https://www.overleaf.com/project/63fd2a7afd12d57ea9958f51

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 aims to combine vocal and tactile cues to provide our blind users with the opportunity to play online chess through a custom-made chess board. This one-of-a-kind system will be capable of providing feedback based on moves made within 300ms and retrieving information from a public chess server within 500ms of an opponent making a move.**

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

1 INTRODUCTION

With the recent growth in the number of online chess users, 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 made chess-board, with accessible features for blind users, which can connect to an online chess platform, like *lichess.org*. 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.

For the 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 on the pieces. This will help them differentiate pieces and locate them when vocal cues are provided. The board will differentiate black and white tiles by raising one of the colors and lowering the other. 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 pieces and a hole in every tile. To make the game play seamless, the user will have to press a button on the board to start an online chess game, if the board is set up correctly.

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.

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. Furthermore, such boards are not accessible to blind users.

2 USE-CASE REQUIREMENTS

Our requirements are mostly focused on accessibility features and user experience.

- The system will be able to distinguish black and white pieces with 100% accuracy and distinguish between specific game pieces with 95% accuracy. In order to guarantee the integrity of our board state we have to be able to accurately distinguish pieces on the board. We believe we can distinguish between white and black pieces fairly easily by varying magnet polarity for each color. For individual game pieces, we will vary magnetic strength slightly by using varying magnet strengths, sizes, and distances from the Hall-Effect sensors.

- The board will take 300ms to give user vocal feedback based on a user's move. In order to have a seamless experience, we have set a goal of 300ms since this is the average human reaction time.

- Once an opponent has made a move, the board will vocalize the move for the user to then replicate on the board. The board will be able to do this in 500ms, the reason for this is *lichess.org*, the public source chess API we are using, has a latency time of 500ms.

- The system will provide at least 4 hours of battery life. The average chess game lasts anywhere from 10-60 minutes and the average chess player plays 2 games per day. Considering our stakeholders are either blind users or beginners, we expect the game to take more than the average length of a chess game. The average player plays chess for at most 2 hours per day - by providing 4 hours of battery life, the user will be able to use the system for longer without charging it.

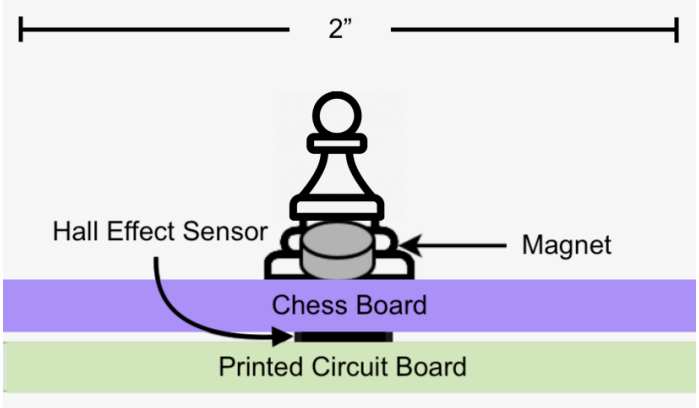
- Users will be able to finish setting up their device, which involves connecting to an online chess game, within 60 seconds. We will achieve this by having a push-button mechanism to help start an online game without too many steps.

- The board will be made accessible. We will achieve this by creating a custom chess board and chess pieces that use braille notation to allow users to easily identify a current piece and tile coordinates. To help blind users differentiate between black and white pieces, one of the colors will be raised and the other will be lowered. We will also have a speaker on the board to help provide tactile cues.

3 ARCHITECTURE AND/OR PRINCIPLE OF OPERATION

3.1 Chess Board & Pieces

We will create a custom chessboard and game pieces using the Rhino 3D modeling software in order to incorporate the distinct cues used in typical blind chess. These cues include the varying heights of the tile based on its color, braille engraving for tile coordinates, braille engravings for each chess piece, and a key and lock mechanism between chess pieces and each tile. The pieces and board will be made using plastic since this will guarantee minimum interference with the magnetic fields being measured. Below the chessboard, we will mount a custom printed circuit board that will contain all of the circuits and electronics needed to add to help simulate the physical game, on the online platform. Each unique game piece will contain a magnet that will vary in size, polarity, and shape based on the color and piece type - this aims to trigger the linear hall effect sensors embedded in the PCB, by producing varying voltages, which will help us distinguish pieces.



(a)

Figure 1: 3D-modeled chess set

3.2 Hardware

3.2.1 Printed Circuit Board

To help in sensing the pieces, we will have a PCB underneath the 3D-printed chessboard. The PCB will house our entire circuitry to detect varying magnetic fields with the Hall Effect Sensors. Each tile will have one sensor, resulting in 64 total sensors. We plan to group these sensors into 8 groups of 8 to allow for convenient 8:1 multiplexing. The PCB will also have holders for batteries and power regulation. This will allow the board to not be tethered or connected to a wall outlet. The output of the multiplexers will then be connected to an 8 channel analog to digital converter which will feed digital signals to the Raspberry Pi.

3.2.2 Raspberry Pi

The Raspberry Pi will help update and store the board state. The RPi will also serve as a central hub for data to and from the website. The bulk of the move validation and legality checks will be done here. The RPi will also communicate with *lichess.org*, to update moves made by the user, and vocalize moves made by the opponent.

3.3 Components on Chessboard

We plan to have LEDs on the chessboard, to help our beginner and novice users play with ease - this will be integrated post MVP. The chessboard will also have a speaker to help provide vocal cues to our blind users, and also give our users feedback and updates regarding the game. Finally, the board will contain several buttons to connect to *lichess.org* and perform certain actions, such as starting or ending a game.

3.4 Software

3.4.1 Legality Check

In order to guarantee the communication with *lichess.org* is minimized - given the API's latency, we will be running state and move checks locally before communicating the information to the API and web application. In order to perform move and state validation we will be using the Stockfish API which will allow us to retrieve the current game state (FEN Notation) via HTTP GET requests to *lichess.org* and create a local board state through Stockfish using that board state (FEN Notations). The Stockfish API is an open-source stockfish class, 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 300ms.

3.4.2 Web Application

Our custom made web application will be deployed on an EC2 AWS server to allow user's to view their live games,

match history, and solve chess puzzles. The website's frontend will be coded in reactjs, while the backend will be coded in nodejs. Through HTTP requests to the public *lichess.org* endpoint, the frontend will be able to GET and POST information about a user's live games and moves to then display on the web application. The website's main focus is to serve as a central information hub for the user to interact with past games or view their moves live from *lichess.org*. This aspect of our system is mainly focused towards beginner players that will also be using our product.

4 DESIGN REQUIREMENTS

The following design requirements are related to the use-case requirements that are outlined in Section 2. In order to allow users the seamless transition from the physical board to the online chess game, the board should distinguish the piece color with 100% accuracy and the type of piece with 95% accuracy. In order to achieve this, the sensors must be able to distinguish first the presence of a white piece, a black piece, or no piece at all; then they must distinguish whether its a pawn, knight, rook, bishop, queen, or a king. The software needs to achieve 100% accuracy in determining which piece was moved based on the sensor output, as well as having a 100% accuracy in determining the legality of moves based on a local system checker. The legality check will ensure users don't make an illegal move when playing against an A.I. or an online opponent.

Our system must take 1500ms or less to validate the user's move and reflect the move on the online game or provide audio feedback to the user, and then vocalize the opponent's move through the board. This process goes through multiple subsystems and so the 500ms is the bound given to us by *lichess.org* for the speed that a user can make a move using their API and the move reflecting on their platform. To ensure we have minimal latency experienced by our users, we have set realistic latency goals for each of our subsystems. The latency goals and steps are: receive the user button press input (20ms), collect sensor output data (40ms), convert analog outputs to digital outputs (50ms), check the legality of the move (50ms), record the move to store in software (20ms), send the correct move to *lichess.org* through our Web-App (500ms), receive the opponents move on our Web-App (500ms), send the move data to the board (30ms), and vocalizing the piece type and the move start and finish based on the standard chess annotations (30ms). The latency goals are a bit on the higher side for some of the steps, however, our latency goal of 1500 ms gives us wiggle room. This would allow us to account for any communication delay between subsystems or any delays in vocalizing feedback.

The entire system should have enough battery power to last a user 4 hours on a single charge. The average chess game can last anywhere between 10 to 60 minutes and the average chess player plays 2 games a day. This would equals about 2 hours of playtime on the high end of the bounds. Considering our stakeholders are visually impaired or beginners, we estimate their average playtime for their games will be higher than normal. Taking into account the 2-hour-per-day playtime, we want our users to be able to play more than one game without worrying about charging the device. We are limiting the system to 4 hours because this keeps the form factor of the board relatively small for the convenience of the user.

Lastly, geared more towards our beginner chess player users, we want our Web-App to display users past match history, win-loss ratio, and allow users to learn chess strategy through chess puzzles. The users will be able to complete the chess puzzles using the board and the system will provide audio feedback to help them learn and complete the puzzles. Our physical board and pieces will have braille engraved to help our visually impaired users identify the piece type and location on the board. Since our board will have buttons embedded into the board, there will be braille engraved above each button to identify their function.

5 DESIGN TRADE STUDIES

5.1 Sensors

For our sensors, we chose to use Hall Effect Sensors to help detect a magnetic field. We chose to use the Texas Instruments Hall Effect Sensors. We needed to choose between unipolar or 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 since 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. So we plan to use varying magnetic fields for each unique piece, to help identify pieces and maintain board integrity.

5.2 Analog to Digital Converter

We chose to use an 8-bit, 8-channel Analog-to-Digital Converter (ADC) because we wanted an efficient solution to sample 8 multiplexers. We chose an 8-bit ADC to avoid loss of resolution, and reducing the number of bits doesn't give us significant cost gains.

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 columns of 8, we chose to use an 8:1 multiplexer, and an 8-channel ADC. This architecture will help optimize our cost with modularity and increased efficiency.

5.4 PCB Fabrication

For our PCB, we will be having a modular approach to help fabricate a circuit board for one tile (2"x2") and

replicating it for the other tiles. The dimensions of our board will be 16"x16" (without the external board components). We plan to fabricate the entire board in sections of columns, instead of one single PCB to minimize cost. Each of the columns will have an 8:1 multiplexer, the output will then be fed into an 8-channel ADC, to send the information over to our Raspberry Pi.

5.5 Batteries

To achieve our desired performance and expected battery life of 4 hours, we have to size our batteries accordingly without sacrifice the size of the system. We need a compact solution that has not only enough power to keep our system powered, but also enough connectivity to power our hardware subsystems. Lithium-ion cells satisfy all these requirements better than lead-acid, alkaline, or lithium-iron-phosphate batteries. We will be using a battery pack that has multiple output types, such as USB, USB-C, and a DC5521 connector. In order to achieve a smaller form factor we have experience removing the 18650 cells from the encasing of the battery pack and putting them into a cell holder that we can solder directly into the PCB as an option.

5.6 Raspberry Pi

For our given system, we chose the Raspberry Pi 4 Model 4 4GB. One of the main reasons we chose this model RPi was for the connectivity that it offered. The 40 pin GPIO header would allow us to communicate with the signal from the AC DC converter. 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 speaker and other peripherals that we might need. The 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless would allow the RPi to connect to our Web-App. The 5V DC via USB-C connector is compatible with our battery pack for the system. We are also going to supply the RPi with a 16GB micro-SD card to store the OS, our python scripts, and 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 need to know the initial and final positions of the moved piece, and make sure the piece has been moved in its allowed pattern (Bishop can only move diagonally, Rook can only move in straight lines, etc.). We could also generate a list of legal moves and compare it to the move made. 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 ove 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 analyse 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 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.

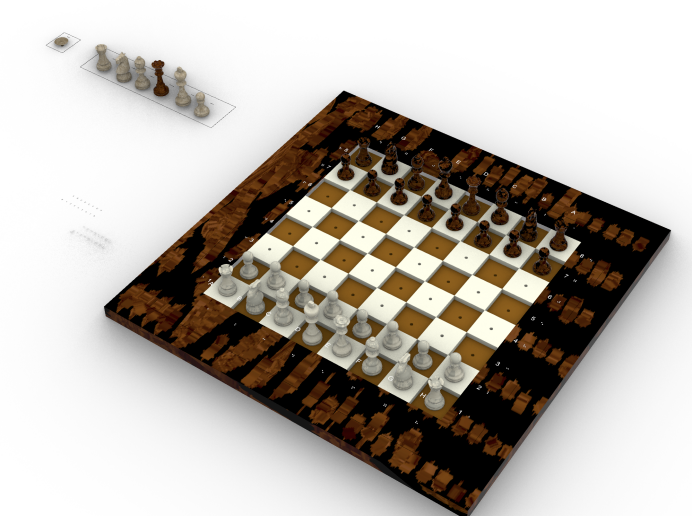


Figure 2: 3D-modeled chess set

6 SYSTEM IMPLEMENTATION

6.1 Chessboard & Pieces

Our custom chessboard is broken down into two layers, the top layer will consist of a custom 3D printed blind-friendly chessboard surface - where the user will interact with game pieces, and the bottom layer will consist of a PCB which will be 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 will consist of 64 hall effect sensors spaced out by 2 inches between each sensor to allow for a standard chessboard size. Game pieces will also be 3D printed and made blind-friendly. In order to achieve blind-friendly game pieces we designed a key and lock mechanism between the pieces and board while also engraving braille notation on each individual piece to help the user identify between different pieces and piece colors. Inside each of the pieces, there will be a magnet, which will vary in size, polarity, and shape based on the game piece and piece color. The PCB will also consist of several LEDs, these LEDs will be provided power and utility after the MVP checkpoint has been reached. The purpose of these LEDs is to add visual cues for beginner or novice chess players to easily navigate the board and receive game moves from their opponents.

6.2 Move Sensing

In order to facilitate move sensing we designed a PCB which will be installed in the bottom layer of our chessboard. The PCB will use the DRV5055 radiometric hall effect sensors from Texas Instruments to detect when pieces are placed on a tile. Furthermore, the sensors will 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 radiometric hall effect sensors spaced out by 2 inches. The outputs of the sensors in each row will have a dedicated 8:1 multiplexer, since there are 8 rows on a chessboard we will be using a total of 8 multiplexers (74HC4051D IC MUX 8:1 4 OHM 16SOIC 74HC4051D). The 8 multiplexers will feed their analog outputs to an 8-pin ADC (8-channel ADC TLA2518IRTER), which will then communicate the digital output to the Raspberry PI. The RPi will process the data sent by the sensors and run legality checks on the user's input before communicating the information to *lichess.org* and the web application. 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 move will be registered through our move sensing technology, after which the user will receive vocal feedback notifying them if the move was replicated correctly or not.

6.3 Hardware User Interface

The board's user interface will consist of a speaker and 3 buttons. The speaker will be used to provide vocal feedback about opponent moves and board and move validity. It will be powered by the Raspberry Pi and will be utilized by a python script in the RPi during different instances of a live game. Furthermore, the 3 buttons will have specific behaviors assigned to each of them. The first button will be used to start seeking a game with an online opponent,

the second will be used to start a game with an AI opponent, and the third will be used to resign a game or cancel a seek. In order to power these buttons and connect them to the Raspberry Pi we will use a basic switch circuit - see figure 3. Once the Raspberry Pi is notified of a request, we will use a Python script to case on the pressed button. 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. Once a game start event is received, our script will move to a game state in which user and opponent moves will be interpreted. In the case of an AI seek, the system will follow a similar flow as the regular online seek, however, the HTTP POST request being made will hit a different endpoint. As for the resign or cancel button, the script will first verify if a connection with lichess has been established and if a game start event has been received. If a game start event has been received, the system will then communicate with lichess through an HTTP POST request indicating the service that the user is resigning the game - if a game has not yet begun, the seek will be canceled.

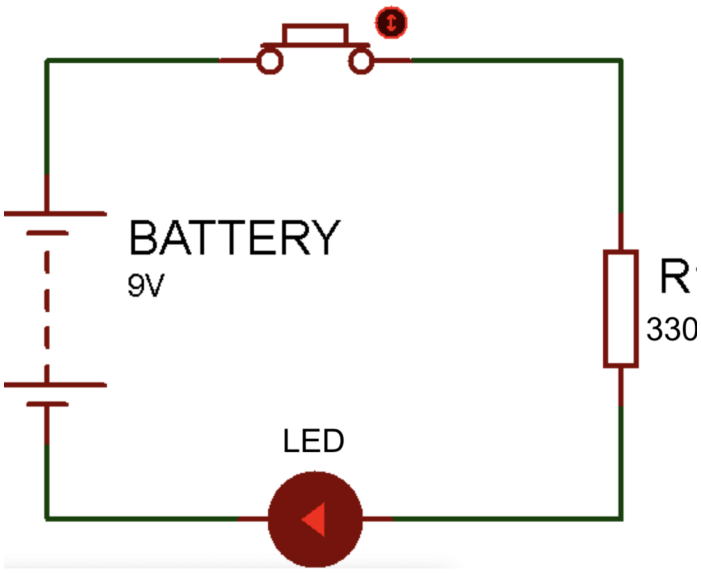


Figure 3: Basic push-button circuit

6.4 State Validation

State validation will be performed in the Raspberry Pi through a legality move checker script written in python. The python script will use a library called Stockfish to create a local stockfish instance, which is essentially a chessboard object, on which moves will be replicated. This object has certain attributes such as the current fen of the board and methods that can be used to make moves on that fen. The script will also be able to communicate directly with *lichess.org* by using different methods of HTTP requests. Once a game is initiated, a stream is started between lichess and the Raspberry Pi, furthermore a stockfish object is created. Once the player makes a move the object will read the input and attempt to replicate the move on the fen which represents the state of the board. If the move is valid, the script will then make an HTTP POST request to *lichess.org* with the move specifications - if the move is not valid, the user will receive vocal feedback about the move. Once the opponent has made a move, the script will detect it through the already established stream and notify the user of the move that must be replicated. At this point a second copy will be created of the stockfish instance - call this the expected board, which includes the opponents move. In order to verify that the user replicates that move correctly on the board, the script will then compare the result of the user replicating the move with the expected state. If they match, the game will proceed - if they are different the user will receive verbal feedback through the speakers on the board.

6.5 Web Application

The website frontend will use ReactJS and CSS in order to display different user information, as well as live games. The backend will use NodeJS and will be mainly used to communicate with *lichess.org* using different methods of HTTP requests. 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. We chose to write the backend in JavaScript to keep the coding language consistent through the full stack of the website. Furthermore, 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 will display the current live game by acquiring the current live game's id through a GET request from our backend to *lichess.org*. After identifying the game id, the page will make a second GET request to *lichess.org*, this time from our frontend, which will establish a stream between our web application and lichess. Through this stream lichess will be sending game updates including the current FEN and the last move made. With this information, the frontend will render the current live FEN using a library called react chessboard and will create a local chess state using Chess.js which will be linked to the rendered chessboard. Throughout the rest of the game, the frontend will update the chessboard state via the moves being streamed to the frontend by lichess. The NodeJS backend will mostly be used to make HTTP requests which don't require streams, such as getting a game id or authenticating users via *lichess.org*. The web application will be deployed on an EC2 server using AWS to allow users to view their match history or live games from any platform.

6.6 Power

In order to power our system we are going to be using a lithium-ion battery pack. The pack that we want to use can provide 5V at 12Ah or 12V at 6Ah DC output. Our Raspberry Pi will use 5W, therefore over a period of 4 hours, it will require 20 Watt-hours of power. In order to split the output of the battery pack and reduce the noise, we will have a linear regulator circuit for the PCB. The 12Ah provided by the battery pack will give us plenty of room to power our system.

7 TEST & VALIDATION

To help our testing plans, we hope to follow test-driven development, to ensure each subsystem is tested before integrating with the existing system. To do this, we plan to develop testing scripts for each subsystem, to test every function and the overall performance of each subsystem.

7.1 Tests for Accessibility

To ensure that our board caters to our target stakeholders, we plan to test our chessboard by going to the Blind & Vision Rehabilitation Services of Pittsburgh, to test if our board is blind-friendly. We also plan to test our board for beginner and novice users, to make sure the board caters to all of our target stakeholders.

7.2 Tests for Piece Detection

To test the accuracy of piece detection, we hope to test our sensors and magnets, by ensuring we have unique readings for each unique piece, and opposite values for different color pieces. To ensure we hit our accuracy goals, we plan to test piece detection by placing different pieces in different locations to ensure we are able to differentiate between them. We will also be simulating a blindfolded game of chess to help ensure if any pieces are knocked down, we can maintain the board's integrity by providing appropriate vocal cues, to reach the board's latest state. We hope to achieve a 100% accuracy in distinguishing colors and 95% accuracy in detecting each unique piece.

7.3 Tests for Legality Check

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 hope to have a testing script with simulated games and ensure our script catches all the expected errors and detects the legal and illegal moves.

7.4 Tests for Components on Board

To test the functionality of the button on the board to start the game, we hope to ensure that there is smooth connectivity between the board and *lichess.org*. For the speaker, we plan to use our testing script to raise errors and ensure the errors are being vocalized accurately.

7.5 Latency Tests

To ensure our system latency is within our desired goal, we plan to test every individual section and also the entire system for response time. We will test and record the amount of time taken for a piece to be detected. The next step will be to calculate the time taken for information to reach the Raspberry Pi and then be checked for legality. We plan to test this portion with our test script and record the time taken to provide feedback to the user or communicate the information to *lichess.org*. Our final test for latency will be to check the time taken for an opponent's move to be vocalized on our board. For all these latency tests, we either plan to use a minimum of 40-50 test cases, to help get an accurate average latency for the system, or simulate at least 2 games with 40-50 moves per color for each game.

7.6 Power Tests

To measure our power consumption, we plan to test our RPi and sensors at idle and at their peak consumption using a power supply instead of our batteries. We will then simulate at least one chess game, played by blind users. We will then test the state of charge of the batteries before and after the game, and extrapolate it to the amount of time/number of games that can be played. If we need to increase our battery capacity, we have space to add our batteries and using this information, we can choose to be more cost-effective with our battery consumption.

7.7 Usability Tests

To test the overall usability of the system, we hope to test it with two groups; the first one is our primary stakeholder, blind users, and our other stakeholders, beginner and novice chess players. We have a set of user tests we hope to perform, and collect feedback. For our blind users, we will ask them to play 3 games:

- One regular game with the chess AI (game starts with a button push by the blind user)
- One regular game with an online user (game starts with a button push by the blind user)
- One random ongoing game (simulate a chess game from the middle of gameplay)

For our other users, we hope to have them play a similar set of games:

- One regular game with the chess AI
- One regular game with an online user
- One random ongoing game (simulate a chess game from the middle of gameplay)

We will ask our users to play these games in this order, and then ask them a series of questions, that measure various components of usability and the test metrics:

- Latency:
  - *"Did you feel any noticeable lag between piece movement and feedback from the speaker?"*
- User Experience:
  - *"Was the board difficult to use, or difficult to start a game on?"*
- Accessibility:
  - *"Was the board easy to understand and use?"*
  - *"Were you able to play an entire game without any interruptions?"*
  - *"Was the board intuitive to use?"*

The goal of the above set of questions is to understand if the system we built has helped solve our initial problem statement, without changing the usability and experience of a regular chess game. We would like to see if our product's accessible features, and training features for beginner users, achieve their expected functionalities without affecting the experience of the game of chess. We hope to test at least 6 different users (3 blind users and 3 beginner/novice chess players). We hope to receive a standard response of "No" for Latency and User Experience, and a "Yes" for the Accessibility section (at most 1 "No").

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule in Fig. 4 is organized by subsystems to accommodate for each members strengths and weakness.Most of the and deadlines can be done individually, however later down the pipeline it becomes crucial to complete the integration steps in series. We took into account the first week of March for Spring Break. Depending on how the subsystems are progressing, it is up to the team member in charge of it to mitigate how they will make sure deadlines are met on time. We also made sure allocate enough time for integration testing and user testing for the system as a whole.

8.2 Team Member Responsibilities

Each of our members have both primary and secondary responsibilities. There are some sections that overlap, however this allows us to maximize each of our members' strengths.

8.2.1 Edison

Web-App Development, Project Management, U.I Designer, Raspberry Pi to Web-App communication, Legality Checks

8.2.2 Juan

PCB Design and Fabrication, Sensor Testing and Integration, 3D printing and laser cutting

8.2.3 Mukundh

PCB Design and Fabrication, Board Accuracy and Integrity, Legality Checks, Web-App Development, Unit Testing for Subsystems, CAD design of board and pieces, and Braille Communication

8.3 Bill of Materials and Budget

A large portion of our budget is going towards hall effect sensors and 3D printing. We have been using the printers and laser cutters in TechSpark. We have also managed to save a good amount of money for components from personal supplies and the Raspberry Pi through donations from the ECE department. To see the full BOM you can refer to Table 1.

8.4 Risk Mitigation Plans

One of the main risks that we must mitigate is that no one in our team has any experience designing and fabricating a PCB. In order to deal with this challenge, Mukundh and Juan will research and learn how to use AutoDesk in order to design the PCB. Fortunately for the purposes of our project we will only need to design a single layer PCB as well as having several manufacturing options at our disposal.

Our second main risk is that no one on our team has experience using radiometric hall effect sensors. In order to figure out how the strength of certain magnets affects the output of the radiometric hall effect sensors that we order from Texas Instruments and set up a simple circuit on a breadboard. We will then connect the analog output of the sensors to an 8:1 Analog multiplexer and then to our Raspberry Pi in order to visualize the differences in the magnetic fields create by our 12 distinct magnets.

Our third main risk is on the software side. No one in our group has worked with communicating a Raspberry Pi with our Web-App, as well as running a custom python script on Raspberry OS. In order to overcome this challenge, Mukundh and Edison have been using a python script to simulate the physical board inputs and sending the board information to our Web-App. The unknown comes in the latency from communication from the Raspberry Pi all the way to lichess.org. In order to keep this value as low as possible, the goal is to keep all legality and game accuracy checks locally on Raspberry Pi.

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	5	$3.18	$15.90
Radiometric Hall Effect Sensors	DRV5055A4ELPGMQ1	Texas Instruments	5	$3.40	$16.98
Raspberry Pi 4 Model-B 4GB	B 4GB	Raspberry Pi	1	$0	$0
Filament for F170	333-60304	TechSpark	8.520	$1.24	$10.56
Talentcell Lithium ion Battery Pack		Talentcell	1	$39.99	$39.99
Radiometric Hall Effect Sensors	DRV5055Z4QDBZR	Texas Instruments	64	$1.19	$76.07
Push Buttons	Red	Arduino	4	$0	$0
IC MUX 8:1 4 OHM 16SOIC	74HC4051D	Toshiba	9	$1.30	$11.71
8-channel ADC	TLA2518IRTER	Texas Instruments	3	$4.63	$13.89
Analog Speaker		Ideate	1	$0	$0
LEDs		Ideate	1	$0	$0
Custom PCB		Seeed Fusion Quick Turn	8	$22.98	$183.86
					$375.93

9 RELATED WORK

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

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.

10 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 opponents 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 competitive as they want with the speed of the board matching even the most experienced of chess players. Foreseeable challenges in the remaining time of the project would be PCB validation, identifying the correct magnetic strength for all 12 pieces, laser cutting and 3D printing the housing for the entire hardware system, and doing user testing. We are confident that we will overcome these challenges and meet our use-case and design requirements to deliver a high quality product.

Glossary of Acronyms

- ADC - Analog-to-Digital Converter
- API - Application Programming Interface
- HE - Hall Effect
- PCB - Printed Circuit Board
- RPi – Raspberry Pi

References

- Zhelyabuzhsky, I. (2023, February 28). Stockfish. GitHub. https://github.com/zhelyabuzhsky/stockfish

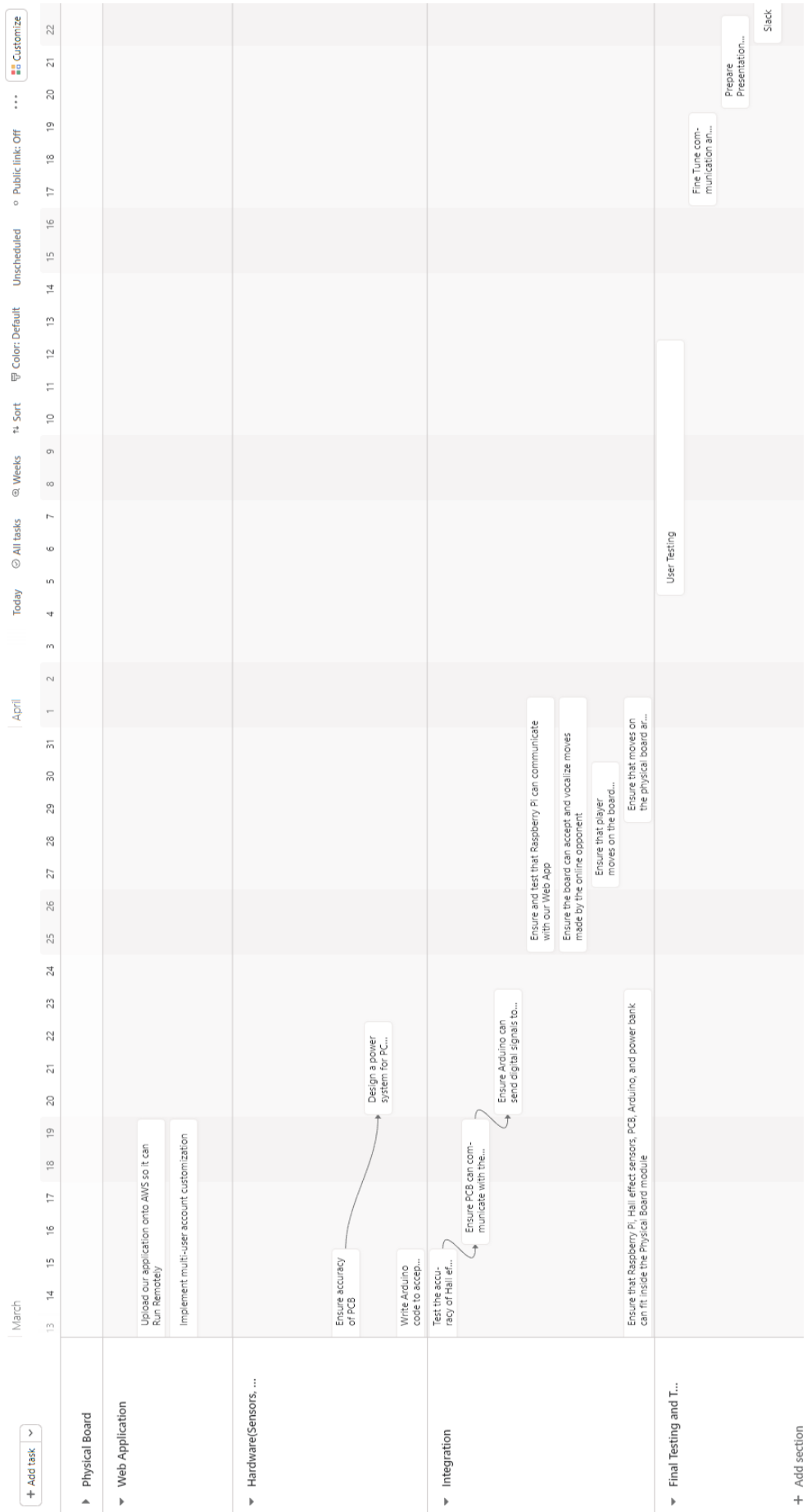


Figure 4: Gantt Chart