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.**

I. Introduction

In recent months, a global pandemic has locked us in our homes, unable to have those in-person interactions with the people we care about. We hope to, for people who enjoy playing Chess with their friends and family, be able to provide a more similar experience to playing a game with those people that's more than just 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 an internet-enabled separate device. These additions let a player use the board and play with an opponent connected remotely through the internet while the board reflects the opponent's moves physically.

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, only you can't see the hand. More specifically, our goal is to be able track a game state and move the opponent's pieces accordingly. 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 100 ms. This requirement is so the board has enough time to communicate this to the website and the changes happen nearly instantaneously.

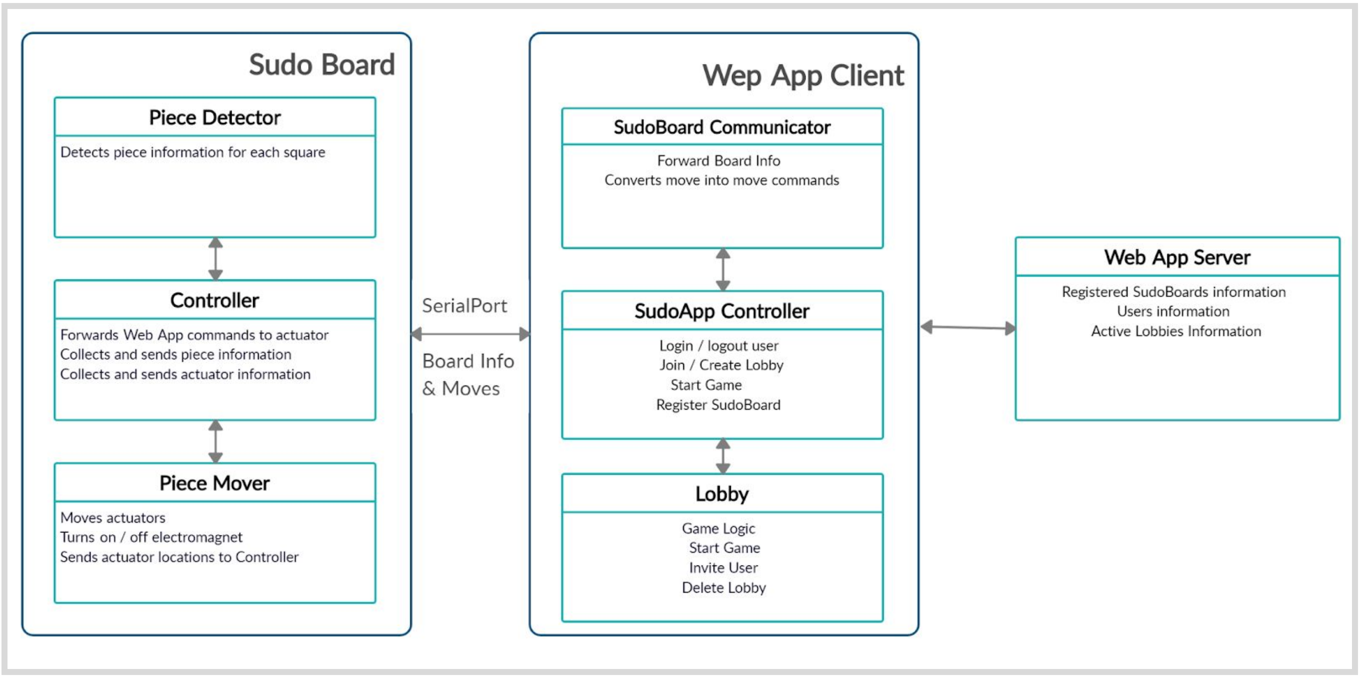

Sudo Board
Piece Detector
Detects piece information for each square
Controller
Forwards Web App commands to actuator
Collects and sends piece information
Collects and sends actuator information
Piece Mover
Moves actuators
Turns on / off electromagnet
Sends actuator locations to Controller
SerialPort
Board Info & Moves
Wep App Client
SudoBoard Communicator
Forward Board Info
Converts move into move commands
SudoApp Controller
Login / logout user
Join / Create Lobby
Start Game
Register SudoBoard
Lobby
Game Logic
Start Game
Invite User
Delete Lobby
Web App Server
Registered SudoBoards information
Users information
Active Lobbies Information

Fig. 1. Sudo Chess System Architecture

The last 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.

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 includes castling, capturing pieces, and 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.

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.

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, 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.

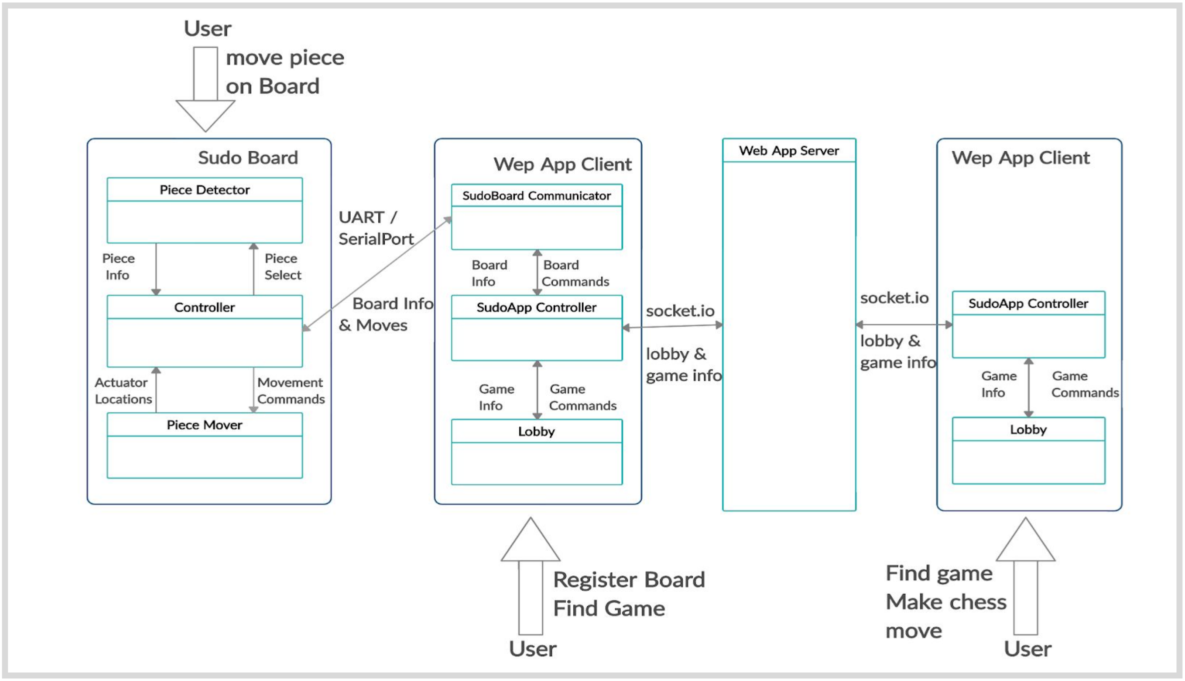


Fig. 2. Sudo Chess System Interaction Diagram

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.

B. *The Web Application*

First, we separate the web application into two distinct components: the client and the server. 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.

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.

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 application will check that the move was a legal move and, if it is, it will first update the board graphic on the screen. Next, the web application will tell the server to forward the move to the user's opponent in the game and then the opponent will be prompted to make a move. 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.

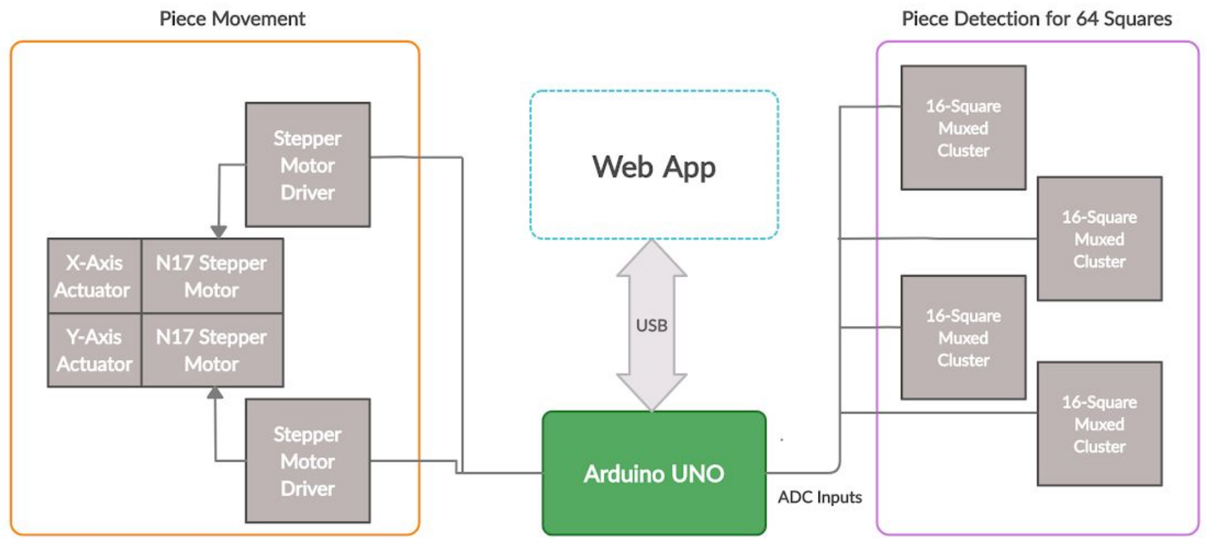


Fig. 3. Sudo Chess Physical System Block Diagram

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. The resistances for each path will be from 5K to 12K Ohms,with values picked for user safety.

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 then using an electromagnet to move the pieces around (which will have a magnet at their base).

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.

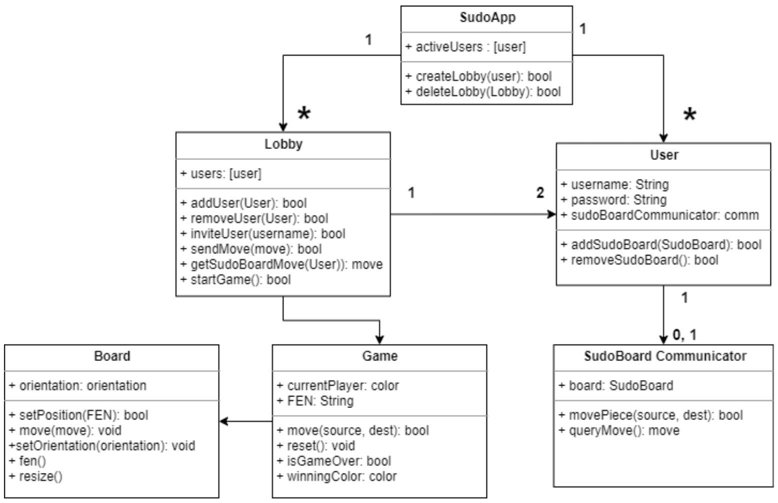
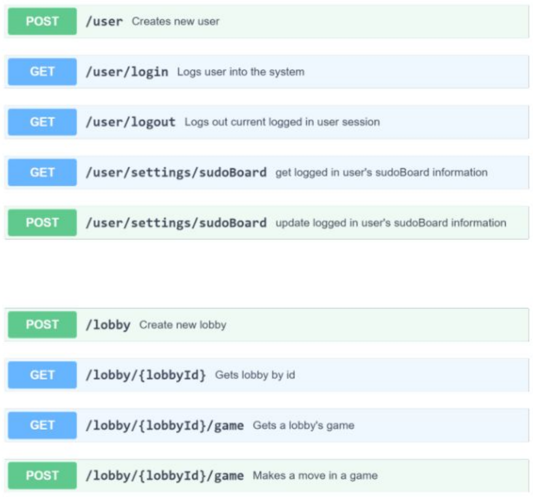


Fig 4. Software Class Diagram for web application



POST `/user` Creates new user

GET `/user/login` Logs user into the system

GET `/user/logout` Logs out current logged in user session

GET `/user/settings/sudoBoard` get logged in user's sudoBoard information

POST `/user/settings/sudoBoard` update logged in user's sudoBoard information

POST `/lobby` Create new lobby

GET `/lobby/{lobbyId}` Gets lobby by id

GET `/lobby/{lobbyId}/game` Gets a lobby's game

POST `/lobby/{lobbyId}/game` Makes a move in a game

Fig 5. Route Table for web application

These 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 will drive with an Arduino using the AccelStepper library via A4988 Stepper motor drivers.

C. *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 UART, and socket.io provides a nice interface to send and receive data using websockets.

As seen in figure 4, 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 SudoBoard Communicator.

For deployment, we will use DynamoDB for the database and deploy to AWS Elastic Beanstalk. This gives us the benefit of deploying to an autoscaling service without us needing to worry about the complex infrastructure.

III. Project Management

A. *Schedule*

Our full, detailed schedule at the end of this document in *Fig. 6* is divided into the subsystems with one team member taking care of each, and in addition another section for testing and integration.

It has suffered some push back of tasks, mainly due to waiting for delivery of parts and not accounting for how busy each member would be during specific times with other classes. However, we had made it with a margin for error, such that even with the set-backs we are still on schedule. The blank spaces in the schedule have not been spent on a specific task, but rather in miscellaneous preparation for those specific tasks, such that once we have all the materials and resources we can quickly do the task listed.

B. *Team Member Responsibilities*

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

Danié is taking care of the piece detection circuit. This includes planning the circuit, then prototyping it and integrating the detection with the Arduino. After that, they are 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 is in charge of the software subsystem of the project. He is responsible for creating the web app, handling communication from the webapp to the arduino controller, and deploying the web app to AWS . After everything has been put together he and Danié will work 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 lacks the resources to physically work on the other subsystems.

Tony is tasked with constructing the linear actuator system that moves the pieces from underneath the board. This means 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 is done, he and Danié will meet in Pittsburgh for integration of the two physical subsystems. 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 have acquired so far along with what we plan to acquire. Expenses for techspark are a generous estimate of the materials we might need to acquire, mainly wood and metal, but also PCBs. We have budgeted only $366.46 out of the $600, giving us plenty of slack to work with.

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.

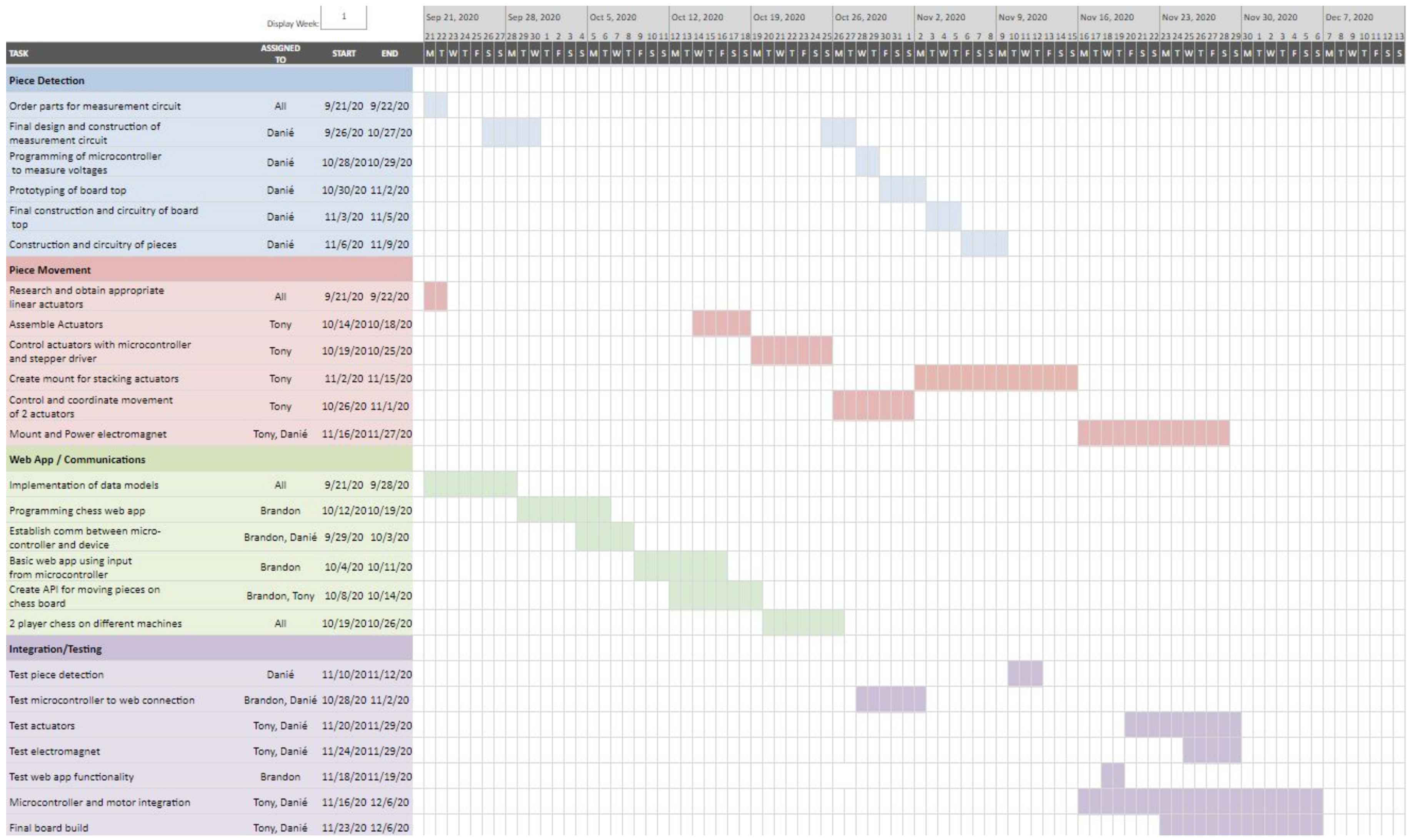
TABLE I. Tentative Budget

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
Digikey MAX306CPI+ 16:1 analog multiplexer	$14.00	$9 + $5 shipping cost
Planned Future Expenses		
Wooden Chess pieces	$12.00	set from Amazon
x2 V-Slot® NEMA 17 Linear Actuator Bundle	$171.98	
NEMA 17 Stepper Motor	$35.98	
x3 Digikey MAX306CPI+ 16:1 analog multiplexer	$32.50	
Techspark estimated costs	$100.00	
Total Planned Expenses	$366.46	
Total Budget	$600.00	
Leftover	$233.54	

References

[1] OpenBuilds Part Store, https://openbuildspartstore.com/v-slot-nema-17-linear-actuator-bundle-belt-driven/

[2] squareoffnow.com



Display Week: 1

TASK	ASSIGNED TO	START	END
Piece Detection			
Order parts for measurement circuit	All	9/21/20	9/22/20
Final design and construction of measurement circuit	Danié	9/26/20	10/27/20
Programming of microcontroller to measure voltages	Danié	10/28/20	10/29/20
Prototyping of board top	Danié	10/30/20	11/2/20
Final construction and circuitry of board top	Danié	11/3/20	11/5/20
Construction and circuitry of pieces	Danié	11/6/20	11/9/20
Piece Movement			
Research and obtain appropriate linear actuators	All	9/21/20	9/22/20
Assemble Actuators	Tony	10/14/20	10/18/20
Control actuators with microcontroller and stepper driver	Tony	10/19/20	10/25/20
Create mount for stacking actuators	Tony	11/2/20	11/15/20
Control and coordinate movement of 2 actuators	Tony	10/26/20	11/1/20
Mount and Power electromagnet	Tony, Danié	11/16/20	11/27/20
Web App / Communications			
Implementation of data models	All	9/21/20	9/28/20
Programming chess web app	Brandon	10/12/20	10/19/20
Establish comm between micro-controller and device	Brandon, Danié	9/29/20	10/3/20
Basic web app using input from microcontroller	Brandon	10/4/20	10/11/20
Create API for moving pieces on chess board	Brandon, Tony	10/8/20	10/14/20
2 player chess on different machines	All	10/19/20	10/26/20
Integration/Testing			
Test piece detection	Danié	11/10/20	11/12/20
Test microcontroller to web connection	Brandon, Danié	10/28/20	11/2/20
Test actuators	Tony, Danié	11/20/20	11/29/20
Test electromagnet	Tony, Danié	11/24/20	11/29/20
Test web app functionality	Brandon	11/18/20	11/19/20
Microcontroller and motor integration	Tony, Danié	11/16/20	12/6/20
Final board build	Tony, Danié	11/23/20	12/6/20

Fig 6. Project Schedule