Seamless Autonotator

Authors: Ryan Guan, Patrick Joyce, Vikram Marmer
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of automatically logging notation during a game of chess. This is an improvement on the state of the art since notation is still largely done by hand or not done at all by more casual players as well as tournament level players. The seamless notation is done faster than human reaction speed, which is much faster than a human could perform notation manually. It is also 100% accurate, so it is slightly more accurate than human notation which may have some errors.**

***Index Terms*—Chess, Hall Effect, Sensor, Move Generation, Multiplexing**

1 INTRODUCTION

Our use case is to serve as an all-in-one chess set (board and pieces) that automatically notates moves, which can be used by casual and professional chess players alike. It will also record the moves played in a database and send moves to a website where the games can be viewed.

For professional chess players, the system will be able to be used in a tournament setting. It has a 10 hour battery life, long enough for a day of tournament use. The board does not require externals such as a camera, so it can be easily transported and set up for play. It will remove the need for professional chess players to notate their games, freeing up some time during the game and removing the possibility of any notation errors due to poor handwriting or human error.

For casual players, the main benefit is providing the notation itself. Many casual players do not bother to notate their games, as it is an extra layer of work that is not enjoyable in and of itself. It also requires knowledge of algebraic notation, the standard method of recording chess moves, which beginners may have difficulty understanding. Automatically notating games and storing them in a database will provide casual players the opportunity to review games that they otherwise would have forgotten. If at the end of the game they think, "I wonder if I made the right move with my knight on turn 10," they can find the game in our database and export it to a chess engine or website, such as Chess.com, to analyze it. In this way, casual players using our system will be able to learn more and improve their understanding of chess.

A competing technology is the new product ChessUp [1]. It is also a chess board that can detect moves on it, record games and store them in a database, and find legal moves for the current board state. It also has additional features, such as an automatic chess clock that switches when the board detects a piece has moved, and the ability to play online or against an AI by highlighting the opponent's moves on the board. The main advantage of our system over ChessUp is cost - ChessUp is $400, while our prototype alone will likely cost under $500. Another advantage of our system is that it aims to be usable for professional play in tournaments. The ChessUp board features lights under the squares and built-in AI assistance, which means it will not be approved for tournament use, so our products have different markets.

2 USE-CASE REQUIREMENTS

Our requirements are centered around the user experience while using the system:

- The system will record notation with 100% accuracy. If the system does not perfectly record notation, it will not be viewed as an improvement to the current standard of human notation in professional play. To accomplish this, the system will check the legality of each move to ensure that illegal moves are not recorded. Illegal moves will be highlighted to the user so they can replay the move correctly.

- The system will take 300ms or less from the time a move is made (signaled by a button pressed on the board) to the system's response of the move's legality (with red or green LEDs on the board). This is around human reaction time, so the system will not be so slow that it will noticeably impede the pace of the user's game.

- The system will provide 10 hours of battery life on a single charge. This is long enough for a full day of tournament use between charges.

- The system will store the most recent 10 games for each user. Users will be able to access these games with an account on the system's website.

- The system will translate completed games into exportable PGNs, which can be downloaded or copied from the website and imported into chess engines or websites such as Chess.com for analysis.

3 ARCHITECTURE AND PRINCIPLE OF OPERATION

3.1 Chess Board

We will have a chess board style top layer for our design made from acrylic. This will have a chess board pattern engraved on it so the user can play a regular chess game on it. Underneath the top layer, we will have a printed circuit board that contains all of our in-board electronics. Fig. 1 demonstrates the layer stackup of our design. The realization of this board is shown in Fig. 2

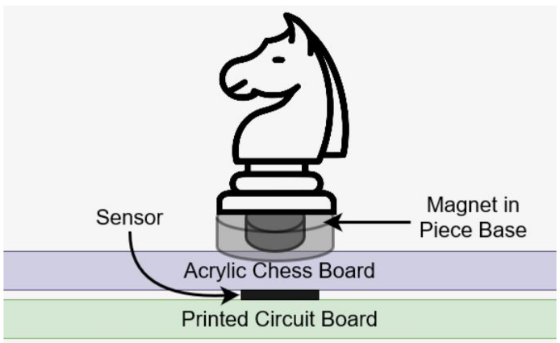

Figure 1: A cross section of the chess board, showing the PCB and sensor beneath the physical board.

Figure 2: An example of a modified chess piece with a 3D printed holder for the magnet underneath

3.2 Sensing

In order for our autonotation system to function, our chess board system will need to sense the color and location of every piece on the board. To accomplish this, all of our chess pieces will have magnets in the base. White pieces will have the magnet oriented in one direction while the black pieces will have the magnet flipped to create the opposite polarity of magnetic field. The difference in magnetic field will be sensed by Hall effect sensors within the chess board.

3.3 Hardware

Fig. 3 demonstrates our hardware structure. Since the chess board must sense magnetic fields, there will be a printed circuit board underneath the top layer that houses Hall effect sensors that can sense magnetic fields. This PCB will have one sensor per square, yielding 64 total sensors for the chess board. These sensors will be grouped in 8 sets of 8 to allow for convenient 8:1 multiplexing for each column of the board into an ADC. This ADC will communicate with an RPi so the RPi can gather the board state. In addition to the sensing, this PCB will also contain holders for batteries, power regulation, and the user interface. The power regulation and batteries will allow the board to not be tethered to an outlet as well as last a full day of tournament play per our requirements. The final aspect of the hardware is the user interface. This consists of a button and two LEDs for each player. These buttons and LEDs are connected to the RPi to allow the user to interact with the system.

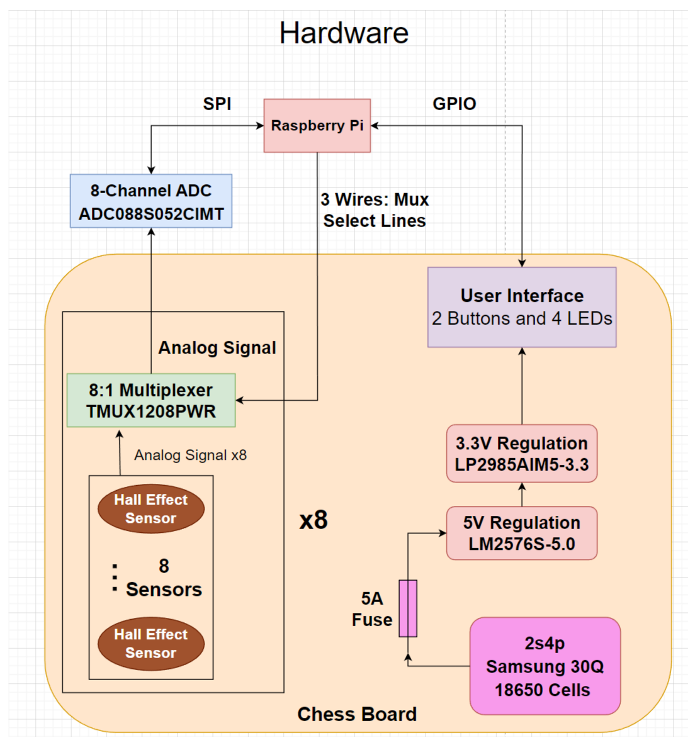

Figure 3: Hardware Block Diagram

From a hardware perspective, the final product did not change from what was presented in the design report. The final implementation is shown in Fig. 4. A photo of the PCBs without the acrylic is shown in Fig. 5. This shows the sensors, multiplexers, and ADC.

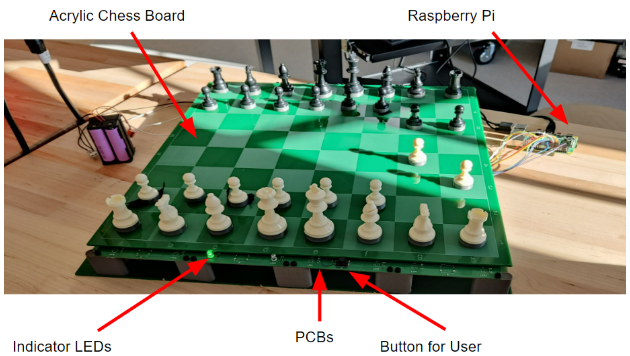

Figure 4: Annotated Final Hardware

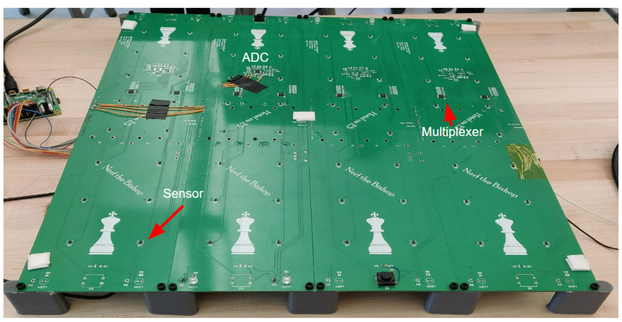

Figure 5: Annotated Final PCBs

3.4 Legality Check

A legality check program will run on the RPi. The program will keep track of the current board state, and when a player makes a move, it will check if the move is legal for the current board state. If the move is legal, a green LED will light up on the board to tell the players that the move is legal and it is the next player's turn. If the move is illegal, a red LED will light up to tell the players that the move is illegal so the same player must make a different move. Legal moves will be forwarded to the website and database backend program to record the game.

3.5 Server, Website, and Database

The autonotator website will have features such as individual user login (and registration), the ability to view current games, and the ability to view a user's past games. Furthermore, users should be able to upload their past games to websites such as Chess.com for further analysis. The website will store all data inside of the database, which will hold the user login data for each user and the notation of up to previous games. Altogether, the website will consistently refresh to show the user notation updates that are occurring in their current chess game (which will send data to the website). @ Ryan for final software implementation and how it changed from the design report

4 DESIGN REQUIREMENTS

The following design requirements are related to the use-case requirements that are outlined in Section 2. The system should record notation with 100% accuracy, so the sensors must be able to differentiate between the presence of a white piece, black piece, or no piece on each square with 100% accuracy. The software must be able to determine which piece was moved with 100% accuracy based on the sensor output. The software must also be able to determine the legality of the move played with 100% accuracy to ensure that illegal moves are not recorded.

The system must take 300ms or less from the time a move is made to the system's response of the move's legality. This process involves multiple subsystems and therefore the 300ms requirement is a limit on the sum of the latency of several steps in the process. To ensure that the entire process takes 300ms or less, we created latency requirements for each step:

- Receiving the user button press input in firmware (20ms).
- Collecting the sensor data (30ms).
- Translating the sensor data into a move in software (40ms).
- Checking the legality of that move in software (30ms).
- Returning the result of that legality check by illuminating a green LED or a red LED (20ms).

The latency requirements for each step are quite pessimistic, and even then they only sum to 140ms in total. If the requirement for any individual step cannot be met due to some unforeseen factor that we failed to consider when creating these requirements, the process would still likely take under 300ms because there is 160ms of slack.

The system should have enough battery power to function for 10 hours on a single charge. We would like our system to be useful in a tournament setting in addition to a casual setting. A day of tournament play could consist of up to 10 hours of playing time, so our board would need to last at least this full 10 hours to be useful for a tournament.

Lastly, we want the system to provide opportunities for the user to replay past games in the future and will therefore need a database that allows each user to store and access at least 10 games worth of notation. Finally, since much of this tool should be used for analysis, the user should be able to export the games in a custom PGN format (the input format accepted by Chess.com) such that they can use the extensive analysis resources on Chess.com for analysis.

5 DESIGN TRADE STUDIES

5.1 Sensor Choice

We narrowed down our sensor choice to two sensors from Texas Instruments using the selection filters on Digikey.

The main remaining design choice was to decide between a unipolar and a bipolar sensor. We decided on a bipolar sensor since it is a more robust sensing architecture. It is more robust because we can simply differentiate between piece color by flipping magnets rather than having to differentiate between two different magnet strengths. This robustness is crucial due to our 100 percent accuracy requirement.

5.2 Magnet Types

We tested several different strengths and sizes of magnets using a small test circuit board with our sensors and a 7mm spacer between the magnets and the sensors. Our findings were that the smaller magnets, regardless of strength, were not sensed accurately in that the sensor had trouble differentiating between them, while the larger magnets were sensed with reasonable accuracy. The larger diameter magnets also would lead to less sensitivity in piece placement on a chess board square compared to small magnets. This was verified in our testing as seen in Fig. 6. The chart lists the magnets we tested along with the error, which is the difference in mT from the expected magnetic field strength at 7mm away compared to the measured field strength. Expected magnetic field strength was calculated using the magnet datasheet's value for surface field strength and the relation that field strength drops off with the square of distance. The measured magnetic field strength was taken from the sensor output voltage during testing using the datasheet value for sensitivity in Volts per milliTesla (magnetic field strength). In the end design with the bipolar sensors and flipping magnets, the high error magnets were avoided because the change in voltage at the sensor was worryingly low and we did not feel confident in the sensor's ability to read pieces with high accuracy, especially if pieces were placed in a non-ideal location.

Magnet Name	Error (mT)	Diameter (mm)
9149	-51.33316327	6.35
8019	-78.24633929	6.35
9144	-108.9707283	6.35
8004	-36.95306122	6.35
8176	**-6.338968112**	**12.7**
8005	-34.26718857	6.35

Figure 6: Data showing that larger diameter magnets have a much smaller error in sensing. Error is defined as the difference between the actual measured field strength and the expected magnetic field strength.

5.3 PCB Sizing

We decided to fabricate the board in four sections of two columns instead of one single circuit board due to cost. A full chess board for our specification is 18 in. in width and height, this made fabrication costs prohibitively high for our budget given the cost of our Bill of Materials. We also wanted to have some money left over to buy extra parts if needed towards the end. We also experimented with decreasing the board size, but even the minimum board size was at the limit of what could we afford given the 600 dollar budget. The four boards are connected such that they all share common power and ground nets, and sensors on every board can be read by the single ADC. For our prototype, we included batteries on one of the boards. However, there is space for batteries to be added on all four of the boards.

5.4 Multiplexing Architecture

The multiplexing architecture was influenced by the PCB sizing. Since the board is being fabricated in columns, the board must be made of columns with a modular design such that each column can function alone as well as together. Having 8:1 multiplexing and an 8-channel ADC makes this easier to design from a circuits perspective. Having an 8:1 multiplexer was also optimal in terms of balancing cost with modularity.

5.5 ADC Choice

We chose our 8-bit, 8-channel ADC because we wanted a single ADC that would efficiently sample from all 8 multiplexers. We chose an 8-bit device because we don't need high resolution to distinguish white pieces from black, and the least amount of bits is optimal for fast SPI transactions to read the sensors with low latency.

5.6 Batteries

We are sizing our batteries to last a full 10 hours, so we want a compact solution that has a large capacity. Lithium-ion cells satisfy both of these requirements better than lead-acid, alkaline, or lithium-iron-phosphate. Using cylindrical cells over lithium polymer cells also is easier because we were able to find cell holders that are able to be soldered directly to the PCB. This takes care of the mounting and electrical connection aspects. Based on all of these factors and our previous experience using 18650 cells, we will be using these cells on our system.

5.7 Legality Check

The obvious way to check the legality of the most recently played move is to identify what piece was moved, check that the piece was moved to a position that its movement ability allows it to move to (Rooks only move in straight lines, King can only move one square, etc.), then check the positions of other pieces on the board to determine that the move doesn't break any other special rules. Another method is to generate all possible legal moves from the previous position, store them in a list, then check if the most recently played move is in that list. The second method clearly requires much more computation, as the program will have to determine the legality of many moves (typically 40-60 moves in complex positions) instead of just the most recent move. However, a key advantage of the second method is that the computation can begin before the move is played, as it only needs the previous position to start generating legal moves. After the move is played, the first method will need to compute its legality, while the second method only has to check it against a list. With our design requirement of 30ms to check the legality of the move in mind, we care much more about the latency of legality checking after the move is played than how much computation is done before, so we chose to implement the second method. Running on the RPi, this legality check method should be fast enough to comfortably support speed chess.

5.8 Legal Move Generation

Many programs exist to generate a list of legal moves for a given chess position. One solution that was considered is Gigantua, the world's fastest legal move generator. Gigantua can generate over 500 thousand legal moves per second, meaning the 40-60 moves we would need to generate for a single position would take around 0.2ms on the RPi [2]. As noted in the previous section, there is not a strict requirement on the latency for generating a list of legal moves, because it occurs before the user has made their move. The legal move list needs to be generated in the time between each player making their moves, which we can reasonably assume to be at least a second. In most games, players will take several seconds, if not minutes to decide on their next move, but it is possible to play very fast chess and we want our system to support those types of games. Even in very fast play, it should take at least a second for a player to physically move a piece and press their button to complete their move. Therefore, we do not need to select the fastest legal move generator, and can also consider other factors: memory usage and ease of implementation. Gigantua uses around 10MB of memory, as it achieves its impressive speed through the use of large look-up tables. We decided to use the legal move generation from Stockfish, a free and powerful chess engine. Stockfish generates legal moves using structures called bitboards, which use 1 bit for each of the 64 squares on a chessboard to represent piece positions, available moves, attacked squares and more [3]. It performs operations on bitboards to generate legal moves, which is slower than Gigantua, but still fast enough for our purposes with much less memory usage. Stockfish takes about 2ms to generate all the legal moves for a single position. Stockfish is also easy to integrate with our software, as it is well-documented and includes structs for positions and moves that are useful in the rest of the legality check program.

5.9 Server Backend

We chose to use Django to build the server back-end for the web application because it's built in the REST framework and because of its community support. REST framework support is important because a portion of our project, having data sent to a remote server, is done through a REST framework. While there are many solutions to sending notation from the chess board to the web server, since our goal is to send small packets of data at discrete times, a REST API was the least effort and most reliable to set up.

Other Python backend frameworks such as Flask fell short of the advantages that Django was able provide. In Flask, it is much harder to set up a basic data storage system to store moves. Furthermore, the documentation for Flask falls short in comparison to the Django documentation in both depth and access. Django also allows for an easily accessible upgrade from our initial SQLite database (our designated database in our MVP) to a larger, more flexible PostgreSQL database that is more suitable for production. Lastly, the Django framework is preferred due to the teams extensive background with the technology. Using a framework in which the team is already familiar is preferable as that allows us to build a cohesive website in a faster and more efficient manner (compared to learning a new framework that does not offer our use case any benefits).

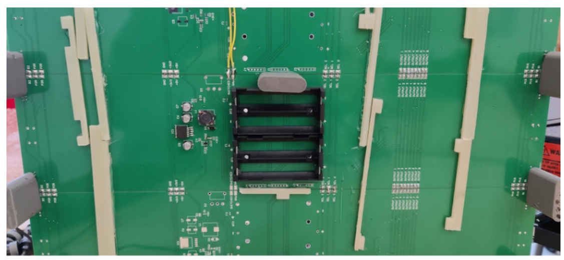

Figure 7: The underside of the PCB, showing the battery holder and structural FR-4 strips.

6 SYSTEM IMPLEMENTATION

6.1 Mechanical Board

The chess board will be built from 2 layers as demonstrated in Fig. 1. The bottom layer is the printed circuit board that contains all of the sensing, power, and interface with the RPi. Sensors will be spaced every 2 inches as seen in Fig. 8 in order to create a normally-sized chess board. This sensor spacing worked perfectly by detecting piece color in nearly every case. The one time the sensing would not work properly is when pawns (the smallest piece) were placed at the very edge of squares. However, in our tests of playing a game, this situation never really arose. The rest of the board worked as designed aside from 1 small fix that had to be done. The buttons that players press were connected incorrectly in the schematic. This was easily solved post-fabrication by cutting 2 traces and soldering a wire on the board. Finally, the interface with the RPi also worked as expected.

Above the PCB will be an acrylic layer that is engraved with a chess board pattern. Below each engraved square will be a Hall effect sensor to sense the pieces. The circuit board will have 3-D printed mounts to support itself and keep all of the individual column boards together. The 3D printed mounts worked to stabilize the individual board segments, but we had not considered the sag in the middle of the board and necessary supports in that area. To remedy this, we added connective struts of FR4 to remove stress from any PCB components and to keep the board level.

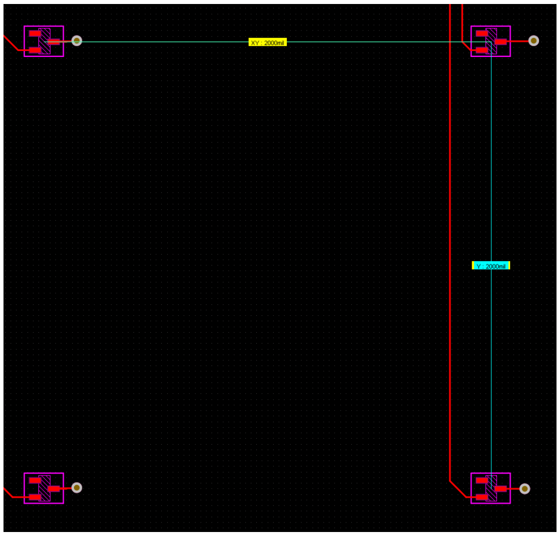

Figure 8: Layout of the PCB, showing 2 inch spacing between sensors. The red footprint in each corner is the location of a sensor.

6.2 Sensing Pathway

For our Hall effect sensors, we will be using the DRV5055 from TI. The schematic for these sensors is seen in Fig. 9. These sensors are bipolar and have a sensitivity of 12.5 mV/mT, with an output voltage of 2.5V when no magnetic field is applied. Each column of 8 sensors has a dedicated 8:1 multiplexer. We chose the TMUX1208PWR multiplexer. Since there are 8 columns in the chess board, there are 8 multiplexers. This sensing pathway worked well. The one unexpected modification tour our exact design was that we needed a small delay to allow the multiplexers to settle when acquiring data. Otherwise, sensing of pieces functioned well.

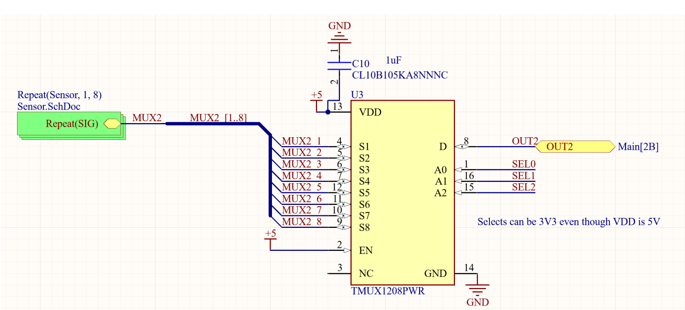

Figure 9: Multiplexer Schematic

These 8 multiplexers feed into an 8 channel ADC, the ADC088S052CIMT from TI. The schematic of all connections to the ADC are shown in Fig. 10. This ADC communicates to the RPi over SPI. The RPi then can process this sensor data to compute legality and respond to the user.

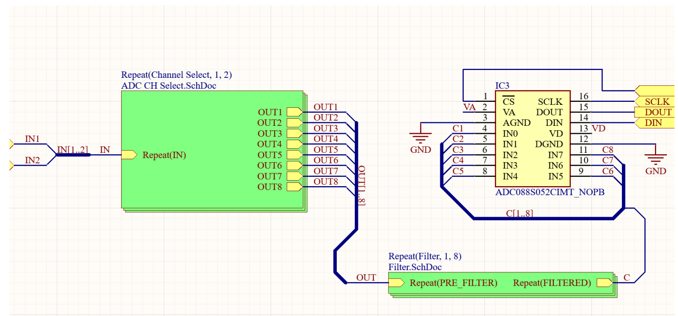

Figure 10: ADC Schematic, showing the outputs of the 8 multiplexers on the ADC's 8 input pins.

6.3 Power

Our battery will be a 2-series lithium ion battery. Our expected maximum power use is about 8.2W, of which, the RPi will use 5W. The rest is from the nominal values for current draw in datasheets for the multiplexers, ADC, and sensors. For 10 hours of play time, we will therefore need 82 Watt-hours of power. Given that the nominal voltage of our battery cells is 3.7V, we will need a maximum of 11.1 Amp-hours of capacity. The capacity of the Samsung 30Q is 3 A-h, so we will use 4 in parallel for a capacity of 12 Amp-hours. Therefore, our battery will be a 2-series, 4-parallel configuration. This power estimate was nearly perfect. The actual power use when tested with a power supply instead of batteries was just above 8 Watts, so the calculations for required battery capacity hold. The board can hold up to 8 parallel of 2 series batteries, which means the board can handle up to 24 Amp-hours of capacity, which well exceeds the required 11.1 Amp-hours.

The battery power will be regulated to 5V using a switching regulator (LM2576S-5.0). This 5V will power the sensors, ADC, and the RPi. For the 3.3V reference for the buttons, an linear regulator (LP2985AIM5-3.3) is used to convert the 5V to 3.3V. The schematic for the power regulation is in Fig. 12. These regulators worked well and successfully powered everything in the system.

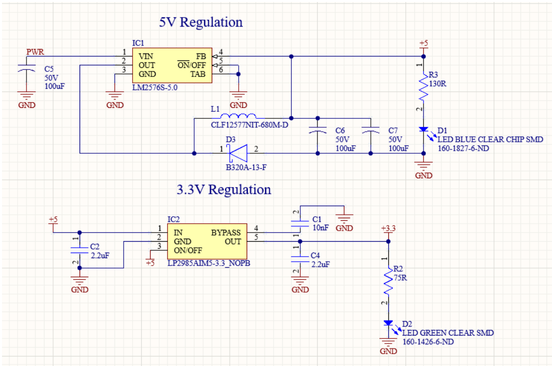

Figure 11: Power Regulation Schematic

6.4 Hardware-User Interface

Our hardware user interface consists of one button and two LEDs per user. The buttons in Fig. 12 are tactile switches (TS15-1212-70-BK-160-SCR-D), and the LEDs in Fig. 13 are green (LTL2T3TGK6) and red (LTL2V3EV3JSR) to indicate legal and illegal moves. The sensors, multiplexer, and ADC all run at 5V. The RPi takes in 5V, but uses a 3.3V logic level. This means that all GPIO inputs and outputs to the RPi must not go above 3.3V. Therefore, the 3.3V regulator on the chess board PCB is used with the buttons to create a 0V or 3.3V input signal. The indication LEDs worked perfectly. They were bright enough, and the control from the RPi worked as expected. The buttons also worked once connections on the PCB were fixed (as mentioned in 6.1), and the debouncing circuit in the form of an RC low-pass filter seemed to work in conjunction with some software low-pass filtering as well.

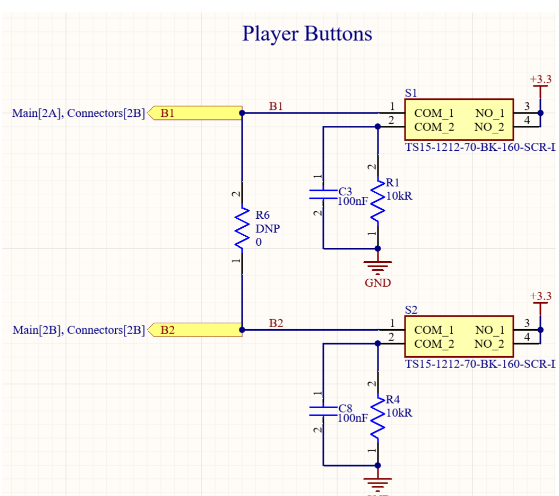

Figure 12: Tactile Switch Interface Schematic. There is one switch for each player.

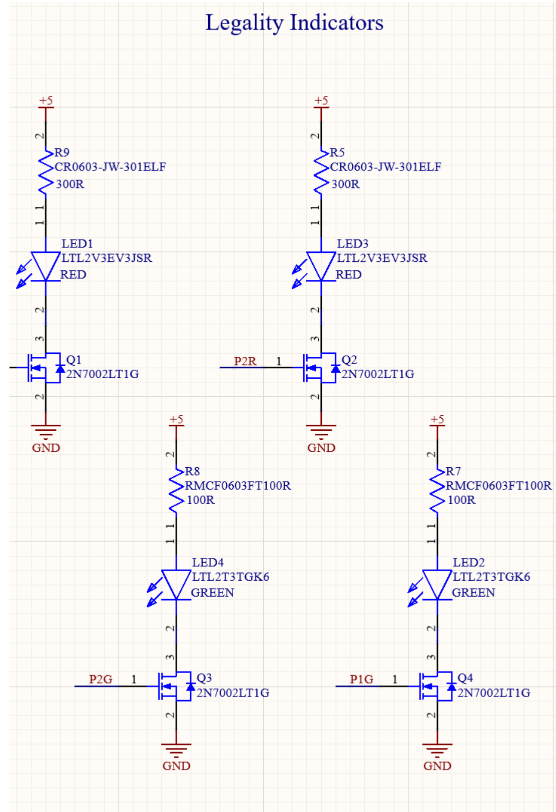

Figure 13: LED Control Schematic. There are 2 red and 2 green LEDS

For controlling illumination of the LEDs in Fig. 13, we also had to work with a logic level change from the 3.3V of the RPi to the 5V system on the board. This level change was necessary because the forward voltage of one of the LEDs is greater than 3.3V and the current limit of a RPi GPIO port is 16 mA. We may not need to deliver more current through an LED than 16 mA, but we did not want needlessly to restrict ourselves. To solve these issues, the outputs from the RPi control transistors (2N7002LT1G) that connect the LEDs from 5V to ground. There are current limiting resistors in series with the LEDs that can be changed if we desire increased or decreased brightness while testing the board. This level change worked well for our setup providing all control lines were actually connected to the RPi. If lines were left floating, the some of the transistors would turn on, and some would remain off. Using a pull-down resistor on the transistor gate could have avoided erroneous LEDs being lit while testing. However, this really isn't an issue in normal operation because the transistor gates aren't floating while the program is running.

6.5 PCB Interconnects

Since our circuit board is made from several identical PCBs, they all need to be connected in order to share common signals. At the edges of each board, there are large pads placed for all major power and signal nets. A subset of these nets are shown in Fig 14. Jumpers (0 Ohm Resistors) can be placed between all of these pads and the pads on the neighboring boards to connect them all. Having all of these boards connected means that power regulation and communication with the RPi only need to happen on one of the boards. These interconnects worked as the electrical linkages between boards. However, some of the jumpers had too much stress on them before supports were added. These components cracked and had to be replaced. Once the supports were added underneath the board, this problem disappeared.

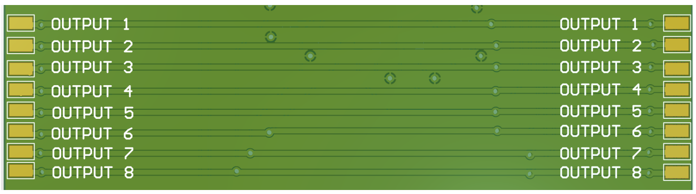

Figure 14: Inter-PCB Jumper Pad Locations. These jumpers carry the output of the multiplexers to other boards, so that the single ADC can read multiplexers, and thereby read sensors, that are on the other boards.

6.6 Firmware

The firmware is the interface between the sensors on the chess board and the software. The firmware will interface with the ADC on the PCB over SPI. Data will be received by the RPi in a format designated by the ADC. The data will then be processed into a format that is convenient to use for the legality check. We chose to write the firmware in C++ for low latency and for easy integration with the legality check software, also written in C++. The firmware worked generally as expected. During the integration period, we worked out some timing issues in the communication between the RPi and the ADC, but there were no other major issues in communicating with the hardware.

6.7 Legality Check

The legality check program will be written in C++, as it is one of the fastest and most efficient programming languages. It will implement the legal move generation from the Stockfish chess engine. At the beginning of the game, and after each legal move is played, the program generates a list of every next legal move from the current position. Once a move is played and sensor data for the move is received, the program compares the new board state from the sensors with the previous board state to determine what piece was moved (source square and destination square). It also checks that only one piece was moved, and the correct color of piece was moved. Then, this new move is checked against the already-generated list of legal moves for the position. If the move is legal, the program will update the state of the board, light up a green LED on the board and begin generating the legal move list for the newly-updated position. If the move is illegal, the program will throw out the move, preserve the previous board state and already-generated legal move list, and light up a red LED on the board.

6.8 Server and Website

Our software infrastructure diagram is shown in Fig. 15. The website front end will use HTML and CSS to display the proper information, and use AJAX (asynchronous javascript) to handle the consistent update logic. We chose to use HTML and CSS for the front end visuals of the website because not only are they industry standard, but they also have a large existing infrastructure. Tools such as Bootstrap allow us to build a better, visually appealing, themed website without too much additional integration work.

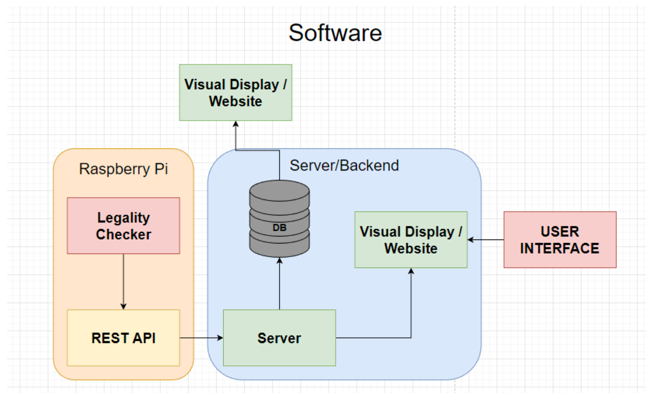

Figure 15: Software Design Architecture

The backend of the server is written using the Django framework and will be modeled using the MVC design (the model, view, and controller) design. This separates the code that loads the HTML, the code that deals with the database, and the code that handles the back end of the server.

It should be noted that the Django framework is written in the Python language, which, due to being an interpreted language, is not the most latency conscious. However, the main goals in mind when building the website are to prioritize being robust and usable.

We can clearly see the MVC in play above as the server backend controls the Visual Display that the user is able to interact with. Furthermore, the database holds the chess notation information that allows for the user to upload their previous games to an external analysis source (Ex. Chess.com). Finally, the user can view the current game being played on the web application, which receives that information through the RPi (located on the chess board itself) which communicates to the backend of the server through the REST API.

6.9 Database

For the MVP, we will choose to use a local SQLlite database. While this initial database doesn't have great scalability, it works great for a few users and under <10000 total games stored. This is due to the built in nature of the SQLlite database into the Django framework: it takes little time to set up, and works well for a smaller production environment. Using a database with a SQL protocol allows for easy query access, which is a primary usage of our web application (say that a user wanted to query the database for all games which they played at the black position before 2022).

7 TEST & VALIDATION

7.1 Accuracy Tests

To test accuracy, we will first make sure that chess pieces that are placed off center (up to 37.5% or 3/8” off center) are still detectable with perfect accuracy. Next, we played through several games of at least 30 moves. In these games, we did not have a single move detected incorrectly. This met our goal of 100% sensing accuracy driven by the need for correctness in the success of our system. One way we could exceed the accuracy metric is if we increase the distance pieces are placed off-center. This could lead to greater tolerances in the sensing system. In terms of our project development, our iterations of sensing were done early on with different magnets and sensors. Our selection of sensor and magnet yielded the best results out of all the options we tested, but with more time we could have tried other options and likely found something better.

7.2 Latency Tests

To test the latency of the legality check, we ran the board through several moves of pieces, and used high frame rate video to time the latency from each move being made (user pushes button) to the legality output (LED on board lights up). Moves will be selected to create high-complexity positions, where the legality check will be slower due to a higher number of legal moves. In our testing with 480 frames per second video, we found latency to be at a maximum of 5 milliseconds. This greatly exceeds our requirement of 300 ms. However, about 1 in 30 moves would take much longer (2-5 seconds) for a response. We were able to identify that this time was spent after the button was pressed but before the sensors are read by the RPi. This leads us to believe that while the firmware is idling and simply waiting for a button input, the RPi may begin running other tasks, and wakes the firmware up after a short time. We could improve thelatency and overall feel of the game by eliminating these long waits, which may be accomplished by telling the RPi kernel that the firmware is a high priority process that should not be interrupted, if possible.

7.3 Power Tests

To test power usage, we measured power usage at idle and and peak computation of the RPi and sensors using a power supply in place of our batteries. This usage will be extrapolated to 10 hours since that is our design requirement. With the power supply, we saw a power draw of about 8W continuous. This power matches our design estimate, so the 2 series, 4 parallel designed configuration will be enough to give the board 10 hours of battery life using the Samsung 30Q cell. To exceed 10 hours of battery life, we can simply add more cells in parallel. The board can hold up to 8 parallel of 2 series batteries, which means the board can handle up to 24 Amp-hours of capacity, which would provide around 20 hours of battery life.

7.4 User Input Tests

To test the overall usability of the system, we will have a suite of user tests. Users will be asked to play a set of 3 games:

- One Blitz Game (with a time control between 3-5 minutes)
- One Rapid Game (with a time control between 10-15 minutes)
- One longer-paced game (with a time control between 30-60 minutes)

Between the second and third games, we will ask the users to attempt to register and login to the web application. They will then attempt to view their past games, and pull up a continuously updating view for their current game. Finally, after they finish playing the third (and longest game), we will ask a series of questions that measure various components of usability.

- Latency
  - “Did you feel any noticeable lag between pressing the clock and the legality checker?”
- Intuitiveness
  - “Did you find the website to be confusing or unintuitive to use?”
- Normality
  - “Did the experience feel irregular, or differ from a normal over-the-board chess game?”

The goal of the above set of questions is to measure whether the technology that we built into the chess board fundamentally changed the usability and playing experience that a normal player might have. We want to incentivize players to use our chess board because it reflects normal usability, but also has many of the initial features that we attempted to implement (automatic notation, chess database). Users will be periodically tested during the implementation phase of the project. Finally, a measure of success (after testing 6 different users) will be to receive less than 2 “No” responses on the total of 18 total questions asked (16/18).

7.4.1 Results

Ultimately, out of 3 candidates, we found that there was no noticeable lag between pressing the clock and the legality checker. The main criticism that we received were that the pieces were too magnetic. We also address this concern in the end results of the program; one improvement is to find magnets that are both larger in size and weaker with regards to magnet strength. The software was intuitive and easy to use, but on more than one occasion, the candidates noted the absence of decoration (CSS, JavaScript).

We interpret these results in a positive light, we are overall ecstatic that our project was able to fulfill all of the basic user needs out of an autonation chess set (and application). However, one of our primary motivations for making this was to prioritize the user experience. Unfortunately the lack of aesthetics on the website and the overall over-magnetization of the pieces caused the user experience to fall short of what we desired it to be.

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule in Fig 16 is organized to accommodate the individual contributions of each teammate. Each teammate's role in the project is, for the most part, individual. The firmware, hardware, and software of the final chess board can be built in parallel, and there are multiple weeks allotted at the end of the schedule for integration among the three parts. Furthermore, there is a break week allotted for fall break to allow for flexibility: if team members feel that they are ahead of schedule, they can take the break accordingly, and if team members feel that they are behind schedule, they can use the week to catch up on remaining work. Lastly, proper time is allotted for tests and validation (most crucial for hardware), shipping, and the physical construction of the board.

Our schedule went through a couple changes post-fall break. The chess board mechanical work extended until the second-to-last week of the semester because it didn't block much of the testing and integration, so the integration was prioritized. For firmware and software, the completion dates on components were pushed back 1-2 weeks, which then pushed the integration timeline into extending two more weeks. The final testing stage and some final integration therefore used up the slack time we had planned for in the last 2 weeks before final presentations.

8.2 Team Member Responsibilities

Vikram is responsible for the hardware portion of the project. This includes the mechanical chess board design and the circuits and PCB design. Patrick is responsible for the firmware interface with the hardware as well as the legality check and state machine logic of the system. Ryan is responsible for the software portion of the project that includes the the server, database, and web interface as well as the RPi's interface with the server. As individual portions of the project are completed, we begin the integration process. Integration will include making the hardware and RPi work together and then making the whole hardware and firmware system work with the software. The final step that will be done together is testing and validating our system and making any necessary revisions to any aspect of the design.

We ensure that all parts are as parallelizable as possible, by building interfaces that act between each individual part. Vikram's work will interact with Patrick's firmware, and Patrick's firmware will interact with Ryan's software. However, significant progress can be completed as the majority of the work can be done without extensive knowledge of the interfaces.

Throughout the project, these individual responsibilities remained constant. The division of work between hardware, software, and the interfaces in between proved to be a feasible setup for the semester.

8.3 Bill of Materials and Budget

Our bill of materials in Fig. 17 consists of all of the PCB components, PCBs, acrylic sheets, and the chess pieces. The total cost comes out to $301.95. We are comfortably within the $600 budget, and we have enough money left to buy a second revision of the PCBs or more spare components if absolutely necessary. The breakdown of the BOM is in Fig. 17. Some costs that were hard to estimate were the 3D prints used to support the board and hold the magnets. In comparison to the cost of the boards and electronics, this cost is small.

8.4 Risk Mitigation Plans

The largest risk from the hardware perspective is that the PCB does not work as expected. To mitigate some of this risk, I had the group review the schematics to make sure everything made sense. Additionally, the board is large enough and only 2 layers. This means manual alterations to the board are possible, but not necessarily easy. Another solution is that we could order a second revision of the PCB and reuse the existing bill of materials that we already bought. A benefit of the modular PCB design is that the cost allows for a second revision while staying under the 600 dollar budget. Stock is no longer a risk since we have purchased all of our PCB components, and nothing was backordered. In the end, we did not need to order a second revision of the board. One or two small changes to the board were made by adding wires and cutting traces, but these were minor enough to not warrant a new board revision.

The largest risks from software include packet loss. It is important to note that while the code can be tested, and the website can be made hard to bypass, it is possible that information from the RPi might be lost in transit or corrupted while being transferred to the website. We can

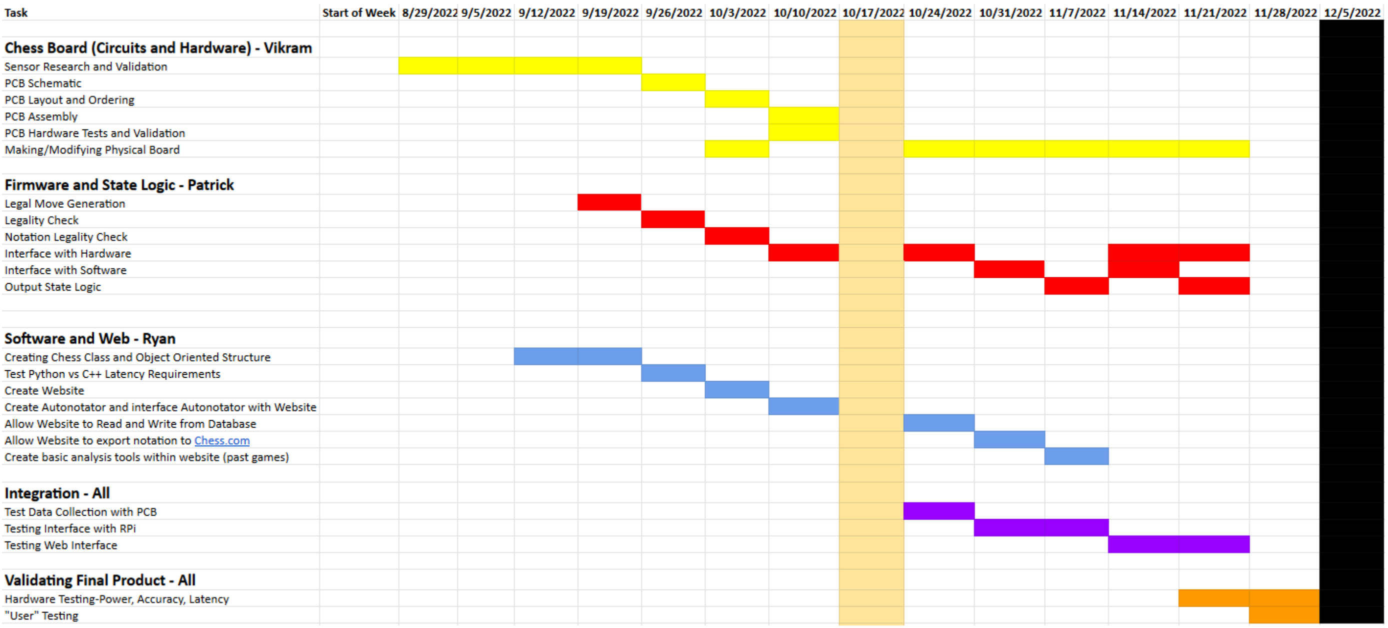

Figure 16: Gantt Chart

Description	Vendor Part Number	Manufacturer	Quantity	Unit Price		Item Total	
MAGNET 0.5"DIA X 0.25"H CYL	469-1050-ND	Radial Magnets, Inc.	16	$	1.58	$	25.22
51Ω Resistor 0603	118-CR0603-FX-51R0ELFCT-ND	Bourns Inc.	1	$	0.10	$	0.10
Ratiometric Hall Effect Sesnsor	296-DRV5055Z4QDBZRCT-ND	Texas Instruments	67	$	1.08	$	72.31
10kΩ Resistor 0603	CRT0603-BY-1002ELFCT-ND	Bourns Inc.	2	$	0.38	$	0.76
0603 Jumper	RMCF0603ZT0R00CT-ND	Stackpole Electronics	11	$	0.01	$	0.14
Schottky Diode 20V 3A	B320A-FDICT-ND	Diodes Incorporated	2	$	0.42	$	0.84
75Ω Resistor 0603	RNCP0603FTD75R0CT-ND	Stackpole Electronics	2	$	0.10	$	0.20
300Ω Resistor 0603	CR0603-JW-301ELFCT-ND	Bourns Inc.	3	$	0.10	$	0.30
130Ω Resistor 0603	CR0603-FX-1300ELFCT-ND	Bourns Inc.	2	$	0.10	$	0.20
1kΩ Resistor 0603	CR0603-JW-102ELFCT-ND	Bourns Inc.	8	$	0.10	$	0.80
100Ω Resistor 0603	RMCF0603FT100RCT-ND	Stackpole Electronics	3	$	0.10	$	0.30
1206 Jumper	2019-RK73Z2BTTDCT-ND	KOA Speer	68	$	0.05	$	3.20
Tactile Switch	179-TS15121270160SCR	CUI Devices	2	$	0.69	$	1.38
Slide Switch	118-5MS1S102AM6QES	Dailywell	1	$	3.17	$	3.17
N-Channel MOSFET	863-2N7002LT1G	onsemi	6	$	0.27	$	1.62
10 nF Ceramic Capacitor 0603	810-C1608X7R2A103M	TDK	2	$	0.10	$	0.20
100 nF Ceramic Capacitor 0603	187-CL10B104KB8NNNC	Samsung Electro-Mechanics	14	$	0.02	$	0.21
Red Led Through-Hole	859-LTL2V3EV3JSR	Lite-On	2	$	0.47	$	0.94
Green Led Through-Hole	859-LTL2T3TGK6	Lite-On	2	$	1.29	$	2.58
3.3V 150 mA Linear Regulator	926-2985AIM53.3/NOPB	Texas Instruments	2	$	1.14	$	2.28
5V 3A Switching Regulator	926-LM2576S-50	Texas Instruments	1	$	6.27	$	6.27
8 Channel ADC	926-AD088S052CIMTNPB	Texas Instruments	2	$	4.45	$	8.90
8:1 Analog Multiplexer	595-TMUX1208PWR	Texas Instruments	10	$	0.87	$	8.68
Green SMD LED Indicator	859-LTST-C190GKT	Lite-On	1	$	0.24	$	0.24
2.2 uF Ceramic Capacitor 0603	187-CL10A225KO8NNNC	Samsung Electro-Mechanics	4	$	0.10	$	0.40
1 uF Ceramic Capacitor 0603	187-CL10B105KA8NNNC	Samsung Electro-Mechanics	12	$	0.02	$	0.28
68 uH Power Indudctor	810-CLF12577NIT680MD	TDK	1	$	2.00	$	2.00
Dual 18650 Battery Holder	534-1049	Keystone Electronics	2	$	6.37	$	12.74
Blue SMD LED Indicator	859-LTSTC193TBKT5A	Lite-On	1	$	0.30	$	0.30
6 position right-angle 100 mil header	575-8292200620	Mill-Max	3	$	6.92	$	20.76
32V 5A SMD Fuse	594-MFU0603FF05000P1	Vishay	4	$	0.27	$	1.08
100 uF Electrolytic Capacitor	647-UCQ1E101MCL1GS	Nichicon	4	$	1.29	$	5.16
Custom PCBs	N/A	PCBWay	5	$	16.68	$	83.40
16" by 16" 1/4' Acrylic	N/A	Professional Plastics	1	$	25.00	$	25.00
Chess Pieces	N/A	The Chess Store	1	$	10.00	$	10.00
				Total		$	301.95

Figure 17: Bill of Materials

manage this data loss by creating a checksum, or a secondary verification that the move sent did not corrupt in transit. This checksum works by checking to see whether the given information packet details a valid board state, and can be derived from a hash of the board. This creates a quick and reliable method of double checking the validity of an information packet sent from the RPi. In the case that an information packet sent from the RPi is valid, the website will send a confirmation to the RPi that the proper information is received. If the confirmation is not received within a given time period, the RPi will send the information once more.

In terms of personnel and timeline resource and risk management, the main risks were that some section of the project would not get done on time and we could not finish by the demo. To mitigate these risks, we planned slack time into our schedule to compensate for busy weeks and overflow of work. We ended up using all of the slack time, so it was good that we planned for it. Additionally, we were able to make up some work during the fall break to get the team back on track and on schedule.

One of the risks that we did not foresee was sending too many requests to the website during the demo. We ultimately decided to deploy the website to repl.it rather than using the website locally as that would allow for guests to access the website and create their own accounts. However, during the demo, with some live users, the website went down due to volume of traffic pushed through (this is due to repl.it offering free but limited deployment services). Ultimately, during the demo we had to revise the number of seconds between each refresh of the page to lower the stress on the website. To mitigate this risk in the future, we should deploy to an EC2 instance which is much more robust and offers larger flexibility.

9 ETHICAL ISSUES

There are a couple important considerations about the morality of our project. Since this board will be used in a tournament context, we want to ensure that our board preserves the integrity and secrecy of the games being played.

The primary concern of this board is that in a tournament scenario, one which we would imagine that our board would be use frequently, our board may generate a false positive. That is to say, that the board may say that an illegal move is legal (the opposite is also concerning, but has much smaller ramifications). In this case, if both of the players fail to notice, then the chess game would proceed in an illegal manner. Especially if this was a tournament with incentives, this would cause conflict and would be the fault of the board.

The second concern that we had in mind was the secrecy of the games played in a private setting. One of the features we implemented was the ability to share games. In this sense, we wanted users to be able to share their notation in a tournament setting (removing the need for either player to take notation). However, while this feature certainly makes life easier in a tournament setting, malicious players can certainly gain an advantage by viewing the preparations of their opponent before an important match. With this in mind, we prevent unnecessary notation by designating each game with an owner. Being the owner allows for the owner of the board to end games played during a tournament setting so that no extraneous notation is written to the end of the game.

10 RELATED WORK

As mentioned in the Introduction, a similar product is ChessUp. It is also a custom chess board that can detect moves made on it. It includes a built-in chess engine, which finds possible moves and can recommend the strongest ones to the user. While it has some features that our product does not, the cost of our board is lower. Given the allocated time for this project, the scope of our design is more reasonable. Our final project is very similar to what we set out to do, so ChessUp is still a similar product to what we created.

11 SUMMARY

The Chess Autonotator system is an all-in-one product that offers the familiar look and feel of a standard chess set with the conveniences of automatic game notation, legality checking, and a website to review past games and export them for analysis. The notation and legality checking will be too fast to even be noticeable by most player and will be perfectly accurate. The board will also offer 10 hours of playing time.

We were able to meet the specifications of our design. Two things from the hardware standpoint that could easily improve the design are slightly decreasing the square size or increasing the magnet diameter. Both of these options help the issues of smaller pieces (pawns) being placed far off-center and then being undetectable by the sensor. One potential issue that could arise is that pieces would be more likely to attract or repel each other due to the internal magnets. Therefore, doing some simple searching to find a larger, but weaker magnet would be useful.

From a software perspective, we focused a lot of our effort on making the game interface efficient and smooth from a user perspective. The priority on the software side was to build a MVP of the system that could efficiently play games, rather than to spend time making the website look as good as possible. As a result then, the website is plain HTML, but functions well in the context of the system (satisfies the use case requirements). One of the biggest improvements that we can make to our overall visuals is to use a CSS Framework (Ex. Bootstrap or React) to improve the look of our website.

While we defaulted to using a GET request to send moves to the server, using a REST API (a slight improvement from a GET request) would improve on the overall parallelism of the server (improving the performance under load). While this was not relevant for our demo (as there was a very light load), when thinking about our project in a commercial lens, we would want to implement a more robust system using a REST API.

As mentioned in the testing section, we would like to eliminate the rare long-latency moves. This would likely require some more in-depth learning about the RPi and its operating system. We would also like to add support for starting games from any position, as multiple visitors to our demos expressed interest in this. This could be implemented with an input on the website that accepts a FEN string, which encodes a chess board position. The software could easily accept a FEN string to begin a new game from the given position. Another possibility would be having a chess board UI on the website where users can set up the game, just like on Chess.com or other chess websites, and have the website backend translate that position to a FEN string.

One lesson learned was that we needed to validate a hardware sensing architecture early. We did this by ordering a variety of magnets to test on the sensors early on in the semester to give us confidence that the final product would work. This contributed to a much easier integration timeline because we knew sensing would work with minimal effort on the mechanical side. One aspect we should have tested more was the tolerance on square size and how close the magnets had to be to attract each other. This was largely not an issue in the end, but we had not validated this in advance.

A lesson learned on the software side was that small changes to the overall system can cause large rippling changes in the overall software. Much of our design process had to be changed when different issues were raised. The original software design was inflexible, and would not have allowed the current game setup (where users were able to add games created by other people). As a design principle, software is much easier to modify than hardware, and therefore should be designed with flexibility as a top priority.

A lesson learned from the firmware development was to test the needed functionality of IO libraries early on. The firmware was initially written with a library that sounded very convenient, but hadn't been supported since this January and was no longer completely functional on the RPi. Finding other libraries to accomplish the same function was not too difficult, but rewriting the firmware to account for the changes between libraries was tedious.

Glossary of Acronyms

Include an alphabetized list of acronyms if you have lots of these included in your document. Otherwise define the acronyms inline.

- RPi – Raspberry Pi
- PCB - Printed Circuit Board
- API - Application Programming Interface
- ADC - Analog to Digital Converter
- SPI - Serial Peripheral Interface
- GPIO - General Purpose Input Output
- HTML - Hyper Text Markup Language
- CSS - Cascading Style Sheets
- JS - JavaScript
- REST - Representational State Transfer
- MVC - Model, View, and Controller

References

[1] *ChessUp*. URL: `https://playchessup.com/`. (accessed: 12.16.2022).

[2] Daniel Inführ. *Gigantua*. URL: `https://github.com/Gigantua/Gigantua`. (accessed: 12.16.2022).

[3] J. Kiiski T. Romstad M. Costalba and the Stockfish community (https://github.com/official stockfish/-Stockfish/blob/master/AUTHORS). *Stockfish*. URL: `https://github.com/official-stockfish/Stockfish/`. (accessed: 12.16.2022).