was used to simulate the Arduino code and its outputs. The code sets up seven GPIO pins to act as the seven bits of binary. The eighth GPIO pin is set up as an input. It takes an input from a button that acts as the switch to the competitive clock. The switching of the clock means that a move has been made, therefore the computer needs to know the new position of the piece that was moved. The depressing of the button triggers the counting code. The code activates the GPIO pins corresponding with the binary count. For example, nine in binary equals 001001 so when the count reaches nine the Arduino activates pin one and four. The set up and output of the code for the count of one is displayed in Figure 32. These pins stay active and output five volts for 330 microseconds as that

is the length it takes for the RFID reader to send its block of data. This full circuit can be seen in Figure 33.

```
void setup()
{
  Serial.begin(9600);
  pinMode(8,INPUT_PULLUP);
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);

}

void loop()
{
  int sensorVal = digitalRead(8);
  Serial.println(sensorVal);
  digitalWrite(1, HIGH);
  digitalWrite(2, HIGH);
  digitalWrite(3, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);

  if (sensorVal == LOW) {
    digitalWrite(1, HIGH); //1
    digitalWrite(2, LOW);
    digitalWrite(3, LOW);
    digitalWrite(4, LOW);
    digitalWrite(5, LOW);
    digitalWrite(6, LOW);
    digitalWrite(7, LOW);
    delay(.33);
    digitalWrite(1, LOW); //2
```

Serial Monitor

Figure 32: Arduino Code

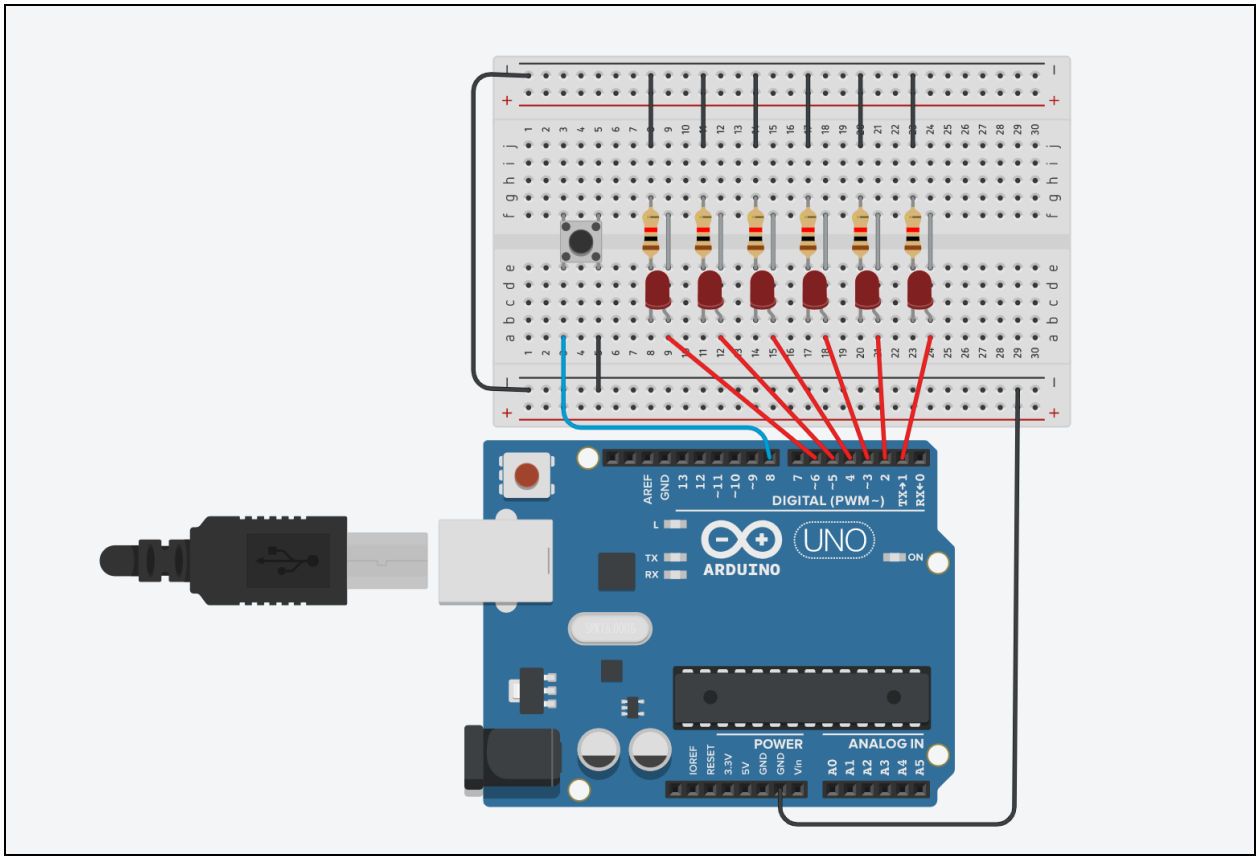


Figure 33: Arduino Circuit

The Arduino's GPIO pins output into a seven to sixty-four binary decoder which is simulated in LTspice. This decoder takes a seven-bit binary signal and converts it into sixty-four separate outputs. The output from the binary decoder links to sixty-four RFID readers. The output of the RFID readers connects into a common RS232 port on the Raspberry Pi. This circuit can be seen in Figure 34. A close-up of the RFID reader, its antenna, and how it connects to the RS232 port can be seen in Figure 35.

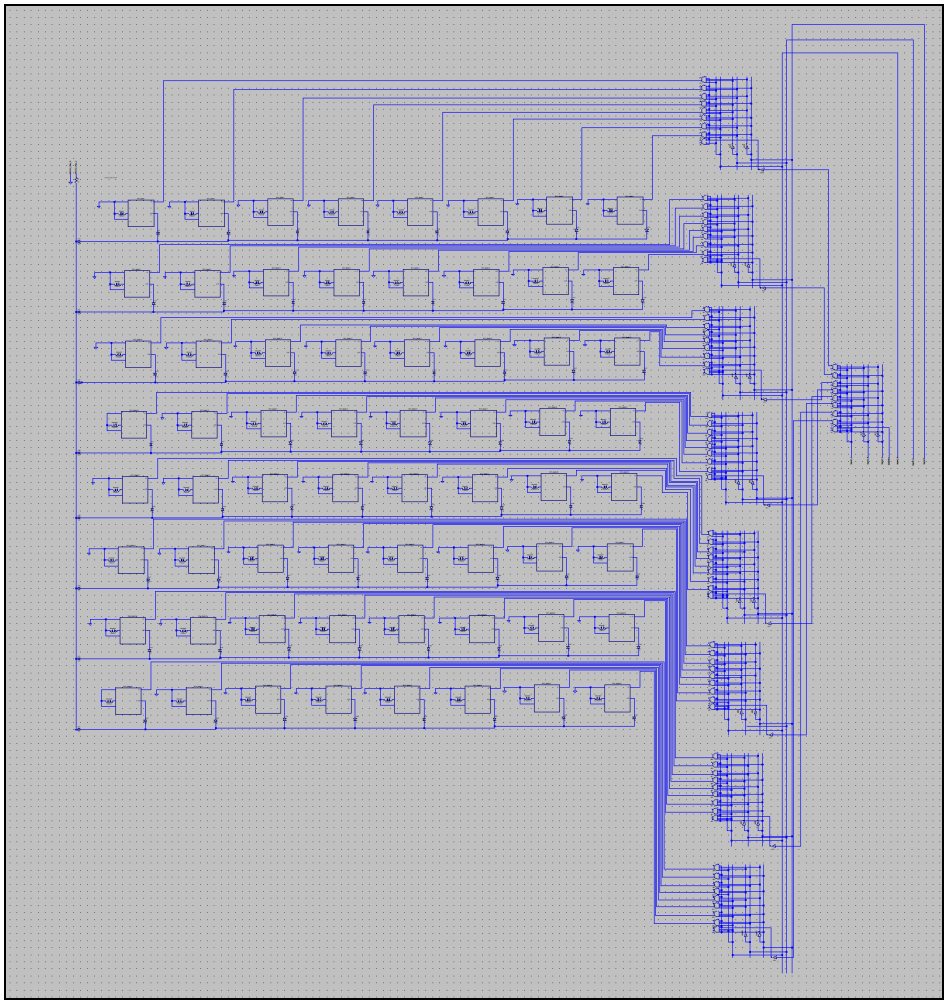


Figure 34: Full LTspice Circuit

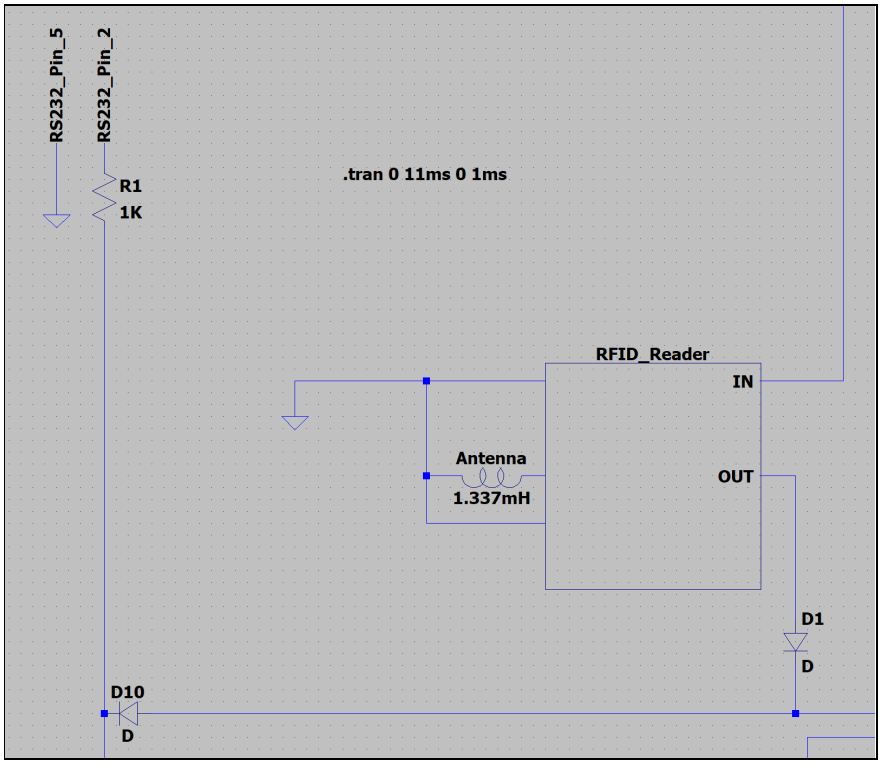


Figure 35: Close-up of the RFID reader

The data outputted from the RFID reader is a UID or Unique Identifier. Each RFID tag has a UID that corresponds with it. This UID has to be deciphered from the data sent by the RFID reader. A code was written in Python with Spyder that deciphers UIDs and matches them with a bank of known UIDs. This code takes in a matrix of UIDs that corresponds with a position on the chessboard. It then matches the UIDs in the matrix with a bank of known UIDs to determine piece identity and location. This new matrix of piece locations is compared with the previous matrix of piece locations to determine the move made. The program then outputs the previous location of pieces, the location of the pieces after the move, and the text of the move in PGN format. The full move detection code can be seen in Appendix L. This PGN string is sent to the chess engine to inform it of the most recent move. An example of the output of the move detection code can be seen in Figure 36.

```
Original:
['bR', 'bP', '0', '0', '0', '0', 'wP', 'wR']
['bN', 'bP', '0', '0', '0', '0', 'wP', 'wN']
['bB', 'bP', '0', '0', '0', '0', 'wP', 'wB']
['bQ', 'bP', '0', '0', '0', '0', 'wP', 'wQ']
['bK', 'bP', '0', '0', '0', '0', 'wP', 'wK']
['bB', 'bP', '0', '0', '0', '0', 'wP', 'wB']
['bN', 'bP', '0', '0', '0', '0', 'wP', 'wN']
['bR', 'bP', '0', '0', '0', '0', 'wP', 'wR']


Move:
['bR', 'bP', '0', '0', '0', '0', 'wP', 'wR']
['bN', 'bP', '0', '0', '0', '0', 'wP', 'wN']
['bB', 'bP', '0', '0', '0', '0', 'wP', 'wB']
['bQ', 'bP', '0', '0', 'wP', '0', '0', 'wQ']
['bK', 'bP', '0', '0', '0', '0', 'wP', 'wK']
['bB', 'bP', '0', '0', '0', '0', 'wP', 'wB']
['bN', 'bP', '0', '0', '0', '0', 'wP', 'wN']
['bR', 'bP', '0', '0', '0', '0', 'wP', 'wR']


e2 e4
```

Figure 36: Move Detection Code Output

6.5 Subsystem 5 - Piece Movement

Prepared by: Heather Converse

One of the main issues with existing chess boards for the blind was that the chess pieces knocked over easily while the user made a move. This was indicated by 10 members of the United States Blind Chess Association (USBCA) that the team spoke to when considering customer needs. The USBCA members explained that current chess boards they used had a hole and peg design which did not properly keep the chess pieces in place. They expressed concern about choking hazards for pets and children if the pieces fell on the ground due to ineffective attachment mechanisms. After brainstorming to solve this issue, the team decided the best option to secure the pieces was to use magnets for each of the squares.

The main purpose of this subsystem is to control the electromagnets for each square on the chessboard. When the user wants to move a piece, they press down on the particular chess piece. Thus, there is an input force that turns off the magnet for each square. The other squares with no force will still have the magnets turned on so the other chess pieces will stay in place on

the board. To make this possible, a solenoid wrapped with wire was placed under each square of the chessboard and powered by 4.18 Volts from the Power subsystem.

To simulate this, two different programs were used. First, Tinkercad simulated the voltage a square received based on the force present. The circuit used a 220 ohm resistor, 10k ohm resistor, an Arduino, one LED, and a force sensor. The force sensor represented the input force by the user while the LED represented the output voltage. To power the system, the Arduino provided 4.18 Volts and was coded using C++. The code is shown below in Figure 37 and includes guided comments that explain the function of each line.

```
int fsrAnalogPin = 0; // FSR is connected to analog 0
int LEDpin = 11;      // Red LED connected to pin 11 (PWM pin)
int fsrReading;      // the analog reading from the FSR resistor divider
int LEDbrightness;

void setup(void) {
  Serial.begin(9600);
  pinMode(LEDpin, OUTPUT);
}

void loop(void) {
  fsrReading = analogRead(fsrAnalogPin);
  Serial.print("Analog reading = ");
  Serial.println(fsrReading);

  if (fsrReading > .10)
  {
    LEDbrightness = 0;
  }
  else
  {
    LEDbrightness = 255;
  }
  // LED turns off above 0.1N
  // From 0-0.9N the LED is on
  // Changed the range from the analog reading (0-1023) down to the range
  // used by analogWrite (0-255) with map!
  //LEDbrightness = map(fsrReading, 0, 1023, 0, 255)

  analogWrite(LEDpin, LEDbrightness);

  delay(100);
```

Figure 37: Arduino Code

When the user did not press down on a square, there was no input force, which was defined as ranging from 0 to 0.1 Newtons. This resulted in an output voltage of 4.18 Volts and was shown by the LED remaining on, which represented the electromagnet staying on. The LED/electromagnet remained off until the input force was no longer applied. This is demonstrated below in Figure 38 when a force of 0 Newtons was read by the force sensor and the LED remained on.

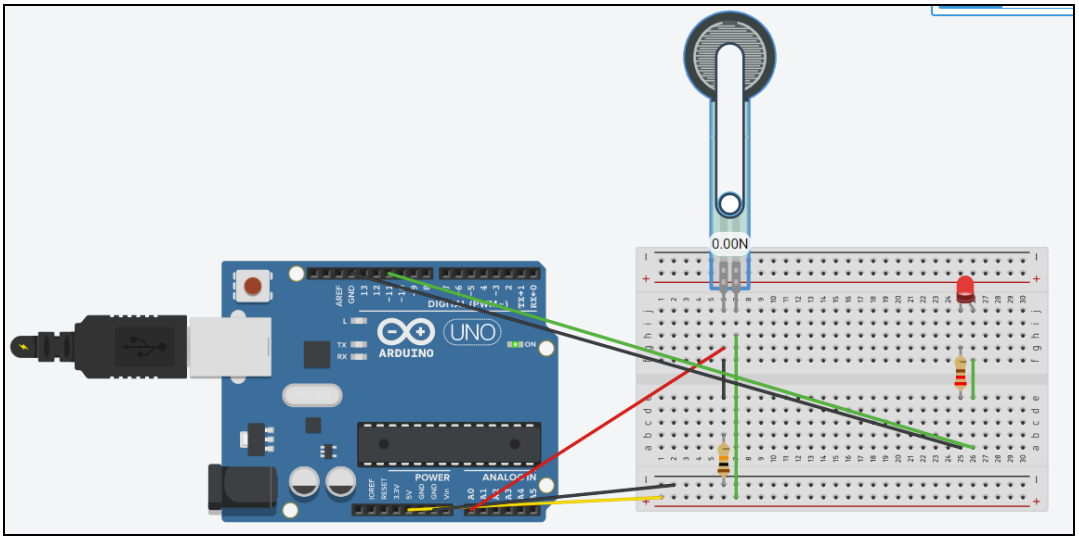


Figure 38: Tinkercad, Electromagnet/LED On

In the next case, a user pressed down on the square and there was an input force greater than 0.1 Newtons. This resulted in an output voltage of 0 Volts, depicted by the LED turning off which represented the electromagnet turning off. Figure 39 shows an input force of 0.46 Newtons and the LED remaining off.

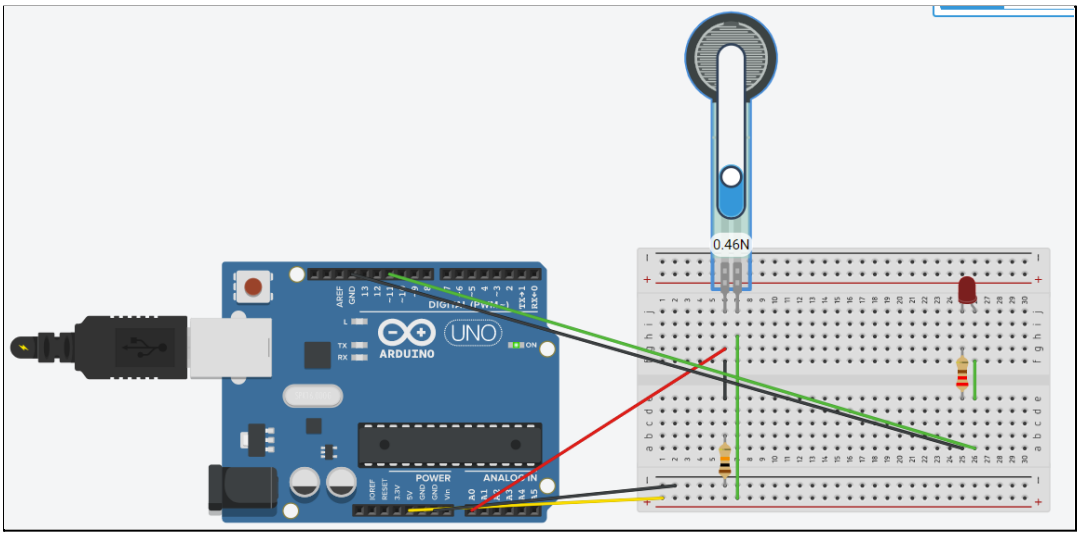


Figure 39: Tinkercad, Electromagnet/LED Off

Subsequently, Python code was used to show what square a force was being applied to and the corresponding output. An array was used to simulate the 64 squares on the chessboard. The first 8 x 8 matrix represented the input force and the second matrix represented the output voltage. The code, which contains comments explaining each line, is shown in Figure 40.

```python
import numpy as np
import random

A = np.array([[1,2,3,4,5,6,7,8],[9,10,11,12,13,14,15,16],
              [17,18,19,20,21,22,23,24],[25,26,27,28,29,30,31,32],
              [33,34,35,36,37,38,39,40],[41,42,43,44,45,46,47,48],
              [49,50,51,52,53,54,55,56],[57,58,59,60,61,62,63,64]])

F = np.zeros([8,8]) #F is the Force the user applies in Newtons (placeholder)
V = np.zeros([8,8]) #V is the output Voltage in Volts (placeholder)

for i in range(1):
    F[random.randint(0,7),random.randint(0,7)] = .3
    #Chooses a random squares force to be 0.3 N

print(F) #Prints the matrix that shows what square the force is applied to

for i in range(len(F)):
    for j in range(len(F[i])):
        if F[i][j] < 0.1:
            V[i][j] = 4.18
        else:
            V[i][j] = 0
    #Checks each squares applied force, if the Force is < 0.1 N then
    #output voltage = 4.18V
    #If Force is > 0.1 N then the output voltage = 0V

print(V) #Prints the matrix that shows the square where the magnet is
         #turned off (output voltage = 0)
```

Figure 40: Spyder Code

An example is shown below in Figure 41.

```
[[0.  0.  0.  0.  0.  0.  0.  0.3]
 [0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.  0. ]]
[[4.18 4.18 4.18 4.18 4.18 4.18 4.18 0.  ]
 [4.18 4.18 4.18 4.18 4.18 4.18 4.18 4.18]
 [4.18 4.18 4.18 4.18 4.18 4.18 4.18 4.18]
 [4.18 4.18 4.18 4.18 4.18 4.18 4.18 4.18]
 [4.18 4.18 4.18 4.18 4.18 4.18 4.18 4.18]
 [4.18 4.18 4.18 4.18 4.18 4.18 4.18 4.18]
 [4.18 4.18 4.18 4.18 4.18 4.18 4.18 4.18]
 [4.18 4.18 4.18 4.18 4.18 4.18 4.18 4.18]]
```

Figure 41: Input and Output Matrices

The first matrix in Figure 41 shows an input force of 0.3 Newtons applied to the 8th row up, 8th column from the left. This corresponds to position h8 on the chessboard. The other

squares in the matrix have an input force of 0 Newtons because no force was applied to those squares. The second matrix shows an output voltage of 0 Volts on the corresponding square. Since the user applied a force to square h8, the electromagnet turned off just for h8, which allowed the user to move the chess piece. The output voltage remained 4.18 Volts for all of the other squares because no force was applied to them, so the electromagnets stayed on.

Electromagnet Calculations

The force of the electromagnets was calculated to ensure it would be strong enough to keep the chess pieces in place. The following steps show the calculations (Banas, 2018).

$$F = (n * i)^2 * \mu_0 * \frac{a}{2(g^2)}$$

Where:

F = *Force of Electromagnet*
N = *Number of turns of the wire* = 687 turns
$\mu_0 = magnetic\ constant = 4\pi * 10^{-7}$
I = *current* = 4.18 Volts
A = *cross sectional area* = $\pi(.0125)^2$
g = *length of gap between solenoid and metal* = 1.5

1. convert the current from Volts to Amps:

Power = 1800 W

$$amps = \frac{watts}{volts} = \frac{1800\ W}{4.18\ V} = 430.62\ Amps$$

2. Plug in the numbers to solve for F

$$F = (687 * 430.6\ A) * (4\pi * 10^{-7}) * \frac{\pi(.0125)^2}{2(9.18^2)}$$

F = 12 Newtons

The Force of each electromagnet on the board is 12 Newtons which is equivalent to 2.7 Pound-force.

6.6 Subsystem 6 - Power

Prepared by: Eileen Beres

The chessboard required power distribution to four of the subsystems. The purpose of the power subsystem was to create the appropriate circuitry to take in an input voltage from a standard U.S. outlet and distribute output voltages to each of the subsystems requiring an

electrical power input. LTSpice was used to simulate a circuit including all of the various components needing power in our design.

The board was designed to plug into a standard U.S. outlet via an insulated power cord. The standard U.S. outlet has an output of approximately 120V, which was the power subsystem's input. The target outputs for the power subsystem were delivering approximately 5V to each subsystem requiring it. A transformer, consisting of two inductors coupled together, was placed into the circuit next to the voltage source to step down the voltage. Diodes were also added in order to step down the voltage even further in order to reach the desired output of approximately 5V. A zoomed-in shot of the transformer and diodes in the circuit are shown in Figure 42. The actual value of the output voltage delivered to each subsystem was slightly off from 5V but within 1V of tolerance. Each of the other subsystems were adapted for the values produced in the simulation to account for the slight differences in output.

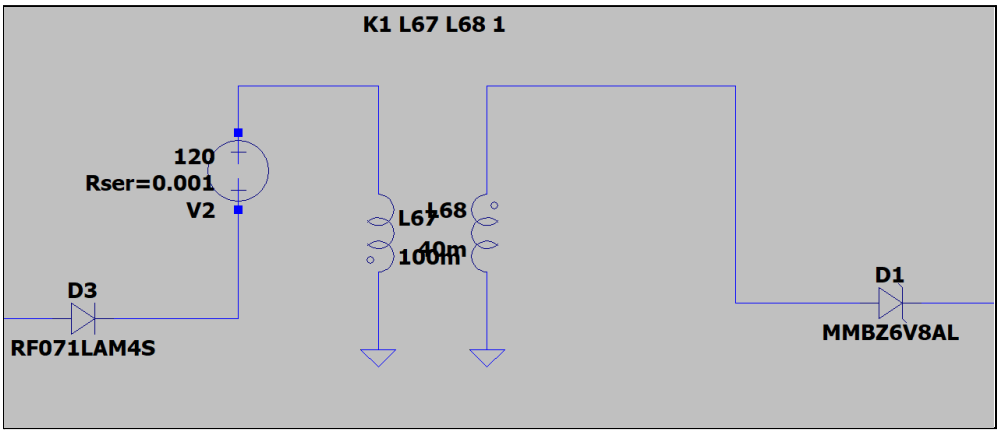


Figure 42: Voltage Source, Transformer, and Diodes

For the piece movement subsystem, each of the 64 electromagnets (one for every square on the board) required 5V to be delivered to it. In the circuit, each electromagnet is composed of an inductor and a resistor in series. The electromagnets were configured in parallel to receive 4.18V each. The output voltage of the electromagnets was about 5.44V. For the game feedback subsystem, each of the 8 Arduinos required approximately 5V of power to be delivered from the power subsystem. The Arduinos are represented by a load sandwiched between two 10 ohm resistors in series in the circuit. The circuitry of the Arduino itself is gone into more depth in the game feedback subsystem's section. The Arduinos were configured in parallel as well and 4.18V of power was delivered to each of them. Their output voltage was about 5.44V. For the piece detection subsystem, a single Arduino required approximately 5V of power to be delivered to it via the power subsystem, which was actually 5.44V in the simulated circuit. The Arduino was composed of a resistor and load in series. The circuitry of the Arduino itself as it relates to its functionality as an RFID reader is gone into more detail in the piece detection section. Finally, the user interface subsystem, composed of a speaker system, required approximately 5V of

power to be delivered to it. The power subsystem actually provided 5.44V in the simulated circuit.

The speaker system is simulated with an RLC circuit. This accommodates both the mechanical components of the speaker which include the suspension compliance, suspension resistance, moving mass, and a load of air that is pushed through the driver, as well as the electrical components of the speaker which include the coil inductance and coil resistance (Gupta). A close-up of the speaker system circuit is shown below in Figure 43.

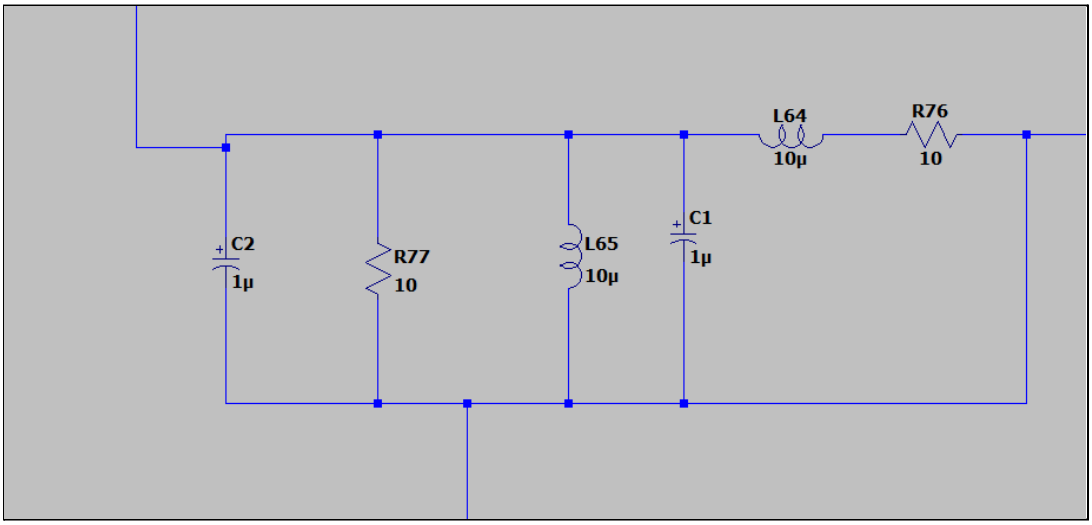


Figure 43: LTSpice Circuit of Speaker System

An image of the complete circuit, simulated in LTSpice, is shown with the appropriate input and output values to each subsystem in Figure 44. The electromagnets are cut off in the image to fit the detailed circuit, but there are 64 of them in parallel in the full circuit.

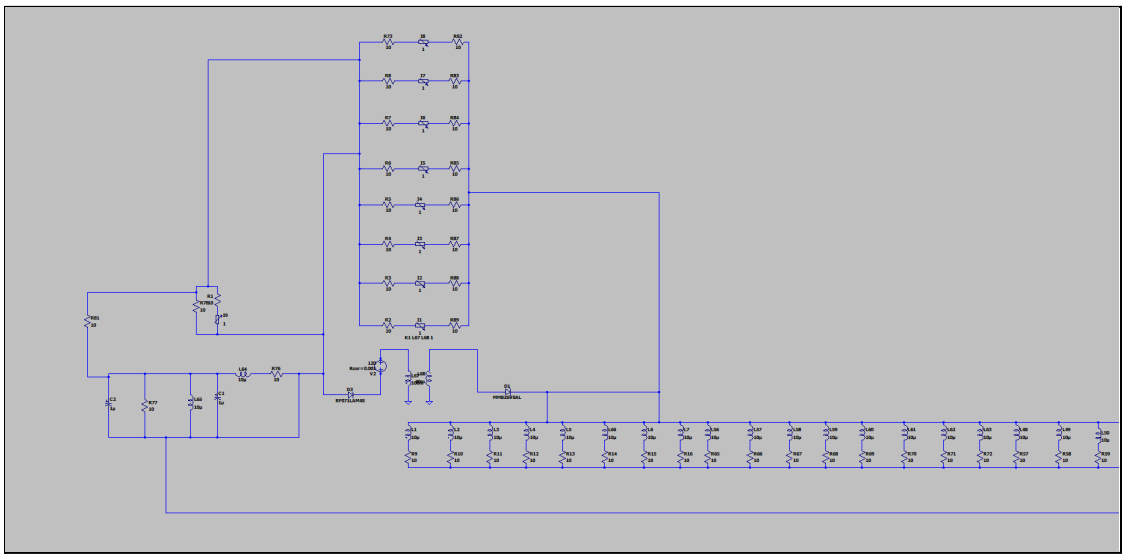


Figure 44: LTSpice Power Subsystem Circuit

7 Results and Discussion

7.1 Results

The final concept successfully accomplished our mission of creating a functional simulated chessboard for a blind user with increased functions compared with existing blind chessboards on the market. The board included 64 electromagnets (one underneath every square) to create a system of holding the pieces in place during play. This system included a push release system to allow the player to easily release a singular piece to make a move as they desired as well. This solved the customer requirement of creating a mechanism to keep pieces from toppling over or moving out of place when scanning the board with the sense of touch, while also allowing players to easily pick up a piece and move it during gameplay. In talking to members of the USBCA, they specified that while they had never played on a board with electromagnets, it could be a revolutionary new development to solve the common issue of knocking over pieces which is very disruptive to their gameplay.

The board also includes an easy-to-use, minimalistic user interface that is built into the board. This streamlines the timer and board as a single unit, simplifying gameplay for a blind user who can have difficulties finding a timer button that is separated from the board. The user interface also includes voice commands to instruct the blind player what move has been made by an opponent, how much time is remaining, which buttons are which, and in which mode the game is being played. The audio functionality is essential to communicate with the blind player. Each of the buttons also include raised dots to signify the buttons to a blind player and the number of buttons is kept to a minimum to eliminate confusion.

There are multiple modes of play built into the chessboard. A learning mode enables a learner to ask the built-in chess computer for the next best move available with a depth of twenty moves. In having the best available move generated and output via audio for the learner to hear, blind chess learners are given the opportunity to learn the game on a device specifically designed for them considering their additional needs. There is also a built-in computer opponent option that allows a blind player to competitively play against a built-in chess computer with timing. This allows a blind player to practice for tournament play. There are multiple levels available for the computer opponent as well including beginner, intermediate, and expert modes. Finally, the board can be used for play against a physical opponent. The user interface includes a button on either side of the board to allow each player to switch their turn at the end of the play. The built-in timing system manages the game for the players by keeping track of playtime for each player.

The blind player also receives in-game feedback to further alert them to the location of the opponent’s move via vibrating the square of the opponent's move during the game. This is accomplished via solenoids underneath each square of the board connected to the system of RFID chips on each piece and readers on each square that sense exactly which piece was moved and where on the board it was moved.

Additionally, the board includes in-game move recording which streamlines the process of keeping track of the game for blind players who have complained of the difficulty of recording moves via audio. The system of RFID chips and readers in the piece detection subsystem allows each move to be kept track of and stored in a USB output. Players no longer have to worry about keeping track of their moves because the list of moves generated by the piece detection subsystem can be printed after the game.

Finally, the board structure itself is built to accommodate blind players via texturally differentiated pieces, squares, and buttons. The board body is designed to be built from hard plastic to ensure durability and protection for the electrical components housed inside. The board is built to be plugged into a standard U.S. outlet, inputting 120V into the system. Power is then distributed via the implementation of a transformer to step down the voltage. This delivers approximately 5V of power to the electromagnets, the Arduinos for the game feedback and piece detection subsystems, and the speaker system of the user interface. The power cord also has the proper wire coating to prevent electrical accidents. A model of the final blind chessboard concept created in Rhino 5 and rendered in Unreal Engine is shown below in Figure 45.

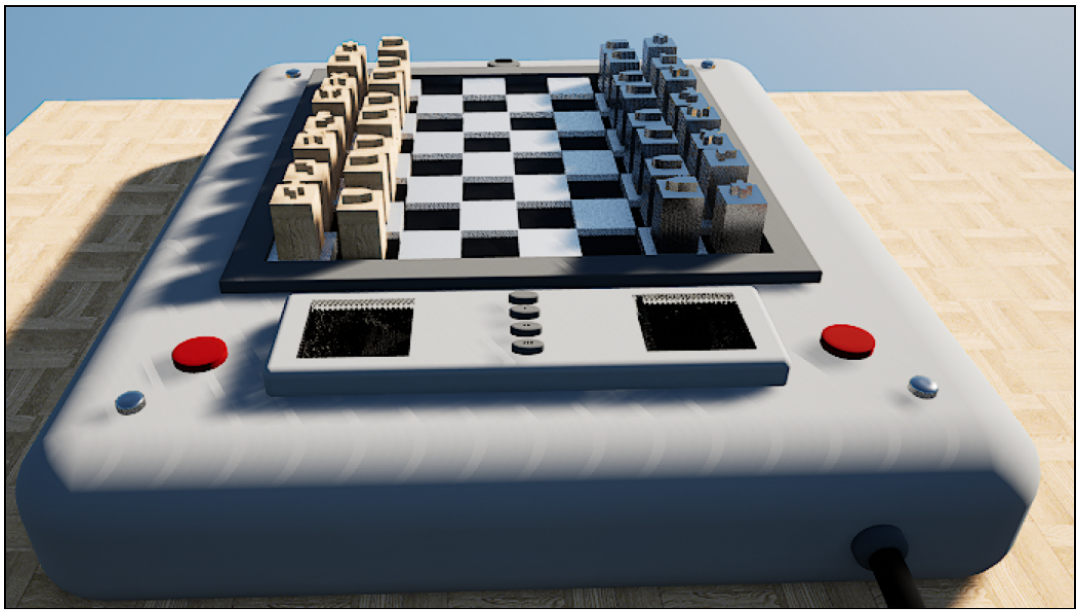


Figure 45: Final Rendering of Proposed Design in Unreal Engine

7.2 Significant Technical Accomplishments

For this project, the team was required to learn many simulation packages that pushed the boundaries of our technical skills. It was difficult to find a simulation package for the game user feedback subsystem. Initially, the subsystem was designed so that the user was alerted of the chess piece's location by raising the square it was attached to. Piece raising required the use of solenoids, which were very difficult to find in simulation packages for Arduino or Raspberry Pi. Brooke made a virtual machine on her computer in an attempt to use Raspberry Pi, but the free version downloaded did not have a simulation mode. After ten plus hours of researching more

simulation packages, it was decided that the piece raising idea had to be scrapped. The only simulation package that had solenoids cost $400. Brooke decided to vibrate the squares instead, as Tinkercad had vibration motors in its part library. The vibration motors were cheaper than the solenoids, cutting down the total project cost.

For the game functionality subsystem, a large quantity of code was downloaded from the Stockfish website, with very little explanation of how it worked. Modifying this code and having the final version compile with the already existing Stockfish code, and accept the correct inputs with correct outputs, was a significant technical accomplishment.

8 Conclusion

The market for blind chess boards is not large, and this is possibly because there are few unique and well-thought-out designs. This is both a result and a cause of chess not being extremely popular among the blind and visually impared. This is unfortunate because chess is a game extremely well suited to the blind. Once players reach an advanced enough level, they can even play a full game in their heads by simply speaking their moves. However, for blind beginners, a lack of chess boards that provide a good learning experience can be a large hindrance. Most current designs are regular chess boards with small changes such as pegs and holes for the pieces. However, these designs fail to meet many of the needs of blind chess players.

Through careful customer research and engineering design, this project simulates a chessboard that meets those specific needs and offers unique features for students learning chess and playing competitively. The contents of this memo provide a strong foundation for the chessboard to actually be constructed. Due to the quality of the simulations, the only challenges to construction would be physical, such as housing and part manufacturing. The next steps in this project would be combining CAD models between subsystems, testing the code on a raspberry pi, and searching for usable parts to build a prototype.

At its core, this project aimed to solve a simple problem - to make a chessboard that excelled at physical and audio interaction. Technically, this involved a lot of user interface work which presented unique challenges involving lots of electronics and coding. The team members went above and beyond to learn new software and programming languages to address these challenges. Additionally, a large portion of the project emphasized the importance of communication between subsystems. This involved determining what inputs and outputs each subsystem required and how they were related to other subsystems' inputs and outputs. For instance, to be compatible with the raspberry pi and the game feedback subsystem, the game functionality subsystem must output a 6 bit binary signal.

Overall, the mission statement was fulfilled. Each of the subsystems performed to the required specifications. More importantly, the designed device succeeds at improving chess for the visually impaired. It makes the game easier to learn and play for the blind. Chess is a game that has been around for thousands of years and it should be as available to a blind person as it is for anyone else.

9 References

Balata, Jan & Mikovec, Zdenek. (2015). *Problems of blind chess players*. 179-183. https://doi.org/10.1109/CogInfoCom.2015.7390587.

Banas, T. (2018, March 13). *How to Calculate the Force of an Electromagnet*. Sciencing. Retrieved April 6, 2021, from https://sciencing.com/calculate-force-electromagnet-5969962.html

Chess Baron. (n.d.). *10 inch Chess Set for the Blind – Magnetic*. Chess Baron. https://www.chessbaron.com/product/E2043/

Costalba, M., & Kiiski, J. (2008, November 2). Stockfish 13. Retrieved March 17, 2021, from https://stockfishchess.org/

Ferreira, Diogo R. (2013, June). *The Impact of Search Depth on Chess Playing Strength.* ICGA journal. Retrieved May 1, 2021, from http://web.ist.utl.pt/diogo.ferreira/papers/ferreira13impact.pdf

Gupta, S. (2018, November 16). *Simulate Speaker with Equivalent RLC Circuit*. Circuit Digest. https://circuitdigest.com/electronic-circuits/simulate-speaker-with-equivalent-rlc-circuit.

H. (2019, March). *Swallowed object*. Retrieved April 12, 2021, from https://www.health.harvard.edu/a_to_z/swallowed-object-a-to-z#:~:text=Complications%20can%20include%20tears%20in,together%20and%20erode%20through%20tissue.

Hess, R. (1998). *U.S. Patent No. 5,848,788*. Washington, DC: U.S. Patent and Trademark Office.

McKean, Duncan. (2021, February 22). *Chess Set for the Blind and Visually Impaired*. Duncan

McKean Portfolio. https://duncanmckean.com/blog/portfolio-item/chess-set-for-the-blind-and-visually-impaired/

Rec.games.chess (1994, March 12). *Portable Game Notation Specification and Implementation Guide.* Retrieved March 17, 20221, from https://ia802908.us.archive.org/26/items/pgn-standard-1994-03-12/PGN_standard_1994-03-12.txt

Reoyamanaka. (n.d.). Reoyamanaka/chesstimer. Retrieved May 02, 2021, from https://github.com/reoyamanaka/chesstimer/blob/master/chess_timer.py

Silva, A. (1996). *U.S. Patent No. 5,957,455*. Washington, DC: U.S. Patent and Trademark Office.

Stanborough, R. J. (2020, October 19). *The 10 Best Benefits of Playing Chess.* Retrieved April 12, 2021, from https://www.healthline.com/health/benefits-of-playing-chess#deepens-focus

The Blind Superstore. (n.d.). *Chess Set, Wooden (Tactile)*. The Blind Superstore. http://www.braillebookstore.com/Chess-Set,-Wooden.1

Tons of Electronic Waste Thrown Out. The World Counts. (n.d.). https://www.theworldcounts.com/challenges/planet-earth/waste/electronic-waste-facts/story.

World Health Organization. (2017, December 8). *Global data on visual impairment*. World Health Organization. https://www.who.int/blindness/publications/globaldata/en/#:~:text=Globally%20the%20number%20of%20people,are%2082%25%20of%20all%20blind.

Wx.button¶. (n.d.). Retrieved May 02, 2021, from \ https://wxpython.org/Phoenix/docs/html/wx.Button.htm

10 Appendix A: Selection of Team Project

First, the team generated a list of ideas within the categories of biometrics and sustainability and alternative energy. Robotics was ruled out early on because the group agreed that these two project categories interested all members of the group more. Figure 46 shows this initial list of ideas that the team came up with. This list of team project ideas included any idea a team member had for the project. It allowed the team to think creatively and narrow down the options later. The team was able to rule out the ideas of alternative energy using this list, since the ideas that were brought up for the biometrics category were more appealing to the majority of members. The top five project ideas from the list were a smart face shield, an anti-vape device, a body temperature measuring clothing, a chess board for the blind, and a self disinfecting pen. These are highlighted in yellow.

Biometrics

- Talking calculator for blind or older populations
- Smart face shield - tracks something
- Exercise equipment for disabled people
  - useful for studies disease transmission, or useful in public places
- Vape device designed to help someone stop through slowly decreasing nicotine concentration
  - comes with pods of different concentrations
  - vape recommends daily pod to use, decreases over time
- Dehydration sensor
- Drawing/painting device for the blind
  - Various textured materials to signify colors
- Helping people with ALS?
- Running shoes that track usage w/ easily replaceable soles
  - Cuts cost of having to buy new shoes as often
  - Create recommendations for sole replacement for individual based on impact, senses wear and tear
  - Sustainable, bamboo memory foam?
- Shoe repair kit
- Sneeze detector
- Electric motorcycle
- Something that helps elderly feed pets
- COVID smart watch
  - Social distance alert, vaccine passport, contact tracing
  - Also used for other colds such as the flu
- Something to help elderly during covid
  - Entertainment, groceries
  - Chess or some other game- remotely play with other people
- Workout gear for cold weather
  - Can cool you off if you're too hot
  - Adjusts to body temp throughout the run
  - Runners and the elderly
- Chess for the blind
  - Read off commands for pieces and chess piece moves to that spot
  - Says out loud where opponent moves
- Self-disinfecting pen
  - Uv or some kind of clorox spray

Sustainability and Alternative Energy

- Solar powered camping gear (heated tent, cooking device, water filtration, etc)
  - To help the homeless?
- Umbrella-like space heater for public spaces - homeless
- Sink tap water tracker - powered by water going through it
  - Mechanical part - small turbine powering it, as well as small display with nice graphic saying how much water has been used
- Car breathing mask window
- Compost bin that generates heat for household
  - Heats water
- Solar kettle
- Tileable phone case
- Sea water -> drinking water tank
  - cool sea water makes condensation form, collected for drinking water

Figure 46: Team Project Ideas

Next, a concept selection matrix of the five remaining ideas from Figure 46 was created as seen in Figure 47. The team ranked the five best ideas and compared them based on specific conditions and properties. The team discussed which constraints were considered to be the most important for the project's success and used the criterion to rank the project ideas. The criteria used to evaluate the project ideas were: the ability to incorporate enough subsystems, the ability to incorporate mechanical engineering components, presence of a defined customer, complexity, presence of defined needs, and cost effectiveness.

PROJECT SELECTION MATRIX					
	Blind Chess	Disinfecting Pen	Smart Face Shield	Anti-Vape Vape	Heated Workout Clothes
Enough Subsystems	5	2	4	4	5
Mechy Enough	5	2	4	3	4
Defined Customer	4	5	3	5	5
Complex Enough	5	2	4	4	4
Well Defined Need	4	5	4	2	3
Low Enough Cost	2	3	2	3	2
Total	25	19	21	21	23

Figure 47: Project Selection Matrix

Determined through the selection matrix, the champion design was the chessboard for the blind. The disinfecting pen was originally a group favorite, but after discussion, it was ruled out. Through benchmarking, the team realized this idea was simpler than it appeared. All that was needed was a silver-plated pen which did not allow for enough complexity or subsystems. A smart face shield and anti-vape both ranked the same in second to last. The smart face shield was complex enough by tracking temperature, time of day, and daily applications, but had a high expense compared to a normal mask. The anti-vape vape was introduced and looked down upon by members because vapes were created to stop people from smoking, but have been abused and become problematic in themselves. The team discussed this possibility further and concluded the anti-vape product would most likely have high negative impact when considering health and safety factors. The runner-up was a heated clothing device that regulated heat based on the user's body temperature. The target consumer was an older aged individual or athlete that needed to monitor and maintain a certain body temperature during a workout.

The team came to a final consensus on a chessboard for the blind based on the selection matrix ranking. The goal was to create a chessboard that allowed blind users to enjoy chess the same way a fully sighted player does. This project allowed for enough subsystems for the whole group and was a complex project that interested all group members.

11 Appendix B: Customer Requirements and Technical Specifications

Table 7: Complete Customer Requirements Table

Customer Need	Requirement/ Technical Specification	Target Value / Range of Values
Pieces Can't Be Knocked Over	Strong Magnets	Magnet Strength >= Weight of Piece
Distinguishable Pieces	Differentiable Piece Material for Black/White	Stainless steel & wood coated stainless steel
To compete	Up to Tournament standard	Each square for tournament play should be between 5 cm to 6 cm that is equivalent to 1.97 inches to 2.36 inches
Different computer difficulties	Easy, medium, hard, expert modes	1000, 2000, 3200 elo stockfish
*No touch clock**	Clock automatically stops when move is made	When reader detects valid move, clock stops, starts for other player
*Take quick notes of moves played**	Moves are recorded and extractable via usb or sd card	Board records PGN for match, saves on USB or SD card
*Integrated clock**	Clock timer is attached to the chessboard	Reads out each players time every 30s
*Pieces of same shape**	Square pieces (two different materials used for black and white) & each piece indicates the direction of play for that piece	3.5cm x 3.5cm x 3.5cm blocks
Relatively small board	Around 10in x 10in	Within 2in on either side of 10in square board
Pieces stay centered on square	Magnets in center of piece and square	Electromagnet centered on each square
Touch read out of squares	Speaker read out of square position with force applied to the square	Instantaneous readout w/ force applied to square

Raised signifiers on pieces	Rectangular prism shaped pieces with raised symbol on top face	Symbols raised by 2mm on top face of piece
Push release system for piece holding system	Electromagnet system that releases with application of force on square	Electromagnet releases with x force applied to square
Mechanism for finding lost pieces	Sound that plays when piece is separated from board	When piece is more than 3 ft from board, beeping sound plays

Table 7 is the customer requirements table that includes all of the needs that were collected from interviewing prospective customers, including a representative from the Lighthouse for the Blind and members of the United States Blind Chess Association, and online research into our user demographic. The needs were then translated into requirements and specifications.

12 Appendix C: Gantt Chart

				Week 1							Week 2							Week 3							Week 4						
		Project Facilitator:		28 Feb 2021							7 Mar 2021							14 Mar 2021							21 Mar 2021						
Group 1		Heather Converse		22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
WBS	Task	Lead	% Done	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su
1	**Customer Needs/ Project Specs**	All																												
1.1	Specify Customer Base	Nico	100%																											
1.2	Generate General Needs	Heather	100%																											
1.3	Research Needs online	Brooke	100%																											
1.4	Reach out to potential customers	Eileen	100%																											
2	**Requirements/Specs**																													
2.1	Translate Needs to Requirements	Carter	100%																											
2.2	Translate Requirements to Specifications	Julian	100%																											
3	**Concept Generation**																													
3.1	Benchmarking	Heather/Brooke	100%																											
3.2	Brainstorming	Carter/Eileen	100%																											
3.3	Concept Grading	Nico	100%																											
3.4	Concept Selection	Brooke/Julian	100%																											
3.5	Decide on best design	Heather/Nico	100%																											
3.6	Modify design based on restraints	Carter	100%																											
4	**Milestone 1**																													
4.1	Split up presentation tasks	Heather	100%																											
4.2	Create PowerPoint slides	Eileen/Brooke	100%																											
4.3	Run through PowerPoint presentation	Nico/Carter	100%																											
4.4	Split up memo responsibilities	Heather	100%																											
4.5	Write Memo	Julian/Nico/Brooke	100%																											
5	**SUBSYSTEM - Piece Detection**	Carter																												
5.1	Component Selection		100%																											
5.2	Code Move reader		100%																											
5.3	Wiring		100%																											
5.4	Code Move recorded		100%																											
5.5	Integrating/Troubleshooting		100%																											
6	**SUBSYSTEM - User Interface**	Nico																												
6.1	timer		100%																											
6.2	buttons		100%																											
6.3	board structure	Eileen	100%																											
6.4	Speaker		100%																											
7	**SUBSYSTEM - Game Functionality**	Julian																												
7.1	Valid move checking		100%																											
7.2	Game recording		100%																											
7.3	Computer move - stockfish		100%																											
7.4	Training Mode, different computer levels		100%																											
8	**SUBSYSTEM - Piece Movement**	Heather																												
8.1	Master magnet power button		100%																											
8.2	voice command for magnet release		100%																											
8.3	detects when piece is picked up		100%																											
9	**SUBSYSTEM - Power**	Eileen																												
9.1	ciruitry/ wiring		100%																											
9.2	modeling of circuitry		100%																											
9.3	power capacity		100%																											
9.4	integration with other systems		100%																											
9.5	photo realistic cad model		100%																											
10	**SUBSYSTEM - Game User Feedback**	Brooke																												
10.1	Tinkercad circuit for piece vibration		100%																											
10.2	Code for Tinkercad circuit		100%																											
10.3	Model components under board in NX		100%																											
10.4	Test Simulation		100%																											
11	**Milestone 2 - Subsystem Demos**																													
11.1	Make Recordings of subsystems	Julien/Nico	100%																											
11.2	Create PowerPoint slides	Eileen/Carter	100%																											
11.3	Practice Presention	Heather/Brooke	100%																											
12	**Milestone 3**																													
12.1	Split up memo responsibilities	Heather	100%																											
12.2	Write Memo	Nico	100%																											
12.3	Create PowerPoint slides	Julien	100%																											
12.4	Practice Presention	Carter	100%																											
12.5	Edit Memo	Brooke/Eileen	100%																											

				Week 5							Week 6							Week 7						
		Project Facilitator:		28 Mar 2021							4 Apr 2021							11 Apr 2021						
Group 1		Heather Converse		22	23	24	25	26	27	28	29	30	31	1	2	3	4	5	6	7	8	9	10	11
WBS	Task	Lead	% Done	M		W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su
1	**Customer Needs/ Project Specs**	All																						
1.1	Specify Customer Base	Nico	100%																					
1.2	Generate General Needs	Heather	100%																					
1.3	Research Needs online	Brooke	100%																					
1.4	Reach out to potential customers	Eileen	100%																					
2	**Requirements/Specs**																							
2.1	Translate Needs to Requirements	Carter	100%																					
2.2	Translate Requirements to Specifications	Julian	100%																					
3	**Concept Generation**																							
3.1	Benchmarking	Heather/Brooke	100%																					
3.2	Brainstorming	Carter/Eileen	100%																					
3.3	Concept Grading	Nico	100%																					
3.4	Concept Selection	Brooke/Julian	100%																					
3.5	Decide on best design	Heather/Nico	100%																					
3.6	Modify design based on restraints	Carter	100%																					
4	**Milestone 1**																							
4.1	Split up presentation tasks	Heather	100%																					
4.2	Create PowerPoint slides	Eileen/Brooke	100%																					
4.3	Run through PowerPoint presentation	Nico/Carter	100%																					
4.4	Split up memo responsibilities	Heather	100%																					
4.5	Write Memo	Julian/Nico/Brooke	100%																					
5	**SUBSYSTEM - Piece Detection**	Carter																						
5.1	Component Selection		100%																					
5.2	Code Move reader		100%																					
5.3	Wiring		100%																					
5.4	Code Move recorded		100%																					
5.5	Integrating/Troubleshooting		100%																					
6	**SUBSYSTEM - User Interface**	Nico																						
6.1	timer		100%																					
6.2	buttons		100%																					
6.3	board structure	Eileen	100%																					
6.4	Speaker		100%																					
7	**SUBSYSTEM - Game Functionality**	Julian																						
7.1	Valid move checking		100%																					
7.2	Game recording		100%																					
7.3	Computer move - stockfish		100%																					
7.4	Training Mode, different computer levels		100%																					
8	**SUBSYSTEM - Piece Movement**	Heather																						
8.1	Master magnet power button		100%																					
8.2	voice command for magnet release		100%																					
8.3	detects when piece is picked up		100%																					
9	**SUBSYSTEM - Power**	Eileen																						
9.1	ciruitry/ wiring		100%																					
9.2	modeling of circuitry		100%																					
9.3	power capacity		100%																					
9.4	integration with other systems		100%																					
9.5	photo realistic cad model		100%																					
10	**SUBSYSTEM - Game User Feedback**	Brooke																						
10.1	Tinkercad circuit for piece vibration		100%																					
10.2	Code for Tinkercad circuit		100%																					
10.3	Model components under board in NX		100%																					
10.4	Test Simulation		100%																					
11	**Milestone 2 - Subsystem Demos**																							
11.1	Make Recordings of subsystems	Julien/Nico	100%																					
11.2	Create PowerPoint slides	Eileen/Carter	100%																					
11.3	Practice Presention	Heather/Brooke	100%																					
12	**Milestone 3**																							
12.1	Split up memo responsibilities	Heather	100%																					
12.2	Write Memo	Nico	100%																					
12.3	Create PowerPoint slides	Julien	100%																					
12.4	Practice Presention	Carter	100%																					
12.5	Edit Memo	Brooke/Eileen	100%																					

		Project Facilitator:		18 Apr 2021							25 Apr 2021							2 May 2021							
Group 1		Heather Converse		12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3
WBS	**Task**	**Lead**	**% Done**	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M	T	W	Th	F	Sa	Su	M
1	**Customer Needs/ Project Specs**	All																							
1.1	Specify Customer Base	Nico	100%																						
1.2	Generate General Needs	Heather	100%																						
1.3	Research Needs online	Brooke	100%																						
1.4	Reach out to potential customers	Eileen	100%																						
2	**Requirements/Specs**																								
2.1	Translate Needs to Requirements	Carter	100%																						
2.2	Translate Requirements to Specifications	Julian	100%																						
3	**Concept Generation**																								
3.1	Benchmarking	Heather/Brooke	100%																						
3.2	Brainstorming	Carter/Eileen	100%																						
3.3	Concept Grading	Nico	100%																						
3.4	Concept Selection	Brooke/Julian	100%																						
3.5	Decide on best design	Heather/Nico	100%																						
3.6	Modify design based on restraints	Carter	100%																						
4	**Milestone 1**																								
4.1	Split up presentation tasks	Heather	100%																						
4.2	Create PowerPoint slides	Eileen/Brooke	100%																						
4.3	Run through PowerPoint presentation	Nico/Carter	100%																						
4.4	Split up memo responsibilities	Heather	100%																						
4.5	Write Memo	Julian/Nico/Brooke	100%																						
5	**SUBSYSTEM - Piece Detection**	Carter																							
5.1	Component Selection		100%																						
5.2	Code Move reader		100%																						
5.3	Wiring		100%																						
5.4	Code Move recorded		100%																						
5.5	Integrating/Troubleshooting		100%																						
6	**SUBSYSTEM - User Interface**	Nico																							
6.1	timer		100%																						
6.2	buttons		100%																						
6.3	board structure	Eileen	100%																						
6.4	Speaker		100%																						
7	**SUBSYSTEM - Game Functionality**	Julian																							
7.1	Valid move checking		100%																						
7.2	Game recording		100%																						
7.3	Computer move - stockfish		100%																						
7.4	Training Mode, different computer levels		100%																						
8	**SUBSYSTEM - Piece Movement**	Heather																							
8.1	Master magnet power button		100%																						
8.2	voice command for magnet release		100%																						
8.3	detects when piece is picked up		100%																						
9	**SUBSYSTEM - Power**	Eileen																							
9.1	ciruitry/ wiring		100%																						
9.2	modeling of circuitry		100%																						
9.3	power capacity		100%																						
9.4	integration with other systems		100%																						
9.5	photo realistic cad model		100%																						
10	**SUBSYSTEM - Game User Feedback**	Brooke																							
10.1	Tinkercad circuit for piece vibration		100%																						
10.2	Code for Tinkercad circuit		100%																						
10.3	Model components under board in NX		100%																						
10.4	Test Simulation		100%																						
11	**Milestone 2 - Subsystem Demos**																								
11.1	Make Recordings of subsystems	Julien/Nico	100%																						
11.2	Create PowerPoint slides	Eileen/Carter	100%																						
11.3	Practice Presention	Heather/Brooke	100%																						
12	**Milestone 3**																								
12.1	Split up memo responsibilities	Heather	100%																						
12.2	Write Memo	Nico	100%																						
12.3	Create PowerPoint slides	Julien	100%																						
12.4	Practice Presention	Carter	100%																						
12.5	Edit Memo	Brooke/Eileen	100%																						

The Gantt chart was heavily relied upon during the planning stages of the project. It was consulted when dividing up tasks for milestones 1, 2, and 3. It was helpful to visualize how much work each member had to complete before each of the milestone deadlines (shown in red). However, the chart was not consulted to see the progress of every member. Group members gave an update on their progress at the beginning of each biweekly meeting instead. The deadlines for each individual task were not strictly followed. It was hard to commit to dates the team had set during the planning phase when new homework assignments, papers, and projects were assigned in other classes throughout the semester. At the end of the project, it was interesting to look back at all the group had accomplished in the 10 week period.

13 Appendix D: Expense Report

Table 8 : Expense Report

Component	Type	Fixed/Variable	Qty	Cost Each	Total Cost
Raspberry Pi	Standard	Fixed	1	75	75
Wires	Standard	Variable	1	18.96	18.96
Buttons	Custom	Variable	6	0.41	2.46
Electric Plug	Standard	Fixed	1	3.99	3.99
Speaker	Standard	Fixed	2	1.95	3.9
Electro-magnets	Standard	Fixed	64	0.636	40.704
Vibration Motor	Standard	Fixed	64	1.2	76.8
RFID Tag	Standard	Variable	64	3.95	252.8
Steel	Custom	Variable	16	3.8	60.8
Wood	Custom	Variable	16	1.5	24
RFID Reader	Standard	Fixed	64	25.95	1660.8
Arduino Uno	Standard	Fixed	9	23	207
Misc Hardware	Standard	FIxed	1	10	10
Components Total					2437.214

By not building a physical prototype we rarely thought about expenses during this project. This can be seen in Table 8, showing the team's expense report. The report estimates the prototype to cost approximately 2437 dollars. With some redesigns, the total estimate could be lower. Specifically, a redesign of the user feedback subsystem so that it uses one Arduino instead of eight. By far the most expensive subsystem is the piece detection which is responsible for almost eighty percent of expenses. If a cheaper RFID reader was found to be sufficient, it could lower the prototype's estimate or funds could be reallocated into purchasing higher quality components for other subsystems.

14 Appendix E: Team Members and Their Contributions

14.1 Team Member 1 - Nico Nigohosian

As a hyperactive member of this team it was easy for me to kickstart the conversation or brainstorming of ideas for any conversation. I was a team player that cared about the opinions of others and loved to collaboratively build upon ideas for the betterment of the whole group. I took on the task of the subsystem assignment User interface, my very little computational knowledge of highschool python proved to be an advantage when learning python code for the testing of my subsystem and through this project it furthered my understanding of computational language while creating something that enjoyed and I looked forward to see complete. My subsystem had me experimenting with various programs trying to find one to mimic Arduino audio output but came to no avail as I settled on coding a wX.app which is python GUI to model the instrumentation of the user interface. This required extensive research of code and education of the topic and I successfully completed my task. On the memo I dealt with the Subsystem Analysis and Design, Selection of Team Project, and Subsystem 1 User Interface.

14.2 Team Member 2 - Julian Matthews

For this project, I took on difficult technical challenges and created quality visuals for both my code and project diagrams. I was given a large amount of the coding work including integrating Stockfish and receiving and sending several inputs and outputs to other subsystems. It took a considerable amount of time to read through the Stockfish code and understand how to compile, run, and modify it to work with our project.

I attended every meeting and was a collaborative team member in all presentations and memos. For this memo, I wrote the introduction, mission statement, technical specifications, and conclusion, as well as the section for my subsystem. I created important diagrams such as the subsystem breakdown diagram (Figure 17) and the block diagram (Figure 18) with help from the rest of the team.

14.3 Team Member 3 - Brooke Zatowski

I was a team player for the duration of this project. I attended every meeting and completed all of my tasks before deadlines. I completed all aspects of my subsystem and volunteered to make a CAD model of the components underneath each square of the chessboard.

I put in extensive research to design my subsystem. Originally, I planned to alert the user of the chess piece's position on the board by raising the chess square using solenoids. I spent ten plus hours trying to find a simulation package that contained solenoids. I was not successful, but I did learn how to run a virtual machine on my computer. I decided to vibrate the squares instead because Tinkercad had vibration motors in their simulation library. I learned how to wire an Arduino circuit to a breadboard and how to code in C++.

For the final memo, I wrote the assessment of relevant existing technologies section, the game user feedback section, the Gantt chart section, and first try paragraph in appendix G, and the first paragraph in the significant technical accomplishments section.

14.4 Team Member 4 - Carter Wynn

I was an efficient, hard-working team player. I consistently produce high-quality work and meet every deadline. Before I started to build my subsystem I had to do extensive research on RFID tags and readers. My subsystem was complex and required three separate simulation packages to be tested. I had to learn C++ to code the binary code for my Arduino. I created a complex LTspice logic gate binary decoder and a full circuit of sixty-four RFID readers. The RFID readers were a custom integrated circuit that I built. I also had to teach myself how to create custom integrated circuits for this project. Despite the complexity and amount of work required to demonstrate my subsystem, I was able to complete my test on time and delivered a quality product.

For the memo, I wrote the professional and society considerations section, my subsystem, and the expenses report.

14.5 Team Member 5 - Heather Converse

For this project, I was chosen as the team leader. This involved scheduling meetings, creating meeting agendas, checking in on team members' progress, and making sure deadlines were met. I made sure our team followed the Gantt Chart we created to stay on track during the different phases of the project. As well, I assigned equal memo sections for each team member and communicated team information to both professors. I also sent out weekly reminders for upcoming meetings and assignments.

My subsystem involved the circuitry and function of the electromagnets that keep the chess pieces in place. I tested the output voltage, which turned the magnets on and off, based on the force present on a square. I used code to determine what square a user was applying force to so that the output would turn the corresponding magnet off. Before this class, I had no coding experience, so I was challenged with researching and learning how to code in both C++ and Python. I also calculated the force of the electromagnets that were used for each square.

For the final memo, I wrote the system concept development and selection, the introduction for project objectives & scope, my subsystem, and the user manual. I also organized, proofread, and edited the entire memo before submitting the final version.

14.6 Team Member 6 - Eileen Beres

In the project I was an effective team member who consistently stayed on top of deadlines and attended all team meetings. I wrote my assigned memo sections as expected and completed my subsystem work. I also designed the outside structure of our board to provide a visual representation of our product using Rhino 5 CAD software and Unreal Engine for texture, color, and lighting touches.

My subsystem involved creating the overall circuitry for the synthesis of the four subsystems requiring electrical power as an input. My task was to take an input voltage from a standard outlet and create a circuit that delivered specified voltages to the piece movement, game feedback, piece detection, and user interface subsystems. I learned how to use LTSpice to simulate this.

For the final memo, I wrote the executive summary, the customer requirements section, my own subsystem, the final concept and results sections, and one of the lessons learned paragraphs. I also helped organize, proofread, and edit the entire memo before submitting.

15 Appendix F: Statement of Work

Team: Heather Converse, Julian Matthews, Carter Wynn, Eileen Beres, Nicolas Nigohosian, Brooke Zatowski

Semester Objectives:

1. Research alternative chess boards designed for the blind and improve upon available designs in order to create a better experience for the user.
2. Generate needs specific to the customer and the challenges they face due to being blind and incorporate them in our design.
3. Design and simulate a working chessboard that allows blind people to play chess with either a non-blind opponent or a computer simulation.
4. Design and deliver a simulated model using CAD and programming that demonstrates the form and function of our design.
5. Develop a set of team rules critical for the success of the project and develop a leadership structure.

Approach:

Rapidly design and model all necessary subsystems required, with enough simplicity to ensure that a physical version would work as intended. Simulate the model to ensure that it is error-free in concept, and addresses the needs of blind chess players.

Deliverables and Dates

1. Milestone 1 - System Concept Reviews, presentation, and written proposal (3/11)

2. Informal subsystem team demos (4/11)

3. Milestone 2 - Prototype Demonstration (4/26)

4. Milestone 3 - Design review presentation and technical report (5/3)

16 Appendix G: Professional Development - Lessons Learned

Keep - Shared Google Drive

At the start of the project, the team created a shared Google Drive in which all of the documents for the entire semester were stored. Folders were created for each milestone deadline and for team-specific information. This helped keep all of the documents organized and made it easy to find past information when writing the final memo. As well, it made it simple to simultaneously create and edit large reports, such as the Milestone 1 and final technical memos. Team members were able to add information at their convenience and see what other members contributed. This accelerated the editing and reviewing process.

Keep - Positives and Negatives of the Week

With a team composition predominantly of introverts, it was hard to initially breed discourse. It was recommended that the team try a team-building activity at the beginning of each meeting where each member shares a positive and negative from the week. This allowed team members to get to know each other better and made discussing the project easier as everyone had already started speaking. This increased the team's effectiveness and comradery.

Problem - Choosing the Project

Picking a project was a struggle for the group because we were a collaborative group type. The team wanted everyone to be involved in the process of picking a project so that when it came to actually executing the project in the long run, everyone was enthused about the overall concept. A democratic approach was chosen when developing project ideas. This was helpful in the long run because every team member voiced an opinion in the project choice. However, it made the process of picking a project lengthy and frustrating as a result. In the future, the team would be more decisive in picking a project earlier to avoid spending an excessive amount of time in the stage of choosing a project.

Try - Incorporating All Individuals for Discussions

In the earlier meetings, not everyone was comfortable sharing their ideas with the group. As a result, not everyone was involved in the project brainstorming session. It may be helpful to ask everyone individually what they think of each proposed idea in the future. This would be especially helpful for introverts in the group, for they tend to think through their ideas first before sharing. This sometimes results in the group moving on before they can express their opinion. This would also help to organize the discussion since there would be fewer interruptions if everyone had a turn to speak.

Try - Mini Subsystem Demos to the Team

Something worth trying in the future is short technical team demos to update each other on the progress of subsystems. Similar to in class subsystem demos, these could be shortened

and presented to the group weekly or bi-weekly. There was a slight need for more accountability of each member's technical work, and this would provide that. Although the team made sure to give updates on their subsystems, small demos would have given each member more incentive to get ahead on their work.

17 Appendix H: Software / Technology Used

17.1 *Collaboration Among Team Members*

- Cisco WebEx Teams and Meetings
- LMS Discussion Boards
- Shared Google Drive
- iMessage

17.2 *Subsystem Design*

- Siemens NX
- Tinkercad
- LTspice – for schematics and simulation
- Rhino 5
- Unreal Engine
- Ubuntu (C++)

17.3 *Programming*

- Spyder (Python)
- Arduino (C++)
- Ubuntu (C++)

17.4 *Subsystem Testing/Simulation/Emulation*

- Siemens NX
- Spyder (Python)
- LTspice – for analog circuits
- TinkerCAD
- Ubuntu (C++)

18 Appendix I: User Manual

Congratulations on your purchase! Before using this device, please read and pay careful attention to the safety notes and operating instructions below. Store this user manual with the device so you can refer to it later if needed.

Table of Contents

General Safety Notes

This unit is not intended for children under 5 years old. If you let your children use the device, instruct them accordingly and ensure the device is only used as intended.

WARNING! This device contains magnets and electrical wires. In the event that magnets are swallowed, contact your doctor immediately.

- Keep packaging away from babies and small children
- Do not expose the device to heat
- Do not open the device under any circumstances. In the event of a malfunction, contact the service address listed.

Electrical Safety

- At all times avoid putting the outlet cord near water. Never try to plug the cord into a wall outlet with wet hands.
- Never plug in frayed or damaged cords. Contact the provider if the cord is damaged.
- Never try to repair energized equipment.
- When not in use the device should first be turned off (using the on/off switch on the board), and then unplugged from the wall outlet.
- If a person comes in contact with a live electrical conductor, do not touch the equipment, cord, or person. Pull the plug out using a leather belt or another non-conductive material.

- Do not store highly flammable liquids near the electrical equipment
- If you smell anything burning, immediately turn off and unplug the device

Cleaning

- If needed, clean the device's surface with a slightly damp cloth and allow it to air dry. Make sure no moisture can penetrate into the surface.
- Do not use solvents or other cleaning agents besides water, this can damage the device's surface

Parts Included:

The package contains the following components

- 1 chessboard (chess computer encased inside the board)
- 16 white chess pieces (made of steel)
- 16 black chess pieces (made of steel with wooden coating)
- 1 user manual

After receiving and unpacking your product, please check to make sure all of the parts are included and check for possible signs of damage. In the event that parts are missing or are damaged, contact the dealer where you obtained the product.

Setup:

1. Remove the chessboard and chess pieces from packaging.
2. Plug the cord into a wall outlet and flip the button on the chessboard to "on".
3. Set up the black and white chess pieces at their respective starting positions.
4. Choose between player and CPU mode.

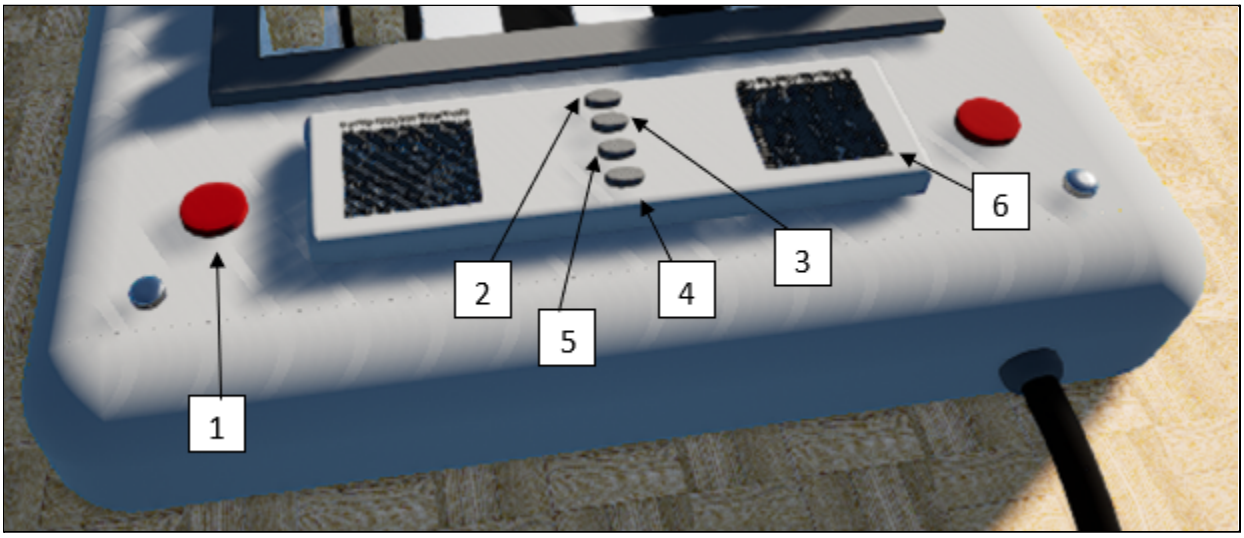
Refer to the "Operation" section below for directions on how to play.

Operation:

Using the chess pieces

The chess pieces can be positioned on the device's game board at the starting position. The built-in electromagnets make sure the pieces stick to the game board and are protected from toppling over. To make a move, press down on the chess piece you want to move and then immediately pick the piece up. Place the piece on the square you want to move it to.

Buttons and their functions



The KeyDescriber outputs audio that describes the functions of each button listed below. Each button has braille on it so they are easily distinguishable.

1. *Switch* - The red button on each side of the board. Press this button when you are done making your move. This switches the timer to the opponent's side.
2. *Start time* - Press the button to start the clock timer. The speaker will announce when one minute is remaining on the timer.
3. *Pause* - This pauses the clock timer.
4. *Add time* - This adds 5 min to the clock timer.
5. *CPU/Player mode* - Choose whether you want to play against a computer or in 2 player mode.
6. *Speaker* - Outputs audio feedback to help guide players through the game of chess.

Computer Mode

When a player selects computer mode, the chess engine will read out the best move for the current position and keep track of each move received. After the computer reads out the move, the player will physically move the chess pieces according to the computer's instruction. The starting square where the piece is being moved from, as well as the square that the piece is being moved to, will each vibrate for 10 seconds. This makes it easier for the player to tell which square the computer's chess piece needs to be moved to. The computer also checks for move errors so no illegal moves can be made. For tournament compatibility, a record of the moves made for the entire game will be sent to a raspberry pi via a usb which can then be printed off.

Troubleshooting

Problem	**Possible Solution**
Computer doesn't react/freezes during the game	a. check to make sure the chess engine is properly plugged into the wall outlet b. call distributor
I have a broken piece, or cannot resolve my issue	Contact your distributor
The pieces no longer stick to the magnets and the pieces fall off easily	Ensure the power cord is plugged into a wall outlet and that the board is turned "on."
You cannot hear the audio output	Reset it by pressing the off button on the chessboard and then unplugging the cord. Plug the cord back in and press the on button.

**Please note that to be accessible to our customer base, this user manual would be available in braille or audio.

19 Appendix J: Code - User Interface

```
import time, threading, wx, os, sys
from subprocess import call
from os import system
import pyttsx3
import numpy as np
import random

#Creating a window for my program with a specified title and size
app = wx.App()
window = wx.Frame(None, title = "Nico's Chess Timer", size = (700,400))
panel = wx.Panel(window)
popup = wx.Frame(None, title = "Time's up!", size = (200, 150))

startTimeEntry = 0.0

#Function to close the main window
def closeWindow(event):
    window.Destroy()
    os._exit(0)

#start time
startTime = 0.000001
whiteTime = 0.000001
blackTime = 0.000001

#startTimeEntry design
def initialSet(event):
    global startTime, whiteTime, blackTime, whiteRemainingTime, blackRemainingTime, timerThread
    engine = pyttsx3.init()
    engine.setProperty('rate',130)
    engine.say("Added 5 minutes to the timer.")
    engine.runAndWait()
    startTime += 300
    whiteTime = float(startTime)
    blackTime = float(startTime)
    whiteRemainingTime = wx.StaticText(panel, label = "Remaining time: \n%d seconds" %whiteTime, pos = (50, 50))
    whiteRemainingTime.SetFont(labelFont)
    blackRemainingTime = wx.StaticText(panel, label = "Remaining time: \n%d seconds" %blackTime, pos = (400, 50))
    blackRemainingTime.SetFont(labelFont)



    timerThread = threading.Thread(target = double_timer)
    timerThread.start()



#Function to close the popup
def hidePopup(event):
    popup.Hide()

#font options
labelFont = wx.Font(30, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
buttonFont = wx.Font(19, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
popupFont = wx.Font(38, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
```

```
#White
whiteLabel = wx.StaticText(panel, label = "White", pos = (50, 10))
whiteLabel.SetFont(labelFont)

#Black
blackLabel = wx.StaticText(panel, label = "Black", pos = (400, 10))
blackLabel.SetFont(labelFont)

turn = 0
paused = 1
mode = 0

def double_timer():
    global whiteTime, blackTime
    global whiteRemainingTime, blackRemainingTime
    global turn, paused


    while whiteTime > 0 and blackTime > 0:


        if turn == 0 and paused == 0:
            whiteTime = whiteTime - 1.0
            whiteRemainingTime.SetLabel("Remaining time: \n%d seconds" %whiteTime)
            timeFeed()

        elif turn == 1 and paused == 0:
            blackTime = blackTime - 1.0
            blackRemainingTime.SetLabel("Remaining time: \n%d seconds" %blackTime)
            timeFeed()

        time.sleep(1)




    if paused == 0:
        popup.Show(True)
        call(["osascript -e 'set volume output volume 50'"], shell = True)


        if whiteTime <= 0:
            pass
            engine = pyttsx3.init()
            engine.setProperty('rate',130)
            engine.say("Whites time is up..Game Over")
            engine.runAndWait()
        if blackTime <= 0:
            pass
            engine = pyttsx3.init()
            engine.setProperty('rate',130)
            engine.say("Blacks time is up..Game Over")
            engine.runAndWait()
```

```
def timeFeed():
    global whiteTime, blackTime
    global whiteRemainingTime, blackRemainingTime
    global turn, paused
    engine = pyttsx3.init()
    engine.setProperty('rate',170)
    if whiteTime > 59 and whiteTime < 60:
        engine.say(" 1 minute remaining of white's time")
        engine.runAndWait()
    if blackTime > 59 and blackTime < 60:
        engine.say(" 1 minute remaining of black's time")
        engine.runAndWait()

def switchTurn(event):
    global mode
    global turn
    global blackTime
    #pieces = (['knight'],['pawn'],['rook'],['bishop'])
    engine = pyttsx3.init()
    engine.setProperty('rate',99)

    if mode == 1:
        if turn ==0:
            engine.say(" Pawn position F2 moved to. F4")
            engine.runAndWait()
            turn = 1
        if turn == 1:
            time.sleep(5)
            blackTime = blackTime - 5.0
            engine.say(" Pawn position C7 moved to. C6")
            engine.runAndWait()
            turn = 0
    if mode ==0:

        if turn == 0:

            engine.say(" Pawn position F2 moved to. F4")
            engine.runAndWait()
            turn = 1

        else:
            engine.say(" Knight position B8 moved to. C6")
            engine.runAndWait()
            turn = 0

def startGame(event):
    global paused
    engine = pyttsx3.init()
    engine.setProperty('rate',130)
    engine.say("Timer has begun, Game has started! May the best chess player win")
    engine.runAndWait()
    paused = 0

def pauseGame(event):
    global paused
    paused = 1
def interval(event):
    global startTime, whiteTime, blackTime, whiteRemainingTime, blackRemainingTime, timerThread
    startTime += 300.0
```

```
def KeyDescriber(event):
    engine = pyttsx3.init()
    engine.setProperty('rate',170)
    engine.say(" Button 1 is to switch to opponents timer alternating turns. Button 2 is to start the timer be
    engine.runAndWait()

def playerCPU(event):
    global mode


    if mode == 0:
        engine = pyttsx3.init()
        engine.setProperty('rate',99)
        engine.say(" Initiated C P U mode")
        engine.runAndWait()
        mode = 1



    else:
        engine = pyttsx3.init()
        engine.setProperty('rate',99)
        engine.say(" Initiated into player mode")
        engine.runAndWait()
        mode = 0




#CPU/playerButton
playerCPUButton = wx.Button(panel, label = "CPU/Player(5)", pos = (240, 380), size = (200, 50))
playerCPUButton.Bind(wx.EVT_BUTTON, playerCPU)
playerCPUButton.SetFont(buttonFont)


#KeyDesriberButton
KeyDescriberButton = wx.Button(panel, label = "KeyDescriber", pos = (240, 430), size = (200, 50))
KeyDescriberButton.Bind(wx.EVT_BUTTON, KeyDescriber)
KeyDescriberButton.SetFont(buttonFont)

#intervalButton
intervalButton = wx.Button(panel, label = "Add 5min (4)", pos = (240, 330), size = (200, 50))
intervalButton.Bind(wx.EVT_BUTTON, initialSet)
intervalButton.SetFont(buttonFont)

#startButton
startButton = wx.Button(panel, label = "Start(2)", pos = (240, 230), size = (200, 50))
startButton.Bind(wx.EVT_BUTTON, startGame)
startButton.SetFont(buttonFont)

#switchButton
switchButton = wx.Button(panel, label = "Switch(1)", pos = (140, 160), size = (400, 50))
switchButton.Bind(wx.EVT_BUTTON, switchTurn)
switchButton.SetFont(buttonFont)
```

```
#Button to pause the game
pauseButton = wx.Button(panel, label = "Pause(3)", pos = (240, 280), size = (200, 50))
pauseButton.Bind(wx.EVT_BUTTON, pauseGame)
pauseButton.SetFont(buttonFont)



#Button to close the window
quitButton = wx.Button(panel, label = "Close", pos = (240, 480), size = (200, 50))
quitButton.Bind(wx.EVT_BUTTON, closeWindow)
quitButton.SetFont(buttonFont)

#label and button for the "Time's up!" popup frame
popupNote = wx.StaticText(popup, label = "Time's up!", pos = (3, 20))
popupNote.SetFont(popupFont)




popupButton = wx.Button(popup, label = "Okay", pos = (60, 90))
popupButton.Bind(wx.EVT_BUTTON, hidePopup)

window.Show(True)
app.MainLoop()
```

20 Appendix K: Code - Game Functionality

```
string decToBinary(int n){
  // Converts decimal to binary
    // array to store binary number
    int binaryNum[32];

    // counter for binary array
    int i = 0;
    while (n > 0) {

        // storing remainder in binary array
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }

    // printing binary array in reverse order
    string output;
    for (int j = i - 1; j >= 0; j--)
        output.append(to_string(binaryNum[j]));
    return output;
}

void removeLastMove(string &PGN, string &PGN_command){
  // Removes the last move from the PGN string
  PGN = PGN.substr(0, PGN.size()-5);
  PGN_command = PGN_command.substr(0, PGN_command.size()-5);
}

string getLastMove(string PGN){
  // Takes the substring of the last move from the PGN string
  string themove = PGN.substr(PGN.size()-5, 4);
  return themove;
}

string getLastMovedSquare(string PGN){
  // Takes the substring that is just the square the last piece was moved to
  string themove = PGN.substr(PGN.size()-5, 4);
  return themove.substr(2, 2);
}
```

```
void pieceRaisingOutput(string PGN, vector<string> PGN_vec){
  // Outputs all piece raising information
  if (PGN.size() == 0){
            std::cout<<"No pieces raised"<<std::endl;
          }else{
            std::cout<<"\nLast Move:\t\t"<<PGN_vec.back()<<std::endl;
            std::cout<<"Piece to raise:\t\t"<<getLastMovedSquare(PGN)<<std::endl;
            int square = moveToNumber(getLastMove(PGN));
            std::cout<<"Decimal Equivalent\t"<<square<<std::endl;
            string square_binary = decToBinary(square);
            std::cout<<"Binary Output:\t\t"<<square_binary<<std::endl;
            std::cout<<printPieceRaising(PGN_vec.back().substr(2, 2));
          }
}
```

```
void UCI::ChessboardLoop(int argc, char* argv[]){
  // Waits for a command from stdin, parses it and does the appropriate operation.
  // Modified from UCI::loop in original code.

  // Object Setup
  Position pos;
  string token, cmd;
  string PGN;
  vector<string> PGN_vec;
  string PGN_command = "startpos moves ";
  StateListPtr states(new std::deque<StateInfo>(1));

  pos.set(StartFEN, false, &states->back(), Threads.main());

  // Parsing
  for (int i = 1; i < argc; ++i){
      cmd += std::string(argv[i]) + " ";
    }

  do {
    // Waiting for input
    if (argc == 1 && !getline(cin, cmd)){
            cmd = "quit";
      }

        istringstream is(cmd);

        token.clear();
        // Avoid a stale if getline() returns empty or blank line
        is >> skipws >> token;

// Operations based on inputs:
        if (token == "quit" || token == "stop"){
          // Quit
          Threads.stop = true;

        }else if (token == "printposition"){
          // Prints current position
          istringstream is 2(PGN command);
```

```
// Operations based on inputs:
        if (token == "quit" || token == "stop"){
          // Quit
          Threads.stop = true;

        }else if (token == "printposition"){
          // Prints current position
          istringstream is_2(PGN_command);
          position(pos, is_2, states);
          std::cout<<pos<<std::endl;

        }else if (token == "move"){
          // Records input move, checks legality of move
          string themove;
          is>>skipws>>themove;
          if (themove.size() != 4){
            std::cout<<"Invalid move format"<<std::endl;
          }else{
            PGN.append(themove);
            PGN.append(" ");
            PGN_command.append(themove);
            PGN_command.append(" ");


            int PGN_vec_size = PGN_vec.size();
            if (    move(to_move(pos, themove), false) != "(none)" &&
                    move(to_move(pos, themove), false) != "0000"){
            // Checks move legality
              PGN_vec.push_back(move(to_move(pos, themove), false));
            }

            if (PGN_vec_size == int(PGN_vec.size())){
              // The move is not a legal move
              std::cout<<"Not a legal move"<<std::endl;
              removeLastMove(PGN, PGN_command);
            }else{
              // The move is a legal move
              istringstream is_2(PGN_command);
              position(pos, is_2, states);
              std::cout<<pos<<std::endl;

              std::cout<<"PGN_vector: ";
```

```
            std::cout<<"PGN vector: ";
            for (int i = 0; i < int(PGN_vec.size()); i++){
              std::cout<<to_string(i+1)<<". "<<PGN_vec[i];
              if (i != int(PGN_vec.size())-1)
                std::cout<<" ";
            }
            std::cout<<"\n\n";
            // Piece raising output visualization
            pieceRaisingOutput(PGN, PGN_vec);
          }
        }

      }else if (token == "bestmove"){
        // Outputs depth 22 best computer move
        istringstream is_2("go depth 22");
        go(pos, is_2, states);

      }else if (token == "removelastmove"){
        // Removes last move
        if (PGN.size() != 0){
          removeLastMove(PGN, PGN_command);
        }
        if (PGN_vec.size() > 0){
          PGN_vec.pop_back();
        }

      }else if (token == "getpieceraise"){
        // Piece raising output visualization
        pieceRaisingOutput(PGN, PGN_vec);

      }else if (token == "getPGN"){
        std::cout<<"PGN vector: ";
            for (int i = 0; i < int(PGN_vec.size()); i++){
              std::cout<<to_string(i+1)<<". "<<PGN_vec[i];
              if (i != int(PGN_vec.size())-1)
                std::cout<<" ";
            }
            std::cout<<"\n\n";
      }else
        sync_cout << "Unknown command: " << cmd << sync_endl<<std::endl;
      std::cout<<"\n**************************************\n\n";


  } while (token != "quit" && argc == 1);


}
```

```
//               Custom written and modified for RPI IED Spring 2021 Group 1 - Blind Chessboard
//             |-------------------------------------------------------------------------------|

int moveToNumber(string themove){
  //Converts a move such as "e2e4" to a number from 0 to 63 corresponding
  //to the 64 squares ordered bottom left to top right
  string secondsquare = themove.substr(2, 2);
  int index = toupper(secondsquare[0]) - 'A';
  return index + 8 * (stoi(secondsquare.substr(1,1))-1);
}

string printPieceRaising(string thesquare){
  // Prints visualization of the single square that is raised on the chessboard
  string output;
  std::string s = "+---+---+---+---+---+---+---+---+\n";
  vector<string> ranks{"8", "7", "6", "5", "4", "3", "2", "1"};
  vector<string> files{"a","b","c","d","e","f","g","h"};
  int i = 8;
  for (vector<string>::iterator it = ranks.begin(); it != ranks.end(); it++){
    output.append(s);

    for (vector<string>::iterator itr = files.begin(); itr != files.end(); itr++){
      output.append("| ");
      if (*it == thesquare.substr(1, 1) && *itr == thesquare.substr(0, 1)){
        output.append("#");
      }else{
        output.append(" ");
      }
      output.append(" ");
    }
    output.append("|");
    output.append("  ");
    output.append(to_string(i));
    output.append("\n");
    i--;
  }
  output.append(s);
  output.append("  a   b   c   d   e   f   g   h\n");
  return output;
}
```

21 Appendix L: Code - Piece Detection

```
1   import random_input
2
3   #Known UID bank
4   bpawns = [0b1,0b10,0b11,0b100,0b101,0b111,0b1000,0b1001]
5   wpawns = [0b1010,0b1011,0b1100,0b1101,0b1111,0b10000,0b10001,0b10010]
6   brook = [0b10011,0b10100]
7   wrook = [0b10101,0b10111]
8   bbishop = [0b11000,0b11001]
9   wbishop = [0b11010,0b11011]
10  bknight = [0b11100,0b11101]
11  wknight = [0b11111,0b100000]
12  bqueen = [0b100001]
13  wqueen = [0b100010]
14  bking = [0b100011]
15  wking = [0b100100]
16
17  uidlist = [bpawns,wpawns,brook,wrook,bbishop,wbishop,bknight,wknight,bqueen
18
19
20
21  def match(uid,array):
22      """A method for matching UIDs with UID bank"""
23      for i in range(12):
24          for j in range(len(uidlist[i])):
25              if (uid == uidlist[i][j]):
26                  return i
27
28  def decode(matrix):
29      """A method from decoding UIDs"""
30      decoded = list()
31      for i in range (8):
32          decoded.append(['a'])
33          for j in range(7):
34              decoded[i].append('a')
35      for i in range (8):
36          for j in range (8):
37              r = match(matrix[i][j],uidlist)
38              if (r == None):
39                  decoded[i][j] = 'O'
```

```
            elif(r == 0):
                decoded[i][j] = 'bP'
            elif(r == 1):
                decoded[i][j] = 'wP'
            elif(r == 2):
                decoded[i][j] = 'bR'
            elif(r == 3):
                decoded[i][j] = 'wR'
            elif(r == 4):
                decoded[i][j] = 'bB'
            elif(r == 5):
                decoded[i][j] = 'wB'
            elif(r == 6):
                decoded[i][j]= 'bN'
            elif(r == 7):
                decoded[i][j]= 'wN'
            elif(r == 8):
                decoded[i][j]= 'bQ'
            elif(r == 9):
                decoded[i][j]= 'wQ'
            elif(r == 10):
                decoded[i][j]= 'bK'
            elif(r == 11):
                decoded[i][j]= 'wK'

        return decoded


original, move, text = random_input.random_inputs()

ded = decode(original)

print('Original:')
for i in range (8):
    print(ded[i])

print("\n")
demove = decode(move)
print('Move:')
for i in range (8):
    print(demove[i])

print("\n")
print(text)
```

22 Appendix M: Code - Game Feedback

Text

```
// possible 6bit binary inputs
// each 6bit binary input represents position on the chess board
int A_1 = B000001;
int B_1 = B000010;
int C_1 = B000011;
int D_1 = B000100;
int E_1 = B000101;
int F_1 = B000110;
int G_1 = B000111;
int H_1 = B001000;

//Assign each vibration motor to a GPIO pin
int vA1 = 2;
int vB1 = 3;
int vC1 = 4;
int vD1 = 5;
int vE1 = 6;
int vF1 = 7;
int vG1 = 8;
int vH1 = 9;

//6 bit binary input recieved from Game Functionality Subsystem
int position = B000001;

//Define GPIO pins as outputs
void setup()
{
  pinMode(vA1, OUTPUT);
  pinMode(vB1, OUTPUT);
  pinMode(vC1, OUTPUT);
  pinMode(vD1, OUTPUT);
  pinMode(vE1, OUTPUT);
  pinMode(vF1, OUTPUT);
  pinMode(vG1, OUTPUT);
  pinMode(vH1, OUTPUT);
}

//initialize loop for each board position
void loop() {

    posA1();
    posB1();
    posC1();
    posD1();
    posE1();
    posF1();
    posG1();
    posH1();
```

```
49
50  }
51
52  // position A_1
53  void posA1(){
54    if (position == A_1) {
55    for(int i = 1; i < 8; i++)
56     {
57     digitalWrite(vA1,HIGH);
58     delay(1000);
59     digitalWrite(vA1,LOW);
60     delay(500);
61    }
62    }
63  }
64
65  // position B_1
66  void posB1(){
67    if (position == B_1) {
68    for(int i = 1; i < 8; i++)
69     {
70     digitalWrite(vB1,HIGH);
71     delay(1000);
72     digitalWrite(vB1,LOW);
73     delay(500);
74    }
75    }
76  }
77
78  // position C_1
79  void posC1(){
80    if (position == C_1) {
81    for(int i = 1; i < 8; i++)
82     {
83     digitalWrite(vC1,HIGH);
84     delay(1000);
85     digitalWrite(vC1,LOW);
86     delay(500);
87    }
88    }
89  }
90
91  // position D_1
92  void posD1(){
93    if (position == D_1) {
94    for(int i = 1; i < 8; i++)
95     {
96     digitalWrite(vD1,HIGH);
97     delay(1000);
98     digitalWrite(vD1,LOW);
99     delay(500);
100   }
101   }
102 }
103
```

```
// position E_1
void posE1(){
  if (position == E_1) {
  for(int i = 1; i < 8; i++)
   {
   digitalWrite(vE1,HIGH);
   delay(1000);
   digitalWrite(vE1,LOW);
   delay(500);
  }
  }
}

// position F_1
void posF1(){
  if (position == F_1) {
  for(int i = 1; i < 8; i++)
   {
   digitalWrite(vF1,HIGH);
   delay(1000);
   digitalWrite(vF1,LOW);
   delay(500);
  }
  }
}

// position G_1
void posG1(){
  if (position == G_1) {
  for(int i = 1; i < 8; i++)
   {
   digitalWrite(vG1,HIGH);
   delay(1000);
   digitalWrite(vG1,LOW);
   delay(500);
  }
  }
}

// position H_1
void posH1(){
  if (position == H_1) {
  for(int i = 1; i < 8; i++)
   {
   digitalWrite(vH1,HIGH);
   delay(1000);
   digitalWrite(vH1,LOW);
   delay(500);
  }
  }
exit(0);
}

```