Final Design Specification + BOM for NobleMate Automated Chess Game

Date: April 7, 2025

Course Name: ENEL 400

Team Name: NobleMate

Team Number: Group 15

Team Members:

Saud Amjad (UCID: 30170966)

Hamzah Nadeem (UCID: 30180744)

Megh Patel (UCID: 30170343)

Grayson Bibby (UCID: 30172744)

Kunj Patel (UCID: 30170506)

Nihar Trivedi (UCID: 30170517)

Instructor: Dennis Onen

TA: Devin Atkin

Table of Contents

\

1. Acknowledgements

We would like to acknowledge the Department of Electrical and Software Engineering at the Schulich School of Engineering at the University of Calgary. Furthermore, we were honoured to have the guidance of Dr. Denis Onen and Devin Atkin for their brilliant insight and technical experience, as well as their constructive criticism and feedback on the project. In addition, we would like to thank the entire team at the Electrical Makerspace at the Schulich School of Engineering for offering parts and technical assistance/insights to this project.

2. Predecessor Works and AI Tool Use

An automated chessboard was developed previously by a user named Greg on Instructables [1], which was the inspiration for the solution developed over the semester.

Figure 2.1: Automated Chess Board

Many aspects of the project were inspired by this example, such as the automatic movement of the pieces through an XY gantry system and the implementation of a computer to play against the user. The gantry system was used in conjunction with an electromagnet as a method of movement. Magnets were attached to the base of the pieces so that when the electromagnet was activated underneath a certain piece, it would move along with it.

Additional features were implemented to make the automatic chessboard appeal to a wider audience (such as those unable to move the chess pieces themselves). Furthermore, Stockfish was integrated into the project as opposed to the one used in the Instructables reference [1] because it is a far more powerful and customizable engine that can be modified to include more features in the future.

The final product uses an Arduino and a Raspberry Pi that communicate serially. The Arduino controls the stepper motor drivers for precise, calibrated movement, and the Raspberry Pi is used for Stockfish and the Google API. Voice-detection and communication are used between the user and the chessboard so the user can control the difficulty of play and the chess pieces.

To move a chess piece, for example, the user would say “Pawn from A2 to A3” into a microphone. The voice-to-text module would process the sound data and update the chessboard position in the code. Once the move is detected, the gantry system will move the electromagnet to the piece, activate the electromagnet, and move the piece to the desired position. The gantry updates the chessboard in parallel with the software.

Use of AI Tools:

Stockfish Algorithm Implementation:

The Stockfish Algorithm is an AI model and is used as the player's opponent for the chess game. The difficulty and search depth are set by the user at the start of each game and passed through a C++ file that uses boost process to communicate with Stockfish through UCI protocol [11]. The Stockfish files are included in the project folder on the Raspberry Pi and directly embedded into the program for simpler communication between the programs.

Serial Communication Requirements:

ChatGPT was used to help with the installation of packages required to run the serial communication between the Raspberry Pi and Arduino. Facilitated the control of the stepper drivers and communication with the gantry system using data transferred between the Arduino and Raspberry Pi.

Voice Recognition - Google's Speech-to-Text Implementation:

There was a single Python file composed of five functions:

- Obtaining User Move Input (i.e. "Knight from E2 to F4")
- Choice for Piece Colour ("White" or "Black")
- Choosing A.I. difficulty (i.e. "Easy", "Medium")
- Move/View Functionality
- Rematch/Exit Functionality

ChatGPT used to understand how to set up the Google Cloud API authentication default credentials (ADC). Required as a prerequisite to create requests for the paid version of the API. The o3-mini-high ChatGPT model was used in collaboration with research from resources available online for Google Services [5] on how to track requests (including pricing information for the Speech-To-Text API) and understanding how function calls are possible in Python for the speech recognizer (Google provides sample code examples to call the speech recognizer in Python). ChatGPT was used to help understand how the SpeechRecognition in Python library can work to capture live audio. Sample code found online often uses pre-existing audio files, which are not sufficient for the project's use case.

The differences between Google's free Web Speech-To-Text API and the Google Cloud paid version were compared. The tradeoffs between both were taken into consideration for the final product. This interaction with ChatGPT was extremely helped determine the limitations of both speech-to-text APIs.

Raspberry Pi Package Installation:

Utilized ChatGPT for installing necessary files to run the modules and APIs integrated into the software due to the lack of expertise and experience in using a Linux operating system among the Noblemate project members.

The packages and files installed through the direction of ChatGPT include the necessary Python files required to run the SpeechRecognition, pyaudio and google-cloud-speech modules. This includes setting up the virtual environment needed to run these processes due to errors received when trying to install the modules outside of the virtual environment.

Created a Makefile with ChatGPT to efficiently compile and run the code without having to specify all the directories and manually type all the files needed to include in the project to compile. Although subsequent additions to the Makefile were completed manually.

3. System Block Diagrams

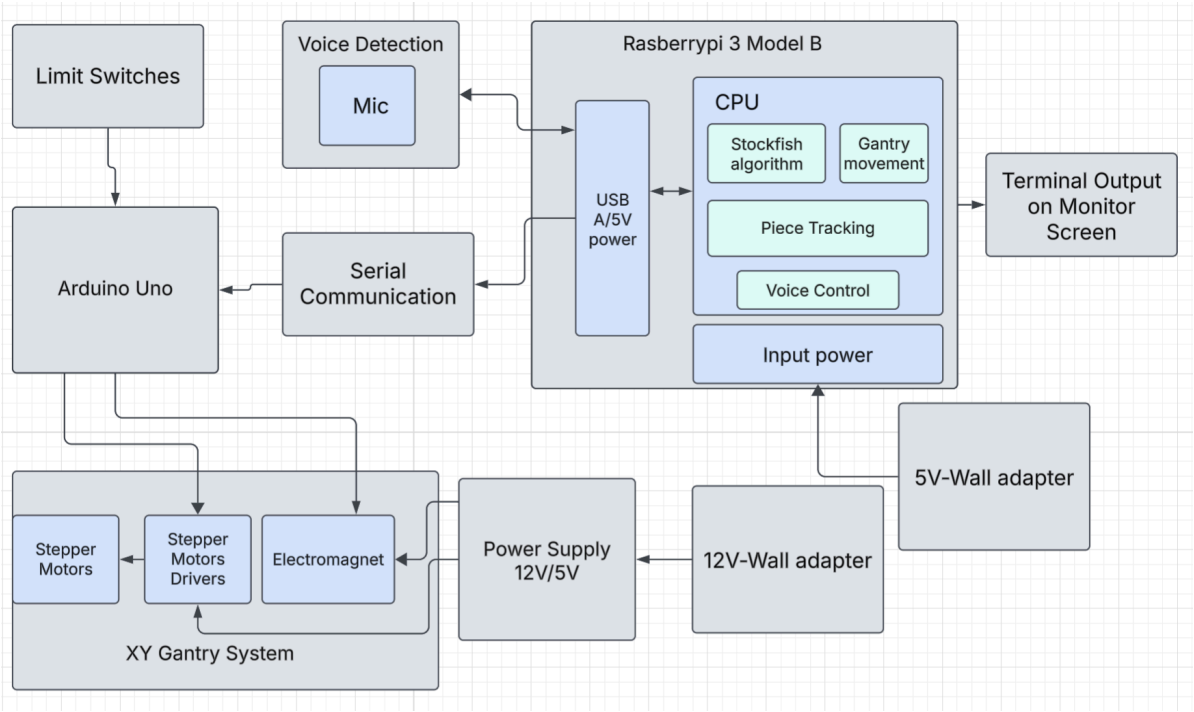


Figure 3.1: Hardware System Block Diagram

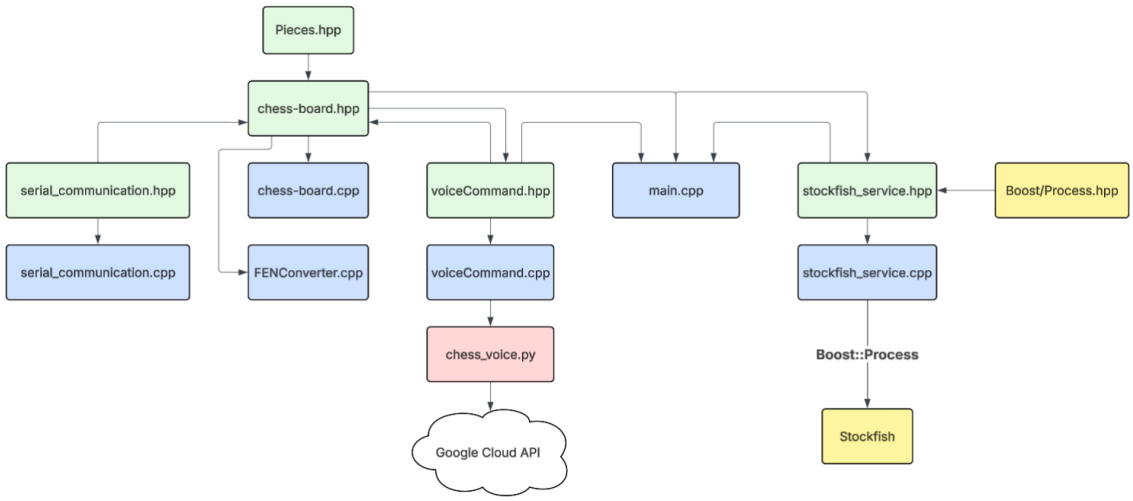


Figure 3.2: Software Block Diagram to Demonstrate the Connectivity of the Modules

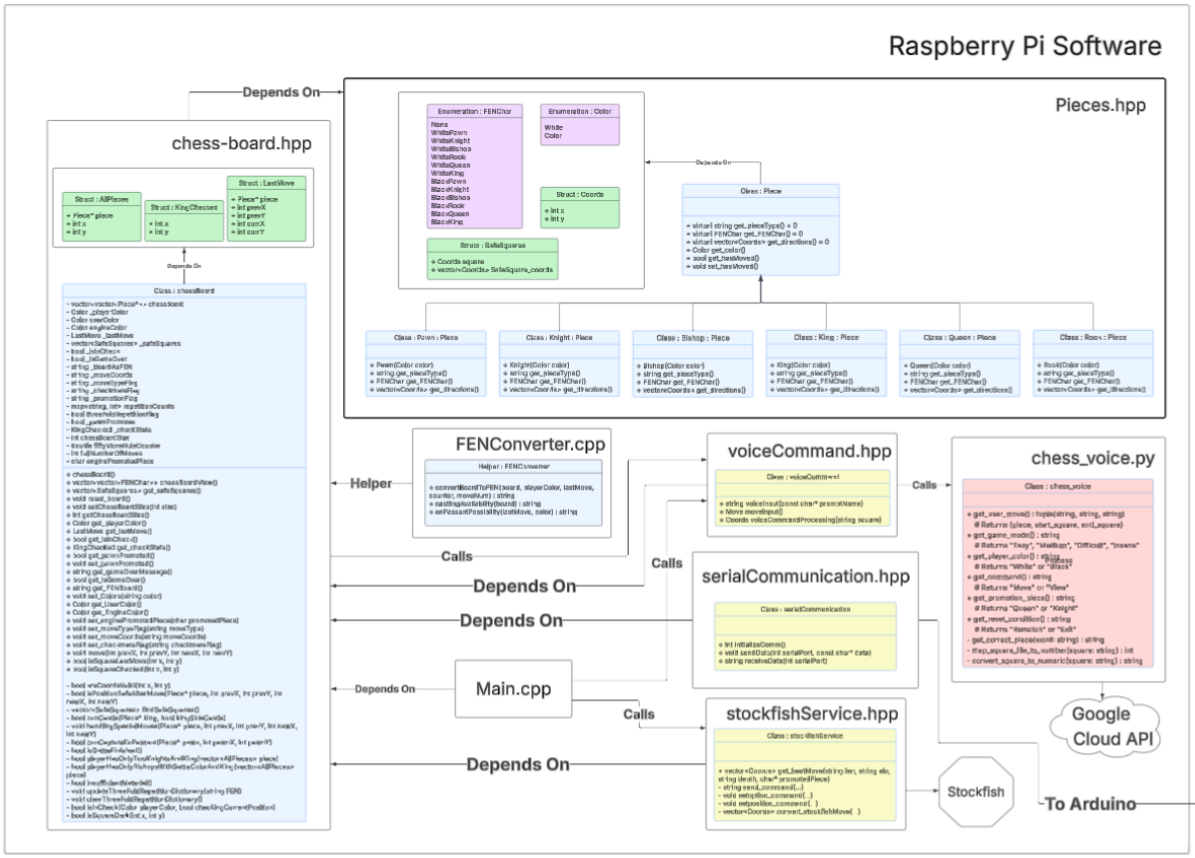


Figure 3.3: UML Class Diagram of the Software Showing Classes and Methods of Classes along with Interaction Between Files/Modules

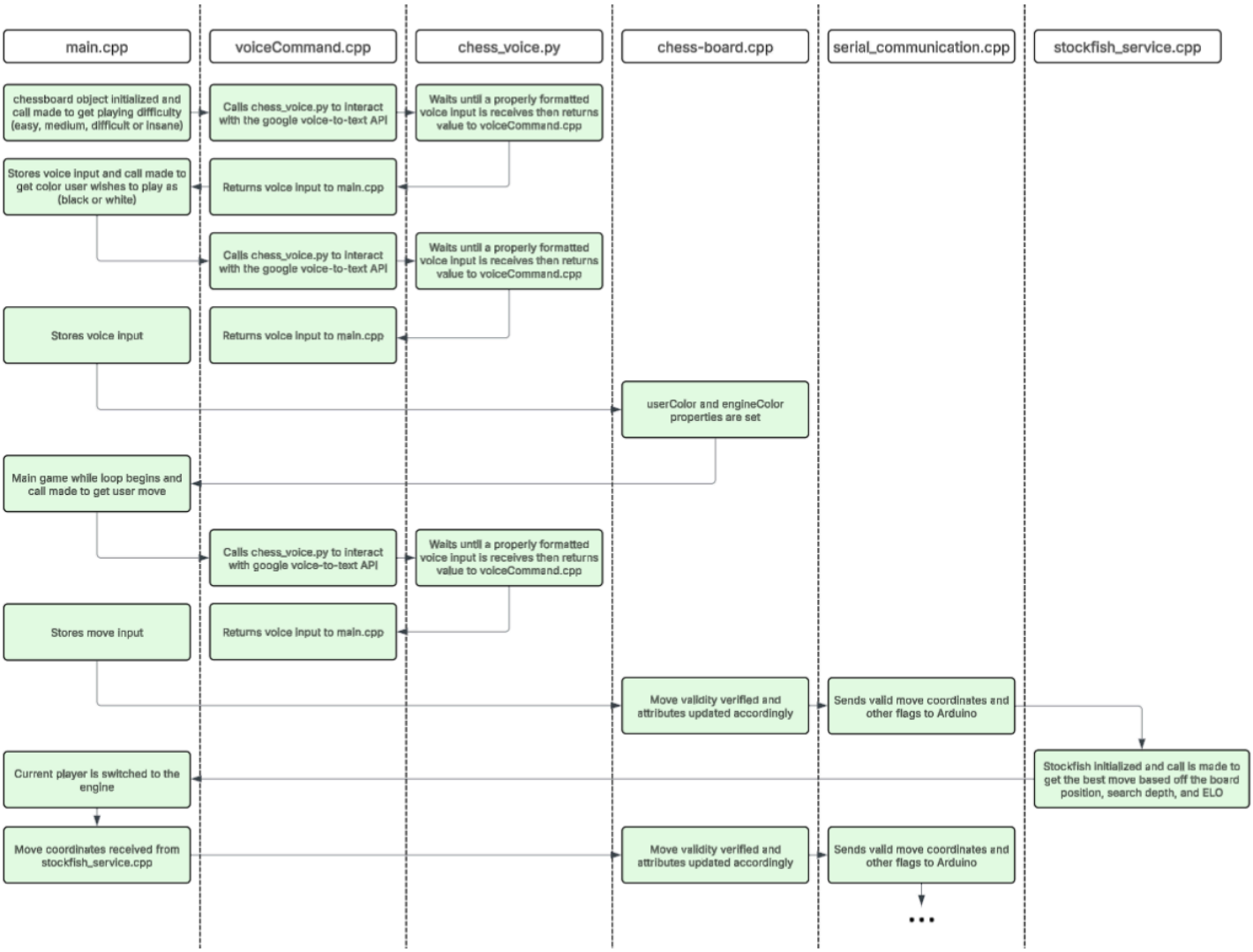
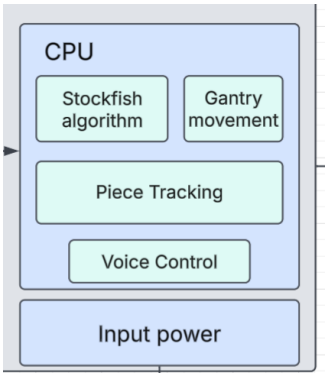


Figure 3.4: Gameflow Diagram Demonstrating the Relationship Between Functions Under the Assumption that the User Chooses White and all Moves are Valid

4. Block Analysis

Hardware Block Analysis



Raspberry Pi 3 Model B *(Pre-existing, integrated into the system)*

Figure 4.1: Raspberry Pi 3 Model B System Diagram

The Raspberry Pi 3 serves as the central processing unit (CPU) of the system, responsible for high-level decision-making, processing user inputs, and controlling various subsystems. It is a pre-existing board that has been integrated into this design to handle tasks such as running the chess engine, processing voice commands, and interfacing with other hardware components. A key role of the Raspberry Pi is running the Stockfish chess engine, an open-source AI that evaluates possible positions and outcomes to determine the best move in any given situation. The Raspberry Pi updates Stockfish with the current board configuration and, in return, receives the best move recommendation, which is then played by the opposing AI. Additionally, the Raspberry Pi is responsible for processing the user's voice inputs for the gantry movement.

Stockfish Algorithm (*Pre-existing Module, Integrated*)

Stockfish is one of the strongest open-source chess engines available, capable of evaluating board positions and calculating optimal moves within milliseconds. Running on the Raspberry Pi, Stockfish enables the chessboard to function as an intelligent opponent that can respond to user moves in real time. It evaluates thousands of possible moves and their outcomes to select the best strategy. Additionally, it can be set to various difficulty levels, allowing players to customize their experience. Stockfish is set to 4 modes: easy, medium, hard and insane, each with an ELO of 1200, 1800, 2400 and 3200, respectively. Stockfish works alongside the piece tracking system to ensure that the actual positions of the chess pieces on the physical board align with the digital game state.

Piece Tracking (*Designed, Software Module*)

The piece tracking module ensures that all chess moves are accurately recorded and that the board state remains consistent with the software's expectations. This system keeps a digital footprint of the playing field, creating a more manageable system to keep track of the movements of pieces, preventing errors such as incorrect moves. The tracking system is essential for verifying valid moves before sending the information to the automated gantry system, moving the correct pieces to the intended positions.

Voice Control (*Designed, Software Module*)

The voice control system enables hands-free operation of the chessboard, allowing users to issue verbal commands instead of manually inputting moves. This is a core aspect of the project due to its target for those who have limited mobility. The system interprets spoken chess moves and converts them into commands that can be processed by the Raspberry Pi. This is done by sending these voice commands to Google's API, which is then converted to text. This converted text is then processed into a move that can be sent to the automated gantry. The voice control module is programmed to recognize standard chess notation (e.g., "Knight from D3 to E5") and distinguish between different move types, including captures, castling, and pawn promotions.

X-Y Gantry Movement (*Designed, Software Module*)

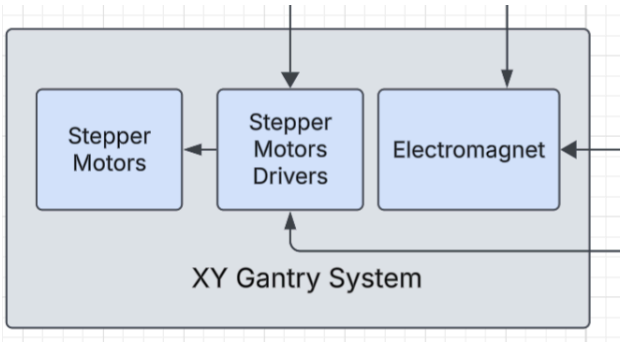
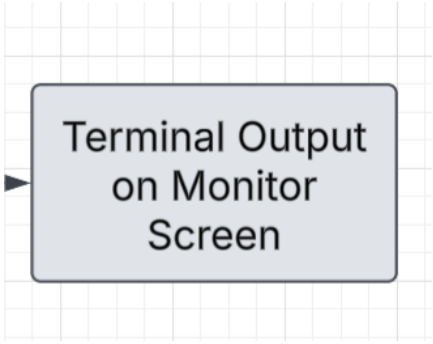


Figure 4.2: X-Y Gantry Movement System Block Diagram

The gantry movement system translates computed chess moves into physical actions, ensuring that the XY gantry system accurately picks up and places pieces. This involves precise coordination of stepper motor movements along two perpendicular axes. This system also ensures that no pieces collide when a move is being made. Additionally, the gantry system is also responsible for controlling an electromagnet based on the moves sent by the Raspberry Pi.

Terminal Output on Monitor Screen(*Pre-existing, Integrated into the System*)

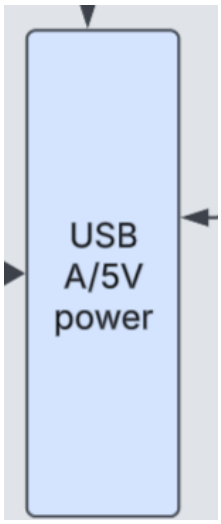
Figure 4.3: Terminal Output for HMI



The terminal screen serves as the human-machine interface, acting as the primary communication bridge between the user and the Raspberry Pi. It displays all essential information related to the game, including the current game mode, which player or color is active, and a log of the moves made by each player. This allows users to follow the flow of the game in real time. The terminal will also display a matrix or visual representation of the move history. This matrix can highlight the sequence of moves and indicate how the algorithm evaluates potential options, offering a clearer view of its strategic reasoning. Overall, the terminal provides an informative and interactive interface, supporting gameplay tracking and insight into the system's internal logic.

USB A Port/5V (*Pre-existing, Integrated into the System*)

Figure 4.4: USB A Port/5V System Block Diagram



The USB A/5V port on the Raspberry Pi plays a key role in both power delivery and data communication. It serves as the connection point for the Arduino Uno, enabling serial communication between the two microcontrollers. The Raspberry Pi sends commands to the Arduino, which controls the XY gantry system, electromagnet, and stepper motors. The Arduino, in turn, reports back positional data and limit switch status, ensuring precise movement of chess pieces.

The microphone for voice detection is also connected to the Raspberry Pi via the USB port, allowing it to capture voice commands from the user. The Raspberry Pi processes these commands using built-in voice recognition software, interpreting the player's move requests and integrating them with the Stockfish chess engine for decision-making. This seamless communication enables a fully automated, voice-controlled chessboard experience.

Arduino Uno (*Pre-existing, Integrated into the System*)

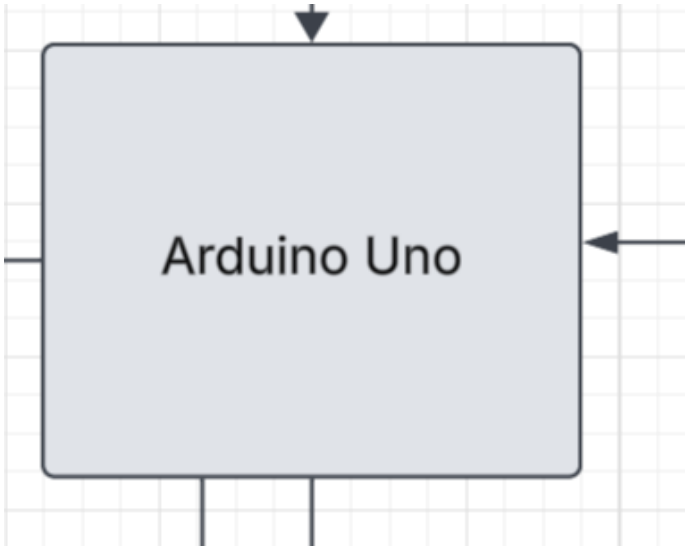


Figure 4.5: Arduino Uno System Block Diagram

The Arduino Uno is responsible for direct hardware control, particularly for the stepper motors, electromagnet, and limit switches. While the Raspberry Pi processes higher-level chess logic and strategy, the Arduino translates these commands into precise motor movement. Additionally, it processes inputs from the limit switches to ensure that the gantry system does not exceed its operational boundaries. The Arduino also manages updates for the LCD screen, ensuring that it displays the most current game information.

Limit Switches *(Pre-existing, Integrated into the System)*

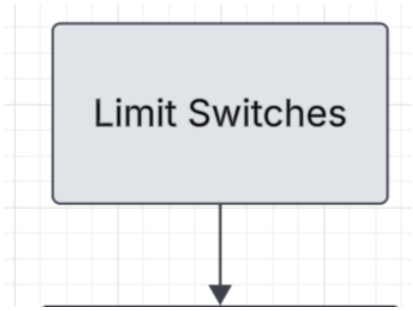


Figure 4.6

Limit switches play a crucial role in the safety and calibration of the XY gantry system. These sensors detect when the gantry reaches its movement limits, preventing mechanical damage. During system startup, the limit switches help calibrate the motors by defining the "home" position of the gantry. They ensure that every move starts from a reference point, which is essential for maintaining alignment between the digital and physical board states.

Microphone *(Pre-existing, Integrated into the System)*

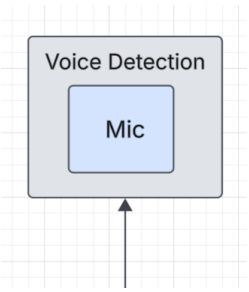


Figure 4.7: Microphone

A microphone captures spoken commands, which are then processed by the Raspberry Pi's voice recognition module. The system employs speech-to-text algorithms to extract chess-related commands while filtering out background noise. Once converted into text, the move is validated against legal chess rules before being executed on the board. The microphone interface ensures that the system accurately detects and interprets user inputs, making gameplay more fluid and intuitive.

Stepper Motors (*Pre-existing, Integrated into the System*)

Figure 4.8: Stepper Motors

Stepper motors are responsible for controlled movement along the X and Y-axis. Stepper motors move in discrete steps, allowing precise positioning. They receive pulse-based movement commands from the stepper motor drivers, ensuring accurate piece placement.

Electromagnet (*Pre-existing, Integrated into the System*)

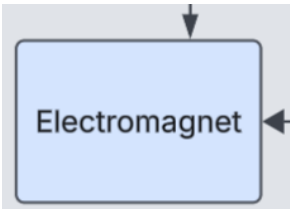


Figure 4.9: Electromagnet for Magnetic Piece Movement

The electromagnet is a key component in the automated movement system, responsible for lifting and placing the pieces. When activated, it generates a magnetic field that attracts and holds the magnetic pieces. When deactivated, the magnetic field is released, allowing the piece to be placed at the desired location. The electromagnet ensures a grip on the pieces while preventing unwanted movement or misalignment.

Stepper Motor Drivers (*Pre-existing, Integrated into the System*)

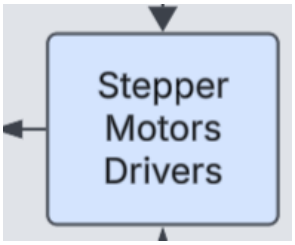


Figure 4.10: Stepper Motor Drivers

The stepper motor drivers regulate power and control signals for the stepper motors, ensuring smooth movement. The drivers translate control signals from the Arduino into signals needed to drive the motors.

Power Supply (12V/5V) ***(Designed, Integrated into the System)***

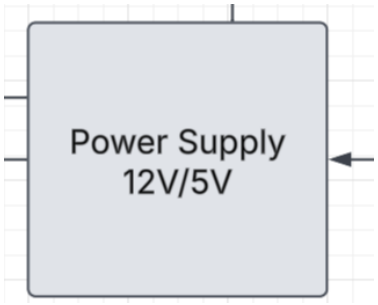
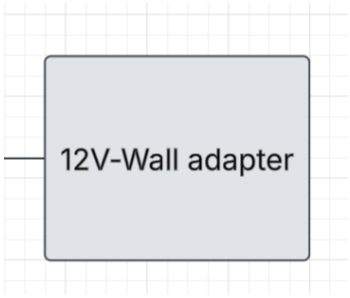


Figure 4.11: Power Supply for Distribution

The power supply unit converts the electrical input from a standard wall adapter into 12V and 5V outputs. The 12V supply powers the stepper motors, while the 5V supply is used for the components such as the Arduino. This was designed and implemented in the PCB, which steps down the 12V to 5V for applications. The original plan was to use the power supply to power all the applications in our system, but due to complications with the current supply, the power to the Raspberry Pi had to be rerouted to an individual wall adapter.

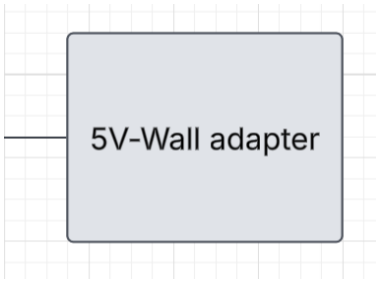
Wall Adapter 1 (*Pre-existing, Integrated into the System*)

Figure 4.12: 12V Adapter for Power Supply



The wall adapter serves as the primary source of electrical power for the entire system. It provides a stable AC input that is converted by the power supply into usable DC voltages. This ensures that the automated chessboard operates continuously without the need for battery replacements.

Figure 4.13: 5V Adapter for Raspberry Pi(*Pre-existing, Integrated into the System*)

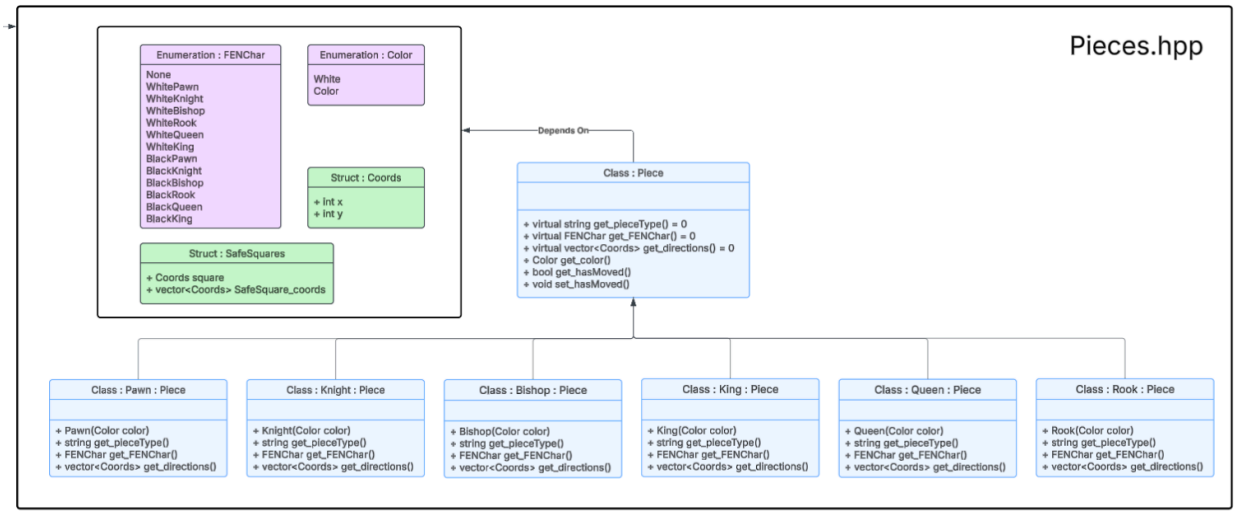


This wall adapter is specifically used as a power source for the Raspberry Pi [2] due to the insufficient current being provided by the power supply.

Software Block Analysis

Note: all of the methods and attributes in Pieces.hpp and most of the methods and attributes in the chess-board.hpp file, along with the subsequent function declarations, were originally made in the reference shown in [9]. The tutorial was created in TypeScript, though each file was manually transcribed into C++. Some files were modified to accommodate the specific needs and implementations of modules such as Stockfish, whose code is included in the project file as opposed to interfacing with it through the Stockfish Rest API [10]. Interfacing with the chess game is different since the video's implementation is intended to be used online as a website whereas the project's chessboard is physical and being controlled by voice.

Figure 4.14: Pieces.hpp

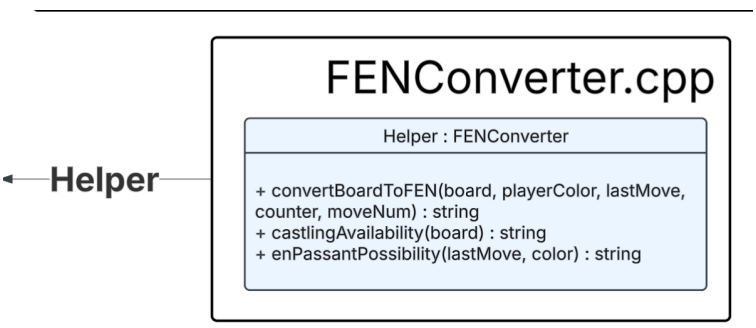


Where the chess pieces are created and given properties which include direction of movement, color, and piece type. Each piece type is a subclass of Piece and each has its properties, such as direction of movement and have moved properties for the king, rook, and pawn which each have an added functionality before they move (moving two spaces forward for the pawn and castling for the king and rook). They each have constructors to set the piece colors and getters to get the piece type and to get their FEN character representation [8]. In the parent class, *Piece*, its getters and setters for *hasMoved, color, FENChar,* and *pieceType,* attributes are overwritten in each of its subclasses.

Figure 4.15: chess-board.hpp

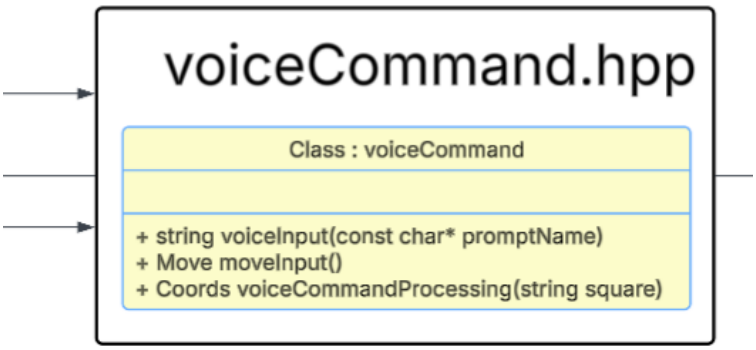
The constructor for the *chessBoard* class initializes the board position with pieces belonging to the *Piece* class in the *Pieces.hpp* file. For subsequent games, the board is reset by the *reset_board* method that deletes existing pieces and reinitializes the board with a new set of chess pieces. Various getter and setter methods are used to access the private attributes of the *chessBoard* class. An important functionality is its ability to detect whether a move is valid. Valid moves for each respective piece of the current player color (managed by the *_playerColor* attribute) are initialized by calling the *_getSafeSquares* method. Coordinates are obtained from the *findSafeSquares* method which calls *areCoordsValid*, to ensure valid user input, *isPositionSafeAfterMove,* to simulate the move, and *isInCheck,* that iterates through the board and every piece to verify whether the king is being attacked by a piece. The *move* method controls piece movement. Two coordinates, start and end position, are inputted as parameters. The valid move checker methods mentioned above determine move validity. The function returns without updating the board properties in the case of an invalid move. Otherwise, at the end of the *move* function, the necessary attributes are updated and information is sent to the Arduino through the *serial_communication.hpp* file. This is how a valid move is communicated to the gantry system. Methods such as *updateThreeFoldRepetitionDictionary* and *isGameFinished* determine and set game over conditions. Other methods such as *handlingSpecialMoves*, *canCaptureEnPassant,* and *canCastle* manage special moves (which are executed in the *move* method).

Figure 4.16: FENConverter.cpp



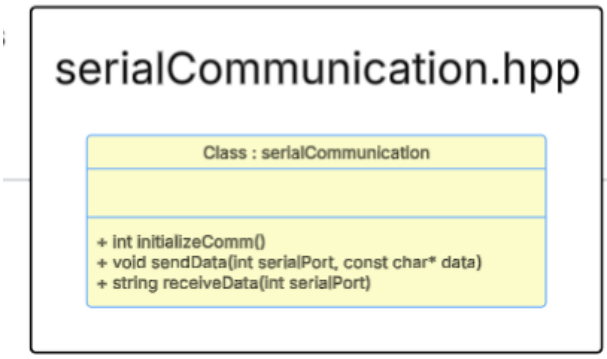
Interfaces with the *chess-board.cpp* file to assist in the conversion of the *chessboard* attribute to a FEN string [8]. It interfaces with several *chessBoard* methods and attributes such as *fiftyMoveRuleCounter* and *fullNumberOfMoves* as well as implementing its own checkers such as *enPassantPossibility* and *castlingAvailability* in *convertBoardToFEN* to fully convert the chessboard to a FEN string [8] to eventually be sent to Stockfish to obtain the best move. Called in the *get_FENBoard* method in *chess-board.cpp.* It is a Stockfish requirement to set the board position through a FEN string [8].

Figure 4.17: voiceCommand.hpp



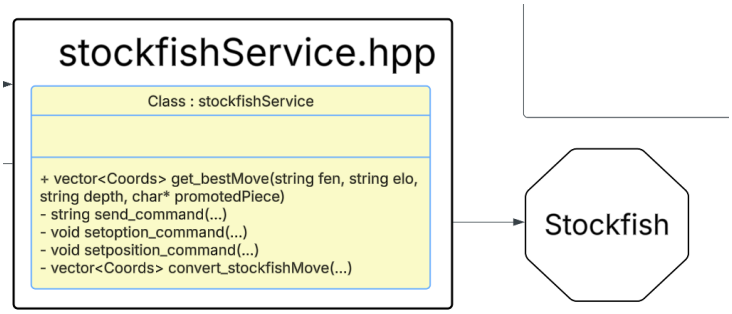
Interfaces between the game initialization user inputs and move inputs in *main.cpp* and called in *chess-board.cpp* to handle pawn promotion. *moveInput* is used to handle user moves while *voiceInput* handles all other voice commands (both have different return values). They each interface with the *chess_voice.py* file (which directly interacts with the google voice API) by initializing a link to the .py file, specifying the *chess_voice.py* function to interface with then calling the function and converting the return values to C++ variables to be subsequently sent to the variables in the *main.cpp* and *chess-board.cpp* files. The *voiceCommandProcessing* converts the string returned by the *chess_voice.py* function into integers that can be inputted into the *move* method in *chess-board.cpp* from the *main.cpp* file.

Figure 4.18: serialCommunication.hpp



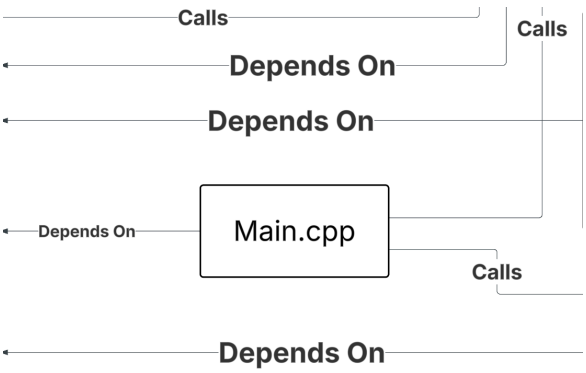
Interfaces with the *chess-board.cpp* file in the *move* function and serves as the link between the Raspberry Pi and the Arduino. Occurs in the *move* function to be sent after the move is confirmed to be valid. *initializeComm* initializes the communication channel by providing the USB port to which the Arduino is connected (the Arduino is interfacing with the Raspberry Pi from its USB port). *sendData* takes a C++ string as the input in the following order: initial file, initial rank, final file, final rank, move type, game over type (if it's checkmate or stalemate for example), and the promotion piece.

Figure 4.19: stockfishService.hpp



Interfaces with *main.cpp* only. Uses Boost Process to interface with Stockfish through UCI [11]. Commands inputted into Stockfish are listed in the Stockfish UCI commands [11]. *get_best_move* is called in the game loop in *main.cpp* in which the saved variables set at the start of the game for engine ELO and search depth are inputted along with the FEN string [8] obtained in *main.cpp* from *chess-board.cpp* that sets the board position. *send_command, setoption_command,* and *setposition_command* each represent possible commands in the Stockfish docs and are all called in the *get_bestMove* function.

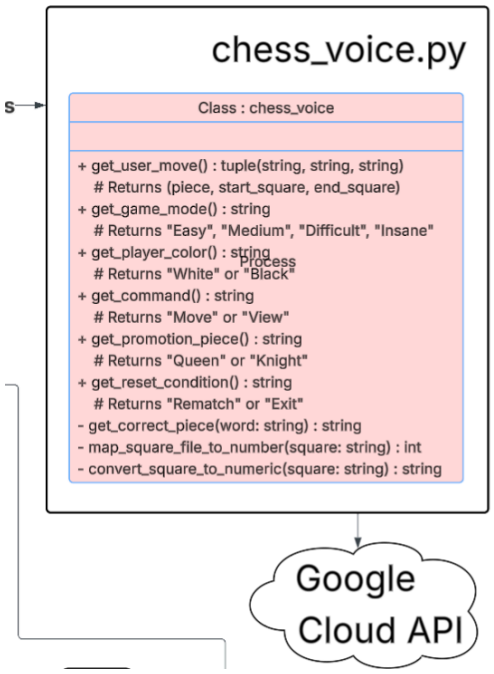
Figure 4.20: main.cpp



main.cpp is the core block of the software and interacts with all the modules sequentially demonstrated Figure 3.4. It is composed of two main while loops. The outer one loops over the entirety of the main function (excluding the creation of the chessboard object and the initialization and deinitialization of the C++ and Python communication channel through the PyRef() and PyDeref() functions). The inner

while loop loops around the gameplay functionality, where the user and Stockfish are continually prompted for moves until the game ends. An additional feature of *main.cpp* is its ability to reset the chessboard once the game is over by calling *voiceCommand.cpp* to ask for user input from *chess_voice.py* to either play a new game with different difficulty settings or rematch the chess AI with the same difficulty but with the opposite colour. Once this setup is complete, the game loops back to the beginning and a call is made to *chess-board.cpp*'s *reset_board* to delete the old *Piece* objects present on the chessboard and reinitialize all the other properties of the *chessBoard* class are reset to their initial state.

Figure 4.21: Chess_voice.py



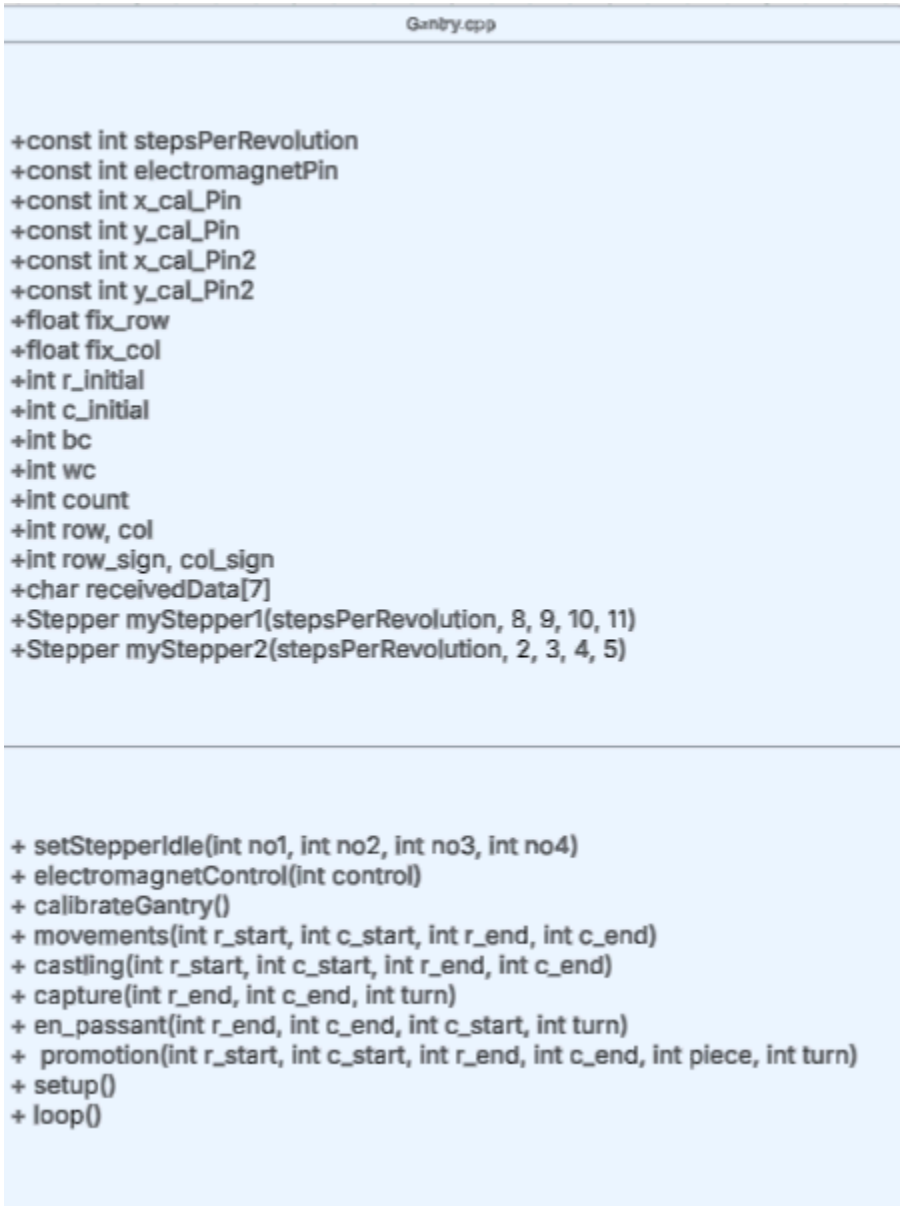
The program, *chess_voice.py*, consists of 6 main core voice-input dependent functions that are interfaced with *Google Cloud Speech-to-Text v1 API*. As functions within this script are called, requests are made to the Google Cloud API [5] and the raw output is then processed within this code, including matching similar words, splicing output data for desired keywords and fuzzy matching the code produces the best possible match to the user input.

The core functions for *get_game_mode()* and *get_player_colour()* occur during initialization, where the user is required to state select keywords in order to advance further into the actual game. From these functions, the user/Stockfish AI colour is determined, along with the skill rating (ELO) of the Stockfish AI. This interfaces with *voiceCommand.cpp* which is where it is called and where the python strings and characters are converted to C++ strings. The core functions for *get_user_move()*, *get_command()* and *get_promotion_piece()* are called post initialization and are called prior, during and after a user makes a move. The function *get_user_move()* has 3 accessory functions called *get_correct_piece(), map_square_file_to_number()* and *convert_square_to_numeric()*, which are basic functions that allow for the processing of inputs such as "Pawn from A1 to A3", which splices moves "A1" and "A3" and then convert the file from a letter to a number, such as "A" mapping to "1". Thus, the output value from the get_user_move() function will return a string "1113" . This string is returned to *voiceCommand.cpp* which then converts it to four integers in the *voiceCommandProcessing* function which is then returned to *main.cpp* in the Move structure. This structure is then unpacked in *main.cpp* where the four integers are passed into *chess-board.cpp*'s move function to verify the move is valid.

The final core function is the *get_reset_condition()*. This is a post GAME END condition, where following the winner being printed to screen, the user is given the option to either reset/exit the game. This variable interfaces with *main.cpp* where the value returned from *voiceCommand.cpp* which calls and processes the Python function and its return values. *main.cpp* takes the returned value and determines whether to restart the game from the very beginning where the user chooses the engine ELO strength or trigger a rematch in which the user's colour is swapped but the chess engine ELO strength remains constant.

Gantry Code Analysis

Figure 4.22: Gantry.cpp



The program gantry.cpp contains 9 functions. None of these functions return any values, they all just return void. The first function is setStepperidle(). This function is used to turn off the motor drivers when the motor is not being used so that they do not overheat. As inputs they take in the pins that the motor driver is connected to. The next function is electromagnetControl. This function is responsible for turning on and off the electromagnet when needed. The calibrate function is used to calibrate the gantry to a specific corner. *movements* is a standard movement function for each of the pieces and implements functions to move the piece around other chess pieces in a coordinated fashion to prevent contact between pieces. The *castling, capture, en_passant,* and *promotion* functions handle the special moves of the same name since each require additional functionality apart from the standard chess piece movement. For example, castling requires the rook to move around the king after the king moves two spaces and promotion requires swapping the pawn for another piece (in the case of the project either queen or knight).

loop constantly iterates over the code to check for whenever the information is received from *serialCommunication.cpp* called in the *move* function in *chess-board.cpp* from the Raspberry Pi, the corresponding move indicated by the flags sent from the Raspberry Pi is made. The first four flags correspond to the original and destination coordinates of the piece in that position. The next flag indicates the movement type; whether it is a capture, en passant or castling move. Second last flag indicates if promotion occurs or if checkmate or stalemate occurs. The last flag indicates the type of promotion piece.

5. Engineering Analysis

PCB Design:

A schematic was developed to determine the power required for each component. Although most of the crucial components were chosen and added to the schematic, a few of the smaller components had to be adjusted to fit the project's needs. Therefore, the crucial components of the circuit were simulated on both a simulation [12] and testing on a breadboard. The other components were only tested on simulation [12] before being incorporated into the final PCB.

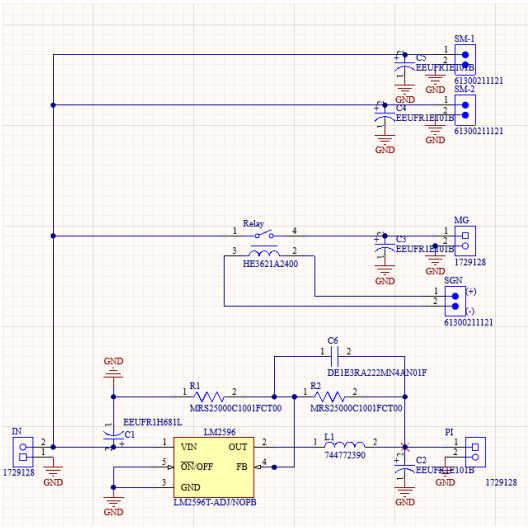


Figure 5.1: PCB Schematic Diagram

The following circuit showcases a stable 5V DC-DC converter, allowing us to regulate the output voltage from a 12V power supply. This 12V power supply is also used to power 2 stepper motor controller drivers and an electromagnet. The design schematic along the way has passive elements to help support stable power delivery, minimizing voltage ripple at the output. The design was made with careful consideration of the sensitivity of the components connected to the PCB. The 5V DC-DC converter was carefully designed to provide a stable 5V and a maximum 3A current draw.

To design the 5V DC-DC converter, a LM2596T-ADJ voltage regulator chip was used. The LM2596T-ADJ was chosen due to its efficiency, ease of implementation and its ability to output 5V and have a maximum current draw of 5A [3]. This is enough to meet the power demands of the Raspberry Pi[2]. The input capacitor at the input is 100uF rated for 16V [3], as they were the recommended capacitance for **C1** according to the LM2596T-ADJ datasheet. Since we are only supplying 12V to the circuit, a 16V rated 100uF capacitor [3] was a good choice. To configure the LM2596T-ADJ voltage regulator feedback pin, we used a simple voltage divider circuit as

mentioned in the datasheet. Resistor one **(R1)** was $1\,k\Omega$ [3] as mentioned in the datasheet [3]. The feedback resistor **(R2)** was calculated**,** with the given equation in the datasheet [3]:

$$R_2 = R_1\left(\frac{v_{OUT}}{V_{REF}} - 1\right) = 1k\Omega\left(\frac{5V}{1.23v} - 1\right) = 3.065\,k\Omega.$$

Connecting to the feedback circuit is another capacitor (**C6),** known as the feedback capacitor. Its value was calculated using a given equation in the datasheet [3]:

$$C_{FF} = \frac{1}{(31)(10^3)R_2} = \frac{1}{(31)(10^3)(3.065\,k\Omega)} = 10nF$$

We ensured that the capacitor used was rated for 5V or higher. For the inductor **L1,** the value was 33uH [3]. This value was chosen according to the given equation in the datasheet [3] to find the inductor volt ● microsecond constant:

$$E * T = (V_{IN} - V_{OUT} - V_{SAT})\left(\frac{V_{OUT} + V_D}{V_{IN} - V_{SAT} + V_D}\right)\left(\frac{1000}{150\,kHz}\right) = (12V - 5V - 1.16V)\left(\frac{5V+0.5V}{12V-1.16V+0.5V}\right)\left(\frac{1000}{150\,kHz}\right) = 18.883$$

Later, using this constant to find the inductor value via a graph provided in the datasheet [3]:

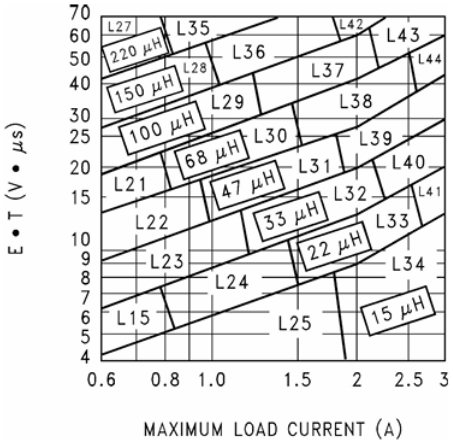


Figure 9-8. LM2596-ADJ

Figure 5.2: Inductance Chart for Respective Load

Given the constant of 18.883 and a maximum current requirement of 2.5A, L40 is where these two numbers intersect. Later using a table [3] to determine its value.

Table 9-1. Inductor Manufacturers Part Numbers (continued)

	INDUCTANCE (µH)	CURRENT (A)	SCHOTT		RENCO		PULSE ENGINEERING		COILCRAFT
			THROUGH-HOLE	SURFACE-MOUNT	THROUGH-HOLE	SURFACE-MOUNT	THROUGH-HOLE	SURFACE-MOUNT	SURFACE-MOUNT
L33	22	3.10	67148390	67148500	RL-1283-22-43	—	PE-53933	PE-53933-S	DO5022P-223
L34	15	3.40	67148400	67148790	RL-1283-15-43	—	PE-53934	PE-53934-S	DO5022P-153
L35	220	1.70	67144170	—	RL-5473-1	—	PE-53935	PE-53935-S	—
L36	150	2.10	67144180	—	RL-5473-4	—	PE-54036	PE-54036-S	—
L37	100	2.50	67144190	—	RL-5472-1	—	PE-54037	PE-54037-S	—
L38	68	3.10	67144200	—	RL-5472-2	—	PE-54038	PE-54038-S	—
L39	47	3.50	67144210	—	RL-5472-3	—	PE-54039	PE-54039-S	—
L40	33	3.50	67144220	67148290	RL-5472-4	—	PE-54040	PE-54040-S	—
L41	22	3.50	67144230	67148300	RL-5472-5	—	PE-54041	PE-54041-S	—
L42	150	2.70	67148410	—	RL-5473-4	—	PE-54042	PE-54042-S	—
L43	100	3.40	67144240	—	RL-5473-2	—	PE-54043		—
L44	68	3.40	67144250	—	RL-5473-3	—	PE-54044		—

Figure 5.3 Part Numbers for Inductor

L40 gives us a value of 33uH as an inductance. At last, the output capacitor (**C2**) was chosen, once again using a table provided in the datasheet [3]:

Table 9-6. Output Capacitor and Feedforward Capacitor Selection Table

OUTPUT VOLTAGE (V)	THROUGH-HOLE OUTPUT CAPACITOR			SURFACE-MOUNT OUTPUT CAPACITOR		
	PANASONIC HFQ SERIES (µF/V)	NICHICON PL SERIES (µF/V)	FEEDFORWARD CAPACITOR	AVX TPS SERIES (µF/V)	SPRAGUE 595D SERIES (µF/V)	FEEDFORWARD CAPACITOR
2	820/35	820/35	33 nF	330/6.3	470/4	33 nF
4	560/35	470/35	10 nF	330/6.3	390/6.3	10 nF
6	470/25	470/25	3.3 nF	220/10	330/10	3.3 nF
9	330/25	330/25	1.5 nF	100/16	180/16	1.5 nF
1 2	330/25	330/25	1 nF	100/16	180/16	1 nF
1 5	220/35	220/35	680 pF	68/20	120/20	680 pF
2 4	220/35	150/35	560 pF	33/25	33/25	220 pF
2 8	100/50	100/50	390 pF	10/35	15/50	220 pF

Since the output voltage we had was 5V, we picked the closest voltage output from the table [3], that being 6V. So this gave us a capacitor value of 470uF rated for 25V. C1 and C2 act as decoupling capacitors to help smooth the voltage ripple for stable voltage input and output.

After configuring the LM2596T-ADJ[3] to meet the requirements we had. We also added 2 12V output ports to the circuit to supply power to the stepper motor drivers, directly from the 12V input. In addition, we used a relay rated for 12V and 0.500mA to help control the electromagnet, this relay is controlled via the Raspberry Pi. This allows for the Raspberry Pi to safely control the electromagnet. After designing the circuit, we simulated the WEBENCH power designer [12] on the TI website, the DC to DC converter to verify its stability, voltage regulation and current output:

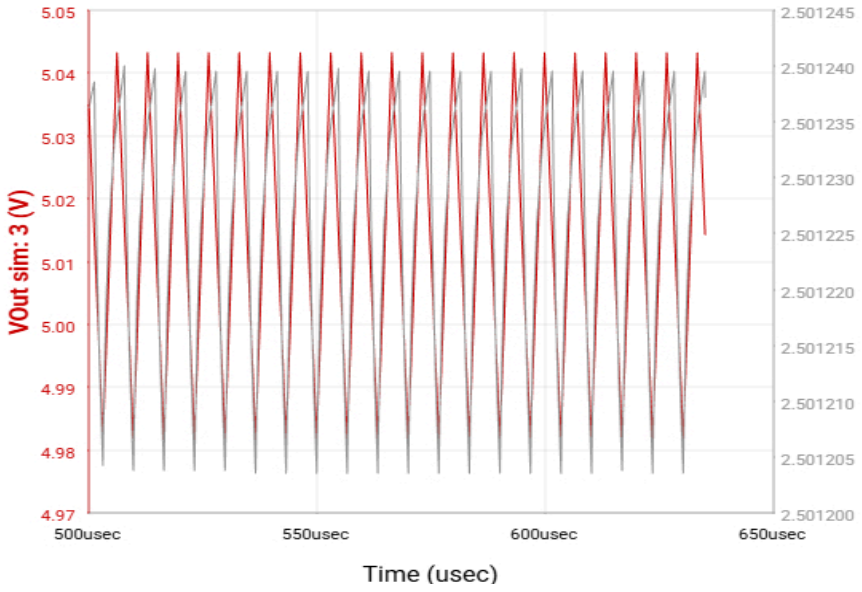


Figure 5.4: Graph of the Output Voltage (Right) and Current (Left) from the LM2596T-ADJ

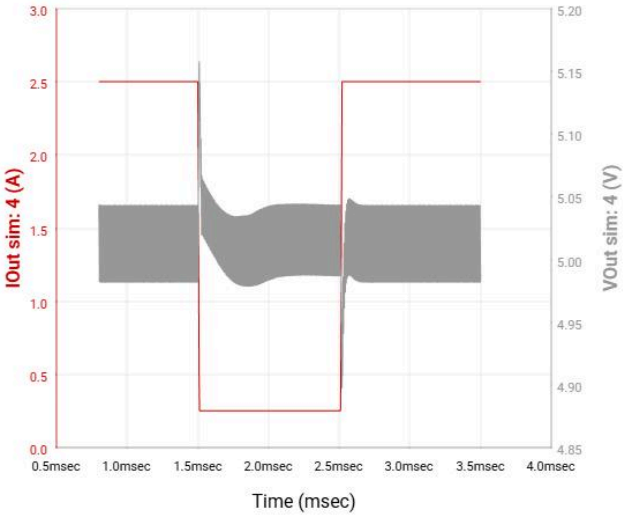


Figure 5.5: Graph of the Output Voltage (Right) and Current (Left) from the LM2596T-ADJ under load

Since the 12V step motor driver port is just a wire. Therefore, no simulation is needed for that part of the circuit. The relay was tested using a circuit made on a breadboard, the circuit behaved just as expected. It passed 12V to the electromagnet once a signal from the Arduino board was sent to the relay.

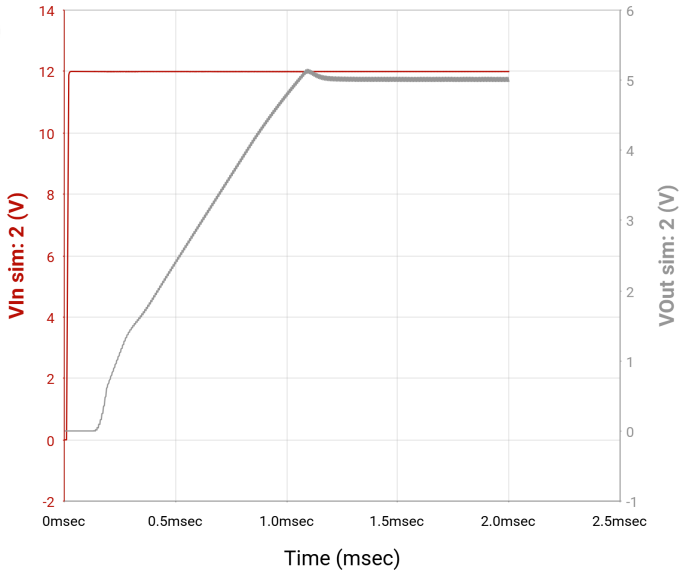


Figure 5.6: LM2596-ADJ Startup Simulation

XY Gantry System:

To control the movement of the chess pieces, two designs were considered. The first one was a XY gantry system, while the second option was a robotic arm that would pick up the pieces and move them to their desired location. We ultimately decided on using the gantry system with an electromagnet as it was easier to implement. With the robotic arm, there were potential issues of calibration and inaccurate movements. Plus, lifting pieces up would have been a huge problem to overcome, as the tops of some chess pieces, such as the pawn or bishop, are round in nature and thus harder to get a firm grip on. Also, the heights of standard chess pieces vary as well, so tuning the arm to each pieces' height would require the chessboard matrix to be accessible on the motor controller or the arm would have to be controlled by the Raspberry Pi. Both of these options would be much more difficult to implement over the gantry system. The gantry system allowed us to very easily move the chess pieces by moving an electromagnet around underneath the pieces and then turn it on when a piece needs to be moved. Also the mechanical side of creating a gantry is much simpler than creating a robotic arm. Moving the pieces this way also enhanced the feel and functionality of our project, as now it appears that the gantry moves all by itself, rather than seeing an arm moving about.

For the chess pieces, we had to choose between purchasing wooden pieces or 3D printing custom chess pieces. We chose 3D printed pieces because they caused the least amount of interference with the magnetic field that was required for moving the pieces across the board. This conclusion was sought after through an analysis that we conducted on the internet. Additionally with 3D printed pieces it would be much easier to secure the magnets to the pieces, as the holes made at the bottom of the pieces can be custom made to fit any sized magnet we chose. Additionally, the size of the pieces could be adjusted when using 3D printed pieces, ensuring all the pieces have the same diameter and appropriate size to move. Using 3D printed pieces allowed us to get the proper diameter size required to move the pieces in between other pieces.

For the stepper motor drivers, there were a few options that were considered, those being A4988, DRV8834, L298N and TMC2209. The most important factor considered when picking a stepper motor was the current limit per phase. The Nema 17 stepper motors needed 1.5 Amps per phase; both the L298N and DRV8834 could support this current. From these the L298N was picked due to them being easily accessible. As testing on the gantry system began, an overheating issue on the L298N drivers was encountered, where the heat shrinks present on the L298N drivers began overheating, resulting in poor control of the motors. This was due to the motor drivers pulling more current to stall them, as the power output was the same. When moving pieces, the motors pulled 0.5 A at 12 V, but when not moving, they pulled 1 A at 6 V. Since the motors were always pulling current, the heat shrink was not well equipped to disperse that much thus resulting in the overheating issues. During this time a change in stepper motor

driver was considered, as the L298N are not very efficient and convert a lot of input power to heat. However this issue of overheating was dealt with using software intervention. In the gantry code, whenever a stepper motor was not being used an idle function was implemented. This essentially turned the motor driver off, giving it time to cool off and not overheat. This completely fixed our overheating problem, as even turning off the drivers for a second or so allowed the drivers to cool off enough such that the overheating would not result in poor control of the motors.

To pick an electromagnet we mainly focused on finding one that would work with the existing power plan. This would mean that we would need to find a magnet that could be powered by 12 volts. This resulted in us getting a magnet that had around 180 N of force at 12V which was sufficient for our purposes. As testing on moving the pieces started, it was evident that the magnet was too strong and needed to be made weaker. To achieve this, power resistors were put in to limit the current to the magnet. A total resistance of 54 ohms was used, this number was determined through trial and error. Varying resistances were tested till multiple pieces would not be attracted to the magnet. Initially the magnet would draw 12 V at 0.450 amps which gave an internal resistance of 26.67 ohms. With the added 54 ohms the current became, $I = 12\,V/(26.67\,\Omega + 54\,\Omega) = 0.15\,A$. This is a reduction of current by a factor of 3.

Software Design:

Composed of several key blocks. These blocks are chess logic, stockfish move processing, voice command processing, and serial communication management.

Firstly, C++ has superior performance capabilities to other languages, such as Python, given its strong memory management capabilities, which was considered important when also integrating other more computationally intensive features such as Stockfish. Additionally, its object-oriented ability made it a preferred choice over C because it facilitated duplication of chess pieces and allowed simpler management of piece allocation and deallocation.

Given the choice of C++, Stockfish was also quite simple to communicate with through UCI [11] because the communication is done through Boost Process [7] (only available for C++). Stockfish provided ample documentation [6] for its integration along with commands to the set engine playing strength and commands to obtain the best move and also provided a link to Boost Process [7]. This made the Stockfish communication process simpler because through Boost Process [7], strings are sent to the engine to determine the best move.

It was quite simple developing a communication flow throughout the entire program; obtaining strings through user input, updating the board through the usage of strings to indicate the player move, using getter methods to obtain a FEN [8] string needed to set the board position in Stockfish, obtaining a best move string from Stockfish to update the board with its move and

finally, sending a string to the gantry through serial communication to interface with the physical board.

The voice control module was implemented using Python due to its available libraries for voice control such as SpeechRecognition and pyaudio to manage voice control capabilities. To communicate with the Python module, the Python.h C header was used which facilitates the conversion of Python characters and strings (which are the only two data structures returned) into C++ characters and strings (which includes strings inferred through the std namespace).

Serial communication is used to send data to the Arduino to determine the moves made by the gantry system. The serial communication interface was established through a C++ script and available header files. Chose to use the Arduino to interface with the gantry as opposed to the Raspberry Pi 3 Model B because the Pi was unable to supply a sufficient amount of power to the gantry system.

Microcontrollers:

Raspberry Pi:

Availability & Familiarity: One of our group members already owned a Raspberry Pi and was familiar with its setup and programming, which reduced costs and minimized the learning curve.

Computational Requirements:

The Raspberry Pi's quad-core 1.2 GHz ARM Cortex-A53 CPU and 1GB RAM provided sufficient power for handling system control, data processing, and user interface functions. Furthermore, four cores enables Stockfish to operate faster as its tasks can be split between the four cores while it searches for a move. This allows the increase of search depth with around the same time delay as it would take to search at a lower depth level with the same delay.

Multitasking Capability:

Since our project required simultaneous tasks such as sensor data processing, motor control logic, and communication with external components, the Raspberry Pi's ability to run a full OS (Raspberry Pi OS) made it an ideal choice.

Data Throughput Analysis:

We estimated the required data processing speed for sensor inputs, peripheral control, and logging. The Raspberry Pi's USB, GPIO, and serial communication interfaces were analyzed to ensure they could handle the expected data rates efficiently.

The Raspberry Pi acted as the central hub, running the high-level logic and interfacing with the user, while delegating real-time control tasks to the Arduino.

Arduino Uno:

Real-Time Motor Control & Peripheral Management:

To handle precise stepper motor control and hardware-level interactions, we selected the Arduino Uno based on the following considerations:

Availability & Familiarity:

The Arduino Uno was readily available in the Electrical Makerspace, and our team had prior experience programming Arduinos, making it an accessible and practical choice.

Precise Stepper Motor Control:

Stepper motors require carefully timed pulses to operate smoothly. The Arduino's dedicated digital I/O pins and real-time processing capabilities ensured accurate movement without the latency issues associated with the Raspberry Pi's non-real-time OS.

Limit Switch & Electromagnet Control:

The Arduino provided an efficient way to manage limit switches and an electromagnet, allowing for simple and reliable digital control.

Low-Latency Operation:

Since the Raspberry Pi runs an operating system that introduces processing overhead, the Arduino was chosen to handle critical real-time tasks without interruption.

By leveraging readily available hardware and our existing knowledge, we developed a system where the Raspberry Pi 3 Model B handled high-level processing, system logic, and data management. The Arduino Uno managed real-time motor control and hardware-level tasks with low latency.

6. Enclosure and 3D Print

We chose wood as the material for our enclosure for two key reasons. First, weight was a critical factor in our design considerations, as the overall system weighs approximately 3 kilograms. Given this constraint, wood provided the optimal balance between durability and lightweight construction. While other materials were available, wood offered the necessary structural integrity without adding excessive weight. Second, wood allowed us to secure the gantry system, ensuring stability and precision. By screwing the gantry into place, we minimized the risk of misalignment or unwanted movement, which could have affected the system's performance. The rigidity of wood helped maintain the accuracy of the gantry's positioning, ultimately enhancing the reliability of the entire system. Third, wood was easily accessible and easy to create our enclosure with .The alternative of 3D printing our enclosure was swiftly rejected due to the sheer size required, as just our base measures 24 in x 24 in. To achieve this in 3D printing, multiple blocks would have to be printed and then assembled, which wouldn't be as strong as a single block of wood.

Our chess pieces are 3D printed because we wanted to insert the circular magnets on the bottom. This allowed us to have iterative designs of the pieces to play around with different block sizes for the chessboard squares. The supports for holding the rails and electromagnet were also 3D printed. This was done to customize the supports to best fit the needs of our gantry system. Creating the supports would also allow us to secure the whole gantry system into the enclosure by creating screw holes in the support blocks. There were many iterations of the support blocks, especially regarding the ones resting on the middle rail. This includes the addition of support walls so that the middle rail does not derail off the 3D support, the adjustment of the length and thickness of the shaft used to hold the pulleys, and ventilation for the electromagnet. The final designs for the chess pieces and support blocks are shown below (also in Appendix E):

Figure 6.1: 3D Print of Chess Pieces

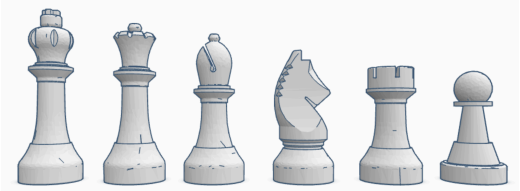


Figure 6.2: 3D Print of Gantry Support Blocks

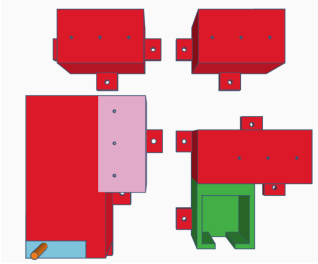


Figure 6.3: 3D Top View of Rail Supports

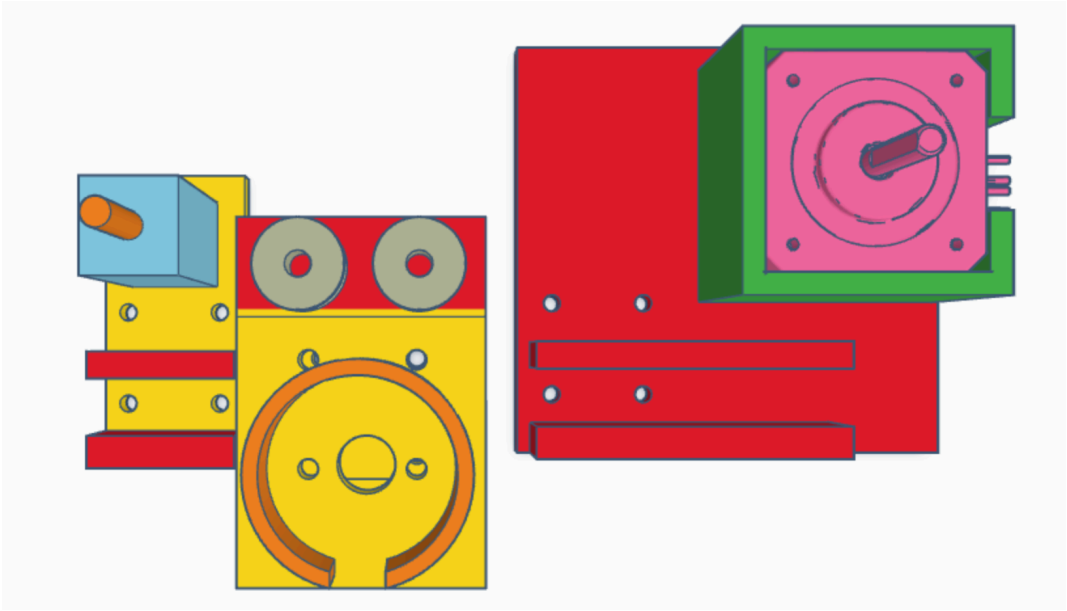
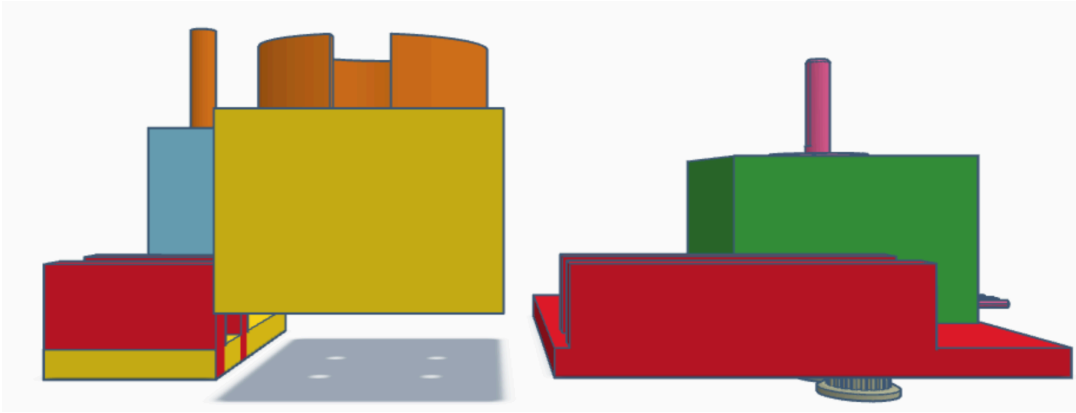


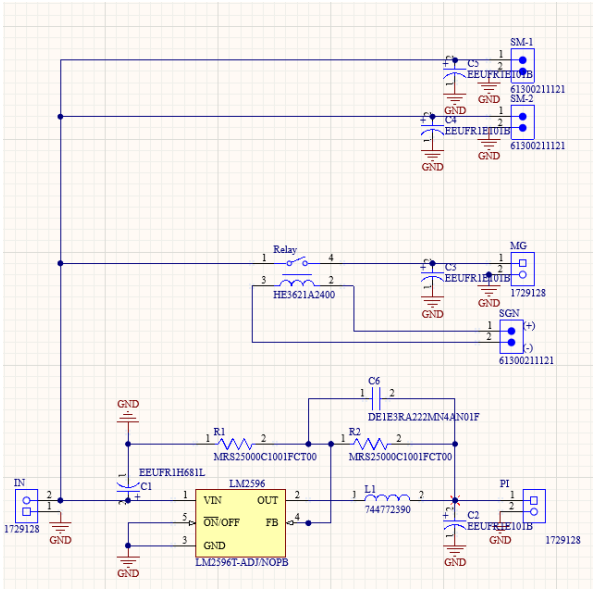
Figure 6.4: Side View of Middle Rail Supports



7. Printed Circuit Board

The Printed Circuit Board (PCB) serves as a critical component in distributing power to the various components and systems connected to it. It features a 12V to 5V 3A buck converter powering a Raspberry Pi, ports to connect two 12v Step motors and a 12v electromagnet. We utilized Altium to create the schematic and PCB for this project.

Figure 7.1 Circuit Schematic for the PCB:



The following figure, shown above, illustrates the circuit diagram for the PCB. We utilize a LM2596T-ADJ/NOPB voltage regulator, 33µ*H* inductor [3], 9001-05-00 Speed relay, various resistors and capacitors. These are all through-hole components. To connect the external systems to the board, we are using screw terminal connects and header pins, allowing for secure connection to the PCB.

Figure 7.2 PCB Layer 1:

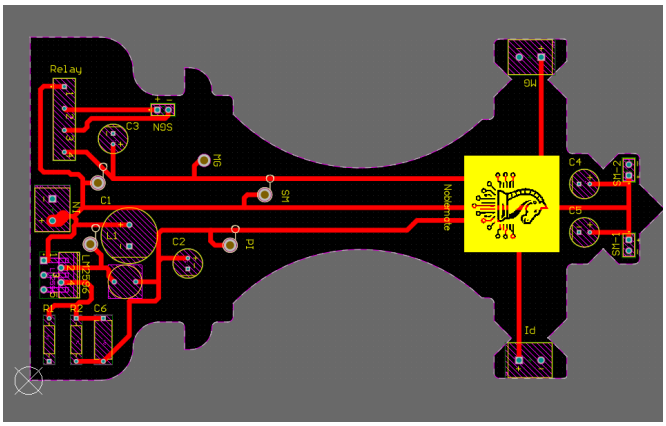
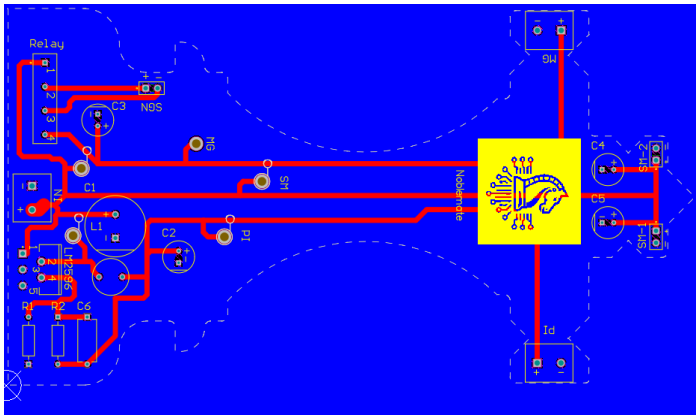


Figure 7.3 PCB Layer 2:



We designed a 2 layer PCB to ensure minimal complexity and cost for this project. First layer is used for traces and connection between the components and power flow within the PCB. We have various different trace widths from 1.27mm to 3mm to correctly support the flow of current. Layer 1 is $0.035\ mm$ thickness with a weight of 28.3 grams and $927.7\ mm^2$ of copper used. Layer 2 serves as the ground layer for this PCB. Layer 2 is $0.035\ mm$ thickness with a weight of 28.3 grams and $16090.9\ mm^2$. In between the 2 layers, we a dielectric layer with a thickness $0.032\ mm$. After 3 iterations for the PCB Design,we finalized on the third iteration being the one we ordered from JLCPCB. The PCB arrived within one week of ordering. The finalized board dimensions are $146.2mm \times 79.8mm$, with 17 components being successfully populated onto the board.

Figure 7.4 3D model of the PCB:

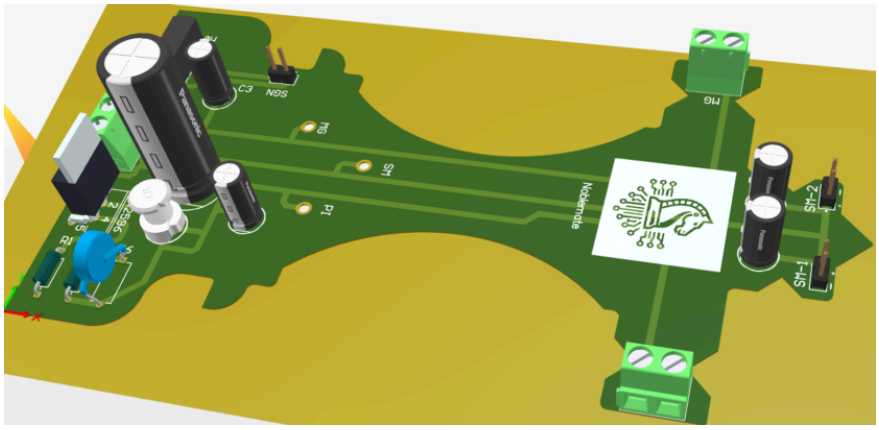


Figure 7.5 The Actual PCB:

Figure 7.6 PCB with components populated:

8. Regulatory Codes

A. Electrical Safety Standards
    a. UL 60950-1 / UL 62368-1
        i. These two standards are used for consumer electronics and IT equipment
        ii. This is to protect against shock, overheating and fire hazards
B. Electromagnetic Compatibility (EMC)
    a. ICES-003
        i. This is required for devices that emit electromagnetic waves
        ii. This ensures that the electromagnet and other devices do not interfere with other devices
C. Hazardous Substances & Environmental Compliance
    a. RoHS
        i. For global sales
        ii. It ensures that the devices are lead-free for sales
D. ASTM Standards for Wood Safety
    a. ASTM D1037
        i. Hardness, durability and resistance to wood splitting
        ii. Ensures that the wood enclosure does not split or crack over repeated uses
    b. ASTM F963-17
        i. Toy Safety for Kids
        ii. Requires wood to be smooth and splinter-free
        iii. Non-toxic finishes or coatings
E. Low-Voltage and Data Transfer/Communications Wiring Standards
    a. CSA C22.2 No. 210
        i. Ensures that the wires can withstand mechanical stress, heat and chemical exposure
        ii. Requires insulation to prevent fraying, cracking, or breakdown
    b. CSA C22.2 No. 49
        i. Cover low-voltage power cords for devices
        ii. Ensures flexible cables are protected from overheating and degradation
    c. CSA C22.2 No. 214
        i. Applies to signals and data wires
        ii. Ensures insulation and shielding to protect from electromagnetic interference
F. Choking hazards and small parts regulation
    a. ASTM F963-17
        i. Cover chess pieces, magnets and moving parts
        ii. Small parts must not detach during tests
        iii. A small piece is considered if it is smaller than 31.7mm

9. Alternatives Considered

Hardware Alternatives:

An alternative to the gantry system is a magnetic robotic arm designed to transport pieces efficiently. This system operates by first positioning the robot arm, either mounted on a mobile base or a fixed track, underneath the piece to be moved. Using cameras or sensors such LiDAR for distance and accurate positioning, the robot precisely aligns itself before extending its arm upward to engage an electromagnet or magnetic gripper, securely attracting the piece from below. Once lifted, the arm transports the piece to its designated location, utilizing articulated joints or a mobile platform for flexibility. Upon arrival, the electromagnet deactivates, releasing the piece into position before the robot resets for the next task. This approach offers flexibility compared to a gantry system, allowing for dynamic movement within a workspace while reducing mechanical complexity and space requirements. Additionally, it can be more energy-efficient, though challenges such as weight limitations and surface material dependency must be addressed through stronger magnets, hybrid gripping solutions, or vision-based alignment systems.

An alternative system for detecting chess piece locations involves using reed switches embedded beneath each square of the chessboard. Each chess piece would have a small magnet in its base, and when placed on a square, the magnet would close the corresponding reed switch, signaling the microcontroller that a piece is present at that position. The microcontroller continuously scans the switches, updating the board state in real time as pieces move. This approach offers low power consumption, high reliability, and immunity to electrical noise, making it a practical alternative to vision-based or RFID-based tracking. However, it has limitations, such as being unable to distinguish between different piece types, requiring precise magnet alignment, and involving complex wiring for 64 switches. To address these challenges, a multiplexing circuit could be used to reduce GPIO pin requirements, and Hall-effect sensors could be added to differentiate pieces based on magnetic field strength. Despite these challenges, a reed switch-based system provides a simple, durable, and efficient solution for chess piece location detection.

Wood was initially considered as an alternative for the chessboard surface, but it was quickly deemed infeasible due to several critical drawbacks, including its high coefficient of friction and potential magnetic interference. The increased friction could significantly hinder the smooth movement of the pieces, making their transitions less reliable and consistent. Additionally, wood surfaces are prone to slight warping or unevenness over time, which could introduce further inconsistencies and disrupt the precision required for the system to function effectively. Another major concern was the interaction between the wood and the magnetic field. The material could interfere with the strength of the magnet, potentially reducing its ability to attract and move the pieces accurately. In some cases, this interference might even weaken the magnetic pull to the extent that pieces fail to reach their designated positions. Given these challenges, wood was ultimately ruled out as a viable option for the chessboard surface.

Software Alternatives:

We were initially using Google Web Speech-to-text API and moved to Google Cloud Speech-to-Text v1 API. There are numerous tradeoffs with both models, where the Google Cloud API has created unexpected delays to the Stockfish interface and requires payment while the Google Web Speech API has a daily limit in the number of requests (limit is 50 requests/day).

As opposed to using Stockfish, the usage of a chess engine provided in the Instructables automated chessboard reference [1] was considered due to the simplicity of its implementation. All that would have been required to implement it was to provide a matrix of the chessboard with the piece position as opposed to converting the chessboard to a FEN string [8]. Ultimately, Stockfish was chosen because the way the code was structured made it easy to convert the board to a FEN string [8]. Stockfish is also a more powerful chess engine and is much easier to customize to the chess game. It also has many features that can be integrated into the project in the future.

Directly embedding and including Stockfish header and source files was initially considered but ultimately rejected as a solution because it would have made the implementation far more complex for the reward of more efficient operation of the code since the communication with Stockfish through Boost Process [7] would not have been required.

10. Future Work

Create a custom voice-to-text API/embed it directly into the system to avoid high API costs and to allow the chessboard to function with voice control without an internet connection.

To run the Stockfish algorithm more efficiently, program a custom Linux system for the microcontroller to only include the necessary dependencies. Additionally, use a microcontroller with more processing power to achieve a greater search depth within a reasonable amount of time for the Stockfish algorithm.

To minimize latency, all of the system-to-hardware functionality should be incorporated on one microcontroller. That is, instead of using the Raspberry Pi 3B for the chess algorithm, Voice API integration, Stockfish, and the Arduino for the LCD and motor control, integrate everything solely on the Raspberry Pi.

Allow the ELO to be adjusted more variably to allow for a more dynamic range of skill sets to compete with the chess computer algorithm. Instead of simply having an easy/medium/hard/insane mode, allow the user to say their ELO to play against that exact level of play.

Since this is defined to be a tool used to practice/train/learn the game of chess, include additional functionalities. These additional functionalities include but are not limited to adding a redo feature if the player wishes to undo a move, allow the player to go back and analyze the game once it is complete (which would require the storage of all previous moves in some short algebraic form) and including a game control meter to show which color has the advantage (can be realized by Stockfish).

11. References

[1] Instructables, “Automated Chessboard,” *Instructables*, Mar. 2022. https://www.instructables.com/Automated-Chessboard/ (accessed Apr. 03, 2025).

[2]“Raspberry Pi 3 B+ Power Supply - Raspberry Pi Forums,” *Raspberrypi.com*, 2018. https://forums.raspberrypi.com/viewtopic.php?t=228020 (accessed Apr. 03, 2025).

[3]“LM2596,” *Ti.com*, 2023. https://www.ti.com/product/LM2596?utm_source=google&utm_medium=cpc&utm_campaign=app-null-null-GPN_EN-cpc-pf-google-ww_en_cons&utm_content=LM2596&ds_k=LM2596&DCM=yes&gad_source=1&gclid=CjwKCAjw47i_BhBTEiwAaJfPpmrCxVqF6Bb0SYfDzyw2P4BH9e-5SQsLN_PNVi8jcv_pt-2FHiSLHRoCJZsQAvD_BwE&gclsrc=aw.ds (accessed Apr. 03, 2025).

[4]“What power supply can I use with my Arduino board?,” *Arduino Help Center*, Jan. 29, 2024. https://support.arduino.cc/hc/en-us/articles/360018922259-What-power-supply-can-I-use-with-my-Arduino-board (accessed Apr. 03, 2025).

[5]“Cloud Speech-to-Text API | Cloud Speech-to-Text Documentation | Google Cloud,” Google Cloud, 2023. https://cloud.google.com/speech-to-text/docs/reference/rest (accessed Apr. 03, 2025).

[6]”Stockfish Docs | Stockfish Development Setup” Stockfish https://official-stockfish.github.io/docs/stockfish-wiki/Developers.html#using-stockfish-in-your-own-project (accessed Apr. 04, 204).

[7]”Boost Process C++ Libraries | Chapter 29. Boost.Process” Boost Process https://www.boost.org/doc/libs/1_64_0/doc/html/process.html (accessed Apr. 04, 2025).

[8]”Forsyth-Edwards Notation” FEN https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation (accessed Apr. 04, 2025)

[9]”Code a Chess Game with Stockfish API - JavaScript Tutorial”, https://www.youtube.com/watch?v=fJIsqZmQVZQ (accessed Apr. 06, 2025)

[10]”StockfishOnline | Stockfish Rest API” Stockfish REST API, https://stockfish.online/ (accessed Apr. 06, 2025)

[11] “Stockfish Docs | UCI and Commands” Stockfish https://official-stockfish.github.io/docs/stockfish-wiki/UCI-&-Commands.html (accessed Apr. 07, 2025)

[12] Webench Power designer, https://webench.ti.com/power-designer/switching-regulator/customize/6 (accessed Mar. 2, 2025).

12. Appendices

Appendix A Images:

Figure 2.1, Predecessor's Automated Chessboard

Figure 3.1: Overall Block Diagram of the System

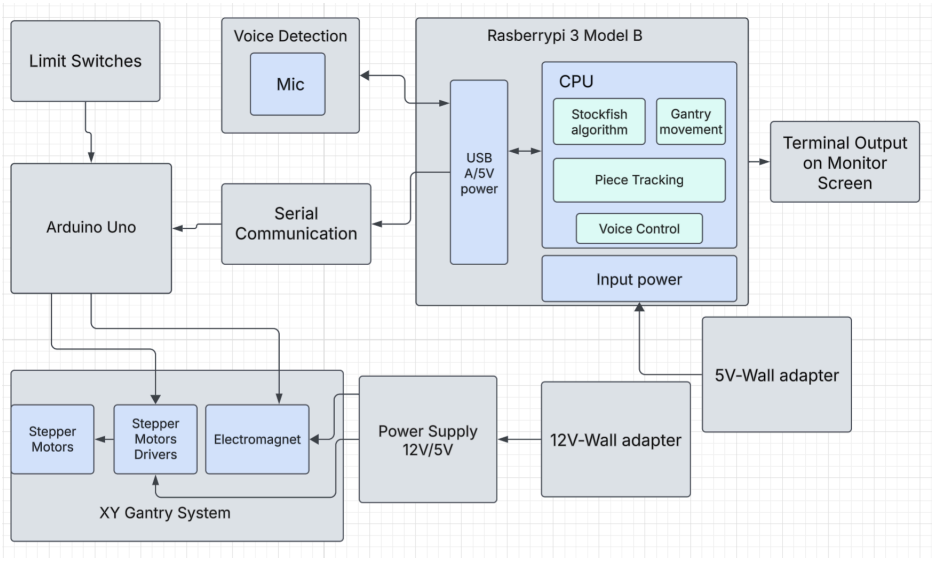


Figure 3.2: Overall Software Block Diagram to Demonstrate the connectivity of the modules

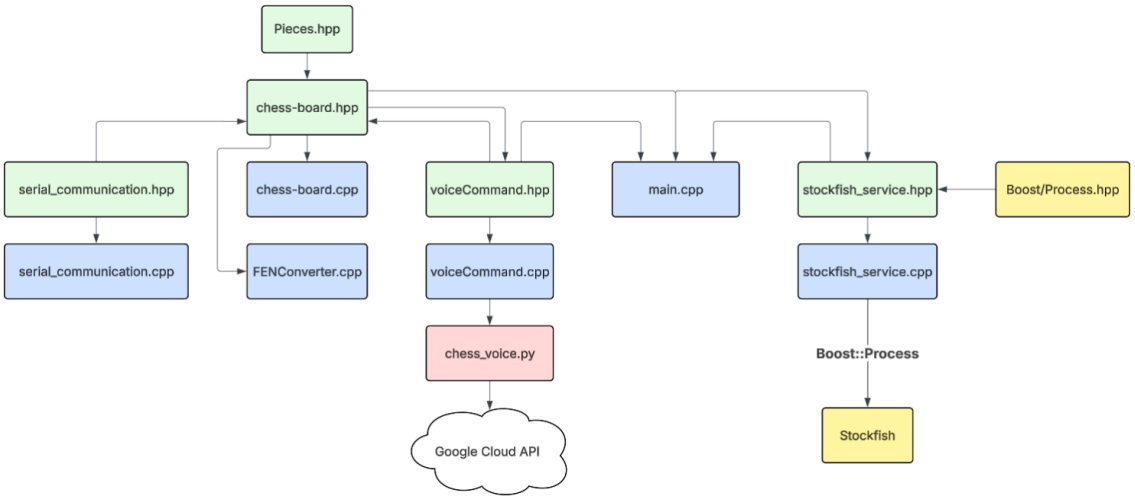


Figure 3.3: UML Class Diagram of the Software Showing Classes and Methods of Classes along with Interaction Between Files/Modules

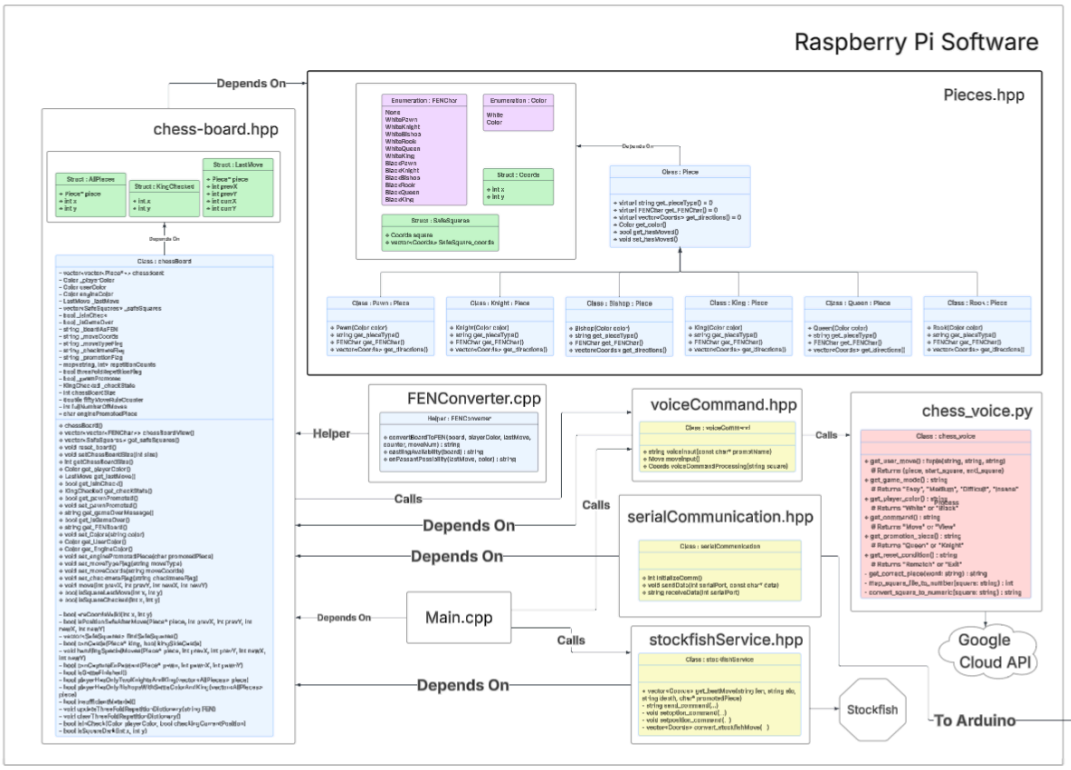


Figure 3.4: Gameflow Diagram Demonstrating the Relationship Between Functions Under the Assumption that the User Chooses White and all Moves are Valid

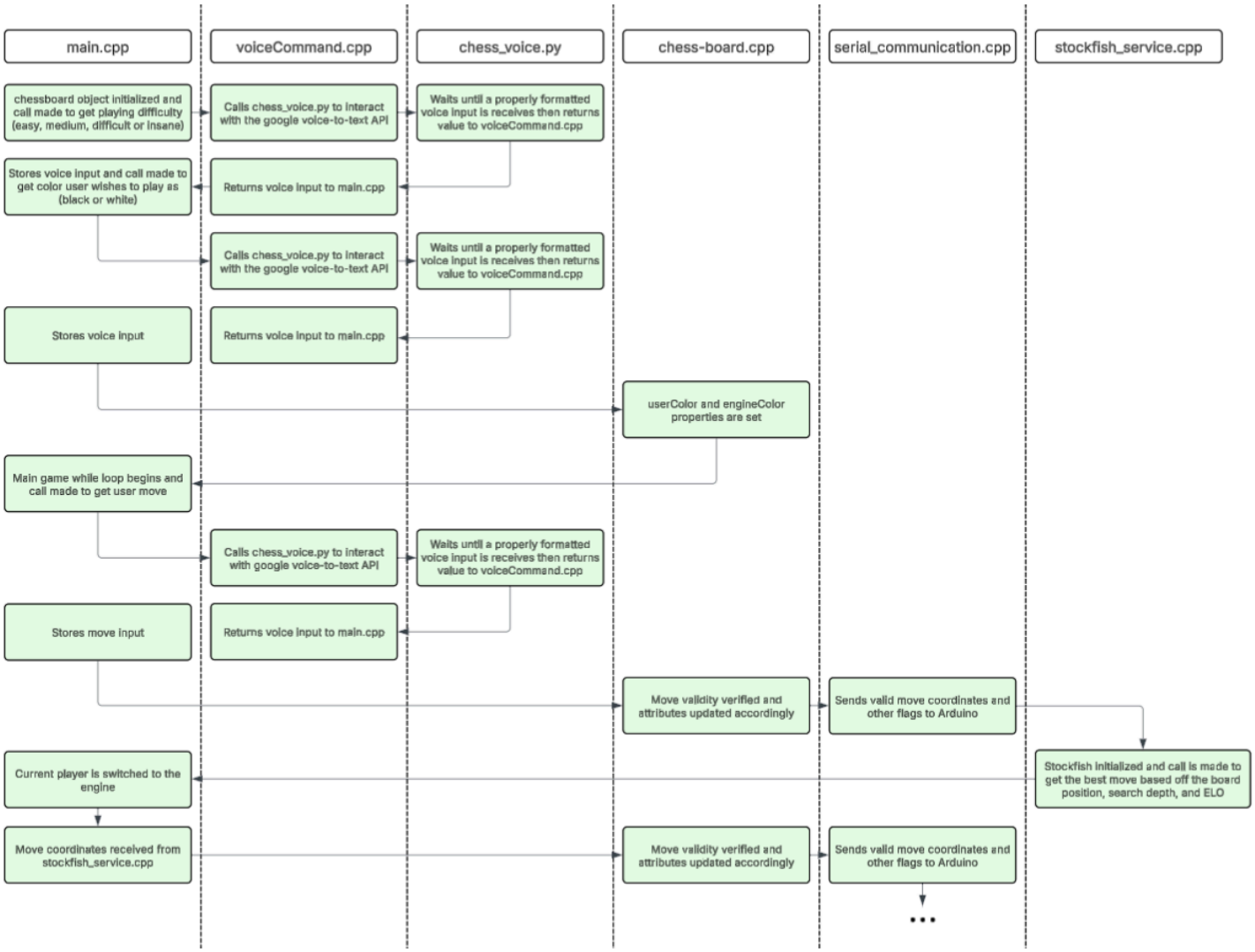

main.cpp
voiceCommand.cpp
chess_voice.py
chess-board.cpp
serial_communication.cpp
stockfish_service.cpp
chessboard object initialized and call made to get playing difficulty (easy, medium, difficult or insane)
Calls chess_voice.py to interact with the google voice-to-text API
Waits until a properly formatted voice input is receives then returns value to voiceCommand.cpp
Returns voice input to main.cpp
Stores voice input and call made to get color user wishes to play as (black or white)
Calls chess_voice.py to interact with the google voice-to-text API
Waits until a properly formatted voice input is receives then returns value to voiceCommand.cpp
Returns voice input to main.cpp
Stores voice input
userColor and engineColor properties are set
Main game while loop begins and call made to get user move
Calls chess_voice.py to interact with google voice-to-text API
Waits until a properly formatted voice input is receives then returns value to voiceCommand.cpp
Returns voice input to main.cpp
Stores move input
Move validity verified and attributes updated accordingly
Sends valid move coordinates and other flags to Arduino
Stockfish initialized and call is made to get the best move based off the board position, search depth, and ELO
Current player is switched to the engine
Move coordinates received from stockfish_service.cpp
Move validity verified and attributes updated accordingly
Sends valid move coordinates and other flags to Arduino

Figure 4.1: Raspberry Pi 3 Model B System Diagram

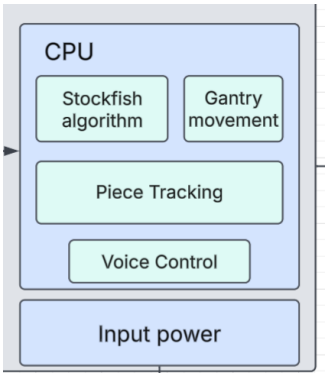


Figure 4.2: X-Y Gantry System Movement System Block Diagram

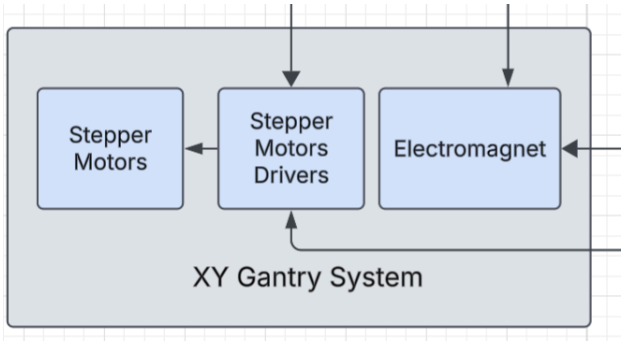


Figure 4.3: Terminal Output for HMI

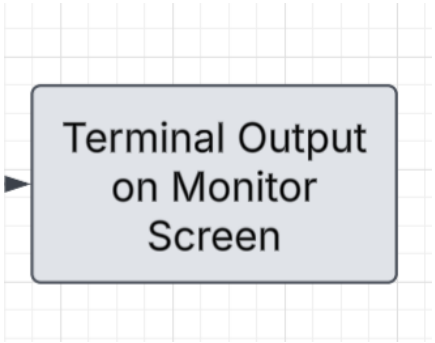


Figure 4.4: USB Type-A/5V power supply for different modules

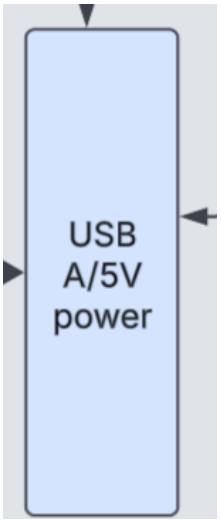


Figure 4.5: Arduino Uno System Block Diagram

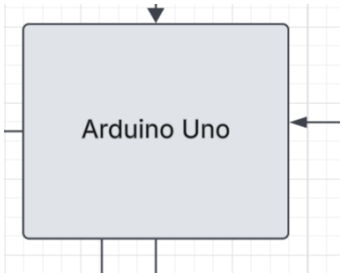


Figure 4.6: Limit Switches

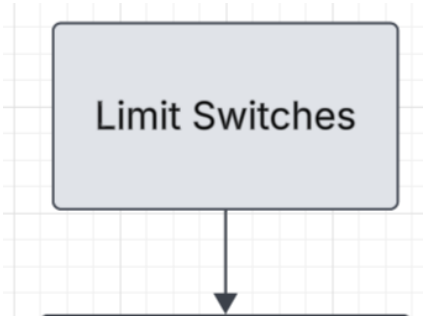


Figure 4.7: Microphone

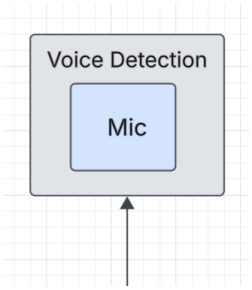


Figure 4.8: Stepper Motors

Figure 4.9: Electromagnet

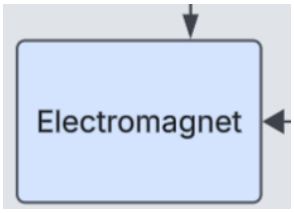


Figure 4.10: Stepper Motor Drivers to Control the Stepper Motors

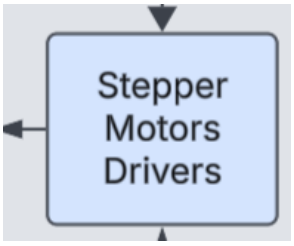


Figure 4.11: Power Supply for Distribution

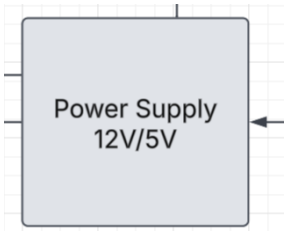


Figure 4.12: 12V Adapter for the Power Supply

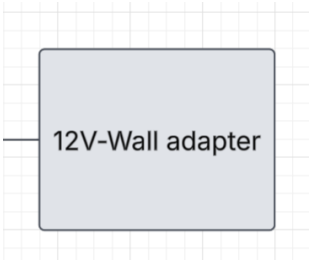


Figure 4.13: 5V Adapter for Powering the Raspberry Pi

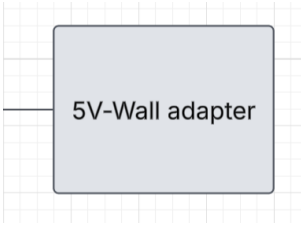


Figure 4.14: Pieces.hpp FIleDiagram

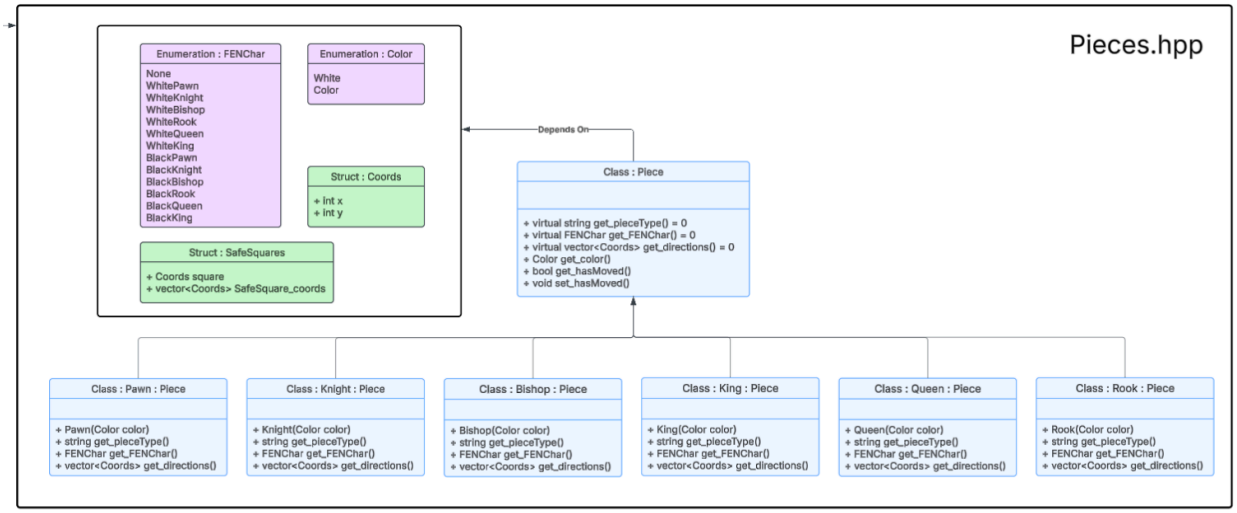


Figure 4.15: chess-board.hpp File Diagram

Figure 4.16: FENConverter.cpp File Diagram

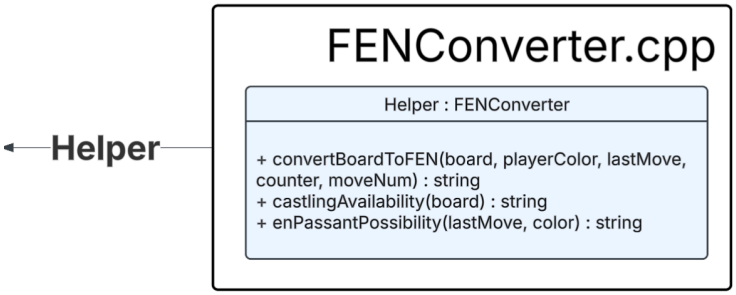


Figure 4.17: voiceCommand.hpp File Diagram

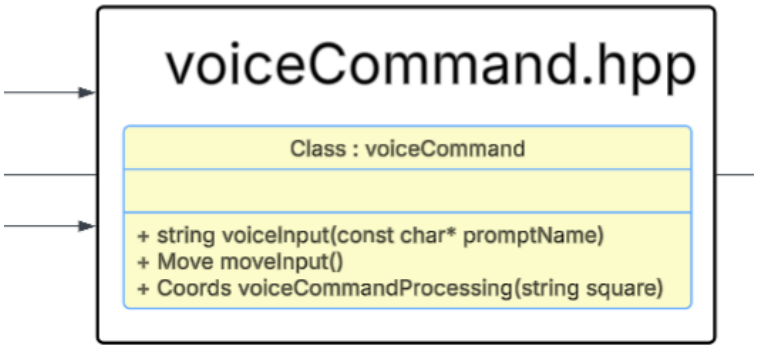


Figure 4.18 serialCommunication.hpp file Diagram

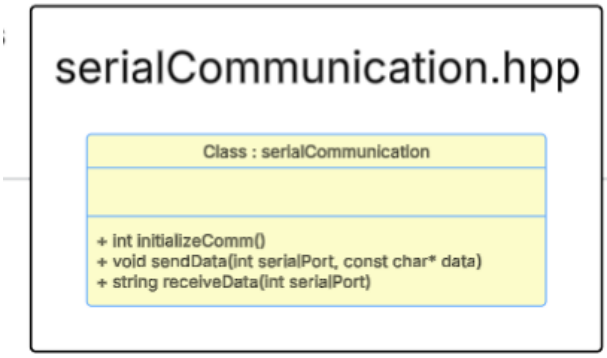


Figure 4.19: stockfishService.hpp File Diagram

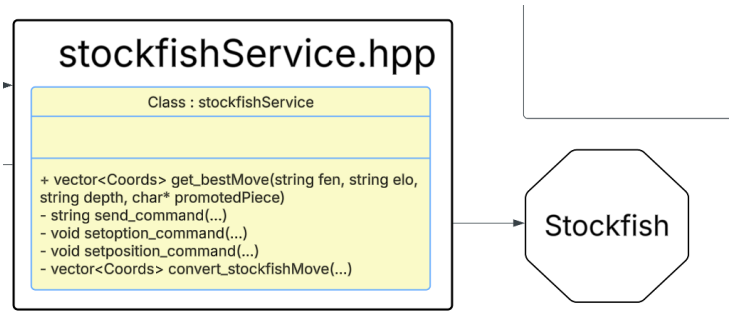


Figure 4.20: main.cpp File Diagram

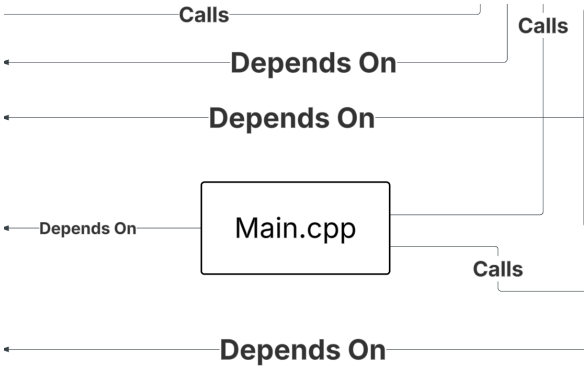


Figure 4.21: Chess_voice.py File Diagram

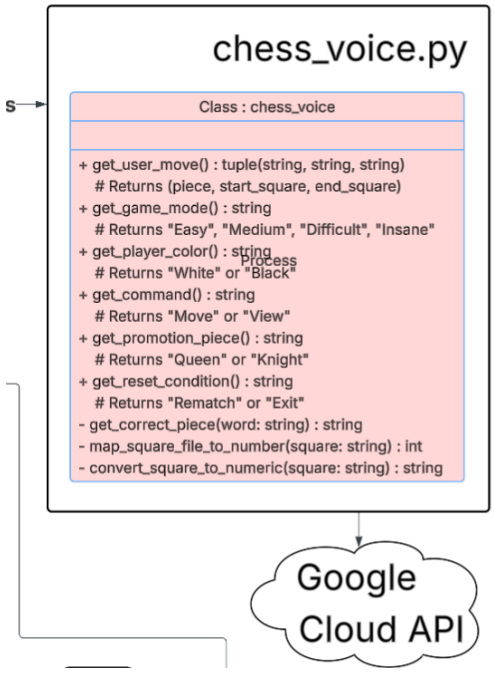


Figure 4.22 - Gantry.cpp

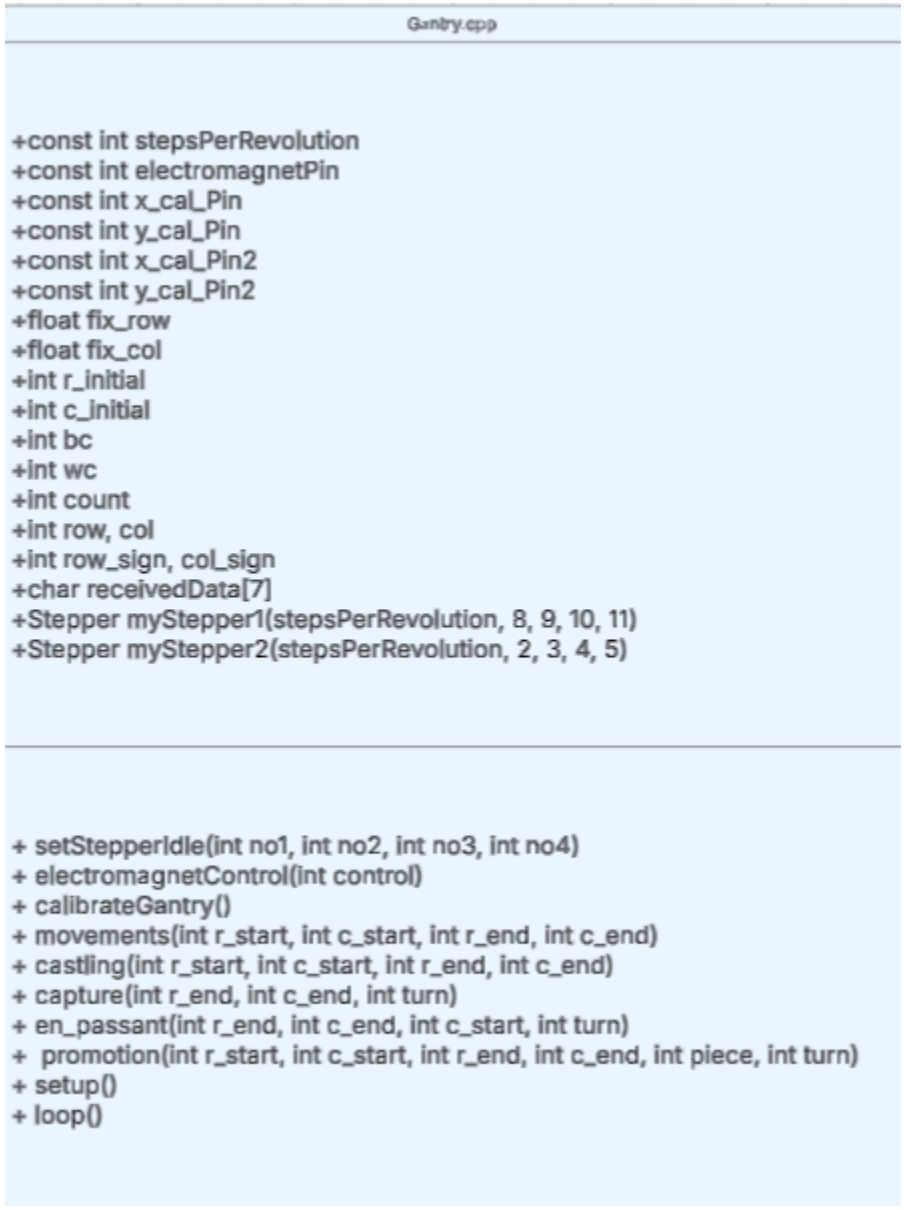


Figure 5.1: PCB Schematic Diagram

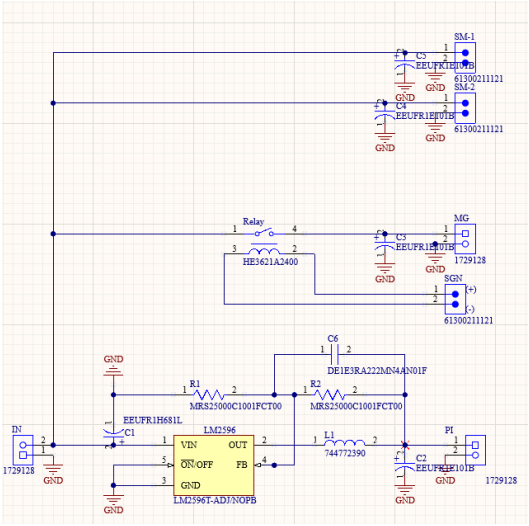


Figure 5.2: Inductance Chart for Respective Load

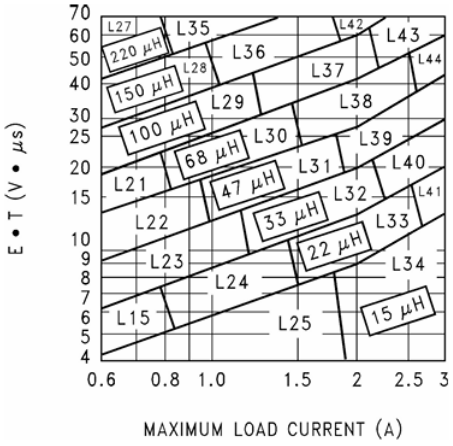


Figure 9-8. LM2596-ADJ

Figure 5.3: Part Numbers for Inductor

Table 9-1. Inductor Manufacturers Part Numbers (continued)

	INDUCTANCE (µH)	CURRENT (A)	SCHOTT		RENCO		PULSE ENGINEERING		COILCRAFT
			THROUGH-HOLE	SURFACE-MOUNT	THROUGH-HOLE	SURFACE-MOUNT	THROUGH-HOLE	SURFACE-MOUNT	SURFACE-MOUNT
L33	22	3.10	67148390	67148500	RL-1283-22-43	—	PE-53933	PE-53933-S	DO5022P-223
L34	15	3.40	67148400	67148790	RL-1283-15-43	—	PE-53934	PE-53934-S	DO5022P-153
L35	220	1.70	67144170	—	RL-5473-1	—	PE-53935	PE-53935-S	—
L36	150	2.10	67144180	—	RL-5473-4	—	PE-54036	PE-54036-S	—
L37	100	2.50	67144190	—	RL-5472-1	—	PE-54037	PE-54037-S	—
L38	68	3.10	67144200	—	RL-5472-2	—	PE-54038	PE-54038-S	—
L39	47	3.50	67144210	—	RL-5472-3	—	PE-54039	PE-54039-S	—
L40	33	3.50	67144220	67148290	RL-5472-4	—	PE-54040	PE-54040-S	—
L41	22	3.50	67144230	67148300	RL-5472-5	—	PE-54041	PE-54041-S	—
L42	150	2.70	67148410	—	RL-5473-4	—	PE-54042	PE-54042-S	—
L43	100	3.40	67144240	—	RL-5473-2	—	PE-54043		—
L44	68	3.40	67144250	—	RL-5473-3	—	PE-54044		—

Figure 5.4: Graph of the Output Voltage (Right) and Current (Left) from the LM2596T-ADJ

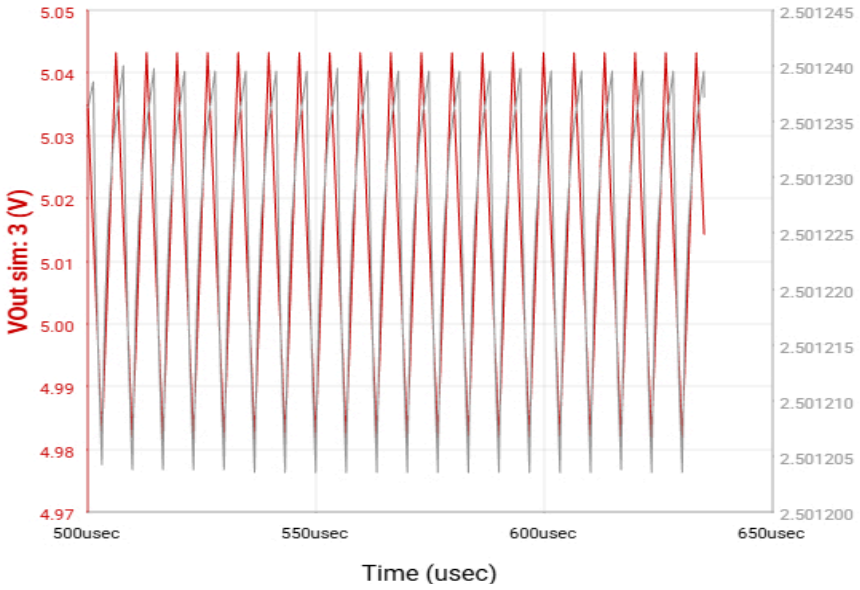


Figure 5.5: Graph of the Output Voltage (Right) and Current (Left) from the LM2596T-ADJ under load

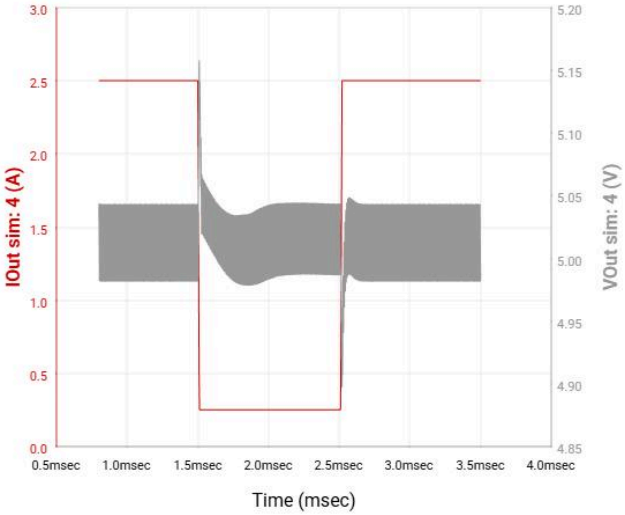


Figure 5.6: LM2596-ADJ Startup Simulation

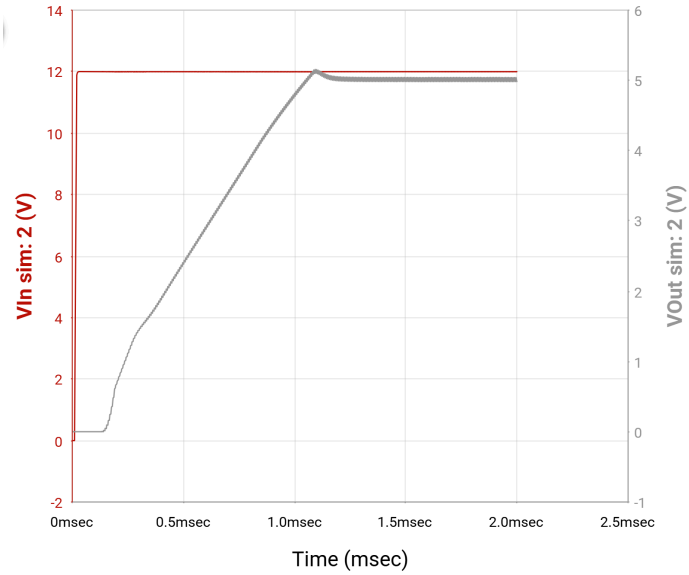


Figure 7.1: Circuit Schematic for the PCB

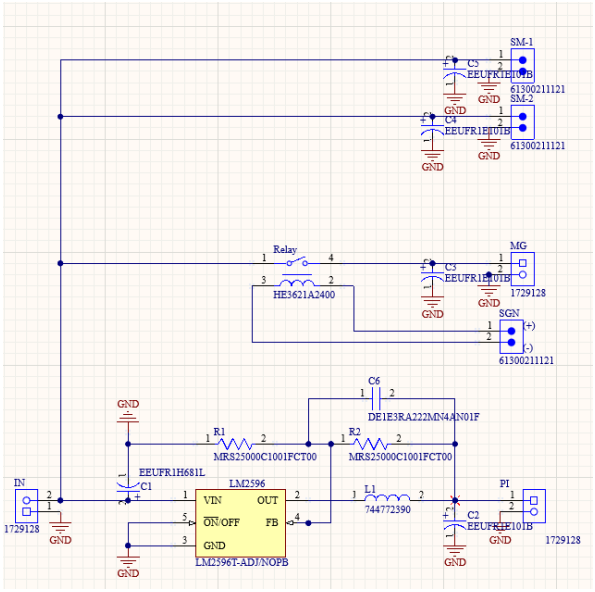


Figure 7.2: PCB Layer 1

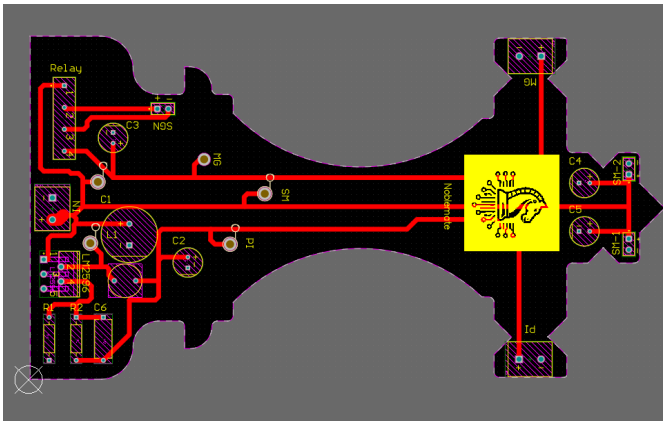


Figure 7.3: PCB Layer 2

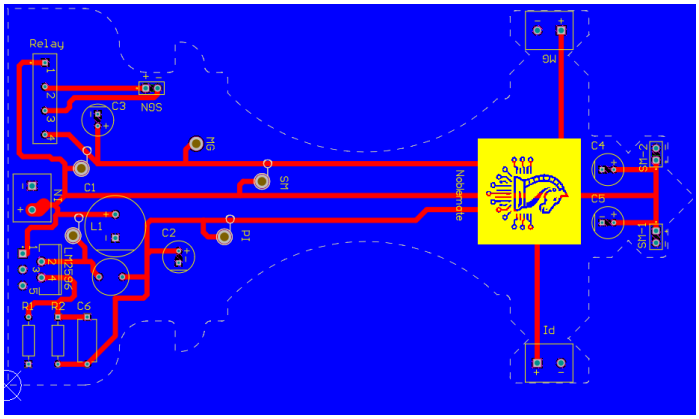


Figure 7.4: 3D model of the PCB

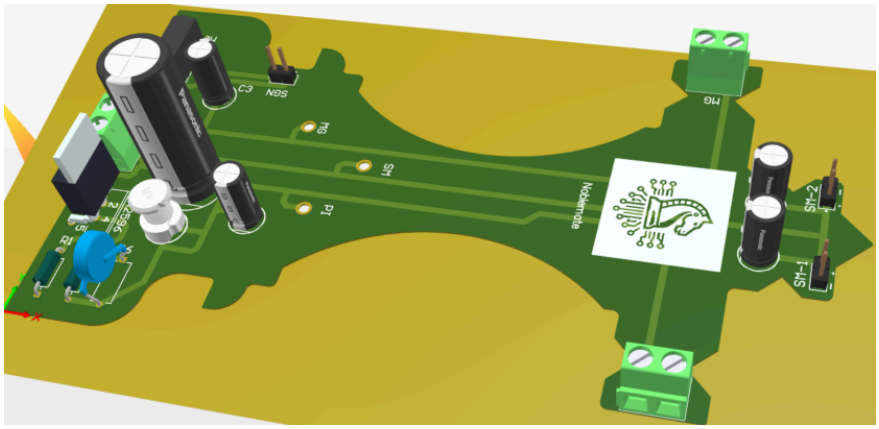


Figure 7.5: The Actual PCB

Figure 7.6: PCB with components populated

Appendix B Bill of Materials:

Item	Price in CAD($)	Quanitity	Total Price in CAD ($)	Total in CAD($)
Linear Rails (Pair of Three), Belt, Pully Wheels	120.95	1	120.95	
Stepper Motors(Pair of Two)	31.49	1	31.49	
Belt	11.54	1	11.54	
Reed Switches(Pack of 64)	12.59	1	12.59	
Electromagnet	17.84	1	17.84	
12mm by 3mm magnets(40 pieces)	15.51	1	15.51	
12V/6A Power Supply	24.14	1	24.14	
LM2596T Chips(Pack 5)	21.29	1	21.29	
Assorted Inductors	12.66	1	12.66	
12mm by 3mm magnets(90 pieces)	18.66	1	18.66	
(20" by 30") Foam Board	7.34	2	14.68	
(24" by 38") Wood	13.46	1	13.46	
Lapel Microphone	12.99	1	12.99	
I2C Adapter(Pack of 3)	9.99	1	9.99	
Wood Screw(100 pieces)	6.28	1	6.28	
PCB(Pack of 5)	38.61	1	38.61	
M3 Screws(10 pieces)	9.20	1	9.20	
				391.88

Figure 12.1.1: Total Bill of Materials

The total cost of the materials is approximately $283.53. This price was calculated by accounting for only 36 total magnets, 1 LM2596T chip, 2 inductors, 1 I2C Adapter and 1 PCB.

Item	Price in CAD($)	Quanitity	Total Price in CA	Total in CAD($)
Linear Rails (Pair of Three), Belt, Pully Wheels	120.95	1	120.95	
Stepper Motors(Pair of Two)	31.49	1	31.49	
Belt	11.54	1	11.54	
Electromagnet	17.84	1	17.84	
Magnets(90 pieces) - used 32	10.93	1	10.93	
12V/6A Power Supply	24.14	1	24.14	
LM2596T Chips(Pack 5) - used 1	4.26	1	4.26	
Assorted Inductors(100 pieces) - 2 used	0.25	1	0.25	
(20" by 30") Foam Board	7.34	2	14.68	
(24" by 38") Wood	13.46	1	13.46	
Lapel Microphone	12.99	1	12.99	
I2C Adapter(Pack of 3)	3.33	1	3.33	
Wood Screw(100 pieces) - used 12	0.75	1	0.75	
PCB(Pack of 5) - used 1	7.72	1	7.72	
M3 Screws(10 pieces)	9.20	1	9.2	
				283.53

Figure 12.1.2: Total Bill of Prototype

Our initial budget was set at $420.00, which provided flexibility for developing the NobleMate game. Ultimately, our total expenditure came in slightly under budget, with a variation of $28.12, placing us close to our budget limit. The slight difference between our

original projected cost and the final total is primarily due to removing multiplexers, which we had initially planned to include in the design.

Appendix C Schedules:

Original Gantt Chart for Schedules:

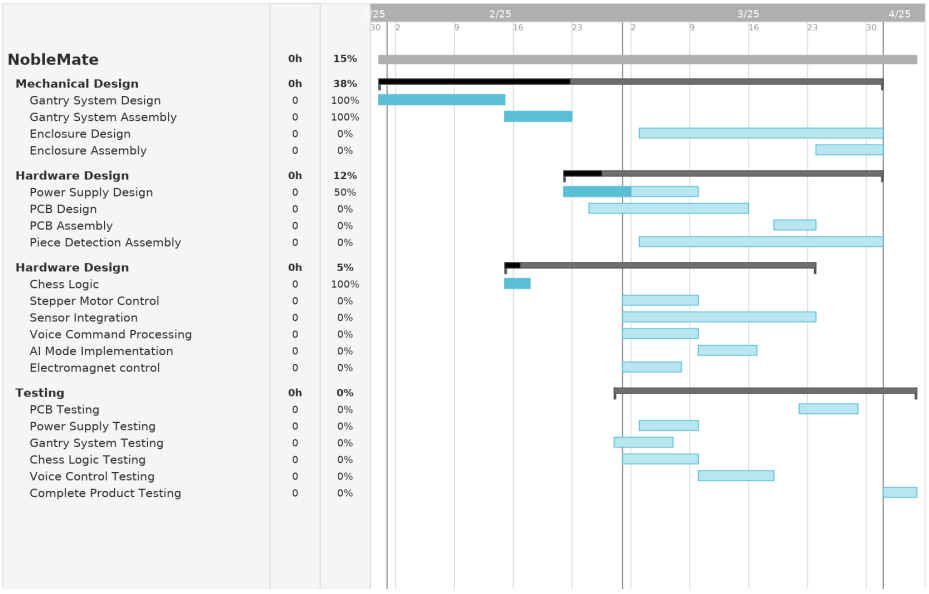


Figure 12.2.1 Original Gantt Chart for Scheduling

Updated Gantt Chart

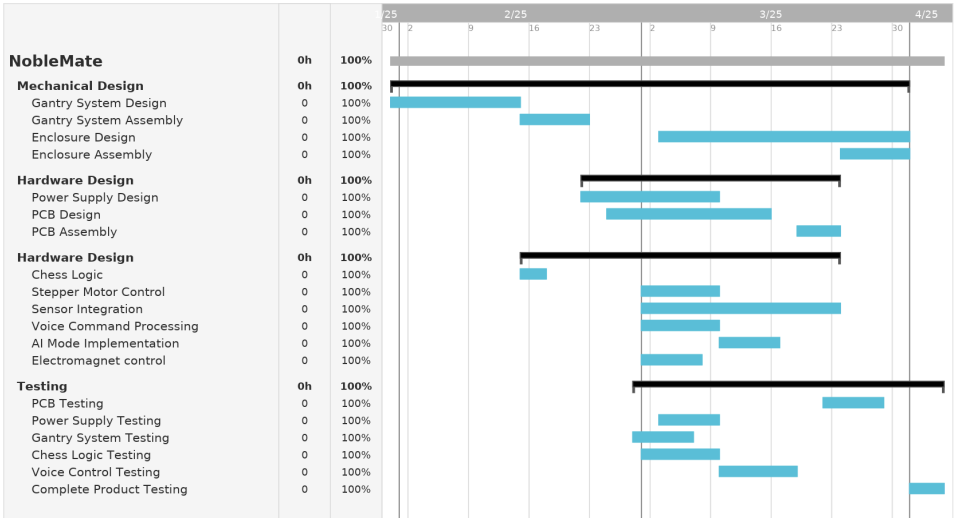


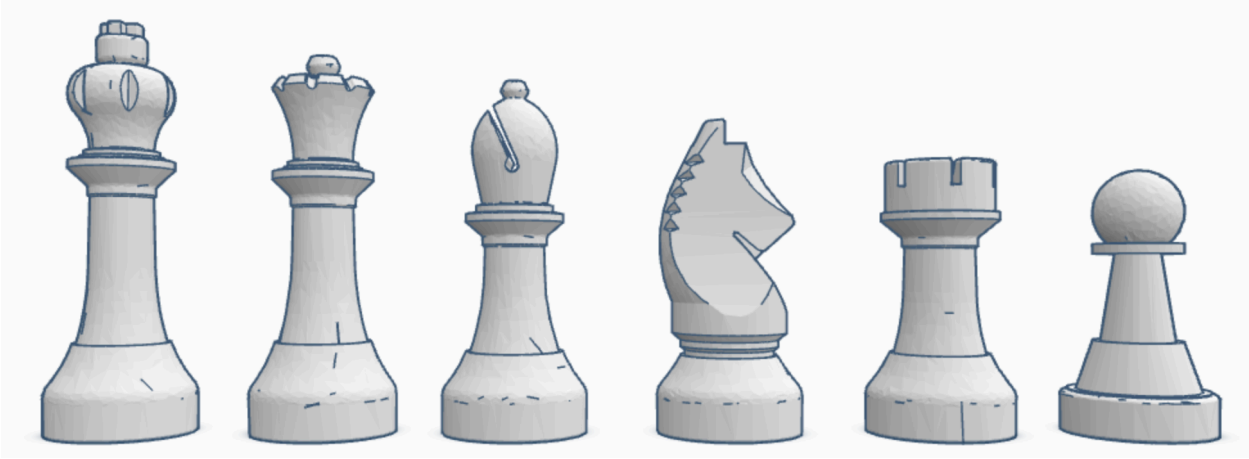
Figure 12.2.2

The variation between the original and the updated schedule is due to the removal of the piece detection assembly. The piece detection was not pursued because a more complex chess piece movement system was used. The original timelines were met as per the schedule except for the AI Mode Implementation and Enclosure Assembly, which in turn delayed our complete product testing. The original plan was to finish the AI mode implementation on March 16, which was delayed by nearly a week. It was completed on March 22. The duration of the AI mode implementation was underestimated. The enclosure assembly took two days longer than originally planned due to an error in trimming the board sizes to fit the gantry system. The overall testing of the system was delayed due to the delay in the enclosure assembly and responsibilities in other classes. Additionally, the additional focus on documentation also caused a delay in the overall testing, which was completed on the 8th of April.

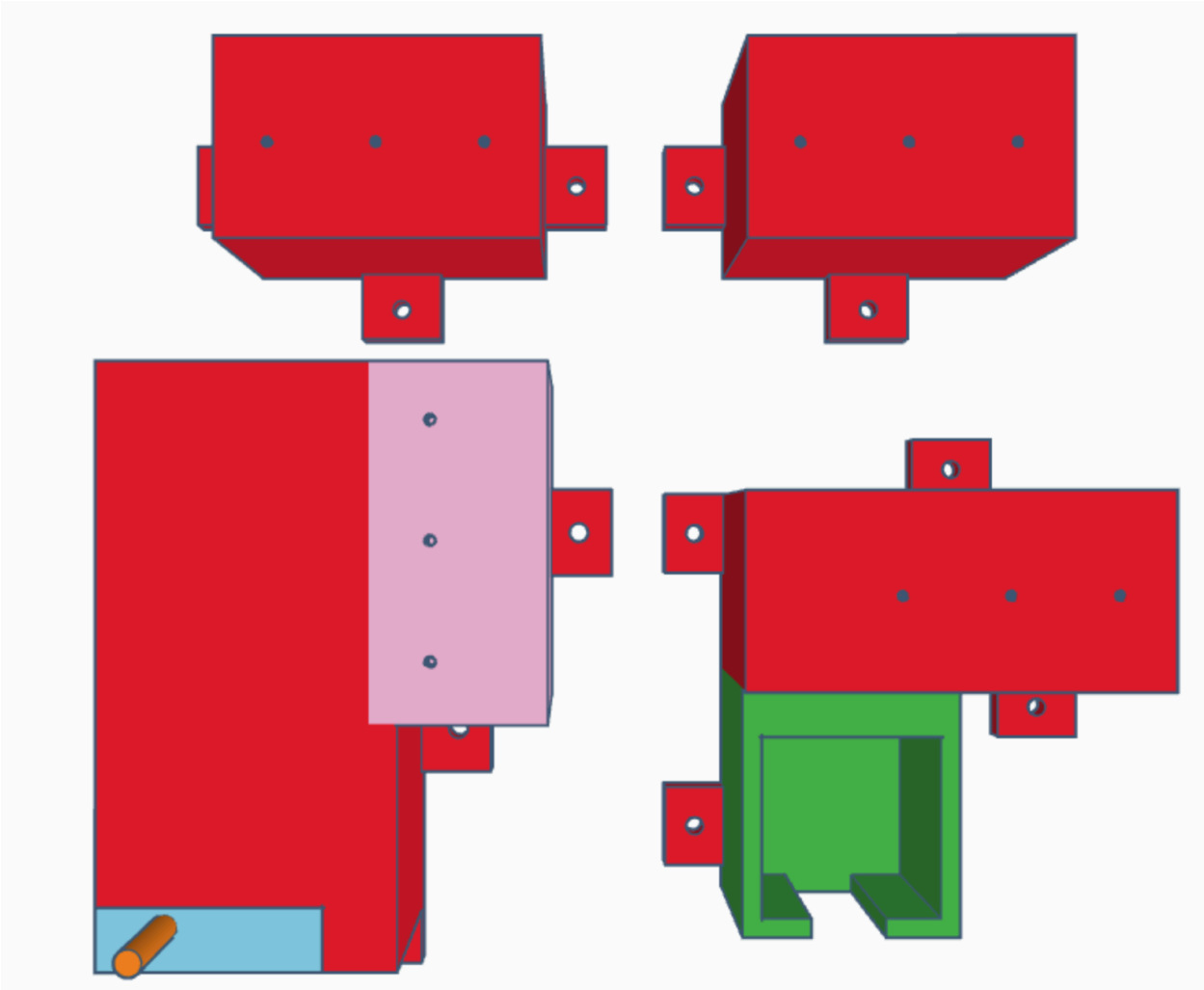
Appendix D Pictures of Prototypes:

Appendix E 3D Printed Parts:

Chess Pieces:



Gantry Support Blocks:



Middle Rail Supports: