Tactile Chess

Authors: Mukundh Balajee, Edison Aviles, Juan Mejia
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system, predominantly for blind users, capable of communicating the user's moves on a physical chess board to an online platform, while also providing feedback based on different game states and user actions. This system combines vocal and tactile cues, to provide blind users with an opportunity to play an online game of chess through a custom-made chess board. This one-of-a-kind system is capable of providing feedback based on moves made within 500ms and retrieving information from a public chess server within 500ms of an opponent making a move, thereby giving the user a seamless experience.**

***Index Terms*—3D Printing, 3D Modeling, Chess, Hall Effect Sensors, PCB Fabrication, Tactile, Vocal Cues**

1 INTRODUCTION

As of the end of 2022, the number of users in the online chess community is over 100 million [1]. We realized that a section of the market was neglected: blind users. In order to address this problem, our team decided to build a custom chess-board, with accessible features for blind users, which can connect to an online chess platform, like *lichess.org*[4]. Our main stakeholders are blind people. However, with the use of a web application, we hope to provide beginner and novice chess users with a physical board, to automatically simulate an online chess game.

Currently, there is no technology that provides blind users the ability to play online chess without the use of an iPad or another such device. There are custom-made boards, which help simulate moves in better ways, however, these systems are way more expensive (around $\$2,000$) and not affordable [7]. Furthermore, such boards are not accessible to blind users. Our board aims to bridge the gap between the online chess community and blind people as well as offer a more affordable solution in the market.

For blind users, our board will provide vocal cues. To make our board more accessible, we are engraving braille notations for the board's coordinates, and a distinctive feature on the pieces so that users can differentiate the color of the pieces based on touch. This will help them differentiate pieces and locate them when vocal cues are provided. The board will have tactile features to allow the blind user to identify black and white tiles on the board. To ensure the pieces are placed in the correct spot and do not fall over during the game, the board and pieces will have a lock-and-key mechanism with a peg on the base of every piece and a hole in every tile. To make the gameplay seamless, the user will be able to start, end, and confirm games using three buttons on the board. For the rest of our stakeholders, we hope to provide a chessboard that will help simulate an online chess game on a physical board, to help understand the basics of the game and visualize the board.

2 USE-CASE REQUIREMENTS

In order for our project to satisfy our users, we have focused our efforts on user experience, accessibility, board integrity, accuracy, and latency.

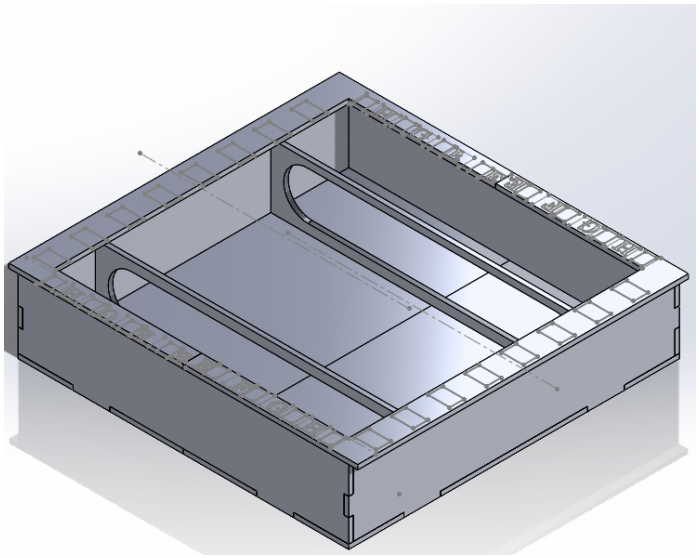
- For our users to have a seamless experience in playing a game of online chess on our board, users need to be able to set up games in minimal time. Our board must be able to connect to *lichess.org*, and start the game experience for the user in under 25s. During a game of chess, our board must be able to assist blind users in making their moves and help reflect opponents' moves on the board with vocal and tactile cues. With our custom board and pieces users must be able to make moves with a 100% accuracy rate, almost instantaneously. We also want to make sure that our user is able to interpret the audio cues that our board provides by ensuring the audio level is high enough that they can be heard from a maximum of 4ft from the board. Currently, we hope to cater to only English-speaking users. However, we hope to add language flexibility in the future.

- In order for accurate game-play and to maintain the board and game integrity, we need to ensure the physical and online chess games are identical. Our board will be able to detect the color, type, and location of every active piece on the board at all times with 100% accuracy. With the help of software checks and verification, we hope to ensure that the moves made by the user and the game state are valid at any given point. The board will provide feedback to the user regarding their every move, allowing the user to know what is currently happening in the game.

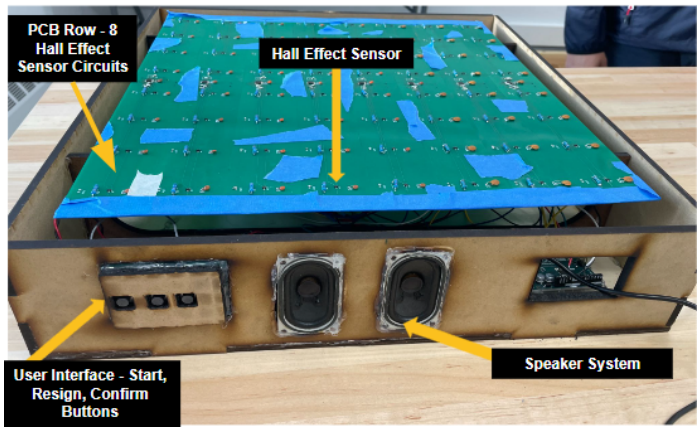
- To ensure that the user has a similar experience to playing a game of 2-player chess, we hope to ensure a 100% accuracy with our detection systems in our physical board (piece detection, move generation, FEN generation, etc.) We hope to ensure minimal latency for a seamless game experience for our users. To help our users navigate the board and get used to standard chess coordinates and notations, we provide a chess board with etched coordinates and braille notations, to help the users map the cues to the actions on the board. Finally, we want our feedback cues to

have an accuracy of 100% to ensure the entire system works as expected.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION



(a)



(b)

Figure 1: System description. (a) Board Design Model (b) Final Chessboard and Encasing

3.1 Chess Board & Pieces

We created a custom chess board following standard chess guidelines and incorporated accessible features for our blind users. The accessible features include varying heights of black and white tiles, braille engraving for the chess coordinates, pointed tips on black pieces, and a lock-and-key mechanism between each piece and tile. Below the chess board, is a PCB with all our circuits and electronics needed to simulate the physical game of chess on the online platform. With the help of linear Hall Effect Sensors, we hope to be able to distinguish between pieces and help simulate an accurate chess experience.

3.2 Hardware

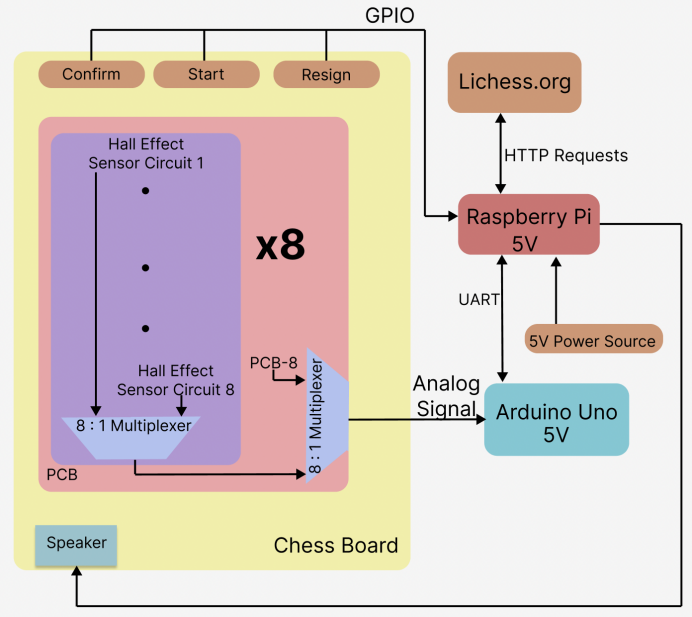


Figure 2: Hardware System Block Diagram

3.2.1 Printed Circuit Board

To detect pieces on the board, we mounted a custom PCB underneath our chess board. Our circuitry ensures each tile has one Hall Effect Sensor, resulting in 64 Hall Effect Sensors. Hence, we have grouped our PCBs as rows and connected each row to an 8:1 analog mux. The output of these 8 PCBs is then connected to another 8:1 analog mux. The select lines of the final mux are controlled by the Arduino, which helps us cycle through the 64 sensors, before sending the move information over to the Raspberry Pi.

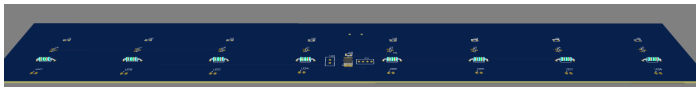


Figure 3: 3D-modeled PCB

3.2.2 Raspberry Pi & Arduino Uno

The Arduino Uno serves as an interface between information received from the PCB and information being sent to the RPi. The Arduino controls the select lines for the

muxes, which help us cycle through the 64 sensors, to detect voltage changes and subsequently send the detected change information over to the RPi. The Raspberry Pi houses all our software systems, and also serves as a central data hub for our system, for data being piped to and from *lichess.org*. The bulk of user move validation and board integrity checks happen at the RPi. The RPi then sends valid moves made by the user to *lichess.org*, and vocalizes opponents' moves from *lichess.org* back to the user.

3.2.3 Chessboard Components

The chessboard consists of components that help provide feedback cues, and tactile cues to the user, and help control a game. The board consists of a speaker circuit, which helps provide feedback cues regarding the current chess game, to the user. The board also has a button system, which allows the user to start and end a game, and also an 'OK' button to help in various processes like calibration, and set up of the chess board.

3.3 Software

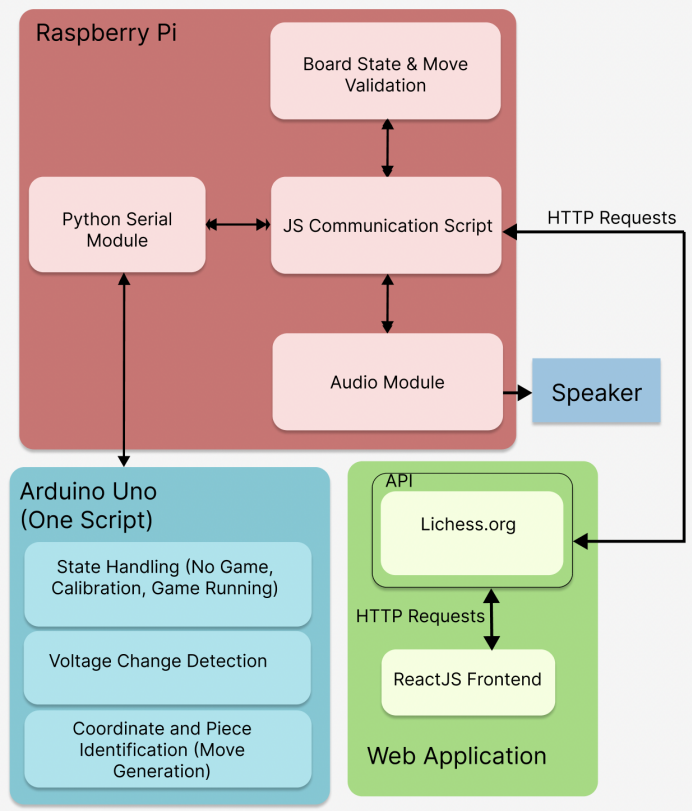


Figure 4: Software System Block Diagram

3.3.1 Legality Check

To ensure minimal latency between subsystems, we are performing state and move checks locally before communicating the information to the API, given the API's response time. We are validating board states and move legality, with the help of the *Stockfish API*[6]. The Stockfish API is an open-source stockfish class that allows the system to integrate the Stockfish chess engine to check the current board state, check for illegal moves, stalemates, and checkmates, and provide feedback through the RPi's speakers within our goal of 500ms.

3.3.2 Audio Module

To help vocalize feedback cues from our RPi, we have an audio module, which parses through data received and vocalizes information needed by the user to play the game. The audio module also communicates back to the other subsystems, in the case we expect our user to correct or perform a specific action, thereby ensuring correct game flow between our systems.

3.3.3 Web Application & Communication Script

In order to centralize the communication through our entire system and *lichess.org*, we created a JavaScript module that simultaneously cases on the information being piped between our subsystems and *lichess.org*, and on the information piped to and from the Arduino Uno. The module calls on other custom modules, such as our Move Legality Checker and Audio Module, based on the information being processed. Using our existing systems, we were able to create a Web Application based on the game state retrieved from *lichess.org*, to ensure users do not have control of the virtual board. The Web App renders the live game using the React Chessboard[2] and Chess.JS[3] libraries and serves as a central hub for the user to interact with their live games. This aspect of our system is mainly focused on beginner and novice players that will also be using our product.

4 DESIGN REQUIREMENTS

The following design requirements are quantified metrics related to our requirements in Section 2.

4.1 Accuracy Requirements

In order to ensure a seamless transition between the physical board and the online game, we aim to achieve a 100% accuracy in distinguishing colors, and 100% accuracy in identifying pieces. We want our audio module to vocalize feedback accurately, and ensure a 100% accuracy in vocalization and communication with other subsystems. With regards to our serial communication script, we aim to achieve a 100% accuracy in information being sent between the RPi and the Arduino. With regards to our software systems, we want to validate moves and check our board's integrity with a 100% accuracy to ensure the gameplay is accurate, across our subsystems.

4.2 Latency Requirements

Our system latency goal is 1500ms or less, to validate a user's move and reflect the move on the online platform, if it is valid, simultaneously provide vocal feedback regarding the user's move, and vocalize the opponent's moves through the board's speaker circuit. This process involves multiple subsystems with their latency breakdowns highlighted below:

- **User button press** - 20ms (delay set in the Arduino, to ensure smooth flow of control
- **Collect sensor data and relay information to Arduino** - 320ms (almost instantaneous reads from each sensor, but a delay of 5ms in Arduino code for 64 sensors to ensure the MUX resets its state properly)
- **Check the board integrity and move legality** - $<$ 50ms (instantaneous check to see if given FEN is legal and given move can be made on the FEN)
- **Send move from RPi to *lichess.org*** - 400ms (as mentioned on the *lichess.org* website [5])
- **Vocalizing feedback to the user** - 300ms (to account for minor delays in vocalizing longer texts)

Our latency goals are optimistic, yet achievable. The total latency comes out to 1000ms, which gives us at least 500ms to allow for any extraneous factors which could potentially affect our latency goals.

4.3 Power Requirements

The entire system will be powered by a wall outlet. The RPi will be connected to a constant power source, and will also power our buttons, speakers, and Arduino. Since we need only 5V for each Hall Effect Sensor, we will use the 5V supply on the Arduino Uno to power our PCB.

4.4 Accessibility requirements

We want our board to be as accessible as possible, especially to our blind users. We want them to be able to distinguish pieces and colors with a 100% accuracy. With the help of the braille notations for the coordinates, we want them to be able to identify coordinates with a 100% accuracy. We want our users to be able to identify and distinguish between a black and a white tile with 100% accuracy. Finally, we want our users to be able to control our speakers to ensure it isn't too loud or too soft.

5 DESIGN TRADE STUDIES

5.1 Chess Board and Pieces

Since we decided to make a custom-chess board and include several accessibility features, we were between two options: Laser cutting and 3D printing. Since we had no experience with 3D modeling, we quickly realized the number of iterations that would be needed to perfect the board, would result in budget issues. Hence, we pivoted our design to be more economical and made our board and encasing by laser-cutting wood. We chose to still 3D print the pieces to ensure we can get accurate models which are easy to recognize. The reason we chose wood and plastic as the two materials for our physical chess board and pieces, was because of the minimal interference offered by these materials in the presence of a magnetic field, thereby allowing us to detect pieces.

5.2 Sensors

For our sensors, we chose to use Hall Effect Sensors to help detect a magnetic field. We chose between unipolar and bipolar sensors, and also between linear (analog) and digital Hall Effect Sensors. Since we plan to distinguish colors based on the polarity of the magnet, we plan to use bipolar sensors as it provides a more robust architecture. This will help us distinguish pieces' colors by flipping magnets (polarity) instead of varying magnetic strength. Between the linear and digital sensors, we chose to use a linear Hall Effect Sensor. A linear sensor helps us receive varying voltage outputs for varying magnetic fields which helps us identify each unique piece on the board, in order to maintain board integrity. Hence, we chose to use the Texas Instruments Hall Effect Sensors (DRV5055) as these sensors operated in the range of voltages that we planned to use to power our sensors, with high magnetic sensitivity to ensure we can achieve a wide range of values, provide low-noise output, and cater to all of our above requirements better than other brands and sensors, and at a more affordable rate.

5.3 Multiplexers

The choice of using an 8:1 multiplexer was because of our choice for PCB Fabrication. Since the board is being fabricated in rows of 8, we chose to use an 8:1 multiplexer for each PCB and connect the output of the 8 PCBs to another 8:1 MUX to help sequentially cycle through 64 sensors. This architecture helps optimize our performance and removes the need for an ADC, which would be more expensive and have more steps to integrate into our project. Regarding latency, the MUX we have chosen allows us to reach our goals, as we provide a 5ms delay between each sensor, to allow the MUX to reset and read the next value. this way, when we cycle through our sensors, the maximum amount of time it will take is 320ms.

$$\text{Delay} \times \text{Number of Hall Effect Sensors} \tag{1}$$

$$= 5ms \times 64 = 320ms$$

5.4 PCB Fabrication

We designed our PCBs as individual rows, which consist of 8 Hall Effect Sensor circuits in parallel, to ensure each sensor gets the same voltage (5V), which are all connected to a single 8:1 MUX. Each hall effect sensor is 2" apart from each other, to ensure it follows standard chess guidelines. The outputs of the 8 8:1 MUXES on the PCBs are connected to a 9th MUX which is connected to the Arduino, to allow us to cycle through the inputs. The reason we chose this architecture for our PCB was to facilitate any debugging we had to do during the process of fabrication, minimize costs and the lack of enough Analog pins on the Arduino Uno, to be able to read the outputs of all 8 MUXES.

5.5 Power

To power our board we had two options: a Lithium-Ion battery or a wall outlet. We initially decided to use Lithium-Ion batteries, however, since we ran into budget issues and realized that all our systems could be powered by the 5V and 3.3V power sources from our RPi and Arduino, we changed our approach to use a wall-outlet instead, to power the RPi, and power the rest of our systems (Arduino, buttons, speaker, etc.) through the RPi's USB ports and GPIO pin headers.

5.6 Raspberry Pi & Arduino Uno

For our system, we chose to use an Arduino Uno, instead of an ADC to read the values outputted by our sensors, as an Arduino provides more robust functionality for our post-MVP goals, and also helps us implement the MUX logic for all the MUXES easily, without creating a new IDE (as would be the case for an ADC). We chose the Raspberry Pi 4 Model 4 4GB. One of the main reasons we chose this model RPi was the connectivity that it offered. The 40-pin GPIO header allows us to power our buttons and other circuits (if needed), without the use of an external power source. The 4-pole stereo audio port would allow us to connect a speaker to provide vocal cues. The 4GB ram would give us ample storage and performance to store the game state and check the legality of moves. The 4 USB ports would give ample ports to power the Arduino, and other peripherals that we might need for testing purposes. The 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless connectivity would allow the RPi to connect to our Web-App. We are also going to supply the RPi with a 16GB micro-SD card to store the OS, our Python scripts, and a list of moves for different games.

5.7 State Validation

In order to efficiently give our user feedback on the current state of the board, we decided to incorporate state validation prior to communication with any external platforms. This will minimize latency by locally keeping track of a chessboard state and validating the users move based on the stored state. To accomplish this, we looked at several valid move generators and chess engine APIs - including Stockfish API (an open-source API that gives developers the ability to use the Stockfish chess engine) and Chessnut. We decided to move forward with using Stockfish since it provided extensive documentation and a large variety of chess related methods, such as loading a game based on a specific FEN, making moves on a chessboard, and determining if a stalemate or checkmate is present. Moreover it provides feedback in around 2ms, which makes up for the large amount of latency generated by making calls to *lichess.org*. Furthermore, lichess also uses the Stockfish engine on their platform, which allows us to keep communication and behavior consistent throughout the entire system cycle.

5.8 Legality Check

To validate a move, we will receive information regarding the moved piece and its initial and final positions. With this information, we can generate a move that needs to be validated by our checker, to ensure the piece is moved in its allowed pattern (Bishop can only move diagonally, Rook can only move in straight lines, etc.) We plan to use the Stockfish API (described above) to help us validate moves and check for other game states (stalemate, checkmate, casting, etc.) Even though *lichess.org* uses the Stockfish API, it takes up to 500ms to provide feedback from *lichess.org* to the RPi. Since we want to provide feedback to our users before their move is reflected on the online platform and hit our latency targets, we decided to use the *Stockfish* chess engine in our RPi, to help reduce the latency. This way, we validate a user's move before communicating with the external platform and provide any feedback regarding the user's move back to them before changing the game state.

5.9 Website Infrastructure

Despite our main target audience being blind users, we decided to create a website to display user and chess-related information. The main reason for incorporating this web application into our project was to reach other communities, especially novice or beginner chess players. By allowing novice players to analyze their games, view daily puzzles, and observe their current live games we hope to provide a platform in which users can learn how to play chess through the use of visual cues on the board and on the website. We also want to make sure our users cannot make any changes on the virtual board, thereby ensuring the game is played from one source. We chose to implement our web application using a ReactJS frontend and NodeJS backend. The main reasons for this decision were to utilize React's asynchronous rendering functionalities and to use a JavaScript framework since lichess documentation was written in JavaScript. Furthermore, the team found a variety of chess-related libraries that were all supported in

JavaScript - including Chess.js and React Chessboard. It is also incredibly simple to create a React app and connect it to a Node backend, which in turn allows for easy setup and deployment.

5.10 Online Chess Platform

For our online chess platform, we looked at various options. We began by contemplating using the largest online chess platform, *chess.com*, but quickly realized that their API was limited to exporting data and wouldn't allow users to make moves during a live game. After coming to this conclusion, we pivoted and began considering two main options, *lichess.org* and creating our own online chess platform. We considered lichess to be the most reasonable answer since it would allow our users to connect with one of the largest online chess communities. Moreover, the service provides a variety of endpoints that would allow us to interact with live games and utilize an already-established tool set for chess games. The only limiting factor lichess posed was its latency limits - communication between all of our systems and lichess will take at least 500ms. In order to mitigate this and maximize all communication with the platform we decided to incorporate move validation before sending moves to the online server - this way the user will receive feedback on their move faster than they would if they communicated directly with lichess in order to receive feedback.

6 SYSTEM IMPLEMENTATION

6.1 Chessboard & Pieces

Our custom chessboard is broken down into two layers, the top layer consists of a custom laser cut and 3D printed blind-friendly chessboard surface - where the user will interact with game pieces, and the bottom layer consists of a PCB which is used for piece and move detection. The chessboard has been made blind-friendly by varying the height between white and black tiles and incorporating the use of braille notation to identify tile coordinates. On the other hand, the PCB layer consists of 64 hall effect sensors spaced out by 2 inches between each sensor to allow for a standard chessboard size. Game pieces have been custom-made and 3D printed in order to make them blind-friendly. In order to achieve blind-friendly game pieces we designed a key and lock mechanism between the pieces and board while also adding a pointed tip to each individual piece to help the user identify between different pieces and piece colors. Inside each of the pieces, there is a varying amount of magnets, which vary in size and polarity based on the game piece and piece color.

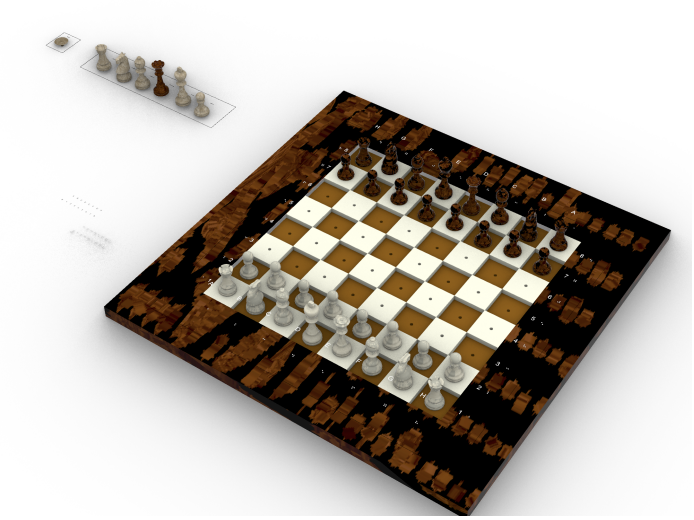


Figure 5: 3D-modeled chess set

6.2 Board Calibration

To ensure all our Hall Effect Sensors are reading reasonable values, we calibrate our board at the beginning of every game by removing all the pieces from the board. This way if we see an erroneous value, we can immediately identify the problem. We also use this calibration step to know our min and max base voltages (when no sensors are present), to help us generate a FEN. Once we get the base voltages, we set up the board to a starting game position, and scan our board again to find our min and max values for each piece, and also ensure that the pieces are within the pre-determined ranges of voltages. This way, our system always checks for any errors before a game is started, and flags any errors with the board, preventing the user from playing or asking them to fix it before moving along (if it is a hardware problem).

6.3 Move Sensing

In order to facilitate move sensing we designed a PCB which is installed in the bottom layer of our chessboard. The PCB uses the DRV5055 ratiometric hall effect sensors from Texas Instruments to detect when pieces are placed on a tile. Furthermore, the sensors allow us to differentiate between pieces and color by the varying polarity and magnetic strengths produced by the pieces. The PCB design will be separated into rows which will house 8 ratiometric hall effect sensors spaced out by 2 inches. The outputs of the sensors in each row have a dedicated 8:1 multiplexer, since there are 8 rows on a chessboard we are using a total of 8 multiplexers (TMUX1208PWR). The 8 multiplexers feed their analog outputs to an Arduino Uno, which then identifies the piece being moved and the coordinate of the voltage change and sends it over to the Serial Module Script running on the RPi. The module processes the data and formulates a move when it identifies a move is finished being made. Afterwards it runs a legality check

on the user's move before communicating the information to *lichess.org*. Once the opponent has made a move and it has been vocalized through the speakers, the user will then proceed to replicate the move on the physical board. The system will detect the replicated move the same way it detects the user's inputted move, after which the user receives vocal feedback notifying them if the move was replicated correctly or not.

6.4 Hardware User Interface

The board's user interface consists of a speaker and 3 buttons. The speaker is used to provide vocal feedback about opponent moves and board and move validity. The speaker's circuitry is powered by the Raspberry Pi and is utilized by our Audio Module during different instances of a live game to provide feedback.

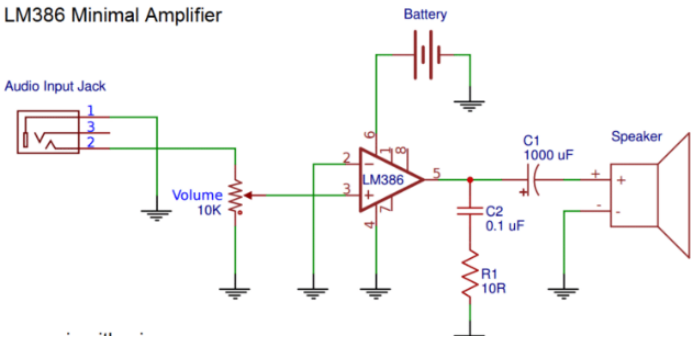


Figure 6: Amplified Speaker Circuit

Furthermore, the 3 buttons have specific behaviors assigned to each of them. The first button is used to start seeking a game with an online opponent, the second is used to resign the current live game, and the third is used to confirm an action. In order to power these buttons and connect them to the Raspberry Pi we will use a basic switch circuit - see figure 7. Once the Raspberry Pi is notified of a request through its GPIO pins, we case on the pressed button in our JavaScript communication module. In the case of an online seek, the script will send an HTTP POST request to *lichess.org* which will establish a stream of events between the script and *lichess.org*. In the case of a resign game, the system will verify if a game exists and if it does it will resign the live game. As for the confirm button, the script will first verify if there is an action awaiting to be confirmed and if there is it will confirm the action.

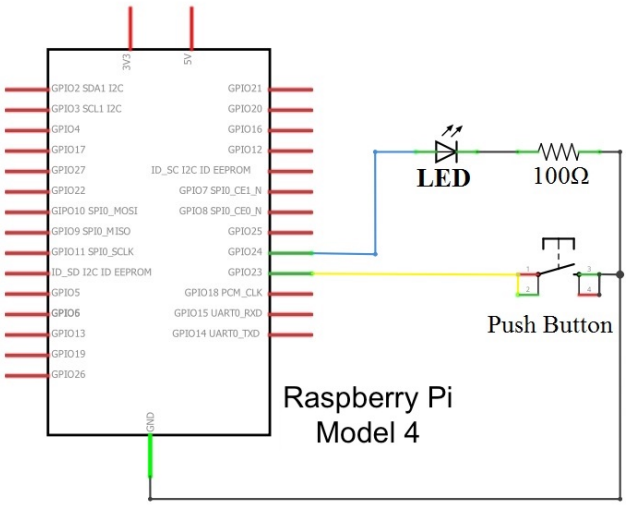


Figure 7: Basic Push-Button Circuit

6.5 State Validation

State validation is performed in the Raspberry Pi through a series of modules - see figure 8. Once a game start signal is received by our JavaScript communication module, the system will case on whether it is the opponent's or user's turn. If it is the user's turn to make a move, the module will begin to listen for information from the Arduino. Once a minimum of two coordinates are received (representing a basic move), our Serial Communication script will formulate a move that consists of the piece and the two coordinates sent by the Arduino in the following string format - piece + first coordinate + second coordinate. The JavaScript module will then send the move and the current game's FEN to our Move Validation module to verify if the move is legal. The Move Validation module will create a stockfish object, using the Stockfish API, and load the passed FEN as the current game state. Afterwards, the script will use the object to verify the legality of the passed move on that FEN. If the move is legal, the Move Validation module will return an updated FEN and the JavaScript module will call upon the Audio Module to give the user feedback before sending the move to *lichess.org* through an HTTP POST request and updating the local state. In the case the move is invalid, the JavaScript will indicate the Audio Module to give different feedback and wait for the user to move the piece back to its original position before making a new move. In the case it is the opponent's turn, the system will wait for *lichess.org* to give a game state update through the established pipeline before informing the user through the Audio Module of the opponent's move. The user will then need to replicate the opponent's move on the physical board before proceeding to make their own move.

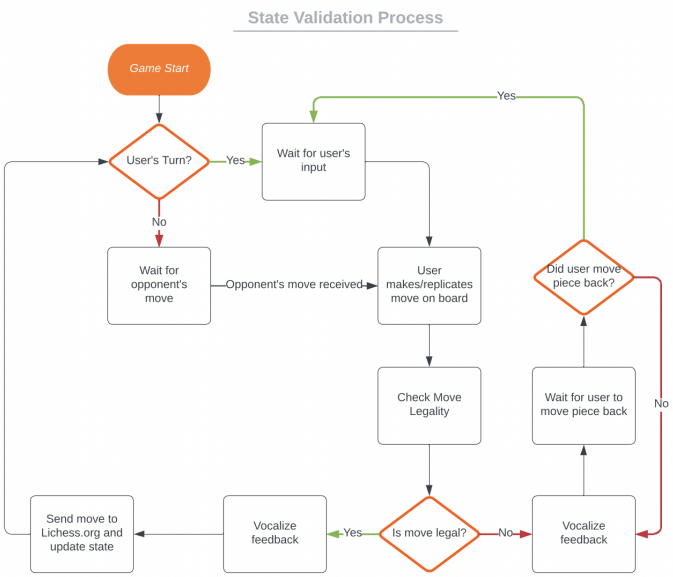


Figure 8: State Validation Flowchart Diagram

6.6 Web Application

The website frontend was created using ReactJS and CSS in order to display different user information, as well as live games. Furthermore, the ReactJS frontend is also responsible of communicating with *lichess.org* through a series of HTTP GET requests - see Figure 4. We chose to use ReactJS and CSS for frontend purposes since ReactJS allows for an easy way of incorporating JavaScript into HTML, moreover by using ReactJS's state management we can easily display live games by using React states to asynchronously update the website's local game state copy. Additionally, most of the documentation available for *lichess.org*'s API was written in JavaScript which made it easier to navigate and translate to our website's specific needs. The frontend displays the current live game by acquiring the current live game's id through an HTTP GET request to *lichess.org*. After identifying the game id, the page will make a second HTTP GET request to *lichess.org* which will establish a stream between our web application and *lichess.org*. Through this stream *lichess.org* will be sending game updates including the current FEN and the last move made. With this information, the frontend is able to render the current live FEN using a library called react chessboard and create a local chess state using Chess.js which will be linked to the rendered chessboard. Throughout the rest of the game, the frontend updates the chessboard state via the moves being streamed to the frontend by *lichess.org*.

6.7 Power

We are powering our system through our RPi, which is connected to a wall outlet using a 12V cable. The Arduino Uno is connected to the RPi using a USB B to USB A cable. The Arduino is then powering our PCB via a 5V pin; we are able to do this since all the components on our PCB are in parallel and require a maximum of 5V. Moreover, the amperage draw from the mux and hall effect sensor components are almost negligible.

7 TEST & VALIDATION

To help our testing plans, we plan to follow test-driven development, to ensure each subsystem is tested before integrating with the existing system. Our system will be tested against a testing script that simulates several games with different edge cases, to increase the robustness of our system, and also ensure a 100% accuracy. We also plan to do separate user testing in order to improve our user experience and accessibility.

7.1 Results for User Experience and Accessibility

In order for us to get a sense of how our piece and board design was affecting a user's ability to play chess, we ran a user study of 40 CMU students. Our main objective was to see if they could easily identify the color and type of piece they wanted while blindfolded using just their sense of touch. We would also ask them if they could identify whether a series of tiles placed in front of them were either black or white tiles. The results are as follows:

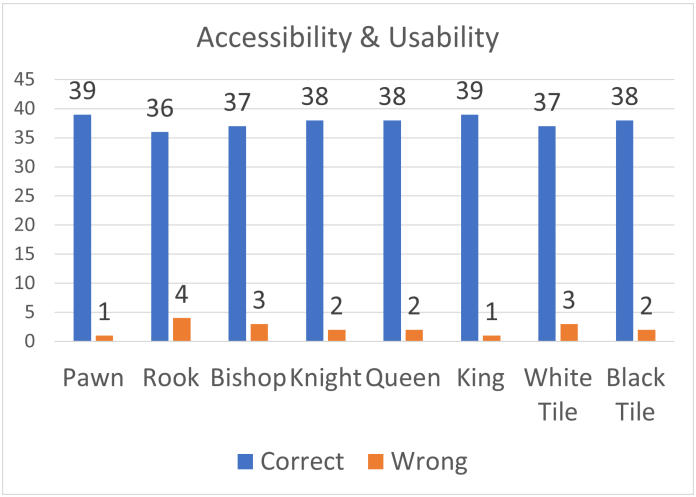


Figure 9: Accessibility & Usability Measurements

7.2 Results for Piece Detection

As achieving a 100% piece detection is one of the most important use-case and design requirements for our project, we tried to be as thorough as possible in our unit and in-game testing. Before we ran our system through an entire chess game, we first performed unit testing on each individual hall effect sensor on the board. We placed all 32 pieces on each individual tile to make sure that the voltage output that our Arduino was receiving was corresponding and similar to the voltage output that we were expecting

for each type of piece. After establishing an initial range for each individual type of piece, we moved to test all 64 sensors in unison while playing 5 games of chess and recording the voltage output for all the tiles and the type of piece that was on top of it. After gathering the game data, we had to make minor adjustments to some of the ranges and set a margin of error of +/- 0.01 V. The ranges for each piece are as follows:

Piece	Lower Limit (V)	Upper Limit (V)
White Pawn	2.74	2.77
Black Pawn	2.16	2.21
White Rook	2.95	3.01
Black Rook	1.96	2.01
White Knight	3.07	3.12
Black Knight	1.84	1.89
White Bishop	3.16	3.28
Black Bishop	1.69	1.74
White King	3.39	3.44
Black King	1.55	1.6
White Queen	3.33	3.37
Black Queen	1.63	1.73

Figure 10: Piece Voltage ranges

One of our major concerns was the interaction between different pieces in close proximity, due to a magnetic presence. Since we glued our magnets to the center of our base, for increased accuracy, and placed it at the bottom of a solid chess piece, the magnetic interactions do not affect gameplay or the other pieces on or off the board. Our current piece detection logic accurately identifies all pieces placed on or removed from a given tile, however, because of certain hardware issues with our Hall Effect Sensors, we are experiencing some issues when using the same voltage ranges across all 64 sensors, because of minor variations in the base voltages across the sensors.

7.3 Results for Board Legality

To ensure our users perform legal moves, we hope to use the Stockfish API to ensure the moves executed are valid. To test our script for legality checks, we have a testing script that simulates a game of chess with legal moves, but also sends random illegal moves to ensure our Legality Checker can catch all these errors. We were able to achieve a 100% accuracy on this system.

7.4 Results for Latency Test

Since our project involves several different hardware and software components, one of our biggest challenges was to keep latency as low as possible. After running several timing software and testing scripts on our different systems individual and as a whole we were able to get our overall latency to at a maximum of 1 second. The other latency results are shown below:

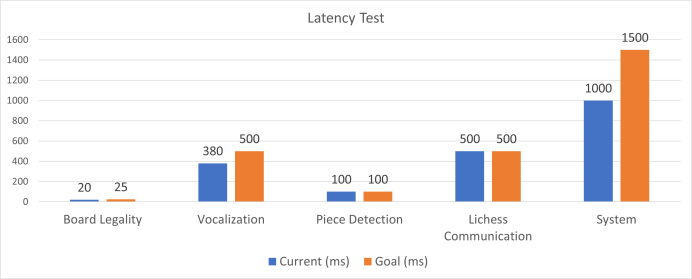


Figure 11: Latency Ranges

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule in Fig. 12 is organized by subsystems to account for members' strengths and weaknesses. The schedule has been modified a couple of times to account for shipment delays, and technical challenges faced during the process (Eg: PCB Fabrication, RPi-Arduino communication, 3D modeling, etc.) We also factored in various breaks during the semester to ensure we can complete our project on time. Finally, we have left the last 2 weeks for system integration and testing, to ensure optimal performance.

8.2 Team Member Responsibilities

Each member has been focused on specific portions of the project as well as helping the other members in their tasks as well. This allows us to maximize each of our members' strengths.

8.2.1 Edison

Web-App Development, U.I Designer, Software Systems, PCB Fabrication and Testing, Sensor Testing and Integration, Piece Detection System, Serial Communication between RPi and Arduino

8.2.2 Juan

PCB Design, Fabrication and testing (helped Edison), and laser cutting

8.2.3 Mukundh

PCB Fabrication and Testing (helped Edison), Board Accuracy, Integrity and Move Validation, Unit Testing for Subsystems, CAD design of board and pieces, Braille Communication, Sensor Testing and Integration, Piece Detection System, Serial Communication between RPi and Arduino, Speaker and Buttons Systems

Table 1: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
16GB Micro SD card Class 4	16GB	SanDisk	1	$6.97	$6.97
Radiometric Hall Effect Sensors	DRV5055Z4QDBZR	Texas Instruments	85	$1.30	$110.19
Radiometric Hall Effect Sensors	DRV5055A4ELPGMQ1	Texas Instruments	5	$3.66	$18.30
8:1 Analog multiplexer	TMUX1208PWR	Texas Instruments	19	$1.33	$25.36
3.3 K Ohms Resistors	EFR-W0D50-A:MF	EDGELEC	100	$0.06	$5.99
SMT Breakout PCB	1207	Adafruit	3	$6.00	$18.00
Lead Free Solder Paste 30 grams	SP-Sn42Bi58	Wonderway	2	$14.99	$30.00
5v Linear Voltage Regulator	MC7805ABTG	onsemi	12	$1.40	$16.84
Custom PCB with original pad sizes		JLCPCB	5	$9.94	$49.69
Custom PCB with bigger pads and Stencil		JLCPCB	10	$9.59	$95.92
0.01uF Ceramic Disk Capacitors	B-0004-H15	E-Projects	50	$0.22	$10.96
Strong Neodymium Magnets	SRM-2	THCMagorilla	50	$0	$18.01
N42 Neodymium Disc Magnet	42DNE3208-NI	MagnetShop	20	$4.87	$97.36
12 x 24 x 1/4 inches Plywood		Ideate	5	$3.00	$15.00
Filament for F170	333-60304	TechSpark	284	$0.47	$133.48
Proto Boards		Ideate	5	$0	$0
0.01uF Ceramic Disk Capacitors	B-0004-H15	Ideate	30	$0	$0
Raspberry Pi 4 Model-B 4GB	B 4GB	Raspberry Pi	1	$0	$0
Push Buttons	Red	Arduino	3	$0	$0
Analog Speaker		Ideate	2	$0	$0
Arduino Uno		Ardunio	1	$0	$0
					$652.07

8.3 Bill of Materials and Budget

Our total cost is an estimated value based on our estimate of 3D printing costs in Techspark. The majority of our budget was spent on 3D printing, PCB Fabrication, and our strong magnets (42DNE3208-NI). To see the full BOM you can refer to Table 1.

8.4 Risk Management

During the development process, we encountered several technical challenges and logistical challenges. We were able to deal with all these challenges individually and in a timely manner.

8.4.1 3D Modeling

No one on the team has any 3D modeling or printing experience. To ensure we were able to design custom pieces that fit our requirements, Mukundh was able to learn the basics of Rhino well in advance, to design and print our pieces.

8.4.2 PCB Fabrication

Our team has no experience designing or fabricating PCBs. Hence, Juan was able to learn some PCB design software tools like AutoDesk and EasyEDA. Edison attended training on how to use the soldering ovens in the PCB fabrication lab, which helped us fabricate our surface-mount components.

8.4.3 RPi & Arduino Uno Communication

Prior to this project, no one on our team has worked with a Raspberry Pi. So Mukundh was able to help setup the RPi and ensure our systems worked on the RPi as well as our local systems, by installing the needed dependencies. Establishing quick, uninterrupted communication between our RPi and Arduino, Mukundh and Edison were able to implement a quick and seamless communication using Serial Communication (PySerial).

8.4.4 Piece Detection

When testing our sensors and varying the magnetic strengths for each unique piece, we realized our hardware components had small margins of errors, which affected our program in subsequent systems. Hence, Mukundh and Edison were able to change their existing piece detection logic to account for marginal errors with the Hall Effect Sensors.

8.4.5 Budget and Logistical Challenges

We were short on our remaining budget and needed to pivot from some ideas like using batteries, which resulted in some system design changes highlighted in 5. We also experienced several logistical issues with shipping companies, which lead to several major delays in the arrival of our parts.

8.4.6 Team Challenges

Our team members were involved with other organizations on campus, which did take up more time than expected. We were able to talk to our teammates immediately and express our concerns and ensure we were able to get back on track during breaks like Carnival/Spring Break.

9 ETHICAL ISSUES

Our project seeks to primarily add a solution that would allow blind people to play online chess using a physical board. A secondary approach to our board is a teaching tool for novice and intermediate chess players to allow them to visualize moves and learn chess coordinates on a physical board while playing against online opponents. The two primary customer bases would be visually impaired people and beginner/novice chess players.

In order for our project to meet the needs of our users, it needs to be able to detect the type of piece, the color, and the location of all active pieces at all times with 100% accuracy and with a maximum latency of 1 second. It also needs to be able to give clear and loud audible cues of the state of the game. If our board doesn't meet those requirements then our primary customer base would be the most at risk. The worst thing that a user could do with our product is, play an incorrect game of chess by accidentally moving the wrong piece or changing the state of the board too fast, thereby causing a technical issue. Our blind users are trusting our product to translate their actions on a game of chess to the online platform. If we lose the trust of users because of a fault in our requirements then we lose our primary customer base. Our secondary customer base would be less affected because they could verify on a computer screen if the move that they made is reflected in the online game and if the audible cue given is actually the move that the opponent made in the online game, however, if they are still learning the rules of the game then they still might not catch the errors.

At the moment we don't collect any personal data on the user as for every game that is played on our board, we create a unique user id and game id that is associated with each board. The only data that we collect is the output voltage of all 64 hall effect sensors when a game is being played.

Tactile Chess is a rather safe product in terms of the overall health of our target consumer. The biggest health hazard that our product could pose is that our pieces could be a choking hazard if the chess board is set up around small children. This would force us to provide the appropriate age range for consumers and have proper warnings when marketing this product. We plan to address this by making the size of our pieces as big as possible without making the pieces too cumbersome for the chess player. Since our product has a lot of wires connecting the PCBs together, within a wooden encasing, this could cause a potential fire hazard if any electrical component begins to smoke.

Our project has a low impact on public safety. Since our product involves a single player playing chess online, there is a low chance of a physical argument occurring between the two players. Since it involves an online platform, we also create a platform for players to play online chess on a physical board if there is ever a global pandemic that requires global quarantine. Our board doesn't collect any personal data on the user so there is a low risk of privacy infringement.

Our project seeks to bring two communities together, the visually impaired and the online chess community. Using our product we want to give the visually impaired the most seamless transition to play chess on an online platform through a physical board. We will be extremely mindful of the accessibility of our board by implementing tactile and audible cues throughout the use of our board. We plan to have braille engravings throughout each piece and on the board to identify the coordinates of the board.

10 RELATED WORK

One of the most similar projects/products that we have found is a product called Phantom Chess [7].

Phantom Chess is an automatic chess board that allows users to play online chess on a physical board. It achieves this by using 500 hall effect sensors to sense where the pieces are on the board, and it uses a robotic arm on the inside of the board that moves the pieces throughout the board to reflect the moves made by the user and opponent on lichess.org. Not only is this option over our budget, but it also is expensive, and not accessible to blind users.

11 SUMMARY

The Tactile Chess system is an all-in-one system that provides accessibility to play online chess on a physical board for the visually impaired community. The system provides users braille identification on the pieces and board, vocal feedback and queues of their own and their opponent's moves, legality checking, access to their past game data, and a learning platform through puzzles for beginner chess players. Players will be able to play at their own pace as competitively as they want with the speed of the board matching even the most experienced of chess players.

11.1 Future Work

A research group from the University of Pittsburgh has reached out to us in order to potentially take this project ahead. Since Mukundh and Edison will be Masters students at CMU for the academic year 2023-24, if they receive funding and financial support, they will carry this project forward.

11.2 Lessons Learned

Our advice to students wanting to tackle this application in the future would be to find a way to turn the product into a smaller form factor or to use fewer sensors as that would help bring costs down. This product can potentially change some people's lives for the better so the lower the price point the better.

Glossary of Acronyms

- ADC - Analog-to-Digital Converter
- API - Application Programming Interface
- GPIO - General Purpose Input/Output
- HE - Hall Effect
- MUX - Multiplexer
- PCB - Printed Circuit Board
- RPi – Raspberry Pi
- UART - Universal Asynchronous Receiver-Transmitter

References

[1] Chess.com. "Chess.com Reaches 100 Million Members!" In: *Chess.com* (Dec. 2022).

[2] "https://github.com/Clariity/react-chessboardreadme". In: ().

[3] "https://github.com/jhlywa/chess.js/blob/master/README.md". In: ().

[4] "https://lichess.org". In: (2010).

[5] "https://lichess.org/forum/lichess-feedback/connection-speed-in-bullet-tournaments". In: *Connection Speed in Tournaments* (2018).

[6] "https://stockfishchess.org/". In: ().

[7] Wonder Substances. "https://www.indiegogo.com/projects/phantom-the-robotic-chessboard-made-of-real-wood/". In: *Phantom Chessboard* (2021).

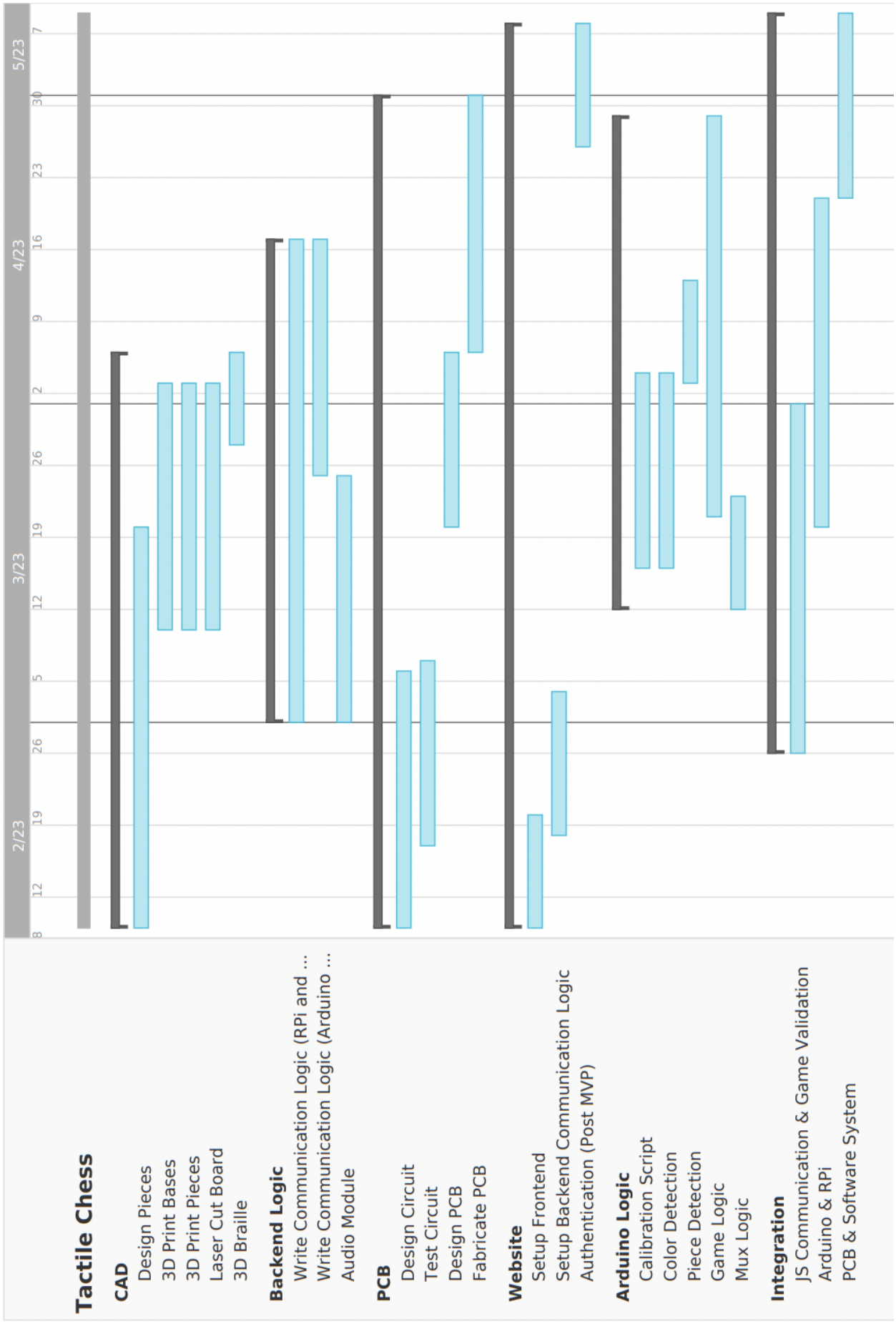


Figure 12: Gantt Chart