The Arcane Game Board

Lucas Lage, Fernando Valdes-Recio, Kayla Freudenberger, J. Anton Strickland

Dept. of Electrical and Computer Engineering, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — This paper will present the design approach and implementation of the Arcane Game Board. The Arcane Game Board is a computer and electrical engineering project designed to automate physical player movements and introduce the possibility of long-distance gameplay while still utilizing a tangible game board. For the purposes of this project, the Arcane Game Board is programmed and designed to play Chess. This project is implemented using stepper motors and an electromagnet beneath the board's surface to accomplish hands-free piece movements along with an interactive web application for user inputs. The Arcane Game Board will accommodate exactly two players playing against one another and can be played from the same or separate devices provided both have access to the web application.**

***Index Terms* — Robot Control, Printed Circuits, Electromagnetic Devices, Application Software, Autonomous Systems, Mobile Applications**

I. Introduction

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. With the COVID-19 pandemic keeping people socially distanced, our group wanted to create a game board that was able to operate long-distance while still maintaining the classic board game feel.

The Arcane Game Board utilizes a combination of software and hardware to create and interactive and unique gameplay experience. The key feature of this project is the game board's ability to move pieces without human intervention. This feature will be accomplished with an X-Y cartesian robot beneath the play surface which uses Nema 17 Stepper Motors to move an electromagnet to all possible piece locations. When actively moving the pieces, the electromagnet will be powered on and subsequently move the chess pieces, of which each have a small magnet embedded inside. We will be using an ESP-WROOM-32D microcontroller to control all the moving parts of this project; namely the stepper motor drivers and the relay which toggles the electromagnet.

On the user end, we developed a web application which is where the player will indicate their desired moves by clicking and dragging the pieces. These moves will then be communicated to the microcontroller, allowing the physical board to replicate these piece movements in real time.

II. Major System Components

During our project research we decided on several components that fulfill our hardware needs. The figure below is a diagram of our major hardware components. In this section we will discuss these components in detail.

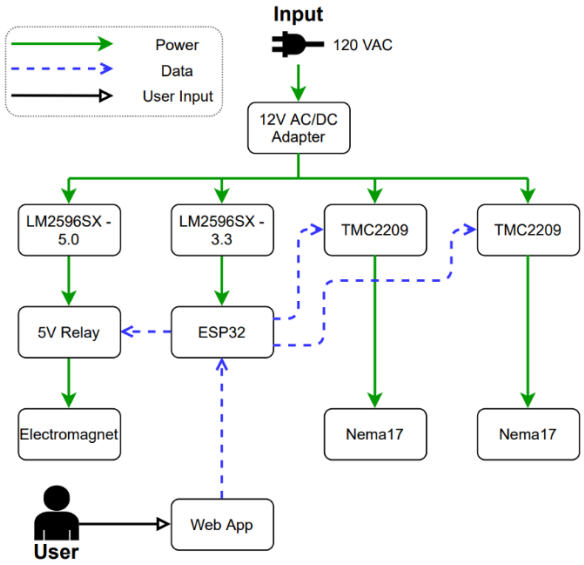


Fig. 1. Overview flowchart of major system components and corresponding power and data connections.

A. Microcontroller

The main component that makes the whole project possible is the ESP32 – WROOM – 32D manufactured by Espressif Systems. This microcontroller features a dual-core 32-bit LX6 microprocessor at 160MHz and a 240MHz Core Clock making it a fine choice for our project. One of the major benefits of using this microcontroller is that it has an integrated Bluetooth and Wi-Fi module which vastly saves both money and complexity in terms of the hardware design. The ESP32 will be used to control the stepper motor drivers and the electromagnetic relay based on information sent to it from our web application.

B. Motors

To accomplish our movement, we are using 2 Nema 17 stepper motors. We chose stepper motors because they have reliable open loop position control which allows us to have accurate positioning without the need for additional sensors. The Nema 17 motors are popular motors for use in 3D printers which has led to their mass production and subsequent cheap cost. The low cost and high positional accuracy of these motors made them the best choice for our movement design.

C. Stepper Motor Drivers

To drive our Nema 17 motors, we are using the TMC2209 manufactured by Trinamic. The major benefits of using this motor driver is the sensorless homing, StealthChop mode, and large voltage range. The sensorless homing saves us additional complexity by reducing the need for sensors and the large voltage range gives us more freedom in our power system design. Additionally, the StealthChop mode negates most of the noise when the robot is operating, creating a more enjoyable user experience.

In order to use the sensorless homing feature and StealthChop mode, we must operate the TMC2209's by utilizing the UART connections to pass information back to the microcontroller. Unfortunately, the ESP32 crashes when using UART2 meaning we will not be able to use sensorless homing and StealthChop in our final design. [1]

D. Electromagnet and Relay

The electromagnet will be toggled on and off to move pieces across the board. For our design we selected a 5V electromagnet with 50N holding force manufactured by Uxcell. This electromagnet was selected because of its low power consumption and reasonable size, measuring 25mm x 20mm. To toggle the electromagnet on and off, we are using the SRD-05VDC-SL-C which is a simple and commonly used electromagnetic relay. 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.

E. Power System

The Arcane Game Board is designed to be plugged into a wall outlet using a 12V 5A AC/DC adapter. This adapter connects directly to the custom board via a barrel jack connector. Our hardware design can be broken into three main sections: the microcontroller, the motor drivers, and the electromagnet; each of these require different voltage inputs. The table below details the voltage and current demands for these components.

VOLTAGE AND CURRENT DEMANDS			
Component	*Voltage Range (V)*	*Operating Voltage (V)*	*Operating Current (A)*
TMC2209	4.75 – 29	12	0.85 each
ESP32	3.0 – 3.6	3.3	0.06
Electromagnet	–	5	0.67

Table 1. Summary of component voltage and current draw

As such, our design requires two step-down voltage regulators to power the microcontroller and relay; the motor drivers do not require any voltage step-down since the input from our AC/DC adapter is exactly 12V. For our design, we selected the LM2596 Switching Step-Down Voltage Regulator manufactured by Texas Instruments. We are using the 5.0V and 3.3V Fixed Output versions. The LM2596 was chosen because it is simple, requiring only four external components, and highly efficient, over 70%. In addition, this component accommodates a large range of input voltages, up to 40V, and up to 3A output load current.

III. X-Y Cartesian Movement

Cartesian movement of the electromagnet is handled by an H-Bot system beneath the playing surface much like the one shown below.

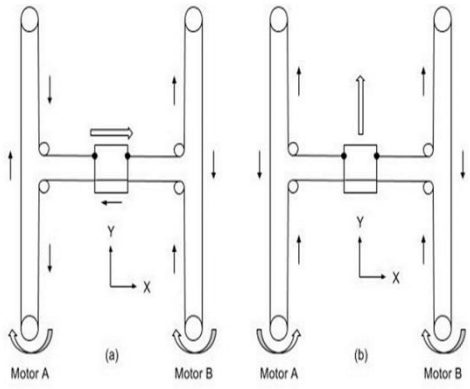


Fig.2. H-bot Diagram

Its kinematic and inverse kinematic equations of motion are as follows:

$$\mathbf{\Delta X} = (\mathbf{\Delta A} + \mathbf{\Delta B})$$

$$\mathbf{\Delta Y} = (\mathbf{\Delta A} - \mathbf{\Delta B})$$

$$\mathbf{\Delta A} = \mathbf{\Delta X} + \mathbf{\Delta Y}$$

$$\mathbf{\Delta B} = \mathbf{\Delta X} - \mathbf{\Delta Y}$$

In the equations above $\mathbf{\Delta X}$ is the displacement along the x-axis, $\mathbf{\Delta Y}$ is the displacement along the y-axis, $\mathbf{\Delta A}$ is the

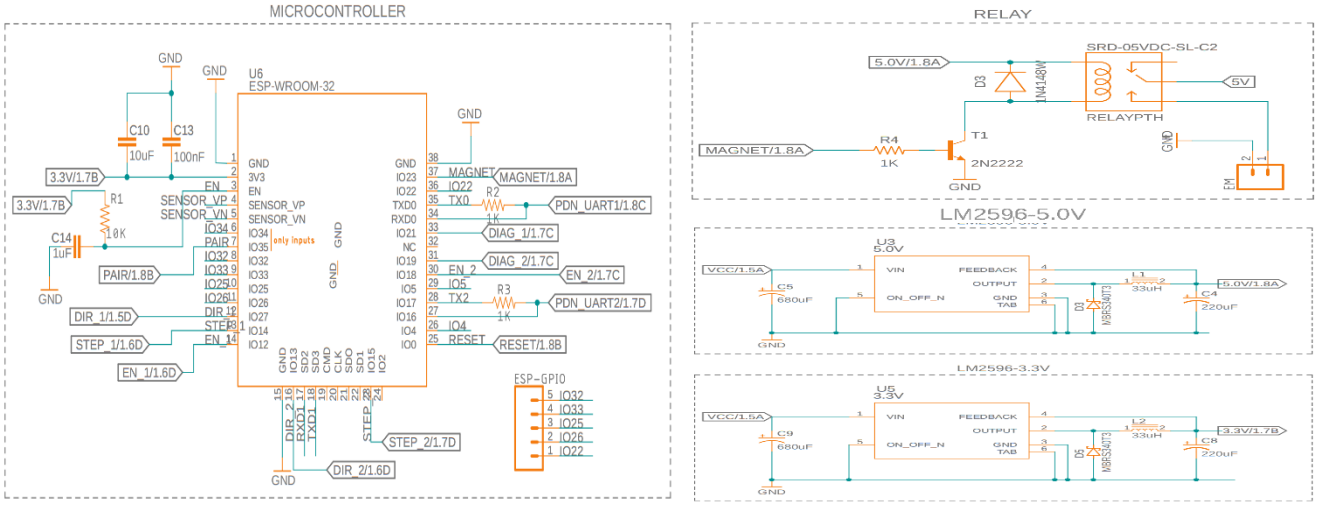


Fig. 3. Custom PCB Schematics – Microcontroller, Voltage Regulators, Relay

arclength displaced by the pully when rotating motor **A**, and **ΔB** is the arclength displaced by the pully when rotating motor **B.**

IV. PCB Design

Autodesk Fusion 360 was used to design the schematics and custom printed circuit board for this project. The goal of the PCB design was to incorporate all of our major systems into a single custom board. This entailed designing all of the necessary schematics as well as generating and organizing the board layout. Due to the COVID-19 pandemic affecting supply chains, we were forced to modify our original PCB design as a result of the TMC2209 motor drivers going out of stock. Consequently, we will instead be slotting in our TMC2209 development boards directly into our custom board.

A. Schematics

The goal for the custom printed circuit board was to design it such that it could interface with all the major system components discussed in Section II. This involved designing the necessary circuitry for each component using the schematic design functionality of Fusion 360. Fig. 3 depicts the schematics designed for the Microcontroller, LM2596 Voltage Regulators, and Mechanical Relay.

The microcontroller must interface directly with the two TMC2209 motor drivers and the relay by sending and receiving signals. Therefore, there is very little external circuitry needed to operate. As per manufacturer's guidelines, we added capacitors to the power supply to avoid potential power rail collapse as well as adding an RC delay circuit to the EN pin to ensure power supply to the ESP32 during power up. Additionally, since we will be using the two UART connections on the microcontroller for the motor drivers, it is necessary for us to connect a 1KΩ resistor to both TXD0 and TXD2.

The voltage regulator schematics consist of four external components and the values of these components are selected based on the operating conditions. Based on the datasheet information, it was determined that the inductor value must be 33uH given our input voltage and load current for both regulator circuits. For the output and input capacitor selection, we determined a 220uF and 680uF low ESR electrolytic capacitor would fit our specifications, respectively. Both capacitors needed to have a voltage rating of 1.5 times greater than the output/input voltage therefore, we selected input capacitors with a rating of 25V and output capacitors with a rating of 10V. The datasheet also provided us with a table of catch diodes to use which included the MBRS340T3 being used in our design. [2]

The SRD-05VDC-SL-C is an electromagnetic relay with 5 pins: 2 for the coil, middle pin for common, and the last two are for normally open and normally closed. This relay has a nominal coil voltage of 5.0V and a nominal current of 71.4mA. The ESP32 will send a signal to toggle the relay switch which will then turn the electromagnet on and off. As discussed earlier, we must use the 2N2222 transistor to amplify the current going to the relay coil from the ESP32. We have also connected a 1N4148 flyback diode across the two relay coils to protect against large voltage spikes.

B. Board Design

The final board layout is depicted in Fig. 4 below. Our board is 2 layers and measures 107mm x 76mm x 1.6mm which is just small enough to not interfere with the robot. Each major circuit is outlined and labeled on the board making it much easier to identify and test any potential

problems. The design was created such that it would be convenient to connect the external components. The motors connect on opposite sides of the board to reflect the location of the motors relative to the board, similarly with the electromagnet connection.

We are using JST connectors for the motors and the USB serial connection to ensure a more solid connection between the pins. This is helpful, especially for the motors as a better connection means our motors will run much smoother and ultimately result in more precise piece movements on the board. We are soldering the electromagnet wires directly into the board to ensure a solid connection.

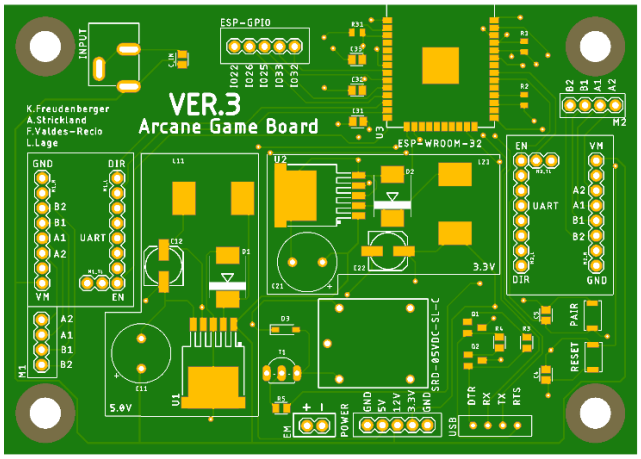


Fig. 4. Custom Printed Circuit Board Design

V. Firmware

The robot functions using firmware developed by our team to establish and maintain a WebSocket connection to the web-app, maintain the game state of the board, and operate the robot according to the moves a player makes. Each of these function groups were loaded into a corresponding controller class to be referenced by the core program, which contained high-level logic control such as scheduling and passing Queues from the Game Controller to Robot Control.

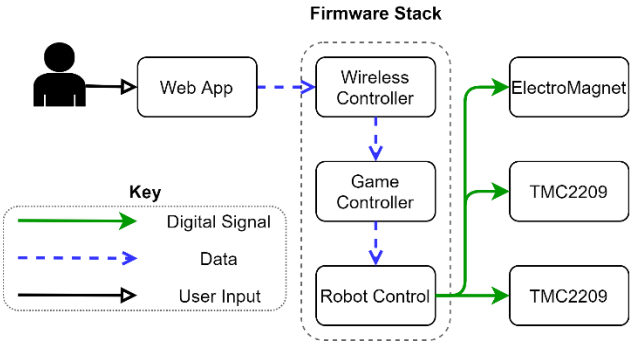


Fig. 5. Firmware Stack Diagram

A. Controller Class Breakdown

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.

B. Key Methods and Functions

WirelessController Methods	
Method Name	*Description*
getMove(doc)	Takes a deserialized JSON file and loads it into a struct containing only unsigned integers and a char – saves space in the stack during runtime and avoids memory leaks.
connectWifi()	Activates the ESP32s WiFi module and connects to the provided network with hardcoded credentials.
socketConnect()	Connects to our server's webSocket – will automatically connect to WiFi if needed.

GameController Methods	
Method Name	*Description*
retirePieceAt(pos)	Identifies the Piece at the given position and queues a path to the graveyard.
movePieceAt(pos, end)	Identifies the Piece at the given position and queues a path to be executed
xyToMotors(dx, dy, magnet)	Creates an individual Move instance which can be fed into RobotControl.
transpose(pos)	Determines the optimal direction to move a Piece such that it does not interfere with other Pieces.

RobotControl Methods	
Method Name	*Description*
queueMoves(Queue)	Takes a Queue of movements and adds it to itself – the original queue is deallocated.
stepMotors()	Sends a 'step' signal to the TMC2209 Stepper Motor Drivers while adhering to step scheduling.
loadMove()	Pops a Move off of the Queue, loads the direction and magnitude of the motor vectors, and de-allocates the memory of the popped off Move.
disable/enableMagnet()	Activates or deactivates the magnet – when the magnet is activates a small delay is inserted.

C. Development and Testing

Most of the embedded software authored by the team focused on the physical and internal layers of the program, as opposed to the communication layer in WirelessController. Working with the constraints of C++, the Arduino Framework, and Embedded realities such as watchdog timers and relatively primitive error detection and recovery were all hazards.

All of the firmware authored by our team was written in Microsoft Visual Studio Code utilizing PlatformIO – an extension which allows for embedded system development and testing. A large part of this decision centers around the ease-of-use regarding library management and unit testing, as well as the team's previous experience and familiarity.

We followed a test-driven-development (TDD) cycle over the course of development. When a class was implemented, any feature more complex than a simple return or set statement had a corresponding test procedure designed alongside it in a separate test stimulus file for that class, file, or functional group. These tests use Unity for their framework. Not only is Unity native to PlatformIO – it allows for native testing on a connected ESP32. This ensured that our tests would not face porting issues. Unity also supports successive tests separated by stimulus files, meaning that testing the final PCB designs with the full battery of tests required only a single function call.

D. Implementation

Our team needed to develop a method of effectively storing Piece information for the GameController to be able to effectively plan moves and retire pieces correctly. Knights – with their unique ability to hop pieces – were of particular concern to us. In order for the board to be truly hands-free they had to skirt around other pieces on the board with enough precision to not attract the magnet of the other pieces. Capturing and the subsequent retiring of pieces also posed a challenge; we wanted the pieces to be stored in formation. These goals required 3 data points:

- Piece Faction [Black or White]
- Piece Type [King, Queen, Bishop, Knight Rook, or Pawn]
- Starting Column [A-H]

While a class could be created containing all of these separately, we wanted to achieve as light-weight of a solution as possible. Using enumeration and bit-wise masking we devised an ID system that contains all of the above data in a single 8-bit data point (in our code an unsigned 8-bit integer.) The 0^{th} bit signals the faction – AKA color – of the piece. Bits 1 through 3 signal the Piece's Type (King, Queen, etc.) All of the remaining 4 bits are used to specify the starting column as an unsigned integer. Allowing functions to move and retire pieces by specifying their positions, looking up the ID, and executing

the appropriate actions. This includes transposition for both Knights and retirement – as shifting the pieces to the vertices of the board removed the requirement for computationally complex path planning.

Chess moves also required chained actions by the robot. These are classic examples of a First-In-First-Out queue structure – so we implemented a wrapper class around the standard C++ queue. In addition to supporting all of the C++11 functions, our Queues are capable of enqueueing other Queues and other small streamlining actions. These functions – when paired with the dynamic sizing and simple structure – allowed for computationally efficient, straightforward, and flexible methods to be written for the GameController class.

Transposition was one of the last major hurdles for piece pathing. We define transposition here as the proper movement of a piece from the center of a chess square to the vertices of the board – ensuring that pieces do not collide so long as they only move in the X or Y direction. The code treats the direction of transposition as a function of overall movement. Because we assume that moves received by the web-application are valid, no error checking is required.

$$dX_T = \frac{dX}{2 * |dX|}$$
$$dY_T = \frac{dY}{2 * |dY|}$$

While the above was sufficient for most moves, it does assume that dX and dY are not 0. This meant that it was only valid for moving knights. For retiring pieces, we instead base it on the quadrant the piece is currently in – moving towards the center of the board to ensure we never collide with the edges of the board.

During all phases of implementation, we strove to maintain proper segmentation of the code. If data was passed to another function it was always done either as a struct/class defined by our team, or as a c++14 standard object. This ensures that any library swapping can be done quickly and at a local level, rather than having to modify the entire program.

VI. Software

While the software design has taken several pivots throughout the semester, a majority of the final design matches the original plan in terms of tools used and overall design layout. The web application that serves as the main interface between the user and the chessboard, consists of two main parts, a front-end and a back-end. The full stack web application can be accessed in several different ways. Some of these ways include through desktop, tablet, or mobile browser. The web application can also be downloaded directly to a users cellphone and ran locally like a native application, due to its progressive web application properties.

Fig. 6 shows the final design for the entire web application, including both the front-end and the back-end.

As we can seem the front end design can be accessed through various medians, such as mobile or desktop. The data flow is then split into two directions. We store some data locally on the users device to using client-side caching, in order to quickly enable local features without the need of a server call. We also send data regularly to the bidirectional WebSocket in order to bridge the connection between the chessboard and the application.

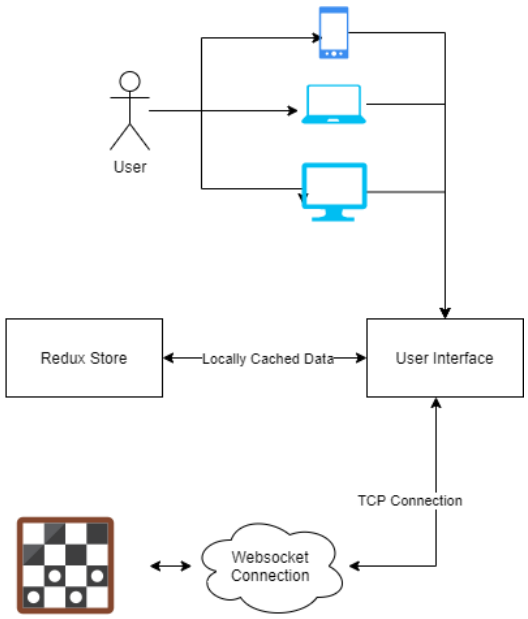


Fig. 6. Web Application Design

A. Front-end

The front-end of the web application has been extensively developed to handle logic, caching, dataflow, and component rendering. The front-end contains a wide array of standard features found in most applications, including routing, user validation, and responsive design. We chose a React framework which allowed us to develop a component-based application. Using the component system that React focuses on, our software team was able to develop reusable JavaScript components for rapid prototyping. The React framework also enabled us to use other React compatible tools, such as Redux, WebSockets, and the Material UI CSS library.

Using Redux, we were able to cache data locally to the device of the user, in order to quickly access and render essential data for the game's history feature. Since the data needed for this feature is relatively small, we use Redux in this instance in order to bypass the need for an external database. Doing this allows us to cut down on hosting fees, reduces latency within the application, and simplifies the code.

Using React also allowed us to utilize the extensive Material UI package. Using this package for the skeleton of the application provided use the tools necessary to both rapidly prototype the application, while still allowing some flexibility in terms of appearance. Material UI has several premade components that we can "plug and play", while still opening those components to receive a fair amount of customization, in order to best fit the needs of the app. The greatest benefit that we experienced from using React with Material-UI was the ability to quickly develop a responsive design all from one code base. This means that we could offer the application across several platforms, including mobile, tablet, and desktop, while only developing one central codebase. This is achieved by defining several breakpoints to detect which device the user is developing on, and only showing the proper components associated with their device of choice. Another direct benefit of using these two tools together, was the overlap that is found between the mobile browser version of the application, versus the native mobile version of the application. Because React allows us to convert our web application into a progressive web application, we can develop both the native and browser versions of the application from one central codebase. This played a crucial role in the development process as we would not have had enough time to develop all of the different platforms without it.

The final most critical tool used by the front-end was the client-side WebSocket connection. Alongside the front-end, our team hosts a back-end in order to open a data stream between both the chessboard, and other players logged into the web application. Using the client side WebSocket allows us to take in data without needing to make an API call. This was crucial for two essential features; sending moves out to the chessboard, and receiving moves from other players playing the same game. Our application allows users to log in and play each other over the internet. In order to provide the real-time experience without any delays while two players are playing, use of the WebSocket was crucial. Using the WebSocket also allowed us to eliminate polling on the board side of things, which cleared up memory for other processes as well as reduced the overall power demand. Overall, the bidirectional nature of the WebSocket was crucial for the success of our application.

B. Back-end

Our application used the WebSockets on a Node.js server in order to enable a data stream between clients. We also utilized Express.js after we deployed the application in order to run as the engine for the Node.js server. Both Node.js and Express.js are incredibly lightweight and quick back-end frameworks, and worked well for our server.

For our specific case, the back-end served as a middle ground between our application's clients. Both the two player gameplay, as well as the board interaction relies on our back-end. In the final iteration of the application, we were able to host the back-end in order for anyone around the world to be able to establish a connection, as opposed to only those on local wifi.

C. Gameplay and Final Features

Our application grew significantly since it was first conceptualized with our Adobe Xd prototype. Originally, we decide on a desktop only application which would host a local chess game that users can play on the same device. After several iterations, we finalized an app that allows gameplay across two separate devices, anywhere in the world, which is hosted online. Our application also includes a stretch goal of "Player vs AI" feature that utilizes the Stockfish AI in order to provide players with a computer of average difficulty to play against. In addition to these two play styles, we also offer a custom "Senior Design Chess Game Experience", with custom pieces. Finally, our largest feature that went beyond the original scope of the project was the responsive design of the front end. Being a progressive web application with responsive design, the application can be played on a computer, mobile browser, tablet, or downloaded and played locally on either a tablet or mobile device. This feature was added with the idea of increasing accessibility to our application to a large variety of users. Overall, the software team was able to meet and exceed all of the originally listed goals.

VII. HARDWARE

A. Design Considerations

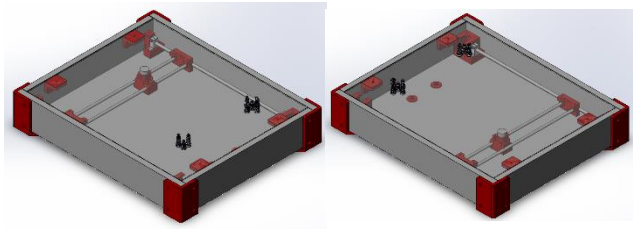
The majority of our purchased hardware components are Sourced from commonly used 3D printer parts. We chose to use these parts because of their wide availability and low cost due to their mass production. Our team is self-funded, so the significant cost savings of buying individual components rather than purchasing a kit drove us to design and build our own Cartesian motion system. Off the shelf components such as the linear rails, pulleys, NEMA 17 stepper motors, etc, were model and loaded into SolidWorks so that supporting fixtures could be designed

to complete the build of our H-Bot system. These components all use metric fasteners, so the custom components were designed to also use metric fasteners to cut down on the complexity of future assembly and disassembly. In fact, over 90 percent of the arcane game board is designed to be held together using M5 socket head cap screws.

B. Specifications

- **Exterior Dimensions:** 27 in (685.8mm) by 23 in (584.2mm) by 4.5 in (114.3mm)
- **Traversable Area:** 16.56 in (402.38 mm) by 13.62 in (346 mm)
- **Playing Board Weight:** 15 lbs (6.8 kg)
- **Playing Square dimensions:** 1.44 in (36.58 mm) by 1.44 in (36.58 mm)
- **Chess Piece Dimensions:** 0.59 in diameter (15 mm) by up to 1.58 in (40mm) tall
- **Movement Speed:** 1.12 in/s (28.41mm/s)

C. CAD



Above are two images of a partially assembled version of our CAD model of the arcane game board. The walls and bottom are cut sections of 3/4th in thick unfinished spruce pine fir Board. This was wood was selected for its light weight yet sturdy properties so that we can stay within our weight requirement of 20 lbs while not compromising overall strength. Metal inserts were drilled into the wood so that M5 bolts could be repeatedly screwed in and out of our boards without compromising any tapped threads.

ACKNOWLEDGMENTS

The authors wish to acknowledge the assistance and support of Dr. Samuel Richie and the University of Central Florida ECE Department.

REFERENCES

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

[2] Texas Instruments , "LM2596 SIMPLE SWITCHER® 4.5V to 40V, 3A Low Component Count Step-Down Regulator," 5 April 2021. [Online]. Available: https://www.ti.com/product/LM2596. [Accessed 14 July 2021].

ENGINEERS

Lucas Lage is a senior at the University of Central Florida and will be graduating with a Bachelor of Science in Computer Engineering in August 2021. After 3 years of Full-stack development experience, Lucas will be moving out to Salt Lake City, Utah, where he will be designing and overseeing application development as a Full-stack application architect.

Fernando Valdes-Recio is a senior at the University of Central Florida and will be graduating with a Bachelor of Science in Computer Engineering in August 2021. After graduation He hopes to take what he learned about engineering and robotics from schooling and internships and apply them in an automation geared position, ideally at Lockheed Martin.

Kayla Freudenberger is a senior at the University of Central Florida and will be graduating with a Bachelor of Science in Electrical Engineering in August 2021. After graduation, she plans to pursue a career in Forensic Engineering.

J. Anton Strickland is a senior at the University of Central Florida and will be graduating with a Bachelor of Science in Computer Engineering in August 2021. After graduation he plans to pursue a career in embedded system software in Boston, MA.