The Arcane Game Board

University of Central Florida

Department of Electrical Engineering and Computer Engineering

EEL4915 Senior Design 2

Group 22

Lucas Lage – Computer Engineering

Fernando Valdes-Recio – Computer Engineering

James Strickland – Computer Engineering

Kayla Freudenberger – Electrical Engineering

Customers

Self-Funded Project for Public Consumption

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 EXECUTIVE SUMMARY

When our team began discussing potential projects it was clear we were all looking for a project capable of showcasing our technical skills and passions. Not only, but we also wanted the project to be fun and appeal to our personal interests as well. Through our discussions, we ultimately decided on the Arcane Game Board. This project idea appealed to our shared interest and enjoyment in board games while also being the perfect project to demonstrate each team member's skills. Our overall goal is to create a Chess game board that moves the pieces without human intervention and utilizes a web application to control the gameplay. The intention is to put a unique and fun spin on the centuries old game of Chess.

This project is designed such that the user can play an entire game of chess without having to move any pieces. Instead, the user interacts with a web application to indicate their desired moves. These commands are then be sent to the PCB within the game board which controls an x-y cartesian robot and electromagnet beneath the play space to move the game pieces. The software also includes a capability for the user to play against their friends and watch as each move is displayed on the board in real time. The game board also moves "captured" pieces off to the side of the play area when needed and be able to reset after a game is completed. In addition to these core functionalities there are also several additional functionalities we'd like to add to this project to make the game board more versatile and fun given we have enough time and resources.

Moving forward with the overall design idea, the next step was developing specific objectives and requirements that we determined must be met to bring the project to life in its entirety. As engineers, it is imperative we develop these requirements and objectives to help us accurately gauge the work that needs to be completed to deliver an end product. We identified many objectives, the first and foremost objective being the piece movement on the board. The movement of the pieces on the board is the core and basis of the whole project design and therefore it is necessary that we accomplish this objective and ensure that the mechanism works as efficiently as possible. Another key objective is to design the movement mechanism such that it can perform the piece movements quickly and accurately without colliding with stationary pieces. We want the gameplay experience, from a user perspective, to closely mimic a typical game of chess in terms of speed and piece movement. This also means that the path that the piece moves on should be the shortest path possible and close to how the user would move the piece. We plan to accomplish this by creating an x-y cartesian robot that can move to any location on the board as well as ensuring the electromagnet has the right amount of strength to move the desired piece without affecting any pieces around it. Additionally, we are handling all the movements through software and this includes collision avoidance and therefore must be keeping track of all pieces on the board in real time. Since the software is handling the piece tracking, pathing, and collision avoidances entirely, we must also ensure our web application communicates effectively with the PCB which is controlling the actual physical movements. To accomplish this, we intend for the software to communicate with the hardware, most likely, over Wi-Fi meaning we must design our PCB such that it can receive the necessary data this way.

Another objective we have pertains to the size and weight of the gameboard. Ultimately, we want to avoid making the board too heavy or large because it would negatively affect the ease of use and portability of the game board. It should be easy to move, set up, and should take up a reasonable amount of space. We intend to keep the board within a few inches of a standard chess board as we still want it to fit on a standard size table. The additional inches we are adding along the perimeter accommodates the internal mechanisms as well as allowing for a "piece graveyard" where all captured pieces go throughout the duration of the game. The decision regarding the material for the physical game board was made based on a lightweight but sturdy design.

In addition to these engineering objectives, we also must consider our project from the consumer's perspective and assess the potential marketing requirements. In addition to a light-weight design and ease of use, we also want to make the board relatively low-cost for the potential consumer. Since this project is not one that satisfies a need but instead serves as entertainment, we must strongly consider the cost in terms of the value that this project realistically has. If the board is too expensive then there won't be a market for the product anymore. However, we must also keep in mind that by making it low-cost, we are sacrificing quality and accuracy therefore we must find a reasonable balance.

To accomplish both the engineering and marketing requirements laid out for this project, we spent a considerable chunk of time researching related projects, relevant technologies and parts to determine the best design given all of our constraints. The goal of our research is to determine which parts and software will yield a functioning product that meets all of our objectives and goals. Most importantly, we must investigate potential avenues for our x-y movement mechanism as well as determining the software and firmware that is be most efficient and effective for the designated requirements necessary for this project. As engineers, we recognize that thorough research will greatly benefit the actual prototype construction and design therefore we considered many different technologies and parts to weigh and carefully weigh the advantages and disadvantages against our requirements.

We must also take into consideration our constraints during our research and design and continue to assess our project realistically against those constraints. Our biggest constraint moving forward is time. The nature of this project requires us to deliver a working prototype within a set period of time and therefore we must carefully consider how much time our design will take to implement realistically. Additionally, we must consider our financial constraints as well, this project is entirely self-funded therefore we are limited in how much we can safely spend on this project as a team. To ensure we are operating realistically within both our time and financial constraint, we developed a budget and timetable to adhere to throughout each phase of project research and development. In addition to the base functionality of the project, we discussed many potential stretch goals as well however the time and financial restraints were the most significant barrier for these goals and needed to be considered on a case-by-case basis.

The Arcane Game Board combines engineering and fun along with an overall goal of facilitating a unique and magical experience using only a simple grid-style game board. This project is not only an opportunity to showcase each member's skills, but it is also an opportunity to learn. There are many aspects, familiar and otherwise, that must be considered when designing this project and we must ensure we are prepared to tackle each piece of this project effectively. This project challenged us in many ways but by the end we have grown as engineers and professionals alike. Our goal, ultimately, is to deliver a finished, working prototype that each member is proud of.

2.0 PROJECT DESCRIPTION

Board games have long been a staple in leisure activities, with many using a simple grid system of tiles to constrain possible moves. A good example of this is Chess. Chess has been around for hundreds of years and utilizes a simple 8x8 grid - within which the players can move their pieces about. Despite the board's simple layout, the game itself is anything but. Another popular example is Tabletop Roleplaying Games (TTRPGs), such as Dungeons & Dragons, which use a standard grid system comprised of 1_{inch} x 1_{inch} tiles and function much the same way that chess tiles do.

As a result of recent mass media, Chess and other boardgames have seen a rise in popularity among the general public and the ability to play these games in a unique way may appeal to many. However, due to current world events and social distancing, many people are now resorting to virtual versions of these classics as a means of entertainment and online interaction. Additionally, games such as Dungeons & Dragons are often forced to be played online due to the time requirements, number of players, and distance between long-term game groups. While these online versions suffice, the physical component that is so integral to the game experience is missing. Well, what if you could play these games on a physical board while still maintaining the benefits of playing online?

Enter the Arcane Game Board. Utilizing a mechanical system concealed beneath the play space, the Arcane Game Board moves pieces about the board without the need for human intervention. This means the game board is responsible for all piece movement and piece tracking. The user must indicate all moves through the web application.

2.1 PROJECT MOTIVATION

Board games have withstood the test of time and continue to be enjoyed by young and old alike. Chess, in particular, has been around for at least 1500 years and has long been hailed as a true game of strategy and competition. The simplistic grid layout has kept this game accessible while still maintaining unique gameplay and challenge; easy to learn, difficult to master. We see this grid layout being implemented in many alternative games over time, most notably we've seen the rise in Tabletop Role Playing Games such as Dungeons and Dragons.

Mainstream media recently has breathed new life into these games, bringing them back to the forefront in entertainment. Most notably we have seen a jump in popularity for Chess over the past several months. Chess is a very old game and has been adapted to be played on mobile devices and computers online, either against AI or against friends, removing the need to physically be present to play. This has become much more relevant with the current pandemic, requiring people to socially distance and avoid physical contact with each other. While these games have translated well to online play, there is still a certain charm to playing on a physical board with physical pieces to move. The challenge then becomes, how can we incorporate physical gameplay with online opponents. Herein lies the motivation for this project.

With the help of technology, we are able to create a physical game board that moves the pieces and accurately recreates a classic game of Chess from anywhere in the world. With the Arcane Game Board, not only can you play against an AI or friend, you get to watch the pieces move in real-time, almost like magic. This project is perfect for those who prefer a "real" game without needing to be in the presence of another person. Not to mention that it takes a game often considered to be “boring” and adds an element of intrigue and encourages engagement from an audience that perhaps may not have found interest in it before.

We were also motivated to pursue this project in part because of a shared interest in board games among the members of this team. Whether it be Chess or other table-top board games, we all find enjoyment from this kind of game and wanted to use our engineering knowledge to create something exciting and fun. It is important for us to feel a personal connection to our project and we were intrigued by the possibilities this project idea had in addition to the base design. Through having this personal interest in board games, we are then able to make specific design choices we know will enhance the gameplay overall and resulting in a better end product.

Our idea was also partially inspired by a scene in the popular movie and book *Harry Potter and the Philosopher's Stone.* In the scene, the characters must play a real-life game of *Wizard's Chess*, where the pieces magically move on their own. We want our project to encapsulate that same magic feeling from the comfort of your own home. Afterall, child and adult alike, we are fascinated by the idea of magic and the impossible being possible. With the current technology available to us, we are able to create this for all ages to enjoy.

2.2 Project Details

The Arcane Game Board utilizes a combination of software and hardware to create an interactive and unique gameplay experience. The overall design and functionalities were carefully considered and evaluated against both time and financial constraints. The following sections will outline the key functionalities needed to fully realize our overall concept.

2.2.1 Motion

In order to achieve motion, the pieces are fit with a small neodymium magnet at their base. Beneath the board a cartesian robot operates an electromagnet able to reach all of the play area, as well as significant areas of the border. When the system needs to move a piece across the game board, it moves to the tile beneath the piece, activate the electromagnet, and then drag it across the gameboard. Pathing algorithms allow for obstacle avoidance and temporary obstacle clearance when pieces would be able to “jump” over each other.

All motion is driven by the on-board controller, with commands being translated into cartesian movement via commands similar to G-code. Belts guide the 2 axes along aluminum extrusions, with each having a dedicated stepper motor capable of providing precise movement. The x-axis is moved along the y and contains the electromagnet.

In order to ensure that it stays within alignment, the y-axis has 2 belts on either side of the game board and the rod run through the stepper motor. If the lengths of the tracks are sufficiently close the axis should be impossible to misalign.

2.2.2 External Controller-App

The arcane gameboard utilizes an external computer in order to reach all of our goals and stretch goals. The eternal laptop can be connected by either a USB wire, or a wireless Wi-Fi connection. This connection allows the commands to be sent from the laptop, similar to how a 3D-printer utilizes a laptop. This allows our design to reduce the on-board hardware necessary for event planning and decision making. The arcane gameboard also includes computer enabled controlling of the pieces.

A User Interface (UI) is designed in order to provide a friendly User Experience (UX). The UI is carefully designed in order to be intuitive and easy to use. The UI includes a live virtual representation of the physical chessboard and a historical log of all of the moves made during that game.

The UI communicates with a database via an API in order to store move data. The database also receives event data from the PCB. This data is then used to update the UI and keep it real time with the physical game board. The database runs locally on the same laptop as the UI. The web application allows two players to play from anywhere in the world with each other.

2.2.3 On-Board Controller

A micro-controller is required to drive the motors, act as an interface with the external controller, and track the state of the game board. Existing platforms which are capable of this are widely available due to the similar requirements of a 3d-printer, easing cost of development and manufacture. However, the mapping and pathing required novel solutions.

2.2.4 Gameplay

Gameplay is comprised of turn-based movement between the board and the user. The order of play is determined by which player is the lighter colored team as in normal chess.

On an AI turn the board plans a move, locates the piece on the board according to its internal map, and then instructs the cartesian robot to move the piece as needed. For knight pieces it may be necessary to dodge other pieces on the board. Moving by snapping the piece to the outer edge of the square was the most common solution. In cases where the knight is surrounded by many pieces the AI attempts to move the knight between 2 of them without disturbing the other pieces.

Player turns operate much the same way, designating moves on their system which is then be carried out by the board without the need for physical intervention. Since only one move is allowed per turn, the player does not need to signal that they are done; only move the piece in virtual space.

Spaces off of the center of the board are designated retirement zones for pieces removed from play. When captured, pieces are be dragged back to their designated retirement tile, allowing them to be retrieved by players in the event of a pawn promoting.

2.2.5 Tracking

Piece tracking is done entirely through the software since there was not enough time to implement sensor tracking for the pieces. This means that pieces can only be moved by the robot and cannot be physically moved on the board by the player because the board is unable to detect the movements.

2.3 Goals and Objectives

This section is divided into subsections that will provide a detailed description of our project goals, stretch goals, and objectives. It is imperative, as engineers, that we develop realistic and measurable goals and objectives in order to create a fully functioning final prototype.

2.3.1 Main Goals

Our first goal is to design a mechanism beneath the board which moves the pieces to the intended location using an electromagnet. The mechanism must be able to perform precise movements on a singular piece without affecting the surrounding pieces. The precision of this movement is necessary because there are many instances in a single game of Chess where pieces must move around other pieces. A well-known instance of this is with the Knight, the only piece capable of "jumping" over other pieces. The mechanism has to maneuver along the edges of a tile to avoid collision with other pieces. These pieces should also take the most efficient path to their destination and avoid confusing paths that make the piece movement unclear to the player.

Another goal is to design the board such that the game requires no human intervention. Pieces should move to their intended location without the player needing to interact with them. This includes moving "captured" pieces off the board to a designated area located on both sides of the board. This requires that each piece has its own unique identification to ensure the correct placement of pieces in the graveyard.

In addition to the first two goals, the pieces should move at a reasonable speed to avoid long delays and keep the pace of the gameplay. If the movement mechanism takes too long, it will break immersion and severely alter the flow of gameplay.

In order for the game board to function without human intervention, we need to develop a web application that communicates with the PCB over Wi-Fi. The goal is to have the piece movements and identification handled through the web application which sends commands to the board to move the pieces. The user is be able to input their desired move and the other player will respond with a corresponding move. This requires a clean User Interface the player can interact with that also displays the current piece location to assist with the players decision making during gameplay. Additionally, the application must also be able to prevent the player from making any illegal moves during a game, such as putting their King in Check or moving a piece in a direction it is unable to.

Finally, our last goal is to create a seamless and enjoyable gameplay experience for advanced and novice players alike. We want this game board to be accessible and easy to use. We've set a goal to keep the game board reasonably sized and lightweight to allow for portability. Anyone should be able to pick this up and play without any issues. The Arcane Game Board is about having fun and playing Chess in a new and exciting way.

2.3.2 Stretch Goals

This section will detail our stretch goals we'd like to implement given we had enough resources and time. We know that this idea has a lot of potential to elevate and expand the gameplay experience. Below, we discuss these stretch goals in order of implementation priority.

Our first stretch goal would be for the Arcane Game Board to support voice control, allowing the player a completely hands-free experience. The player would be able to select which piece to move and where to by saying their move out loud and the board would respond accordingly. The player would have to announce the piece location that they would like to move, followed by the location that they are targeting.

The next stretch goal would be to implement the capability to allow the player to move their own pieces physically and the board would track these movements. This would give the player the choice to play hands-free or they would have the option to move their pieces without needing to use the web application to indicate their moves. This would require sensors being installed in the physical game board to keep track of piece location. Each piece would also need its own unique identification, so the board is able to determine what piece is occupying what location.

Another stretch goal we have is allowing for a player to play against AI on the board through the web application. The player would take turns selecting their moves and the board will move the corresponding pieces. The main idea behind this is that the player could be alone and still play a full game against the AI. This would require that the web application is able to communicate to the board for both the player inputs and the AI inputs. Additionally, we would need to set up different levels of difficulty to accommodate players of different skill levels.

To add more interest and excitement to the game, our next stretch goal is to add LED lights to the board that are triggered during certain scenarios. For example, having the lights flash blue when the King is in Check or red when you reach Checkmate. This goal is primarily aesthetic and would add an element of intrigue to the game board as well as keeping the player engaged and excited.

Due to the grid layout of the Arcane Game Board, it has the potential to accommodate tabletop games other than Chess. Our next stretch goal would be to support multiple different games that could be configured through the web application. This would make the game board significantly more versatile and reach a larger audience. As long as the piece movement follows the same conventions as Chess, we would be able to implement more games like Checkers or D&D and the player would be able to select which game they'd like to play via the web application. As and add-on to this goal, we have also considered implementing the ability to create a custom game setup in addition to having multiple game options.

Our last stretch goal is to leverage display technology and the PC to show suggestions and effects during a match. This would be implemented as a feature you could turn on to show possible moves and overlay a threat map to assist the player with move selection. Having this feature would allow for less experienced players to grasp the game rules and strategy.

2.3.3 Objectives

This section will break down our goals into the main objectives that guided us towards a fully functioning, finished prototype. Each objective is necessary in order to accomplish our goals stated in section 2.3.1.

Our top priority objective is the piece movement across the board. Our project's success is dependent on our board being able to move the game pieces accurately and consistently on the board to their intendent locations. Without this functionality, our project would be nothing more than a simple chessboard. To accomplish this objective, we have an x-y cartesian robot underneath the board connected to an electromagnet which turns on when moving the pieces. Each piece needs to have a small magnet fixed to the bottom, allowing the electromagnet to move the pieces during the game.

The x-y cartesian robot is connected to our PCB and its movements is commanded by the player's inputs into the web application over Wi-Fi. This objective requires that the x-y cartesian robot is able to maneuver the electromagnet to any location on the board and can make precise movements along the intended path. Additionally, we must ensure that the correct amount of voltage is supplied to the electromagnet such that it only moves the desired piece without affecting any pieces in the surrounding area as it moves from point A to point B.

Software must be developed such that the robot is given commands – EG movement, electromagnet On/Off, and homing, to execute during the stage of the game. The MCU of the board itself needs to be able to interpret these commands, as well as handle basic events such as collisions and errors. The movement system of the physical robot must also be accounted for – as different families of cartesian robots require different control schema. Finally, the MCU must have a serial connection to an off-board controller capable of handling more complex event handling as well as game state tracking.

The off-board app handles most of the heavy calculation and internet traffic. This includes updating the shared game state, which is edited by the players, collision avoidance for pieces such as the knight, and providing the UI which players use to execute moves. In order to connect to the board, there must be a serial connection between a client instance of the controller app and the MCU of a specific board. Commands are sent through this serial connection once they are generated by the controller app.

The next objective is the construction of the physical board. The board itself must house the PCB, power supply and x-y cartesian robot with a dimension of less than 24 inches by 24 inches. A standard chessboard measures 21 inches by 21 inches and we added additional surface area to accommodate for the “graveyard”; where pieces go after being captured. In addition, the top of the board, where the game is played, must be thin enough for the electromagnet to affect the pieces above and the surface must be smooth the allow for the pieces to move . The top of the board must also depict the grid layout of the game with 64 alternating black and white tiles.

In addition to the construction of the overall board, we also much build the x-y cartesian robot and connect it to the PCB. This required careful planning and identifying where wires need to be placed in order to not interfere with the movement of the electromagnet. Additionally, we must ensure that the x-y cartesian robot moves smoothly and is able to traverse the entire game board without getting stuck. We also must design the PCB such that it is capable of receiving information from the web application and translating that into physical movement such that the pieces move to the intended locations consistently and accurately.

Our final objective is to ensure the final design is user friendly, meaning that the design is reasonably light weight and easy to set up. This objective ties in with the pervious objective in terms of being light weight, the smaller we are able to make it, the less material we use and is therefore lighter overall. We also want the user to easily be able to set up the board for a game both in regard to setting up the physical pieces and preparing the web application for a new game. The user interface for the web application should be simple to understand and intuitive for the player.

2.4 Requirement Specifications

This section contains the engineering specifications that must be met in order to deliver a working prototype that fulfills our goals and objectives for this project. We identified both hardware and software specifications that must be met. Additionally, we noted any specific measurements and values that must be achieved in order for the final product to operate successfully. The following subsections will detail the requirements and specifications identified as part of our goals and objectives. Both sections utilize a table to identify and describe the requirements and corresponding engineering specifications, if applicable.

2.4.1 Hardware Specifications

This section specifically details our requirements and specifications for the hardware and build design aspects of our project. These requirements have been determine based on what is needed to create a functional prototype that realizes the goals and objectives we have set out to accomplish. Additionally, we considered potential constraints such as time and finances when developing the specifications. The table below contains the written requirements we need to accomplish as well as the engineering and numerical specifications for the corresponding requirement (*Table 2-1*). It should be noted that some requirements are more general and do not have a corresponding numerical specification; regardless these requirements are equally fundamental to the overall design.

Hardware Requirements and Specifications	
Requirements	*Specifications*
Must be able to locate the position of all chess pieces on the board	1 – 32 pieces
Pieces should move at a reasonable speed	> 2 inches/second
Board moves should be completed within a reasonable amount of time	< 15 seconds
Board should be able to react quickly to user inputs	< 2 seconds
Board should be reasonably light-weight	< 10lbs
Board dimensions should not exceed standard size by more than 3 inches	< 24in x 24in
Must be able to move pieces over the entirety of the game space smoothly	64 tiles
Must be able to distinguish between different chess pieces, either with digital mapping or physical identification.	
Must be able to move chess pieces to desired location without human intervention	
Must be able to handle piece movements that involve “jumping” over other pieces	
Electromagnet must only affect the current piece being moved and should not move any other stationary pieces	

Table 2-1: Hardware Requirements and Specifications

2.4.2 Software Requirements

This section specifically details out the requirements and specifications for the software design aspects of our project. These requirements have been determine based on what is needed to create a functional prototype that realizes the goals and objectives we have set out to accomplish. We decided include our software requirements in a different section from our hardware requirements because while they do interact heavily with each other, we felt the software requirements necessitate a different approach and handling.

The table below contains the written requirements we need to accomplish as well as the engineering and numerical specifications for the corresponding requirement (*Table 2-2*). Due to the nature of the project, our software requirements have minimal numerical specifications associated with each requirement in comparison with the hardware requirements in the previous section.

Software Requirements and Specifications	
Requirements	*Specifications*
Must track all piece movements	>= 32 pieces
Must have a user interface with a desktop view	
Must have virtual representation of the physical board	
Must be able to programmatically send commands to PCB via cable or Wi-Fi	
Must have a database in order to record game moves	
Must provide opponent play decisions via web communication	
Must be able to determine the quickest path from point a to b	
Must ensure pieces do not collide with one another during movements	
Must recognize a check-mate and cease any active gameplay movements on the board until the next game	

Table 2-2: Software Requirements and Specifications

2.5 House of Quality

This section contains our House of Quality (*Figure 2-1*) which is used to show the tradeoffs between consumer requirements and engineering requirements. This tool is useful to show us, visually, the benefits and drawbacks of adding in certain functionality from both an engineering perspective and marketing perspective. The House of Quality breaks down key aspects of the project into categories and shows the interactions between them to help us create balance within our design goals.

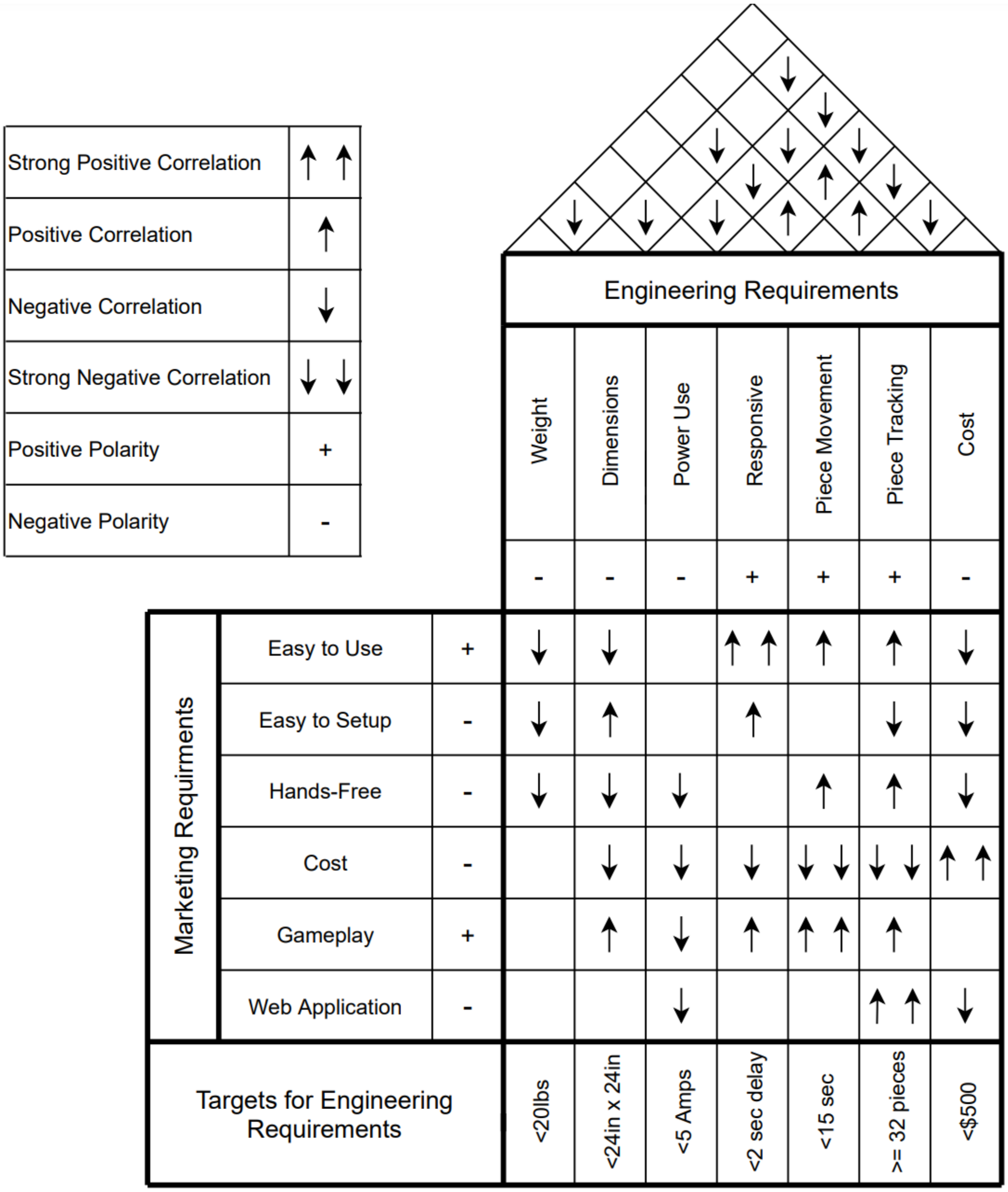
First and foremost, the Arcane Game Board is a game, therefore we must ensure that it is appealing to the intended consumer. We must compare the aspects that make this project marketable against the engineering requirements required for the project to function. We needed to consider both the basic expectations one would have when deciding to purchase a gameboard as well as the additional features that make the Arcane Game Board unique and entertaining. We have taken into account both of these aspects when generating the House of Quality diagram. The intended consumer is a typical adult without a background in a technical field (computer science, engineering, or other STEM field.) It is assumed that they are unfamiliar with both the process and tools of coding, are incapable of bypassing mechanical failures, and are disinterested in maintaining complex functions over the lifetime of the device.

We were able to determine seven major requirements from a marketing perspective. These include ease of use and setup; both of which are essential for this project. Were the final product to be too complex to use, the customer would have little incentive to choose the Arcane Game Board over a basic game board that takes little time and effort to enjoy. In addition to this, the cost of the product plays a significant role when it comes to the marketability of this project. A normal chess board does not cost much and with the prevalence of smart phones you can easily find a free app to play on, therefore, we must take into consideration the cost of this product. With the additional features of the Arcane Chess Board, we must still balance the price accordingly to ensure the purchase of such a product would be a worthwhile investment for the consumer.

Once we determine the main expectations of potential consumers, we can then analyze them against the engineering and technical requirements necessary to deliver such a product. Through this analysis, we can assess the correlation between each marketing and engineering requirement. In the House of Quality diagram, we indicate positive correlation with up-arrows and negative correlation with down-arrows. To have a positive correlation would indicate that improving one aspect would have a positive impact on the corresponding requirement. To have a negative correlation would indicate that improving one aspect would have a negative impact on the corresponding requirement. For example, if we increase the responsiveness of the board, it would have a positive correlation with its ease of use because the board would be able to more accurately imitate a real game of chess and therefore easier for the player to follow along. We make this comparison with each requirement and are able to obtain and overall polarity, indicated with a plus or minus.

Lastly, we have the “roof” of the House of Quality which indicates the correlation between the individual engineering requirements. This is particularly helpful from a design and construction standpoint as we can visually represent how each major technical part interacts with one another and the impact they have on each other. Similarly, positive correlation is indicated with an up-arrow and negative correlation is indicated with a down-arrow. For example, when we compare the piece movement against the responsiveness, we find that there is a positive correlation between the two meaning the better the piece movement is the more responsive the board is.

Overall, the House of Quality allows us to approach the project design from both a technical standpoint and marketing standpoint to deliver a well-rounded prototype. We can visually represent how our technical objectives impact our goals to help us make critical design choices moving forward.



Strong Positive Correlation	↑ ↑
Positive Correlation	↑
Negative Correlation	↓
Strong Negative Correlation	↓ ↓
Positive Polarity	+
Negative Polarity	-

			Engineering Requirements						
			Weight	Dimensions	Power Use	Responsive	Piece Movement	Piece Tracking	Cost
			-	-	-	+	+	+	-
Marketing Requirments	Easy to Use	+	↓	↓		↑ ↑	↑	↑	↓
	Easy to Setup	-	↓	↑		↑		↓	↓
	Hands-Free	-	↓	↓	↓		↑	↑	↓
	Cost	-		↓	↓	↓	↓ ↓	↓ ↓	↑ ↑
	Gameplay	+		↑	↓	↑	↑ ↑	↑	
	Web Application	-			↓			↑ ↑	↓
Targets for Engineering Requirements			<20lbs	<24in x 24in	<5 Amps	<2 sec delay	<15 sec	>= 32 pieces	<$500

Figure 2-1: House of Quality

2.6 Block Diagrams

We created block diagrams to visually represent both the hardware and software sides of the project. These block diagrams serve a purpose to provide a general idea for how each portion of this project connects to one another. The diagrams are immensely helpful for determining the division of labor in addition to how each block could potentially impact others. Additionally, by breaking down the project into smaller sections it is easier to digest and understand the overall functions of both the hardware and software. For each block diagram we include the status of the block which is indicated by the fill color and arrows indicating how each component of the project interacts with the others. Each block diagram includes a key that clarifies the meaning of each color.

The figure below, *Figure 2-2,* gives an overview of the main components of this project and shows which team members are responsible for each component. The primary lead for each part is indicated by the fill color for the block. The specific team member(s), both primary and secondary leads, working on each section is indicated by a diamond with the first letter of their name in the top left corner of the block. There is substantial overlap between each team member's skill sets and therefore many parts of this project involve collaboration between each other. It should be noted that no one member is restricted from working within other areas of the project and may assist where needed. Overall, this figure serves as a general guide for how our team plans to tackle the main points of the project and establishes how the project is divided amongst the team.

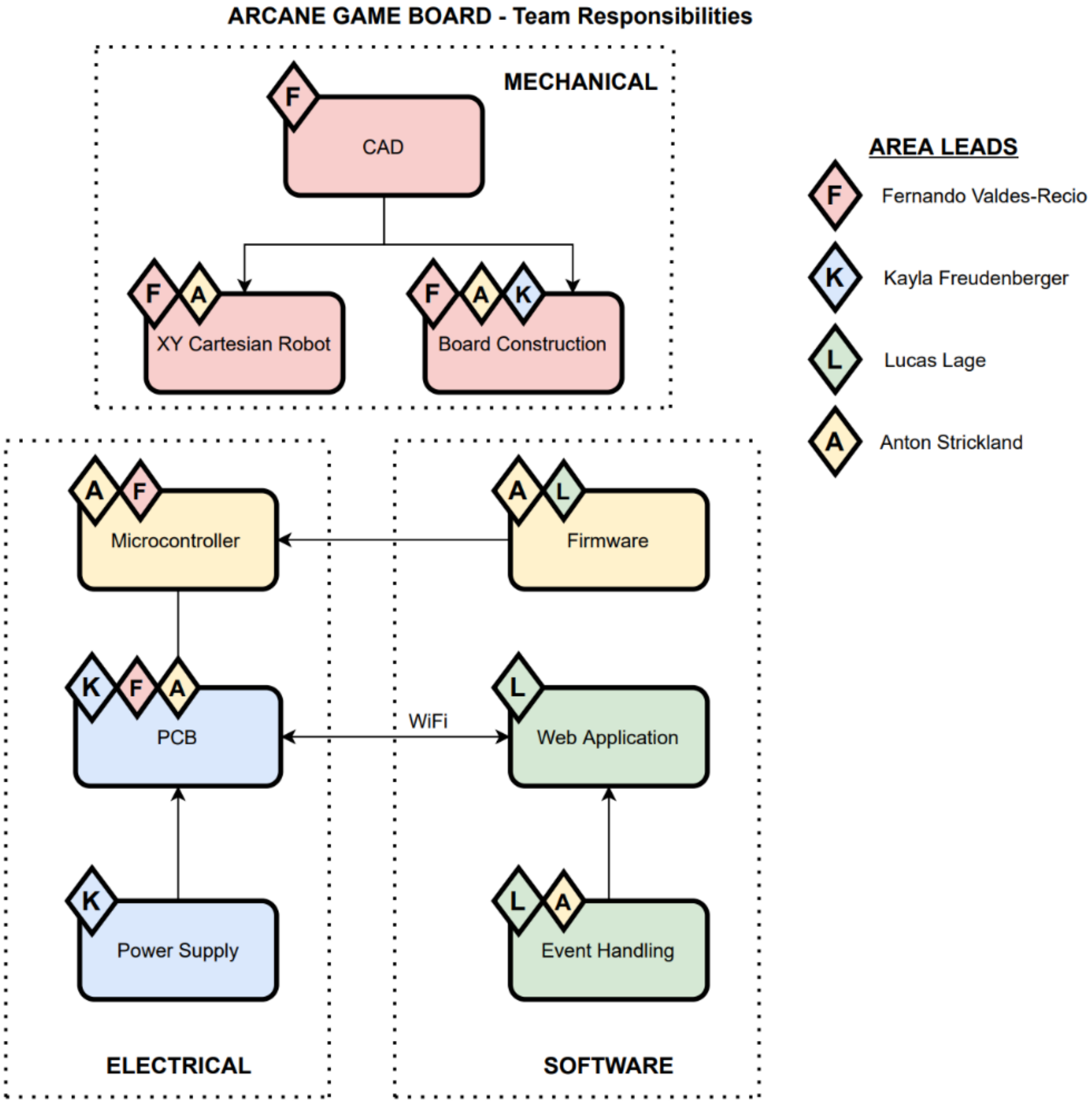


Figure 2-2: Block Diagram Team Responsibilities

In addition to the general project overview block diagram, we have created block diagrams for the hardware and software as well. *Figure 2-3*, shown below, depicts our hardware block diagram which details how the hardware components of this project connect to one another. This block diagram serves as a guide for the assembly of the prototype as well as breaking down the major pieces of the hardware for this project. This diagram indicates how the project operates from main up to where our microcontroller on the PCB interfaces with our software running on a PC.

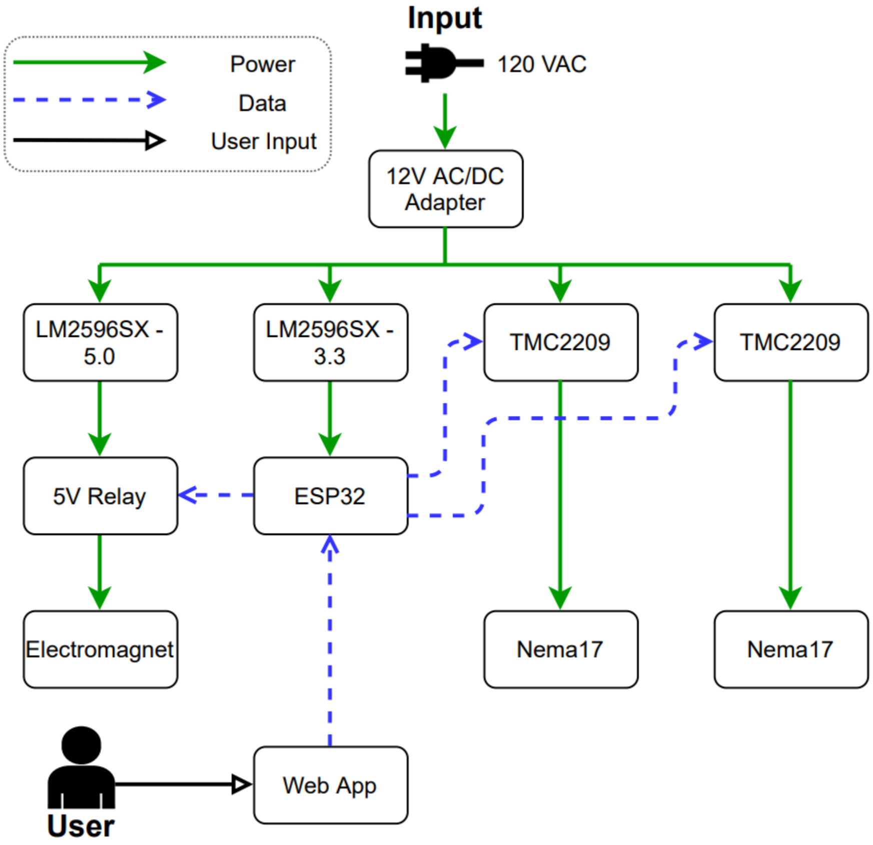


Figure 2-3: Hardware Block Diagram

The figure below, *Figure 2-4*, depicts our software block diagram which details how the software components of this project connect and function. It should be noted that the input for the software diagram differs from the hardware diagram. We have instead indicated the input as “User” since the physical board is controlled by the player who is solely interfacing with the game board through a web application. This block diagram indicates how the project operates starting from user inputs up until it reaches the microcontroller on the game board.

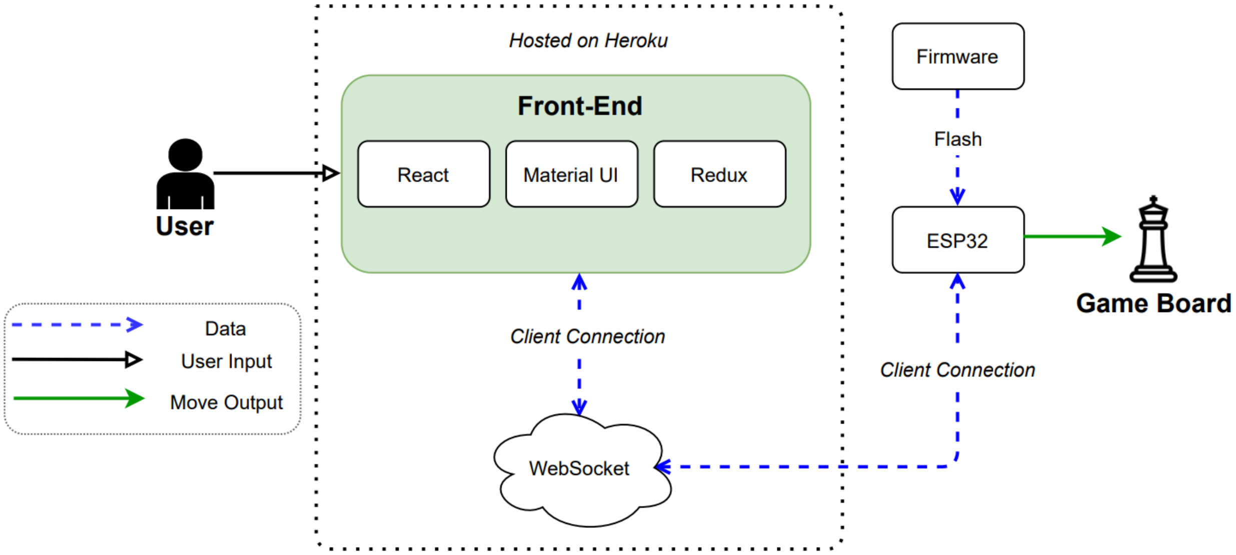


Figure 2-4: Software Block Diagram

3.0 Project Research and Part Selection

This section is meant to document and summarize the research that went into designing and building the final prototype. The research phase of this project is vital to ensure we can deliver a product that meets all goals and objectives described in section 2. The first step we must consider in our research are existing products that perform a similar function to our proposed design. This allowed us to identify places where we can improve our design as well as give us ideas for potential functionalities we could implement.

The next major step is part and technology research. It is critical that we identify the appropriate components we need to implement our design while also considering our budget constraints. Additionally, we need to research the current technology and the specifics of what is required in order to fully realize our design. Each major part of this project must be thoroughly researched so we are able to make the best decisions when it comes time to implement our ideas. In section 3.3, we will compare parts and components and indicate our selection based on the outcomes of our research.

3.1 Existing Projects

Once we established what we wanted our game board to do, we searched for projects that were similar in hopes of finding different implementations and features that could be combined and implemented for a great experience at a low cost. Chess, being a very old yet very popular board game, has been the center of many different engineering projects over the years.

One of the oldest and most iconic chess playing “machines” is The Turk. This “automated chess machine” was built in 1769 and beat many historical figures through its life including Benjamin Franklin. Sadly, it was destroyed in a fire in 1854 and its last owner later revealed that the machine was an elaborate hoax. It was not controlled by complex machinery, but rather a man hiding within the machine. Still this hoax managed to inspire roboticists for years to come as they came up with new and exciting ways of automating chess.

The two most popular and most modern ways people have automated chess revolve around ether a robotic arm to manipulate pieces, or a moving electromagnet under the playing board. Robotic arms that are suitable for manipulating game pieces are complex enough to be their own senior design project if we were to build one and are outside a comfortable price range if we were to buy one off the shelf. Thankfully, there were plenty of examples of automated chess boards that use electromagnets to manipulate game pieces online. We concluded that this approach would be less complex, more cost effective, and more practical for our application.

Two designs that stood out in particular are the Square-On and Square-Off chess boards. Below we discuss what these projects are and the key features they have that made them stand out.

3.1.1 Square On: The Magic Chess Robot

Square On is an amateur's remake of the commercial product "Square Off". Square On is a chessboard that features self-moving pieces as well as a software program that allows hands-free chess gameplay. The board utilizes an X-Y axis robot that uses an electromagnet to magnetically attach itself to individual chess pieces and drag them across the board. The mechanics of the board can be seen through the glass top in *Figure 3-1* [1]. We can also see the wire that connects to the external computer which serves as the brains for this chessboard. The board also utilizes a computer program that provides an adaptive AI in which the player can play against in the case that no second player is present. The program serves a web-facing GUI that the user can make chess moves on, and then see the pieces move to the spots selected online. The design has a clear, checkered, chess board that allows a preview of the hardware that drives the board. This design also shares the similarity of our board in that the brains behind the operation can be handled from a laptop. Unlike the Square-off board, which has all of the brains joined together on the actual physical board, both our design and the design of the Square-on board enable users to play Chess through a custom online interface, without needing to touch the pieces. The board uses a wooden housing to store all the components. This amateur approach did influence our design, by giving us a better idea of the feasibility of using an electromagnet and an X-Y axis robot.

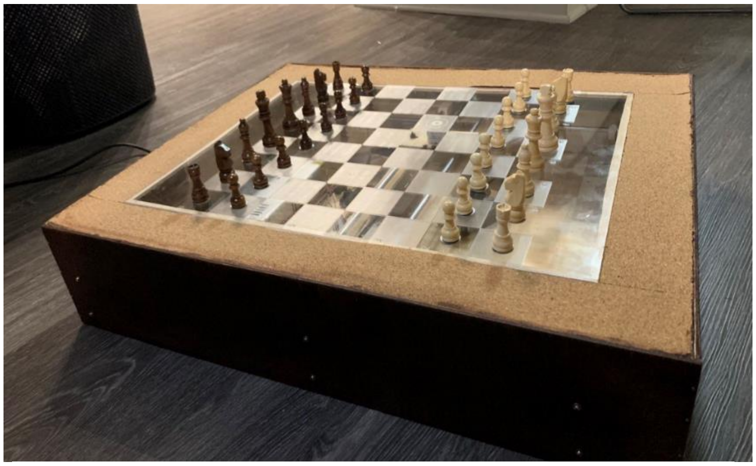


Figure 3-1: Square On Chessboard

3.1.2 Square Off: The World's Smartest Chessboard

Square Off is an automatic, self-moving chessboard that allows for hands-off gameplay for chess. Square Off is a commercial product, and sells for either $399 or $449, depending on the model that you select [2]. Square Off offers several features like those proposed in this senior design project with the main feature being that the pieces are self-moving. Our design was inspired by that of Square Off, in the way we plan on moving pieces across the board without human contact. The premium model (*Figure 3-2*) offers a series of software features and assists that really make this a smart board. These software features have a served as an inspiration for some of the software features that we would like to implement in our application, mainly the adaptive AI feature, that provides an AI for the main player to play against in the case that only one player is available for a game. Square off utilizes an on-board computer which is controlled through an application. This differs from our design which is intentionally set up to run on a laptop in order to enable off board playing through the laptop/desktop user interface. While our design shares some similarities with the Square Off boards, we originally designed our board without any influence from Square Off, and will create a board that differs in aesthetic, feature set, and construction.

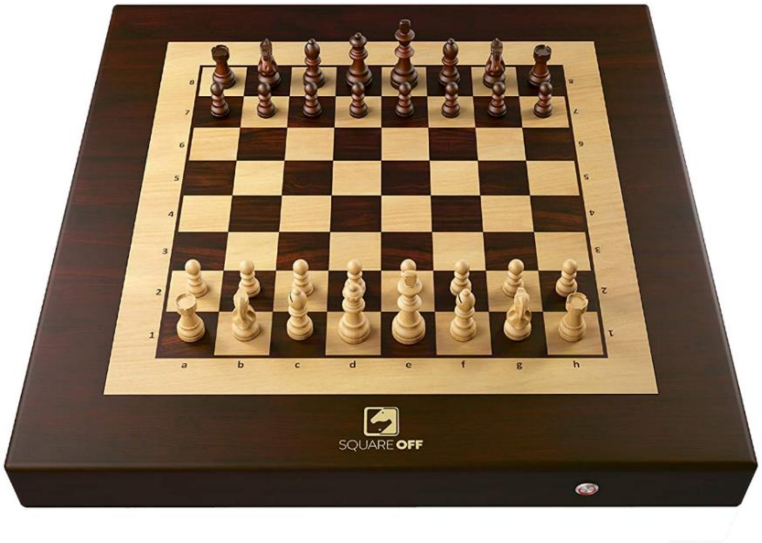


Figure 3-2: Square Off Chessboard

3.2 RELEVANT TECHNOLOGIES

Many technologies already exist which are relevant to creating the game board. These generally fall into 2 categories: hardware and software. Movement systems for the robot itself have 3 popular methods which should be considered. Existing open-source firmware for 3d printers and other robots exists which could streamline the process of sending commands from the controller. Software libraries are the most numerous and readily available resource at our disposal which could be used to create a comprehensive Windows based app to actually play the game through.

3.2.1 Mechanical Components

The robot used to move the pieces will need to be compact, reliable, and able to be precise to a millimeter. Movement systems for 3d printers give 3 main design families to choose from, with the code for each requiring unique calculation and firmware from the on-board microcontroller.

All of the motors, linear rails, and belts described in later sections are compatible with any of the design families.

3.2.1.1 X & Y Movement Mechanisms

All of these systems are capable of navigating the “head” of the robot to a specific point on a 2d plane. Dimensional efficiency, speed of navigation, and reliability are the direct consequences of whatever movement mechanism is chosen. Due to the mechanics at play, the firmware will also be impacted by the design implemented. Delta robots in particular are more difficult to calculate linear movement, while linear stages are most simple. The number of motors needed will also vary depending on the movement system chosen – impacting the overall cost of the project.

3.2.1.1.1 Cartesian Linear Stage

Linear stage robots consist of 2 motors, each attached to a separate axis. One of them travels with the axis which it controls, requiring that control and power wires move with the motor itself. Most linear stages on this scale use 1 of 2 basic designs: belt or chain stages, and lead screw stages.

Belt (or chain) stages operate by wrapping the titular belt around a gear and apply tension. A stepper motor spins the gear, causing the belt and affixed carriage to travel. As the belt pulls the assembly with it, a rail or guide rod ensures that the carriage maintains its orientation while moving. This can take the form of wheels in grooves, a linear rail with cyclic ball bearings, or a smooth rod with cylindrical bearings connecting it to the main assembly. Belts and chains do wear out more often when compared to the lead screw method detailed below. Materials such as rubber can also stretch when under high load, leading to inaccuracy in the stage during high speed (>100mm/s) movements. *Figure 3-3* illustrates a slotted V-Wheel linear stage using a belt assembly.

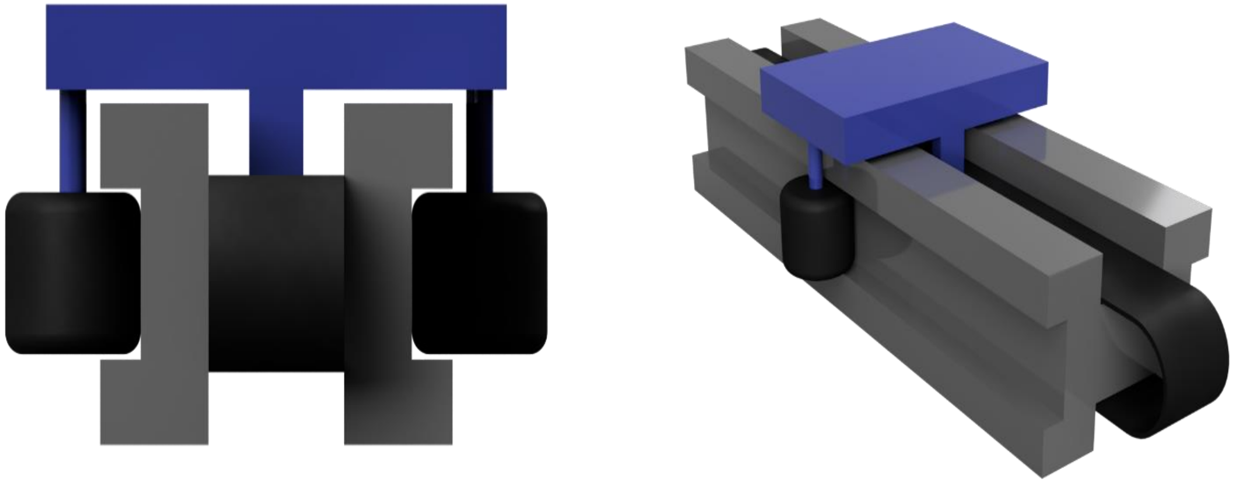


Figure 3-3: Mock-up of Basic Belt Linear Stage Assembly

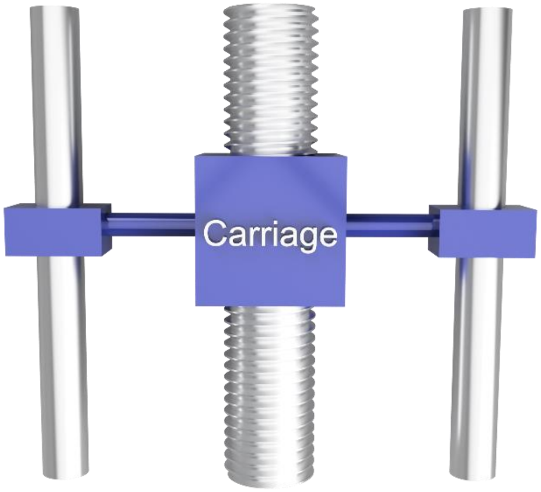


Figure 3-4: Example of a Lead Screw Linear Stage

Lead screw stages utilize a screw fixed to a motor with guide rails fixed parallel to the screw. Lubricant is usually applied to the guide rails and screw to ensure smooth movement, with ball bearings being used exclusively on the guide rails. This is due to the lack of threading – which makes ball bearings on the screw assembly impossible. There should be as little friction as possible on the guide rails in order to reduce load on the motor driving the lead screw. The guide rails maintain the orientation of the carriage while the screw turns in the threaded mount, resulting in movement. See the figure to the right, *Figure 3-4*, for a basic example. Due to the precision of screws and durability of the metal which makes up the rod, this approach is commonly used in stages where reliable movement must be executed under load. Speed is often a limiting factor though, with the stepper motors requiring many rotations to travel small distances when compared to the belt or chain approach.

As previously mentioned, the power and control wiring for at least 1 axis must travel on the carriage of another axis. This introduces a point of failure which could render the machine inoperable without user intervention. If the wire is soldered in – as opposed to plugged – it could even lead to tears which a standard user would be unable to fix themselves. Many designs implement wire chains which mitigate this type of failure.

When controlling linear stages, it is very simple to execute commands. Each motor has the movement-per-step defined and stored in memory which is used to calculate the number of steps required to move to a location. Since each motor is entirely responsible for the axis it manipulates no other math is required to coordinate the motors. For example, if the robot is instructed to move 3mm in the x direction and 7mm in the y, the control unit then executes the number of steps required to move 3mm for the x motor while simultaneously executing the number of steps on the y motor. When observing the robot this appears as diagonal motion followed by motion solely in the y direction. If the robot wants to travel in a diagonal other than the 45-degree offset from any given axis, the motors can be directed to spin at varying speeds.

3.2.1.1.2 Delta

Delta movement systems consist of 3 motor-driven carriages affixed to 3 parallel rails – forming a triangle (delta) shape, see *Figure 3-5*. When one of the carriages moves, it pushes or pulls the head of the robot with it. This approach leads to high speeds and accuracy relative to a cartesian robot, but the drawbacks make it inappropriate for this project.

The speed of the delta robots comes from the weight of the moving parts. Unlike the cartesian linear stage robots which must move at least 1 motor, delta robots do not require any motors to be moved. Lightweight materials like plastics, carbon fiber, and aluminum reduce the weight even further, making the heaviest moving part the actual head of the robot. This allows the motors to work very quickly without losing accuracy. [3]

Another benefit of this design is the simplicity of the wiring. As the motors are fixed on the frame of a delta robot there is no chance of the wires moving and being snagged unless they are improperly arranged or shaken loose of their anchors. The only moving wire would be the power source and control signals to the head of the robot, which should not have any obstructions near it anyways.

While the approach does offer these benefits, it does come with drawbacks. The largest of which is that they require more vertical space than cartesian or CoreXY designs. Since the movement comes from the attached arms and their position relative to one another, there will always be more vertical (Z) travel than horizontal (XY.) Our design constraints call for a compact design which is incompatible with this approach.

Additionally, the movement of these machines is more computationally complex. In order for linear movement to occur it must be translated into new positions for each motor. The math is repetitive but has many more points of failure when translated into mechanics. Particularly as linear motion is achieved through breaking the line into segments, then quickly running the motors through those positions in sync. If an error is present, it would take significantly more effort to identify at what stage it is occurring, or even which motor(s) is/are behaving incorrectly. [4]

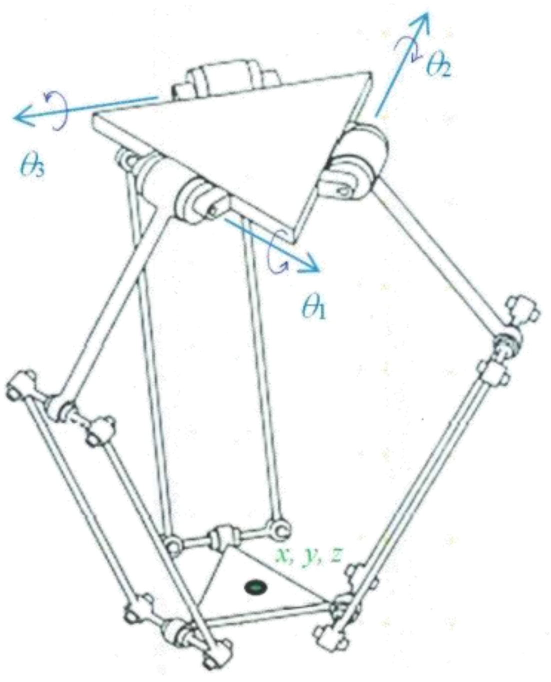


Figure 3-5: Delta Robot Diagram [25]

3.2.1.1.3 CoreXY

CoreXY is like a hybrid of Cartesian and Delta systems. Much like the Cartesian systems, CoreXY tends to run on linear systems such as belt and pully or linear rails and linear bearings. It also has a rectangular movement space, unlike the Delta systems, and does not require much more vertical space to operate. However, much like the delta systems, multiple motors contribute to moving the end effector of the system.

The two main drawbacks for the system are the increase in hardware such as the additional pullies and longer belts, as well as the non-intuitive kinematics of the system making it harder to control than the cartesian system, but easier than the delta system.

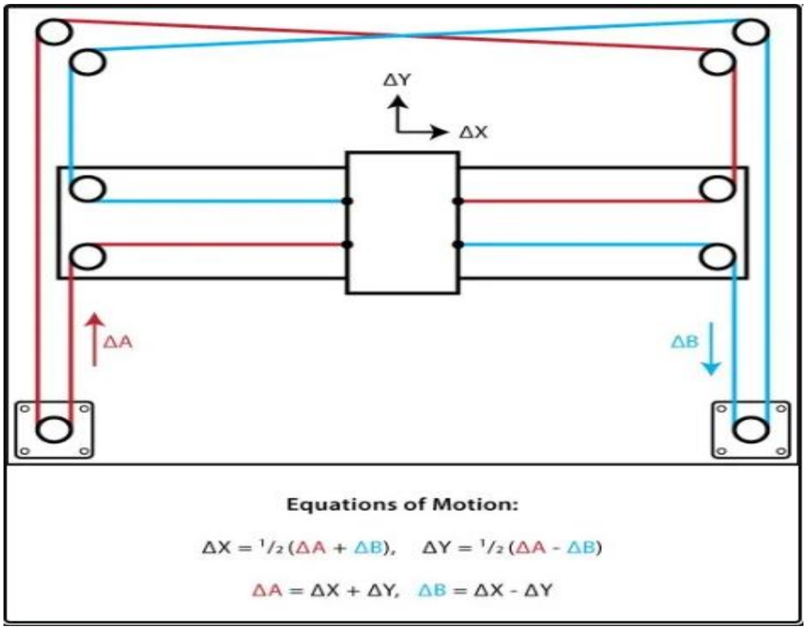

ΔY
ΔX
ΔA
ΔB
Equations of Motion:
$\Delta X = {}^{1}/_{2}(\Delta A + \Delta B), \quad \Delta Y = {}^{1}/_{2}(\Delta A - \Delta B)$
$\Delta A = \Delta X + \Delta Y, \quad \Delta B = \Delta X - \Delta Y$

Figure 3-6: CoreXY Setup and Equations

The above image, *Figure 3-6*, is an image of a typical CoreXY setup as well as the kinematic equations that would be used to position the end effector. Notice that the system as 2 district belts, noted by the red and blue color, and that these belts cross. This crossing of belts is what calls for an increase in vertical space so that the belts can run on 2 different planes and swap planes at the crossing point. The crossing of belts does bring and advantage which is that it eliminates a torque vector which would put less strain on the linear movement frame when the end effector moves on the X axis.

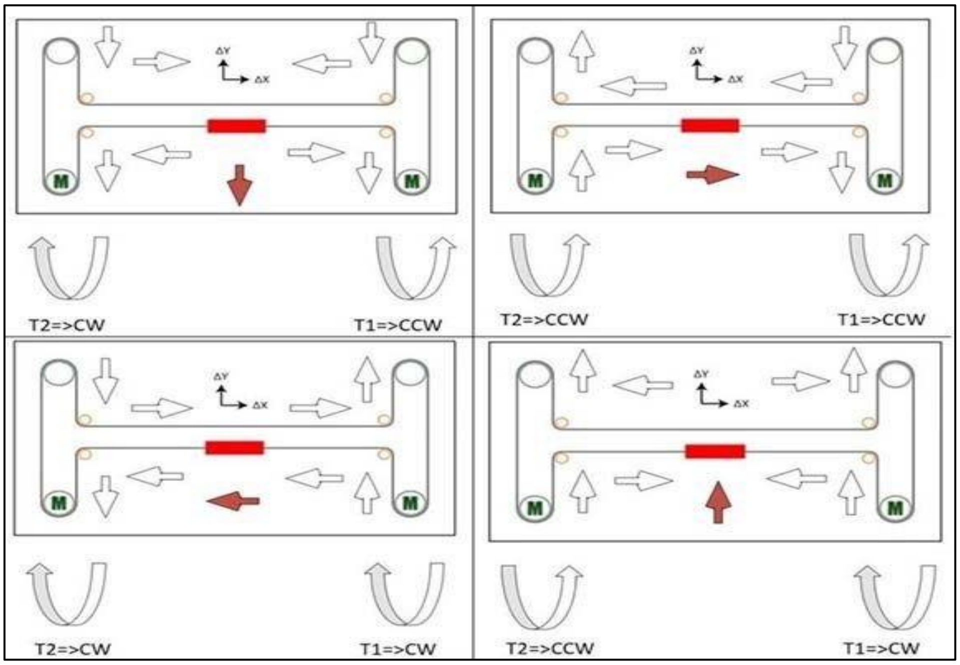


Figure 3-7: Hbot Setup

After a bit more research, we came across a CoreXY variant called Hbot which you can see above in *Figure 3-7*. Notice the lack of belt crossing. This brings a host of advantages such as functioning on a single plane removing the height increase requirement, less hardware since 2 of the pullies in the original CoreXY are no longer required, and a single continuous belt instead of 2. This system follows the same kinematic equations as the original CoreXY and the only disadvantage is there is increased torque on our linear system when moving along the X axis. We found that with a sturdy enough linear system, this additional torque is not an issue.

After considering our house of quality and our target requirements, we decided that the H-Bot configuration would be the best mechanism to move our electromagnet end effector. It has an extremely efficient footprint with its in-line single plane construction and has a relatively small hardware requirement.

3.2.2 Microcontroller

Since there are so many hardware and software components in this project, we wanted to choose a microcontroller that would be the least amount of headache to work with while getting what we need done. Our team has far more experience with the Arduino IDE over Code Composer Studio, and PlatformIO (PIO) is mostly platform agnostic, so we narrowed our microcontroller selection to devices that can be programmed in and are compatible with either Arduino or PIO. The next factor we considered was extensive documentation and project use within the maker community, as it would be easier to troubleshoot issues on a well-known chip than an obscure one. With these two factors in mind, we considered the ATMEGA2560-16AU used in the Arduino Mega, the Cortex-M7 used in the Teensy 4.1, and the ESP32 – WROOM – 32D used in the ESP32S developer boards. All of which feature extensive documentation and support for projects of similar complexity to this project.

3.2.2.1 ATMEGA2560-16AU

This is by far the most popular of the three previously mentioned microcontrollers. This microcontroller is at the heart of the popular Arduino Mega 2560 board and is considered a darling in the hobbyist maker space. There are thousands of well documented projects that use this board available online that use this board. This makes it easy to find robust methods of communicating with many different pieces of hardware and there is plenty of available source code that shows how to use the many features on this board.

The board specs are as follows:

- Architecture: 8-bit RISC
- Instruction set length; 135
- General purpose registers: 32 x 8-bit
- Flash memory: 256kB
- EEPROM: 4kB
- SRAM: 8kB
- Timer/counter: 2 x 8-bit
- ADC: 16-channel 10-bit
- GPIOs: 86

- Package: 100 lead TQFP
- Throughput: 1 MIPS per MHz
- Max oscillator frequency: 16MHz

There are several cons to using a board like this on our project, but the two most glaring are the 8-bit architecture and the relatively slow max clock frequency of 16MHz. Since we are using a CoreXY system, our positional accuracy is very susceptible to our mathematical accuracy. The final positional resolution is directly tied to our microcontroller's mathematical resolution. This 8-bit board will not be able to get the same accuracy as the other 2 32-bit boards we considered for this project.

Accuracy aside, we intend to use parts of the open-source 3D-print firmware, Marlin, to handle our electromagnet movement. Using parts of this code would significantly cut down our time and effort writing positional movement code; however, this firmware is meant to run natively on 32-bit processors.

3.2.2.2 ARM Cortex-M7

This is the least popular of the three microcontrollers listed in the maker space. Its most popular development board is the Teensy 4.1 which is quickly gaining traction. It has by far the fastest processor of the microcontrollers we considered, but it also comes at a much higher price.

The board specs are as follows:

- 600 MHz core clock
- Float point math unit, 64 & 32 bits
- 7936K Flash, 1024K RAM (512K tightly coupled), 4K EEPROM (emulated)
- QSPI memory expansion, locations for 2 extra RAM or Flash chips
- USB device 480 Mbit/sec & USB host 480 Mbit/sec
- 55 digital input/output pins, 35 PWM output pins
- 18 analog input pins
- 8 serial, 3 SPI, 3 I2C ports
- 2 I2S/TDM and 1 S/PDIF digital audio port
- 3 CAN Bus (1 with CAN FD)
- 1 SDIO (4 bit) native SD Card port
- Ethernet 10/100 Mbit with DP83825 PHY
- 32 general purpose DMA channels
- Cryptographic Acceleration & Random Number Generator
- RTC for date/time
- Programmable FlexIO
- Pixel Processing Pipeline
- Peripheral cross triggering

This microcontroller is by far the fastest microcontroller considered for this project going off its core clock. It is much more precise than the ATMEGA2560-16AU and offers more flash and RAM as well. Though it does consume more power than the ATMEGA2560-16AU, power use can be mitigated by entering and exiting low power modes. The key flaw for this and the ATMEGA2560-16AU is the lack of any Bluetooth or Wi-Fi communication. Those radio features are not exactly common among controllers, but given that our project would greatly benefit from having them available, we continued searching for a microcontroller that may already include them. If we did not find a microcontroller with Wi-Fi or Bluetooth, we would likely select this board and have to communicate through an external Bluetooth or Wi-Fi module which could further increase cost and complexity for our PCB.

3.2.2.3 ESP32 – WROOM – 32D

The ESP32 is the second most popular microcontroller in the makerspace among the three boards considered in this paper. As IOT (Internet of Things) have become more and more popular, so had this microcontroller. This is largely to do with the fact that this microcontroller has a Bluetooth and Wi-Fi module onboard. There are hundreds of projects documented online using this board making it easy for us to figure out how to leverage the Wi-Fi and Bluetooth capabilities. This would be important because it would save us time designing our PCB by eliminating the need to implement a USB connection. Alternatively, it saves us some money because we would not need to add a Wi-Fi or Bluetooth module to our PCB like we would if we decided to use on of the other microcontrollers.

The board specs are as follows:

- CPU: Xtensa dual-core 32-bit LX6 microprocessor, operating at 160 MHz
- Ultra low power (ULP) co-processor
- Memory: 520 KiB SRAM
- Wi-Fi: 802.11 b/g/n
- Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
- 12-bit SAR ADC up to 18 channels
- 2 × 8-bit DACs
- 10 × touch sensors (capacitive sensing GPIOs)
- 4 × SPI
- 2 × I²S interfaces
- 2 × I²C interfaces
- 3 × UART
- SD/SDIO/CE-ATA/MMC/eMMC host controller
- SDIO/SPI slave controller

After careful consideration, it was decided that this board would be the best fit for our project. This board seemed, to us, to be a middle ground between the ATMEGA2560-16AU and the ARM Cortex-M7. Its precision, memory space, clock speed, and cost sits squarely between the two other considered boards. The icing on the cake for this board is its integrated Wi-Fi and Bluetooth module. We would not need to worry about getting a microcontroller to communicate with an external module to get these features and we would not need to add complexity, and thus, further points of failure on our PCB by including and external radio module.

3.2.3 Power Technologies

The power system is absolutely fundamental to any electronic device and there are many different ways we could supply power to our device. One of the advantages our project has is that the design is stationary and is typically played indoors which means we can have a wired connection to main in order to power our device. However, there are disadvantages to this as well and we must consider other power alternatives and weigh them against our overall goals and requirements before we can come to a final decision.

In this section, we will be exploring two different methods for powering our device, namely batteries versus connecting directly to main. There are advantages and disadvantages to both and we will explore both of these in order to make the best decision in terms of our overall design goal.

3.2.3.1 Batteries

Batteries are commonly used to power many present-day electronics and for good reason. Batteries have the capability to convert chemical energy into electric energy which is then released to power devices. The overwhelming advantage is that they can be used in portable devices and continue to power the device on-the-go without needing a wired connection. There are two main types of batteries: Primary batteries and Secondary batteries. Primary batteries are single-use meaning they can only be used one time and then discarded. The reason they are single use is because during discharge, the electrode materials are changed, and this change is irreversible. Conversely, secondary batteries can be used more than once since they are rechargeable through using an applied electric current [5].

Both types of batteries positively affect the portability of a device. However, battery powered devices do need to be either replaced or recharged regularly in order to continue being operable, which can be a draw-back. This disadvantage is most notable when using primary batteries because the consumer would be financially responsible for replacing these batteries and they would be unable to use the device until a replacement is put in. In regard to secondary batteries, needing to consistently recharge a device can be a burden and is substantially disappointing if consumer were to forget to charge it before playing.

Overall, we must determine the importance of portability from both an engineering and marketing perspective for this project as batteries with have the most significant impact on portability. Also, the main scenario in which opting for our project to be battery powered would substantially outweigh a wired connection to main is in circumstances where there is no outlet available to hook up to. Through our research we've determined that while this circumstance can happen, it is relatively unlikely. We have determined that the marketing impact of having this project be battery-powered or otherwise is not particularly significant for our project overall; meaning in would neither help nor hurt our overall design.

3.2.3.2 Mains Electricity

Our second option for our project's power system is mains electricity, also commonly referred to as 'household power' or 'wall power'. Main's electricity is an alternating current electric power supply commonly used to power appliances such as microwaves or televisions by plugging into a wall outlet. This electric power is delivered to homes and businesses through established electric infrastructure and is widely available throughout the developed world. The two main properties of this form of power supply is the voltage and frequency; this varies by region but in North America it is commonly 120V and 60Hz [6].

The most notable advantages of using a direct connection to main is the ease of implementation as well as the ability to power the device anywhere as long as there is a working outlet. However, despite the abundance of outlets in most modern establishments, this consumer would be limited to these spaces only in order to use this product. This means that playing outside or playing in the center of a room, provided they do not have access to an extension cord, would not be possible. This limits the portability and versatility of the project which could potentially affect the appeal of the product from a marketing perspective.

However, the nature of this product leans heavily towards being a primarily indoor device. Through research and experience, it is less common for a chessboard to be taken out of the household meaning that the portability of this product is not a fundamental feature from a marketing perspective. Nowadays there is an abundance of outlets in homes and businesses, therefore, designing this project to have a wired connection to main would impact the product and appeal very little. Granted, we must consider that typically board games are played in the middle of a table so the power cord should be long enough to accommodate this within reason.

3.2.3.3 Voltage Regulators

We are supplying the board with power via mains and we need an adapter to convert the standard 120VAC to a useable DC voltage. The resulting DC voltage is the input to our PCB, however not every component requires the same amount of voltage. We must very carefully evaluate the proper operating voltages and currents of each electrical component and ensure we can safely operate our game board. To achieve this, we need to select the proper voltage regulators needed and include them in our PCB design. This stage of design is massively important because if we are not careful with choosing our voltage regulators, we could damage our components and end up with a compromised end product.

This section will discuss our potential options for voltage regulators. More specifically, this section will detail the differences between linear and switching voltage regulators. In addition, we will explore the advantages and disadvantages of both and what these mean in terms of our design and operation. Using this information, we can make the best decision when it comes to the final PCB design.

3.2.3.3.1 Linear Regulators

Linear regulators are relatively simple and are particularly advantageous in low powered devices. Linear regulators utilize a feedback loop which generates more current into the base of the transistor. This results in a reduced voltage drop across the transistor and consequently a reduced output voltage. Basically, linear regulators simply require an input voltage which is some small amount higher than the desired output voltage in order to function.

While linear regulators are extremely simple and cheap to implement, they are also relatively inefficient and burn a lot of power. The difference between the input and output is continually dissipated as heat, resulting in a linear regulator heating up very easily. This would require fairly significant heat sinking on our board to adequately combat and keep the regulator from overheating. In terms of efficiency, linear regulators are only most efficient when the input voltage is very close to the desired output voltage [7]. Unfortunately, our design requires, at minimum, a 3V difference in voltage which is much too large to get any substantial efficiency boost in comparison to a switching regulator.

3.2.3.3.2 Switching Regulators

Switching regulators are more complex than linear regulators, however this additional complexity means that a switching regulator is also significantly more efficient. Switching regulators are able to temporarily store input energy and release that energy to the output at a different voltage level. These regulators have very little power loss unlike the linear regulators and are also much more versatile. The main reasons that switching regulators are used is for their high efficiency and low power dissipation as well as being relatively small inside.

Switching regulators, despite being more complicated than linear regulators, are still considerably simple to use. The main drawback a switching regulator has as opposed to linear regulators are that they tend to be more expensive. Despite the slight increase in cost, we decided to use switching regulators in our final design due to their high efficiency and versatility.

3.2.4 Communication

For the board to function over long distances some sort of wireless communication will be necessary. This can take the form of a wired or wireless interface with an internet-capable device, or by connecting the on-board MCU directly to the internet itself. For each approach there are drawbacks and introduced challenges which must be understood in order to develop a working product.

3.2.4.1 Wi-Fi

Wi-Fi is the standard internet connection for many devices due to its prevalence in everyday life. The vast majority of users will have a network available in their homes, allowing for the board to connect to the web directly without relying on the connection to the computer running the UI. This would also allow for expanded functionality later such as a board which plays out games being streamed by other users or removing the UI component entirely in lieu of a piece tracking system.

Several MCUs come with built-in support for Wi-Fi connection already, making the impact on PCB design negligible. The rise of IoT devices requiring an internet connection has risen quickly in the past years; this has led to the development of not only the Wi-Fi capable MCUs themselves, but many open-source codebases on how to properly connect them to existing Wi-Fi networks.

The proposed design of checking and modifying a database to control the gameboard does not require a router capable of <1Mbs transmission, meaning that the protocol itself has more than enough data bandwidth for our needs. Response time will not be greatly affected if using Wi-Fi as well – improving user experience and granting more time for processing movement commands if needed.

There are drawbacks to using an on-board Wi-Fi module. The largest is connecting to secure networks – as without a more robust interface allowing text input there is no way to load new networks onto the board. Potential workarounds include using an alternative connection to login to the network during setup. These workarounds introduce complexity to both the design process and the user experience which may not be welcome compared to the simplicity of the other types of connections.

3.2.4.2 Bluetooth

Bluetooth connection would allow for the on-board MCU to receive updates from an intermediate device rather than from the web itself. Many variants of Bluetooth exist, each with its own advantages and disadvantages. If this communication is to be used in any form it will be important to verify that the hardware connected to the MCU is capable of supporting the desired features and specification.

Most MCUs support at least Bluetooth 4.2 due to the features offered for IoT devices. [8] This specification allows for communication with routers using the IEEE 802.11 standard to stream and pull data from the HTTP servers. An example of this could be a smart home's door camera, a field device's sensors, or a smart watch's health trackers. All of these require different bandwidths which are supported by the protocol as well – which will allow us more freedom when finalizing the distribution of computation.

In order to utilize this technology a method for users to connect to the board would need to be established and documented in such a way that non-technical users would be able to understand it. Due to the prevalence of Bluetooth devices in everyday life, a phone app or browser-based solution would be apt – increasing development time significantly. An alternative to this would be a Bluetooth serial connection terminal app, which would allow for development and testing to be unimpeded. This would make the user experience worse at the cost of development simplicity.

Some MCUs utilize the same radio systems for Bluetooth and Wi-Fi. This is a key constraint when considering which hardware to purchase and how to develop software which can run on the physical device. In order to circumvent this there are 2 immediate options available.

The first being a Bluetooth 4.2 capable device to connect to an internet capable device and bind all activity to a single HTTP server through. This limits bandwidth and moves more processing onto the server itself but allows for a simpler approach to connectivity. On startup the device would broadcast a pairing request until a user connects an internet-capable device such as a smart phone or laptop. This approach is illustrated below in *Figure 3-8*.

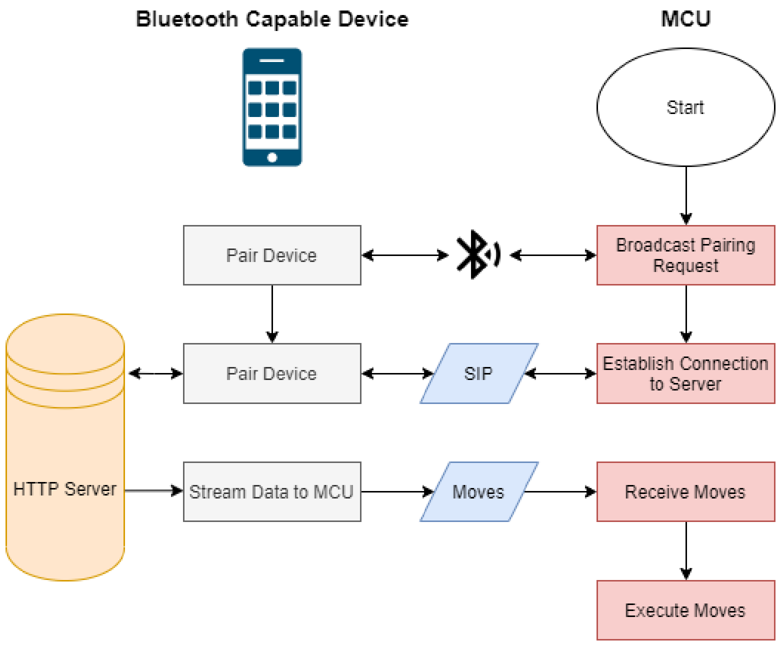


Figure 3-8: Bluetooth 4.2

The second option, shown in *Figure 3-9*, is to integrate a pairing button to the board itself, which would switch the on-chip radio to switch to Bluetooth mode in order to receive Wi-Fi credentials. This method introduces design features to be incorporated but offers larger bandwidth and removes the need for a paired device to stay near the board. If this method were to be used settings could be saved onto the memory of the MCU, removing the need to reconnect on startup once the initial setup is complete.

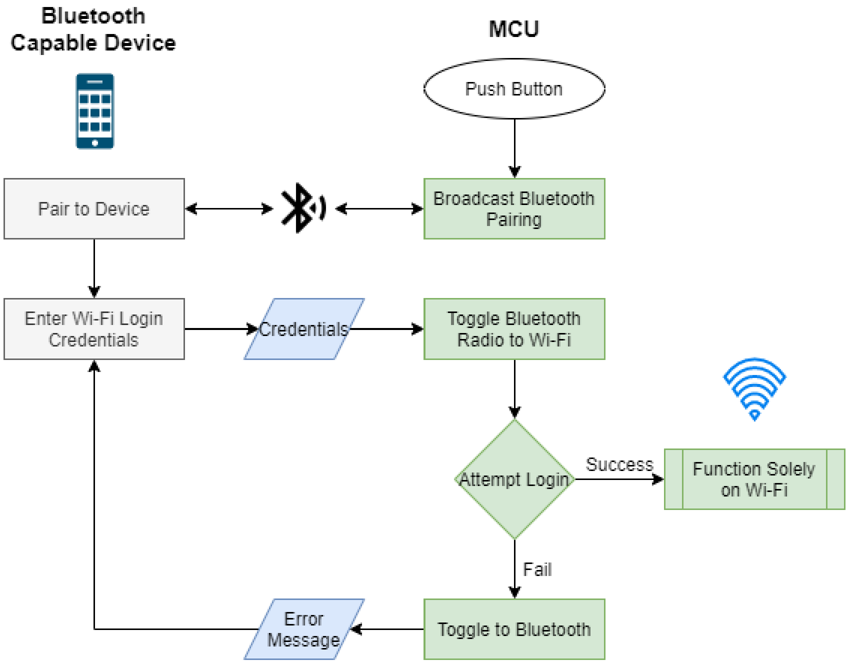


Figure 3-9: Bluetooth with Pairing Button

Unfortunately, due to time constraints, the Bluetooth functionality of the board was not implemented. The theory behind its use remains sound however – and there is more than enough overhead in memory and computational capacity to implement it at a later date.

3.2.5 Software Tools

Several embedded projects already exist which served as jumping-off points for our firmware design. While neither Marlin nor Klipper ultimately suited our needs, Marlin was a strong enough contender that over 2 months of development was put into a Marlin-based solution before being abandoned for an in-house firmware stack. The final stack still took reference from Marlin, but shares no code base with it due to the complexities and additional features which Marlin offers.

3.2.5.1 Marlin Firmware

This is an open-source codebase which has been in active development since December of 2014. While much of the code is overly complex for the purposes of this machine, it does contain movement translation functions for all of the movement systems described earlier in section 3.2.1. It already supports a wide variety of MCUs, which is what made it catch our eye and attempt to adapt it to our own purposes.

The math behind a CoreXY and Delta robot in particular are error prone and using an existing library would save time and wasted effort. Both of these movement systems are supported within C++ files in the main branch which translate linear instructions to either movement system at runtime. Allowing for the external controlling app to send simple linear commands and not have to encode the motor actions.

Several key features are missing from Marlin which would have needed to be written to meet our MVP. These could have been added off- or on-board, likely decided by the computational complexity and memory size. Path planning around a stationary piece is one example. On a 3d printer all path planning is done in the slicing phase – meaning that base marlin only executes movement commands directly. If the on-board MCU has the state of the board saved in memory it could be possible to add an obstacle avoidance check and function which would allow pieces to move to any square or be retired without colliding with other pieces. When compared to the simple functions we developed this was simply not worth the trouble just to get semi-circles while avoiding other pieces.

Marlin also includes many features and code which would be irrelevant to the operation of the robot. While it is intended to run on MCUs with only MBs of memory, it does take up to 16MB to install properly. If all of this code is present on the Arcane Game Board it may impose memory restrictions on any code which needs to be added and run to suit our purposes. An alternative would be to cannibalize the movement functions, the command interpreter, and other features we deem necessary while we develop our own core structure. This would also ensure that the machine does not behave unpredictably while interacting with functions typically used to enhance print quality.

MCUs running Marlin are intended to run GCODE which has been precompiled, or take commands from an external system. This means that the off-board app would need to generate GCODE commands to execute the entire move from start to finish, and then deliver it to the MCU via a communication channel. Increasing the complexity on the off-board computation is a necessity with this system due to the memory and computational requirements of event handling. If it becomes necessary to handle some events on the board, the movement capabilities of Marlin could still be overhauled to fit into a new system. The figure below, *Figure 3-10*, depicts the flow diagram of an MCU running Marlin firmware

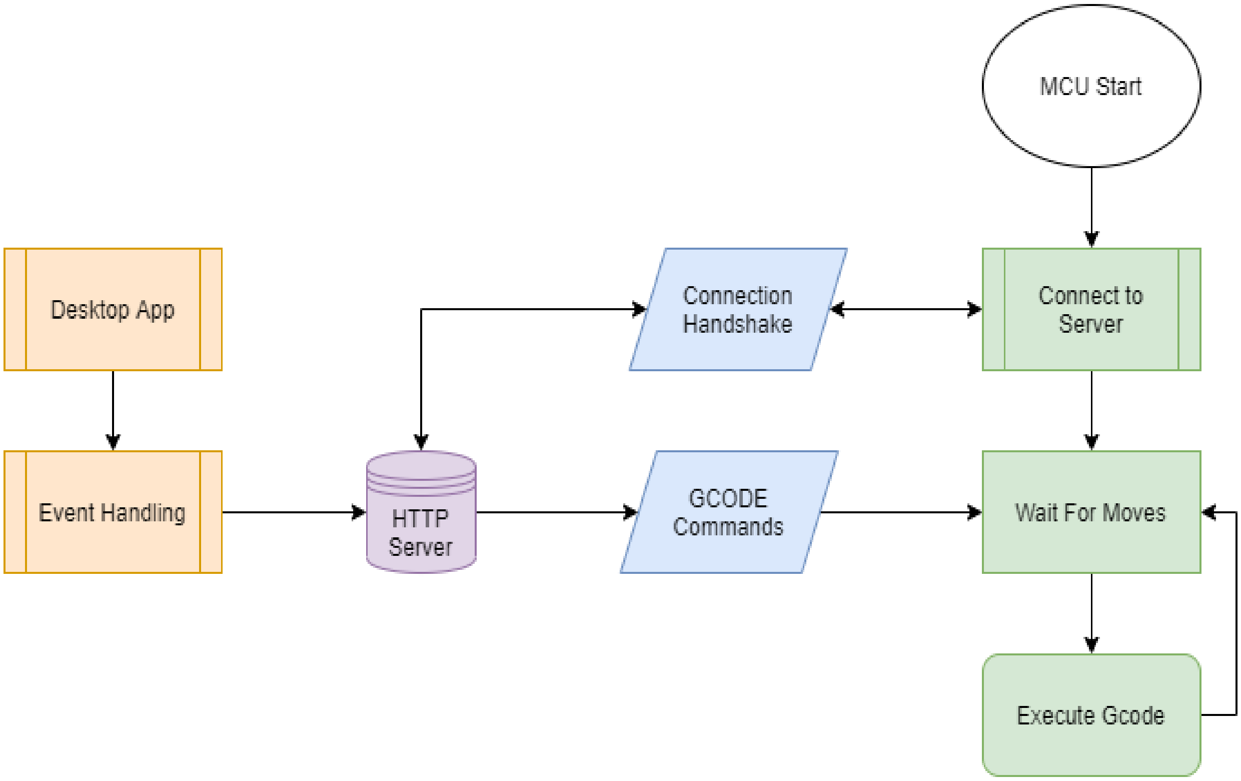


Figure 3-10: Flow Diagram of an MCU Running Marlin Firmware

3.2.5.2 Klipper3d Firmware

Klipper is another 3d printing firmware project which has all of its code available under GNU General Public License. It runs on a 64-bit application processor such as a raspberry pi which is capable of connecting to a wireless network. The user can then use a SSH utility to remotely operate not only the application processor, but any microcontrollers attached to it (usually 1-2.) The firmware then interprets complex data inputs from external sources, compresses it into a stream of movement commands, and then sends it over low-bandwidth channels to MCUs in a slave configuration. All of the code written for Klipper is in Python as well, allowing programmers to utilize the many libraries which exist in the language.

Event handling would be the most useful feature to take from this project. Since the device running the master firmware uses a 64-bit controller and has access to the power of a consumer laptop computer it will be able to handle much more complex calculations faster. Klipper then converts the resulting commands into a stream of data which is capable of being transmitted using relatively low-bandwidth protocols such as UART or Bluetooth. When the robot must avoid other pieces, calculate maximum appropriate speeds, or interpret the commands of the chess engine Klipper3d would save time and data bandwidth. The Flowchart below, *Figure 3-11*, shows this, with the green shaded portions containing processes or subprocesses which would be handled by a control unit running Klipper3d derived firmware.

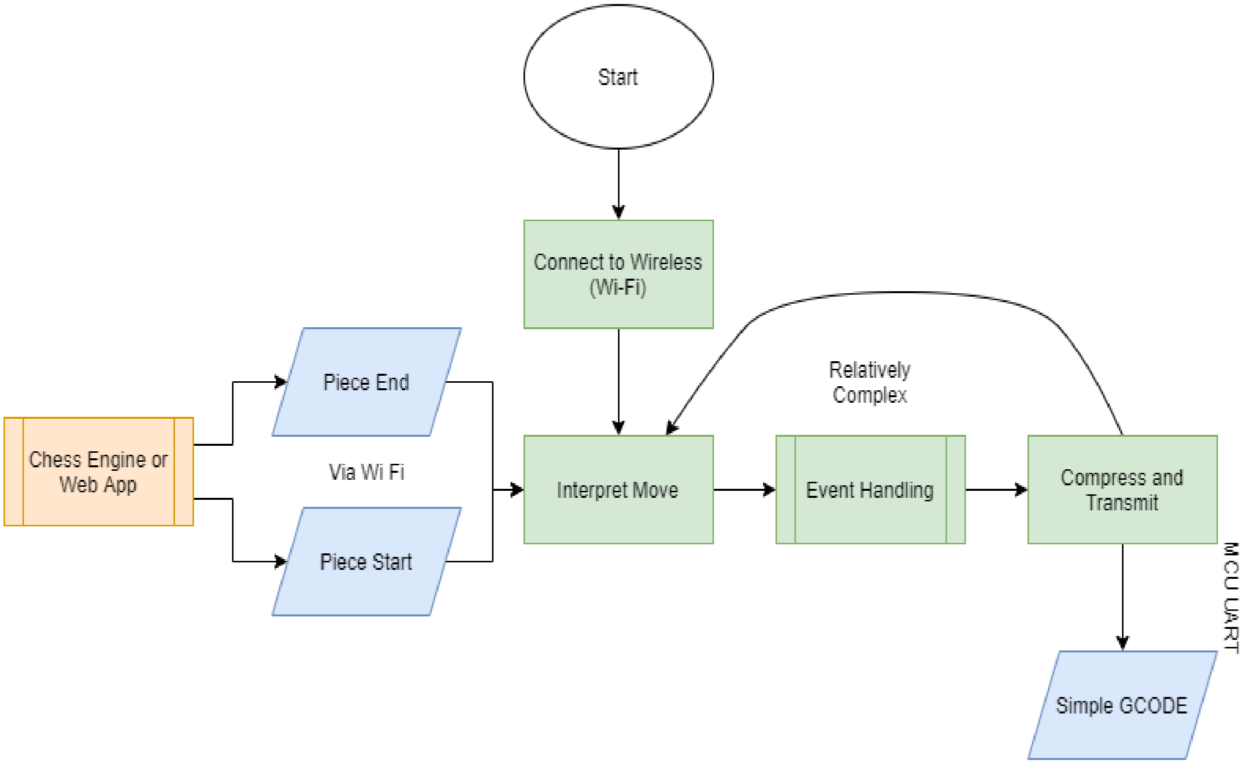


Figure 3-11: Flowchart Demonstrating How Klipper3d Would be Used in the Gameboard

The time-per-turn could also be reduced by utilizing the “Input Shaping” feature found within the firmware. What this does is take in calibration data measured by the user and modify the commands being sent to mitigate errors observed during calibration. At lower speeds this is irrelevant, but at higher speeds this can reduce positional errors during movement. It is possible that the magnetic force from the head of the robot on a piece being moved will be a limiting factor before this advanced code is necessary.

As Klipper is written in Python which then communicates with base firmware of a chained MCU, it would be easier to develop and test new features for the robot. So long as the slave device can interpret GCODE commands there would be no need to modify the software on the device itself – eliminating the need to flash new firmware with every update. A high-level programming language such as Python also allows for many different functions and applications to be pieced together – such as the web app, MCU firmware, chess engine, and any peripheral features which may be added.

While Klipper is easier to develop in once installed, it would take extensive modification to reach a point at which the development could occur. Since it requires a fully functional slave device capable of receiving GCODE commands we would need to either find or develop firmware capable of doing this. Any bugs which are not discovered in the slave device would also obfuscate any errors which arise from the master controller, potentially losing all of the efficiency gains which would have come from developing and testing from the laptop itself.

3.2.5.3 Visual Studio Code

Visual Studio Code is Microsoft's simple IDE intended to provide a flexible platform for communities to develop bespoke functionality for. Extensions – the method by which users can install these bespoke functions – are the primary appeal for our team with regards to this project. Linters, code libraries, and even embedded system firmware uploading tools have already been developed. Some members of the team have prior experience with them which makes onboarding more straightforward as we begin to move towards development.

Any software developed using this development environment is under very few restrictions by Microsoft in their use license as well [9]. Developers are free to develop and deploy their solutions without any involvement on Microsoft's part – something which is relatively standard for low-power IDEs such as this, but it is a consideration we must make when choosing software. Each extension is free to have its own terms of use though, meaning that just because Visual Studio Code may not have overly restrictive terms of use, we still need to verify individual extensions before developing with them.

3.2.5.4 PlatformIO

This is an existing extension for Microsoft Visual Studio Code which offers tools to support developing code for existing platforms. A .pio file is generated for each project, which then contains configuration settings where the developer can specify hardware names, memory parameters, and unit testing scripts. Once these have been prepared the code to be built and loaded is developed with analysis tools capable of detecting potential errors which would cause the build target to stall or behave unpredictably.

PlatformIO also supports remote development – meaning that wireless MCUs can be written by any member of the design team from their respective workplaces. Due to the constraints of current events, this would allow for greater collaboration and quicker turnaround times not dependent on any individuals availability. Unit testing can also be performed wirelessly, with diagnostics and debug data being transmitted back to the developer's terminal.

The main advantage of this tool is that it is able to target nearly MCU and deliver the development experience of Microsoft Visual Studio Code. The IDE has a wide variety of features such as in-depth linters, easy extension installation, and debug tools which some of us are already familiar with. Time which would otherwise be spent adapting to an entirely new IDE can now be spent solely on understanding the requirements of the device.

Regarding the legality of using this for our project, PlatformIO is available under the Apache License 2.0 [10]. This is an extremely permissive license which allows for the creation of derivative works with no monetary compensation or prior permission of the author (in this case the developers of PlatformIO.) We are required to submit the NOTICE file contained within PlatformIO if we release source code which requires PlatformIO specifically, but we are not currently pursuing any features which fulfill that criteria.

3.2.5.5 PCB Design Software

Since we needed to design and order a custom printed circuit board for our project, we must investigate which design software tool will be best to use given our project needs. Ideally, we wanted to ensure that the design software we use is reliable and simple to use such that we do not run into issues with our time constraints. Most importantly, we need to make sure the design software is capable of generating the necessary gerber file which is used when ordering the printed circuit board. This section will detail the advantages and disadvantages for different PCB design tools and guide our decision making for when we begin designing our printed circuit board.

3.2.5.5.1 EAGLE

EAGLE was developed in 1988 as a 16-bit PCB design application and initially consisted of only a layout editor and parts library and expanded to include a schematic editor in 1991. Over the years, this application has grown and become a very well-known PCB design software for professionals and hobbyists alike. EAGLE was then acquired by Autodesk in 2016 and changed to a subscription-only model in 2017 for the application [11].

EAGLE stores schematic files with .SCH extension and parts are defined in device libraries with the .LBR extension. You can store board files from the PCB layout editor, and it allows for back-annotation to the schematic which means making changes to the schematic will simultaneously update the PCB layout. Additionally, EAGLE contains an auto-routing feature to connect traces on the board based on the schematics defined connections. This application will also save the PCB layout as a gerber file which is the standard format accepted by PCB fabrication companies.

The benefits of using this software, first and foremost, is that it is already a well-established application that has been in use for decades. Because of this, it is fairly easy to find EAGLE footprints for many components. You can find both schematic files and part libraries that are already designed to work specifically for EAGLE. This is a huge benefit because it will save us significant time since we will not need to build everything from scratch.

Some disadvantages to consider however, is that the application’s user interface is a bit outdated and can be a little difficult to use without a considerable understanding of how the application functions. That being said, our team also has the most experience with EAGLE as opposed to other PCB design applications. Additionally, since we are students, we are able to obtain the Educational version of EAGLE which is free to use. This version does have some limitations including a limit for schematic sheets, number of layers and size. These limitations would not be an issue since our PCB does not need to be extremely complex in order to fully realize our overall design.

3.2.5.5.2 SolidWorks

SolidWorks is a software made by Dassault Systemes which allows people to do solid modeling via computer aided design, or CAD for short. This software was launched in November 1995 and has come a very long way since then. SolidWorks has been adopted by makers, universities, and industries all over the word. Despite its broad reach, it is only officially supported to run on the Windows operating system. Luckily, Windows has been heavily adopted and is one of the most common computer operating systems used on the world.

SolidWorks has become so popular within industry and universities that most manufacturers will often include SolidWorks part or assembly files in the product page on their website. If they do not offer the SolidWorks specific file, an STP or STL file are also usually present within a part’ supporting CAD documentation and can be opened and processed within SolidWorks. This software also allows for dimensioned drawing to be generated off a 3D rendered piece which can then be sent off to machinists for manufacturing.

SolidWorks is a parametric modeling software which means that the parts made with it are bound by host of separate parameters. These could be numeric parameters for lengths or diameters or relational parameters like tangents and midpoints. When just one of these parameters are set or changed, the entire 3D model will change to reflect this. This is just one of the many powerful tools SolidWorks utilizes to generate models.

Another very useful and power thing this software lets you do is add material parameters to a part. This comes in very handy when performing simulations and object studies within this environment. With these material parameters set, this software can find values such as the center of gravity of a component or even where the waterline on a buoyant object floating in the ocean will be. There is also a time-based action system which lets the user conduct motion studies. This can simulate how components will react to external forces like gravity or active motors.

3.2.5.5.3 Fusion360

EAGLE was initially released in 1988 by AutoDesk – but recently Autodesk has tooled their Fusion360 and Inventor applications to now include electronic schematic and printed circuit board design. Perhaps the largest advantage of Fusion360 when compared to Eagle is the modern UI, making the process of designing much more streamlined and conducive to rapid sketched. However, there are many stability issues which can arise after projects get more complex.

This project does not stretch the definition of complex, but it is a concern which that should be considered. Many of these issues are from over a year ago and were not run across when some of our members tested the program to look for issues. The largest issue was that the cloud-based storage is sometimes slow to respond. Syncing across the team will technically be automated, but on the individual level there can be long save times when attempting to shut down the software or during development.

Fusion360 does show great potential despite these issues. Since Autodesk has developed both Fusion and Eagle, all eagle libraries are compatible with Fusion360. This makes Fusion a happy medium between modern UI and the huge number of available parts online. The typical simulation and validation tools are available as well, with the validation tools being more robust in their error descriptions. The added benefit of the 3d models is ultimately a small one, but it would allow for our design to exist as an entirely digital model with accurate depictions of the PCB and cartesian robot.

Since we are students attending UCF, we can use this software under their educational license [12]. This prohibits us from commercializing the product and restrict our use of any parts designed in Fusion360 for educational purposes only. During senior design this is obviously be covered, but we are unable to commercialize the product without buying a full license from Autodesk before release. This license also applies to any other Autodesk products we may use – including EAGLE, Inventor, and AutoCAD.

3.2.5.5.4 EasyEDA

Another application we considered using is EasyEDA which is a web-based EDA tool that would allow us to design schematics and PCB layouts [13]. This application was created in 2010 with the goal of being easy to learn, easy to use, and most importantly, free. EasyEDA launched in 2014 and has developed rapidly since.

EasyEDA allows for the ability to import LTspice schematics which is useful for creating the PCB layout without having to draw everything. The application also accepts a large variety of file formats for importing netlists. There is a significant emphasis on this application being easy to use and learn and it is very intuitive to use. Since it is so easy to use, it ultimately will save us substantial time when designing our PCB.

However, there were some disadvantages to consider, the major one being it is a less established application meaning we may have run into troubles finding the appropriate footprints in comparison to EAGLE or Fusion360. Additionally, EasyEDA does not have nearly as many features as the other two application we discussed above. While this software is great for simpler designs, it is entirely possible we will run into issues down the road as we add more complexity to our PCB design. That being said, we do want to keep an open mind and fully explore this option since, at first glance, it seems it could very well do exactly what we need without any problems.

3.2.5.6 CURA Slicing Software

For any 3D printed tools we design, it was necessary to use a slicer. Slicers are software tools which allow for files to be imported, "sliced" into layers of machine instructions, and then exported to devices such as USB sticks or printers connected to the computer for the actual printing process. While the basic operation of the slicer is relatively straight forward – importing files is often as easy as drag-and-drop – tuning the settings which determine how the print is completed is vital to consistent manufacturing.

Temperature, retraction distance, and the various motion parameters of a printer are often unique to an individual machine. CURA allows for us to create individual profiles for the machines owned by various members of the design team and ensure that the prints come out as quickly and true to form as possible. In addition to printer and material profiles, the underlying algorithms used in CURA produce much more consistent and high-quality results than competitors such as Creality Slic3r. It is for this reason, as well as our own familiarity with CURA itself, that we choose to use CURA.

CURA operates under GNU Lesser Public License 3.0 [14]; meaning that anyone is free to use the software for commercial and educational use as well as modify and distribute any segment of the tool. This is obviously very permissive and would pose no issues for us as we will not even be working on CURA itself, only using it during our manufacturing phase.

3.2.5.7 Adobe Xd Prototyping tool

Adobe Xd is a User Experience design tool that is packaged with the Adobe suite. Using Adobe Xd, users can quickly design a functional prototype of what the application will look like, as well as how the application will flow. Adobe Xd enabled our group to have discussions about the design and dataflow of our application without having to write a single line of code. A feature that the tool offers is its compatibility with other common CSS libraries, such as Material-UI. This was incredibly useful as it not only provided our group with a wireframe but allowed us to visualize what our application would look like down to the details.

Adobe Xd also allowed us to make real-time changes during our group discussion of features that we would like to see implemented in the application. Adobe Xd also allows users to develop a workflow of the application, using its live prototyping features. This was crucial in identifying the number of pages that were needed to meet the full functionality of our core goals. Using Adobe

Xd to draft a prototype also allowed the developers for the web application to have a visual reference when it came time to develop the application. This helped tremendously in terms of dividing up tasks, and assisted in keeping our developers in sync, as they were both referencing the same prototype. Removing this ambiguity shaved down the development time of the front-end. This also facilitated the dividing and distribution of tasks when it came time to develop the front-end.

3.2.5.8 Yarn Package Manager

The entire set of both front-end and back-end software is heavily dependent on the use and proper installation of a series of open-source libraries. These libraries are often packaged and downloaded using a package manager. For this project, we utilized a package manager called Yarn. Yarn is an open-source package manager developed by Facebook, which has active community support and is regularly updated for safe package use. We chose Yarn over more popular option such as NPM because of its added level of security and more recent set of updates. We also selected yarn for this project as it doubles down as a project manager, which allows us to easy maintain control over which dependencies are involved in this project. Using Yarn, we were able to download and install all the other libraries and dependencies listed below in this project.

3.2.5.9 React Front-end

The Arcane game board will offer an external, off-board web application that will allow the player to directly interact with the gameboard without physically having to move any pieces. The web application will run locally on the players device, in this case, being a laptop. The web application will consist of a full stack application, comprised of a separate front-end and back-end. The reason we are separating the two is to boost performance while maintaining a good standard for User Experience (UX). Both the front and back end will utilize various technologies and open-source libraries.

The front-end will make use of various technologies, frameworks, and open-source libraries to show the user who is using the laptop all of the possible controls and interactions that the board will have to offer. React will be the primary tool to organize and show the different tools and features for the front end. We choose React because of its extensive community support, active updates, ability to create light-weight static files on time of build, and overall ease of use. We organized our front-end using React's component style filing system, where we render each individual tool (header bars, virtual chessboards, etc.) as their own components on a need by basis. The filing system organizes our code into separate "slots" or "components" which are then only rendered if an action requests them.

We also chose React for its "hooks" functionality, which allows us to utilize prebuilt functions with UseEffect and UseState being the main hooks of focus for our specific app. These hooks fit well with the design of the application as there will be a direct stream of data ported from the WebSocket. The tool of using states allows us to render individual components if the application detects a state update or if the value of a variable updates within the application. React does this all on its own, without requiring a conditional statement, which makes coding much easier, cleaner, and much more efficient. The UseEffect hook, only runs when a certain set of dependencies is triggered, is also extremely useful in our application the app is continually using and doing actions which updates the state of certain variables. An example of an action would be moving a chess piece, starting up the application, clicking a button to start the game, and many more. By listing the associated variables and functions as dependencies for these UseEffects, we can quickly trigger the reciprocating events that would be tied to the initial action. The figure below, *Figure 3-12*, depicts the lifecycle of React “hooks”.

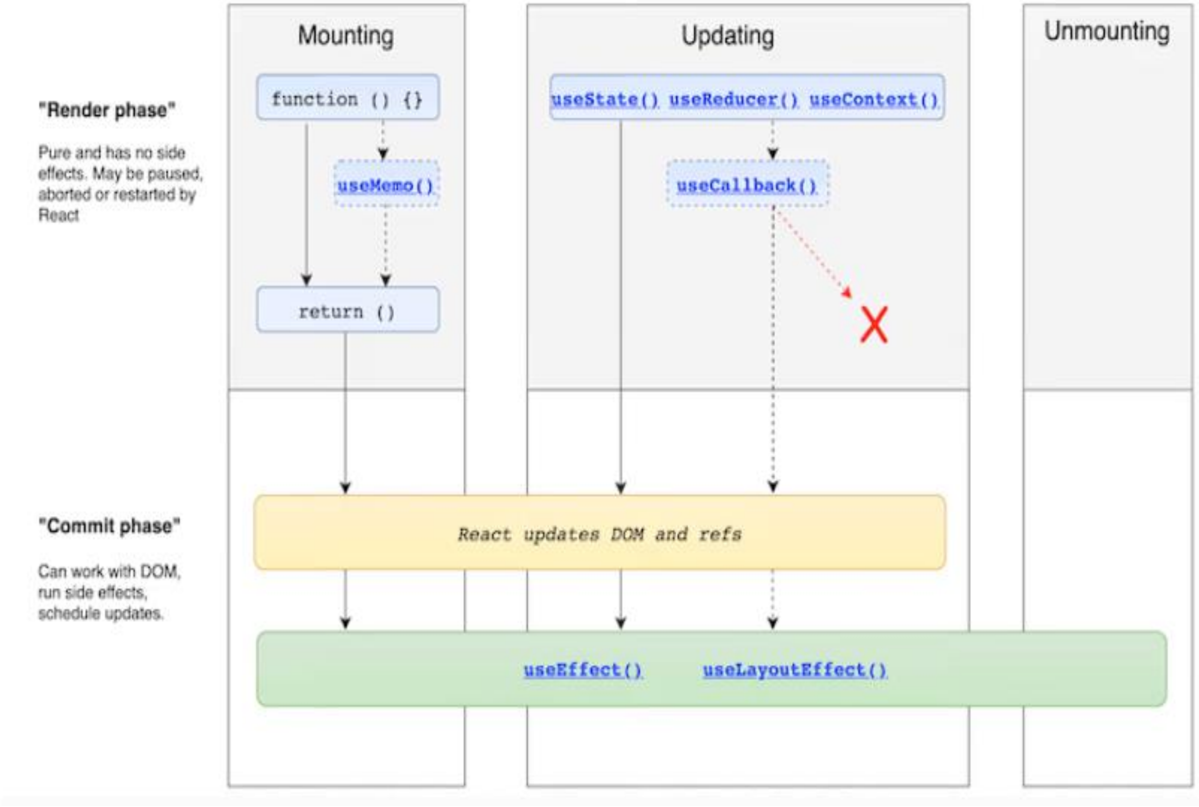

Mounting
Updating
Unmounting
"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React
function () {}
useMemo()
return ()
useState() useReducer() useContext()
useCallback()
X
"Commit phase"
Can work with DOM, run side effects, schedule updates.
React updates DOM and refs
useEffect()
useLayoutEffect()

Figure 3-12: Lifecycle of React Hooks

Our group will provide an entire, navigable interface using Material-UI, to allow quick navigation and game set up with east. We also intend on fully demonstrating a live-action version of the current set up on the physical board, portrayed on a virtual board. The intent behind this is to also allow the user to directly interact and move the pieces on the physical chess board from the web application. The chessboard software that we plan to use is ChessBoard.jsx as well as Chess.js. This tool will be coupled with the React front end, which will work on placing, centering, and rendering tool ChessBoard.jsx. This will also allow us to make use of React’s ability to export lightweight static files, which greatly boosts performance.

3.2.5.9.1 React-Router

React-Router is a Routing library that pairs seamlessly with our React framework. Like the other libraries we are using, React-Router is incredibly lightweight and efficient. React-Router is highly ranked in terms of routing libraries and is the top choice for our application. Our web application will consist of several pages, each offering different functionality to our users. To effectively navigate these pages, a routing library, such as React-Router will be used. React-Router utilizes the URL of the current page and renders all the necessary components associated with that page. This makes development very easy and quick, as all that we must do to change between pages, is alter the URL. React-Router then recognizes the new handles that have been added or removed from the URL and pulls up the related components. We can reference *Figure 3-13* to better understand how React-Router hides and shows application components based on the URL that is currently set.

Our web application will consist of a homepage, a page that is focused on gameplay with two players, and one that focuses on gameplay with a computer. The true value of React-Router comes from its speed. Because we are simply rendering components, we can do all our navigating from the client-side of things. This is much faster than having to ping the backend with new page information, every time that we want to change the page. React builds and ships static files, so all the components are already available the first time that user pulls up the web application, so switching between these components is incredibly quick once they have all been loaded the first time.

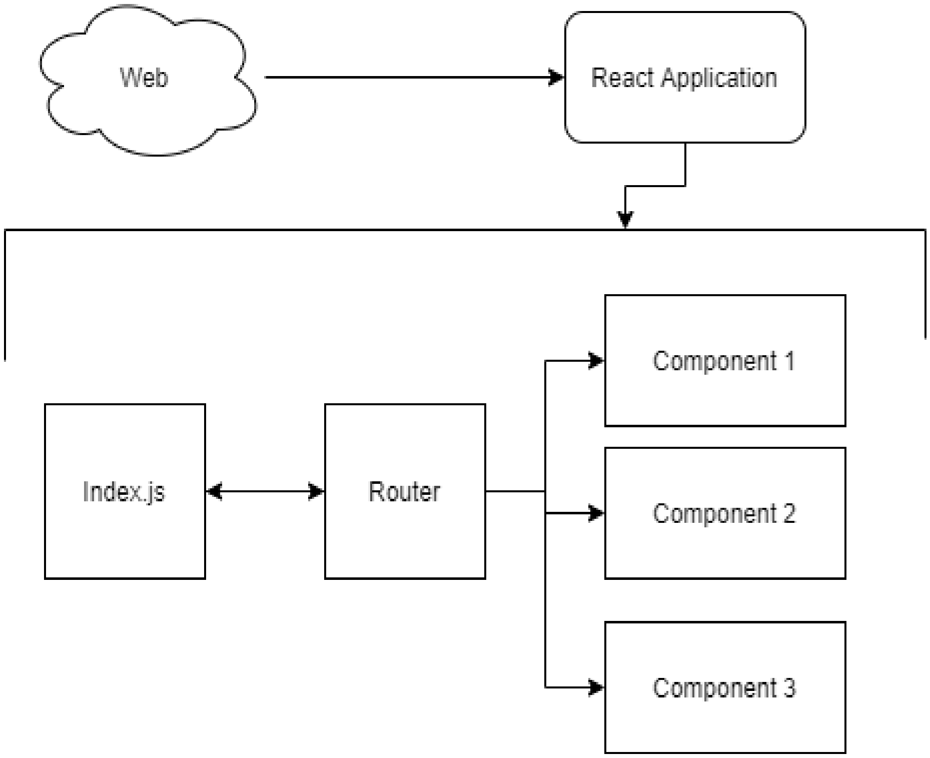


Figure 3-13: React Flowchart

3.2.5.9.2 Redux.js

Nowadays, there is an expectation that good software will run quickly, regardless of the device that is running on. In order to achieve this, we can utilize special tools that enable client-side caching, which allow us to store data on the user's device. For our web application, we will be using Redux.js to achieve this. Redux.js is a development tool that presents two key features. Client-side caching and global state access withing a React application. Below, *Figure 3-14*, depicts the dataflow of Redux.

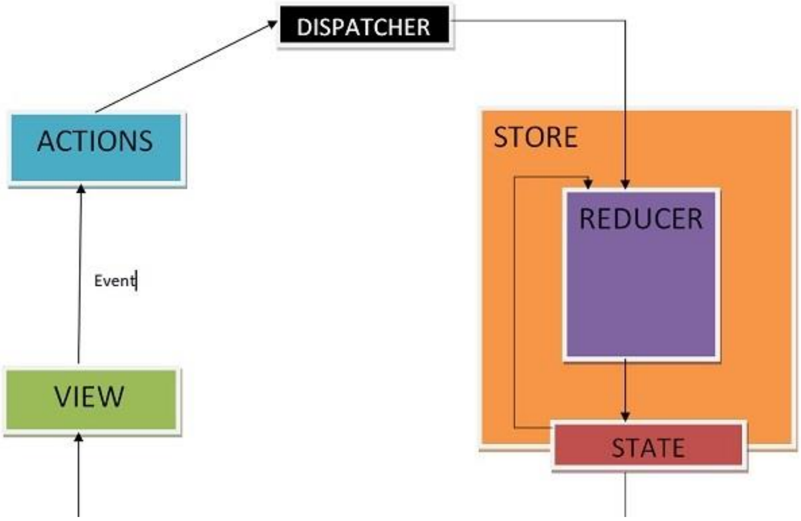


Figure 3-14: Dataflow of Redux

Normally, in base React, we often run into an issue when it comes time to pass functions or method down child components. This is often a necessary task as we have child components that will need to modify the state of parent components and vice versa. While this is not too complex in terms of one or two components, things tend to get quite dicey after trying to handle this down to 3, 4, or more components. Handling the parent state grows infinitely more complex the deeper into the component tree we go and can often cause a slow down in performance. Redux solves this issue by creating a global store, in which we can declare global variables, and set their states in local components. This enables the developer to reference these states from anywhere in the application. We do this using a combination of dispatchers, actions, and global states. Looking at figure XXX, we can see the sequence of the data flow for storing variables to redux. This is extremely beneficial because we no longer must be in a parent component to render that parent's child component. We can render components from anywhere in the application. This simplifies our code tremendously, as well as boosts the overall performance.

The other key feature that is offered by Redux.js is the ability to store cache locally to a user's device. This is crucial in creating a quick and snappy application that runs smoothly on any capable device. The use of a local cache reduces the overall number of API calls that need to be made. Instead of needing to make an API call anytime a desired set of data is needed, we can simply store that data locally in the Redux. We can then reference this data throughout the application using the local store as described above. The one downside of this is that it requires some more data on the user's device in order to run the application. This trade off is often worth the noticeable boost in performance, and reduced number of API calls.

3.2.5.9.3 Material-UI

To provide a navigable interface for the users, we will be utilizing Material-UI alongside React in order to make this possible. Material-UI is an open-source CSS Library that was developed by google to provide a modern, easy to use, and timeless design to web applications. Material-UI works well with React, as it focuses on using individual components which fall nicely into React's component style development. Material-UI also provides the ability to develop in responsive design, which would make it easier and quicker to develop multiple views (mobile, tablet, and desktop), should we decide to expand those features down the road. The performance of Material-UI is also tried and true, as it is light-weight compared to other CSS libraries. The UI will be designed with the desktop view in mind and will later be adapted to either tablet or mobile should there be that demand.

3.2.5.9.4 ChessBoard.jsx

To show the live action chessboard on our web application, we will utilize ChessBoard.jsx. This is an open-source, virtual chess game (*Figure 3-15*), that provides all the necessary graphics, sprites, colors, and assets necessary to set up a virtual chess game. ChessBoard.jsx fits well with the React component filing system and is the best package in terms of being able to integrate with React. ChessBoard.jsx also works very well with Chess.js, making it the prime choice for our application. ChessBoard.jsx also allows the potential to further customize gameplay by providing users the opportunity to use their own sprites. While this isn't listed at the moment as a core feature, it can serve as stretch goal for this project.

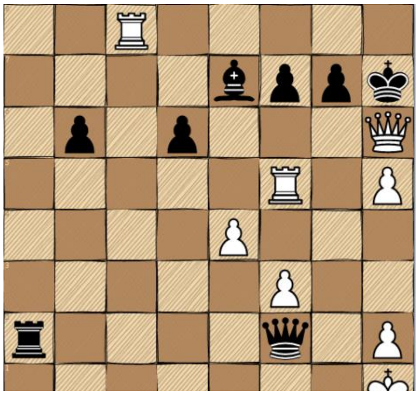


Figure 3-15: ChessBoard.jsx Default Chessboard Assets

3.2.5.9.5 Chess.js

Alongside the visual aspect of ChessBoard.jsx, we will utilize the move set library that is Chess.js. Chess.js is an open-source library that contains all of the legal moves found in a game of chess. It can be easily integrated with the visuals of ChessBoard.jsx, and is also commonly found with a React application structure. Chess.js provides the extensive list of moves in chess, while leaving the AI to the choice of the developer. Chess.js is also a great choice for this application as it breaks down the potential chess moves into an understandable logic that a developer can quickly interpret (2D Arrays). Its extensive API also allows for a lot of key features that come alongside developing a chess application, such as Game over, Reset, and other thing that you would expect from a video game. Another vital feature that is offered by Chess.js is the ability to recall the previous games history. This feature is key in terms of debugging our application and will also allow users to further study where they can have improved on their gameplay. All of these serve as the deciding points as to why we chose to use it in our application.

3.2.5.10 Back-end

Our web application will utilize several technologies in order to provide the brains behind the application. The Back-end's job in this application is to extend communication to the physical board, while also enabling some core features withing the application such as move history. The beauty of using a programming language such as JavaScript is that it allows developers to program both the front and back end, using the same language. We are utilizing JavaScript for the backend, as all of the tools listed have been ported to support JavaScript. In terms of directly communicating with our Microcontroller on the physical chessboard, the back-end will use WebSockets. In order to read and pass on these incoming messages to the front-end, we will use Node.js. Finally, in order to potentially meet a stretch goal, and provide some brains to our application with a playable AI that can compete with the user, we will utilize, and open-source package known as Stockfish. Using these three technologies, we are able to successfully bridge the communication gap between the web application and the physical chessboard, while enabling some desired features such as AI opponents.

3.2.5.10.1 WebSocket

Websockets are a tool that allows for bi-directional communication between web applications, microcontrollers, local programs, and much more. WebSockets is the tool that we chose for our datastream, to enable a direct line of communication between our microcontroller and our web application. The WebSocket technology work very well with both the microcontroller as well as the backend that we have set up for our web application. WebSockets also allows the developer to develop in JavaScript, as this is one of the many languages that it supports. WebSockets also pairs seamlessly with a Node.js API, which this web application will be utilizing. The tool is separated into two separate parts. We can utilize the client-side in order to communicate directly with our React front-end and make use of the server-side which will pair with our Node.js back end. The simplicity of the API makes this a prime choice for developers, as it enables the developer to learn only one skill set which is applicable across two sets of features. The key reason behind using WebSockets is its ability to allow for real time updates across our web application and our physical board. The major benefit in using WebSocket design is that it does not require any polling on either end. Once an update becomes available, and the microcontroller sends an update regarding some information (such as if a player decides to move a chess piece), then the web application will be immediately notified, without having to pole for that information. This in turn replaces an API with poling functionality and allows for real time data stream communication between the two entities. Refer to *Figure 3-16* for the dataflow state diagram.

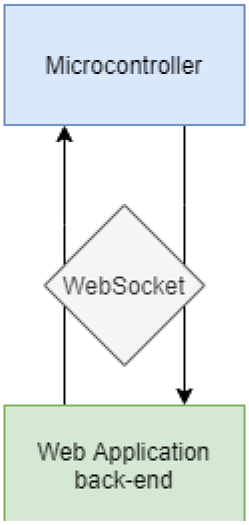


Figure 3-16: Dataflow State Diagram

3.2.5.10.2 Node.js

Node.js is an open-source, cross platform back-end environment that assists in running JavaScript code. Node.js is incredibly lightweight, and runs JavaScript code at class-leading, benchmark speed. In this application, we used Node.js to serve up our React application locally. The two (React and Node) go hand in hand. Since Node uses a lot of callback functions, and React works with Promises, the two work together cohesively. Node.js also make use of non-blocking I/O model, which is very efficient. The I/O in this is referring to input and output, which can mean anything from reading and writing files to HTTP requests from an API. This essentially allows processes to be called asynchronously. A good way to visualize this is by applying the concept of multithreading. With Node.js, multiple processes can be fired at once, and they will run independently of one another. This is how it achieves its class leading efficiency. Another core reason as to why we chose Node.js for our application, is because of the extensive number of libraries that have been developed for programmers to use. Refer to *Figure 3-17* below.

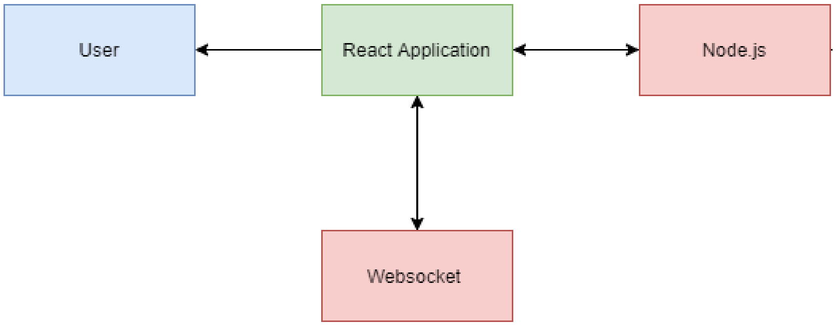


Figure 3-17: State diagram Node.js Role in Dataflow

Utilizing the package manager Yarn, we can import and install an extensive number of essential libraries and dependencies to our application, making them available for quick use. The seamless nature of installing these libraries allows us developers to speed up the development process and enables us to build projects of this caliber in a reasonable amount of time. All of these reasons, paired with the easy-to-use nature of Node.js, serve as key points as to why we chose to use this specific technology.

3.2.5.10.3 Stockfish 13

Stockfish 13 will serve as the brains behind our Arcane gameboard, in creating a ranking system for potential chess moves and enabling the use of an Artificial Intelligence (AI) to play against. Stockfish 13 is one of the highest ranked chess engines available on the internet today. Its speed and ability to quickly rank chess moves was class leading for many years. Its purpose in our application is to provide the intelligence for the AI, in order to allow users to play against a computer if there is no second player present. Stockfish 13 hosts a variety of features, with the AI being the main one of focus. One of our stretch goals is to enable AI opponents. Stockfish 13 in short, ranks potential chess moves based on the existing board layout. An example would be, for instance, given a specific board layout, Stockfish 13 can predict which chess piece holds the best set of moves for that specific play. If Stockfish 13 determines that moving the white bishop holds the most potential gain, it will then evaluate all of the potential moves that can be made from that piece's location on the board and provide a list of scores on which moves are most likely to succeed, vs those that can be more detrimental to the player of the white side.

This in turn, allows users to regulate difficulty. Stockfish 13 does this by allowing the selected difficulty to regulate the range of success in the choices that the AI makes. For instance, lets suppose that Stockfish provides a rating of 0-100 in terms of worst to best chess moves for that specific board layout. We can regulate the difficulty by limiting the ability of the AI to only choose moves that fall in the 40-60 range of ranking (a medium difficulty) or in the 20-40 range of ranking (an easy difficulty).

3.2.5.10.4 MySQL Workbench 8.0

MySQL Workbench 8.0 is an open-source, Database Management Software (DBMS) that can quickly set up database tables in order to store and use application data. MySQL Databases utilize SQL programming language, which can be used in a variety of ways in order to create, modify, manipulate, and delete data. Our application will require a login feature to get past the main homepage, and the user data must be stored in a MySQL Database. We chose MySQL DBMS for its simplicity and wide range of support from the community. We also chose MySQL DBMS for its regular updates and stellar reputation as a successful DBMS. In the end, we opted to keep the validation information stored locally on the users device using a Redux store. In the future, we can expand the app to include full users functionality, in which it would be best to have a database in parallel.

3.3 Components and Part Selection

Taking into consideration our design and financial constraints, we carefully evaluated potential components and parts to come to a final decision of what is used in the final design. We utilized the research completed in the previous section to guide our decision making and ensure we were fully considering our options. Additionally, we evaluated the advantages and disadvantages of each component in terms of our goals, objectives, and requirements. The following sections go into further detail regarding the process and overall reasoning behind our final component and part selections.

3.3.1 Linear Motion

When it comes to linear motion, there are 3 very common ways of achieving this. V-slotted wheels on V-slotted 80/20 aluminum extrusions, threaded carriages, and linear bearings on linear rails. Each have their pros and cons which will be discussed below.

V-slotted wheels on 80/20 have become quite popular in the 3D printing world. This is because 80/20 can double as structure as well as guides for linear movement. There are many masses-produced and metallic parts that are manufactured to interfaces with 80/20. When a company that produces 3D printers buys these products in bulk, they may actually be saving a bit of money by not having to produce custom hardware to interface with their own linear guide solution. They also save time by not having to design their own structural solution. However, the big downside to these wheels is they are very susceptible to uneven sliding as debris gunk up the bearings and the rollers wear out. V-slot wheel orientation can also make a difference in how fast the wheels wear out, Ideally, they would be set up such that the load on the rollers is not being applied axially. This causes increased friction to the sides of the wheels and thus they deteriorate faster and get uneven sliding.

The threaded carriage method involves having a long lead screw rotating within a threaded hole attached to the carriage you wish to move linearly. Motors tur the threaded rods and as the carriage is prevented from rotating, the carriage moves in the direction of the threaded rod axis. Preventing carriage rotation can be done by either having an additional linear rail or v-slot to grab onto, or having multiple threaded rods running through it. The multiple threaded rods method is what is commonly used on the Z axis in 3D printers and is why we made threaded carriages a separate consideration from the v-slot wheels and linear bearings on linear rail. There is a major point of failure when it comes to having multiple lead screws running through a single carriage. If the motors that drive the leas screws where to ever unsynchronize, the carriage would become skewed. This could have a major impact on a system that is relying on precise positioning of the carriage, like our project. If a motor starts to lag by a severe amount, there could be damage to the system.

Linear bearings riding on linear guide rails have become more popular on high end 3D printers. The bearings and guide rails may actually cost less in small batches than the v-slot wheels and v-slot 80/20 in small batches. This is one of the reasons we decided to use this method of linear translation for our electromagnet in our project. However, a slight drawback is that there are not many mass produced carriages that interface with these linear bearings, so it is up to companies to design their own. Our team needed to design our own carriage, but the time spent doing so should be minimal given our design experience with programs like SolidWorks. The cost of these carriages should be significantly less than a metal carriage designed for v-slot wheels running on V-slot 80/20 given that we plan to make them out of PLA plastic. Linear bearings on linear guide rails tend to run much smoother than v-slot wheels on v-slot 80/20. This is because there is significantly less friction on each moving component. We don’t have the friction between the sides of the v-slot wheels rubbing on the v-slot 80/20 and we also don’t have to worry about the friction in the v-slot wheel bearings. We also remove the need to tighten eccentric nuts to get the proper hold on the 80/20 which would have been required for the v-slot wheel linear motion method. Without having a restriction on how load can be applied on the linear bearings, unlike when using v-slot wheels, we gain more freedom in our carriage design. This could allow us to create a carriage that is more space efficient than a carriage that was deigned to interface with V-slot wheels running on v-slot 80/20.

3.3.2 Motors

Motors come in a few different varieties and it is important to consider the pros and the cons of each when it comes time to integrate them in a project. The most common types of motors are brushed DC motors, brushless DC motors, stepper motors, and servo motors. Each of these motors are fundamentally different in how they are controlled and how they operate.

Of the motors listed, the servo motor is the simplest to control. This is because servo motors typically have an embedded controller to control their position, if they are a non-continuous, or their speed, if they are continuous. This embedded controller usually interoperates the frequency of a PWM signal and matches that frequency to a radial position or a clockwise/counterclockwise speed. Most microcontrollers have a built in PWM signal generator so there is usually no additional hardware required to control these motors after the microcontroller. The power supplied to the motor comes via dedicated motor voltage and ground, so the logic and power signals are independent from each other. Given that we need to be able to know the position of our motors to position our electromagnet, we would need some form of position feedback. Continuous rotation servos don’t provide this, but non continuous rotation servos can be assumed to be at the position corelated to their sent PWM frequency, unless their load is greater than what they are geared for. We would need a motor that can complete a rotation several times, has positional feedback, and for a low cost. For these reasons the servo motor was eliminated from consideration.

The brushed motor is the next easiest motor to control. This is done by sending a PWM signal to the motor, but unlike the servo motor, the signal that is modulated is not independent positional logic, but the main motor voltage itself. Large motor and motors under load can easily surpass the peak rated current of microcontrollers, so a brushed motor electronic speed controller (ESC) is used to interoperate commands from a microcontroller and modulate motor voltage. The same ESC is also responsible for switching the polarity of the voltage supplied to the motor which makes the motor spin in the opposite direction. Brushed motors have the shortest lifespan of the motors being considered. This is because they use brushes that make physical contact with commutators on the rotor of the motor to handle the timing of the electromagnets that cause the motor’s rotation. The brushes wear out over time and eventually stop making contact with the commutator, which makes the motor stop spinning. Brushed motors do not have any native features that give position control, so an encoder is required to get position feedback from the motor. A microcontroller can interoperate the signal from the encoder to have an idea of where the brushed motor is in its rotation.

A brushless motor is very similar to a brushed motor with its key difference being, as the name would imply, a lack of brushes. This means that the motor does not have an internal commutator either. Electromagnet timing is instead handled by an external electronic speed controller for brushless motors (BLESC). By removing the brushes and commutator, the motor becomes much simpler internally and has a much longer lifespan than a brushed motor. This is because the main point of failure is the bearings that the shaft spins about. Again similar to the brushed motor, the brushless motor has no native features to provide position feedback or control. The use of an encoder is one possible way to get this feedback and control the motor position.

The stepper motor is roughly as internally complex as a brushed motor, but also has the no contact approach of a brushless motor. The rotor is generally divided into a north and south rotor cap and generally has 50 teeth per cap (for a 1.8 degree per step motor) which are out of phase with each other. The stator in this motor usually has less teeth than the rotor and it is this difference which helps control the position of the rotor. At any given time (using a full step commutation) the south cap teeth of the rotor are perfectly aligned with the north end teeth of the stator and the north cap teeth of the rotor are perfectly aligned with the south end teeth of the stator. This gives the motor a high holding torque when not traveling. By changing which electromagnets are turned on at a time, the motor can step through its rotor teeth and thus change its position by 1.8 degrees in the clockwise or counterclockwise direction. This gives the stepper motor reliable open loop position control so long as the load torque does not exceed the motor's rated torque. Since we need accurate positioning for our project and we would like to minimize costs, we chose to use stepper motors to avoid buying additional sensors to monitor position.

3.3.3 Motor Drivers

As mentioned earlier, controlling the direction and speed of a brushed motor comes down to voltage seen on the brushes of the motor. The main job of the brushed motor's ESC is to take in some sort of signal representing speed and direction of the target motor and translate that to the appropriate voltage on the brushes of the motor. Speed is often controlled by taking the source of the motor voltage (like say a 12volt battery) and pulsing that voltage at a precise width and frequency. This pulse width modulation, or PWM, is what makes the motor feel like it is receiving some fraction of the source voltage. So, if we wanted to run the motor at 50% top speed, the ESC would set a motor voltage at a PWM signal of 50% duty cycle. If this motor voltage was analyzed on an oscilloscope, the peek voltage would be present around 50% of the time on any given time window. This would make the brushes effectively see 50% of the peak battery voltage and the motor would run at half the speed it was rated for our battery's peak voltage.

To make the motor change direction, the internal circuity would be in charge of changing the vase of the voltage seen on the brushes. Essentially the ESC would need to change which brush is the battery ground and which brush is some positive PWM voltage. A very popular circuit that allows this phase switch without making any physical changes to the connections in the circuit is the H-Bridge. The key to this circuit is 4 transistors which are arranged in such a way that 2 transistors share a connection to the battery positive terminal and then connect to a single brush. The next two transistors again share a motor brush and also connect again at battery ground. The ESC sends precise signals to the proper transistors to ensure that at any given moment, only one transistor ties the battery's positive terminal and a brush and only one transistor ties a brush to motor ground. By alternating which two transistors are active, the flow of current through the motor is changed. This in turn changes the polarity of the electromagnets within the motor and causes the motor to spin either clockwise or counterclockwise. The timing of when the electromagnets within the motor turn on are handled by the internal commutator making contact with the brushes. Below is an image, shown in *Figure 3-18*, of the H-Bridge circuit mentioned earlier to aid in your understanding of how one would function.

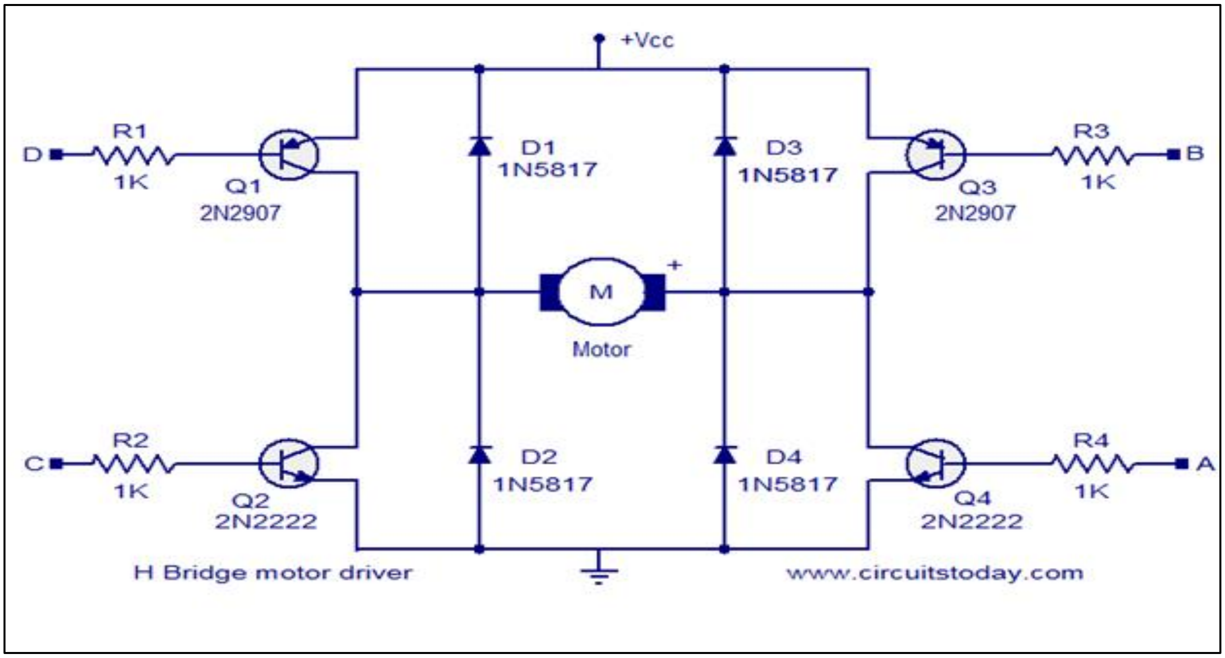


Figure 3-18: H-Bridge Circuit Schematic

Brushless ESCs also have to regulate how much voltage the motor sees, but these speed controllers regulate voltage on the individual phases of the motor rather than on brushes. Brushless motor has no commutator, so the task of timing when each electromagnet fires is up to the ESC. Depending on the motor, there are a few different ways that the ESC can determine when to switch a phase on and off. Some motors have built in hall effect sensors which can register when a magnetic field passes by. These Hall effect sensors will sense when the internal magnets on the stator or rotor pass by and this lets the ESC know when to start firing the next phase of the motor to continue rotor rotation. If hall effect sensors aren't present, the ESC can pick up on the back EMF generated on a dormant coil ass a magnet crosses in front of it. This also informs the ESC to enable and disable the proper phases to continue rotation or slow the motor down depending on the desired behavior. The back EMF and Hall effect sensors provide the esc with the rotor's relative position, but not absolute. This relative position is also very course as the degree between phases is typically quite large in these kinds of motors. Since the brushless ESC has the handle all the functions of the brushed motor ESC as well as handle phase commutation, its circuitry is more complex. This leads to a general higher cost for brushless ESCs over brushes ESCs.

The stepper motor controller functions a lot like a brushless ESC in that it is also in charge of commutating the phases in the motor. However, because of the internal structure of stepper motors, the controller has to be a bit more careful with how much power it applies to each phase in order to avoid resonance. Each step in a standard stepper motor only rotates about 1.8 degrees, so there are over 50 phase switches that need to happen to get a single rotation out of this type of motor. Every time a phase kicks on, a slight vibration in the system occurs, so a stepper motor can get very noisy given how many times a phase needs to turn on and off for a single rotation. Precise control over how the current in each phase is applied can mitigate this. If the current in each phase is applied in a sinusoidal function rather than a typical step function, there is significantly less vibration, resonance, and noise in the motor. High end stepper motor controllers offer a high-resolution sinusoidal signal for each of the stepper motor phases to keep motor quiet and efficient. High end stepper motor controllers can offer a host of other features such as adjusting phase current on the fly as the load on the motor changes, further increasing efficiency. They can monitor for motor stalls and alert the user when one occurs. They can even connect to encoders for a full closed loop feedback system. When selecting our controller, we decided closed loop feedback was not required so we looked for a controller that would run the motor efficiently and quietly.

3.3.3.1 Stepper Motor Driver TMC2209

Trinamic's TMC2209 stepper drivers come with many features useful to this design. Sensorless homing, StealthChop mode, and a large voltage range in particular make it a good candidate. These features make the overall design less complex by reducing the number of sensors and providing a wide range for power supply design. The user experience is also enhanced by the StealthChop mode which negates almost all of the operating noise of the robot.

When using the TMC22090 drivers there are 2 operation modes available. The mode we were going to use utilizes UART to be able to pass information back to the MCU. This is essential to the sensorless homing feature – otherwise we cannot make use of it. As the ESP32 – detailed later – only supports 2 UART connections this means that the 2 motors of a CoreXY cartesian robot are going to eliminate any UART connection to the MCU.

3.3.4 Electromagnet

The electromagnet is a spool of enameled wire wrapped around a ferrous metal core. As a current is passed through the wire, a magnetic field is produced axially about the spool which is focused by the ferrous core. For our project, the electromagnet is positioned underneath the playing surface. We needed a force of attraction that is great enough to overcome the static friction coefficient of the game piece and the playing surface and a magnetic field that is large enough to pass through the playing surface. We also need to make sure that the magnetic field is not so strong that it attracts multiple pieces at a time as it travels under the plating surface.

Regarding the firmware requirements of the electromagnet, the Marlin-based code treats it as a toggle. It is possible to instead use PWM to vary the strength using GCODE during runtime, but this is a feature that the initial implementation likely won't need. Instead, we hard-coded the strength we desire into the EEPROM of the ESP32 and adhere to the toggle method. These commands (M255 and M256 for On/Off respectively) are already coded into the Marlin command library. Typically, this is used for part cooling fans on 3d printers, but the method of control is similar enough to not require modification.

3.3.5 Power Supply

From our research, we decided the best way to supply power to our game board would be through connecting directly to main via a wall plug. Wall outlets in the US supply approximately 120VAC at 60Hz however our design necessitates we convert this AC voltage to DC voltage. Additionally, after converting this voltage to DC we also want to step-down the voltage significantly because our design is not able to handle such a large voltage supply. We have two options for converting and stepping down the voltage.

The first option we have was to include a transformer, AC to DC rectifier, and 12V voltage regulator in our PCB design. The transformer would step-down the 120VAC to a smaller value that is actually usable. The AC to DC rectifier would then convert the AC voltage to DC voltage and then the 12V voltage regulator would step down the DC voltage to 12VDC. From there we would use the 12VDC voltage to supply our components and voltage regulators. While this option isn't too difficult, it would have added significantly more complexity to our PCB design as well as increase our production costs.

The second option is much simpler and cheaper overall. This option is to use a 12V power adapter wall plug. This eliminates any need to design a transformer or AC to DC converter because the power adapter takes care of the conversion for us. We would then need to include a power connector jack on our PCB design allowing us to easily supply 12VDC input voltage. One major consideration we must make with this option is choosing a power adapter that meets our current requirements. You can buy 12V power adapters with varying amounts of output currents therefore we must look at each of our components and determine the necessary current needed to operate. It was estimated that our design shouldn't use more than 2A of current, therefore we chose an adapter that outputs at least 2A to give ourselves some wiggle room just in case.

Since we moved forward with the second option of a 12V power adapter wall plug, we then evaluated possible voltage regulators based off of a 12VDC input voltage. It was absolutely necessary that we know exactly what our input voltage and current is to the system in order to accurately design the entirety of our power system. The next section, section 3.3.6, will go further into depth about the part selection for the voltage regulators given an input voltage of 12V and at least 2A.

3.3.6 Voltage Regulators

As discussed in section 3.2.3 Power Technologies, we addressed the need for voltage regulators in our PCB design. Additionally, we discussed using switching regulators, as opposed to linear regulators, due to their efficiency and low power dissipation. The next step in our design process was to determine how many voltage regulators we needed and the specifications necessary to realize our design. To determine exactly what specific voltage regulators we needed for our PCB design, we first evaluated the current and voltage requirements for each component. From this, we could determine how many voltage regulators are needed to supply sufficient power to each component without exceeding the components current and voltage ratings. The table below, *Table 3-1*, contains the voltage and current demands for each component based on the datasheet.

From this table we can see that there are three different voltages necessary in our design; 12V, 3.3V, and 5V. We are using a 12V AC/DC adapter to power our device; therefore, we do not need any voltage regulator for our motor drivers since this value is already our input. We do, however, need two different voltage regulators for the ESP32 and electromagnet. The ESP32 requires a voltage step-down to 3.3V and the electromagnet requires a voltage step-down to 5.0V.

Voltage and Current Demands						
#	*Component*	*Voltage Range (V)*	*Operating Voltage (V)*	*Max Current Rating (A)*	*Estimated Operating Current (A)*	*Estimated Total Current (A)*
2	TMC2209 Motor Drivers	4.75 – 29	12	2.8	0.85	1.7
1	ESP32 Microcontroller	3.0 – 3.6	3.3	-	0.5	0.5
1	P25/20 Electromagnet	-	5	-	0.3	0.67

Table 3-1: Component Voltage and Current Demands

In addition to considering the voltage, we must also pay careful attention to the current necessary to operate each device. The above table notes the max current ratings, if applicable, and the estimated operating current we designed for. We wanted to ensure that we were generous with our estimates so we can avoid any problems with not meeting the necessary design parameters.

Voltage Regulator IC Comparisons			
IC	*Voltage (V)*	*Efficiency*	*Cost*
LM2596	3.3	73%	$1.774
	5.0	80%	$1.591
LM2675	3.3	86%	$1.906
	5.0	90%	$1.307
TPS6213X	3.3	91%	$0.957
	5.0	94%	$0.742
LM2595T	3.3	78%	$0.559
	5.0	82%	$0.531
XH2596	3.3	73%	$0.397
	5.0	80%	$0.447

Table 3-2: Voltage Regulator IC Efficiency and Cost Comparison

Using this information, we could then begin searching for specific parts that meet our voltage and current requirements. We used *JLCPCB* to order our PCB and in order to simplify the process we were able to search their parts library to find the appropriate voltage regulators that are in stock on their website [15]. If we choose a part that is in stock, we could then request that the component be soldered on when we order it which saved us substantial time at the cost of a smaller selection of parts. Ideally, we wanted to find the most efficient IC for the least cost, particularly since our finances are a considerable limitation for our design. Additionally, to simplify the process we decided to choose voltage regulators that use the same IC for both the 5.0V and 3.3V. After sorting through JLCPCB's parts library, we were able to narrow our selection down to five different models. These five are listed above in *Table 3-2*, which compares efficiency and cost for both the 3.3V output voltage and the 5.0V output voltage.

From this table we can see clearly that there are two ICs which stand out in terms of efficiency – the LM2675 and the TPS6213X. However, the LM2675 is on the more expensive side in comparison to the rest of the ICs. While we are able to get a general overview of the direction we may want to go in with our parts selection, there are still other things that we'd like to consider, especially given how similar each of these ICs are. In order to get a clear picture of the advantages and disadvantages for each IC, we want to compare the datasheets further in depth.

We compiled *Table 3-3*, below, to better compare each IC based on the specifications given in the datasheet. While the differences are relatively minor, we must take all of them into account so we may make the best decision for our overall design requirements. It should be noted that the operating temperature range for all the ICs is between -40°C and 125°C, therefore operating temperature range is not be determining factor.

Voltage Regulator IC Datasheet Comparison						
IC	*Manufacturer*	*V_{in}* *(min-max)*	*I_{out}* *(max)*	*Pins*	*Size* *(mm)*	*External Parts*
LM2596	Texas Instruments	4.5V – 40V	3.0A	5	14.85x10.75x5	4
LM2675	Texas Instruments	6.5V – 40V	1.0A	8	5x5.8x1.75	5
TPS6213	Texas Instruments	3.0V – 17V	3.0A	16	3.3x3.3x1.1	6
LM2595T	HGSEMi	7V – 40V	1.0A	5	14.85x10.75x5	4
XH2596	XINLUDA	7V – 40V	3.0A	5	14.35x10.16x4.57	4

Table 3-3: Voltage Regulator IC Datasheet Information

From the first table, *Table 3-1*, we determined that for our 5.0V and 3.3V voltage regulator we needed a current output of at least 0.67A and 0.5A, respectively. With this in mind, the voltage regulator ICs that have a max current output of 1.0A is cutting it very close to our requirements; to be on the safe side, we wanted to choose an IC that has a higher max output current. This narrows our choices down to the LM2596, TPS6213 and the XH2596 which all have a max output currents of 3.0A. For all of these, our input voltage falls comfortably within the range specified on the datasheet and because of this input voltage was not a deciding factor. If we refer to the previous table, we recall that the TPS6213 has higher efficiency overall and lower cost than the LM2596 and XH2596. However, we also needed to consider that this particular IC also has a greater number of pins and requires more external components than the other two.

Initially, due to the importance of the voltage regulators being highly efficient and well within our design requirements we decided on the TPS6213 regulator family. Despite the simplicity of the LM2596 and XH2596, we decided that the TPS6213 would better meet our standards from both a performance and cost standpoint. Another advantage of the TPS6213 was that it was also the smallest IC from our list meaning the additional components needed is offset by the relatively small size.

The TPS6213X family included both adjustable and fixed output voltage versions. For our purposes we were going to use the fixed output voltage versions. For the 3.3V we would have been using the TPS62132RGTR and for the 5.0V we would have been using the TPS62133RGTR. These ICS are considered to be in ACTIVE status, meaning Texas Instruments does recommend the device for new designs which is ideal for our project. We projected that the efficiency of this part will be around 93%.

Unfortunately, due to supply chain deficits as a result of the global pandemic, the TPS6213 went out of stock before we could order our PCB. Therefore, we had to choose a new voltage regulator for both the 3.3V and 5.0V. Using the tables above, we decided to move forward with the LM2596 switching step-down regulator manufactured by Texas Instruments. Given our requirements, this component was our next best choice and was comfortably in stock on JLCPCB’s website.

3.3.7 Microcontroller ESP32

The ESP32 we decided on using supports 2 different UART interfaces. These 2 UART interfaces play a key role in allowing us to get the most out of our TMC2209 stepper motor drivers. These stepper motor drivers allow for diagnostic information to be sent over UART and one of these

The ESP32 comes equipped with communication antennae which is vital during the operation of the device. There are several key factors which must be considered when developing firmware which uses the Blu-Fi module (the manufacturer’s term for a hybrid radio device) for communication. Notably, the radio cannot operate as both a Wi-Fi and Bluetooth module simultaneously. This means that the modes must be toggled either during runtime or between power cycles.

Class 1, Class 2, and Class 3 Bluetooth transmitters are available. What this effectively means is that with little-to-no effort we can select the power consumption, range, and data transfer rate of the device during design. Bluetooth 4.2 is the overall standard for the Bluetooth mode operation with the additional specifications required to be considered Smart Bluetooth. Allowing for the use of Bluetooth Low Energy mode. More details on the operation of the Bluetooth module can be found in the following section.

3.3.8 Communication

We have decided to pursue wireless communication in order to enhance the user experience. Modern entertainment devices – such as video game consoles, headphones, and streaming services – all come with wireless capabilities which users have come to expect. A wired connection would be simpler to develop – especially with the limitations of the Blu-Fi module on the ESP32, but it would simply be too cumbersome for our purposes.

The available open-source code available from the ESP32 chip serves as a good enough starting point that the added complexity was somewhat negated by the education resources online. Building the board in such a way that it pulls from and posts to a central database also allows for future iterations to potentially negate the need for an offboard app. The primary blocker to this is the need for a system of tracking pieces that is also cost effective.

The ESP32 MCU comes with an integrated Blu-Fi chip – capable of transmitting on both Bluetooth and Wi-Fi channels. It is important to note that the chip cannot transmit on both simultaneously and requires credentials to access a Wi-Fi network directly. We decided that it would be a better user experience to use a Bluetooth device to transmit the SSID and password to the board and then switch to Wi-Fi rather than attempt to integrate a keyboard.

The Bluetooth standard supported is Bluetooth 4.2 – meaning that there are 2 primary modes of operation available. The standard – also known as classic – mode allows for the maximum range and data bandwidth possible. Using the integrated Class 1 transmitter this results in roughly 100m of unobstructed range with 3MB/s. There are also Class 2 and Class 3 transmitters, with 10m and 1m of range respectively.

If the design were to be run off of battery rather than main, we could enable Bluetooth Low Energy (BLE) mode. This comes with significant drawbacks the data transfer rate, particularly when the devices are >1m apart, and available channels in a localized area. In return the power draw is lowered during times of inactivity. Since the Bluetooth operation is only meant to provide credentials for Wi-Fi operation, this could be an acceptable tradeoff. *Table 3-4* shows a table of comparison.

Criteria	*Bluetooth Classic*	*Bluetooth Low Energy*
Channels	79	40 (3 Advertising, 37 Data)
Modulation	GFSK, $\frac{\pi}{4}$, DQPSK, 8DSPK	GFSK
Power Consumption	1 (reference)	~0.01x to 0.5x Reference
Data Rate	1Mb/s-3Mb/s	125Kb/s-2Mb/s
Max Tx Power	Equivalent	Equivalent
Network Topologies	P2P	P2P, Broadcast, Mesh

Table 3-4: Bluetooth Comparison

There are considerably less variables to consider for the Wi-Fi operation. The ESP32 supports Wi-Fi 5 protocols, export over SPI or UART, and support for data rates as high as 54Mbps. The module does support a low-power mode which can be used to minimize the power draw of the antennae. Similar to the Bluetooth module, this is primarily achieved through an inert mode during periods of low activity. During typical operation, the current draw is rated for 190mA during transmission and 100mA for receiving.

3.3.9 Firmware Libraries

Thanks to the popularity of the ESP32, we had many options available for firmware libraries when assessing what functions we were unwilling to implement ourselves. There were 3 areas which we decided to use open-source libraries, rather than implement our own solutions.

3.3.9.1 ArduinoWebSockets

Low-level web code was not a strength of any of the team members. To this end, we wanted a Websocket client library which we could draw upon for the wireless functionality. ArduinoWebsockets provided a solid foundation, with documentation that clearly outlined the methods we would be using. The only major hurdle was integrating it into a class structure, rather than using it on its own in a main loop.

3.3.9.2 ArduinoJson

Another library which greatly simplified our web functionality was ArduinoJson. Since the server delivers JSON files, we anticipated that this library would ensure that the data was parsed correctly every time. This is the most well-documented library of the 3 used by far and has very good computational efficiency.

3.3.9.3 TMCStepper

Rather than attempt to write the UART protocol for the TMC stepper drivers from scratch, we opted to use a library which allows for a wide variety of TMC's products to be controlled by embedded projects built in the Arduino Framework.

3.3.10 Development Environment – PlatformIO

While the Arduino IDE has been used in many projects in the maker community, it is simply not designed for projects with a large scope. This is seen in the lack of directory view, limited hardware support outside of the Arduino – and clone – ecosystem, and lack of support for languages outside of C language derivatives. PlatformIO offers the advantages of Microsoft Visual Studio Code by existing as a plugin. On top of this, it has many features of its own which was very useful in the development process.

The largest advantage of PlatformIO is the generality of its use. It supports most popular processors and allows the programmer to specify variables for custom hardware if needed. In projects with defined parameters for the target system it provides linting for the build process as well. If the program has any clear reason why it cannot run on the target MCU PlatformIO will also fail the build and notify the user – ensuring that errors cannot come from user miscalculation.

The build/flash cycle can also be greatly reduced as we are using a controller with built-in Wi-Fi capabilities. PlatformIO can flash firmware wirelessly to a defined target – allowing for the entire team to test builds on the final product regardless of its physical location. Due to the restrictions of gatherings at the time of writing this could prove an invaluable tool in the final phases of the implementation process.

3.4 POSSIBLE ARCHITECTURES AND RELATED DIAGRAMS

Figures 3-19 and *3-20* show an early concept design for the game board. After seeing the square off and square on auto chess board projects, shown in sections 3.1.1 and 3.1.2, we started designing our won linear motion concept. We were most familiar with v-slotted wheels and v-slotted 80/20 at the time, so we designed around those components. Without having done research on other linear motion setups we made a design around the cartesian linear stage method of translation. This design has a single stepper solely responsible for all motion in the X direction and an additional motor solely responsible for all motion in the Y direction. In this particular configuration, the motor that was in charge of moving the electromagnet carriage in the Y direction, would ride along the X axis as the carriage changed its position along X.

This design had many issues. One was the overall bulk of this system. 80/20 is not a light structural material and having an entire bar with a motor attached to it moving along our X axis would place a lot of unnecessary strain on our X axis motor. The bulk of the 80/20 and the carries designed to interface with the v-slot wheels was also extremely space inefficient. We would not be able to position the electromagnet over a significant chunk of the project board as those areas would be occupied by structural 80/20 or the carriage and v-slotted rollers themselves. Finally, this implementation was very expensive relative to other linear translational methods. 80/20 does not come cheap for small orders and there were many parts that needed to be purchased. This includes a minimum of 3 v-slot wheels per carriage, an eccentric nut per carriage, a bolt per wheel, a nut per bolt, and at least one washer per bolt/nut combo. This led us to do more research on other linear translational methods and go back to the CAD drawing board to come up with a more suitable design.

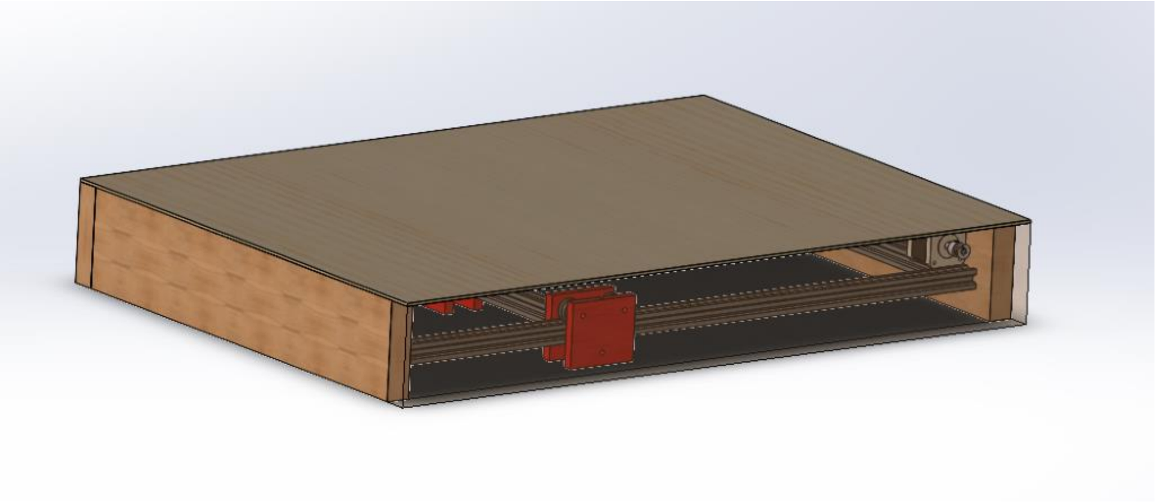


Figure 3-19: CAD Concept Drawing Overview

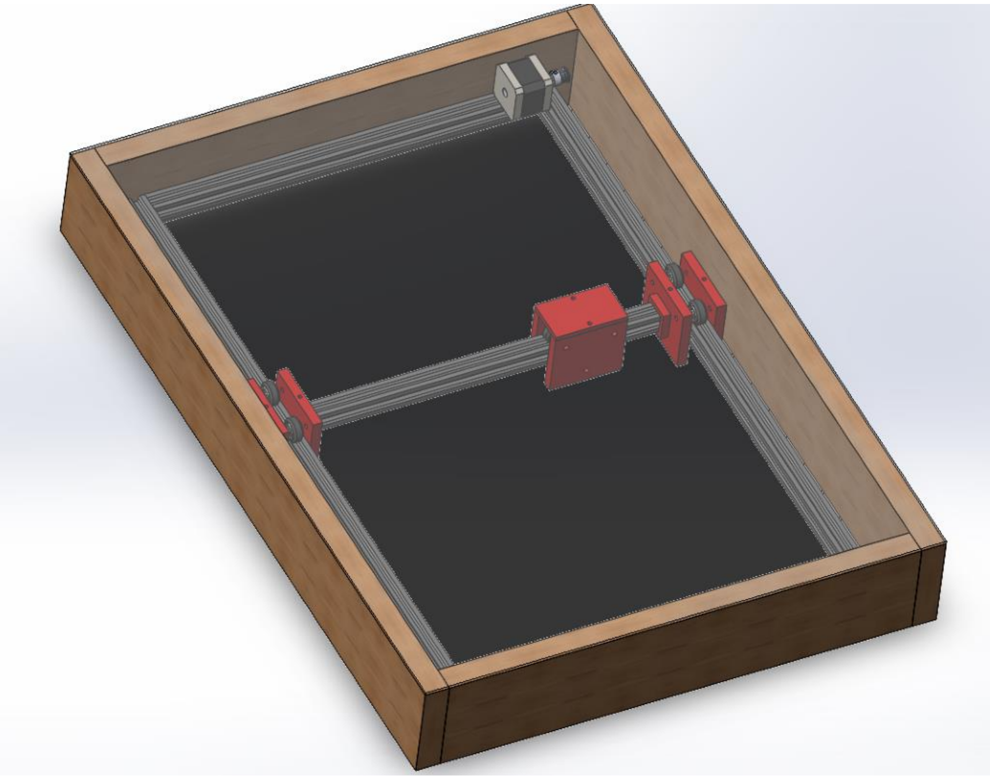


Figure 3-20: CAD Concept Drawing Internal

3.5 Parts Selection Summary

Our team has been very excited to design and construct a working prototype for this project. It was important for us to thoroughly research each aspect of our idea so we can deliver a finished project that we are both proud of and that meets all the necessary requirements. We wanted to ensure we avoided backing ourselves into a corner with our design, so we put significant effort into considering a multitude of different options to implement our idea. Through our research in section 3.2, we were then able to make informed decisions with our parts selection in section 3.3. We had to pay careful attention not just to the overall performance of our components but also to our budget. Since this project is funded out of our own pockets, we held several meetings specifically to evaluate and discuss our options when it came to our parts selection so we could reach a collective decision we all felt was appropriate. With the effort spent deciding on each aspect of this project, we are confident that we will be more than able to deliver a working prototype. The table below, *Table 3-5*, contains a summary of our parts selection. Additionally, we have included a reference to the sub-section which details the reasoning behind our selection for the corresponding part.

Part Selection Summary		
Component	*Part Selection*	*Section*
Linear Movement	8mm Linear Motion Rods LM8UU Linear Ball Bearings	3.3.1
Motor	Nema 17 Stepper Motor	3.3.2
Motor Driver	Trinamic TMC2209	3.3.3
Electromagnet	Uxcell 5V Electromaget Solenoid	3.3.4
Power Supply	12V Power Adapter	3.3.5
Voltage Regulator	LM2596	3.3.6
Microcontroller	ESP32	3.3.7
Communication	Blu-Fi chip on ESP32	3.3.8
Firmware	Marlin	3.3.9
Development Environment	PlatformIO	3.3.10

Table 3-5: Part Selection Summary

4.0 DESIGN CONSTRAINTS AND RELATED STANDARDS

With any project, it is important to consider the potential design constraints and to ensure you are aware of any standards that may relate to the project. Evaluating and adjusting for realistic design constraints and standards is necessary to deliver a working prototype. This section will go into detail about the design constraints we must operate under for this project. Additionally, we researched several standards that pertain to our project and further describe what they are and the impact they have on our overall design.

4.1 STANDARDS

The hardware and software used in this project come with many standards set by organizations such as the Bluetooth Special Interest Group, IEEE, and the HTTP Working Group. In order to effectively design efficient systems, it was important to understand standards relevant to certain areas of design before beginning implementation. Net-code and wireless communication in particular offered many standards which must be adhered to, or else many errors and bugs would be introduced which would block the MCU from delivering reports.

4.1.1 Bluetooth 4.2

The ESP32 MCU is designed to the Bluetooth 4.2 standard, also known as “Smart Bluetooth.” This standard is popular with IoT devices in particular as it allows for wireless transmission even under low-power conditions and is capable of bidirectional communication with HTTP servers. Thanks to the wide use of the ESP32, there are extensive libraries available which all adhere to the capabilities of not only BT 4.2, but the MCU as well.

Due to the abundance of power available in our system, the high-speed mode of BT 4.2 would be of most use to the board. Increasing the power to the receiver will allow for the full use of the Class 2 – or even Class 1 – Bluetooth transmitter found natively on the ESP32. This translates to roughly 10 meters of range in an unobstructed room for a Class 2 transmitter, or up to 100 meters of range in an unobstructed environment for the Class 1 transmitter. Whichever is used will depend largely on the design of the power supply as it would complicate the design for a relatively unimportant feature.

All Bluetooth standards are forward and backward compatible, ensuring that no matter what the age of the device we use to transmit the Wi-Fi credentials there should be no compatibility issue with the ESP32. Due to using the standard (also known as classic) Bluetooth mode, the connection will be confined to P2P operation, which will be more than enough to fulfill any needs of future features built on Bluetooth functionality, such as transmitting WiFi credentials or new play modes.

4.1.2 IEEE 802.11

This standard has been set by IEEE in order to unify communication over local networks and ensure backwards compatibility with most networking devices. Both Wi-Fi and Bluetooth fall into this category and understanding what these standards are is useful when referencing code libraries responsible for the networking aspects of this project. There are many addendums which have been added to the standard since its adoption in 1997, which are denoted by the letters following 802.11. Wi-Fi 5 – which will be discussed later – is contained within addendum 802.11ac and Bluetooth is part of the 802.11b addendum. It is important to note that while Bluetooth must conform to this standard, the standards for different Bluetooth versions are not the same as the LAN protocols which IEEE specifies.

By setting a consistent standard for all devices, 802.11 is meant to ensure that no devices cause unnecessary interference with the operation of unrelated devices. For example – the 2.4GHz band is populated by many devices including Bluetooth and some types of Wi-Fi. If only one device could operate in an area on this band, then it would be impossible to have multiple mobile connections on home Wi-Fi networks. This is also important for amateur radio broadcasts, scientific instruments which emit EM waves sporadically, and commercial equipment such as walkie talkies.

In regards to firmware, the data-link between devices must be handled properly in order to adhere to 802.11 standards. This takes the form of maintaining an logical link control (LLC) layer and a medium access control (MAC) layer. LLC layers handle checking the physical signals arriving from the network for errors and identifying what protocols are in effect based on the received data. MAC layers instead focus on security and access of a network – including packaging frames correctly before physical transmission. Thankfully, the examples and libraries provided by both Arduino and the manufacturer of the ESP32 MCU include code which adheres to this standard.

The ESP32, Wi-Fi router, and Bluetooth capable devices we use in this project are already equipped to handle the standards put in place by IEEE without any effort on our part. If we were to modify the ESP32 to broadcast constantly using hardware specifically designed to ignore incoming signals we could technically go outside of the parameters put in place, but as we are using the guide set out by the manufacturer on how to properly connect to the internet this should not be an issue.

4.1.3 IPC PCB Standards

One of the standards we must consider are the PCB standards set forth by IPC, Association Connecting Electronics Industries. IPC is accredited by the American National Standards Institute and publishes the most widely used standards in the electronic industry. The IPC standards are important because they ensure that manufacturers and designers alike build safe and reliable PCBs. The goal of IPC is to ensure quality, reliability, and consistency in electronics manufacturing [16].

There are many standards put forth by this organization regarding every aspect of PCB design and manufacturing. There are two standards in particular that are most relevant to our project. The first standard, IPC-2581, outlines the requirements and expectations for the exchange of design data between the PCB designer and manufacturer. This standard provides a format that must be followed to help ensure consistent results during production. The second standard addresses PCB design specifically and this standard is called IPC-2221. This standard provides guidelines for a variety of design aspects such as layout, materials, electrical properties, and parts lists [17].

There are several other common IPC standards, but many are mostly regarding the manufacturing of the PCB. While we are designing the PCB for our project, we will not be manufacturing the PCB ourselves. This does not mean that the standards do not apply to us, instead this means we must ensure that the manufacture we use to fabricate our PCB complies with these standards. We intend on using JLCPCB to manufacture our printed circuit board which has been certified by IPC.

4.2 DESIGN CONSTRAINTS

The primary constraints of the design come from various hardware components which support specific functionality. Aside from constraints imposed by the device itself, the typical use environment of a living room imposes many restrictions on the overall size and resources – Wi-Fi, main power, etc – which need to be considered by the design. Additionally, the end user is not intended to be a technical operator. Meaning that the level of complexity for any given stage of use should not be above the average high schooler: no coding, wireless communications, or mechanical background required. Finally, we also have constraints regarding our ability to complete the project. This includes time and financial constraints which we must consider very heavily throughout the duration of this project. This section will detail some of the main design constraints we must consider throughout the design, implementation and completion of this project.

4.2.1 Social

The typical use environment and ease of user experience were considered when we decided the engineering constraints for our Arcane Game Board. We envisioned this board being used on an existing table and wanted to make it so that it was fairly easy to transport. With this in mind we decided to try to make the cumulative weight for this project add up to under 20 lbs. There will be some heavy components within our board such as Nema 17 motors and a power supply. We felt that 20lbs would give us enough allowable weight to fit our components while being light enough to be carried by one person to be transported from place to place. This weight is also less than half of the peak weight that OSHA allows a single person to lift before a second person is required.

With transportation and ease of use in mind, we decided to keep the project under 24 inches by 24 inches. We felt that we could comfortably fit our components within these dimensions. We also thought that making a board any larger than this could complicate transportation by making the board too cumbersome. A chess board using 1.5 inch squares is 1 foot by 1 foot and we would need additional room to move pieces to a grave yard. We would also need additional room to position our electromagnet and linear rail system such that they extend beyond the checkered game surface area.

Remining fairly power efficient was also a priority for us. Our Nema 17 motors are rated to run at 2 amps and we have two of them. We don't expect these motors to reach their max rated torque throughout the game's runtime, so we expect to draw significantly less than the motor's peak rated current. The electromagnet draws around 670 milliamps of current when it is powered and the ESP32 microcontroller we are using draws around 60 milliamps of current when it is in an active state. With all this in mind we decided to require that we draw no more than 5Amps of current. With less current draw we can purchase smaller and cheaper power supplies which overlap with our weight and budget restrictions as well.

Our Arcane Game Board has two separate decision-making centers. One is the microcontroller within the board itself which is in charge of positioning the chess pieces and possibly relaying the chess piece location if we accomplish that stretch goal. The other decision-making center is our graphical user interface which is running on an external computer or phone. Players need to make their moves within the GUI for the board to replicate the turn. Naturally there will be a bit of latency between the devices. Part of the latency is due to the propagation delay of the streamed information over Wi-Fi and the larger part of the delay will be converting the player turn into movement steps that the microcontroller can interoperate and accurately execute. We decided that a delay of 2 seconds between the moment that the user finishes entering their turn in the GUI and the moment the electromagnet starts moving would be both achievable and the maximum acceptable delay to use the system comfortably.

Given that the Arcane Game Board only has a single electromagnet, we are only able to move a single chess game piece at a time. We do our best to minimize friction between each chess game piece and the checkered playing surface, but that friction is the limiting factor in how fast we can move our motors to position our electromagnet and thus a chess game piece. If we actuate our motors too quickly, the friction may overcome the force of the electromagnet and we will lose connection between the magnets embedded within each chess game piece and the electromagnet under the playing surface. We intend to move the electromagnet slowly and need to move at most 2 pieces around 1 foot from their place on the chess board to the chess piece graveyard in a worst-case scenario. We also need to maneuver the chess game pieces we want to move between the chess pieces we want to stay still which can add time and complexity to each movement. Because of these factors we decided that we want every possible chess move to be executed in under 15 seconds where the average move will take significantly less time than this.

4.2.2 Time

Another significant design constraint we have is our overall timeline for the project. We have from January 11th, 2021 until July 25th, 2021 to design and build our project. This gave us approximately 24 weeks and this time constraint has a substantial impact on our design and part decisions. Additionally, since our group is planning to complete Senior Design 2 over the course of the summer semester, we ultimately had less time overall to build our project due to the fact the summer semester is the shortest.

In order for us to ensure a working prototype by our deadline, we had to pay careful attention to our deadlines and plan our work in advance. During the initial concept idea phase of our project, we estimated our deadlines and milestones we must reach over the course of 24 weeks. We wrote down these milestones in *Table 8-2* which can be found in *Section 8 Administrative Content*. The purpose for this table was to give our team a visual idea of the amount of time we have as well as setting a loose time frame to accomplish the major portions of this project. When a project has tight deadlines, it is crucial to carefully plan the time it will take to finish key components of the project.

To further help us work within our time constraint, we held weekly meetings and kept a running list of tasks to be completed which we would then update each week. Each task was explicitly assigned to a team member which helped us greatly in dividing the work evenly. Another strategy we implemented was setting deadlines for our team that were earlier than the real deadline. This ensured that we would still meet our requirements even if something came up that delayed our progress. This was especially useful throughout the report writing because we were able to avoid last minute ‘cramming’ to get our pages done.

Lastly, another major time constraint we needed to consider was lead times for our parts and components. We needed to be very conscious of potential delays for ordering the necessary parts. Due to this, we prioritized selecting and ordering our parts early on to avoid a situation where we would be unable to acquire a part in time to complete the prototype. This was especially necessary because we have very little time to order and build our project during senior design 2, therefore the earlier we could select and order components, the better.

4.2.3 Economic

A major design constraint for this project is our budget. This project is entirely self-funded therefore we must be very mindful when selecting our parts. We set a goal to keep our project well under $500 since we knew that this amount would be substantial enough to build the project. Ideally, we want our budget to be as low as possible, therefore, we are not able to get the most expensive parts since we must balance efficiency with cost. Additionally, the goal is that this chess board can theoretically be reasonably priced from a consumer perspective, therefore, by keeping our build costs down we can essentially create a more affordable product. Unfortunately, keeping costs low does have drawbacks, however, these drawbacks minor and will not prevent us from delivering a project that meets our design goals. Lower budget means that our movements may need to be a little slower and a little less responsive than if we had more money to spend but this will have little impact on the gameplay overall.

To ensure we stay as close as possible to the budget we outlined in section 8.1, we decided the best course of action was to maintain an excel spreadsheet containing all current purchases. Additionally, we agreed as a team to avoid purchasing any items without everyone approving the purchase. This method of handling our finances, while practical and fair, also imposes a slight time constraint as well. This time constraint will typically be negligible; however, it is important that we give ourselves ample time between discussing the purchase and needing to complete the purchase should any issues with the budget arise.

4.2.4 Environmental

Now more than ever it is important to consider the environment impact of technology and our project is no exception. While our project doesn't necessarily have a direct impact on the environment, our project does use electricity to run. Therefore, it is important that we do thorough research into ways we could be more energy efficient in our design. Due to the nature of our project, it would not be practical for us to implement an alternative renewable power source such as solar power.

Another way we are trying to be environmentally conscious in our design is by minimizing our material waste. We will use recycled or scrap wood for parts of the physical game board construction when we are able to do so. Additionally, we are making our part purchases in larger chunks so we can utilize the option to ship our components in fewer boxes when the option is available. This also cuts down on packaging waste along with using parts that we already own wherever possible instead of purchasing new parts. Overall, our project does not have a significant environmental impact, however we are reusing and repurposing materials when we are able to so we can further minimize the negative impact on the environment.

4.2.5 Political

We researched potential political constraints that could be applicable to our project, however, in general there wasn't anything particularly relevant to our project. Our project is, at its core, a game board, similar to those found in most households. Due to this, we concluded there were not any relevant political constraints for the Arcane Game Board.

4.2.6 Ethical

The ethical implications of any product should always be carefully evaluated during the research and design of the product. Ethics guide our behavior in everyday life, we are constantly making decisions based on our ethical beliefs. Therefore, when we are designing a product to be successful, we need to consider if the public would perceive anything in the design and research as potentially unethical. Examples of potentially unethical practices would be things such as animal testing or exploiting cheap labor.

Our team has been very careful to ensure each step of our project is morally sound. This includes crediting any material we used to assist with our design and ensuring anything we use in our final design that is not uniquely designed by us falls under fair use. Additionally, we made sure to research the components we are using to make sure that we are not financially supporting any unethical practices in the creation of our project.

4.2.7 Health and Safety

Health and safety of our potential users is a priority of ours, especially because this product is something which can be used by all ages. This means that this project needs to be completely safe for children to use as well as adults. In order to maximize the safety of our device, we ensured there are absolutely no exposed wires or electronics that could be potentially dangerous for a child.

All electronic components are housed inside the board and not easily accessible. Additionally, we implemented all necessary protections in our circuit board to avoid potentially dangerous circumstances like over-heating or shorting.

Another health and safety consideration we made was the weight of our game board. Game boards are meant to be picked up and moved around fairly regularly, therefore we want to avoid any potential injuries that could be caused by the board being too unreasonably heavy. Our goal is to keep the board as light weight as possible and according to our initial estimates the weight should not pose any significant health and safety problem. Additionally, we want to ensure all the wood we use in our physical board is sanded down and finished to avoid potential splinters or cuts due to jagged edges.

Overall, it is our goal to make the Arcane Game Board safe to use. We avoided using potentially harmful materials in our design and mitigate any risks that accompany the operation of an electronic device. This includes making the project sturdy, so it is less likely to break and hurt someone when moving it. As engineers, it is our responsibility to consider the end-user experience of our product and design realistic safety measures for those users.

4.2.8 Manufacturability

Manufacturability is a crucial part of our project and it was kept in mind during all stages of design. Economic constraints aside, we wanted to ensure that every component's raw material could withstand the stresses it would be under and that each piece could be realistically fabricated. One key example was our playing surface. We needed a material that was low friction and we preferred that the material be transparent and laser engraver safe. The clarity of the material would allow is to see the inner components of our board in motion as the board moved pieces around. We wanted to make sure our material was laser engravable so that we would not have to paint a checkered pattern over our playing surface. These manufacturing and design constraints guided us into purchasing acrylic sheets for our playing surface.

Choosing materials that are easy to machine and are readily available is a huge part of the game board's manufacturability. It is for this reason that we did not choose any obscure components or materials and tried to adhere to popular systems and standards. Metric hardware is mass produced and readily available throughout the world. Aluminum and wood have been extremely popular material of choice for construction for many years now. With the explosion of 3D printer popularity, the world supply and manufacturing of Nema 17 stepper motor and the GT2 pully standard has also exponentially increased making these parts easy to source.

4.2.9 Sustainability

The overall sustainability of our project is something that was important to our team as well. It is particularly important because this product would be a fairly expensive purchase and it needs to last a long time to become a worthwhile investment for the consumer. We considered the lifetime of our components to ensure we designed a reliable product. This product should be able to run for extended periods of time and be able to work just as efficiently when not used regularly. It should be noted that the lifetime of certain components may exceed that of others, therefore, the lifetime of our product would be based off the component with the lowest lifetime; in our case this would likely be the motors or pulleys.

In addition to the sustainability of our components, we must also consider the maintainability of our software and firmware. The software must continue to be effective over time which would include updates to the software to avoid any potential bugs. This contributes to creating a sustainable product. Furthermore, we must pay careful attention to the components on our PCB and ensure that our PCB design does not exceed recommended operating ranges to avoid degrading the components at a faster rate.

5.0 PROJECT DESIGN

The robot consists of a system referred to as a cartesian robot. This allows for movement across a plane in X and Y directions – just like cartesian graphing systems. Firmware is used to guide the robot to move to relative or absolute positioning, allowing for simple piece movement commands to be given from the higher-level software. The on-board firmware converts the chess piece movement into actual robot movements – including event handling if needed. A control app running on a desktop computer, laptop computer, or smart phone is used to send the simple commands to the micro-controller on the board. The highest-level software consists of a UI which informs the user of the state of the board.

If we were to implement piece tracking the MCU would contain all of the necessary hardware and software changes. With piece tracking using sensors which trigger when encountering electric fields being routed through a selection multiplexor array, the MCU would be capable of polling the entire board for piece positions in less than 50ms. Given the previous state of the board and the amount of time required to move a piece, we could use an algorithm to deduce which move was moved and when.

The AI stretch goal was handled entirely on the web-controller end of things – as that is where the actual chess moves are transmitted from. We had the option of either including a chess engine which is run entirely by our server or integrating with a pre-existing chess engine offered through a site such as lichess.org.

5.1 INITIAL DESIGN ARCHITECTURE

The development of *Figure 2-2* in our initial design and conquer document was instrumental in our design architecture. In this section we will go over what our initial concept for how everything in our project will interface with each other. If you haven't done so, please look at *Figure 2-2* in section 2.6 of this document to get a visual representation to aid in your understanding of how the features about to be described will interconnect.

After deciding what we wanted to do for our project, we weighed our priorities started considering what it was we needed to do to get a decent grade in the class. We knew that as EE and CpE majors we were required to make a PCB. So, we placed a PCB in the center of our block diagram and started building outward from there. We knew that we were going to have some form of electronic components in our project, so we would need to incorporate some sort of power supply in our design.

Next, we had to consider what sort of electronics we wanted in our project. In order for us to decide that we had to break down the functions of the game board to get an idea of which electronics would be appropriate to include. We had to get some sort of motion done so we knew we would need some sort of motor. Where there are motors there needs to be a method of motor control and homing, so we agreed we would need a motor controller or controllers to handle a motor or motors. These motors would need to support a homing feedback feature – or we could optionally include limit switches on each axis which would be used to find the limits of travel during startup. We also needed something to coordinate the motion of motors using CoreXY logic, so we needed a microcontroller to interface with the controllers to direct the motors.

With the basic movement requirements set, we started to consider what sort of gameplay our arcane game board would be capable of. We wanted to be able to play single player, against an opponent in the same room or against an opponent somewhere else in the world. This would require some sort of chess logic to make moves for a simulated player. We doubted that a microcontroller would be able to process inter-board communication and motor coordination, so we decided to split the load. We would need to have an application that was run on a processor outside our gameboard. That app would have to communicate to our microcontroller so that the microcontroller could focus on coordinating game piece movement. This communication would have to either be hardwired or done over some radio connection like Bluetooth or Wi-Fi. We decided to use a non-physical method of communication between the app and the microcontroller so that we could simplify our PCB, improve the user experience, and not worry about being locked into a single physical connector type.

Game piece detection is a stretch goal for or project, but we spent time thinking about how we would implement this if we decided to include it. Our team is weighing a computer vision or hardware solution. Game piece detection is crucial for facilitating in person game play and for path planning when moving game pieces. We don’t want to have a moving piece collide with other pieces on the board and we need to maneuver between pieces when one piece takes another and moves the dead piece to the graveyard. For a computer vision solution, we are considering using an Nvidia Jetson Nano paired with a USB webcam to detect which pieces are in play and where they are on the game board. The Jetson Nano would have to connect to Wi-Fi and communicate with the web app to let our game logic program know what the current state of the game is. The hardware solution would involve a bunch of push button under a flexible membrane on the play surface. A player would have to press down on a game piece they wish to move. This would signal to the microcontroller that piece X is about to move. The player would then move the piece by hand and press down on the new square they moved Piece X to. This would send a signal to the microcontroller indicating the piece X is now at Y. The microcontroller would then have to communicate top the web app to let it know piece X is now at piece Y so that the game logic can have an updated version of the game state.

A computer vison based approach would need to have either a neural network that was trained with a large set of images showing an overhead view of a chess board. This network would need to be able to discern piece type and whether the piece was a black or white piece. Once a piece was identified with a high enough confidence level, we would need to divide the image into discreet areas to identify which square that piece is on. Finally we would translate that information into a datatype that our web app would understand so that we would know what move a human player made.

Another possible vision implementation is having active Infrared Beacons within each game piece. This would take form as either an individual or small assortment of IR LEDs embedded within each game piece. We would need to embed a small circuit board to power these LEDs and strobe them in discreet PWM frequencies. We could then have the camera search for points of IR light using thresholding and recognize which piece is which by identifying the strobe pattern. We would once again need to divide the image into discreet portions to identify which square any piece is sitting on at any time.

The app and microcontroller would need their own separate forms of software to be able to do their jobs effectively. We broke that down to firmware residing on the microcontroller and some sort of graphical user interface, GUI, for our app running on something like a laptop or smart phone. The firmware would be written using C++ to leverage existing codebases and development environments familiar to the team. React was used for the GUI for similar reasons. Our house of quality was also cited as a reason to use the graphical interface as it enhances the user experience when playing the game compared to a connection system which did not use a UI.

To house the physical components, we would need to design and build a housing for everything to mount to and work together on. Design was made into its own CAD task and the building was split into both construction and wiring. We would need to fabricate parts to facilitate motion of game pieces, make the pieces themselves, and style the game surface to look like a chessboard. We need to wire all components such that the proper logic signals connect electrical components that need to communicate with each other. We also need to route power to all the necessary components off of our designed power supply.

These tasks were discussed and split among our group members based on the strength we each felt we had and would best be able to complete. Kayla being the sole EE of the group would play a hand in the power supply and PCB design. Fernando has the most experience with CAD software in the group, so he would handle design and aid in other tasks. Lucas has had plenty of experience with the front end of web applications, so he took on making the web app. Anton is a 3D printing hobbyist and has spent extensive time reading through firmware and modifying his printer, so he gravitated towards firmware. As shown in *Figure 2-2*, all the work we do in our separate tasks will culminate in the final working arcane game board.

5.2 HARDWARE DESIGN

The majority of the hardware design was done in the CAD software SolidWorks. This is a very popular CAD software that is available for free for UCF students. There are online CAD libraries, such as grabcad, that are filled with CAD files for many of the off the shelf components purchased for this project that are compatible with SolidWorks. This allowed us to save a significant chunk of time by not having to create models for every component we purchased.

SolidWorks let us lay out all of our components and constrain their movements to reflect how they would behave in the real world. This would give us an idea of how everything would fit together and how each component would interact with the others. This helped us narrow down how things would be assembled and what dimensions we would need for everything to come together.

There were many parts that were not designed to automatically work together. For these instances, new CAD models were made that would later be 3D printed. These models include the carriages that slide on the linear guide rails that are responsible for the electromagnet and game piece placement. Since several members of our team own 3D printers, we decided to make models for any pieces that we could safely manufacture at home. This would save us some money as 3D print filament is relatively cheap and we would not have to wait for shipping or manufacturing lead times if we wanted to get them made elsewhere.

The 3D printing material we designed around is PLA. This is a very cheap and easy to work with material. With the help of our preferred free slicing software, see section 3.2.5.6 CURA Slicing Software, we are able to slice solid parts and make them hollow with an infill support pattern to greatly cut down on material use, print times, and cost, while having a very marginal impact on the component's overall strength.

We designed our PLA components to interface with other hardware using our preferred embedded nut and bolt method. Parts are designed such that a nut can fit snuggly into a cavity in the 3D printed pieces with a through hole to line up an external bolt with the embedded nut. Any parts that are attached to our 3D printed PLA components needed an appropriately sized bolt and washer to spread the tightening force over a larger surface area than the bolt head, Spreading the force this way reduces the likely hood of cracking the plastic in our parts.

Other attachment methods include using helicoil inserts or heat seated inserts. After a bit of research, we found that helicoil inserts only marginally improve the load that the plastic can bear before stripping when compared to threading a bolt directly into plastic. Heat seated inserts are a viable method, but we had concerns on how much the center of the threaded hole can shift while the insert is melting through the plastic. We were already very familiar with using slotted nuts and hadn’t seen issues with holding power or drifting hole centers, so we stuck to what we knew would not have an issue.

Once all the parts and small pieces of hardware were placed in a SolidWorks assembly file, we count up the required amount of hardware. We also use the measure tool in SolidWorks to verify the dimensions of any components we need manufactured by methods other than 3D printing. This way when we purchase parts such as wooden base boards, we can have them cut to the sizes we need for assembling our project. None of us have direct access to the precision cutting tools which would be required to trim down oversized boards on the fly – so that is a particularly important measurement to get right on the first try.

5.2.1 Linear Motion

It was decided that a CoreXY motion system best suits the needs for this project due to the fixed motor system, reduced weight on the moving parts, and the available code base which can be used to accelerate firmware development. It satisfies all requirements of our house of quality, with Section 6 going into the details of the construction. It is very possible to build an H-bot design which will allow for the dimensions we desire as well as minimal wasted space underneath the play surface.

Marlin has supported CoreXY and H-bot cartesian robots in its main branch since 2013, with few updates to the code that will be relevant to this project. From a firmware perspective there is little difference between the 2 designs, as the motion of the device is governed by the same equations. Put in plain English, this means that both motors spinning result in axial motion while a single motor spinning results in diagonal motion. See Figure 3-6 for an image detailing how motion occurs as well as the equations used to calculate movement.

Sensorless homing can be achieved by moving the electromagnet head and then listening to the UART ports to a stepper motor driver which supports sensorless homing. We chose Trinamic TMC2209 stepper motor drivers which – when enabled – send a signal to the MCU when there is significant resistance on the motor during use. If the robot has been constructed and calibrated correctly there should be no way to reliably guess which motor will encounter resistance first. This means that both stepper motors must be driven with UART configured, otherwise small errors will occur every time the machine homes.

5.2.2 Power System

The power system is extremely important and essential to the overall design because it is supplying power to all of our components. There were many things to consider when developing a power system because it must be able to supply a sufficient amount of voltage and current throughout the entire design. It can be very easy to damage components if the wrong amount of voltage or current is supplied, therefore each component had to be carefully considered by looking at the corresponding datasheets. Each component has a recommended operating range given by the manufacturer and we used this information to design an appropriate power system. For components that need varying amounts of voltage input we used voltage regulators to achieve the necessary input; section 5.2.2.1 will go further into depth about this.

Our project is designed such that it is powered via a wall outlet. The US standard for wall outlets is 120VAC and 60Hz however our components and voltage regulators require a DC input. Therefore, we had to decide how we wanted to go about converting the 120VAC to something we can use. We ended up deciding to use a 12V power adapter wall plug which connects directly to our PCB, this means that our input voltage is 12VDC. From there, we needed voltage regulators to step-down the input voltage for our microcontroller and electromagnet.

The figure below, *Figure 5-1,* depicts a block diagram of the power system design, including both voltage and current demands discussed in section 3.3.5. The green box on the diagram represents the printed circuit board and all boxes contained inside are included in our PCB design. The boxes outside of the blue area are components that connect directly to the PCB, this includes the 12V adapter, the electromagnet and the Nema 17 stepper motors.

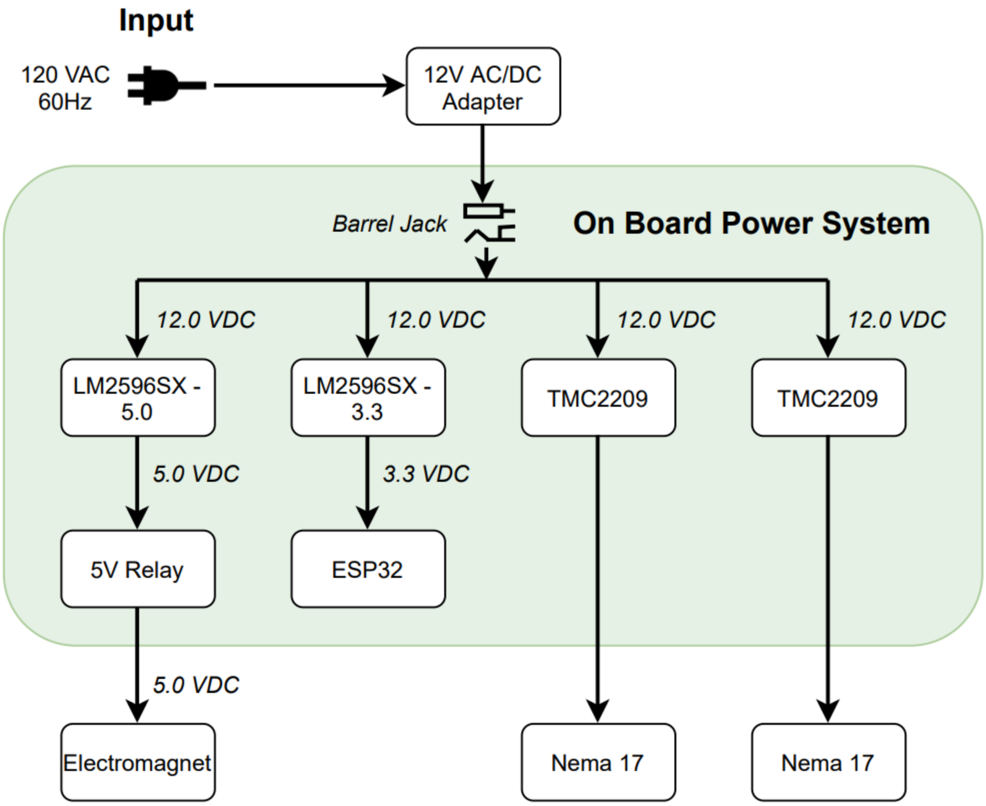


Figure 5-1: Power System Flowchart

5.2.2.1 Voltage Regulators

As discussed in section 3.3.6, we are using the LM2596 family of voltage regulators for both the 3.3V and 5.0V. This voltage regulator is manufactured by Texas Instruments and is a switching step-down converter able to convert a 4.5V-40V input voltage into fixed output voltages of 3.3V, 5.0V and 12V. This is perfect for our design since we are working with a 12V input voltage and a 3.3V and 5.0V output voltage. The design for this voltage regulator needed one input capacitor, a catch diode and an LC output filter. Since we are using the fixed output versions of the voltage regulator, we did not need an additional resistive divider to set the output voltage.

The datasheet for the LM2596 gives us guidance on how best to choose the values we should use for our voltage regulator circuit for all the capacitor, inductors and diodes [18]. Starting with the capacitors, it is recommended to use a low ESR aluminum input capacitor with a voltage and current rating of at least 1.5x the load voltage and current . This input capacitor’s purpose is to buffer the input voltage for transient events and should be placed as close as possible to the IC. In addition, it is required to place a catch diode between the output and ground with a rating of at least 1.3x greater than the output load current. For best performance, the diode should be a Schottky diode selected from the table given in the datasheet.

For the output LC filter, it is recommended to use a 33uH inductor along with the 220uF capacitor we discussed above. The output capacitor should also be a low ESR electrolytic capacitor with a voltage rating of at least 1.5x the load voltage. However, the ESR cannot be too low as it could then result in an unstable feedback loop. It is important we choose an inductor with adequate current rating because exceeding this rating could cause the inductor to overheat which could ultimately result in the overheating of the regulator itself.

It should be noted that we will be using this circuit design for both the 3.3V and 5.0V fixed switching regulators. We will further expand upon the overall schematic design later in 5.2.4. Overall, since we carefully followed the manufacturer's datasheet for this component, we had little issue with our voltage regulator design since we were operating under all the recommended conditions.

5.2.3 Electromagnet and Relay

We selected a 5V P25/20 electromagnet solenoid with a 50N holding force which is toggled on and off to move the pieces across the board. The electromagnet sits in the middle carriage of the robot underneath the board. This is the one electrical component that is moving across the entire length and width of the board; therefore, we needed take this into consideration with the cable management for this component. We want to ensure that our wires have little stress put on them to avoid failure over time. Additionally, we also want to avoid the wires getting caught on anything at any point. We can do this by either binding the wires to keep them organized or we could use a flexible cable long enough to reach each end of the board. We could also use a coiled cable which would allow us to minimize the amount of slack present when the electromagnet is closest to the printed circuit board. By minimizing the amount of slack, our wires would be less likely to get caught on any other components. Ultimately, we were able to manage with having our wires just long enough to reach and did not have any issues with the wires getting caught up on anything.

In addition to the mechanical operation of the electromagnet, we also must be able to toggle our electromagnet on and off based on the corresponding signal from the ESP32 microcontroller. Our first option was a relay which is an electrically operated switch [19]. A relay would allow us to take a single control signal from the ESP32 to toggle the voltage supply for the electromagnet. There are several advantages to using a relay, most notably its ability to handle high voltages and currents. However, relays do tend to be quite large and have a relatively short life cycle which is something to consider as well. Additionally, relays need a fly-back diode to prevent the degradation of the switch contacts. Our other option is to use a MOSFET, which are typically much smaller and has relatively low power consumption. A MOSFET is a type of insulated-gate field effect transistor, specifically it is a metal-oxide semiconductor FET. Field effect transistors are commonly used as switches in electronics because they have a definitive "off" and "on" state.

Ultimately, we decided to use a relay in our final PCB design. We chose the SRD-05VDC-SL-C electromagnetic relay manufactured by Songle. This relay is Form C meaning it is a single pole, double throw (SPDT) switch and has a nominal coil voltage of 5V. It requires at least 71.4mA nominal coil current which is larger than the ESP32 is able to output. Therefore, to operate our relay, we must use a 2N2222 N-P-N transistor to amplify the current.

5.2.4 Microcontroller

Compiled firmware will be uploaded to the microcontroller via a serial USB connection or via a wireless build upload. PlatformIO supports both of these options in their Microsoft Visual Studio Code extension, making a debugging workflow relatively simple to set up. While uploading firmware to the device it will be possible to observe any errors or warnings that are flagged by the system, as well as actions which can be taken to mitigate them.

Unit tests will be developed in order to confirm the functions performed by the ESP32. These are intended to highlight exactly which elements of the device are failing – whether that be hardware, software, or some mix thereof. This can be done without the custom PCB design's delivery through the use of development kits procured well in advance of said PCB. As code is developed, it can be rigorously tested in order to speed up development time and avoid cascading failures when assembling the final board.

5.2.5 Hardware Schematics

Several components required external parts as part of their designs. Most notably our voltage regulators needed additional external components. We discussed in depth the necessary values for each component in section 5.2.2.1 and were able to come up with a schematic for both the 3.3V voltage regulator and the 5.0V voltage regulator. We used these schematics to assist in designing our power system on our PCB. The schematics were obtained using *TI's WEBENCH Power Designer* tool. This tool is very helpful to ensure we design our PCB with the appropriate components for our design requirements.

Both figures below were generated courtesy of Texas Instruments under their terms of service [20]. *Figure 5-2* depicts the LM2596 schematic which gives us an output voltage of 3.3V [21]. This voltage regulator is used to supply power to the ESP32 Microcontroller. *Figure 5-3* depicts the LM2596 schematic which gives us an output voltage of 5.0V [22]. This voltage regulator is used to supply power to the relay which subsequently powers the electromagnet solenoid which is used to move the pieces across the board. We did slightly alter the inductor and capacitor values for this these schematics to better fit our overall design. We made these alterations based on suggestions made in the LM2596 datasheet, finding we could use the same value of inductors and capacitors for both designs. This was really helpful when it came time to order our PCB since we did not need several different extended components for both designs which ultimately saved us money in the end.

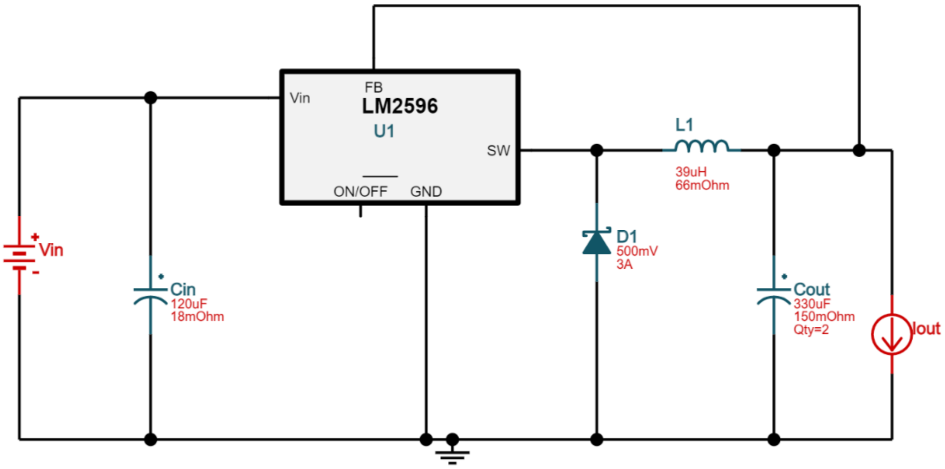


Figure 5-2: 3.3V Voltage Regulator Schematic, Courtesy of Texas Instruments

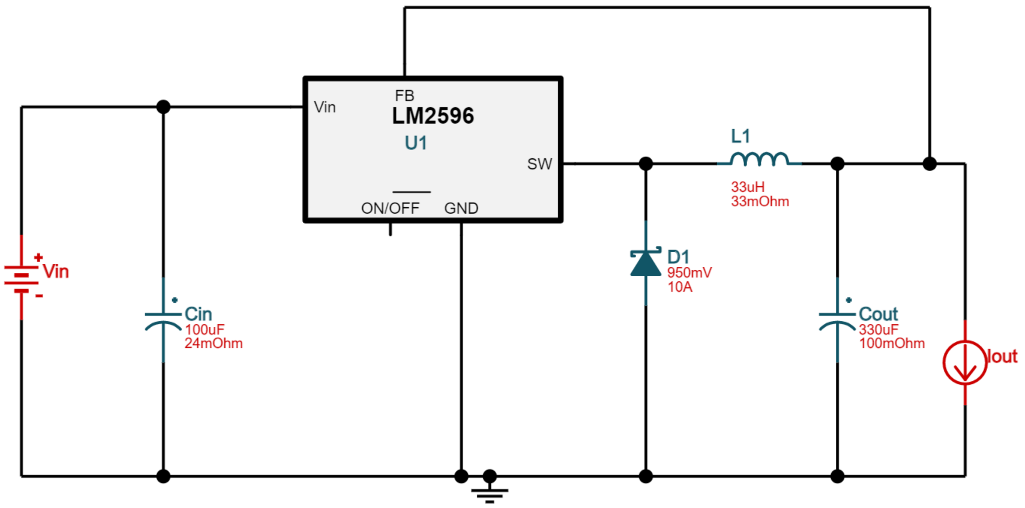


Figure 5-3: 5.0V Voltage Regulator Schematic, Courtesy of Texas Instruments

The other hardware schematic that we needed to determine was the schematic for our TMC2209 motor drivers. We used Fusion360 to draw up the schematic for our TMC2209's which would have been used in our PCB design. We needed two schematics since our hardware design includes two motors and the schematic for both motors will be the same. Using the datasheet to determine the necessary component values and proper connections, we developed the below schematic, *Figure 5-4*.

We intend to supply our motor drivers with 12VDC voltage input and the Vref voltage is approximately 1.1V which we used to calculate the necessary values of R6 and R7. From the datasheet, we know we need to connect a 0.11Ω sense resistor to pin 23 and pin 27. Additionally, C3 – C6 are used for filtering and are connected as close as possible to a ground pin. The datasheet recommends low ESR electrolytic capacitors for the filtering. The capacitor connected to VCC_IO is needed to deal with current ripple. VCP needed to be tied to VS with a 100nF capacitor, as shown on the schematic. Similarly, the charge pump capacitor output should be tied to input with a 22nF capacitor [23].

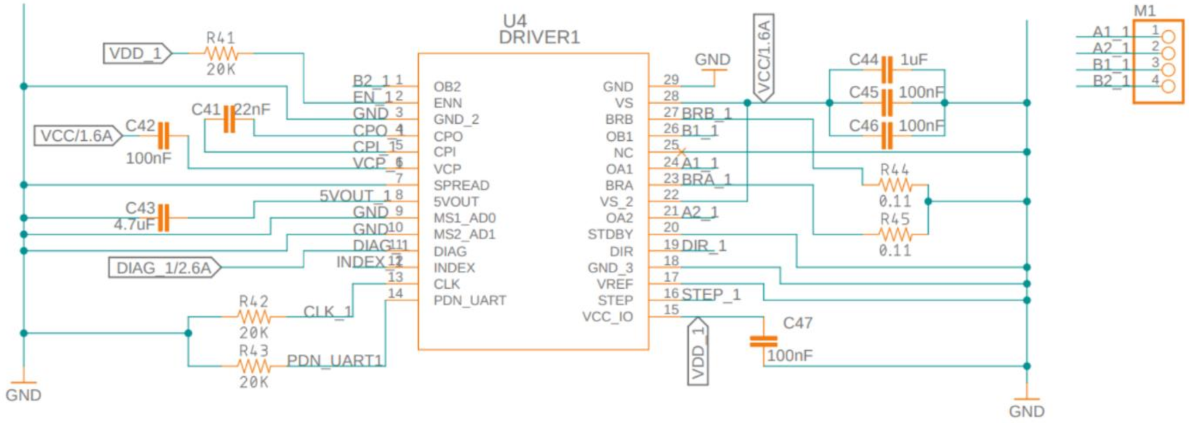


Figure 5-4: TMC2209-LA Schematic

These are the three most complex circuit schematics for our design overall as they involve the most external components. The next step after these schematics is to connect them appropriately to each other and to the ESP32 microcontroller. This can get a little tricky, but it was absolutely necessary that we connected all of our individual schematics together correctly in our final design.

Unfortunately, due to component availability, the TMC2209 went out of stock before we could order our PCB. As a result, we needed to redesign the whole schematic and instead make it so we could slot in our devkits [24].

5.3 Software Design

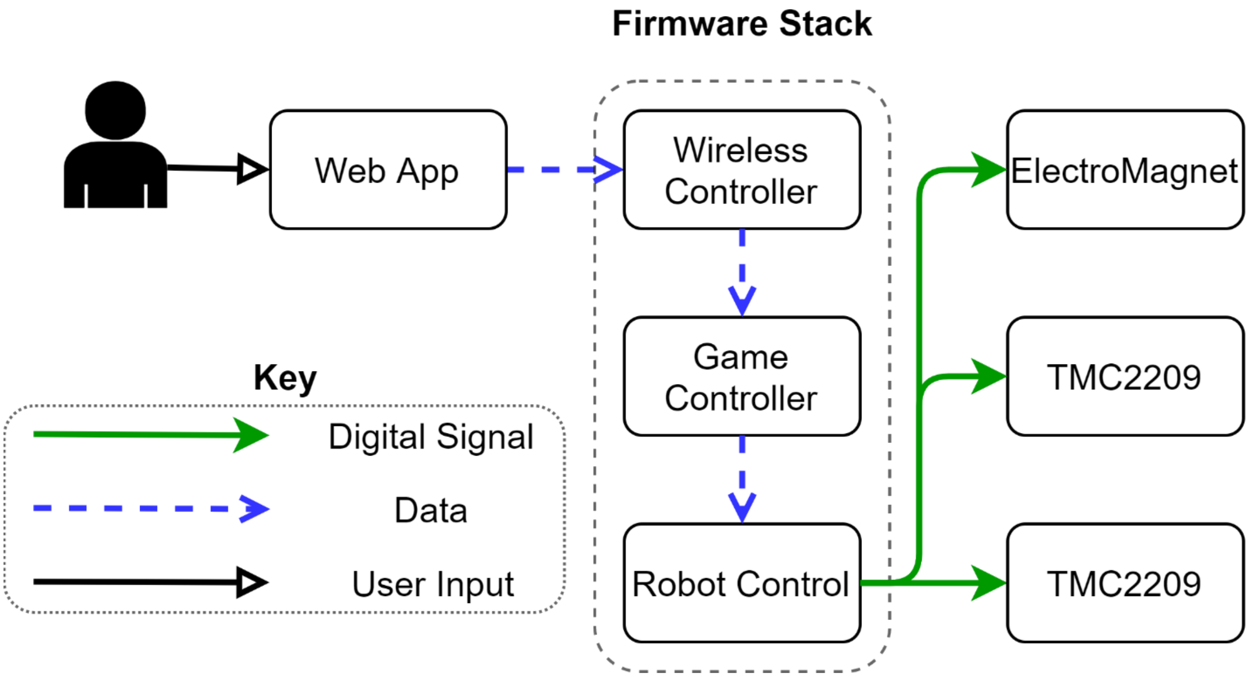
In order for the game board to function as intended, the software both on and off of the board communicates wirelessly while allowing for execution time. The most constraining factor the time required for the mechanical movements of the cartesian robot to complete. To account for this, we implemented a queue structure on the ESP32 with enough room to queue hundreds of moves – more than enough for the players to complete the game

Existing libraries for the ESP32 were used during the development of firmware to both speed up development, as well as avoid potential bugs which would be introduced when developing complex systems from scratch. Rather than using Marlin – as initially planned – the firmware is instead built using our own movement planning software. The libraries we use instead power our wireless communication, as well as supporting the TMC stepper driver functionalities.

It was essential that any functions tied to hardware were vetted before final part selection. The ESP32 has several key limitations which are noted in section 5.3.1, as well as feedback functionality from the Trinamic stepper motor drivers. Since the off-board app is on a windows-based computer there are far fewer limitations, and the libraries used are documented in the PlatformIO project for the convenience of future developers – they will be automatically downloaded upon loading the project in VSCode.

5.3.1 On-Board Software Design

The On-Board Software was designed with a 3-layer, object-oriented approach to the overall operation. At the highest layer is the Wireless Communication. At this level, a websocket connection is maintained over Wi-Fi which delivers serialized JSON files from the server. Before the information is passed down a level, the information is extracted by helper methods and converted into Structs. The next level is responsible for maintaining the game state via path planning – requiring an internal record of the board to be kept. The final and lowest layer is the actual firmware required to drive the TMC2209 stepper motor drivers. See below for an illustration of this concept:



A core program – the ArcaneCore – handles high level logic, such as passing data indicated by the blue lines above, reading the states of each controller class, and following simple actions based on that. The program was designed with the intention that ArcaneCore would be as light and straightforward as possible, with all complex logic contained within the deeper classes we designed.

The WirelessController is hardcoded to a Wi-Fi network and a single IP address to connect to a websocket. As events register to the websocket, they raise flags specific to events such as connecting, disconnecting, and receiving data. A lambda method tied to each WirelessController is initialized with its parent class – allowing for it to access class members. This is vital, as deserializing JSON moves takes time which cannot be taken within an ISR and passing 'this' pointers to static functions is generally bad practice.

The GameController layer contains the digital representation of the board, as well as the functions required to plan piece movement. A queue is publicly available via get functions, allowing the deeper RobotControl class to retrieve moves when needed. Not shown above is the helper classes Piece and Queue, which are used to keep track of the individual pieces and moves respectively. Rather than give each piece several points of data such start position, current position, faction, and piece type, we devised a standard format for an 8-bit unsigned integer which would contain everything but the current position.

Fact.	**Type**			**Starting Col**			
B	B	B	B	B	B	B	B

The starting position of any given piece can be inferred from the faction and type when given the starting column. If the piece is a white pawn it begins in row 2 on the board. Any other type of white piece is in row 1. Black pieces can have their starts determined similarly as well.

Using this ID structure saves memory in 2 ways:

Piece Memory Size: Each piece now only needs a pointer, an 8-bit ID, and the current position expressed as a std::array with 2 8-bit ints. This effectively collapses 4 bytes of data into 1. (96 bytes in total.)

Procedural Piece Generation: The actual memory saving measure. Rather than save 32 pieces worth of data in a configuration file, the program instead uses bitwise operators to assign pieces according to how many columns and rows are present on the board. This also has the added bonus of letting games with more than 16 pieces be played, with rooks, knights, and bishops repeating their pattern until the queen and king are placed in the center. This saves nearly half a kilobyte.

For the ESP32 the above is inconsequential when compared to the 5MB internal flash storage. This measure is mainly useful for porting the code to smaller MCUs. The real memory savings would need to come from building a lighter-weight version of the wireless protocols and stepper motor configuration.

Aside from keeping track of the board state, GameController objects are also responsible for placing robot moves in a queue to move pieces in a specific path. Path planning is simple compared to many other robotic systems – using only simple algorithms which are computationally inexpensive. The robot's position at the end of each move is tracked, allowing for dX/dY to be calculated extremely simply. This delta is fed into the function described above (xyToMotors()) and then fed into a Queue. A flag indicating whether the electromagnet is enabled or disabled is also included, allowing the RobotController to process the move accordingly.

A typical piece – when not capturing – only requires the robot to move to its current position, activate its electromagnet, and then move to the piece's final position. If a piece is being captured, that piece must be retired first. This is done via avoidance transposition. Our pieces are small enough, and the magnetic fields are weak enough, that pieces can pass by one another without moving them. By shifting the piece to the borders of the squares, rather than the centers, we can move pieces in either the X or the Y direction without interfering with any other piece.

When retiring a captured piece, we first pull up the piece's ID. This gives the desired location. Then, according to the overall delta, we transpose the piece to the square borders. The piece travels to the column detailed in its ID, then to the row. Finally, the piece is shifted back into place via another transposition. The robot is then instructed to move the other piece.

Knights were the final movement type we would need to address. After programming the retirement function, this was simple. By calling a transposition according to the overall delta of the piece's movement, the knight could never collide with other pieces. To ensure that the knight didn't move extra tiles, the magnitude of the original dX and dY values is decreased by 1 – as the transposition moves it a whole tile on the diagonal. This results in an optimized route, as well as avoiding the other pieces on the board.

The deepest we implement is the RobotControl class. This is where movements are directly converted into signals to the PCB. It was critical to understand what pins attach to which trace, as well as implement a centralized config file where the pins were defined. That way – in the event that the design changed significantly – our team would be able to quickly adapt without losing development time.

We use a CoreXY system for the cartesian robot. In order for the RobotControl to properly execute moves, the following kinematic expressions were used.

$$\Delta X = \frac{(\Delta A + \Delta B)}{2}, \qquad \Delta Y = \frac{\Delta A - \Delta B}{2}$$

$$\Delta A = \Delta X + \Delta Y, \qquad \Delta B = \Delta X - \Delta Y$$

VARIABLE	MEANING
ΔX	Linear movement on the X plane
$\Delta \mathrm{Y}$	Linear movement on the Y plane
ΔA	Rotational movement seen by the belt on Motor A
ΔB	Rotational movement seen by the belt on Motor B

Table 5-1 Movement Equation Variables

When converting the dX and dY of robot movements (derived from simple end-start equations) into dA and dB, the software multiplies the steps-per-mm and mm-per-square config options to allow the code to reference dX and dY in terms of squares, rather than in steps or mm. This allows for the direct conversion of the JsonMoves output by the wireless controller into dX and dY motions instead.

During the execution of commands in the buffer, there will be a breakout interrupt in the event of many invalid commands. In the case of 5 sequential invalid commands, the MCU will first post to the HTTP server that there has been an error, pausing the game. The last 5 commands logged in the server will be resent and attempted again. If the commands are still invalid, the game will be aborted and an error message with the command history buffer on the MCU will be saved.

5.3.1.1 Development Environment

We opted to use PlatformIO – an extension of Microsoft's Visual Studio Code – to develop the firmware of this project. There were many reasons to use it over one of the competitors, but the primary reasons were the price (free), the ability to use it alongside tools which the team was comfortable with, and the built in support for embedded unit tests in the form of the Unity Testing Framework. As an added bonus, it also allows for concurrent development on the same physical board wirelessly; circumventing some challenges created by current global events.

There are some functions which we wish used which are typically found inside of the Arduino IDE that many assume are features specific to their platform – such as attachInterrupt and dettachInterrupt. These are actually components of a header file which is automatically included in Arduino IDE projects without the user having to specify their inclusion. ProjectIO comes with this header file automatically installed, allowing users to specify whether or not they wish to include it as arduino.h. This allows us to get the best ease-of-use functions from Arduino without having to struggle with the simplicity of their IDE.

PlatformIO is also useful during debugging and testing – allowing for the user to immediately upload to and then monitor connected devices. Monitoring is typically used to read the serial connection in-window of the compiled firmware but can also be used to reset the device in the case of catastrophic code errors rendering the device unusable. A standardized platformio.ini file will be made for the teams use which will ensure that each device is not reset upon serial connection unless desired by the individual working on the board. *Figure 5-5* depicts the flow diagram of the MCU firmware operation.

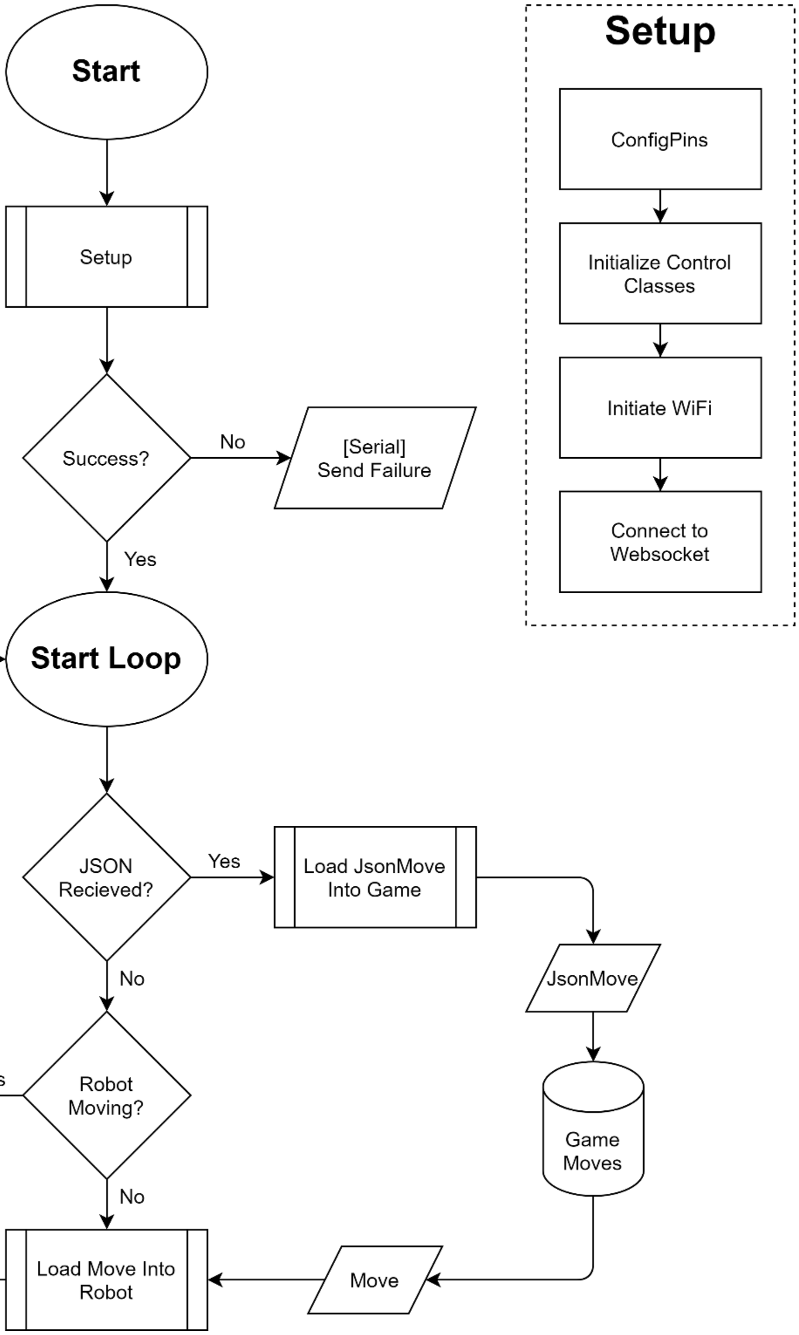


Figure 5-5: Flow Diagram of MCU Firmware Operation

5.3.1.2 Class Breakdown

The firmware code that must be written can be summarized into 5 categories, but in order to avoid having over-specific header files we will instead focus on the 3 functional areas of the MCU. Connectivity, game state functions, and control systems. Therefore 3 primary header files – and accompanying controller classes will be developed using a mix of original code and open-source libraries. The libraries we use are ArduinoJson, ArduinoWebSockets, and TMCStepper. These are all available in PlatformIO, as well as the github repositories cited for them.

WirelessController is responsible for maintaining the wireless connection, as well as processing incoming data into a C++ friendly format to be used by the program later. 2 open-source libraries were used in order to streamline the design process: ArduinoWebsockets and ArduinoJson. The former allows for the creation of a websocket client which triggers actions on received messages. The latter offers a robust method of deserializing JSON files into C++ objects with very quick access, static or dynamic memory allocation, and minimal impact on performance.

GameController handles game-state and piece data which allows for path planning. Queues of moves are build inside of functions which take positions and pieces as arguments – allowing for unique movement conditions to be dealt with. Most notably, knights and the retirement of captured pieces posed a risk of collisions on the board which needed to be dealt with efficiently and without much computational overhead. The resulting position of the robot head is also tracked – allowing for quick dX/dY calculations based on the future positions of the system.

RobotController acts as the interface between the hardware and software layers in the form of signal control, motor stepping, and queue logic. Due to a recently presenting bug regarding the ESP32s hardware serial ports, the UART functionality is disabled. However, once the error is resolved by either Espressif or a suitable workaround is found the RobotController contains motor configuration functions which are confirmed to work. These same functions also allow for the StallGuard™ features of the TMC2209's to be used for sensorless homing. A 3rd party open-source library was used in this class as well – TMCStepper. It allows for the previously mentioned configuration and StallGuard™ functions as well as allowing for future TMC supported stepper motor features.

Below, *Figure 5-6*, are the class diagrams for these vital classes:

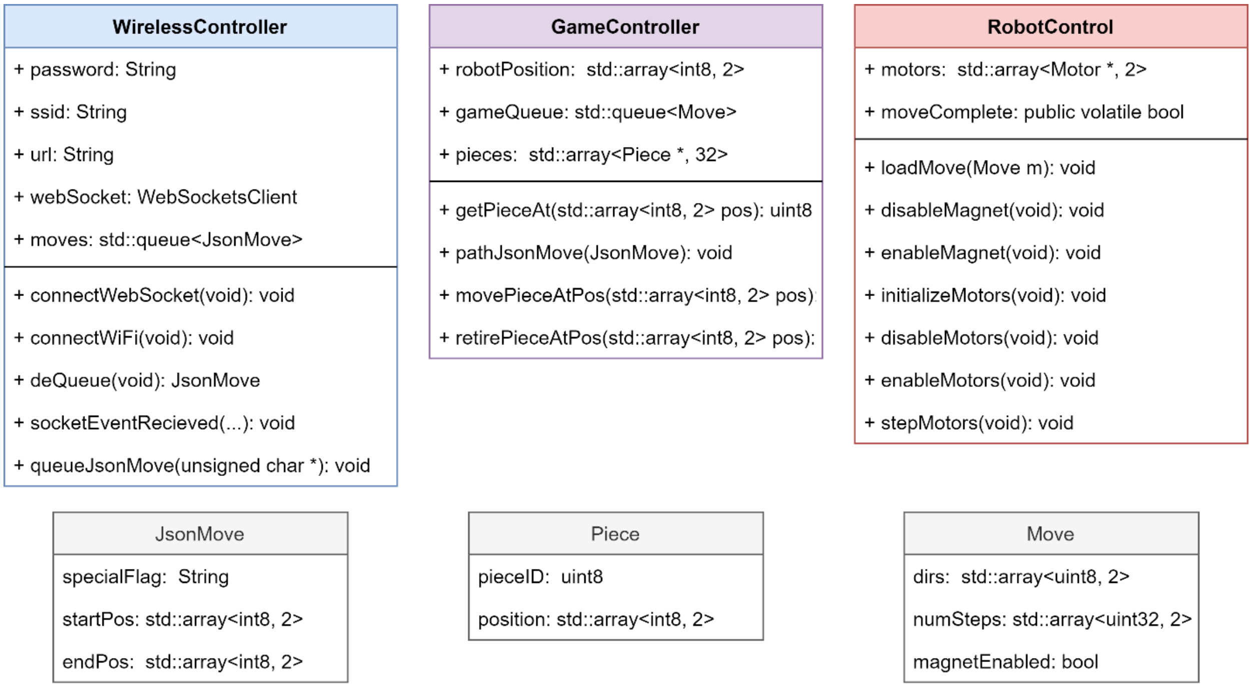

WirelessController
+ password: String
+ ssid: String
+ url: String
+ webSocket: WebSocketsClient
+ moves: std::queue<JsonMove>
+ connectWebSocket(void): void
+ connectWiFi(void): void
+ deQueue(void): JsonMove
+ socketEventRecieved(...): void
+ queueJsonMove(unsigned char *): void
GameController
+ robotPosition: std::array<int8, 2>
+ gameQueue: std::queue<Move>
+ pieces: std::array<Piece *, 32>
+ getPieceAt(std::array<int8, 2> pos): uint8
+ pathJsonMove(JsonMove): void
+ movePieceAtPos(std::array<int8, 2> pos):
+ retirePieceAtPos(std::array<int8, 2> pos):
RobotControl
+ motors: std::array<Motor *, 2>
+ moveComplete: public volatile bool
+ loadMove(Move m): void
+ disableMagnet(void): void
+ enableMagnet(void): void
+ initializeMotors(void): void
+ disableMotors(void): void
+ enableMotors(void): void
+ stepMotors(void): void
JsonMove
specialFlag: String
startPos: std::array<int8, 2>
endPos: std::array<int8, 2>
Piece
pieceID: uint8
position: std::array<int8, 2>
Move
dirs: std::array<uint8, 2>
numSteps: std::array<uint32, 2>
magnetEnabled: bool

Figure 5-6: Class Diagrams

5.3.1.3 Pitfalls and Hang-Ups

It is imperative that the hardware components be understood when developing code. Even during initial installation and tentative prototyping, it became clear that using Marlin as a base was a poor decision – prompting us to make to leap to making our own implementation from a more ground-up approach. Before assessing the code, it was important to read the data sheets for the ESP32 MCU as well as the TMC2209 stepper motor drivers.

Due to the relative simplicity of this project most of the ESP32s pins were available for use, but ensuring that we left enough pins available for UART control of the TMC2209s was critical. With that in mind, using pins 25-33 for push-button interrupts was avoided. The data sheet clearly marks the pins used for analog control, making them easily distinguishable, but the specific pins capable of being used in a UART connection were not as easily spotted.

5.3.2 Off-Board Software Design

This section will discuss the off-board software design which includes both the front-end and back-end of our web application. As part of our initial design, we included prototypes of our software design and explain how we intend to handle the off-board software design for this project. The following sections will include detailed information about the main aspects of the software design and how it relates to the project as a whole.

5.3.2.1 Front-end

In order for users to control the Arcane gameboard remotely, we created a web application that allows the user to play the game with no contact. To achieve this, we designed a full stack web application, using various technologies and open-source libraries. Our full-stack web application is broken up into a front-end and a back-end, with performance and User Experience (UX) being our main priorities. The web application is designed using Responsive design, which offers in a desktop view, with an attempt to provide a mobile view as a stretch goal with time permitting. After completing the project, we were able to design both a mobile and tablet version of the application alongside the desktop version. By utilizing this responsive design, we also create a dynamic webpage that can adapt to various sizes of computer screen, which removes the need for a windowed application while also ensuring the application runs properly, regardless of screen size. The logic behind designing the User Interface is that we begin with the desktop design, and shrink this down to a mobile design, while still offering the same number of features. Normally, we would want to design the mobile application first, and grow the design to include the tablet and desktop from there (mobile-first design), however, our group decided to prioritize the desktop interface for this part of the project.

5.3.2.1.1 Web Application Prototype

In order to design the User Interface (UI), we began by all joining together in a collaborative meeting to design a functional prototype. We utilized Adobe Xd to create the prototype, and using the design tool, we were able to make real time changes to the design so that all of our suggestions could be visualized. We concluded on a clean and modern interface, that consisted of 3 main pages, each providing their own unique functionality. The UI contained a login page where a premade user can log into the application. The reason we opted to include a login is in the case that we decide to expand the application to include multiple levels of users (basic, admin, super admin, etc.), this can be done. Once logging in is done, the user is greeted with a home page where they can select what type of playstyle they would like. Their options are to play either against another player, with one player playing on the physical chessboard, and the other playing on the laptop, or the stretch goal option of playing against an AI.

The front-end of the application is the part of the application that is hosted on the client side (the user’s device). This front-end includes the UI as well as a model to handle the logic locally on the user's device. The reason we chose to include the logic model on the front-end is to boost overall performance, and offload some of the computing power from our microcontroller to the user’s device. The UI was inspired by modern chess applications and was mocked up using Adobe Xd for us to all visualize the game before committing to a design and spending time writing the code.

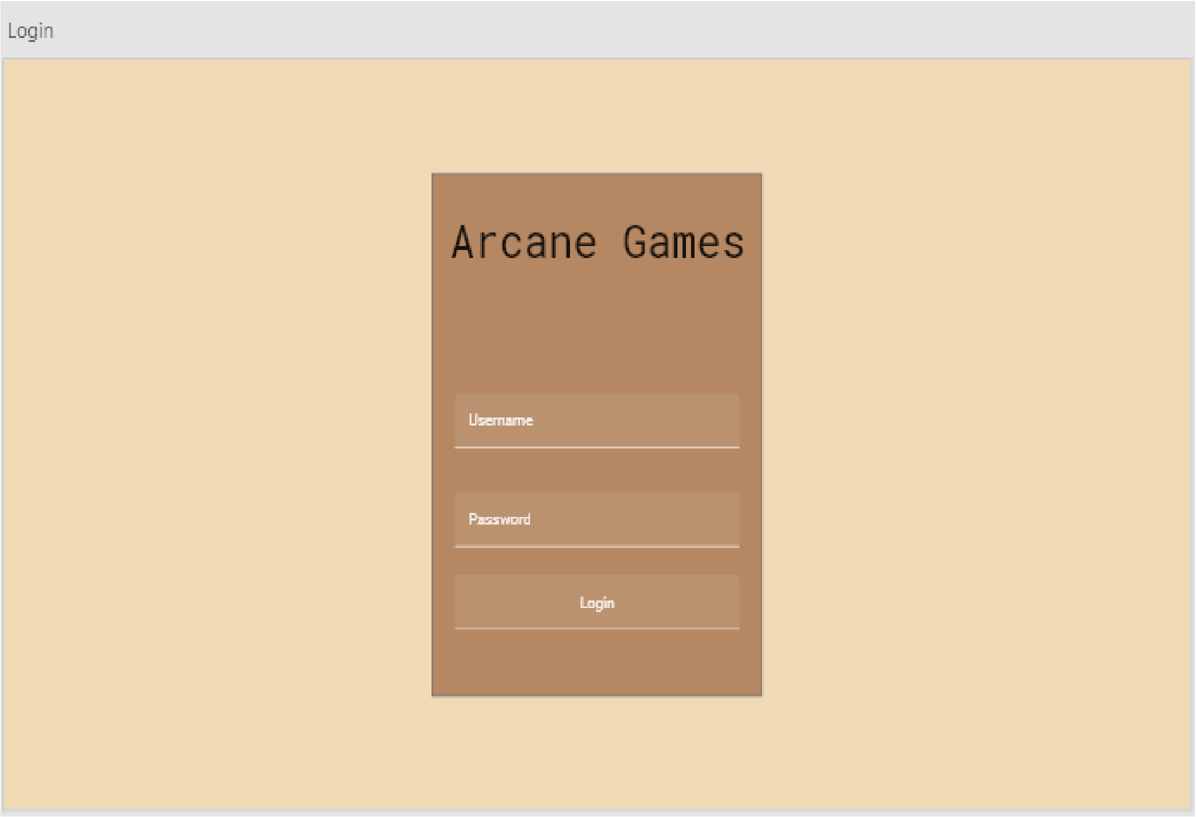
Our prototype utilized various CSS libraries that we wound up using in the real application. Adobe Xd offers various mockup element from real and popular CSS libraries. This allows our prototype to serve as a perfect drawing of what we anticipated our actual application would look like. This is crucial to the front-end developers, as it entirely removes all of the guesswork on how the application should look.

5.3.2.1.2 The Login Page

The login screen utilizes a retro style in order to match the more retro nature of board games. The prototype of the login page (*Figure 5-7*) includes a background that is colored similar to one of the tile on the chess board. The design also consists of a center call-to-action card that is also colored like a the opposite tile of the chessboard. We provide the user with a title, as well as two text fields for both username and password. Using React’s newest hook, useState, we are able to make text fields that update real time, and can check our player’s credentials, before they even hit the login button. Because we are using React-Redux to validate our user's credentials, this speeds up the entire validation process, by completely eliminating the dependency of a network. We create the user credentials on the front-end in order to speed up the development process, and keep to our original goal of only providing one working user. We also allow would the ability to expand the application to include a registration page if needed. In order to achieve this design, we utilized the Material UI, which provides our page with the Roboto Mono font, as well as the familiar, google style text fields.

We handle the login data on the front end and include the standard validations that one would find in a login screen. In order to check the username and password, we have two variables associated with each field. When the user types into each of the data fields, the value of the variable is updated as they type. Once the user has filled in a username and password, they can then click login to validate their credentials. If the credentials that had been inputted matches those of our pre-created user, the application will forward them to the main page. In the event where the credentials do not match up, the user is greeted with a standard denial notice, that appears as a red ring around the text field. This is achieved using Material-UI's text field state, where we can conditionally trigger the denial, based off the value of a state. The use of states is what allow us to have such quick runtime in our application, as it allows us to conditionally render selective components, without having to re-render the entire application.

In order to improve the performance on the login page and speed up the process between the player clicking the login button, and validating the proper credentials, we are utilizing client-side caching on using our software package, React-Redux. React-Redux allows our application to store data on the client-side of things, as opposed to depending on new information to come from the server-side. What this means is, when the application is first loaded onto the player's device, we are already loading the necessary information to conduct our validations on both their username and password. We do this by setting up a local storage unit which will include predetermined credentials that can be referenced anywhere in the application, including from the login page. By doing this, we are our application can quickly compare the submitted credentials for username and password, without having to make a server-side call. This is a very common tactic that is used to majorly improve performance, as you are no longer dependent on internet speed or network calls, but simply the processing power of the user's device. React-Redux also includes several security measures and practices that we utilize in our application. One of these practices is the ability to create predetermined actions that can be used in Redux. This allows multiple developers to work with Redux, and not have to worry about an inconsistent call which could throw an error in the code. By using this practice, we limit the number of actions that can be used in Redux, and provide names to all of the actions that actually have an impact or return some sort of information. Similar to the concept of a real grocery store, the local store now only carries stock of the actions that it has available (and what has been developed to it).



Login

Arcane Games

Figure 5-7: Home Page Design

5.3.2.1.3 The Home Page

The homepage follows a similar style as was initially presented in the login page. The concept of consistent design has been proven to improve the user's overall experience. We can see in *Figure 5-8*, a prototype of the actual web application. This web application prototype makes full use of all of the features included in the Material-UI CSS library, as well some additional custom components in order to render the buttons. We chose the custom buttons in order to keep consistent with the look and feel of the login page. This page includes a centered call-to-action card, which includes two buttons that would allow the player to choose their own custom gaming experience. They have the option to choose from our core goal, of playing against another player, or playing against an computer AI, as mentioned in our stretch goals.

In order to initially arrive to this main page from the login page, we are utilizing a front-end software package known as React-Router. The login page, once given the correct credentials, will update the URL to include a trailing code that can be read by the application. This tactic is a common practice in modern day web development and is known as URL routing. Using the React-Router, the web application is looking for specific trailing notes, and displays the pages associated with those notes, like checking a switch statement. In this case, we utilized the note "/main". When running the application, we can observe the URL change as we click through the pages. This is crucial in times of both usage and debugging, as it allows the user to know which page they are supposed to be on. If the proper trailing notes appear on the end of the URL, but the page does not switch, then it is safe to assume that the error may lie on the backend. Using React-Router also allows users to send specific page links to other users and have them appear on that exact page.

We can often find in older websites, that when a specific link is sent between users, that they are often sent back to the homepage, only to have to navigate back to the desired pages. Another crucial feature that is utilized with React-Router, is that it allows our pages to already come preloaded once the user first enters the web application. Because React-Router acts like a switch statement, our other components that may not be appearing on our specific page, are already loaded, and require a simple “show” command to appear on the screen. This works flawlessly with React’s ability to render components individually, and results in the lighting-fast speed that React applications are well known for.

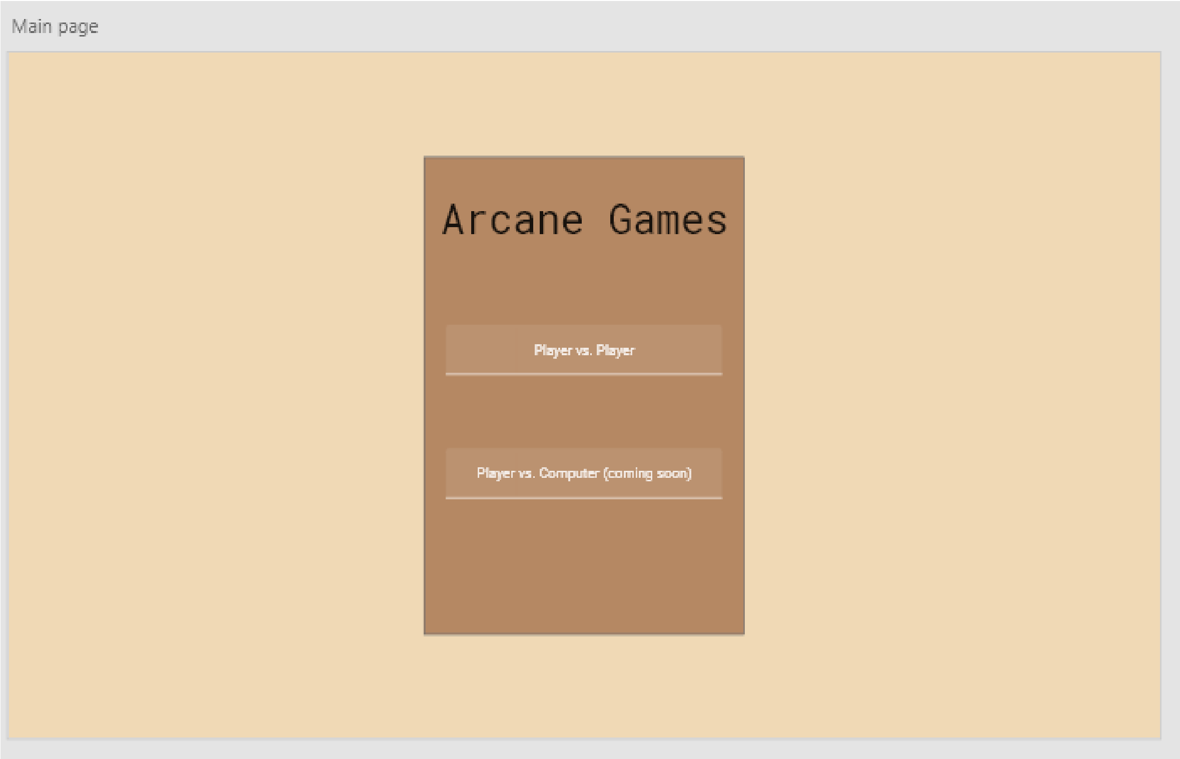


Figure 5-8: Main Page Design

5.3.2.1.4 The Game Page

The game page carries a similar design as the previous two pages, while also offering a simple and easy to use game board for our player. The game page consists of the same chessboard tile color as the background, a functional chessboard in order to play and show moves on, and a subsection on the right of the page that shows a recorded history of the current game. We can reference *Figure 5-9* to see a prototype of all of these features as how we intend to show it on our actual application. Across the top of the page, we find an app bar, with a button that both ends the current game and returns to the main page. Once again, we are utilizing Material-UI for various components, such as the Roboto Mono font, the app bar across the top of the page, and the button found in the app bar. The right subsection for the game history also utilizes some of Material-UI’s CSS, while also requiring some additional custom code in order to create the proper sizing for the section. For the main section of the application, we are using a virtual chessboard, which comes from the software package Chessboard.jsx. This package presents us with a standard chessboard, as well as all of the necessary assets and spirits to play on the board. We are also utilizing the color scheme from Chessboard.jsx throughout our application to provide a very uniform and easy to use UI for our players. While the remaining portion of the application is responsive, both the chessboard and the history section of the application need a minimal screen width of 900px and a minimal screen height of 600px in order to run properly.

Focusing on the mains section of our application, the virtual Chessboard works with an additional software package that we use called Chess.js, in order to create the fully functioning chess game. The combination of the two creates the actual chessboard display using the Chessboard.jsx package, and we pair that with the move set library that comes along with Chess.js. The two are known for working together seamlessly. Chessboard.jsx provides a serious of methods that enable various features within our application. The most basic of these methods is the clear method, that allows us to fully clear the board at hand. We utilize this feature when we click the back button in order to return to the home page and want to clear the current game so that we can start a new one. Another commonly used method is the start method, which is fired once the user initially enters the page. We are also given the ability to trigger a start animation if we choose to. One other key method that we are utilizing in our application is the history() method. This method is tied to the Chess.js software package. When this method is called, it displays an entire history of the current game that is being played. This is crucial for one of our key features of real time player move tracking to work. This is also crucial for a achieving a stretch goal that we have of producing the move history of previous games by selecting the date and time that they were played on originally.

Chessboard.jsx uses all of the moves found in the move sets of Chess.js. Referring to the GitHub for Chess.js (https://github.com/jhlywa/chess.js/blob/master/README.md), we can see that all the moves are related specifically to the moves each piece can make, and not necessarily limiting legal moves that can be done during a game. While this can be very time-consuming to write, Chessboard.jsx fortunately offers a feature where it can evaluate the current state of the board, and limit the moves that players can make to legal ones. In the case that a player makes an illegal move on the physical board, the game will not continue until a legal one is made.

This chess board pairs with the backend's WebSocket in order to receive and display real-time game data coming in from the physical chessboard. We are utilizing WebSockets for the datastream connection. We will go more into detail in the backend section, however, it is worth noting that the way the chess board is being developed, it is open ended and modular in the sense that either end of inputs are open ended. This means that the chess board is designed to be playable from either two laptops, the same laptop, against an AI, or from input coming from the physical chess board. For the purpose of achieving our main goals, we focused on enabling the playability between both the laptop and the physical chess board. With time permitting, we focused on enabling players to play using the physical chess board and the AI which is preloaded onto the laptop.

In order to enable our gameplay while still making use of React's quick component rendering system, we utilize two of React's latest hooks in order to keep performance high. After having a stable release of React hooks in October 2020, hooks have become a pivotal point in both the history of React and front-end development as a whole. In our application, we are going to focus on two main hooks. UseState and UseEffect.

Our game board utilizes various variables that when updated, would be beneficial to render. On top of this, it would be even better if when these variables update, not only if they updated their respective values on the screen, but if they also updated the React components that they relate to. In order to achieve this, we utilize the UseState hook. By tying various data points to states, we can utilize these properties. Some of the variables that we would want to tie to states include board positions, incoming data from the WebSocket, incoming play data from the AI, and many other instances. As one can imagine, if our physical game board sends information to our virtual game of a move update, we would want our application to render this both instantly and seamlessly. Originally, this would require several lines of code, and even in some instances, polling. Using the latest of React's hooks allows us to do this by simply specifying a variable type.

The other React hook that we make serious use out of is the UseEffect hook. Similar to the use case of UseState, there are times in development when we want to trigger a certain action or function call when a related piece of information is updated. Using the UseEffect hook, we can now call specific functions that are tied to variables, states, or other dependencies. In our application, this proves itself necessary when we are working with WebSocket data. Our front-end handles WebSocket data using states, and when those states are updated, we want to kick off a process to handle that data. By tying the states that are being regularly updated as a dependency, our related UseEffect will fire off and make the necessary modifications to our data in order to properly store and use it. Originally, before hooks, this would have required much more complex methods and more development to achieve. Another instance of the use of UseEffects is when we want to update the component that is rendering the current history of the game that is being played. We create a UseEffect with a dependency on the list of moves that have been played in the game and trigger our function that will handle and package this move data and render it onto the screen for the user to reference. Following this, we can then set another UseEffect to fire once the game is complete. That UseEffect can then prepare and package the data in a way that is consumable by the MySQL database, and call an API POST call in order to add that new set of records to the database.

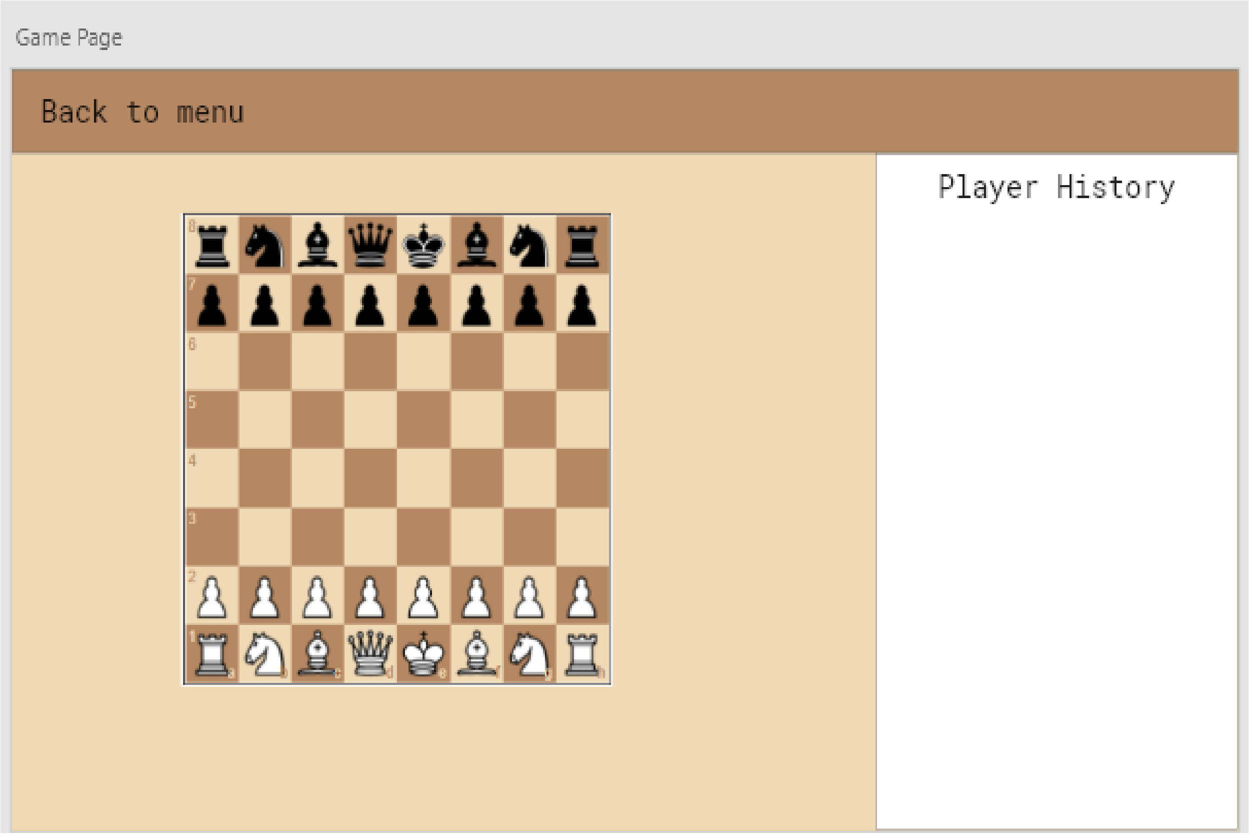


Figure 5-9: Game Page Design

5.3.2.2 Back-end

The back-end of this web application provides a vital role in transferring, storing, and updating data. The back-end is also the primary driver in not only enabling data rich features to the front end of our web application, but also by establishing a line of communication between our web application and our physical chess board. The various components of the backend include a WebSocket connection , Node.js API, a move-to-Gcode converter, and a MySQL database.

5.3.2.2.1 WebSocket

Our WebSocket connection plays a vital role in streaming data between our physical chess board and our virtual chess board on the web application. With the capability to connect to both a physical microcontroller and to our web application, the WebSocket connection provides a bi-directional data path between the two entities. This is crucial for the development of our project as we rely on this two way communication to keep a speedy and seamless experience for our player. Without the use of Websockets, we would have to conduct a timed polling operation on both ends which would be very costly in terms of energy and processing power.

While the WebSocket is both bi-directional and vital to both the on-board and off-board software, in this section we will discuss the design and implementation on the off-board software. By integrating the WebSocket with our React app, develop the ability to receive updates from an external server without having to make an API call. This can be easily visualized as more of a telephone conversation as opposed to an API service which is more like an email service. We can see this clearly in the state diagram, *Figure 5-10*. In order to create the full-duplex channel over a single TCP connection, we need React, Node.js, and Express,js present. We start off by installing the package to our project directory using Yarn. We must then import WebSockets and add it to a local constant. After this we can specify the port in which we want both the API and the WebSocket to run on. After establishing a route (like an API endpoint), we can then enable that line of communication to the front-end components. Switching gears to the front-end, we must then open up the address of that WebSocket connection and establish which route the information is coming in from. Once the front-end is listening to our established route, we need to organize a manner on how to store that data locally for use. We mentioned in earlier sections how we handle the incoming WebSocket data using local React states. Once this is complete, the WebSocket is up and running.

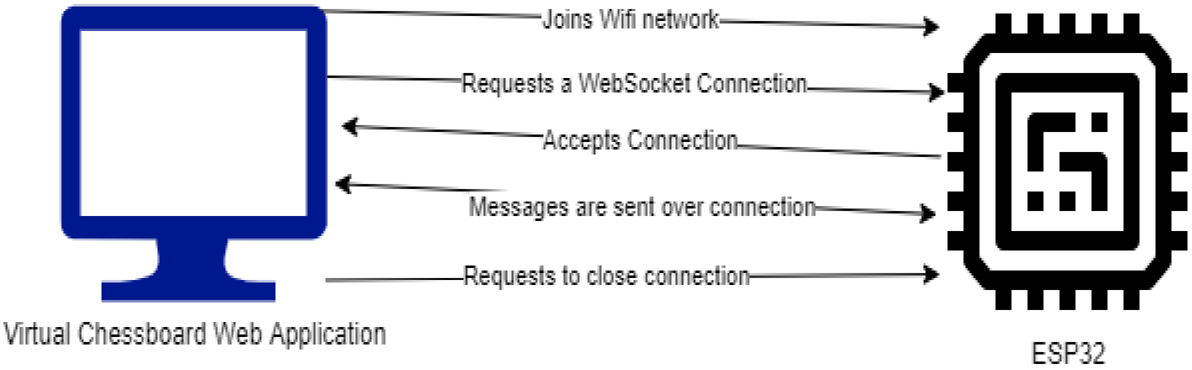

Joins Wifi network
Requests a WebSocket Connection
Accepts Connection
Messages are sent over connection
Requests to close connection
Virtual Chessboard Web Application
ESP32

Figure 5-10: Websocket Lifecycle

As mentioned before, the WebSocket is bi-directional, and the client-side (web application) can also send messages to directly to the game board. On the front-end, we can now use a method WebSocket.Send(), in order to send back data for our physical chess board to use. In order for the data to be digestible by the physical chess board, we developed an encoder that reads in chess piece moves, and translates them into the GCODE necessary for the board to perform the desired action. This is discussed further in a later section, however it is worth noting that the same GCODE is what is sent back by the ESP32, and must also be translated into digestible chess moves, as part of the data preparing process.

5.3.2.2.2 Node.js API

Node.js when paired with an Express.js server runs as a high-performance server and API for our web application. With its ease of use, similar JavaScript language, and proven performance during countless benchmark tests, it was a prime choice for the API and server. For this application, we also focused on expandability, which utilizing a Node.js API would aid.

We use our Node.js API to establish a connection between our MySQL database and our React front-end. There are several bits of data that must be stored from the front-end, and we utilize the API as a bridge for that. The API’s main purpose is to establish a standard on how data is sent between the front-end and the back-end. In our specific case, we are focusing on the example of storing chess games historical data and the move sets that were played during those games. We began by downloading and installing Node.js and Express.js to our web application directory using Yarn. After installing these packages, we went ahead and created their own JavaScript files. A regular naming protocol is used, so we call these files server.js, and create an additional directory to store our database commands and our established routes. In doing this, we go ahead and import both packages. Once both packages have been imported, we store the values brought from these packages locally to our server file. Next, we establish a port on which to communicate on. By default, the port that the server runs on is 3000. This is the same port that we had originally established for the WebSocket. Since we developed locally, we must take care to disable and CORS interference, as this will block the connection in an attempt to improve security. We can work around this since we are using our own, trusted machines.

Once all of the boiler plate code was written for our server.js files, we switched our attention to the database commands and the routes. The routes were established so the front-end can direct via an address to receive and send certain data points. In the example of the historic feature data, we would have two separate endpoints of focus that the front-end would call. These endpoints would be established in our routes section. In doing this, the front-end knows exactly what to anticipate when calling these endpoints. The open standard file format for a Node.js API is JSON, which essentially stands for JavaScript Object Notation. These endpoints are can also be bi-directional, in that we can have a POST and a GET endpoint. Referring to the *Figure 5-11*, we can see an overview on how the API operates.

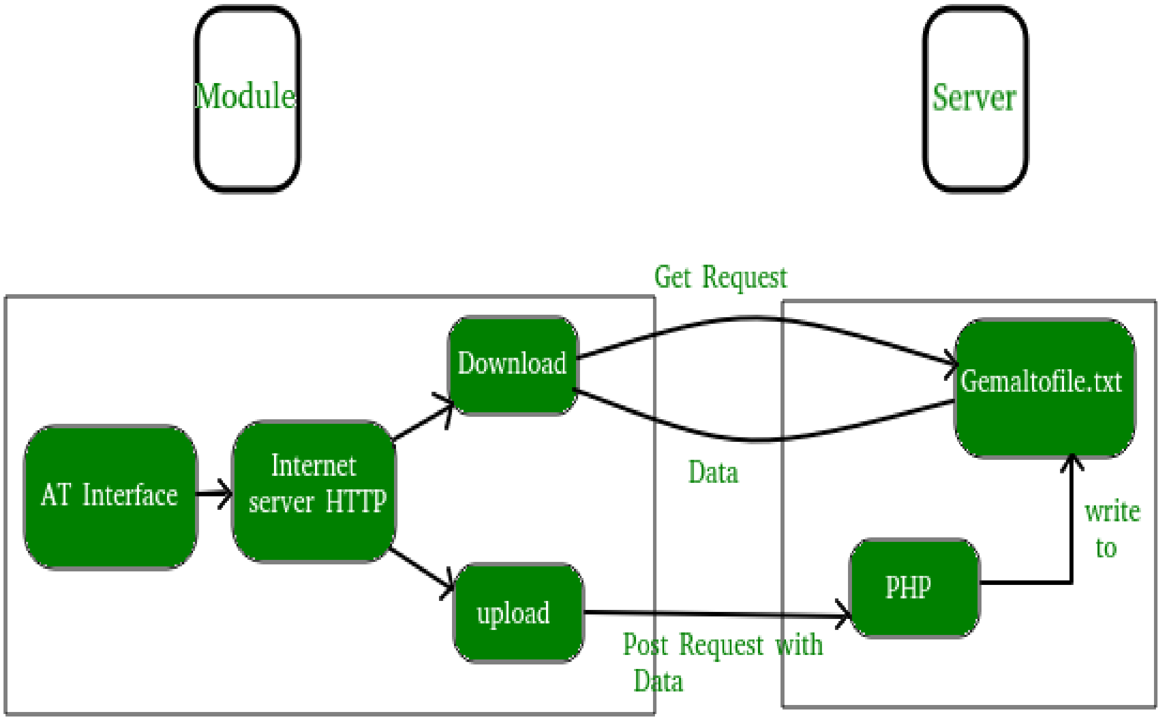


Figure 5-11: API Flowchart

When using a POST endpoint, our front-end sends a desired JSON package through the Node.js API, which then gets interpreted and converted to a SQL INSERT Statement, which is then sent to our database to be implemented. The job for a post call of the API is to parse whatever message was sent as a JSON, and break it down to a level where the database can understand and implement the desired changes. Often times, a post will return some sort of useable data that will indicate that the data was successfully received and interpreted. In the case of our web application, once a chess game is complete, the web application sends a POST Request with several bits of information, including time of game, the set of moves that were played in that game, as well as the type of game that was played.

On the other end of the spectrum, we can use GET calls to extract data from the database. In this circumstance, we provide our front-end with an address, which the front-end can then ping in order ot request certain information. In this circumstance, the address can be open ended, or include some sort of identifying information can that specify the specific type of data that is associated with that ID. The type of route is set up in a similar fashion, where the API must interpret the JSON message sent by the front-end, and call the database with the appropriate select statement.

Request Code	Meaning
200s	Often these codes indacte a success
200	Process went OK
201	Created
202	Accepted
204	No Content was sent
400	Bad Request
401	Unauthorized User
403	Forbidden
404	Not found
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
599	Network Timeout

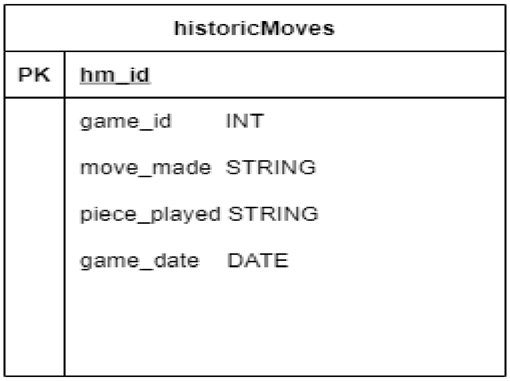
Table 5-3: Request Codes

The final job of the routes for the API is to ensure that the proper request and error messages are being sent in the right circumstances. In the instance where a connection can’t be made, certain permissions haven’t been established, or any other sort of flukes that would prohibit the API from doing its job, the proper error handling must be accounted for so the front-end can read and act accrodingly to it. In relation to errors, the proper success codes must also be sent back so the front-end can have some level of confirmation that the process is working smoothly. Overall, these request codes are vital to proper operation. We can refrence more popular request codes in *Table 5-3*, as well as their associated meanings.

5.3.2.2.3 MySQL Database

The MySQL database serves a key role in delivering various features for our application. MySQL Databases are set up to hold relational sets of data, and can be used to store and provide key elements of data that are necessary for our application to run. Some of this data can include login credentials, such as usernames and passwords, as well as their related status to the application (user, admin, superadmin, etc.). One of the main features that utilizes a MySQL Database is the historic games feature. As one of our additional goals, we are striving to provide users with the ability to virtually recall moves that have been recently played in the current game. Another feature that will relate to the historic data is the ability to store previous games data and show it on demand based on a set of search criteria.

The true strengths of a MySQL database show in these circumstances, where we can create a designated field or column to relate data to a specific search point. In our case, in order to search for specific games and their related move sets, we can pass the database a specific date or game ID that ties those individual moves to a specific game, and return the related data associated only with that game. When using a relational database, we provide each individual row of data in specific tables a primary key. This primary key is used to distinguish that particular row from others within the same table. This also serves as a safeguard in the instance that we have two duplicate rows of data that need to be particularly distinguished. This applies to our application in the case that two games are played simultaneously with the same move sets. If we were to implement an additional table to keep track of which users were playing in those particular games, we would then have to implement foreign keys, which serve to relate one tables data to another. In the case of our specific search column, while our gameID field isn't necessarily a primary key for the historicMoves table, it still provides a unique identifier for us to call related games on. This can be done by proving a list of current game Ids on the front end, and triggering an API call that provides our database with a query with the game Id as a constraint. Once the database has found the related rows for that game Id, it then returns the list of related rows associated with that Id. This is then returned to the front-end via the API endpoint as a JSON package of the found data, and can then be used by the front-end.



historicMoves	
PK	hm_id
	game_id INT move_made STRING piece_played STRING game_date DATE

Figure 5-12: State Diagram for historicMoves

For our historicMoves table, we start by storing the unique primary key to ensure that each row is unique. We then store the game_id in order to distinguish which game this move, which is stored as a row in our table, was made. We also store the piece that was played to provide additional information for our front-end display. We then store the game data in order to allow expendability in case we decide to allow users to search all related games that were played on specific date. We can reference the state diagram (*Figure 5-12*) to get a better visual on how the data is stored.

5.4 SUMMARY OF DESIGN AND OVERALL SCHEMATICS

Our design consists of three main components, the mechanical movement, the electrical components and the software/firmware. All three of these sections have substantial overlap in our project, therefore it is critical that we design each portion such that it will work with everything else. Our team worked diligently to arrive at the most efficient design for our initial project idea and made sure each of our goals able to be met realistically.

The main components of the electrical design are most centered around the printed circuit board because the PCB is supplying power to all of our electrical parts. Using a 12V power adapter wall plug to supply our input voltage, we used a combination of voltage regulators to ensure each component receives the proper operating voltage and current. The components that we are supplying power to are the ESP32 microcontroller, the 5V electromagnet, and the stepper motor drivers. This ensures that we are able to deliver a working and efficient prototype.

The firmware is working largely as an independent system from the off-board controller app. Receiving GCODE commands via an HTTP server and executing them. The on-board command interpreter is configured similarly to the Marlin GCODE system, with several bespoke commands of our own design. In order to bypass the hardware limitations of the ESP32 MCU, the Bluetooth and Wi-Fi functionality can and is toggled during runtime. The main loop consists of polling the HTTP server for commands and waiting for any button interrupts which may occur. Code derived from Marlin is used to drive the TMC2209 stepper motors, allowing for both movement as well as homing functionality without using limit switches.

The mechanics of the Arcane game board revolves around our custom implementation of an H-Bot gantry system. We are combining parts that we designed using SolidWorks with off the shelf metallic components. The custom parts are printed on 3D printers owned by team members and assembled around off the shelf components to fit our custom H-Bot implementation within a wooden playing board. The final assembled Game board utilizes GT2 timing belts and pullies within a game board that is sized within our dimensioned requirements outlined in our house of quality.

The off-board controller app provides a responsive and intuitive web application that enables users to play the game of chess online while watching the real-life pieces actively mimic the moves seen on screen. This is achieved by pairing a React front-end with a Node.js and MySQL backend. This common web stack provides a simple and clean data stream throughout our entire application. The web application is then connected through WebSocket connection using WebSockets to the on-board ESP32. This bi-directional data stream is the sole bridge in communication between our web application and our physical board. The design consists of a single data bridge to keep a simple and easy to maintain design, while also allowing our team to work in parallel, without considerable overlap nor dependencies.

6.0 PROJECT PROTOTYPE CONSTRUCTION

At the time of writing the team has ordered the bulk of the components required to complete the cartesian robot. By wiring a development kit into a breadboard and providing a power supply capable of powering the motors we will be able to begin testing. Due to the lead times involved with ordering PCBs it is not possible to integrate it into the physical assembly yet, but design has begun which has been detailed in section 5.2.3. This construction section will cover how the PCB is assembled once the board is delivered.

By having dev kits of our final electronics on hand such as the ESP32 and the TMC2209, we can write firmware for the ESP to actuate stepper motors while the rest of the Gameboard and PCB are being designed. We can then test that firmware on the dev kits and port over functioning code to the final PCB and into the game board when those parts are completed. This lets us develop in parallel as some programming is being done on a prototype while the final product is being designed and assembled.

Many of the components we use – such as the carriage assemblies – are 3D printed out of PLA and PLA+. This has led to us establishing several in-team standards which are intended to ensure reliable operation and durability. During the construction phase these standards were tested and their effectivity commented on – especially the parts which we are able to rapidly fabricate and put through some physical stress.

6.1 Physical Game Board

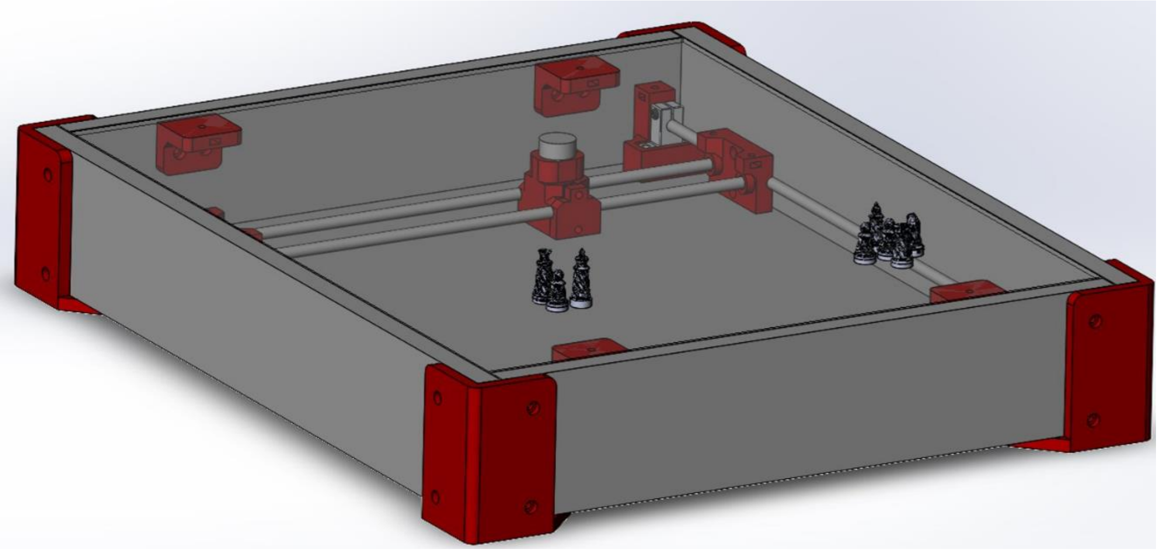


Figure 6-1: Arcane Game Board SolidWorks Design

Above is an image, *Figure 6-1*, showing the internals of our most final Arcane Game Board design rendered in SolidWorks. We have strayed away from using 80/20 design showed in *Figures 3-19* and *3-20*. This is because the linear rail and linear bearing design is cheaper, more light weight, and will hold up to general wear and tear of the running system better than the v slotted wheels rolling in v slotted 80/20.

The bottom grey rectangle represents three quarters of an inch thick MDF board that was cut to size and used as the bottom most piece of our Arcane Game Board. MDF is sold at local hardware stores and can be cut to size on purchase as it has roughly the same density as softwoods such as pine. We used M5 threaded screw inserts to mount our metallic pieces to the bottom board and mount our side panels to the bottom board as well. These inserts are made of metal and only need to be threaded into the MDF once. They have appropriately sized threaded M5 holes to accept variable length M5 bolts. These allow us to screw and unscrew our M5 bolts into our MDF boards and metallic pieces without weakening our MDF material every time a component needs to be removed or added. This is because we have a metal bolt on metal threads rather than a metal bolt on threads carved out of the MDF material. If we were to drill a screw directly into the MDF, we would have to limit the number of times we roved that screw as the metal threads could strip the created groves in the MDF material.

For our H-Bot system to function properly, precise alignment is extremely important. We are using metal clamping rod rail guides to hold on to our linear rails. These guides are sitting on spacers with through holes to allow for M5 bolts to be passed through. These M5 bolts then interface with the metal threaded screw inserts discussed in the last paragraph which are lodged into the bottom base board. At one point our team was considering using wood screws instead of these inserts. This is because wood screws can be drilled into our bottom base board after a fully assembled H-Bot was sitting exactly where we want it. We feared that by using metal inserts, we would run the risk of having misaligned metal clamping rod rail guides which would mess with the operation of our H-Bot. We thought this would have been because we would first need to insert our threaded metal inserts, and then place our metal clamping rod rail guides. This could leave room for error as the insert may walk away from the intended drill spot while its being inserted or our markings on the wood may be off (even after measuring twice, it has happened). However, we discovered that these threaded metal inserts can be screwed into the base board by driving a bolt that is already threaded into the insert. This would allow us to place a bolt through our built H-Bot and then drive the threaded metal insert into the baseboard. This method of threaded metal insert attachment would greatly eliminate error and would give us all the benefits discussed in the previous paragraph.

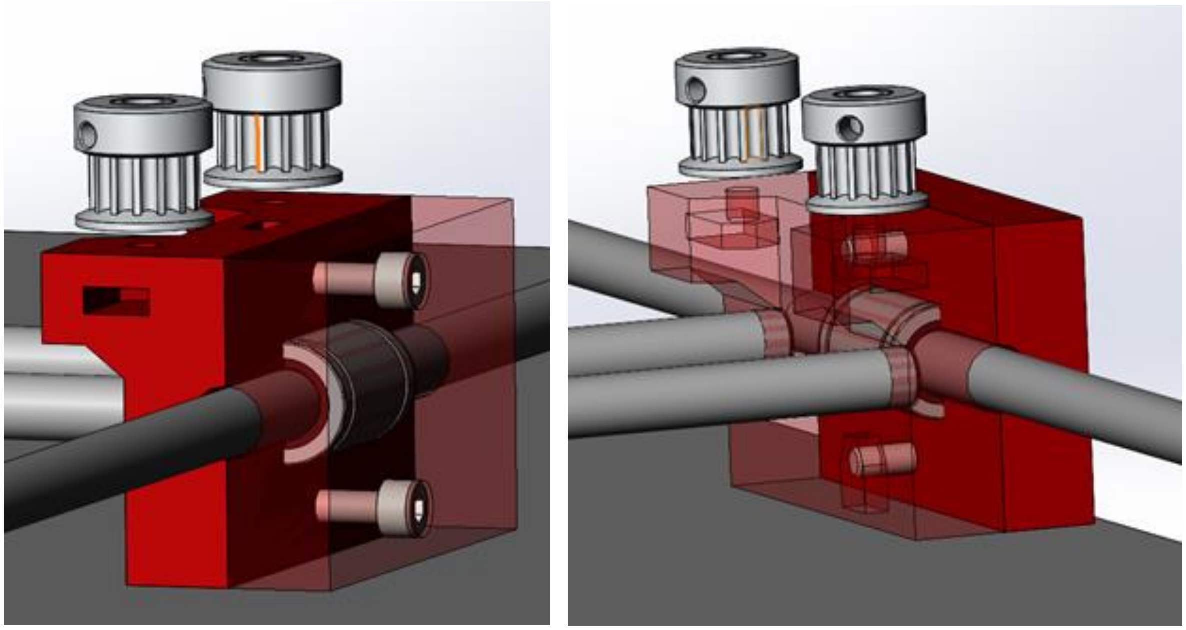


Figure 6-2: Side Linear Carriages SolidWorks Design

The critical components that make our off the shelf components work in unison are our linear rail carriages. Above, *Figure 6-2*, is an image of one such linear rail carriage where the left piece is ghosted in one image and the right is ghosted in another. These make the models semitransparent allow you to see the cavities within the 2 pieces and get an idea of how our off the shelf components are fastened to them. These had to be custom made and 3D printed to get the structural and active components to work with each other. Each carriage had to securely house at least one linear bearing to allow it to slide across the linear guide rails. These linear bearings are secured to the carriage by being forced into friction fit cavities in both of the two side pieces of the linear carriages. Once the bearing is inserted to one piece of the carriage, the second is attached by sliding M5 nuts into the cavities made to house them and passing 2 M5 bolts through the opposite piece and threaded into the receiving nut.

Once both pieces are secured and enveloping a linear bearing, additional M5 nuts are slotted into cut outs to receive the M5 bolts that fasten our pully idlers. Since all the pullies other than the ones on the motors are not active, each pully on our carriages must be free spinning. This free spinning motion is accomplished by the embedded bearing with a bore diameter slightly larger than 5mm within each idler pully. Unlike in the pictures above, the final idlers on the side linear rail carriages are toothless. This is because the part of the belt that is interfacing with the pully idlers at this stage of the H-Bot belt loop is the belt's backside (non toothed side). We feared that having the backs of our belt exposed to toothed pully idlers could result in choppy motion or in a worst case scenario eat at out belt reducing its durability and lifetime.

By now you have probably noticed he frequent mentioning of the M5 screw standard whenever we discuss attaching components together. This is because we were attempting to use the M5 screw standard throughout all the attachment points of this project. The off the shelf metal clamping rod rail guides must be fastened using an M5 bolt, the Nema 17 motor shaft diameter is 5mm which means our pullies must have a 5mm bore, and the fasting of linear carriage components was our chose. We decided to make everything fasten with an M5 bolt so that we only need a single Allen key for assembly and disassembly. We also did not have to worry about mismatching Nut sizes for different bolts since all bolts are the same diameter. The majority of these metallic components were manufactured overseas where the common system of measurement is the metric system. This was a major influencing factor that coincidentally made our off the shelf components all use M5 fasteners.

With our side linear carriages fully assembled the next step would be to insert our 8mm linear guide rods into our embedded linear bearings. The side linear carriages have a through hole of 9mm in diameter to allow the 8mm diameter guide rods to fit through them such that they only interface with the embedded linear bearing. Once the linear guide rods are through each linear carriage, a metal clamping rod rail guide is attached to either extreme of the rod. This is done with the clamping bolt in the metal clamping rod rail guide disengaged to allow the rod to freely rotate within. Both metal clamping rod rail guides were placed on a flat surface before tightening to prevent any misalignment between their surfaces that interface with the bottom board they are mounted to.

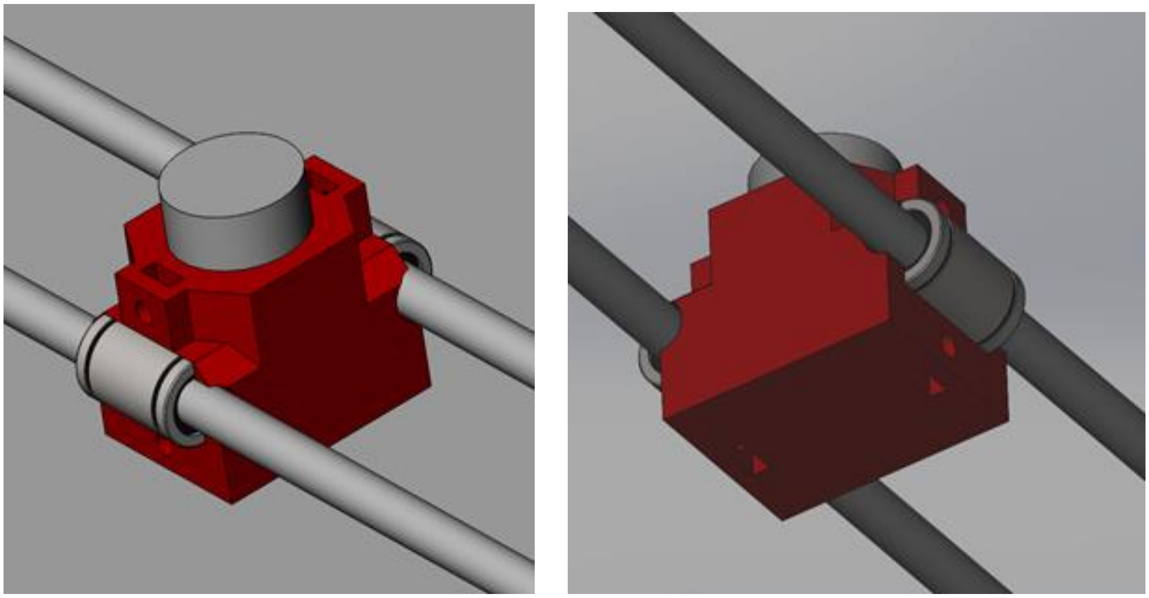


Figure 6-3: Central Linear Carriage SolidWorks Design

The images above, *Figure 6-3*, show a model of the central linear carriage that houses our electromagnet. Intricate groves have been cut into it to allow the carriage to move as close the to linear guide rod extremes which in turn increases its range of motion slightly. Two rods are required to keep this linear carriage from rotating which helps it remain upright. The two rods interface with divots in the side linear carriages shown in the previous 2 images. Side plates are made to interface with the slotted M5 nut cavities shown to securely attach the linear carriage to the linear bearings. One of these side plates also locks on to either end of the single belt that runs in our H-Bot loop, effectively closing the loop. The other side plate has to maintain a low profile to allow the other side of the belt loop to pass uninterrupted by the electromagnet. There also needs to be a through hole added to the center of this linear carriage to pass the wires that power the electromagnet through the bottom. This allows the wires to be dragged between the base board and all moving components which eliminates the possibility of it getting snagged.

Once the center electromagnet holding liner carriage is fully assembled with both side plates attached, two 8mm guide rods are passed through the embedded linear bearings. This center linear carriage once again has 9mm diameter through holes to allow only the linear bearings to engage with the linear guide rods. With the carriage now riding on the rods, the two assembled side pieces consisting of a side leaner carriage, and 8mm guide rod, and both metal clamping rod rail guides are attached at either end.

Spacers needed to be made to increase the distance between the bottoms of the metal clamping rod rail guides and the bottom most base board. This allowed us to gain more clearance between all our moving parts and the bottom base board which allows us more room to fit components inside. The spacers also serve two more purposes. For the spacers on the metal clamping rod rail guides closest to the stepper motor, they double as a stepper motor mount. These keep our motors in place while placing their axles at the appropriate height to run our belt through the necessary pullies to have a working H-Bot system. The spacers opposite the stepper motors also house the toothed pully idlers that complete the belt loop of our H-Bot. These are the only pully idlers that need to be toothed as they are the only ones that interface with the toothed side of the belt.

Once these spacers were fabricated and fit checked, they were placed under the metal clamping rod rail guides which are now holding all the carriages and rods of our H-Bot. With the spacers in place above our base board, A bolt was fully inserted into our metal M5 threaded wood inserts and driven by a drill. We drilled the inserts in place while the M5 bolt is slotted into the through holes of our metal clamping rod rail guides and spacers. As mentioned earlier, this eliminates any possible alignment errors. With everything now fixed to the bottom baseboard, the stepper motors and pully idlers are attached, and the belt is run through all the system's pullies.

With the H-Bot structure fully assembled, the walls can be attached to our baseboard. This could be done with more threaded metal M5 wood inserts and some L-angle metal brackets. The wall material is the same MDF that was purchased but cut at appropriate lengths and widths upon purchase to enclose our game board internals. By this point there was plenty of space between our stepper motors to attach our fabricated PCB as well as our power supply. A cutout needed to be made in one of our walls to accommodate the connection to a PC power cable socket that connects to our internal power supply.

The top piece is a 1/16-inch-thick piece of clear acrylic. This acrylic rests on angled L brackets attached to the side walls or a grove cut out of the walls so that it can sit securely in. We had the acrylic engraved using a laser engraver to produce a checkered board pattern of frosted and clear squares to represent white and black squares. Each square is 1.5 inches long and wide.

Component fabrication was done through several of the team's available 3D printers. All the linear carriages were grown using PLA plastic. Drilling into surfaces was done with the power tools owned by our team members. The MDF housing was measured and cut to size on purchase according to the dimensions shown within the SolidWorks models. The only fabrication that can't be done in house is the laser engraving on our clear acrylic top surface sheet as none of our team members own a laser engraver. We instead used the cutter/engraver located in the UCF innovation lab within the engineering building on the UCF campus. *Figure 6-4* shows the final resulting assembly.

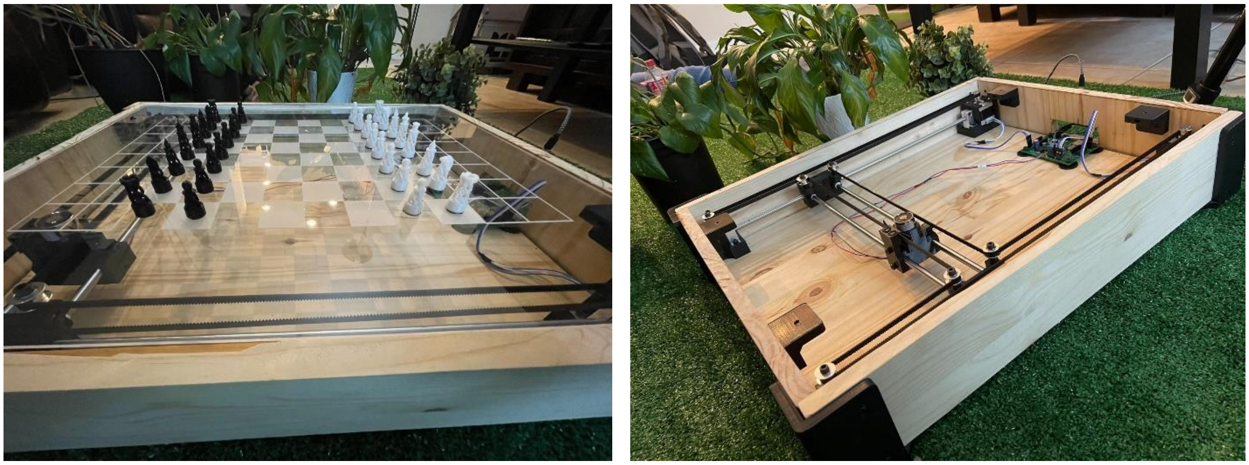


Figure 6-4: Final Board Assembly

6.1.1 Ender-5 3D Printer

In order to complete the cartesian robot we needed to manufacture several components. Given that members of the team already own and are comfortable operating 3d printers we have decided to use them to produce the bespoke pieces which are not subjected to high load forces. As previously states we are using PLA and PLA+ in the construction, which requires a basic printer with the ability to reliably print plastic at 200-220 degrees Celsius. A build area larger than 175mmx175mm would also allow for prints containing multiple pieces on the same bed, speeding up the process with a reliable printer.

A modified Creality Ender-5 printer suits all of the needs stated above – printing more quickly and reliably than a base model is expected to by using several advanced features. Auto-bed leveling has been installed with a BLtouch probe fixed to the print head. This ensures that collisions and proper first-layer adhesion are greatly improved, even without manual bed leveling between each print. Collisions are particularly important when dealing with multiple pieces on the same print bed; they can knock them from the build plate and fail the print.

The mainboard has also been replaced with an BigTreeTech SKR Mini E3 V2.0. This ups the processor to a 32-bit system from the original 8-bit, and most importantly comes with 512kB of memory for the firmware. In the base model there is only 15kB of usable memory which severely limited to capabilities of the printer. TMC2209 stepper motor drivers – the same which we are using for the cartesian robot MCU – are also pre-installed on the board. While the StealthChop feature is a welcome addition for the space the printer operates in, the sensorless homing and greatly improved control consistency are the priority.

In addition to the electronics overhaul, several mechanical components have been replaced and/or modified to ensure that the Z-axis in particular is as resilient as possible. This is again to avoid collisions or layer shifts which would fail the print. By using a lead screw with a pitch a ¼ the length of the original model, the Z-axis is significantly more resistant to slipping under the weight of the buildplate and the print on top. It also gains 4x the resolution (1600 steps-per-mm vs the base 455 steps-per-mm) on the Z-axis which allows for the mesh generated by the auto-leveling probe to be more precise than normally allowed. While seemingly a small detail, many base Ender-5 models struggle with slipping in the Z-axis which consistently causes extended prints to fail.

Given that most of the pieces which we require to finish the cartesian robot were already designed, the lead time of part manufacturing was not a major concern for the group. Regardless, it was possible to print all of the parts in less than 2 days easily with this printer. The top speed is roughly 70mm/s with layer lines 0.22mm thick. Fine details can be printed using smaller nozzles and layer sizes if needed, but the top speed must be reduced to 40mm/s in order to compensate for the smaller layer lines.

6.2 PRINTED CIRCUIT BOARD

The printed circuit board is a significant component of our design and there were several things that needed to be considered when designing our circuit board. A PCB electrically connects electronic components using conductive traces etched from sheet layers of copper. Most notably, our printed circuit board is handling the power supply and all of the connections for our electrical components. This is a major aspect of our project, without it our prototype would not function properly; in fact, it wouldn’t function at all.

The following sections will detail the beginning stages of our schematic and board design process as well as the final product. We had three major redesigns to our PCB to fix flaws and accommodate necessary component changes due to part availability on JLCPCB. We had to be strategic with the amount of time we gave ourselves for the order to come in since it can take between 1-2 weeks for the PCB to be delivered which left us with very little time overall.

6.2.1 Schematic Design

In section 3.2.5.5, we explored different PCB design applications that we could potentially use when we begin designing our PCB. Ultimately, we decided to use Autodesk Fusion 360 since it is very similar to EAGLE, which we were already familiar with. Fusion 360 is much more intuitive than EAGLE and has the additional benefit of being able to collaborate as a team on our design. Fusion 360 is also able to use EAGLE footprints which means finding footprints would not be a significant struggle since footprints tend to be available for EAGLE more often than other PCB design softwares.

For our design, we separated our schematic sheets into four main sections: the ESP32 microcontroller and relay, the voltage regulators, the motor driver connectors, and an external USB serial connection for firmware flashing to the ESP32. Fusion 360 allows us to have common connections across sheets that will continue to automatically update across all connected systems. Separating them into different sheets is not necessary but it is much easier to organize our work and make any changes as needed. Additionally, Fusion 360 is able to take these schematics and create a PCB layout from them. This PCB layout is directly connected to the schematic and will update with any changes made to the schematic automatically provided both files are open simultaneously.

In this PCB design software, nets are used to make connections to the component footprints in the schematic. By assigning a unique name to certain connections/nets, we can ensure each component connects appropriately across our multiple schematic sheets. In the figures below, the nets are shown in green while the components are shown in orange. The green dots indicate a junction where two nets connect to each other. In PCB design there are many instances where junctions are used whether it be to connect certain pins to each other or to connect multiple external components to the same pin. The grey text is used simply for labels which helps us organize our schematics and keep better track of our components when designing the PCB layout. Finally, we also included a frame around each schematic drawing; the frame is useful to assist us with keeping track of version changes and shared connections across pages as we continue to finalize our printed circuit board design.

The first schematic we worked on designing was the circuit for the voltage regulators we chose. We discussed in further detail the schematic we needed to incorporate into our PCB in order to ensure our voltage regulators function as intended in section 5.2.4. In addition to this schematic, we also referred to the voltage regulator datasheet to determine the proper pin assignments and connections. As shown in *Figure 6-5*, component U1 is the 5.0V voltage regulator and component U2 is the 3.3V voltage regulator. Both schematics are essentially identical aside from their output voltages. We also included the barrel jack connector which serves as our input voltage as well as some header pin connectors for ground and voltage outputs in order to simplify testing.

The next schematic we worked on the ESP32 and electromagnet schematics shown in *Figure 6-6*. There were many hardware design considerations that had to be researched when designing this schematic and used many datasheets to ensure everything is connected properly. According to the datasheet, it is recommended to connect an RC delay circuit to the EN pin to ensure power supply to the ESP32 during power up. We followed the recommendation to use a 10kΩ resistor and 1uF capacitor for this RC delay circuit. Additionally, we are using the 3.3V output from the voltage regulator as our input voltage for our ESP32 along with two capacitors to prevent possible power rail collapse. The ESP32 is directly connected to the relay for the electromagnet and the TMC2209 motor drivers and is responsible for the operation of these two components. After reviewing the pin layout for the microcontroller, we connected the electromagnet relay to GPIO23 and it sends a low signal to toggle the relay. You can also see in the schematic all the connections between the microcontroller and the two motor drivers.

The schematic we next worked on was the TMC2209 motor drivers shown in *Figure 6-7*. As discussed earlier, due to component shortages we had to instead design this schematic to slot it our devkits. Our hardware design includes two Nema 17 motors which are each connected to one TMC2209-LA motor driver. The connections for these motors are indicated by the M1 and M2 connectors. We intended to run the motor drivers in their UART mode, therefore, we connected the UART pin to the TXD and RXD pins on our ESP32. Unfortunately, due to some changes in the Espressif libraries, we were not able to use the ESP32 UART pins that connected to our second motor driver meaning we could not longer use StealthChop mode nor could we have sensorless homing. We also connected EN, STEP, DIR, DIAG and INDEX to the microcontroller for both of the motor drivers. Since we are slotting in the devkits, we had to make sure that the footprints for these header pins connectors were the same size as the pins on the devkits so they would fit straight into our board. Luckily, the spacing of the pins are the standard 0.1in.

Lastly, we have the external USB serial connection schematic shown in *Figure 6-8*. We used an external breakout board connected to header pins shown in order to flash our ESP32 microcontroller. Once the firmware is flashed, the breakout board can be removed. We needed to connect the TX and RX from the breakout board to the TXD0 and RXD0 on the ESP32. Originally, we had it connected to TXD1 and RXD1 however, we could not successfully flash the firmware using those connections. This schematic also includes buttons for pair and reset which are also connected to the ESP32.

Overall, we have included only the final schematic designs in this document, and we have confirmed that this design works mostly as intended. Given our time constraints for this project, we wanted to develop a first draft of our schematics as soon as possible so we can begin to identify any design problems early on. This came in handy since we did run into several problems and had just enough time to redesign our PCB after the initial testing of our version 2.0 design. The simple organization of the schematic files was extremely useful because it made it easy to keep track of changes made to the schematics.

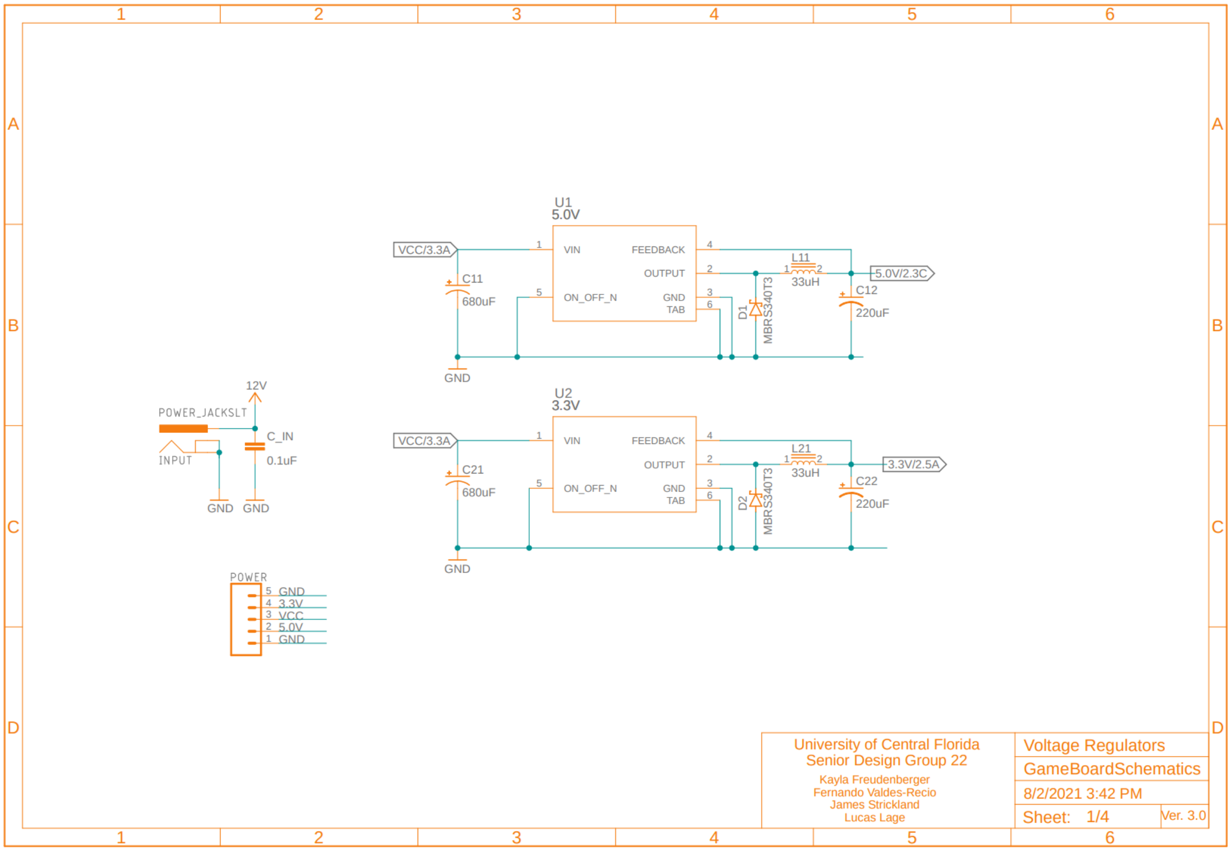


Figure 6-5: Voltage Regulator Final PCB Schematic

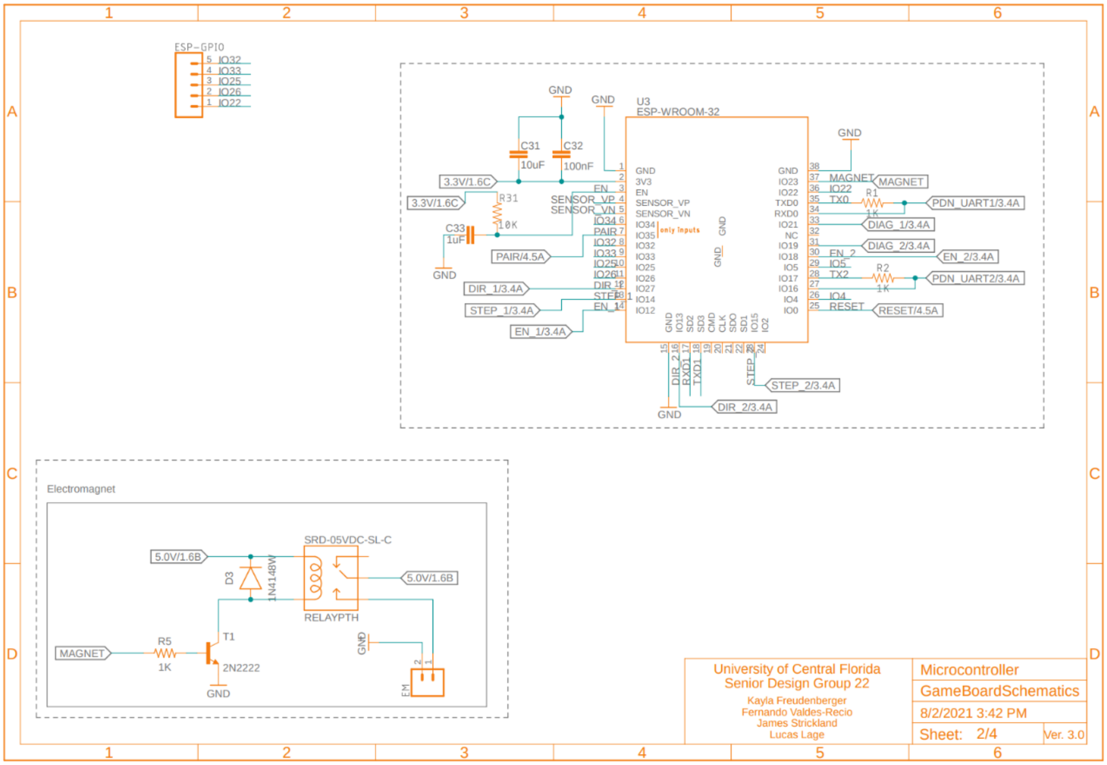


Figure 6-6: Microcontroller and Relay Final PCB Schematic Prototype

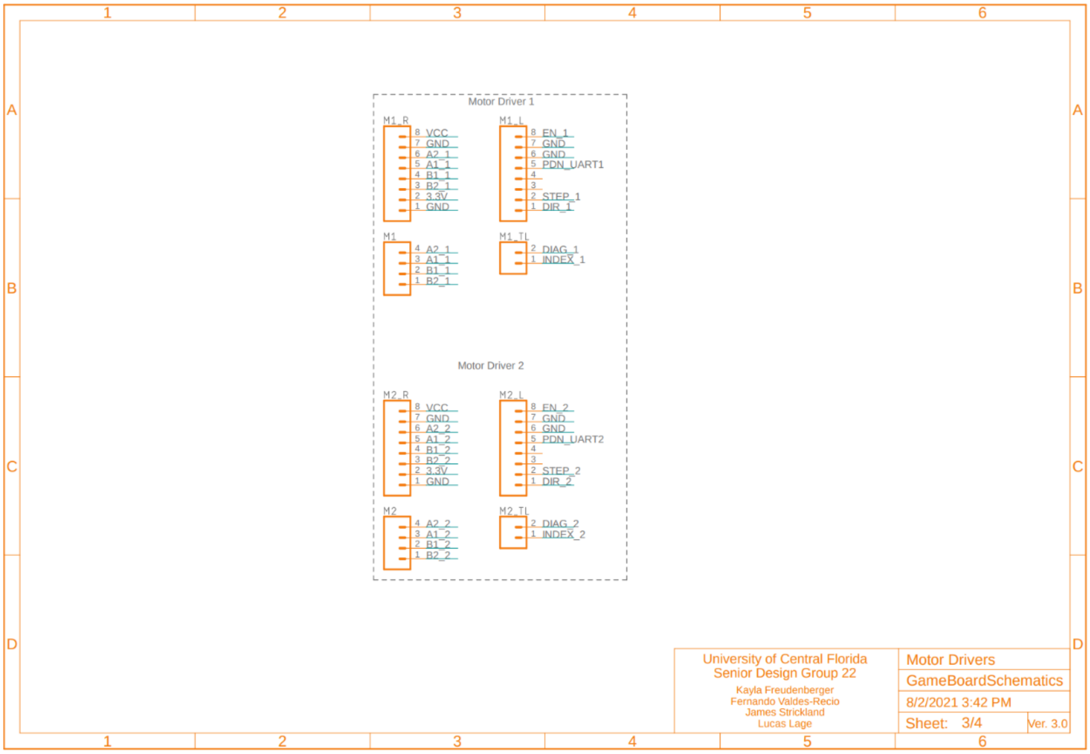


Figure 6-7: Motor Driver Final PCB Schematic Prototype

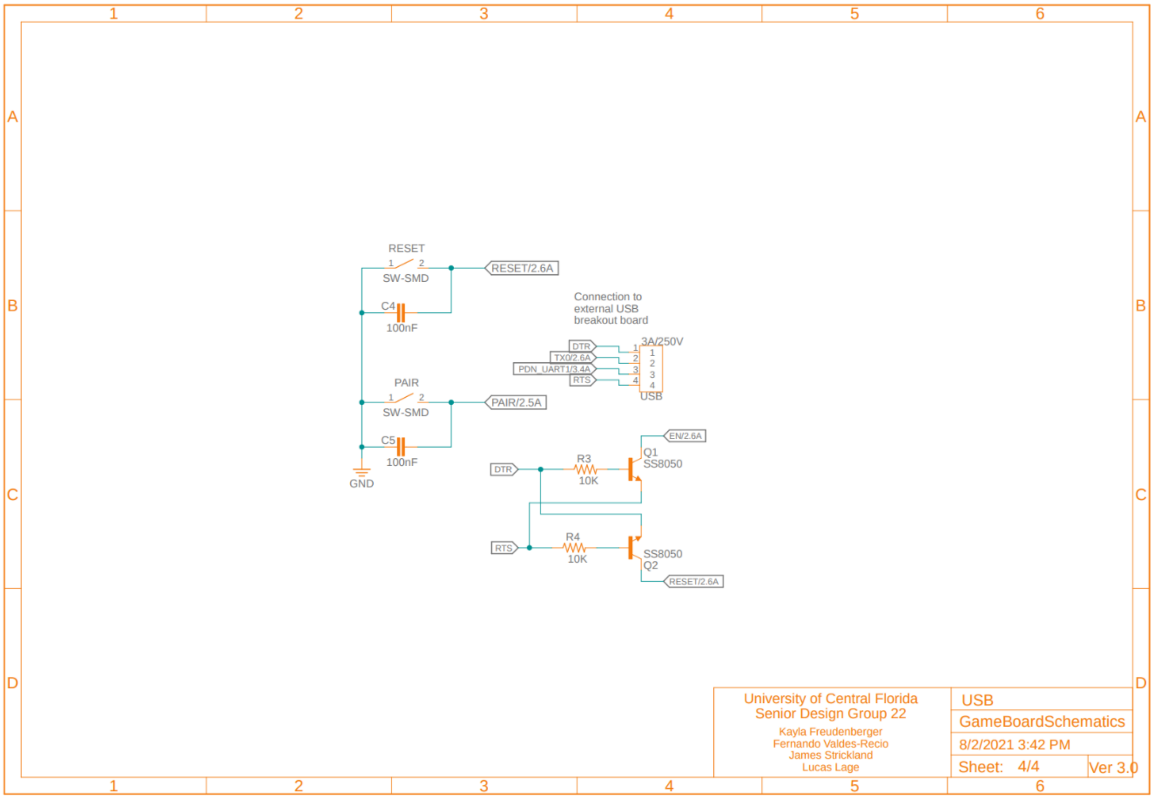


Figure 6-8: External USB Serial Connection Final PCB Schematic

6.2.2 Board Design

When it comes to our actual physical PCB design, we used Fusion 360 to generate a board model from the schematics we created. Each component footprint we used in our schematics has a corresponding model on the physical board and this is used to create an accurate layout. When we generate the board from the schematics, it inserts all our components; however, the layout of the components is not handled automatically. From there, we referenced our schematics to try our best to achieve an optimal layout for our components while also considering which components need to be closer together as recommended by their datasheets. Once we've decided on a layout for our components, we can use the auto-router function to route our traces.

One part of our board design we had to consider is trace width and spacing. Trace width is determined based on how much current you will be pulling through the different copper traces on the board. There are several resources online to determine the trace width necessary for your design parameters, the more current, the wider the trace will need to be. However, you do not want to make your traces too small because that will typically result in higher manufacturing costs. Additionally, we needed to consider the thickness of our board and temperature rise when calculating our trace width requirements. Typically, most PCB signal traces will fall between 6mils and 30mils [24]. Since we are using a 12V 5A AC/DC adapter as our power supply, I based the trace width calculations off of a max current of 5A. Our design does not draw nearly that much current, but we wanted to ensure the traces could still handle that much current. Our PCB is 1.6mm thick; using the max current and thickness, we determined we need trace widths of at least 6.2

mils. To be on the safe side, the PCB design is using 10 mil traces for all traces except for the 12V input which is using a 12 mil trace size.

Part of our PCB design that we had to consider as well is the size constraints. Ideally, we wanted to keep it as small as we could so we could ensure it is not in the way of any mechanical parts, particularly the carriage holding the electromagnet. We also did not want to make the PCB too small since we cannot have our traces getting too closer together. For the design, we wanted to ensure a 10 mil clearance between all pads and traces. Ultimately, given our physical board design, our PCB ended up being 107mm x 76mm which was able to fit into our design without interfering with the electromagnet carriage.

Our design, depicted below (*Figure 6-9*), includes all the components from our schematics in the section above. As mentioned earlier, when Fusion 360 creates a board drawing from the schematics the components are randomly placed, therefore we had to determine the placement of our components. The PCB is a 2-layer PCB where the red traces are the top layer and the blue traces are the bottom layer. Vias are used to run traces underneath the board to prevent traces from crossing one another. Additionally, we followed the recommended placement for our ESP32 microcontroller which, according to the datasheet, should be placed in the top right or bottom right corner with the antenna over the edge of the board. We also made sure to place components with any external connections towards the edge of the board to ensure we can connect these things to the board easily. We also used boxes to outline the main circuits from the schematics as well as label each circuit appropriate. This was a personal design choice made to make testing easier since it would allow us to identify the components without needing to constantly reference the design files.

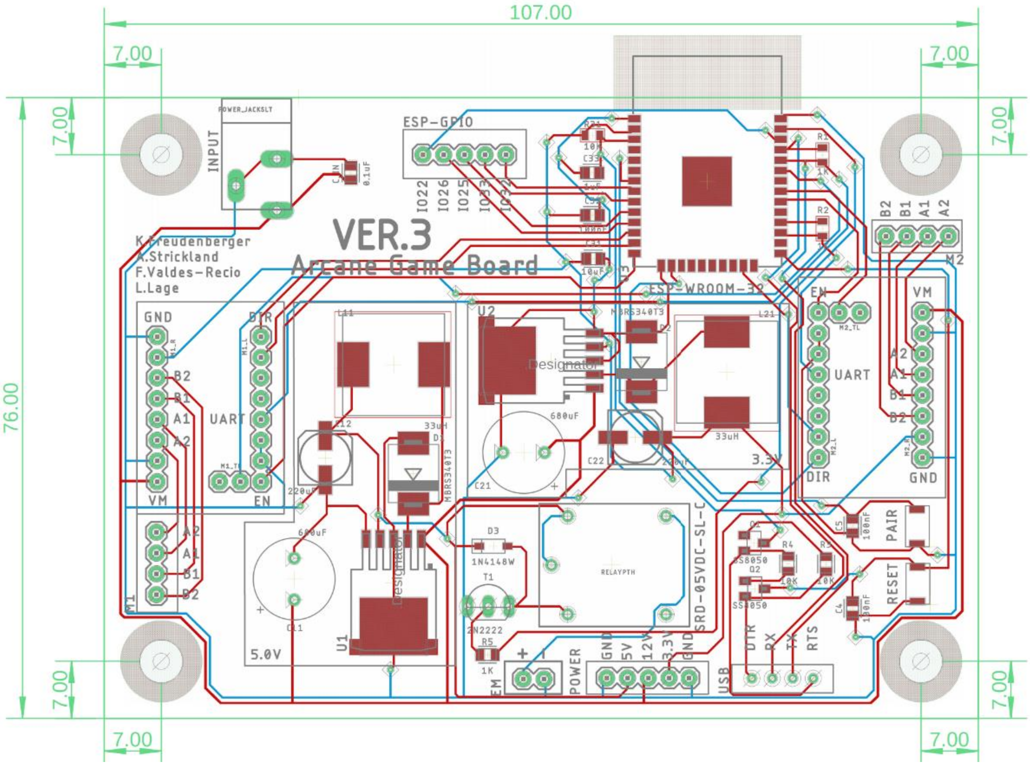


Figure 6-9: PCB Board Prototype

Once we finalized our PCB design, we exported our boards manufacturing files which includes the Gerber and drill files. This file is what we sent to the PCB manufacturer and it contains all information needed to assemble our design. We used JLCPCB to order our printed circuit board for this project. The manufacturer also took care of nearly all the surface mount assembly which meant we needed to also generate a BOM file which included the unique JLC part numbers as well as CPL (*component placement list*) which indicates the precise location and orientation of each component. We did need to solder the header pins, barrel jack and relay ourselves. Once ordered, the engineers at JLCPCB review the design and put it into manufacturing. The whole process from ordering to delivery takes 1-2 weeks and due to this time constraint, we were only able to order two individual board designs with version 3.0 being the final, working version. *Figure 6-10* shows the final PCB minus the header pins, barrel jack, and relay.

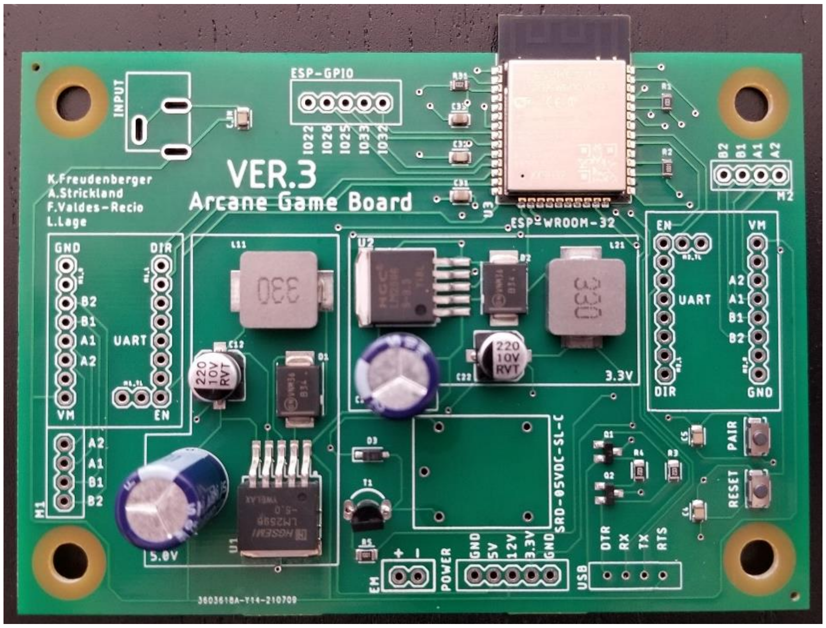


Figure 6-10: Final Ver. 3 PCB

6.3 Firmware Development

As previously mentioned, the team had access to ESP32 development kits before the PCB. Hoping to gain as much ground as quickly as possible, we tested firmware on the dev-kits for most of the semester while PCBs were undergoing development and revisions. Thanks to this parallel processing, the core of the robot movement code, as well as the class structure of the firmware, was completed before the final revision of the PCB was delivered. This allowed the team to focus on refinement and physical tests instead of software unit tests.

6.3.1 Configuring and Testing PlatformIO

PlatformIO was used as a delivery platform throughout the development cycle of the Arcane Game Board. This came with many advantages – including library inclusion, easy access to the Unity Embedded Unit Testing Framework, and rapid deployment options for flashing connected devices. The figure below was made to walk group members through the installation of the program – as well as a corresponding document for internal use.

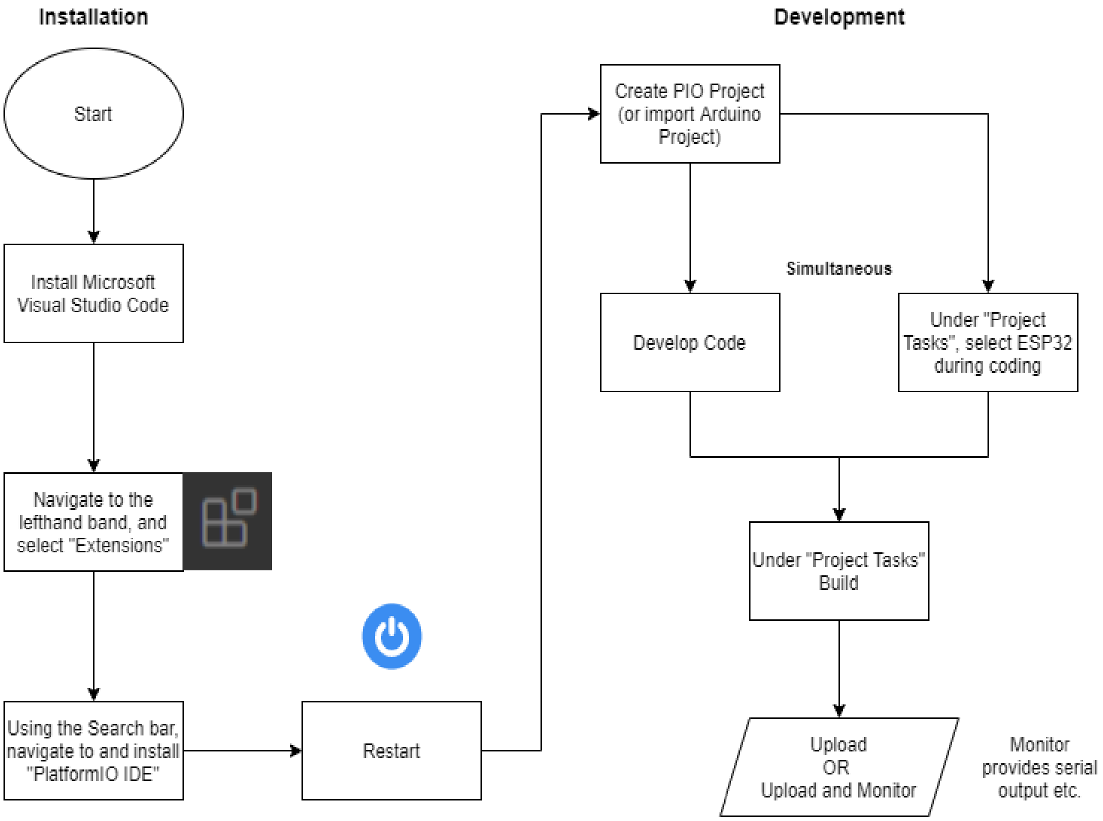

Installation
Start
Install Microsoft Visual Studio Code
Navigate to the lefthand band, and select "Extensions"
Using the Search bar, navigate to and install "PlatformIO IDE"
Restart
Development
Create PIO Project (or import Arduino Project)
Simultaneous
Develop Code
Under "Project Tasks", select ESP32 during coding
Under "Project Tasks" Build
Upload OR Upload and Monitor
Monitor provides serial output etc.

Figure 6-11: Installation and Upload Instructions for PlatformIO

Once PlatformIO was successfully installed on our systems, we downloaded a simple hello world program provided by Espressif – the manufacturers of the ESP32 – and loaded it into a workspace. This code was then flashed to a ESP32 dev-kit through PlatformIO's interface, and the serial response confirmed that both the MCU and PlatformIO was working properly.

Upon confirming the operation of the ESP32 we then moved into familiarizing ourselves with PlatformIO in order to ensure a proper work cycle could be established. Unit Testing is considered an advanced feature, requiring the libraries to be downloaded as well as research into proper implementation. Initial unit tests had been designed with both physical and software pre and post conditions in mind, which is difficult to implement in the same file. In order to overcome this the testing was split into 2 categories: Unit Tests (traditional test-driven development tests in Unity) and Physical Tests (tests which require the user to observe proper behavior in real-space.)

A test file was designed for 4 categories. Game, Queue, Robot, and Wireless functionalities. When a function or class was implemented, a corresponding test was devised to ensure proper functionality. For example, our robot keeps track of the current position of the motors. In order to properly call a step, the motor function has a corresponding test which checks the position of the motor before and after a call to stepMotor() is made. If the position is not equal to the expected result, Unity fails the test. We wrote several checks to allow the program to give more detailed error messages than simple pass fail as well – making debugging a more intuitive process.

Thanks to Unity’s ability to run natively, all of our unit tests were run on-board rather than on a desktop or laptop. Therefore, no issues arose from porting code from a Windows-based environment to the ESP32. This was especially important in the later stages of the project. An update pushed out by Espressif which promised to fix a bug regarding the watch-dog timer was released. While our team had hoped this would aid with the ISR errors which were being encountered at the time, this update caused the ESP32 to crash and then reset whenever a call to Serial2 was made. This was a major issue – as the TMC2209 Silent StepSticks were connected to this port. Ultimately, we had to forgo serial connectivity – despite the fact that the code was confirmed to be working as early as June and was demonstrated during the CDR presentation.

6.3.2 Configuration File

Given the nature of development – as well as extending the lifespan of the codebase – we decided that a configuration file would be useful. This header file was placed in the project directory, and defines the GPIO pins, dimensions of the board, and motor steps-per-mm. This allows for developers to quickly modify values which are used throughout the program, rather than modify numbers within methods or the various header files for each class and functional group.

6.3.3 Initial Wireless Connection

After locking down a workflow, it was decided that testing the wireless connectivity of the ESP32 would be in our best interests. As one of the most complex problems this project faced was the functional netcode, any headway in that area would result in saved time. While the ESP32 was not responsible for overly complex functions – such as AI or image processing – we felt it was best to start early and have working knowledge of the MCUs capabilities when working on the external software. Additionally, having a wireless connection allowed the team to upload firmware to any development kit we had active at the time. Allowing for many test configurations using the materials physically located across the entire team.

With these goals in mind, WirelessController.hpp and .cpp were created and the corresponding test file designed. Initially, the connection was designed such that a HTTP client would be run on the ESP, which would then receive the chess moves in the form of a GET function. This was later dropped in lieu of a WebSocket connection to a server run locally. This allowed for serialized JSON files to be delivered, serialized, and then converted into structs containing only C++ standard components for use in the GameController.

The primary functions of the wireless controller have already been developed in well-maintained libraries which were under the umbrella of an open-source license. The WebSocketsClient library had everything we needed for our device, so the wireless controller was equipped with wrapper functions to connect to the server. While our device is configured to connect to a particular server, it can be quickly modified in the Config to connect to another. The remaining functions to test involved parsing the data as a JSON file, then as data which can be used to drive the game-logic.

To do this, the ArduinoJSON library was imported and called in a queueJsonMove function. This allowed for data to be dynamically read-in, then transferred into a struct containing 2 coordinates (start and end) as well as a special character to flag captures, castling, or other complex chess manuevers. These structs were loaded into a std::queue datastructure to be accessed by the core logic later. This implementation kept our data extremely light-weight; with each JSON move requiring only 9 bytes of data.

$$S_{JMove} = S_{pointer} + 2 * S_{array<int8,2>} + S_{uChar}$$

6.3.4 Motor and Interrupt Service Routine Development

One of the most vital components of the project is the stepping ISR. When building this specific section 3 key concepts were considered essential:

Step Accuracy: The robot should be able to move to as precise a position as possible. While mechanics may constrain this, the code should send exactly as many steps are required to reach a target position.

Watch Dog Timing: It is considered dangerous and extreme bad practice to disable the watchdog timer during operation. Our solution needed to keep its logic as efficient as possible to ensure that the ISR would be completed before the WDT tripped. It is important to note that there are 2 motors – requiring the functions to be called once for each.

TMC2209s: Our stepper motor drivers have physical limitations which much be observed. The steps are counted on the rising edge of a clock signal, with a 10 nano second minimum duty cycle required to count each step. If this is not respected, the device will not function properly.

After identifying the above the team set out to prototype a stepping function. Each motor was put into a class, with a TMC2209 object contained within. This allowed for the registers to be set via UART at configuration, simplifying setup and removing the possibility for user error. In addition to this, a stepper position variable was created. This is a single unsigned-32-bit integer. We opted for the unsigned variant of the integer due to its ability to rollover without any complex logic – allowing the robot to be homed to any location of the board without fear of out-of-bounds motor positions. In the future, a more optimized design would allow a homing function to instead disable step counting and instead only set the position to 0 upon homing, rather than relying on the rollover.

The step function was then called, raising the digital step signal to high (triggering the TMC2209), incrementing the motor location by 1, and then lowering the digital step signal (prepping the ESP for the next signal.) As a proof-of-concept this worked well – but the logic did not account for the step direction of the motor. A check was placed to check which direction the motor was spinning, and either increment or decrement the position by 1 based on this. A more computationally efficient way to implement this would be to enumerate the directions as -1 and 1, for anti-clockwise and clockwise respectively. This would bypass the need for logic in the ISR, as well as allow for rapid assignment elsewhere in the program.

The last component of the step function was to implement a method of ensuring maximum step accuracy. This was done by loading a target position into the motor. If the motor was at said position, the step function would do nothing and raise a flag signaling that the motor was complete. Each check added minimal overhead and required no additional logic. This flag could be used externally by the greater RobotController to signal whether a move had been completed – as both motors needed to complete their movements in order for a move to be complete. The final implementation of the ISR achieves all of the goals set out for it – allowing for it to run regularly without causing un-intended impacts on other functions.

6.3.5 TMC2209 Integration and Concerns

In order to utilize the TMC2209s to their fullest potential, the ESP32 needed to be able to manage a UART connection per device. Future implementations can run TMC2209s as slave devices on the same UART channel, as the step and direction pins are separate from the UART connection and are instead tied to specific GPIO pins. During typical use, this channel is not monitoring the feedback of the motor. But when either configuration or homing, this UART is used to transmit data to the registers of the TMC or to report the resistance faced by the motor. As mentioned above, the Motor class developed by our team included a TMC2209 object. This comes from a library provided by TeamUmlaut which specializes in embedded control systems. Register flashing, StallGuard™ readings, and serial communication maintenance were all supported by this class. It was necessary to provide an active hardware serial port for the driver to use, as well as identifying the address of each of the TMCs in order to use the motor class correctly.

Once the class was tested using a bread-board configuration, some concerns arose based on the physical observations made by the team. We were concerned about the generated heat from the stepper motors and their corresponding drivers – we theorized that over time it may have led to the robot losing positional accuracy and require recalibration too often. These fears were unfounded, as a 30+ minute endurance test demonstrated. It was observed that the robot could maintain accuracy within itself for the entire time moving in every direction possible (the test was a circular pattern.)

6.3.6 RobotControl Class Development

As the base part of the 3-layer approach to firmware, the RobotControl class needed to tie together the Motors already made as well as handle loading moves taken from the game controller. This class is strictly responsible for the physical motion of the robot, and accordingly only includes methods regarding the electromagnet and the motors.

The electromagnet is connected to the ESP32 by a relay. Whichever GPIO pin is set to control the relay is first run through a current amplifier to ensure that there is enough power to throw the switch within. When discussing future design concepts, we did agree that a MOSFET may be desirable, as that would remove the moving parts found within the relay. To ensure maximum configurability, the RobotController class wraps write operations in its own methods. The user can configure whether the relay is active high or low in the config.

In order to determine whether or not a move is complete, the robot controller must monitor the flags raised by each motor. When both flags are raised, the movement is complete. We opted not to introduce an idle() function for the robot controller, as checking 2 flags is easily done within an ISR, but future developers may want to introduce a function which checks the internal logic of the robot controller.

RobotControl is also responsible for loading moves into the motors. The function takes in the direction of each motor, as well as the number of micro-steps to execute. To simplify the traces of the PCB, we opted to hard-code the number of micro-steps to 8. A separate class – GameController – is responsible for generating the moves as that is part of path planning. RobotControl is fed individual moves in the main program – ArcaneCore – whenever there are moves queued by the system, and RobotControl is not currently moving.

6.3.7 GameController Class Development

The middle – and most complex – layer of the firmware stack is the game-state logic. While GameController is the primary class at this layer, it required several support functions to function properly. Specifically, a Queue class which follows a First In, First Out (FIFO) structure and a Piece class to define a piece's starting position and type.

The Queue class was originally a custom solution implemented by a team member. This functioned adequately, but quickly became redundant. Rather than maintain a datastructure ourselves, we opted to use the std::queue provided by C++. The Queue class is effectively a wrapper, with support functions allowing Queues to place other Queues into themselves with a simple for-loop. This was especially useful for the purposes of testing but were otherwise superfluous.

The Piece class required forethought and optimization. At this point, the ESP32 was flashing with roughly 60% of its internal flash taken up. It was determined that memory was the highest priority going forward, as adding memory onto the PCB would complicate the design at a phase where this was unacceptable. This led to the memory-saving implementation detailed earlier.

After writing the piece generation function, the remaining task of the GameController was to process JsonMoves into a series of robot movements in the form of path planning. This required the calculation of dX and dY, a function to retire a piece at a given location using its metadata, and the ability to execute knight movements in such a way that they did not interfere with other pieces on the board.

At the base of all of these is the conversion of dX and dY to motor movements. Our section on kinematics details how the rotation of a motor is translated into cartesian movement with a kinematic expression. This is copied into the code, with the steps-per-mm and mm-per-square factored into the equation to ensure that the robot can be easily tuned via the config.

6.3.8 ArcaneCore

With each layer complete, the final task was to write the core file. Following standard practice for Arduino Framework projects, the program was split into setup and loop. Thanks to the configuration methods contained within the RobotControl class, this was as simple as declaring the 3 layers, calling a pin configuration function defined in Config.cpp, and then connecting to the WiFi and Websocket. The main loop followed basic logic, as detailed and shown below:

Upon starting the loop the controller delays roughly 100 milliseconds. This is to allow the functions to complete using both cores, as the watchdog timer also watches loop() functions in Arduino Framework projects. The program then checks whether or not an event has been logged to the Websocket. The event handler included within the WirelessController class is triggered on an event, but as that handler exists as a lambda function unique to each instance ArcaneCore does not require any implementation of this. We designed the lambda function this way as static methods or normal functions would normally not allow a call to the 'this' pointer.

Once the wireless has been maintained, the RobotControl is polled to see if a move is in progress. If not, it attempts to load a move from the GameController class – with no action being taken if the game has no moves queued.

Finally, if there is a JsonMove parsed from the JSON file received by the WirelessController it will then be loaded into the GameController for path planning. This is placed at the end to be as close to the delay as possible, ensuring that most of the time spent in the time given to the WDT is spent on path planning.

Outside of the main loop, an ISR is running the RobotControl's stepMotors() function. This is constantly running, but future developers could enable and disable the timer fairly easily using the included timer initialization function.

6.3.9 Stretch Goal Assessment

Unfortunately, due mostly to time constraints, Bluetooth functionality and piece recognition were not implemented in our final product. Bluetooth would be a purely software-based solution; the ESP32 does support it natively. With even 1 more week of development it is likely that a user could set the WiFi and Websocket credentials via a terminal, with a potential phone app coming a week after.

\While we were unable to implement piece tracking, we did expend significant energy into researching and designing a solution which could be picked up by future developers. The following details this work, from the underlying principles up to electrical diagrams which were never placed on a footprint – let alone manufactured.

When trying to detect the presence of chess game pieces on a square, there are many factors to consider. Some are how many pieces are we trying to track at a time, what resolution of positional accuracy do we need, and how quickly do we need to know where a piece is at? One solution we thought of that we thought could get the job done while not exploding our budget is to have a reed switch under each square of the game board and have a small magnet embedded within each chess game piece. This could give us a binary signal indicating whether or not a piece is present on a particular tile. Given that there is a consistent setup to chess, each piece has a specific set of moves it can make, and that only one piece moves in a turn, we think we can track the triggering of reed switches to determine which piece has moved and where on each turn. The next thing to figure out is how we're going to be able to read the state of all our 64 reed switches when we only have 34 GPIO pins on our microcontroller.

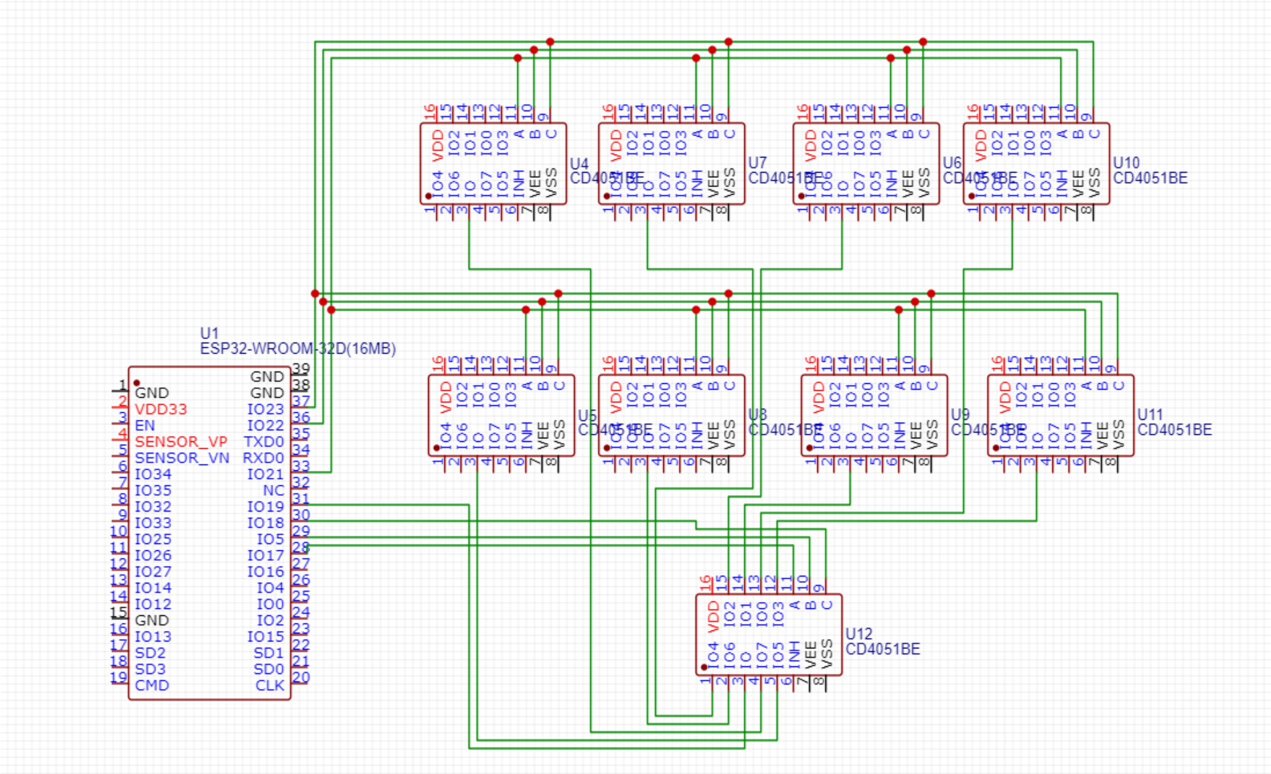


Figure 6-12: Possible Piece Recognition

Above, *Figure 6-12*, is the circuit that is the answer to our prayers roughly drawn in OneNote. It is essentially a 64 to 1 multiplexer. This allows 6 control bits to tie any of the 64 input signals to the output given a specific pattern on the control bits. This means that by cycling the control bits, we can poll each of our reed switches to see if there is a piece sitting just above it. 1 bit is used for the output of the multiplexor, so effectively only 7 GPIO pins would be needed from our microcontroller to read all our reed switches.

Try as we might, we could not find a 64 to 1 multiplexor, which in turn would require the use of 9 8-to-1 multiplexors instead. We found the CD4051chips on amazon which is a DIP style chip that functions as both a multiplexor and demultiplexer while supporting input voltages that range from 3 volts to 20 volts. Having this component available on amazon would mean short shipping lead times and the Dip style interface would allow one to solder them onto a PCB without the use of a microscope and tweezers.

To cut down on the total volume of copper wire being used while also making the top playing surface easier to remove, a daughter board may need to be fabricated. This daughter board would be mounted just under the playing surface and house all of the 8:1 Multiplexors shown in figure 6-6. All of the reed switches adhered under each tile on the checkered board playing surface will be wired to the daughter board rather than running all the way back to the main PCB. This way only 9 wires (the 7 control signals, the output, the ground, and the 3.3v rail) need to run between the daughter board and the main PCB. This could cut down on the number of pins which need to be exposed in the main board and would make a ribbon cable or some other sort of quick-change connector viable. All these wires would carry logic signals, so there would be no need for any thick cables between the main board and this proposed daughter board.

However, we were able to implement our stretch goal of creating a game version allowing the player to play against AI which was really exciting.

7.0 PROJECT PROTOTYPE TESTING PLAN

In order to properly construct the prototype, systems were tested individually before being fitted together. Software errors in particular were difficult to trace when only given physical observations – as the layers between software and physical reaction were full of potential points of failure.

As detailed in section 6.3, unit testing was performed natively on ESP32 dev-kits and our PCB prototypes using the Unity Embedded Unit Testing Framework. PlatformIO has integration which allows for each test to be run back-to-back or individually at the developer's discretion. Upon completing each test, the ESP32 reports a success or a failure via the serial connection. Each test file is compiled and flashed individually.

7.1 H-BOT TESTING

The H-Bot is a crucial part of this project and had to undergo a series of tests to ensure it was functioning as desired before our final project was assembled.

DESCRIPTION OF TEST	CLEAR CONDITION
H-Bot follows X axis equation of motion.	Center carriage moves in X direction when motors are spun in opposite directions.
H-Bot follows Y axis equation of motion .	Center carriage moves in Y direction when motors are spun in same directions.
H-Bot maintains precision.	Center carriage makes contact with limit switches repeatedly wen moving a set distance towards and away from them .
H-Bot is silent.	We record no higher than 50 decibels from 3 feet away from the board when moving a game piece.
H-Bot is accurate.	The carriage moves to a target destination and makes contact with a limit switch

Table 7-1: H-Bot Testing

There are plenty of active electronic components that needed to be tested as well. Core components like the electromagnet must be evaluated to ensure it is functioning as it should. The stepper motors were evaluated to ensure that they are being properly controlled and read from the stepper motor drivers. Refer to the table below, *Table 7-2*, for component testing details.

DESCRIPTION OF TEST	CLEAR CONDITION
Electromagnet stays cool.	Electromagnet does not exceeded 45 degrees Celsius after 15 minutes of 5V power .
Multiplexors correctly tie inputs to outputs.	Continuity checks are evaluated between each input and the output of all 8:1 multiplexors as the selection bits are stepped through.
Stepper motors don't skip steps.	A lever arm attached to motor shaft trips a limit switch the exact number of times the motor is programmed to complete a revolution.

Table 7-2: Component Testing

7.2 SOFTWARE TESTING

In order to verify that the software is operating correctly there will be several areas to focus on independently: the on-board MCU firmware, the off-board controller application, and the internet connection between the 2. Unit tests should be conducted during development to ensure as little debugging as possible when assembling the final product.

7.2.1 MCU Firmware Testing

The MCU is responsible for establishing and maintaining a serial connection, interpreting commands received over that connection, movement of the cartesian robot, and properly responding to button interrupts. Some of these, such as the serial connection and receiving GCODE commands, are interconnected in such a way that an error in one will manifest as an error in the other. This means that it is imperative that critical systems be tested as thoroughly as possible to avoid misdiagnoses – hence the more rigorous testing. The following sections detail these specific tests in *Tables 7-3* through *7-7*.

Section 7.2.1.1-7.2.1.4 is tested using the ESP32 development kit obtained before the delivery of the board's bespoke PCB. This is being done to ensure that as much debugging is completed as possible before the delivery. There is an additional benefit in that there will be a clear delineation between hardware and software errors, if any such errors exist in the PCB design. During the final construction of the Arcane Game board it is critical to properly identify the source of incorrect behavior.

7.2.1.1 RobotControl and Motor Class Testing

To ensure that the mechanical frame could be tested without interference from coding errors, the following Unit tests were devised. This was especially crucial, as discerning a loss of accuracy from erroneous code and failing construction can be notoriously difficult when working with 3d Printers. Some of the following are actually several unit tests condensed into one table entry for the sake of brevity.

DESCRIPTION OF TEST	CLEAR CONDITION
Testing the initial parameters of a Motor Object	After the default constructor for a Motor is called, the motor is set to position 0 with target position 0 with no errors on instantiation.
Testing that the motor can correctly set a target position set	The target position is reported correctly, and the steps-to-go function reports the correct amount.
Testing that setting the direction of a motor results in the correct pin out	The GPIO pin assigned to that motor's direction reads as the appropriate value
Ensuring that a motor can step properly	The current position and steps-to-go value is incremented/decremented correctly.
Testing whether or not a motor raises its complete flag correctly	When a motor steps into its target position, its flag indicating the movement is complete raises.
Checking that a RobotControl object instantiates correctly	The RobotControl does not read as executing a move, and of its motors exists.
Attempt to load a move into the RobotContol's motors	The RobotControl's motors pass all of the tests involving setting target positions and steps.
Run the stepMotors() method	Both motors should take a step in the correct direction and set their flags accordingly
Run stepMotors() until a move is complete	When both motors set their flags, the RobotControl should raise its inactive flag.

Table 7-3: RobotControl and Motor Class Unit Tests

7.2.1.2 WirelessController Class Testing

While assessing the Wi-Fi functionality of the ESP32 firmware we develop we will use a windows-based computer to power and verify the data being transmitted from the board. The router which runs the Wi-Fi network must also be accessible to ensure that the device is properly connecting for the basic connection test. The HTTP server must be online in order to complete the vast majority of tests, as the main purpose of the Wi-Fi connection is to retrieve movement commands from that server. This makes the later tests a lower priority, but the tests which toggle operation between Bluetooth and Wi-Fi operation are vital at the earlier stages.

DESCRIPTION OF TEST	CLEAR CONDITION
Wi-fi credentials are hard-coded to the ESP32 and the device is set to attempt to connect to the network	The ESP32 successfully connects to the network within the defined timeout window
A websocket address is provided and the ESP32 is instructed to connect	Both the websocket server and ESP32 report a connection has been established
A JSON file is provided locally as a string and fed into the JSON deserializer/decoder	A struct containing the start, end, and special flag of the move is returned by the WirelessController
A Webserver is set to consistently send serialized JSON files, with the ESP32 listening wirelessly	The ESP32 registers an event in which it received a string.
Several JsonMoves are placed into the queue, then dequeued	The JsonMoves dequeued match those queued, with the order and data preserved perfectly.

Table 7-4: WirelessController Unit Tests

7.2.1.3 GameController Testing

This is the final firmware testing which can be completed before the cartesian robot is built. The interpreter will post serial outputs which display the state of the buffer and history stack when receiving new commands. If a function is called by a command, the serial output will report that the function has been called. If an interrupt would be triggered, the serial output will contain a report of the state of the system before the interrupt, the actions taken during the interrupt, and the state of the system after the interrupt.

DESCRIPTION OF TEST	CLEAR CONDITION
A dX and dY is given to the xyToMotors() static function	The resulting move has the directions, number of steps, and electromagnet state correct.
Several moves are loaded into the GameController's queue	Each move is popped off of the queue, in order, with each move's data being retained properly.
A call to pop off a move is made to an empty queue	No actions are taken, and the program continues without error.
The ID for the black rook beginning in column A is calculated, and then the initializePieces method is called	The piece found at A8 is the black rook with the correct ID.
A Piece has its position changed in memory, and then the getPieceAtPosition() function is called	The correct piece is identified at being at the given position.
A piece's getRetireRow, getRetireCol, and getPieceType method are called	All 3 get methods return the correct enumeration for their respective components.

Table 7-6: Gcode Interpretation Unit Tests

7.2.1.4 Mechanical Control Testing

In order to complete these tests in their entirety the cartesian robot must be completed. While motor control using the TMC 2209s can be tested, the positional accuracy of the commands can only be verified on the machine itself. Therefore, these are the last unit tests to be completed on the firmware. By using Marlin-based movement control functions there should be minimal debugging to perform at this stage; substituting our own code would introduce a greater chance of failure for the project.

DESCRIPTION OF TEST	CLEAR CONDITION
Queuing several movements into a Queue by calling xyToMotors()	The robot moves the correct distance, according to the parameters in Config.hpp
A transposition is called given the dX and dY of a knight piece	The robot moves diagonally in the direction of the knight's movement, ending on the space between squares.
A Motor object has its direction set	The GPIO pin assigned to that motor's direction reads as the appropriate value
The enableMagnet and disableMagnet methods are called	The robot enables and disables the magnet when the calls are made (the relay should audibly click)

Table 7-7: Mechanical Control Unit Tests

*TMC2209 Stepper motors must be configured correctly, else use homing switches

7.2.2 Web Application Testing

In order to ensure a quality and fully functioning web application, our team conducted a series of common tests on both the front-end and back-end of our web application. The objective with testing our web application was to ensure that it provided all of the features necessary to complete our main goals, while also operating at an optimal speed in which is entirely usable. Our tests were conducted in a way that measured both qualitative and quantitative metrics to ensure a high standard experience for the users. Additional tests were also conducted to ensure that the data transmitted between the front-end and the back-end was coherent, and provided all of the necessary security in order to prevent future vulnerabilities such as duplicate rows or mismatched API calls.

7.2.2.1 React Front-end Testing

In order to ensure that our React front-end was working and provided all of the necessary features, we utilized some testing software, as well as some common industry testing methodologies. Our focus when testing the front-end was to ensure that all of our components were displaying correctly and as intended. Our additional focus was on ensuring that the data that was being gathered by our front-end was accurate. Finally, we created test scripts using the software testing tool Jest, in order to automate testing for several different features in the web application.

DESCRIPTION OF TEST	**CLEAR CONDITION**
Jest script to run on our login page, providing several correct login usernames and passwords	The web application will recognize the correct login information and forward the user to main page.
Jest script to run on our login page, providing several incorrect login usernames and passwords	The web application will recognize the incorrect login information and display a validation error requesting a new username and password.
Qualitative analysis of all visuals and buttons to ensure all components are functional and match the prototype	All of the React components must appear as previously described in the Adobe XD prototype.
JavaScript test script will be written on GET request for historic moves in order to test front-end connection to the back-end	The API connection established on the front-end will return the appropriate request code
Redux variables will be logged to the console when they are initially loaded in order to ensure that they are up to date with local variables	The console will print out the value of both the local variable that is set and the Redux variable in order to ensure validity.

Table 7-8: React Front-end Testing

7.2.2.2 Node.js API Testing

The Node.js API plays a crucial role in linking our React front-end to our MySQL back-end. The API bridges the gap between the two entities by providing a series of code that handles data as a JSON package between the two. The Node.js API also plays a crucial role in creating data in our relational database which can then be later recalled by our application in order to provide desired features. We achieve this through a series of addresses that are known as endpoints. Different endpoints exist for different purposes, such as GET and POST. A series of tests will be done on the API using software tools as well as some direct approaches from the front-end which have already been listed. The main software tool that we will be using to test the Node.js API is Postman API Development environment. Postman allows developers to test individual endpoints to ensure that the correct data is being received. If a segment of stored data is not appearing correctly on the web application, the developer has no way of knowing which entity is causing this issue. Using Postman eliminates any doubt that the front-end may be involved in situations where data is not accurately handled. In the end, we discovered the WebSocket was more than sufficient in transferring data, and we opted to use that for our main source of data transfer.

DESCRIPTION OF TEST	CLEAR CONDITION
All GET Request endpoints will be tested using Postman to ensure that they are returning the correct data	Postman will show the correct JSON data package received from the database and will provide a 200 Request code.
All POST Request endpoints will be tested using Postman to ensure that the data that has been passed to be inserted into the database has accurately been stored.	Postman will show the correct returning JSON data package response received from the database and will provide a 200 Request code. Additionally, the database will be checked to see if the new data has been added accurately.
A stress test script will be ran in Postman on our endpoints in order to accurately determine a baseline for performance for our API.	Postman will return the desired metrics of the stress test and provide a baseline.
JavaScript test script will be written on GET request for from the front-end on historicMoves endpoint in order to test front-end connection to the back-end	The API connection established on the front-end will return the appropriate request code

Table 7-9: Node.js API Testing

7.2.2.3 MySQL Database Testing

The MySQL Database is a crucial tool that serves as the data hub for our backend. It is populated directly from information passed by our React front-end as a JSON package through our Node.js API. To ensure that our data is kept in a safe manner, and that there are no duplicate rows or columns, we utilize some of MySQL Database Management Tool’s features to lock certain incoming data and provide a screen of validity before adding data to the database. The purpose of these tests is to ensure that harmful data potentially cause issues with some front-end features. Throughout the project, we realized the library we were using was capable of providing real time game history, resulting in us opting out of using the database.

DESCRIPTION OF TEST	**CLEAR CONDITION**
Data constrains will be tested by providing repeat data to ensure that new historicMoves data is not duplicated	The historicMoves table will reject the incoming.
Select queries will be ran in order to test database performance and ensure delivery of data is occurring in a timely manner	Query will return desired data rows in a timely manner.

Table 7-10: MySQL Database Testing

8.0 ADMINISTRATIVE CONTENT

This section contains general content that served the purpose to keep our project organized and on schedule. We have included both our budget and our project milestones. The benefits of having planned our budget and milestones in advance was that we were able to stay on track throughout our research, design, and implementation stages. It has been imperative that we stick to a defined schedule to ensure we are able to meet our goals and objectives within the appropriate time frame as well as keeping track of our spending to avoid going over budget given the fact the project is self-funded meaning overall funds are limited.

8.1 BUDGETING AND FINANCE

As we planned our project design, we had to also determine an estimated budget to ensure our design is reasonable from both and engineering and marketing perspective. We also needed to consider the limitations of our available funds since the project is self-funded by each team member.

The table below (*Table 8-1*) contains all the parts we determined to be necessary for this project along with a quantity and cost. Our goal was to keep our budget under $500. Overall, our budget came out to about $405.

Estimated Budget			
Description (#)	*Quantity*	*Estimated Cost*	*Total Cost*
400mm Linear Motion Rod	2	$7.19	$14.39
500mm Linear Motion Rod	2	$7.19	$14.39
LM8UU Linear Ball Bearing (*12*)	1	$10.95	$10.95
Timing belt + Pulley Wheel	1	$16.99	$16.99
Electromagnet	1	$9.99	$9.99
Rail Clamps (*4*)	1	$11.99	$11.99
Nema 17 Stepper Motor (*3*)	1	$25.99	$25.99
PCB Version 2.0	1	$71.41	$71.41
PCB Version 3.0	1	$78.03	$78.03
12V AC/DC Adapter	1	$9.99	$9.99
Threaded Inserts	1	$9.19	$9.19
Bushings	1	$5.99	$5.99
Acrylic Sheet	1	$24.99	$24.99
Wood	2	$14.99	$29.98
Relay (*10*)	1	$7.86	$7.86
Felt	2	$1.89	$3.78
Header Pins (*40*)	1	$5.29	$5.29
Barrel Jack (*10*)	1	$10.99	$10.99
Neodymium Magnets	1	$22.44	$22.44
PLA Plastic Spool	1	$19.99	$19.99
Total:			**$404.44**

Table 8-1: Final Project Budget

8.2 MILESTONES

One of the most crucial steps in project planning is outlining important deadlines and creating a realistic schedule by which to complete critical project goals. Due to the nature of this project, we had to consider each team member’s outside commitments, such as class or work, and plan our schedule accordingly. Additionally, we had to also consider any potential roadblocks we may encounter and do our best to plan for them to prevent problems further into the project.

Milestones and Due Dates		
Senior Design 1		
Week	*Date*	*Task*
1	1/17/21-1/23/21	Project Ideas/Brainstorming
2	1/24/21-1/30/21	Project Selection
3	1/31/21-2/6/21	Project Specifics and Finalization
4	2/7/21-2/13/21	Divide and Conquer 2.0
5	2/14/21-2/20/21	Design Research
6	2/21/21-2/27/21	Decision on piece moving mechanism
7	2/28/21-3/6/21	Complete first 30 pages of Report
8	3/7/21-3/13/21	Research and report
9	3/14/21-3/20/21	Decision on piece identification
10	3/21/21-3/27/21	Decision on power supply
11	3/28/21-4/3/21	Complete/submit first 60 pages of Report
12	4/4/21-4/10/21	Report writing
13	4/11/21-4/17/21	Complete/submit first 100 pages of Report
14	4/18/21-4/24/21	Finalize Report
15	4/25/21-5/1/21	Complete/submit Report (120 pages)
Senior Design 2		
0-2	5/2/21-5/30/21	Plan and Order Parts
3-5	5/31/21-6/20/21	Finalize design and order PCB Ver 2.0
2-3	5/24/21-6/6/21	H-bot assembled
6-7	6/21/21-7/4/21	H-bot movement working
8	7/5/21-7/11/21	Test PCB Ver 2.0
8	7/5/21-7/11/21	Design and order PCB Ver 3.0
8-10	7/5/21-7/24/21	Finish board assembly and pieces
10	7/18/21-7/24/21	Test PCB Ver 3.0
10	7/18/21-7/24/21	Fully functioning project
11-12	7/25/21-8/3/21	Presentation and final report

Table 8-2: Project Milestones and Due Dates

Through creating the table above, *Table 8-2*, we are able to get an idea of what we needed to accomplish and how much time we had to accomplish it. We are then able to plan more specifically how we will divide the workload in order to meet the necessary deadlines. Using the milestones as a guide, we assigned weekly goals to each team member and were able to adjust the goals as needed based on the overall weekly team progress. We then organized these goals into smaller, measurable tasks with assigned deadlines. Implementing this method of task organization greatly assisted our team in meeting our deadlines and ultimately delivering a working final product.

9.0 PROJECT OPERATION

This section will serve as the user operating manual for the Arcane Game Board. We will cover the operating requirements and required materials which will cover everything needed in order to successfully setup and use the game board. We will also cover the full setup, proper use of the device and proper storage.

9.1 REQUIRED MATERIALS

In order to successfully operate the Arcane Game Board, there are a few key things that are absolutely necessary for the game board to work as intended.

- Stable internet/Wi-Fi connection
- 12V 5A AC/DC adapter with 2.1mm barrel jack (*would be included*)
- 32 chess pieces with embedded 3mm neodymium magnet (*would be included*)
  - Magnet polarity must be opposite of electromagnet polarity
- Desktop or Mobile device to access web application

Since the ESP32 microcontroller can retain its memory, there is no need for the external USB breakout board to flash the firmware since everything needed to play a game from the web application is already on the board. The board simply needs to be supplied with power. If this project were to go into production, the unique chess pieces and power supply would of course be included in the set. Mainly, the user needs a device capable of internet connection in order to access the web application and communicate with the board.

9.2 SETUP & PLAY

Setup for the Arcane Game Board is relatively straight forward and little more effort than the setup for a normal chess board.

<u>Step 1:</u> Assemble the pieces on the board for a typical chess game. The primary play space exists within the checkerboard pattern. The empty squares on either side are for the graveyard, pieces should not be set up in these rows. White should be set up on the side nearest to the side with the circuit board/where the board plugs in. *Figure 9-1* below depicts the proper piece setup.

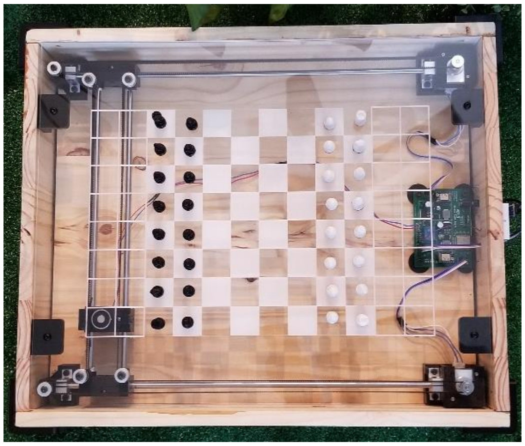


Figure 9-1: Proper Piece Setup

<u>Step 2:</u> Plug in the power supply to the wall and into the barrel jack located on the side of the board. The board is now “ON” and ready to receive information from the web application.

Step 3: Open and Log in to the Arcane Game Board application. There is a username and password field that must be filled in to access the game.

Step 4: Select the game mode you wish to play. You can select either '*Player VS Player*' or '*Player VS Stockfish*' which is our AI opponent. In '*Player VS Player*" you can choose to either play on one device or two. If you wish to play on two, simply open up a '*Player VS Player'* game on another device.

Step 5: Begin playing! To move in the application, click and drag your piece to its desired location. You will then be able to watch as the game board replicates the same move. Do not move any pieces on the board physically, the Arcane Game Board will take care of every movement throughout the duration of the game.

Step 6: When the game ends, you can either reset the board for a new game and refresh the application or you can put it away. When you are done playing, be sure to unplug the game board and store in a dry place ideally at room temperature. Do not store heavy objects on top of the game board as it could damage the acrylic top.

10.0 TEAM DESCRIPTION

Our team consists of three computer engineering students and one electrical engineering student. Each team member has a unique technical background and we used each individual's skills to develop a working prototype of the Arcane Game Board. There is also substantial overlap among each member's knowledge which allows us to collaborate with each other more effectively. Additionally, this overlap allows us to better understand how each part of the project impacts one another and design accordingly.

Anton Strickland is a Computer Engineering student whose background mainly lies in embedded system development and database management. During the course of writing this document his focus was on learning the existing ecosystem of code available for the ESP32, as well as taking Marlin apart in order to repackage the movement code into a functional command set for the cartesian robot. During the development of the actual physical board Anton is the primary developer of the MCU firmware, as well as assisting Lucas in the development of event handling, providing requirements to Kayla for PCB design, and aiding Fernando with the construction of the cartesian robot and game board.

Kayla Freudenberger is an Electrical Engineering student whose experience consists mainly of circuit design and integrated circuits. Additionally, she has extensive experience in the realm of technical writing which assisted the team with the overall organization of our document. During the development of this document, she focused on electrical component selection and schematic designs for the project as well as many of the less technical sections. Being the sole electrical engineering major on the team, she is the primary lead for the PCB design and power system design when we begin developing our prototype in addition to assisting Fernando with the construction of the game board.

Fernando Valdes-Recio is a Computer Engineering student who has a background in programming as well as CAD modeling. His focus in this project was the mechanical aspects of the Arcane Game Board as he had the most experience of the team in that section and he supported software and

PCB development after the other physical portions of the project were completed. He designed custom parts using SolidWorks and had them printed on his 3D printer.

Lucas Lage is a Computer Engineering student who has a background in full stack development. His primary focus is handling the GUI and the necessary backend to support the companion app for this project. He is in charge of configuring our database and he is working closely with Anton to ensure that the web app and the embedded software on the microcontroller are able to communicate to each other.

11.0 CONCLUSION

The Arcane Game Board offers an exotic way to enjoy the classic game of chess with its movement systems and web-interface which will entertain new and veteran players. With the potential for real-time piece tracking on the horizon we consider this product a starting point of an idea which would allow many to enjoy a game with friends who are unable to play with them in person. Many technical hurdles had to be overcome, but they empower a non-technical user to enjoy the product reliably and without the need for comprehensive knowledge of our system.

Our team planned the tasks required to fulfill a baseline product well in advance with a thought-out production schedule tailored to the challenges presented by supply chain disruption and remote development. By adhering to our assigned responsibilities – each of which plays to our team's unique strengths – the physical and software components of the Arcane Game Board are the best that we can make it. No individual member was without support in the event of disruptions during the development phase.

Using a cheap yet powerful processor – the ESP32 – allows our PCB design to be as straightforward as possible while still yielding features such as wireless connection, advanced stepper motor drivers, and robust on-board event handling. Due to the composition of our team this was an important design choice; if a more basic CPU were used, we would have had to reduce the scope of the board in other areas to dedicate more time to PCB development. Power delivery is simplified as well, with the only voltages required across the board being the 3.3V MCU, the 5.0V electromagnet and the TMC2209 stepper motor drivers which comfortably operate between 12V and 24V.

By leveraging existing software such as React, Marlin, and the many open-source projects using ESP32 microcontrollers we were able to achieve a relatively complex system capable of transmitting moves between boards over the internet. Users are able to see the game represented in their digital UIs, as well as the board in front of them as the cartesian robot moves quickly and accurately to keep the physical pieces in sync with their digital counterparts. Software updates to our board can even be uploaded wirelessly with our development environment: PlatformIO.

The physical board features a CoreXY cartesian robot – a movement system which was selected for its ability to move its head without moving the additional mass of any motors. Wear from moving wires is also eliminated thanks to the fixed motor positions – increasing device longevity. Due to the widely available parts used in its construction and development kits for the ESP32 a working prototype was feasible before the delivery of the final PCB; reducing the risk of project failure and ensuring decisions can be driven by real-world observation alongside careful reading of datasheets and example code.

The above is only the beginning of a product with the potential to surpass the products we saw as inspiration – features such as physical piece tracking, interchangeable rulesets, and more are possible with more time and development. It is our hope that devices such as this can bridge the gap between the convenience of digital games and the satisfying intuitiveness of in-person boards. This project has the potential to do this for chess, as well as any other board game which could fit in the same space.

12.0 APPENDIX

[1] M. F. C. B. Baran Usluel, "Square On: The Magic Chess Robot," [Online]. Available: https://baranusluel.com/square-on/. [Accessed March 2021].

[2] "Square Off," Square Off, 2020. [Online]. Available: https://squareoffnow.com/. [Accessed March 2021].

[3] E. Ackerman, "Harvard's milliDelta Robot is Tiny and Scary Fast," 17 January 2018. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/industrial-robots/harvard-millidelta-robot-is-tiny-and-scary-fast. [Accessed 3 March 2021].

[4] S. Lahteine, "MarlinFirmware," 29 October 2020. [Online]. Available: https://github.com/MarlinFirmware/Marlin/blob/2.0.x/Marlin/src/module/delta.cpp. [Accessed 3 March 2021].

[5] "Electric Batteries," Wikipedia, 21 March 2021. [Online]. Available: https://en.wikipedia.org/wiki/Electric_battery. [Accessed 30 March 2021].

[6] "Mains Electricity," Wikipedia, 11 March 2021. [Online]. Available: https://en.wikipedia.org/wiki/Mains_electricity. [Accessed 31 March 2021].

[7] H. Zhang, "Basic Concepts of Linear Regulator and Switching Mode Power Supplies," Analog Devices, [Online]. Available: https://www.analog.com/en/app-notes/an-140.html. [Accessed 14 April 2021].

[8] M. Woolley, "Bluetooth, Meet the Internet. Internet, Say Hello to Bluetooth," Bluetooth SIG, 2015.

[9] Microsoft Corporation, "Microsoft Software License Terms," 2021. [Online]. Available: https://code.visualstudio.com/license. [Accessed 26 April 2021].

[10] The Apache Software Foundation, "Apache License, Version 2.0," January 2204. [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0. [Accessed 26 April 2021].

[11] "EAGLE (program)," Wikipedia, 1 April 2021. [Online]. Available: https://en.wikipedia.org/wiki/EAGLE_(program). [Accessed 15 April 2021].

[12] Autodesk, "Terms of Use," 25 February 2021. [Online]. Available: https://www.autodesk.com/company/terms-of-use/en/general-terms?_ga=2.1134002.2065514942.1619491331-698093324.1617054993#use. [Accessed 26 April 2021].

[13] "AboutEasyEDA," EasyEDA, [Online]. Available: https://easyeda.com/page/about. [Accessed 15 April 2021].

[14] Free Software Foundation, Inc, "GNU Lesser General Public License," 29 June 2007. [Online]. Available: https://www.gnu.org/licenses/lgpl-3.0.en.html. [Accessed 26 April 2021].

[15] "JLCPCB SMT Parts Library," JLCPCB, 2021. [Online]. Available: https://jlcpcb.com/parts. [Accessed April 2021].

[16] "IPC Standards," IPC, Association Connecting Electronics Industries, [Online]. Available: https://www.ipc.org/ipc-standards. [Accessed 25 April 2021].

[17] D. Kumar, "Things to know about the IPC Standards in PCB Designing," Circuit Digest, 23 January 2020. [Online]. Available: https://circuitdigest.com/tutorial/things-to-know-about-ipc-standards-in-pcb-designing. [Accessed 25 2021 April].

[18] Texas Instruments, "LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator," April 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1627900062283&ref_url=https%253A%252F%252Fwww.google.com%252F. [Accessed April 2021].

[19] "Relay," Wikipedia, 10 March 2021. [Online]. Available: https://en.wikipedia.org/wiki/Relay. [Accessed 20 April 2021].

[20] Texas Instruments, "Texas Instruments Online Terms of Use," 29 May 2018. [Online]. Available: https://www.ti.com/legal/terms-conditions/terms-of-use/north-america.html. [Accessed 18 April 2021].

[21] T. Instruments, "Webench Power Designer," [Online]. Available: https://webench.ti.com/power-designer/switching-regulator/select. [Accessed 15 April 2021].

[22] T. Instruments, "Webench Power Designer," Texas Instruments , [Online]. Available: https://webench.ti.com/power-designer/switching-regulator/select. [Accessed 15 April 2021].

[23] Trinamic, "TMC2209 Datasheet," 26 June 2019. [Online]. Available: https://www.trinamic.com/fileadmin/assets/Products/ICs_Documents/TMC2209_Datasheet_V103.pdf. [Accessed March 2021].

[24] S. Kraig, "Picking the Right Trace Width," MacroFab, 25 October 2016. [Online]. Available: https://macrofab.com/blog/picking-right-trace-

width/#:~:text=6%20to%2030%20mils%20is%20typical%20for%20most%20signal%20trace%20widths.. [Accessed 20 April 2021].

[25] R. Williams II, "The Delta Parallel Robot Robot: Kinematics Solutions," October 2016. [Online]. Available: https://www.ohio.edu/mechanical-faculty/williams/html/PDF/DeltaKin.pdf. [Accessed 3 March 2021].

[26] R. W. Budi Hadisujoto, "Development and Accuracy Test of a Fused Deposition Modeling (FDM) 3D Printing using H-Bot Mechanism," *Indonesian Journal of Computing, Engineering, and Design (IJoCED),* vol. 3, no. 1, pp. 46-53, March 2021.

[27] T. Instruments, "Webench Power Designer," Texas Instruments , [Online]. Available: https://webench.ti.com/power-designer/switching-regulator/select. [Accessed 15 April 2021].