Sudo Chess

Authors: Danié Alvarado, Brandon Dubner, Tony Padilla — Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A mechanical chess board capable of replicating the real-life experience of playing versus another player by moving the pieces of one color to match the moves of an opponent playing the game remotely. The other existing product uses a press-based approach to communicate that a piece has been placed; we are trying to improve on that by making the detection seamless, so it will feel exactly like a normal chess board.**

***Index Terms*— Device-to-Computer, Electromagnetism, Linear Actuators, Node.js, AWS Deployment, stepper motor driver, belt-driven, gantry plate**

I. Introduction

In recent months, a global pandemic has locked us in our homes, unable to have in-person interactions with the people we care about. We hope to provide an experience of playing a game with friends and family that's more immersive than through a screen.

To this end, our project design is a Chess board that simulates real-life play. It will look just like a regular board on the top, but underneath will lie a mechanism that can move pieces smoothly and without disruption (no collision with already-placed pieces) across the board, as well as the technology for communication with a seperate internet-enabled device. These additions let a player use the board and play with an opponent connected remotely through the internet while the board physically reflects the opponent's moves.

Our overarching goal is to have our board's play experience as close to a real-life experience as possible. This is why we chose to add the challenge of moving the pieces automatically, making it feel as if it's just another person moving them. More specifically, our goal is to be able track a game state and move the opponent's pieces accordingly with the automatic movement taking a maximum of 5 seconds. This goal also led to our design choice of detecting pieces through simple placement — no need for you to press on the board or press any other buttons, just pick and place as you would in a real game and the board will know your intention.

II. Design Requirements

A. Software Requirements

The first design requirement for the software portion of our project is that a user will have the ability to connect the physical board to their device and establish communication between them. This will demonstrate the ability to provide the board information to the webapp and commands from the webapp to the board. This is critical for our project, since it will serve as the interface between the software portion of the project and the rest of the project.

The next design requirement for the software portion is the ability to connect with another specific user and play a game of chess against them. The game should be able to be created regardless of the number of players that have a Sudo Board, and the users should be able to send communication within 500 ms. The last software design requirement is for an intuitive user interface. Users should be able to start games and send their moves easily, and the website should reflect these operations quickly.

For the first two design requirements for the software part of the project, we can verify that they are met by using unit tests and manual tests of the system. The last requirement will be more subjective, and will have to be verified manually.

B. Board Requirements

The overarching purpose of the board is to be able to detect the pieces on the board and report it to the controller. Following from this, the first design requirement of the board is to be able to be able to determine, for all 64 squares on the board, whether there is a piece on the square and the type of piece and color of piece with 100% accuracy. The requirement for complete accuracy is due to the way that moves in a chess game occur: if a piece gets moved and a separate piece gets detected incorrectly, it will appear as if multiple moves have been made simultaneously, which cannot happen in a game of chess.

Following this, the next design requirement for the board is the capability of detecting all of the piece locations within 500 ms. This requirement is so the board has enough time to communicate this to the website and the changes happen nearly instantaneously. Previously we had specified this to be 100ms, but realized it was too arbitrary. Half a second is the limit at where a user would feel it takes too slow, and as such the requirement was loosened.

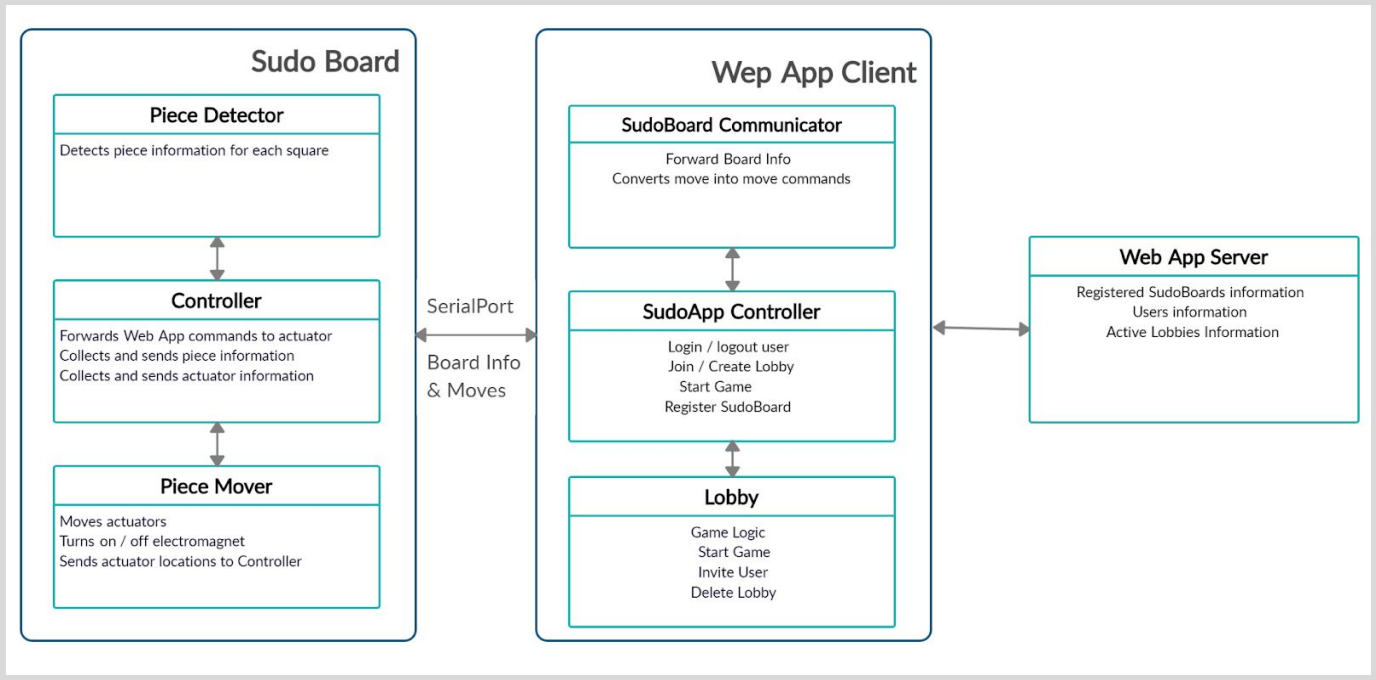

Fig. 1. Sudo Chess System Architecture

The next design requirement for the board is that at most 1 mA of current flows through any of the paths to ground in the piece detection circuit. This is because the circuits will be completed by pieces placed on them, but if someone was to complete the circuit with their finger, we do not want them to get harmed. 10 mA of current is the threshold for harmful amounts of current, but we want to be safe and set the requirement for an order of magnitude less than this threshold.

As for the pieces, the only requirement is that they be less than half the side length of a square of the board, such that the electromagnet is able to move pieces between occupied squares. The length of the acquired actuators resulted in squares that were 40.25mm in side length, so the requirement was that no piece exceed 20.125mm in diameter.

Then we have a requirement regarding the electromagnet: the distance between the electromagnet and the magnetic material inside the piece needed to be no thicker than ¼ of an inch for the magnet to be able to pull the piece.

Yet another requirement is that whatever magnetic material is put inside of the pieces be centered, otherwise the pieces would not remain in the center of each square upon being moved by the electromagnet and run the risk of being budged out of detection position by future movements from other pieces or worse, not being placed in the proper position to be detected by the contact plates.

C. Movement Requirements

For the moves of the game to be replicated automatically onto the physical board, there must be components dedicated to the movement of the pieces. The first design requirement for the Sudo Chess system is to be able to complete any move within 5 seconds. This included castling, capturing pieces, but in reality only applied to any other legal move. 5 seconds was chosen since we want to be within the average attention span of humans, currently estimated to be 8 seconds, while still providing enough time to move the pieces precisely.

To test that each movement will take less than 5 seconds, we use the equation

$$1.\ v = \omega r$$

where ω is angular velocity. We convert RPM to angular velocity using the equation

$$2.\ \omega = \frac{RPM}{60s/minute} \times 2\pi\ rad/rev$$

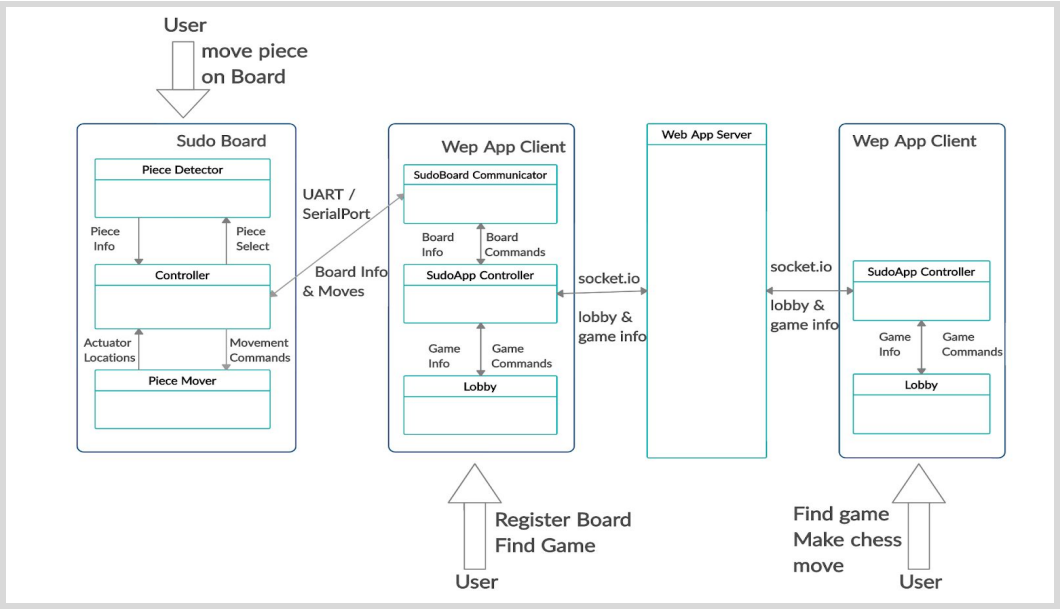

Fig. 2. Sudo Chess System Interaction Diagram

We use 180 RPM for moving the linear actuators to a chess piece's location on the board, and 90 RPM to move the piece, using the linear actuators, to its destination. This results in an angular velocity of 18.85 rad/s and 9.425 rad/s. The radius we used in equation 1 refers to the radius of whatever is making a revolution. Since we are using belt-driven linear actuators, the radius to compute is the radius of the wheels that hold the belt which is 3cm or .03mm. Using equation 1, for movement of the actuators to the piece, we have a velocity of 0.5655 m/s or 565.5 mm/s. For movement of the piece to its destination we have a velocity of 0.28275 m/s or 282.75 mm/s. Because the maximum travel distance on the linear actuators is 402.5mm, we are capable of moving the linear actuators to the piece that needs to be moved in under 1s, and also move that piece to its destination in under 1s, though the movement of a piece to its destination varies. In any case, we are able to achieve automatic piece movement in under 5s.

On the topic of precise movement, the next requirement of piece movement is that only the pieces involved in a move should be changed. There should be no collisions between the moving piece and the other pieces and no other pieces should need to be moved. The largest piece, the king, is 19mm in diameter, d_k. Each square on the board has a length, l_s, of 40.25mm. Since the pieces are placed on the center of the board, this gives a minimum gap of 21.25mm for pieces to travel in between others without collisions, which ensures 0 collisions. This minimum gap, g_m, was calculated using the equation

$$3.\ g_m = 2 * (\ \tfrac{1}{2} l_s\ -\ \tfrac{1}{2}\ d_k)$$

III. Architecture and/or Principle of Operation

The overall design of our system is shown in the system architecture in figure 1. In general, we split our design into 3 conceptual areas: Piece Detection, Piece Movement, and the webapp. The components in our design that aren't directly related to these areas typically either connect those areas together or implement a sub-task within the area.

A. The Sudo Board

Inside the Sudo Board we have the Piece Detector, the Piece Mover, and the Controller that collects information from the submodules. These modules all interact and communicate with the other modules through the Sudo Board Communicator.

The overall objective of the piece movement unit is to be able to move pieces to arbitrary squares without disrupting the other pieces on the board. The piece movement unit consists of two V-Slot® NEMA 17 Linear Actuators [1], the various parts needed to get the actuators to move, and an electromagnet. The linear actuators will work in tandem with each other, moving the electromagnet to locations across the board in either direction. The locations of the linear actuators will be reported to the Sudo Board controller, which will give movement commands in response.

The piece detection unit consists of the physical chess board, the chess pieces, and an underlying detection circuit. The circuit is partitioned into 4 different sampling circuits, that each detect piece information for 16 different squares. After piece information is collected, it is then sent to the Sudo Board controller, which in turn communicates with the web application.

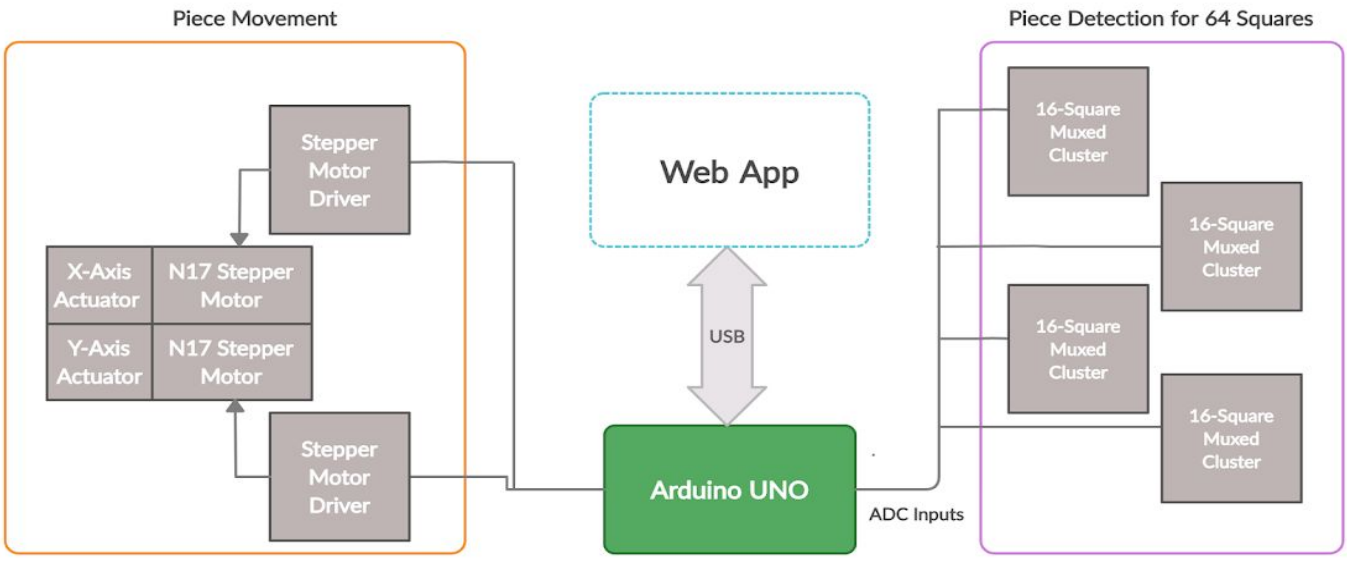

Fig. 3. Sudo Chess Physical System Block Diagram

The design remained mostly the same with exceptions that will be explained in section IV. A minor change in the design included the mounting of the linear actuators. They were to be mounted perpendicularly using screws and L-shaped mounting brackets. Instead, high quality gorilla glue was used for the mounting of one linear actuator above the other. The bottom part of the track belonging to the linear actuator allowing movement along the x-axis was glued to the gantry plate of the linear actuator allowing movement along the y-axis. This was a minor design change from the previous design report.

B. The Web Application

First, we separate the web application into three distinct components: the client, the server, and a communicator for the Sudo Board which connects a Sudo Board. The client is the part of the application that the user directly interacts with, as in logging into their account. The client will then send requests to the server, for example, to create a lobby which the server will fulfill and tell the client as such. The client primarily handles local Sudo Boards and local games whereas the server primarily handles communication between one client and another. Finally, there is a separate process that gets output from the Sudo Board and messages it to the server, which then forwards the update to the client. This process will also send client messages to the Sudo Board, after being forwarded from the server.

The web application is intentionally decoupled from the Sudo Board in such a way that you do not need a Sudo Board in order to use the application. Because of this, the Sudo Board Communicator has the primary responsibility of communicating with the Sudo Board and translating the output to the inner game logic of the web application. Furthermore, the client has no knowledge of the communicator, so the server must mediate between the communicator and the web client.

C. System Interaction

Depicted in the system interaction diagram above (figure 2) is a chess game between one user who is using a Sudo Board and another user who is only using the web application. The two main ways a user can interact with the Sudo Chess system are by creating / finding a game and by making moves in a created game. Both Sudo Board users and web only users will be able to find a game by interacting with the web application client. Sudo Board users will be able to play moves by moving the pieces on their physical Sudo Board whereas web only users will move pieces via the web application.

Under the covers when a piece is moved on the Sudo Board, the system response starts when the piece detection unit reports the updated state of the board. After the Controller gets the information from the Piece Detector, it forwards the information to the web application through the SudoBoard Communicator. The server will check that the move was a legal move and, if it is, it will send the move to each of the web clients, which will then update the board graphic on the screen. In the opposite scenario, where the web player makes a move using the GUI, the system interaction will be largely the

same, except when the opponent (now the Sudo Board user) receives the move, the Sudo Board Communicator will give the Sudo Board a movement command, which will prompt the movement unit to replicate the move.

IV. General Description with Design Trade Studies

A. Piece Detection

How to perform the piece detection was decided after careful consideration of different elements of the subsystem that will perform this task. We needed something that would be cost-effective and easy to implement, as none of us had strong experience with circuits or signals. At first, we thought of using RFID to detect each piece, a simple implementation as this technology is widely available. However, the cost of having so many detectors (one for each square) would drive us way over budget. Then we thought of contact plates that showed on the board, where the pieces act as a simple switch with a resistance on them won among our other possible implementations. These can be made barely visible and made our detection subsystem a simple DC circuit.

Next, for this, we would need to measure 64 different voltages at all times, one for each square of the board. We took into consideration that it only needed to feel, to the user, as if we were detecting all pieces at the same time. We learned that microcontrollers can typically read their Analog-to-Digital Converter inputs in the order of microseconds, so instead of needing 64 separate ADCs, we could multiplex the voltages being read from each square's circuit and read them one at a time fast enough for it to seem simultaneous. The final circuit design for this results in 16 parallel paths from the 5V power source to Ground, each with a Load Resistor and a switch with a resistance, with the node between them being measured by the microcontroller. The "switch" here refers to the pieces that, through metal plate contact, connect the switch with the resistance corresponding to a different piece type and color combination. The voltage between the load resistor and the piece's resistor will be different for each type of piece and that is how the microcontroller will be able to detect which piece it is.

Following are the details of the implementation and the design choices that led to them.

a. Resistance Values

The main consideration for the values of the resistances of each piece was that the board be user-safe. Since the user could practically connect the circuit with a finger, it is a requirement that the current going through any one path not exceed 1mA. A load resistor goes in parallel with the resistor in a piece for a total 5V drop, and the following values meet this constraint

Piece Resistances	
Load Resistance	2.4KΩ
White Pawn	2.2KΩ
White Rook	2.4KΩ
White Knight	2.8KΩ
White Bishop	3.3KΩ
White King	3.6KΩ
White Queen	3.9KΩ
Black Pawn	4.2KΩ
Black Rook	4.6KΩ
Black Knight	5.0KΩ
Black Bishop	5.4KΩ
Black Knight	5.9KΩ
Black Queen	6.6KΩ

Fig. 4. Piece Resistances

A secondary consideration was resistor availability, and so we chose values that were easily attainable from the ECE labs.

b. Muxes

Next we had to choose the multiplexers that would let the arduino poll all 64 squares with only 4 ADC inputs. We picked the MAX306CPI+[10] , an analog 1:16 multiplexer. This multiplexer has a transition time of 250ns, which more than met our requirements.

c. Power Source

The power source chosen for the subsystem was 12V DC, as it could power the Arduino as well as the MAX306CPI+ mux. It was also chosen because this is the voltage needed to drive the motors for the piece movement, and so we would not need more than one different power source upon integration.

d. Contact Plates

Initially, the plan was to craft metal plates to embed on the boards and each piece. When the time came to implement this, the details of it proved to be more complex than initially thought out. There were various inconveniences we faced. Firstly, due to the covid pandemic, Techspark course size was very limited and Danié was not able to secure a spot in the required training to use the metal shop and craft these plates. Placing an order for the plates from Techspark was another option, but it would have put us over budget. We looked then into obtaining the plates pre-made from elsewhere, but it was such a specific item that we could not source it.

Moreover, implementing these on the board would require a plate-sized hole to embed them in, which required more detailed tools that we needed a different Techspark training course to be able to use. Again, due to limited course sizes, this training could not be secured.

Realizing that this was a more daunting task than we had scheduled for, we began to look for alternatives. We arrived at the solution of using copper foil tape, which had the advantage over metal plates in that it was quick to work with, saving us a considerable amount of time. It also only required simple holes to be drilled into the board, which could be easily done with a hand drill that could be used at Techspark without having taken a training course. In addition, copper foil tape is incredibly cheap in comparison.

Of course, this came at a disadvantage. Copper tape is much less durable, but this was a tradeoff we were willing to make because the pros outweighed the cons. We valued being able to complete the project more than getting slowed down by an unnecessarily complicated step.

Finally, one concern was that the copper foil tape would block the electromagnet's magnetic field, preventing it from attaching to a piece. This was not a problem, a magnet was able to attract metal past the tape.

We did pay this price for this, however. Due to its fragile nature, although wires could be soldered to it, it was still very easy to tear off. This resulted in a lot of time being spent on debugging the detection circuit due to detached wires.

e. The Pieces

There were two main options for the pieces: 3D printing and carving out the bottoms of wooden pieces. Each had advantages and disadvantages:

Piece Implementation Pros & Cons	
3D Printed	*Carved wood*
• Exact: easy to measure the amount of space on the inside of the piece. • Time-consuming to design each piece. • Printed out with hole at the bottom already there. • Costly	• Imprecise: difficult to precisely measure how much space to carve out. • Pieces already built. • Arduous work to drill into each piece. • Cheap

Fig. 5. Piece Types Pros & Cons

Ultimately we decided on the wooden pieces because there was not enough advantage to using 3D printed pieces to justify taking a much larger chunk of our budget.

f. The Piece Bottom

There was a short consideration for what should be the bottom of the piece - the lid to close the single resistor that needed to be placed inside. Upon making the switch to copper foil tape, electrical tape was picked because it was thin enough to not hinder the electromagnet and very easy to work with. And being electrical tape, it would not interfere with the circuit.

g. Magnetic Material

To meet the only requirement this part had - that the small magnetic material be in the center of the piece - we quickly found a simple solution: a ferrous nut that the resistor could be inserted in. Because the resistor inside the piece would be soldered onto the bottom, it could be centered, and a nut surrounding the resistor could be glued there to stay put and also be centered.

The nut not being magnetic enough was not an issue, as it was able to be attracted by a fridge magnet from underneath the board, one less powerful than our electromagnet.

h. Board Top

A board top that was thin and could be drilled holes into was all that was needed. Two thin wooden sheets glued together met these requirements; they were ⅛ of an inch thick which met the ¼ of an inch requirement from the electromagnet to the piece.

B. Piece Movement

The technology we would use for piece movement went through a few idea iterations before we arrived at our final idea. At first we thought it would be a good idea to have a system of magnets underneath the board that controlled each piece, but upon conversation with the professors, we realized we might have been not only underestimating how mechanically complex that could be, but also how much it could potentially cost. As none of us were strong in mechanical components, we had to research the price points and complexities of different technologies to finally arrive at the mechanism we're using: two cross-mounted belt-driven linear actuators. We'll be placing these under the board. An electromagnet will be mounted on the gantry plate of the topmost linear actuator which will be used to move the pieces around. The pieces were originally intended to have a magnet at the base, but we found that using a small metallic nut was sufficient for the electromagnet to attract the piece and move it to its intended position.

Other considerations were using a robotic arm to move the pieces and using the same 2D-actuator system, but using a mechanically lifted regular magnet. We decided against the robotic arm because it was riskier, as we would have had to fine tune the precise mechanical movements and we did not have the experience to gauge how easy it would be to implement. And then we picked an electromagnet over a mechanically lifted regular magnet simply because, although we need to do more research to implement the electromagnet, it is less risky in nature.

The linear actuators will be mounted on a stable chassis underneath the board, with an Arduino controlling them. The linear actuators use NEMA 17 Stepper Motors which we intended to drive using an A4988 stepper motor driver coupled with an Arduino via an interface; the AccelStepper library. The AccelStepper library is an Arduino library, but since we opted for a client-side JavaScript approach, we instead used Johnny Five's [9] robust JavaScript Robotics and IOT platform. The Johnny Five platform includes an API for controlling stepper motor drivers such as the A4988 by abstracting away the complexities of microstepping and bipolar driving of motors.

We found the limitations of communication with a server using the arduino platform to be rooted in increased complexity. The Arduino platform has multi-layered libraries that allow an arduino to communicate with a server, and the Johnny Five platform removed these layers, as well as the amount of code needed to decode messages from the server and get the Arduino board to command the A4988 motor driver to drive the motors.

The A4988 motor drivers had their limitations as well. The particular vendor which we obtained the motor drivers from connected pins that were too thick to fit in the conventional breadboards we used. It took a lot of tinkering to figure out that they weren't broken, but simply not inserted into the breadboard correctly. This led to testing the linear actuator's movements limited by physically having to push the motor drivers into the breadboard using a finger. The A4988 motor drivers are extremely delicate. The physical pushing using a finger caused shorts in some of the pins if a finger was touching multiple pins. This led to breaking many of the A4988 motor drivers we had at our disposal. The best workaround to inserting the motor drivers into the breadboard was to simply solder wires to each of the pins and insert the wires into the breadboard. But the problems did not stop there. The A4988 motor drivers had built in support for overheating, and they came with heat sinks that can be mounted on the chip in case of overheating. However, we needed a certain current measurement to drive the motors and which could be adjusted using the A4988 built in potentiometer. The output current could be measured by using a multimeter and getting the current reading across the potentiometer and the V_{mot} ground pin. Touching the V_{mot} ground pin and the V_{mot} pin simultaneously during measurement of the current (by accident) would cause a short resulting in breaking the built in protection for overheating, causing the driver to overheat. This limitation was a quick workaround with steady hands.

C. Web Application

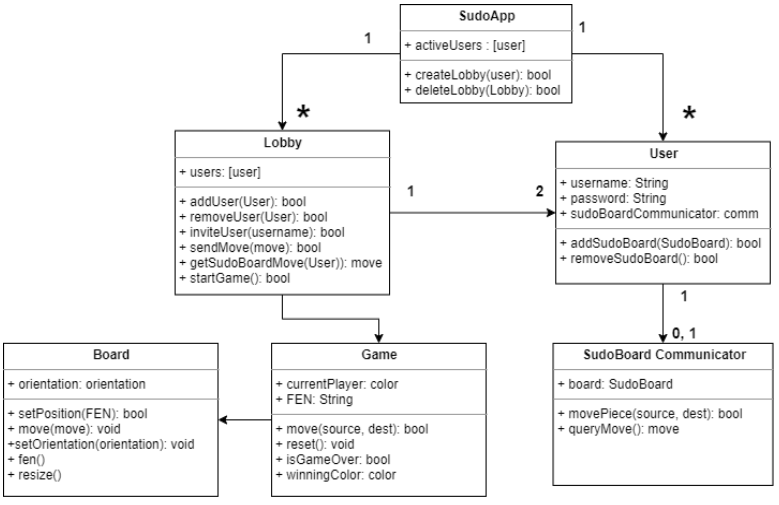

Fig 6. Class Diagram for web application

The main consideration motivating the choice of technology for the webapp is the real-time exchange of data that occurs when a move is sent from one client to another. For this reason, we chose to use Node.js and Express.js as the foundation for the webapp as we knew it had the capability to meet these requirements. An added incentive for using the Node.js framework is its rich amount of libraries available to

supplement our application. For example, we are using chessboard.js to provide the chess board graphics, chess.js gives us game logic and validation, Node Serialport facilitates communication between the web application and the Sudo Board through ZeroMQ, and socket.io provides a nice interface to send and receive data using websockets.

As seen in figure 6, the class diagram for the web application client, the application will have a list of active users and a list of lobbies. Each lobby can have up to 2 users and each lobby has an associated game as well. Every game has a board which visually shows the current state of the game. Every user can register a Sudo Board, and if they do have one registered, they will have a Sudo Board Communicator.

For the communication between the Sudo Board and the web application, we had a few considerations to take into account. Originally, we assumed that we would be able to get access to the serial port of a user's computer directly in the web application client, however due to the way modern web browsers operate, this approach was not feasible. In order to overcome this, we came up with an approach of using a separate process on the user's local machine to send messages to the web application server, which then forwards the information to the user's web application client and the user's opponent's web application client. With this approach, for each new game the server needs to create two different channels to communicate with the local machine communicator: one for the server to send moves from the web application client and one for the server to receive moves from the Sudo Board. Because there are two users per game, this results in four channels per game, which is a meaningful amount of resources for the server to maintain.

The decision of which messaging service to use for our project took several factors into consideration. The primary factor was how much additional resources would be required in order to implement the messaging service. Some services required creating databases in the cloud, some required the creation of local databases, others required using a specific framework to base your application on. ZeroMQ [7] is the messaging service we decided to use, partly because it only required a local redis database, which we could easily spin up using Redis' public Docker image [8]. Additionally, ZeroMQ does well on the other factors we used to come up with the decision: it provides fast delivery time of the messages, minimal downtime, and also has guaranteed eventual delivery of messages sent.

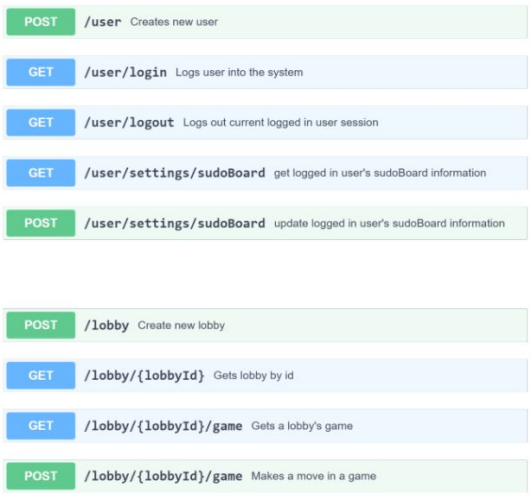

Fig 7. Route Table for web application

III. Project Management

A. Schedule

Our full, detailed schedule at the end of this document in *Fig. 8* is divided into the subsystems with one team member taking care of each, and in addition another section for testing and integration.

It suffered many push backs due to a number of factors. At first, we had team communication issues which happened in between the design phase and implementation phase, setting up poor groundwork for the rest of the project. Secondly, we did not plan tasks to pipeline while waiting for the delivery of parts - mainly because we did not expect parts to take longer than the expected delivery on the website we requested them from, but often they took twice as long and we did plan tasks to do while we waited.

We also did not schedule in logistical or research tasks. These are trivial tasks that seemed simple, but span time and needed to be accounted for. These are what ended up being the blank spaces in the schedule that have not been spent on a specific task, but rather in that miscellaneous preparation for those specific tasks. These ended up pushing the tasks we had planned out towards the end and delaying the project.

B. Team Member Responsibilities

The responsibilities of the team members were split up mainly by subsystem, due to the current state of the world. With a lockdown caused by a pandemic, we as a team were not able to physically meet and so cannot share work on any of the physical components.

Danié took care of the piece detection circuit. This included planning the circuit, then prototyping it and integrating the

detection with the Arduino. After that, they were to plan and assemble the board top and integrate the circuit within it, as well as designing the pieces to work with this board top. Since construction of the board top and pieces is easiest if one has access to campus facilities and they are the only team member on campus this semester, the work was split this way.

Brandon was in charge of the software subsystem of the project. He was responsible for creating the web app, handling communication from the webapp to the arduino controller and vice-versa. After everything had been put together he and Tony worked together remotely to test the integration of the connection between the Arduino microcontroller and the web app. This work was assigned to Brandon because he has experience working with web development and because he lacked the resources to physically work on the other subsystems.

Tony was tasked with constructing the linear actuator system that moves the pieces from underneath the board. This meant putting the actuators together and finding the best way to stack them for 2-dimensional movement, i.e. building a chassis. He is also tasked with finding the electromagnet needed to drag the pieces across the board. Lastly, he needs to program a microcontroller to control both the motors and the electromagnet. After this was done, he and Danié planned to meet in Pittsburgh for integration of the two physical subsystems, but this was not possible due to time constraints and covid complications. He was assigned this work because he has access to enough tools to build a chassis for the linear actuators.

C. Budget

Table 1 is the list of materials that we used or acquired. Many tools we were able to use from Techspark. We used $350.98 out of the budgeted $600.

D. Risk Management

The main risk that we face in our project is when we put our piece detection and piece movement subsystems together. Because we have been working on these individually and separately due to the constraints imposed by the covid pandemic, incompatibilities may arise once we meet and put the two parts together. To manage this risk, we are setting ample time aside for this integration as well as doing the integration itself in Pittsburgh, where we have access to the facilities and tools on CMU campus to make any adjustments to the board that are necessary. In the end, full integration was not possible. When we realized this, we mitigated the risk by focusing on showcasing the integration of separate subsystems, and downscaling our end product.

TABLE I. BUDGET

Component	*Price*	*Notes*
Incurred Expenses		
Breadboard	-	Leftover from another course
Arduino UNO + cables and power supply	-	Kept from 18-220
Resistors	-	Obtained from ECE labs
Hand Drill	-	Techspark
Hot Glue Gun	-	Techspark
Electrical Tape	-	Techspark
Stencil Knife	-	Techspark
Digikey MAX306CPI+ 16:1 analog multiplexer	$14.00	$9 + $5 shipping cost
Wooden Sheets	$8.00	Techspark
DC Power Jack female	$8.49	came in bundle of 12
12v power supply	$12.21	
Wooden Chess pieces	$12.00	set from Amazon
x2 V-Slot® NEMA 17 Linear Actuator Bundle	$171.98	
x2 A4988 stepper motor drives	$19.10	came in bundle of 5
x3 Digikey MAX306CPI+ 16:1 analog multiplexer	$32.50	
Techspark wood costs	$20	

10K pots	$8.18	came in bundle of 5
electromagnet	$21.56	
PCB Breadboards	$7.99	Set from Amazon
Copper Foil Tape	$14.97	from Amazon
Total Expenses	$350.98	
Total Budget	$600.00	
Leftover	$233.54	

IV. Summary

Our final product was incomplete due to time constraints. The limits on the system's performance are largely due to not having a full integration. In section II, it was specified that the system would be able to handle complex moves like castling, and even moving a piece off board after it has been taken. The linear actuator system is able to move pieces in between others; it is demonstrated in code with the movement of the knight. Due to time constraints and some design limitations we were only able to demonstrate the simple movements: the movement of a knight, a diagonal movement, a horizontal movement, and a vertical movement of a chess piece. Given more time, we would find a way to implement more complex movements like pawn promotion and most importantly: taking pieces We also didn't have a chassis for the linear actuators, and this was largely due to not having access to the right materials or the lab space to create a chassis out of wood. With the chess board subsystem, given more time, we could improve performance by aligning each piece to the center of the square it is contained in such that it lies in the center always.

A. Lessons Learned

Although we were unable to fulfil the design we set out to build, there are various lessons we took from this entire process. First is the importance of the initial research. While we did acknowledge that researching was important before we set out to build anything, we still failed to give it the importance it merited.

For example, one variable that determined other details of the implementation was which actuators we would be able to find. Size, price, and the requirements for powering them would affect the other subsystems too. We spent weeks without deciding on one when we should have focused all our energies into researching that as soon as possible.

Probably the most important lesson though was the importance of planning things out to the last possible detail at the beginning. Things that can be figured out at the start should never be relegated to later. We set out into the implementation phase with a very flimsy design, one which had many details yet to figure out, leading to many, many surprises along the way that could have been foreseen.

Lastly, the management skills needed for a large project, in particular when working with a remote team. For some of us it was the first semester-long project, and our time management skills would be put to work. However, the restrictions due to covid were more impactful than foreseen. In addition to the extra discipline needed to meet virtually as opposed to physically, there were so many logistical details that needed to be worked out. Our mistake was not taking the fact that these would be impactful into account. The long wait times for deliveries, the times for shipping tools, the restricted access to campus, and the logistics of planning around a small window of time for integration all had a negative impact on our schedule, an impact that could have been mitigated by taking these inconveniences into account from the start. By working them into the schedule.

References

[1] OpenBuilds Part Store, https://openbuildspartstore.com/v-slot-nema-17-linear-actuator-bundle-belt-driven/
[2] squareoffnow.com
[3] Otka OIDC Middleware npmjs.com/package/@okta/oidc-middleware
[4] Chessboard.js chessboardjs.com
[5] Chess.js github.com/jhlywa/chess.js
[6] Socket.io socket.io
[7] ZeroMQ zeromq.org/languages/nodejs/
[8] Redis hub.docker.com/_/redis
[9] Johnny Five http://johnny-five.io/
[10] MAX306 Datasheet https://datasheets.maximintegrated.com/en/ds/MAX306-MAX307.pdf

Fig 8. Project Schedule