Smart Chess Board

Final Report

Team 3

Austin McCormick

Vivek Amarnani

Rose Chakraborty

Department of Computer Science and Engineering

Texas A&M University

May 1st, 2024

Table of Contents

1 Executive summary

This report summarizes the development of a smart chessboard prototype. The project aimed to create a chessboard that bridges the gap between traditional chess and the possibilities offered by technology, enhancing accessibility, enabling game analysis features, and potentially serving as a learning platform.

Chess is a lucrative industry valued in the billions, attracting millions of players. However, the majority of players engage exclusively in online play. This heavy reliance on digital platforms has made it challenging for young players to enter the game safely and responsibly. Despite the abundance of online resources like chess platforms and tutorials, physical learning tools for chess are scarce. This creates a clear market gap for an educational chess tool that combines the accessibility of digital chess with the tangible, authentic experience of a physical chessboard.

The design strategy aims to steer clear of generic solutions, instead focusing on creating an interactive and effective learning tool. By incorporating an AI-powered opponent, a set of LEDs to indicate rules and game status, and a user-friendly interface and setup, the smart chessboard will elevate the chess experience and learning curve. In developing this product, the team will integrate a microprocessor to enable these functionalities, striving for a system with extended battery life, sturdy and eco-friendly materials, and features that cater to young players' needs.

The final design utilizes a grid of 64 Hall effect sensors under the board to detect magnets attached to chess pieces. Multiplexers channel data to a Raspberry Pi which interprets it to determine the board state. The software utilizes Python and the Python-Chess library for chess logic and explores integration with LED lighting systems for user interaction. The board's software creates a digital 'handshake' between the matrix of Hall effect sensors and the matrix of LEDs to create a seamless combined system. The board was created over the course of one semester and within a budget of $200.

Testing focused on four aspects: individual sensor functionality, chess piece detection and movement tracking, chess logic validation, and overall system performance.

Individual sensor functionality proved successful in controlled settings, but due to magnetic interference during gameplay, there were a few error readings. Chess piece detection and movement tracking achieved 70% accuracy during controlled testing, but real-world scenarios with less precise placement led to misidentification. The implemented chess logic accurately handled documented test positions for accurately detected pieces. Overall system performance demonstrated a response time of less than 1 second for sensor reading and move validation, ensuring a smooth playing experience in controlled settings. In the future, the board would need to be larger to accommodate for the magnetic field of each of the sensors.

2 Project background

According to the Chess in the Olympics campaign, there are 605 to 700 million adults that play chess regularly, making up roughly 8.6% of the world's population. Despite this, data from the United Nations indicates that over 70% of surveyed adults (US, UK, Germany, Russia, India) have played chess at least once in their life. Chess is already a 2.2 billion dollar industry, according to barchart.com, yet despite how profitable the market is, there is low retention between people trying chess and active chess players.

Current chess learning methods often rely on instructors or computers, creating barriers for beginners who lack access, prefer physical interaction, or dislike screen time. The elderly community in particular may feel excluded from enjoying the strategic and social benefits of chess due to technological challenges. It is necessary to propose a learner's chess prototype that is easy for beginners, children, and elders to learn and train in chess in an easy to use way and bridge the gap between the high exposure of chess compared to the low retention of chess players.

Our Smart Chessboard is a computer-aided educational tool with an embedded system specifically meant to enhance the user experience and create an interactive learning environment. Our team designed, implemented, and delivered a multi-featured physical prototype chess board, specifically developed with young novice chess players in mind. Our chess board specifically addresses the difficulty of teaching young players the complicated rules and strategies of chess while also keeping them engaged. We arranged 64 Hall Effect Sensors, with one sensor per chess square to detect the position and movement of our magnetic chess pieces. We used a software-focused implementation to combine our own custom-made algorithms to convert parallel sensor data coming from 64 separate Hall Effect Sensors to be processed by the Raspberry Pi, mapped to their own individual LEDs, and converted to a format that can be read by open-source libraries such as Python-Chess.

2.1 Needs statement

Young novice chess players and those seeking to improve their skills need an interactive, engaging, and educational tool because traditional methods of learning chess can be complex and intimidating. This tool should also provide an enhanced playing experience for all levels of players. This chess learning tool should bridge the gap between the accessibility of computer chess with the ease of use and authenticity of a physical chess board.

2.2 Goal and objectives

The main goal of this project is to create a smart chessboard that bridges the gap between the convenience and useful feedback of computer chess with the engaging and tangible feeling of a physical chessboard, offering personalized learning and enhanced gameplay through AI integration.

Here is a list of our objectives:

- Develop a user interface that accommodates diverse learning styles and provides interactive feedback during gameplay.
- Offer AI opponents with adjustable difficulty levels and hint functionalities to suggest potential moves.
- Each tile is equipped with LEDs to visually convey the game state, highlight potential moves, and enhance player engagement.
- Integrate a microprocessor to manage LED lights, interact with chess software, and enable analysis functions.
- Use illuminated tiles to highlight legal moves, potential threats, and check/checkmate situations.
- Offer long battery life for extended play sessions, prioritizing durability to withstand regular use.
- The cost of constructing the system should not exceed $200.
- Use durable materials and construction to withstand regular use.
- Create a stimulating and visually appealing gameplay environment with customizable features.
- Ensure ease of use and accessibility for different user groups. Users of the system should be able to learn 90% of the system’s functionality within 1 hour.

Out of the original objectives, we were able to achieve all of them except for integrating AI opponents with adjustable difficulty settings. This was due to the reduction in our team size to 3 members limiting the amount we could accomplish within the three month timeframe of the project. On top of this, the hardware implementation took up a significant portion of our time as sensors and multiplexers would randomly stop working, causing us to spend more time on debugging the hardware than expected. We also modified the long battery life objective by having our system use an AC power adapter instead. This was due to the high power consumption of the Raspberry Pi, LEDs, sensors, and Multiplexors.

2.3 Design constraints and feasibility

Cost under $200 - Due to the limitation of $50 allotted per group member, our team must design a solution within this $200 constraint. Since the Majority of competing products cost upwards of $500, cost effectiveness will be one of the primary factors setting our solution apart. Although we had to purchase more components than initially expected, we kept our budget under $200 as we had the Raspberry Pi provided to our team for free by the computer science department.

Size and Weight - Chess Square sizes should have the standard 2 to 2.5 inch side lengths. This is to keep the feel of the Smart Chessboard as similar as possible to a typical chessboard while allowing the system to maintain its portability.

Constraint to track pieces - Since there are 64 chess squares on a board and our budget is $200 and below, we need a method to track pieces without violating the size and cost requirements. We accomplished this by using Hall Effect sensors to detect when a piece is on a given square by attaching magnetic disks to teh base of each chess piece.

Power - The method to power our Smart Chessboard must not be a hindrance to the user. The system must be able to remain portable, and the need to power the system shouldn't become a hindrance. The current design must be able to supply power to all the sensors for piece tracking, 64 LEDs on a LED strip, an OLED screen, and the Raspberry Pi. We modified this constraint due to the high power consumption of the Raspberry Pi and the LEDs. Instead of using a rechargeable battery, we used an AC power adapter so our system would have plenty of power for any duration of time.

Processing Power - The algorithms used must be able to run off the processing power that a Raspberry Pi is capable of. A balance must be met between algorithm power and latency caused by computing. We were able to effectively achieve this with a response time of less than 1 millisecond between system input and the LED output.

Simplicity of Use - The Smart Chessboard is a consumer product, so the initial setup should be clear. Use of the system should be simple, with straightforward features and minimal inputs besides the piece sensors. Everything the user needs to use the Smart Chessboard should come with the system.

2.4 Literature and technical survey

- Chessnut Pro
    - Wooden board/pieces with piece recognition
    - Must connect to an app to play against an AI
    - Connects to chess.com to play against opponents online
    - LED indicators to show where opponents move
    - $900
- Phantom
    - Remote Play with Physical Board
    - Automatically moves pieces
    - Voice controlled option to move pieces
    - Connects to app to track moves, connects to Chess.com
    - Integrated Chess Engine
    - $670
- Square Off Grand Kingdom Set
    - Automatically moves pieces
    - Connects to app to track moves, connects to Chess.com
    - Adaptive built-in AI
    - Record Matches

    - $549
- GoChess Modern 4XR
    - Automatically Moves Pieces
    - LED indicators for recommended moves
    - Connects remotely over Chess.com
    - Can load puzzles and restore game states
    - $519
- eONE Online Chess Board
    - Required app, integrated with Chess.com
    - Transmits moves made for online play
    - LED indicators to show opponent's moves
    - integrated lithium-ion battery
    - No integrated AI
    - $215
- The DGT Smart Board
    - Connects to Chess Software on a PC
    - Connects to Chess.com for Online play
    - Broadcasts matches over the internet
    - No integrated AI
    - $750

The products similar to our proposed design that are available are expensive, typically serve as a physical extension to online play, and many don't come with an integrated AI. Our design will provide a cheaper alternative to the few mainstream alternatives on the market, while keeping simplicity and ease of use at its core. Unlike the majority of alternatives, users won't have to connect to a cellphone or a computer in order to use our Smart Chessboard, and they won't have to spend the average price of $600. Our design will be 1/3 of the average market price for a similar product, while focusing its functionality on being an all-in-one chess trainer.

2.5 Evaluation of alternative solutions

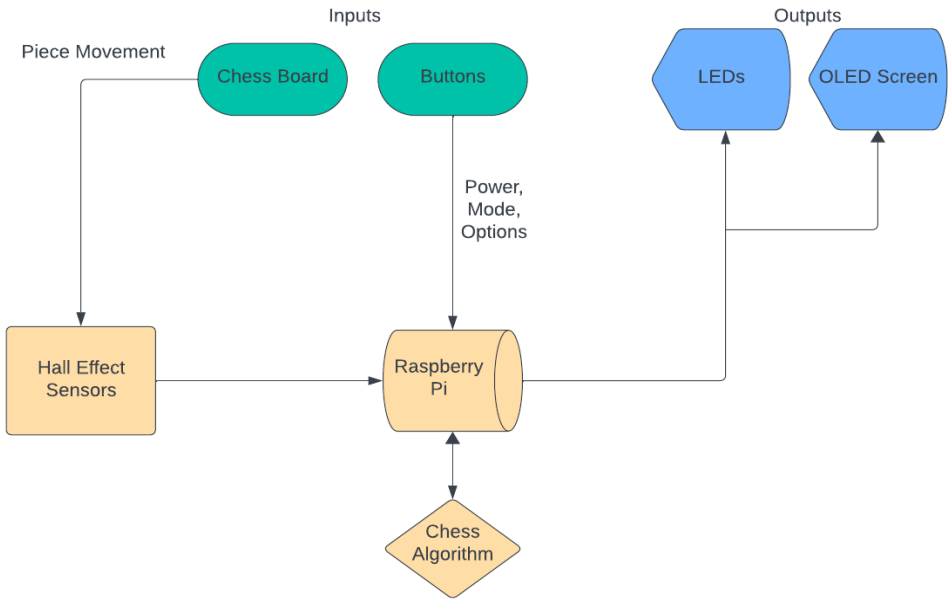
- Alternative solution 1 - RFID tags
- Alternative solution 2 - Magnetic sensors
- Alternative solution 3 - Pressure Sensors
- Alternative solution 4 - Camera detection
- Alternative solution 5 - Color Sensors
- RFID Tags:
    - Pros:
        - High accuracy: RFID tags uniquely identify each piece, enabling precise tracking.
        - Individual piece identification: Allows for advanced features like personalized move suggestions and piece history.
    - Cons:
        - Expensive: Cost of tags and reader adds significantly to overall price.
        - Requires embedded tags: Requires modification of existing chess pieces or creation of custom pieces.
        - Complex setup: Integration with reader and software needs careful planning and configuration.
    - Additional considerations:
        - Tag size and placement impact board aesthetics and playing experience.
        - Security considerations might be necessary to prevent tag cloning.

        -
- Magnetic Sensors:
    - Pros:
        - Relatively inexpensive: Sensors and magnets are generally affordable.
        - Detects metallic pieces accurately: Reliable detection under various conditions.
    - Cons:
        - Limited to metal pieces: Cannot detect non-metallic pieces like plastic or wooden ones.
        - Susceptible to interference: Other magnetic sources can disrupt detection.
    - Additional considerations:
        - Magnet strength and placement need careful calibration for reliable detection.
        - False positives from other metallic objects nearby might occur.
- Pressure Sensors:
    - Pros:
        - Direct piece presence detection: No need for tags or magnets attached to pieces.
        - Potential piece type identification: Pressure distribution might help differentiate piece types.
    - Cons:
        - High complexity: Integration and calibration of pressure sensors can be challenging.
        - Sensitive to wear and tear: Pressure sensitivity might degrade over time.
    - Additional considerations:
        - Sensor placement and board design need optimization for accurate pressure reading.
        - Cost and durability of pressure sensors might not be ideal for this specific project.
- Camera Detection:
    - Pros:
        - Versatile: Can detect both board and pieces using computer vision algorithms.
        - Potential advanced features: Image recognition allows for features like piece identification, move tracking, and augmented reality overlays.
    - Cons:
        - Processing power demands: Requires powerful hardware or cloud computing for real-time processing.
        - Affected by lighting and angles: Variations in lighting and camera angle can impact accuracy.
    - Additional considerations:
        - Computational optimization strategies are crucial for efficient processing on embedded devices.
- Color Sensors:
    - Pros:
        - Relatively inexpensive: Sensors and implementation are generally affordable.
        - Simple implementation: Basic setup and configuration compared to other options.
    - Cons:
        - Limited accuracy: Color variations and lighting dependence can lead to misidentification.
        - Unreliable for individual piece identification: Color alone is insufficient for distinguishing pieces.
    - Additional considerations:
        - Calibration for specific lighting conditions and board colors is crucial.
        - This option might be suitable for basic board detection but not for individual piece identification.

- Hall Sensors:
  - Pros:
    - Relatively inexpensive and simple: Sensors and magnets are affordable and easy to implement.
    - Non-contact detection: Avoids wear and tear on pieces.
    - Consistent accuracy: Unaffected by lighting conditions.
    - Potential for multiple-piece detection: Specific magnet choices can enable detection of metallic and non-metallic pieces.
  - Cons:
    - Requires small magnets attached to chess pieces, introducing some setup complexity.
    - Limited sensing range, requiring precise magnet placement and board design.
    - Might struggle with differentiating multiple closely spaced pieces on a crowded board.
  - Compared to other options:
    - Hall sensors offer a good balance between cost, complexity, and accuracy. They are more affordable and easier to implement than RFID tags or pressure sensors while providing better accuracy than color sensors.
    - They address some concerns of camera detection regarding cost and processing power.
    - However, they cannot identify individual pieces as readily as RFID tags or camera vision.

3 Final design

3.1 System description



The main inputs of the chess board prototype is the readings of the Hall effect sensors housed within the board. By circuiting individual Hall effect sensors to multiplexores and to each other, a large 8 x 8 matrix of sensors has been created on breadboards. Small magnets are attached to each individual chess piece on the board, and when the presence of a magnet is HIGH within the sensing field of the

Hall effect sensor, this information is communicated to the Raspberry Pi and used to detect piece movement. The additional input in the system is the buttons mounted adjacent to the OLED screen, positioned to the side of the playing surface and facing upwards, as shown in Figure 1 above. These buttons are used to perform basic actions within the user interface, such as selecting game modes and navigating menus as well as starting, stopping, and pausing the game.

The Raspberry Pi itself runs Python code using the Python-Chess library that determines which chess piece is picked up based on the sensor information and determines game information based off the moved piece. This include legal moves that the picked up piece can make as maintaining move order and signaling when an invalid move was made, showing the player where the piece was before the illegal move for them to fix the board state. The code will also interface with the Stockfish chess engine to serve as an AI opponent where the opponent's difficulty can be scaled.

The primary output of the system is the LED lighting array housed beneath the playing surface. A large LED lighting strip runs in a snaked pattern within the board's chassis walls. The strip is individually addressable and connected to one of the Pi's pulse width modulation pins. We initially planned for an additional output of an OLED screen.This OLED screen would be used to display game information and help users visually navigate through the user interface, but was revised and removed due to the time constraints and roadblocks we faced, such as our team size reducing and frequent issues with the large amount of small hardware components coupled with 64 magnets in close proximity causing issues.

The hardware, including the sensors, multiplexers, and LEDs is powered via a micro usb supply, providing the necessary 5v. The Raspberry Pi has its own separate dedicated power supply, which will provide ample power for the Python code to run on.

3.2 Complete module-wise specifications

Circuit and logic diagrams of piece detection matrix

Detecting Input - The circuit hardware of our system consists of 64 hall effect sensors arranged in an 8 by 8 matrix. There are 8 subsets of our circuit used to detect chess pieces. Each subset has 8 of the sensors connected to a multiplexor. Each sensor has a pull up resistor connected between its output and VDD, and the output of each multiplexor is fed into a voltage divider, where the second resistor is twice the value of the first. This drops the output signal of the multiplexor from 5v to 3.3v so that it is readable by the GPIO pins on the Raspberry Pi. The pinout of the Raspberry Pi can be seen in figure 6 below. The 8 multiplexor outputs are connected to pins 17, 27, 22, 14, 10, 9, 11, and 0, which are set to input, and the select lines are provided by pins 2, 3, and 4 of the Raspberry Pi GPIO, which are set to output. A schematic example of one of the eight subsets of the sensor board is shown in figure 2 below.

Pins 25, 8, 7, and 1 of the Raspberry Pi's GPIO are used as input and connected to mechanical switches, which in turn are used to signal the start of the match, display the most optimal move for the player, change the AI difficulty, and change game mode from player versus player to player versus AI.

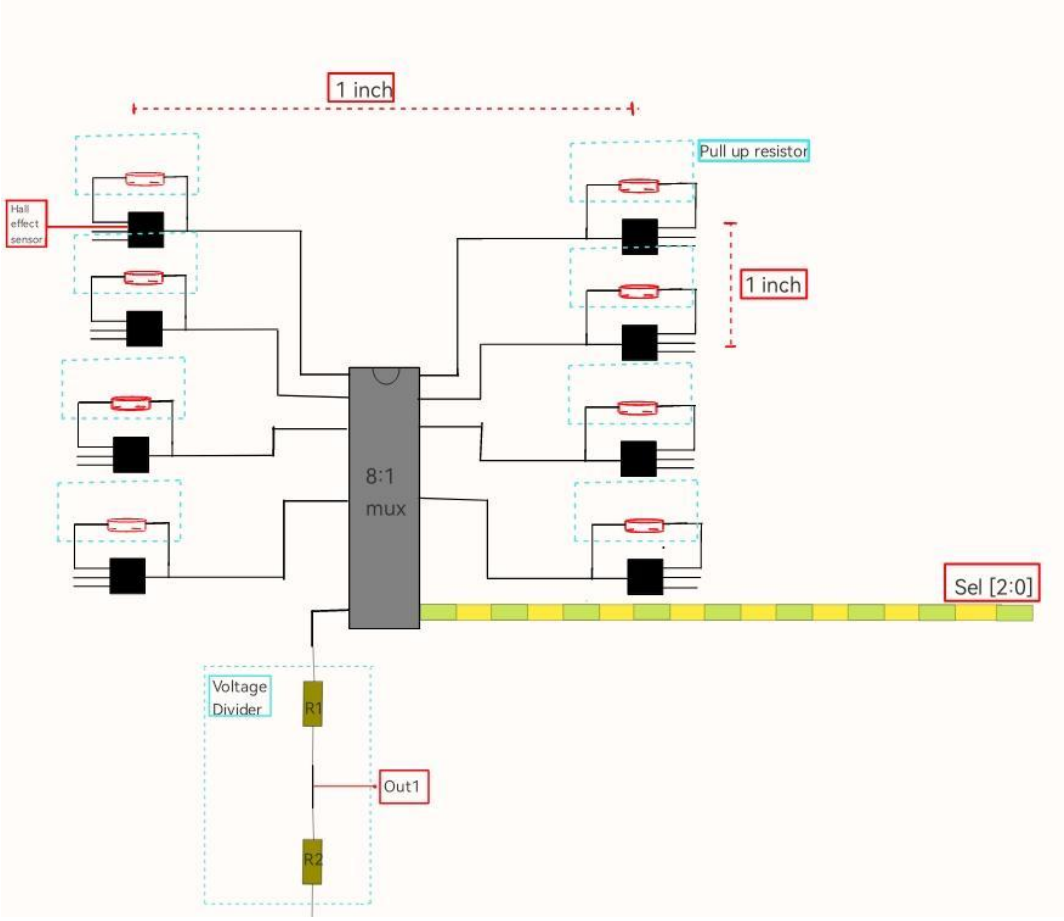


Figure 2: Diagram of sensor connection with multiplexor

Circuit and logic diagrams of LED lightning system

Pin 1 on the Raspberry Pi is set to output and is used to transmit data to the individually addressable LED strip. The LED strip are powered from a separate power supply than the 5v pin on the Raspberry Pi's pinout since the LEDs consume a current of roughly 4A and the max recommended current draw on the Pi is 3A not including the consumption of the Pi itself, which is 0.48A or the consumption of the 64 hall effect sensors, which is 0.544A, or the consumption of the multiplexers, which is 0.01A. We used a separate 5v AC power adapter to power the LED strip.

We revised the routing of the LED strip by skipping two diodes for programming after every 8 diodes that correspond with the Chess squares to avoid cutting and soldering the Strip at 8 different points. Each diode besides the skipped ones corresponds to a chess square. At the end of the strip, the three leads (VCC, GND, and OUTPUT) are connected to both the GPIO of the Raspberry Pi and the 5v power adapter. This configuration will be shown above in Figure 3. Each LED diode is mapped unique identifier in the program and individually assigned a color based on what is needed. We also revised the LEDs by routing them facing up and attached to a layer of plexiglass on the surface of the board instead of on the wall of a 3D printed Chassis since the 3D printed Chassis caused the playing surface to be too far away from the sensors. Laying the LEDs flat with a layer of plexiglass topped with canvas to hide the wires allowed the surface to be well within the effective range of the sensors, so all the sensors can detect the presence of pieces.

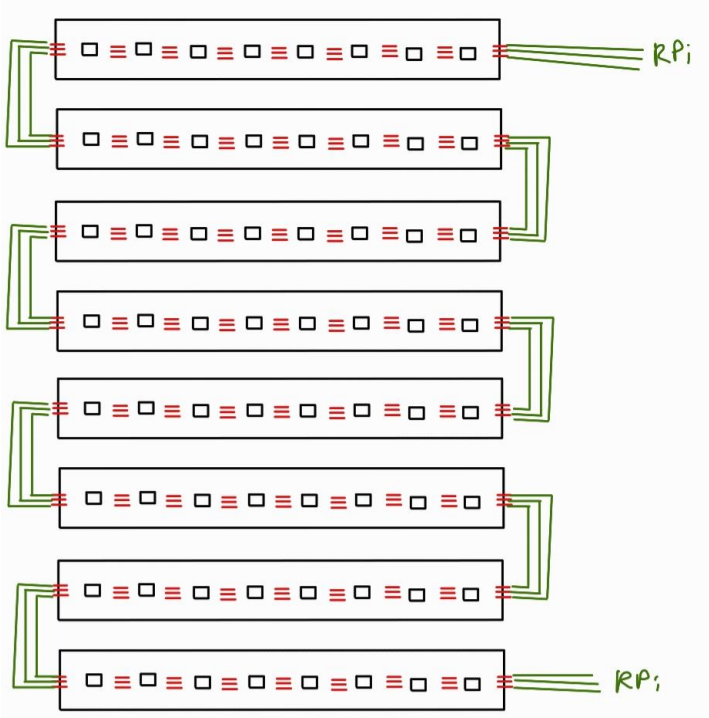


Figure 3: Wiring configuration of eight LED strips

Interfaces and pin-outs

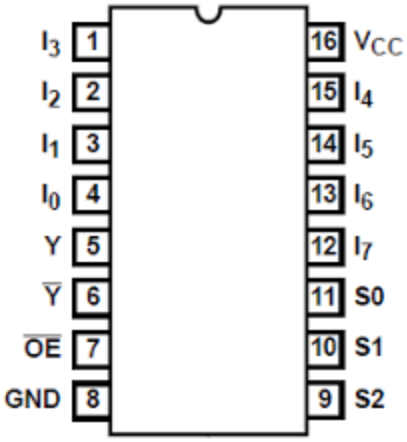


Figure 4: 8:1 Multiplexor Pinout

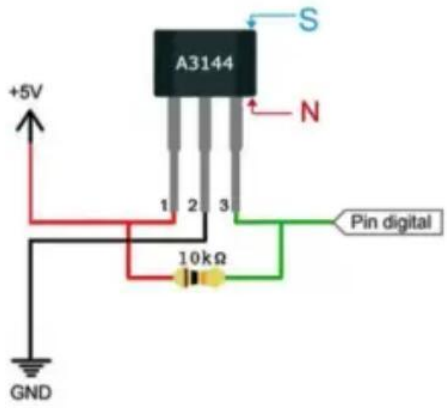


Figure 5: A3144 Hall Effect Sensor Pinout

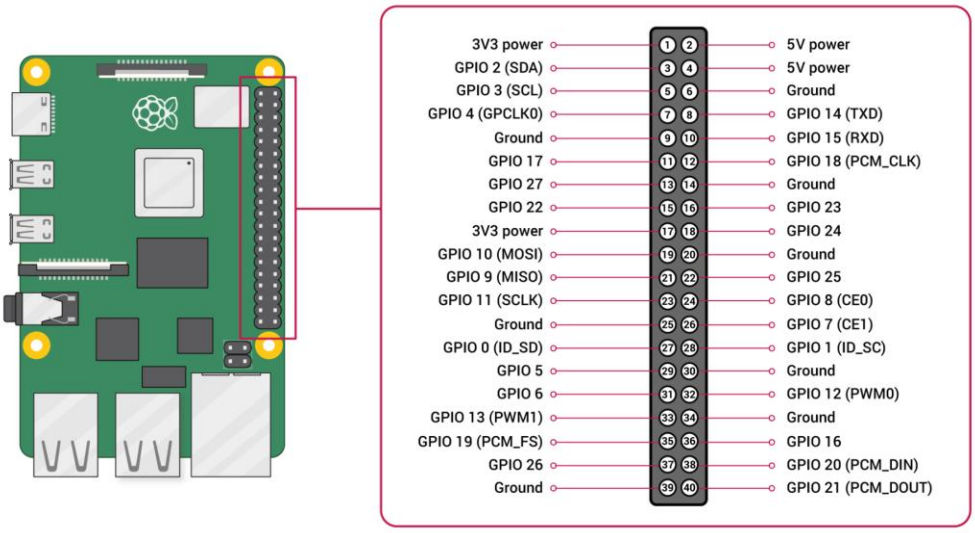


Figure 6: Raspberry Pi 3 Pinout

Timing diagrams and waveforms

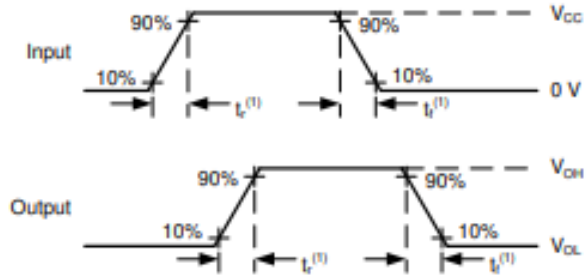


Figure 7: Timing delay waveform for CD74HCT251E 8:1 multiplexor (14nS delay)

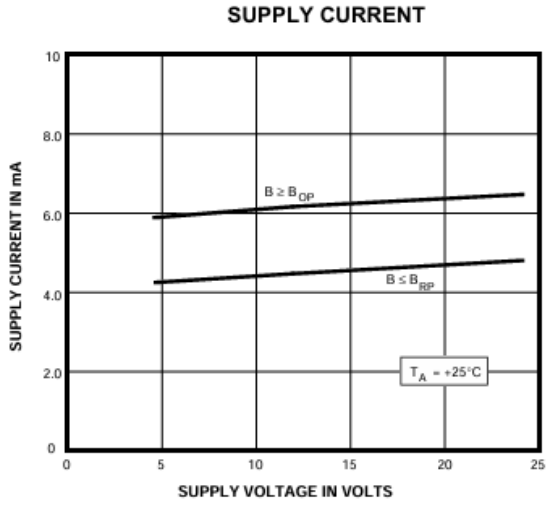


Figure 8: Plot of supply current versus voltage for A3144 Hall Effect Sensor

Software processes with their inputs and outputs

Algorithm 1: Sensor Data Conversion

Inputs: Sensor data from multiplexors
Outputs: Select Lines, 8x8 matrix of character type
Functions involved: read_sensor(), select_mux_channel(), swap_values(), print_8x8_matrix()

The purpose of this algorithm is to translate the binary serial data from the 64 sensors into a format that can be processed by the Raspberry Pi and fed into the chess API. The algorithm cycles through all 8 combinations that can be made by the three select line outputs, which in turn sequentially switches the output of each of the muxes between its 8 hall effect sensor inputs. For each mux, this data is converted into a 4x2 matrix, where the top left position corresponds to the sensor connected to pin I3 on the mux, the bottom left matrix position corresponds to pin I0, and so on so that the 4x2 matrix places the sensor date in the same structure as the sensors are physically placed. Each of these 4x2 matrices are then appended to each other in the order of pins 17, 27, 22, 14, 10, 9, and 11 on the GPIO board to construct an 8x8 binary matrix where each position corresponds to the physical position of the given hall effect sensor. When the first mechanical button is pressed, triggering pin 25, The initial stage of the match is set and the binary matrix is translated to a matrix of characters, where each character corresponds to a chess piece type. Figures 10 and 11 in the preliminary results section show the data being converted from the sensors into a binary matrix and then the binary matrix being converted to a char matrix at the start of a match.

Algorithm 2: Matrix to Forsyth-Edwards Notation Conversion

Inputs: 8x8 matrix of character type
Outputs: board representation as Forsyth-Edwards Notation
Functions Involved: chess_matrix_converter(), sensor_to_square(), generate_legal_moves()

The purpose of this algorithm is to convert the character matrix into Forsyth-Edwards Notation. This notation is what the python-chess API uses to process chess data and is what we will feed the API so that it can determine the legal moves that a given piece can make. This algorithm takes the binary matrix we created and converts it into a character matrix. This character matrix is then used to set the state of the board. When the main function detects a piece has been picked up, the sensor_to_square function outputs the chess square that changed, and the generate_legal_moves() function is called and returns a list of the possible moves for now. After we finish implementing the LEDs, these legal moves will be displayed through the LEDs as green light.

The current Python code for our Smart Chessboard can be found below. Algorithm 1 is fully implemented, and Algorithm 2 is partially implemented. For algorithm 2, we still need to develop functioning code that updates the board's state after a move is made and signals when an invalid move is made.

Python Code:

```
import RPi.GPIO as GPIO
import time
import chess

input_pins = [14, 22, 27, 17, 10, 9, 11, 0]  # Mux outputs
select_pins = [2, 3, 4]  # Mux select lines (S2, S1, S0)
start_button = 25

# Setup GPIO
GPIO.setmode(GPIO.BCM)
for pin in input_pins:
```

```
    GPIO.setup(pin, GPIO.IN)
for pin in select_pins:
    GPIO.setup(pin, GPIO.OUT)
GPIO.setup(start_button, GPIO.IN)

board = chess.Board()

# Invert signal since Hall effect sensors output 1 with no magnet and 0 with a magnet present
def read_sensor(input_channel):
    if GPIO.input(input_channel):
        return 0  # Assuming 0 means no magnet detected
    else:
        return 1  # Assuming 1 means magnet detected

# Function to swap between select line combinations
def select_mux_channel(channel):

    # Set select lines according to the channel number (0-7)
    for i, pin in enumerate(select_pins):
        GPIO.output(pin, channel & (1 << i))

def print_8x8_matrix(sensor_values):
    matrix = []

    # Construct the 8x8 matrix from sensor values
    for i in range(8):  # For each row
        row = []
        for sensor_group in sensor_values:  # For each sensor group
            row.extend(sensor_group[i*2:(i+1)*2])
        matrix.append(row)

    # Print the 8x8 matrix
    for row in matrix:
        print(row)
    print()

    return matrix

# Function to swap sensor values to resemble physical sensor layout per 4x2 chunk
def swap_values(chunk):

    # Swap top left with bottom left
    chunk[0], chunk[6] = chunk[6], chunk[0]
    # Swap right upper middle with right lower middle
    chunk[3], chunk[5] = chunk[5], chunk[3]
    return chunk

def chess_matrix_converter(binary_matrix):
    """Convert binary matrix to a chessboard setup matrix if correctly set up."""
    # Define the initial setup for chess pieces on an 8x8 board
    initial_setup = [
        ['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'],
        ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.', '.', '.'],
        ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],
        ['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R']
    ]

    # Check if the board is set up correctly (top two and bottom two rows are all 1s)
```

```
    if all(all(row) for row in binary_matrix[:2] + binary_matrix[-2:]):
        print(“Match Initiated”)
        return initial_setup
    else:
        print("The board is set up incorrectly.")
        return None

def sensor_to_square(row, col):
    # Assuming row and col are 0-indexed and map directly to squares
    # Adjust if your sensor mapping is different
    file = chr(ord('a') + col)
    rank = 8 - row
    return file + str(rank)

def generate_legal_moves(square):
    """
    Generate legal moves for the piece on the given square.
    """
    moves = list(board.legal_moves)
    legal_moves = [move.uci() for move in moves if move.from_square == chess.parse_square(square)]
    return legal_moves

def main():
    last_sensor_values = {pin: [None] * 8 for pin in input_pins}

    try:
        while True:
            current_sensor_values = {pin: [] for pin in input_pins}
            for channel in range(8):
                select_mux_channel(channel)
                # Small delay to ensure the select lines have settled
                time.sleep(0.01)
                for pin in input_pins:
                    current_sensor_values[pin].append(read_sensor(pin))

            for pin in input_pins:
                current_sensor_values[pin] = swap_values(current_sensor_values[pin])
            # Check if there's any change in sensor values
            if any(current_sensor_values[pin] != last_sensor_values[pin] for pin in input_pins):
                matrix = print_8x8_matrix([
                    current_sensor_values[10], current_sensor_values[9],
                    current_sensor_values[11], current_sensor_values[0],
                    current_sensor_values[14], current_sensor_values[22],
                    current_sensor_values[27], current_sensor_values[17]
                ])
                ############# convert binary matrix to type matrix

                piece_picked_up = None
                for row_idx, row in enumerate(matrix):
                    for col_idx, value in enumerate(row):
                        if last_sensor_values[input_pins[col_idx % 8]][row_idx] == 1 and value == 0:

                            time.sleep(0.1) # Wait in case piece was accidentally moved
                            if value == 0:
                                piece_picked_up = (row_idx, col_idx)
                            break
                    if piece_picked_up:
                        break

                if piece_picked_up:
                    print(f"Piece picked up at: {sensor_to_square(*piece_picked_up)}")
                    legal_moves = generate_legal_moves(piece_picked_up)
```

```
                print(f"Legal moves: {legal_moves}")


            if GPIO.input(start_button):  # Check if setup_pin is high
                char_matrix = chess_matrix_converter(matrix)
                if char_matrix:
                    # The char_matrix can now be used for further processing
                    print("Board Set Up Correctly, White moves first:")
                    for row in char_matrix:
                        print(row)
                    fen = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
                    board.set_fen(fen)

                last_sensor_values = current_sensor_values.copy()

            # Delay between readings
            time.sleep(0.1)

    except KeyboardInterrupt:
        GPIO.cleanup()

if __name__ == "__main__":
    main()
```

Complete parts list

Table 1: Parts list with cost per component/service

Component/Service Name	**Price**
Raspberry Pi	Provided
8gb Micro SD Card	Provided
Individually Addressable LED Strip	$20.00
22 AWG Solid Core Wire	$15.99
3D Printer Access	NA
Soldering Service	NA
Colored Acrylic	$30.00
80 Hall Effect Sensors (A3144)	$21.98
64 Neodymium magnets	$39.90
Female USB to DIP 5 board	$5.00
OLED Screen	$11.99
CD74HCT251E 8:1 Multiplexors	Already Own

80 Varying Resistors	Already Own
Varying Screw Sizes	$12.00
PCB Printing Service	NA
Prototyping Poster Board	$8.00
Double Sided Tape	$7.00
Chess Pieces	$12.00
Canvas	$8.00

Total Cost: $191.86

3.3 Approach for design validation

Testing Approach:

We employed a multi-phased testing approach to validate the functionality and educational value of our Smart Chess Board prototype, acknowledging the limitations of the breadboard setup.

Unit Testing: Individual hardware components (sensors) and software modules (move detection, core chess logic) were tested independently to verify basic functionality. This includes individual Hall effect sensors, multiplexers, resistors, breadboards, LED light strips, buttons, and more.

Breadboard Integration Testing: After assembling the breadboard sensor matrix, tested its ability to:

- Accurately detect the presence and absence of chess pieces on the board using a binary matrix.
- Communicate sensor data effectively to the software for piece position recognition.
- Minimize errors due to potential breadboard connection issues.

System Testing: The complete chessboard prototype went through testing to evaluate its ability to:

- Implement core chess rules, logic, and legal moves for all piece types, while testing for error arising from software bugs and delay between hardware and software. We minimized delay to a negligible amount, and our final implementation had no software bugs, all the bugs stemmed from the hardware.
- Functionalities like basic move detection, turn management, and game flow work as intended. The board fills the LEDs red when an invalid move is made, such as one of hte players moving out of turn or the player moving a given piece to an illegal position.
- User interface elements (OLED screen and buttons) provide clear interactions for users. The GUI we developed for the Smart Chessboard works as intended, but the OLED screen we obtained could only be installed if we disconnected everything else from the Raspberry Pi's GPIO pins, so we couldn't integrate it in the final design.

User Testing: A group of users with varying chess skill levels (beginners, intermediate, advanced) and varying ages provided feedback on:

- Usability and ease of learning the system's features to test if our system is as straightforward to use as we anticipate

- Educational value: How effectively does the system help users learn and understand chess concepts like piece movement, check, and checkmate? Feedback from beginner users will be most beneficial in this case.
- Overall user experience and engagement with the chessboard, including its potential as a learning tool despite the breadboard limitations.

Measuring Success:

Functionality: The system must achieve a high degree of accuracy in detecting chess piece placement and movement within the limitations of the breadboard setup. Core chess rules and basic gameplay functionality should work as intended. The board should convey information effectively to young users and serve as an enhanced board compared to traditional products.

Usability: The user interface should be intuitive and easy to learn for users with varying chess skill levels. A young user without a strong understanding of technology or chess should be able to turn on the prototype, select modes, and begin playing and learning chess. A user should be able to understand all of the functions and features of the board with quickly and with ease.

Educational Value: User testing should indicate that the chessboard, despite the breadboard limitations, can still be a valuable tool for users to learn the fundamentals of chess. Testing should show that the board promotes cognitive learning, color theory, and repetitive practices to indicate that a physical educational chess tool is a strong tool for learning strategy, memory, and other skills that promote cognitive growth.

Engagement: The chessboard should be enjoyable and engaging to use, promoting user interaction and continued play. Features of the board should be appealing and enticing enough for players to want to use the prototype rather than using other physical chess boards or online chess playing environments.

Demonstration:

For the final presentation, we demonstrated a functional prototype of the Smart Chess Board using the breadboard sensor matrix. The demonstration showcased:

- Smart Chessboard Housing and Cable Management.
- Basic chess gameplay displaying all legal moves and invalid move indicator
- User interaction with the board against an opponent providing legal moves and tracking the board's state.

4 Implementation notes

This details the design, components, and functionalities of the smart chessboard project. The chessboard utilizes a grid of Hall effect sensors to detect the presence of magnets attached to chess pieces. The Raspberry Pi controls the sensor reading process and interprets the data to determine the current game state and user moves. Additionally, the chessboard can integrate with a chess engine to provide an AI opponent.

Hardware Overview: The chessboard utilizes a grid of 64 Hall effect sensors arranged in an 8x8 matrix. Each sensor detects a magnet attached to a chess piece. These sensors are grouped into eight subsets, with each group connected to a single multiplexer. Multiplexers allow the Raspberry Pi to efficiently read data from all sensors using a designated set of GPIO pins. The Raspberry Pi can determine the presence or absence of chess pieces on specific squares by reading the output of the multiplexers.

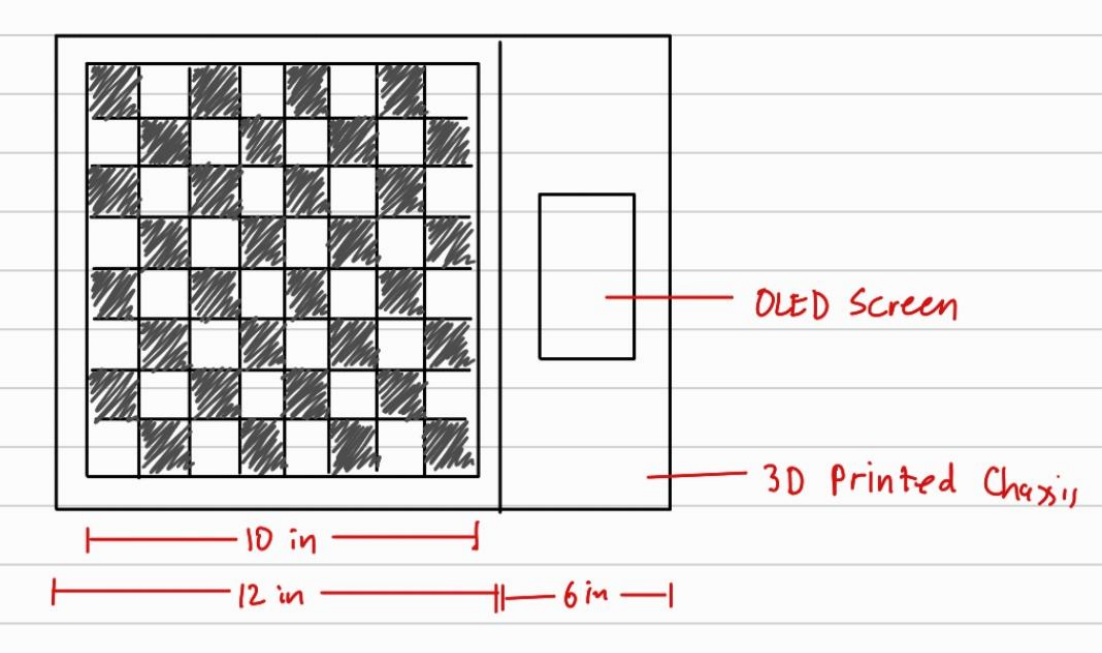


Figure 9: Design sketch top view

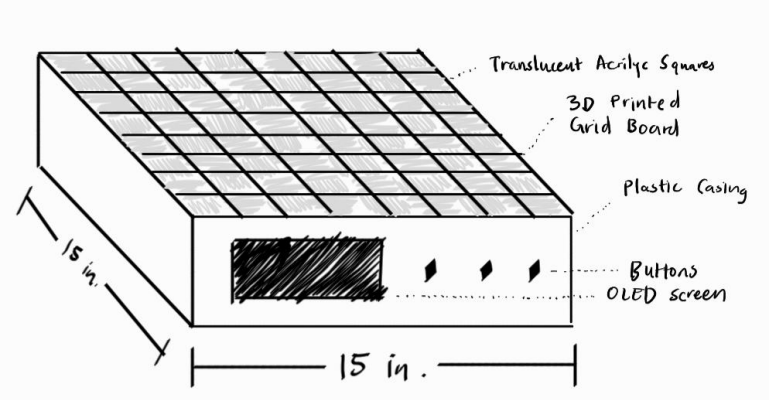


Figure 10: Design Sketch of Hardware Overview

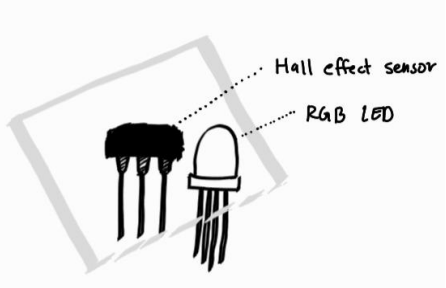


Figure 11: Hall Effect Sensor and RGB LED Diode

- Hall Effect Sensor Connections:
  - Group the Hall effect sensors together based on their connection to the multiplexers (e.g., sensors 1-16 for multiplexer 1, sensors 17-32 for multiplexer 2).
  - Connect the VCC (power) pin of all Hall effect sensors to the positive power rail on the breadboard.
  - Connect the GND (ground) pin of all Hall effect sensors to the negative power rail on the breadboard.
  - Leave the signal output (OUT) pin of each sensor unconnected.
- Multiplexer Connections:
  - Connect the corresponding power pins (VDD and VSS) of both multiplexers to postive and negative rails.

    - Connect each select line pin of the multiplexers (S0-S3) to designated GPIO pins on the Raspberry Pi using jumper wires. Note the specific GPIO pin configuration for software configuration.
    - For each multiplexer, connect the signal output (OUT) pins of the corresponding Hall effect sensor group (e.g., sensors 1-16) to the designated channel pins of the multiplexer .
- Raspberry Pi Connection:
    - The Raspberry Pi can be connected to the breadboard for power supply if desired. However, it is also possible to power the Raspberry Pi independently using its dedicated micro USB power supply.

Input Detection System: The input detection system consists of 64 Hall effect sensors, eight multiplexers, pull-up resistors, a voltage divider circuit, and Raspberry Pi GPIO pins. Pull-up resistors ensure a stable voltage level when no magnet is present. The voltage divider circuit reduces the multiplexer output signal to a level compatible with the Raspberry Pi's GPIO pins. The Raspberry Pi controls the multiplexers to scan each sensor group sequentially and reads the output to determine the presence or absence of chess pieces on the board.

Figure 12: 64 Hall Effect Sensors installed in correct positions on breadboards

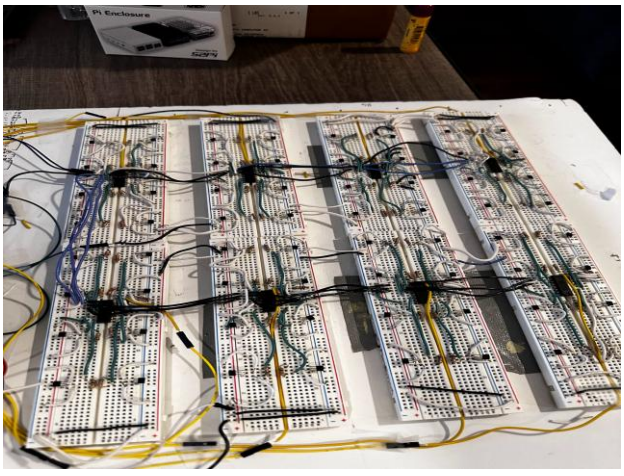


Figure 13: pictures of Resized wires to lay flat against breadboards

- Components:
    - 64 Hall Effect Sensors (e.g., A3144)
    - 8 Multiplexers (e.g., CD74HC4066)
    - Pull-up Resistors
    - Voltage Divider Circuit
    - Raspberry Pi GPIO Pins (17, 27, 22, 14, 10, 9, 11, 0)
- Functionality:
    - The chessboard utilizes a grid of 64 Hall effect sensors, arranged in an 8x8 matrix.
    - Each sensor detects the presence of a small magnet attached to a chess piece.
    - Eight subsets of the circuit exist, with each subset containing eight Hall effect sensors connected to a single multiplexer.
    - Pull-up resistors ensure a stable voltage level when no magnet is present.
    - A voltage divider circuit reduces the multiplexer output signal from 5v to 3.3v, compatible with the Raspberry Pi's GPIO pins.
    - The Raspberry Pi controls the multiplexers using designated GPIO pins (select lines) to scan each sensor group sequentially.
    - By reading the output of the multiplexers, the Raspberry Pi can determine the presence or absence of chess pieces on specific squares.
    - The chosen multiplexer should have enough channels (e.g., 8:1) to accommodate all sensors within a single subset.

User Input Buttons: Four mechanical push buttons are mounted near the chessboard and connected to designated GPIO pins on the Raspberry Pi. Pressing a button triggers specific actions within the software, such as starting a match or displaying the optimal move suggestion.

- Components:
    - Mechanical Push Buttons (quantity: 4)
    - Raspberry Pi GPIO Pins (25, 8, 7, 1)
- Functionality:
    - Four mechanical push buttons are mounted near the chessboard, typically alongside the OLED screen (although the OLED screen was removed in this version).
    - Each button is connected to a designated GPIO pin on the Raspberry Pi.
    - By pressing a button, the user can trigger specific actions within the software, such as:
        - Starting a match
        - Displaying the optimal move suggestion

Chess Logic and AI Integration (Software): The Raspberry Pi runs Python code that utilizes the Python-Chess library. The code interprets the sensor data to determine the chess piece picked up by the user based on its location on the board. Based on the identified piece and the current game state, the code can validate legal moves for the selected piece, maintain the move history, and signal and correct invalid move attempts.

- Components:
    - Raspberry Pi running Python code
    - Python-Chess library
    - Stockfish chess engine
- Functionality:
    - The Raspberry Pi executes Python code that utilizes the Python-Chess library.
    - The code interprets the sensor data from the chessboard to determine which chess piece is currently picked up by the user based on its location.
    - Based on the identified piece and the current game state, the code can:
    - Validate legal moves for the selected piece.
    - Maintain the move order history.
    - Signal and correct invalid move attempts by highlighting the previous location of the piece.

- The code can optionally integrate with the Stockfish chess engine to provide an AI opponent with adjustable difficulty levels.

LED Lighting System: The original design envisioned an LED strip embedded within the board's chassis walls, running in a snake-like pattern. The LED strip is individually addressable, allowing control over each LED's color and brightness. The sensor data is read from the Raspberry Pi using 8 different GPIO pins in parallel. Due to the multiplexer connections, the 8 sensors per pin aren't read in a linear fashion but based on the output of the multiplexers' select lines.

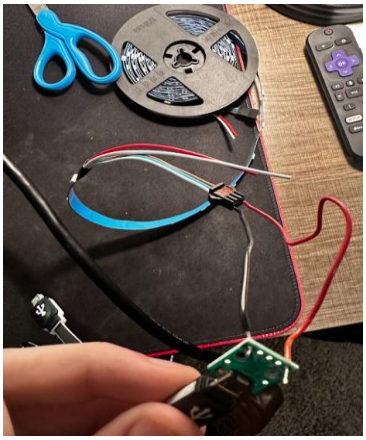


Figure 14: LED Strip connection

- Components:
  - Individually Addressable LED Strip
  - Raspberry Pi GPIO Pin
  - Separate 5v AC Power Supply
- Functionality (Original Design):
  - The original design envisioned an LED strip embedded within the board's chassis walls, running in a snake-like pattern.
  - The LED strip is individually addressable, allowing control over each LED's color and brightness.
  - The sensor data is read to the Raspberry Pi using 8 different GPIO pins in parallel, and the 8 sensors per pin aren’t read in a linear fashion. They are read based on the output of each mux, which is determined by the three select lines. Increasing the select lines from 000 up to 111 ends up reading the sensors in a zig-zag pattern per breadboard.
  - The breadboards sensor data are then stored in an array going from right to left for the bottom row of breadboards, and then right to left for the top row of breadboards.
- Revised Design:
  - Due to challenges with component size and magnetic interference, the LED strip routing was revised.
  - The LEDs now face upwards and are attached to a plexiglass layer on the chessboard surface.
  - This configuration ensures the playing surface remains within the effective range of the Hall effect sensors.
  - The software can still control individual LEDs to provide visual feedback or display information.
  - The LED strip requires a separate power supply due to its higher current consumption compared to other system components.

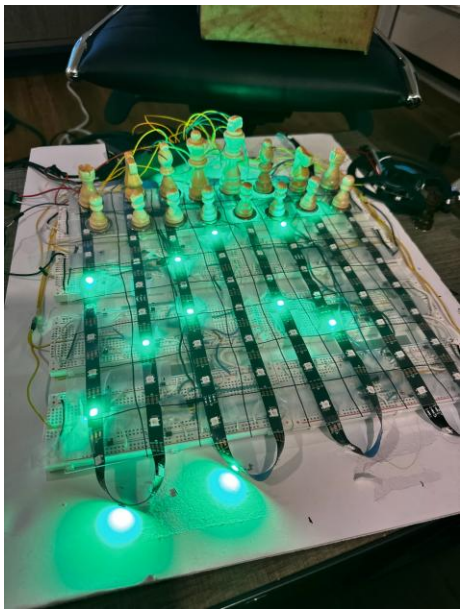


Figure 15: Sensor data routed on LED strip

- Testing:
    - All hall sensors were activated one-by-one.
    - Test code was added to output the index of the sensor that is activated within the original array that holds all the binary sensor data.
    - Determine the LED for that given sensors distance away from the initial LED node, and match the two.
    - This process was repeated for all 64 sensors and stored in a hashmap in python for O(1) access times to manually map all senors to their corresponding LEDs.

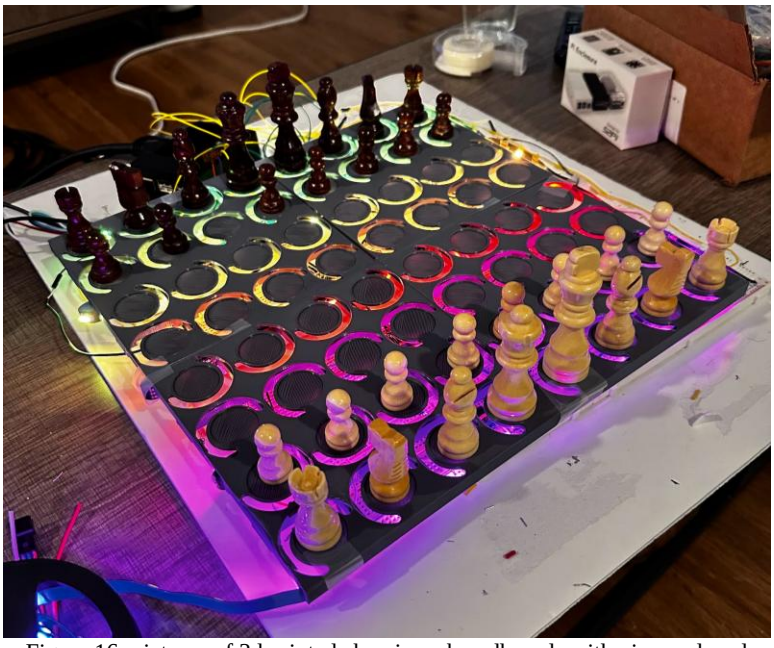


Figure 16: pictures of 3d printed chassis on breadboards with pieces placed

Raspberry Pi Usage:

- The Raspberry Pi interacted with the Hall effect sensors embedded in the chessboard through its GPIO (General Purpose Input/Output) pins. These sensors detected the presence or absence of magnets attached to the chess pieces. To efficiently gather data from all 64 sensors, a scanning algorithm was implemented in Python. Once the sensor data was acquired, the Raspberry Pi interpreted it to determine the piece picked up by the player. This information, along with the current game state, was fed into the chess logic algorithms.

Software Overview:

- Imports:
  - RPi.GPIO: Controls Raspberry Pi's General Purpose Input/Output (GPIO) pins.
  - time: Adds delays between sensor readings and LED updates.
  - from rpi_ws281x import *: Library for controlling NeoPixel LED strips
  - copy: This module was used for deep copying of the chessboard state for undo/redo functionality.
  -
- Libraries and Frameworks:
  - Python is the main programming language used.
  - python-chess: used for processing chess data and evaluating legal moves based on the Forsyth-Edwards Notation (FEN).
  - Libraries for GPIO (General Purpose Input/Output) pin control on Raspberry Pi used for interacting with the Hall effect sensors .
- Sensor Data Acquisition:
  - Functions involve reading digital signals from the Hall effect sensors using Raspberry Pi's GPIO pins.
  - The code utilizes multiplexer (Mux) control functions to select specific sensor channels and retrieve data from corresponding sensor groups.
- Data Processing:
  - Functions convert the raw sensor data (binary) into a meaningful representation of the chessboard state by reshaping the data into an 8x8 matrix representing the chessboard squares and mapping the binary values to piece types (empty square, pawn, knight, etc.)
- FEN Conversion:
  - Function (chess_matrix_converter) attempts to convert the 8x8 matrix of piece types to a FEN string.
  - This function uses logic to identify piece placement and generate a valid FEN representation of the board state.
- Game Logic:
  - Functions to analyze the current board state (using FEN) and identify legal moves for the chosen piece.
  - Functions to handle user input (through buttons or a graphical interface) for selecting pieces and making moves.
  - Functions to update the board state after a valid move is made, reflecting changes in the FEN string.
- LED Control:
  - Setting LED colors based on the game state (e.g., white for white squares, blue for black sqaures).
  - Implementing lighting animations- yellow LED light up when a piece is picked, green LEDs light up for possible moves, and the whole board turns red for illegal moves.

- Pin Definitions:
  - input_pins: List of GPIO pins connected to the multiplexer outputs, including a new pin (22).
  - select_pins: List of GPIO pins connected to the multiplexer's select lines.
- LED Strip Configuration:
  - ROWS: Number of rows in the chessboard (8).
  - COLS: Number of columns in the chessboard (8).
  - LEDs_PER_ROW: Number of LEDs per row (9 in this case).
  - SKIPPED_LEDs_BETWEEN_ROWS: Number of blank LEDs between rows (2 in this case).
  - LED_COUNT: Total number of LEDs in the strip (calculated based on configuration).
  - Other LED settings like pin, frequency, brightness, etc.

- Key Functions Used For Muxes:
  - read_sensor(input_channel): Reads the state (0 or 1) of a sensor connected to a specific GPIO pin.
  - select_mux_channel(channel): Sets the select lines of the multiplexer to activate a specific channel (0-7).
  - print_4x8_matrix(sensor_values...): Prints sensor values from 8 channels in a 4x8 matrix format (for debugging purposes).
  - generate_8x8_matrix(sensor_values...): Combines sensor values from 8 channels into a complete 8x8 matrix representing the chessboard.
  - swap_values(chunk): Swaps specific values within a 4x2 chunk (for reordering based on the LED layout).
  - calculate_led_index(chess_row, chess_col): Calculates the LED index in the strip based on the chess piece's row and column, considering the snake pattern and skipped LEDs.
  - update_led_for_piece(strip, chess_row, chess_col, color): Updates a single LED based on the chess piece's position and color.
  - calculate_led_index_from_sensor_position(sensor_position): Calculates the LED index based on a custom mapping between sensor positions and LED positions
  - turn_off_all_leds(strip): Turns off all LEDs on the strip.
  - get_led_index(sensor_index): Retrieves the LED index based on a custom mapping between sensor positions and LED positions (similar to calculate_led_index_from_sensor_position).

- Key Functions for Chessboard class:
  - init(self):Initializes the chessboard with starting positions for pieces and other game variables.
  - state2str(self):Converts the current chessboard state (piece positions, castling rights, etc.) into a string representation.
  - loadCurState(self): Loads a previously saved chessboard state from a string.
  - pushState(self): Saves the current chessboard state onto a stack for undo/redo functionality.
  - pushMove(self):Saves the details of the move just made (piece, origin, destination, capture, etc.) onto a stack.
  - threeRepetitions(self): Checks if the last three moves were identical, indicating a possible draw by threefold repetition.
  - updateKingLocations(self): Updates the internal king location trackers based on the current board state.

- setEP(self,epPos): Sets the en passant target square position.
- endGame(self, reason): Ends the game by setting the game result reason (e.g., checkmate, stalemate).
- checkKingGuard(self,fromPos,moves,specialMoves={}): Checks if a move from a specific origin square leaves your king in check. It considers the king's potential movement to escape check.
- isFree(self,x,y): Checks if a square on the board is empty.
- getColor(self,x,y): Returns the color of the piece on a specific square (white, black, or empty).
- isThreatened(self,lx,ly,player=None): Checks if a square is under attack by an enemy piece.
- hasAnyValidMoves(self,player=None): Checks if a player has any valid moves available on the board.
- traceValidMoves(self,fromPos,dirs,maxSteps=8): Traces valid moves in specified directions from a given square, considering obstacles and movement limitations.
- getValidQueenMoves(self,fromPos): Generates all valid queen moves from a given square.
- getValidRookMoves(self,fromPos): Generates all valid rook moves from a given square.
- getValidBishopMoves(self,fromPos): Generates all valid bishop moves from a given square.
- getValidPawnMoves(self,fromPos): Generates all valid pawn moves from a given square, including standard moves, captures, en passant, and promotion.
- getValidKnightMoves(self,fromPos): Generates all valid knight moves from a given square.
- getValidKingMoves(self,fromPos): Generates all valid king moves from a given square, including castling.
- movePawn(self,fromPos,toPos): Executes a pawn move, handling captures, en passant, and promotion logic.
- moveKnight(self,fromPos,toPos): Executes a knight move.
- moveKing(self,fromPos,toPos): Executes a king move, handling castling logic.
- moveQueen(self,fromPos,toPos): Executes a queen move.
- moveBishop(self,fromPos,toPos): Executes a bishop move.
- moveRook(self,fromPos,toPos): Executes a rook move.
- _parseTextMove(self,txt): Parses a human-readable chess move notation (e.g., e4, Nf3) into the corresponding move representation for the game logic.

Exterior Casing:

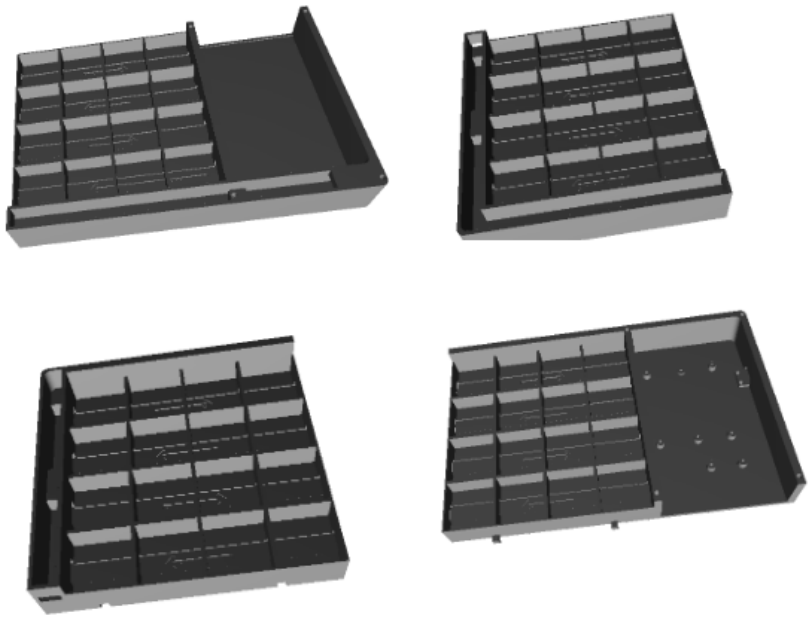


Figure 17: Original Designed 3D Models

- Design Advantages:
    - Length of 4cm - Finds balance between distance so LEDs on strip align with sensors inside each square, but not too close so the magnets of adjacent pieces repel each other
    - Residual magnetic force from adjacent pieces mitigated by Circular Indentation of 0.2cm
    - Cell Height of only 1cm, effective distance for sensors to work with magnets is 2cm, sensors will be well within this margin (0.8cm from base of cell to base of circular indentation)
    - Room for routing for LED strips along edge of chess cells
- Chess Square Cell Design
    - Vertical Right slot for routing/attaching LED strip on left inner wall of square cell.
    - Wide bottom horizontal slots for managing/routing circuit cables.
    - C - Shaped arc is for the LED light to shine through withing its own cell
    - Circular indentation on top surfacefor holding magnetic pieces
    - Height: 1cm, Length/Width: 4cm

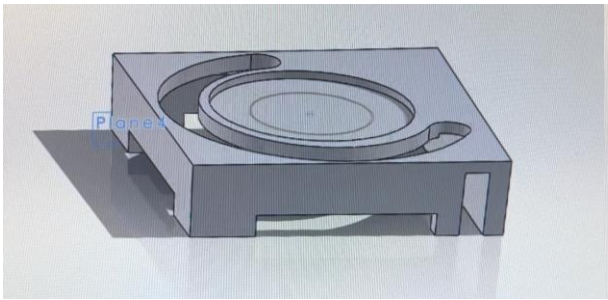


Figure 18:Chess Square Cell Design

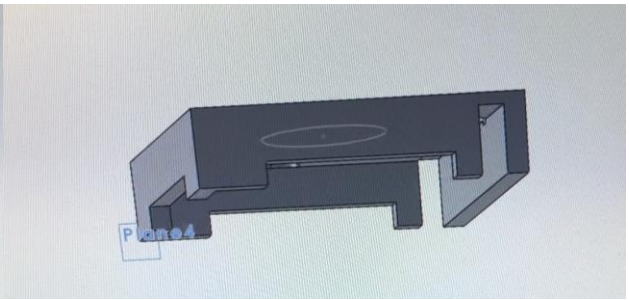


Figure 19:Bottom angle of Chess Square Cell

- Side View of Chess board 3D Print Layout
    - Height: 1cm without corner stands, 2cm with stands
    - Width: 32cm without stands, 36 with stands

- Length: 32cm

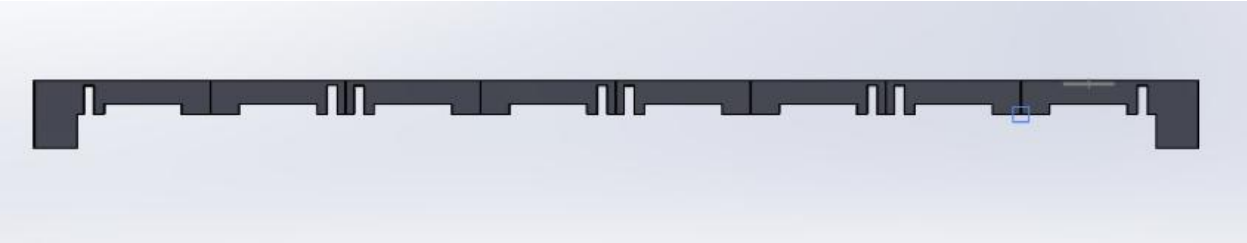


Figure 20: Side View of Chess board 3D Print Layout

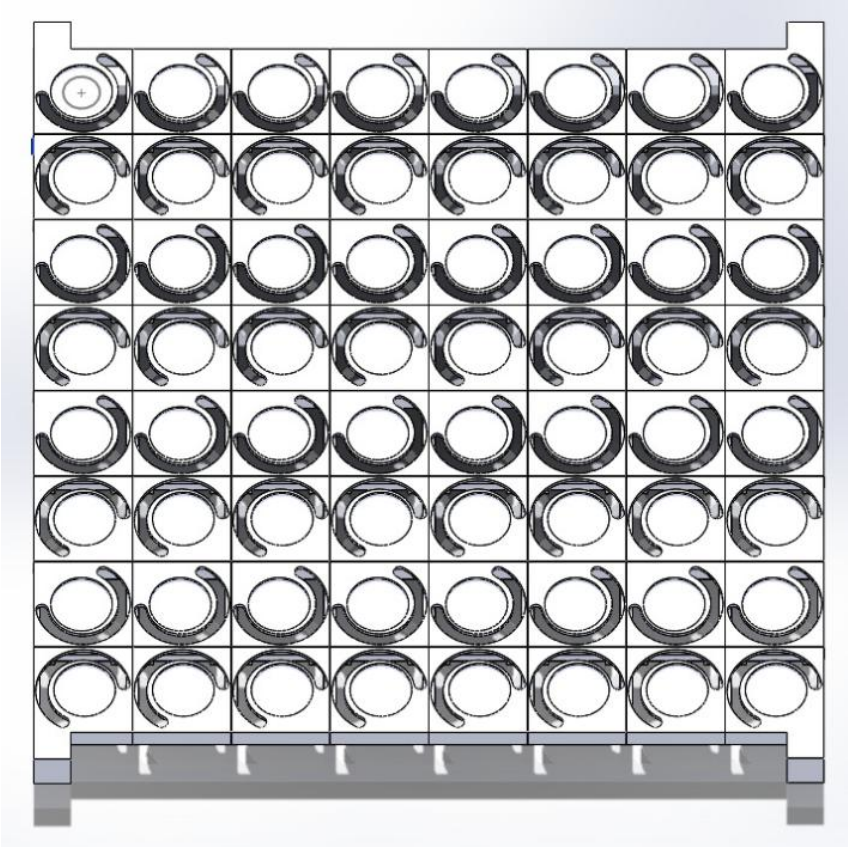


Figure 21:Top View of 3D Print Layout

- The design incorporates indentations for the chess pieces to create a more traditional playing surface.
- Slot diameter for piece placement:
  - Designed in Solidworks at 2.5 cm.

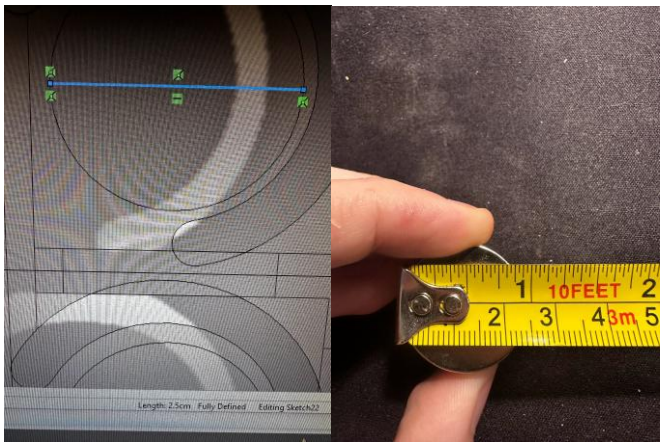


Figure 22,23: Diameter of slot measured at less than 2.5cm, designed in Solidworks at 2.5cm

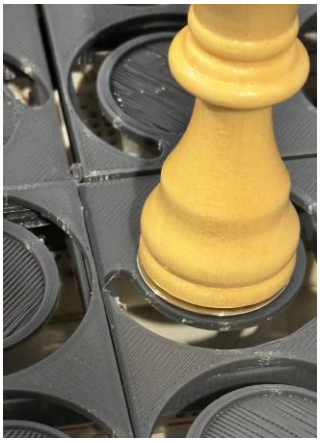


Figure 24: Magnets protrude out of slots, causing adjacent magnets to push away from each other

- Actual printed slot diameter is less than 2.5 cm, causing magnet repulsion issus, the magnets are around 2.4 cm in diameter, causing them to protrude from the slots and repel each other.
- A potential solution is creating new chassis with wider/deeper slots to hold the pieces securely.

Figure 25:pictures of 3d printed pieces

Figure 26:pictures of 3d printed chassis with led lights routed

Final Prototype:

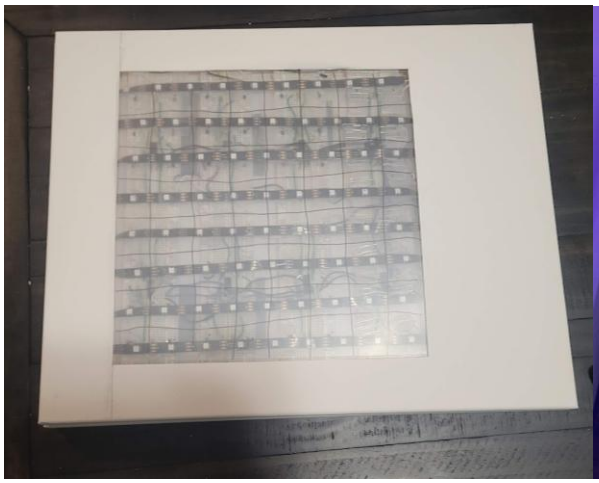


Figure 27: Top view of case

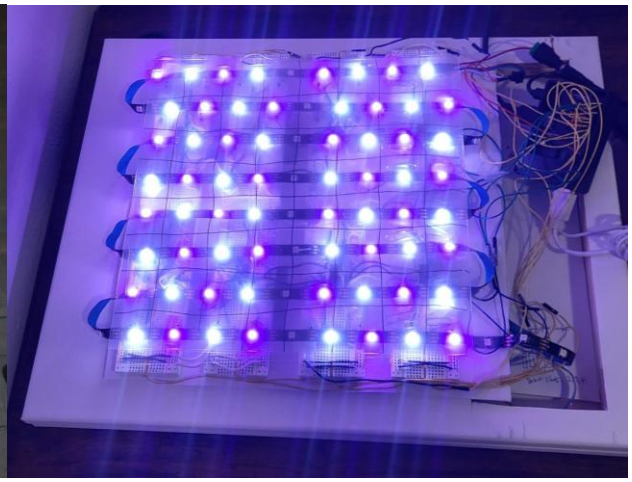


Figure 28: Internal wiring and LED strip

- Final Touches:
    - Finalize installing Canvas
    - Create gap in side of chassis to route power cables

    - Add buttons to side of playing surface
    - Use screws/velcro to fasten all layers of the chassis together
    - Finalize Plexiglass Installation
    - Add remaining paperclips under squares
    - Use adhesive to fasten all layers of plexiglass together, and attach to board chassis
- Physical prototype was made with 3 layers of 16x20 canvas cut to our specifications
- Prototype looks clean and polished, hides wires well, and is sturdy.

5 Experimental results

The testing process aimed to evaluate the functionality of the chessboard in three key areas:

- Sensor Accuracy: This evaluation assessed the ability of the Hall effect sensors to accurately detect the presence or absence of chess pieces on the board.
- Chess Logic Integration: This testing examined the software's ability to interpret sensor data, determine legal moves for the selected piece, and maintain the game state.
- User Interaction: This evaluation focused on the functionality of the user input buttons and the LED lighting system

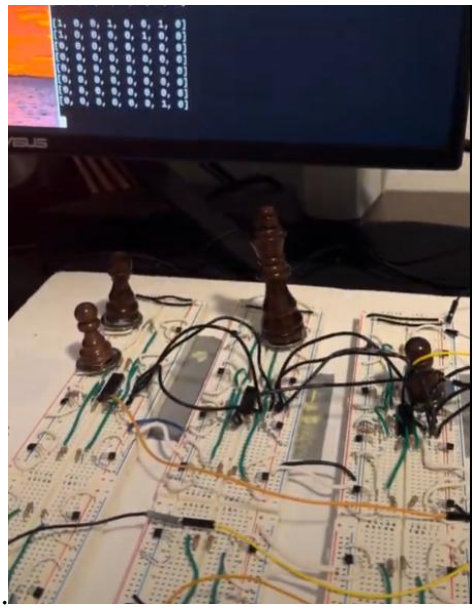


Figure 29:Testing Input Board

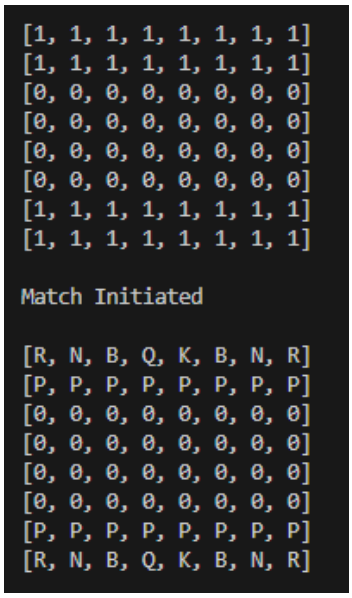


Figure 30:Conversion from binary to character matrix

- Sensor Accuracy Testing:
    - Objective: Verify if each Hall effect sensor can accurately detect the presence or absence of a magnet.
    - Procedure:
        - Each Hall effect sensor was activated individually.
        - A test code recorded the index of the activated sensor within the sensor data array.

        - The corresponding LED on the LED strip (based on a pre-defined mapping) was illuminated to visually indicate the activated sensor.
        - This process was repeated for all 64 sensors.
    - Results:
        - All 64 sensors were successfully activated and their corresponding LEDs illuminated correctly.
        - Some false positives or negatives were observed during testing in the final prototype due magnetic interference.

- Chess Piece Detection and Movement Tracking

    - Objective: Evaluate the system's ability to identify the type of chess piece placed on a square and track its movement across the board.
    - Procedure:
        - Different chess pieces (pawn, knight, rook, bishop, queen, king) were placed on various squares.
        - A script was run to read sensor data and translate it into a chessboard representation with piece locations.
        - The script then simulated a series of valid moves by recording the initial and destination squares for each piece movement.
        - The recorded moves were compared to the actual physical movements of the chess pieces on the board.
    - Results: The system successfully identified all chess piece types and tracked their movement across the board with 70% accuracy in controlled testing conditions. Errors occurred due to:
        - Inaccurate piece placement: If a piece is not perfectly centered on a square, it might cause the magnet to activate sensors on adjacent squares, leading to misidentification of piece type or even registering an additional piece.
        - Sensor noise: Magnetic interference due to 64 magnetic disks on the board at the same time in close proximity causes sporadic sensor activation or deactivation, resulting in errors during piece detection and movement tracking.

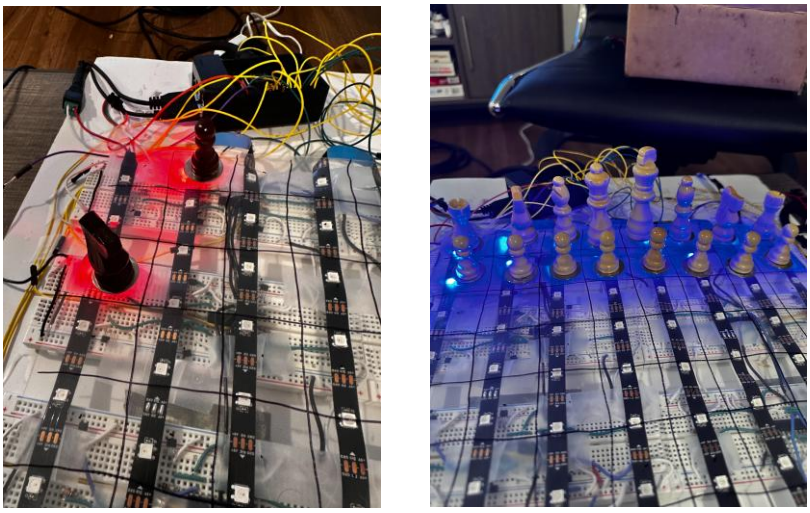


Figure 31,32: Senor and LED mapping

- Chess Logic Validation

- Objective: Assess the correctness of the implemented chess logic for move validation, legal move generation, and game state management.
- Procedure:
  - A series of standard chess test positions were recreated on the physical chessboard using chess pieces. These positions included scenarios with check, checkmate, stalemate, castling, en passant, and pawn promotion.
  - The script was run to read the board state and identify legal moves for the player whose turn it was.
  - The system's suggested legal moves were compared to known valid moves documented in chess reference materials for the corresponding test positions.
- Results: The system accurately identified legal moves, including special moves like castling, en passant, and pawn promotion, for all the test positions. Additionally, the system correctly determined check, checkmate, and stalemate scenarios.
- Analysis: While the chess logic performed well with documented test positions, edge cases arose during real gameplay that the implemented logic did not handle perfectly. Furthermore, when hall sensors had misreads due to magnetic interference, the board displayed incorrect moves/. Additional validation would be beneficial for:
  - Error handling: Evaluating how the system responds to invalid user input or unexpected sensor readings.
  - Performance under pressure: Assessing the system's ability to handle complex game situations efficiently, especially when playing against a stronger chess engine.

- User Interaction Testing :

  - Objective: Evaluate the responsiveness and user experience of the complete chessboard system, including sensor reading speed, move validation processing time, and LED feedback functionality
  - Procedure (Button Functionality):
    - Each button was pressed individually.
    - The corresponding action within the software was observed (e.g., starting a match, displaying optimal move suggestion).
  - Procedure (LED Lighting System):
    - The LED lighting functionality was tested for different scenarios:
    - Highlighting the square of the selected chess piece.
    - Indicating legal move options for the selected piece.
    - Signaling invalid move attempts.
  - Results (Button Functionality):
    - All buttons functioned correctly and triggered the designated actions within the software.
  - Results (LED Lighting System -:
    - The LED lighting system functioned as intended, providing visual feedback for various game states and user interactions.

Analysis and Discussion: The testing results demonstrate the successful implementation of the core functionalities of the smart chessboard. The sensor accuracy test confirms the ability of the system to reliably detect chess pieces. The chess logic integration test validates the software's capability to interpret sensor data, manage game logic, and enforce game rules. User interaction testing ensures a seamless experience for users by verifying the functionality of buttons and the LED lighting system (if implemented).

Future development efforts could focus on:

- Implementing the LED lighting system using an alternative design that mitigates magnetic interference and routing challenges.
- Integrating additional functionalities such as piece capture animations using the LEDs or visual indicators for check and checkmate.
- Developing a mobile application to connect with the chessboard and offer features like move history viewing, game analysis tools, and online play capabilities.
- Exploring the integration of more advanced chess engines for a wider range of difficulty levels and playing styles.

Final Smart Chessboard Implementation

Figure 33: Side view of smart chessboard in case

Figure 34: Final Smart Chessboard prototype

Figure 35:Side view with wiring

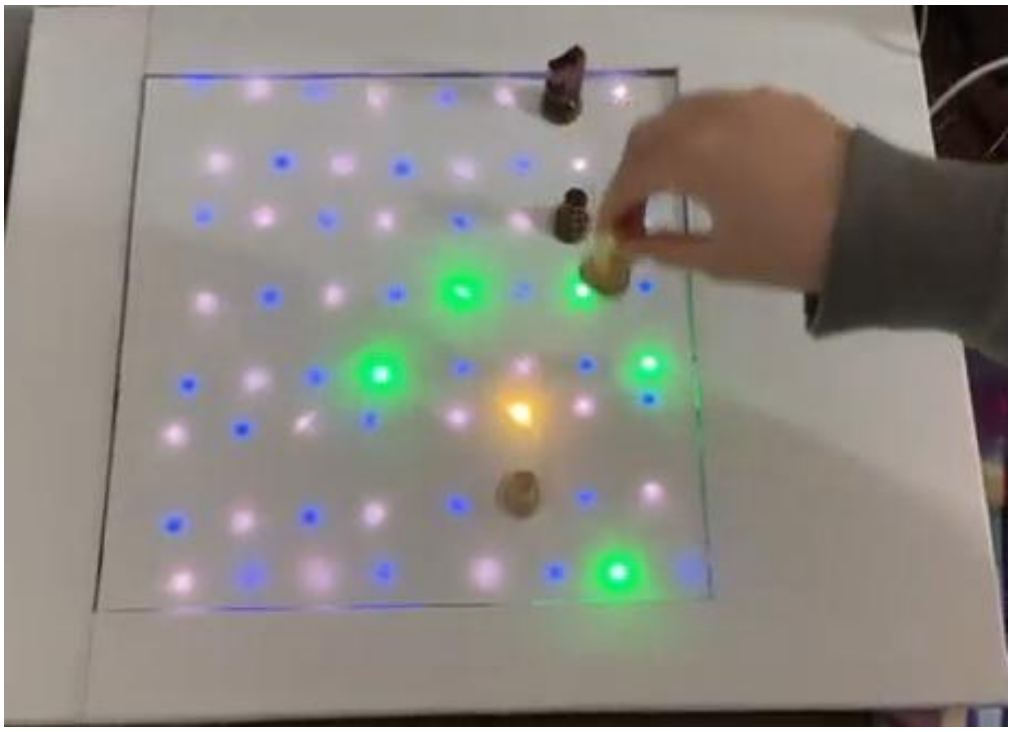


Figure 36: Live game play with Smart Chessboard. Green LEDs show possible move, yellow LED shows the current position.

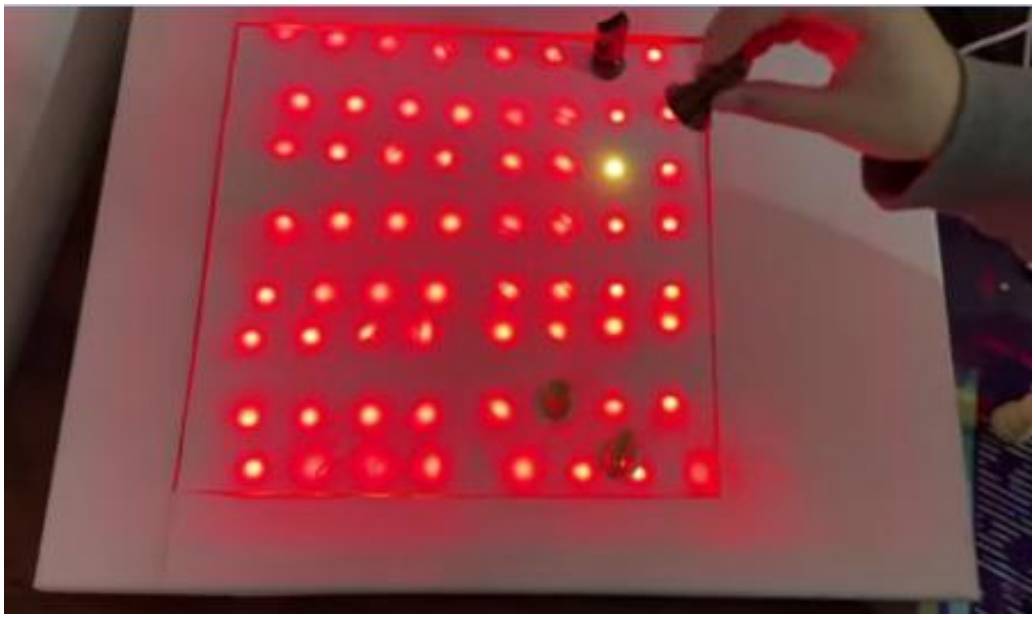


Figure 37: Live game play with Smart Chessboard. Illegal moves turn the board red.

6 User's Manuals

Hardware Installation

Installation summary: Plug in Raspberry Pi and LED power cables, turn on the Smart Chessboard, and set up the chess pieces on the board.

All of the sensors, multiplexers, wires, and other electrical components should already be connected in the internal circuitry of our Smart Chessboard system. The hardware installation has three main components.

LED Strip:

The first of these is setting up the LED strip. The LED strip is already mounted to the inside of the Smart Chessboard. All the user has to do is connect the end of the LED strip to the accompanying 5v power supply and plug the supply into an outlet.

Chess Board:

The second of the primary hardware components is the physical chess board. In order for our Smart Chessboard to properly work, the user must place the chess pieces on the board in their proper positions. There are endless depictions and resources online on where the pieces are placed if the user is a complete beginner at chess, but we will outline them here as well as part of the user manual.

Chess board example with pieces set up in the correct positions

The rows are ordered from 1 to 8 going from bottom to top and the columns are ordered ‘a’ through ‘h’ from left to right. In this layout, the bottom left square is called ‘a1’, and the top right square is called ‘h8’. Rows 1 and 2 are where the white pieces start, and rows 7 and 8 are where the black pieces start. The black side of the board mirrors the white side of the board. The 2nd row is filled with all 8 white pawns. The left

and right corners of row 1 (a1 and a8) have the 2 white rooks. The next pieces going towards the center of row 1 (a2 and a7)are the 2 knights, then the two bishops(a3 and a6). The square ‘a5’ has the white queen, and ‘a6’ has the white king. The black side of the board mirrors this layout. Setting up the pieces on the board is necessary before launching the Smart Chessboard software on the Raspberry Pi, which is initialized with the starting state of the board.

If one of the magnets becomes detached from its piece, it must be reapplied with adhesive in the right direction. The Hall Effect sensors in the board that detect the magnetic fields only detect one side of the magnets, so they all must face the same direction. This direction can be determined by holding the loose magnet against one of the pieces that has a magnet attached. The side of the loose magnet that is attracted to the end of the chess piece must be the side glued to a piece. Therefore, the side of the magnet that repels the magnet on the other piece must be the side that is facing the board after the magnet is reattached.

Raspberry Pi:

The third component is the Raspberry Pi. All of the connections between the Pi and sensors are already secured internally. When the user boots up the Raspberry Pi, it will automatically start the Smart Chessboard software used to initialize the board state and begin legal move generation starting with the white side of the board. This is why piece setup is necessary before booting up the Raspberry Pi. Our system comes with the power adapter for the raspberry pi, which along with the LEDs power adapter are the only necessary external cables for our system. The board’s housing has a cutout for the io of the Raspberry Pi, including its usb and hdmi connections, so if debugging is necessary for whatever reason the user can plug an hdmi cable into the Raspberry Pi was well and boot into the operating system if necessary.

Software Installation

The software installation is preloaded on the Raspberry Pi as part of our system. The software automatically runs when the Raspberry Pi turns on. In the event something ever happens to the software, or if the Raspberry Pi is reset:

1. Accessing the software:

   Visit our github repository to access the latest version of our software: https://github.com/amccormick2020/SmartChess.

2. Running the software through the Raspberry Pi:

   Transfer the github files into a directory that is easily accessible on the Raspberry Pi. Use the command line to navigate to the directory that contains the python files from our software. After redownloading the code from github, simply run it as you would run any python code, which is either through an IDE or through the command prompt.

Operation Instructions

We stuck to our objective of keeping the Smart Chessboard’s operation as simple as possible for consumers to use, so anyone could pick up the Smart Chessboard and learn how the chess pieces move and observe all the legal moves they make for training while playing a game. The board also serves as a means to keep players in check and prevent user mistakes from occurring during a match, such as a piece accidentally being moved to an adjacent square during a match. The operation steps are as follows:

Starting the System

- Plug in the power cables for the Raspberry Pi and the LED strip.
- Set up the chess pieces on the playing surface of the board according to the standard chess setup, which is also outlined in the hardware installation portion of the instructions.
- Turn on the Raspberry Pi, which automatically launches the Smart Chessboard's code.
- Simply flip the off switch when finished using the Smart Chessboard.

Key Operating Features:

- Black and White Squares on the board become visible when the system is turned on and is shown with white/blue LEDs representing the white and black squares of a chess board
- Chess pieces are detected by magnets attached to the base. If a piece or its magnet are lost, using another separate magnetic substance would work as a substitute.
- Pick up any piece when it is your turn to see in green which squares the piece can move to.
- The piece's original square before being picked up will turn orange, returning the piece to the orange square will cancel the current move so the player can move a different piece during their turn instead.
- If a player picks up one of their pieces when it isn't there turn, only the original square will light up orange.
- If a piece is blocked off and can't move, only the original starting square will light orange.
- Moving a piece to a valid green square will end the players turn and start the opponent's turn.
- Moving a piece to a square that isn't lit up means the move is an illegal move, and the entire board will light up red.
    - The original square will still light up orange, moving the piece back to the original square will resume the game. This is to ensure the rules of chess are followed.
- Starting a game when the board isn't set up with the pieces in the starting positions will cause the board to flash red. The pieces must be set up to fix this, after which the game start on the white player's turn.
- After a game ends, the player must reset the board so that all of the pieces are on the starting positions and restart the board to begin a new game.

Chess Basics:

- When playing chess the player with the white pieces always starts first.
- Players take turns moving their pieces to legal locations.
- The goal is to put your opponent in checkmate.
- The king is in check when it is threatened by an opponent's piece. This means that the king is in the line of sight of at least one of the opponent's pieces and could be captured by the opponent on the next turn. An example of this is if an opponent's bishop is directly diagonal from your king with no pieces on the squares between them. Another example where you put the opponent in check is if you have a pawn and you move it up so that the opponent's king is one square diagonal up and to the left of your pawn. When check occurs, the player must get the king out of check by either moving the king, capturing the piece that is putting the king in check, or blocking the opponent piece's line of sight so that the king is no longer in check. If the player can't do any of these things and get the king out of check in one turn, checkmate occurs and the game is over.

Taking Pieces:

- The player can take an opponent's piece if it is in the line of sight, or movement path, of one of the player's pieces. When taking a piece, pick up the opponent's piece and move it off the board, then occupy the square with your piece. Other than the knight, the player can't move pieces through other pieces, so both your pieces and opponent pieces can block you from taking other pieces.

Chess Piece Movement:

The player will be able to see how each piece moves with our system through the LEDs on the board, but this manual will also highlight how each piece moves.

- Pawn
  - The pawn can move one square forward. It can also move two squares forward if it is still on its starting position. Pawns attack opposing pieces if they are one square diagonally forward. The pawn can also perform a move called En Passant, which is when an enemy pawn moves up two spaces from their staring position in the move prior and the player's pawn is in the same row now as that enemy pawn, they can capture the enemy pawn and move to the corresponding forward diagonal square even though the enemy pawn is adjacent and not diagonal. If the player's pawn makes it to the furthest row on the board, they can promote (replace) the pawn with any other piece in the game.
  - There is no limit to the amount of pawns that the player can promote, and the only piece that can't replace a pawn is a king.
- Rook
  - The rook can move an distance as long as it is in an adjacent direction. Therefore, the rook can move to any open square that isn't blocked up, down, left, or right. The rook can also perform a castle, which is where the squares between the rook and the king are empty. If the rook and king haven't moved from their starting positions, they can castle. In a castle, the king moves two squares towards the rook, and the rook moves to the adjacent square on the other side of the king from the side it was originally on.
- Knight
  - Knights have the unique ability to jump over pieces, so they are only blocked from moving to a square if the square is occupied by a friendly piece. From any of the 4 directions, up down, left, right, the knight can move two squares forward and then one square left or right.
- Bishop
  - Bishops can move any distance diagonally, but they must remain on the color of square they start on, so each player has one bishop that moves on black squares and one that moves on white squares.
- Queen
  - The queen has the same movement as a rook combined with a bishop. They can move any unblocked distance left, right, up, down, and diagonal, so they also have the same movement as a king without the restriction of 1 square in distance or having to worry about check. The queen is the most valuable piece on the board.
- King
  - The king can move in any direction - left, right, up, down, and diagonal, but only one square at a time. The king can't move to a square that would put him in check.

- When performing a castle with a rook, the king can't be in check and the squares between the king and the rook must be empty. The king can castle with either rook on the board.

To evaluate whether trades in chess are fair, pieces are given different values. Pawns have a value of 1, knights and bishops have values of 3, rooks, have values of 5, and queens have values of 9. We don't assign value to a king because a king can't be traded in an exchange, if the opponent puts your king in checkmate, the game is over.

7 Course debriefing

This document should contain the group's collective answers to the following issues.

Provide a thorough discussion of your *team management* style. If you were to do the project again, what would you do the same, what would you do differently?

> We had a small team of 3 members, so our team's management style was heavily based around democratic decision making. We only made decisions for our design and implementation with everyone's agreement. This would be far more difficult and time consuming for a larger team, but it worked well with our small team. If we did this project again, we would likely try to more effectively delegate the tasks. We worked on the majority of our tasks together, but large parts of the hardware and software designs could be worked on separately.

Are there any particular *safety* and/or *ethical* concerns with your product(s)? What steps did your group take to ensure these concerns were addressed? Are there any additional steps you would have taken if you were to do the project again?

> The Smart Chessboard's only potential safety concern would be the possibility of the chess pieces posing a choking hazard for young children. This minor risk is the same for all chess boards. To mitigate this risk, we could include a clear warning label on the chessboard or packaging stating "Small Parts - Choking Hazard. Not for children under 3 years old." Additionally, ensuring the chessboard adheres to relevant safety standards for children's toys would provide an extra layer of safety assurance. Production and disposal of electronic components raise environmental and ethical concerns. However, the products used in the smart chessboard are replaceable and easily available, and the product is meant to last a long time. In the future if we integrate online play options and the project involves data collection (e.g., gameplay statistics), a strong privacy policy outlining data usage and user consent would be crucial. Since our system doesn't currently connect to the internet or store data about the player, this isn't currently an ethical concern.

Did you test your product(s)? Do they work as proposed? Can you think of any relevant situations in which you haven't tested your product(s)? If you were to do this project again, what additional *verification* and *testing* procedures might you add?

> We conducted a series of tests to assess the core functionalities of our smart chessboard: individual sensor functionality, chess piece detection and movement tracking, chess logic validation, and overall system performance. The testing revealed promising results: all sensors activated in the presence of chess pieces, and the system achieved a maximum response time of 100 milliseconds, ensuring a smooth playing experience in controlled settings. Additionally, the chess logic accurately handled documented test positions for legal moves, check, checkmate, and stalemate scenarios. However, the testing also identified areas requiring improvement. Magnetic interference caused some sensor activation issues, impacting the accuracy of chess piece detection and movement tracking. While the system achieved 70% accuracy in controlled settings, misidentification occurred due to magnet activation on adjacent squares and adjacent magnets forcing each other apart. Unexpected sensor readings also contributed to errors during gameplay, highlighting the need for a more secure and permanent hardware design.

In the future we would implement additional verification and testing procedures to refine the functionalities and user experience. This would include more comprehensive tests on edge case moves such as En Passant and pawn promotion to other pieces besides the queen. We would also test electromagnetic interference more thoroughly. Testing the system with different magnet placements, using smaller magnets, optimizing their positioning on the chess pieces, and experimenting with alternative magnet shapes and distances between magnets might help minimize magnetic influence on neighboring sensors. We would also perform more stress tests to see how many sequential games the system could handle before the software would crash, since all the bugs in our end product result from the hardware. Developing a comprehensive suite of edge case tests for the chess logic would ensure proper handling of unusual gameplay situations and potential errors in sensor readings.

By addressing these testing considerations and incorporating additional verification procedures, we can significantly improve the accuracy, reliability, and user-friendliness of the smart chessboard for future iterations.

8 Budgets

Component/Service Name	**Price**	**Source**
Raspberry Pi	Provided	TAMU CSCE Department
8gb Micro SD Card	Provided	TAMU CSCE Department
Individually Addressable LED Strip	$20.00	https://www.amazon.com/dp/B01CDTECSG?tag=diymachines04-20&geniuslink=true&th=1
22 AWG Solid Core Wire	99$15.99	https://www.amazon.com/dp/B088KQFHV7/ref=sspa_dk_detail_1?tag=diymachines04-20&pd_rd_i=B088KQFHV7&pd_rd_w=omXkR&content-id=amzn1.sym.999c0877-3704-4f0f-9726-eebf80846a35&pf_rd_p=999c0877-3704-4f0f-9726-eebf80846a35&pf_rd_r=K9QNSRT3HBZE02H358T6&pd_rd_wg=jZdfw&pd_rd_r=bddbaa7c-04cd-4b8c-9286-def3862b22bc&s=gateway&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEzRTY0NjBIT0hJMThUJmVuY3J5cHRlZElkPUEwNTkxNTgyM1BQM1I2N0ExUDZHTyZlbmNyeXB0ZWRBZElkPUEwMTcxODcxMTZRUlNDQjlBQUUzMyZ3aWRnZXROYW1lPXNwX2RldGFpbCZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU&th=1&geniuslink=true
3D Printer Access	NA	https://fedc.engr.tamu.edu/
Super Glue	Already Own	https://www.amazon.com/Loctite-Control-4-Gram-Bottle-1739050/dp/B00ELV2D0Y/ref=sr_1_8?dib=eyJ2IjoiMSJ9.s2Rmm32-8ukUs0LQE4j0MFtRLMIeqwkp-

		Y6umv2H0wxQFfKEib-8SQtBbj2_i0pAg5T0d6wF4lbupniJ2POztHXBVQjgkd2jDrOPS0-Vf2AaLq-1QsND0egQORWUPAlWTxrmTTSkoEDM_eHbTe10oWp3O_55GOWR6dchFpOZVAEZSu-o5rmYi20pNbgzskw3duim7zIfcD91IIDLYewrYNOJ402l8T_aZErtUpnDpxo.pKhO_zA5-ceWJGTjUdCyptqUQ9HeTlgM-8yqMrWCV2M&dib_tag=se&keywords=super+glue&qid=1714502561&sr=8-8
80 Hall Effect Sensors (A3144)	$21.98	https://www.amazon.com/EPLZON-Effect-Magnetic-Detector-Arduino/dp/B09PG3PGH6/ref=sr_1_3?crid=3BGNEQAVMVUCM&dib=eyJ2IjoiMSJ9.OtS-R5fNIFZWX3SHZ0XJgY9uBj_w1ORL0oH4Q63xdwqhVcfjxmciv0ehMjb9uGaNhkfTVkfmo3ra8PmR-ZcEJwJMUlWKfSKn3J8YaYaeFu60ZogCZRRHU8_OJVIAtYtAT1ebhe4Ul1J7SPMP-OQ-vVvhpyLqOyfSPRenxiS2kJ3DqlfW41rDL_Evn1ekSWYNLBY_GxqmFsMEbLSS4jodaYUg-c3N-yW5qGRfIgxYd6g.DHS3_zvLMv_Mje5FblGMAjs5voLnnObnYUYgmszuaNg&dib_tag=se&keywords=hall+effect+sensors+hall+effect+sensor&qid=1708451732&sprefix=hall+effect+sensorshall+effect+sensor%2Caps%2C89&sr=8-3
64 Neodymium magnets	$39.90	https://www.magnet4less.com/neodymium-magnets-1-in-x-1-32-in-n42-disk
Female USB to DIP 5 board	$5.00	https://www.amazon.com/HiLetgo-Adapter-Connector-Converter-pinboard/dp/B07W844N43
OLED Screen	$11.99	https://www.amazon.com/iPistBit-Raspberry-Touchscreen-Monitor-Compatible/dp/B0BJJN9FZN/ref=sr_1_1?crid=3S5TZYWHLPCVB&dib=eyJ2IjoiMSJ9.41FsfEfx_tOQUWTyaqTCiL_o3ed2aW1drqy4dsQBGDn9dcFCD7sesvK87scxCKuAJ-GR24-Pg_HtVjGj3W9MZtMCLfoOPq9nW4J4CuQMTBhWwLCxAwl702c8UovGXjw9xdEBBHjEyixfe3YwYgO98qY9f34yLu3Rj1h3qgac_ZJUWh_s5Jpc_aAppBIipjXwV7GoYyaCc3HFjSc1x_3T1FjNlZ7MjHyvrGjU297ZKis.iUkFcM8ibuCGolCSS_hFnN6MHIutyvLVX8i1PWjb5OM&dib_tag=se&keywords=raspberry%2Bpi%2Busb%2Bscreen&qid=1708457811&sprefix=raspberry%2Bpi%2Busb%2Bscreen%2Caps%2C127&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th

		=1
CD74HCT251E 8:1 Multiplexors	Already Own	https://www.digikey.com/en/products/detail/texas-instruments/CD74HCT251E/38458?utm_adgroup=Integrated%20Circuits&utm_source=google&utm_medium=cpc&utm_campaign=Dynamic%20Search_EN_Product&utm_term=&utm_content=Integrated%20Circuits&utm_id=go_cmp-120565755_adg-9159612915_ad-665604606680_dsa-112117096155_dev-c_ext-_prd-_sig-CjwKCAjwrcKxBhBMEiwAIVF8rDiJgEFP0FihDo3zbprQ3aWZa_pDJT2Rs29HHAuyCpT_lqg4a7vNkhoClC0QAvD_BwE&gad_source=1&gclid=CjwKCAjwrcKxBhBMEiwAIVF8rDiJgEFP0FihDo3zbprQ3aWZa_pDJT2Rs29HHAuyCpT_lqg4a7vNkhoClC0QAvD_BwE
80 Varying Resistors	Already Own	NA
Prototyping Poster Board	$5.50	https://www.target.com/p/10pk-28-34-x-22-34-poster-board-white-up-38-up-8482/-/A-17089030#lnk=sametab
Double Sided Tape	Already Own	NA
Chess Pieces	$10.99	https://www.amazon.com/AMEROUS-Wooden-Figurine-Chessmen-Staunton/dp/B0747MFFGL/ref=sr_1_1_sspa?dib=eyJ2IjoiMSJ9.oe32XtbeBNKZC5OBZtKRO9TPkS-rKcDzOlZ_28ZfNCTmzZfl8jtuc7YnU2wbzZPUpJ5SbRg8Q1yTsxEVHEkjvQkfNTrpQ8czdVhuendOMWCUB8by4q0-Sqo0EloiDKpduyjRBURpMns3a1X-OufN846kx6jPqK9kIV5qEZHHdbxyJI6nIlx2z7ZhD8VdqNUVeHJbuMQYKvLF4wJLm7Xv6j-O8oxhz__j58cVCz9uhC81lB6MeQXc2J41spmfeP6GqhjJbKWGpxwPDWY6uaUoMBwjJjX5jbZLAm71qJyeTQU.lj6F62gMnwZiv-5OcwIQ9Zx1P0a7rS4gRAsHnSX8_GQ&dib_tag=se&keywords=chess+pieces&qid=1714503215&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1
Canvas	$12.00	https://www.amazon.com/Simetufy-Painting-Canvases-Acid-Free-Watercolor/dp/B0CQFTW7MH/ref=sr_1_5?crid=1CYYV8GPJBYZE&dib=eyJ2IjoiMSJ9.dwWvY6pQAL9QizIVBirUqREZ_UlyAoeyaWJBoIAOMmv7BAWnrAAz4oGHRnMMG3V_SWuIZ3O05G2C

		TAVa35JiNbWTyRDT0MH8H1mTPNRYkVftJfZ3Q8S5E7mi9XGGKFB754Bb98TD_4KyxGYaQaMepbaf0VNbWfbnv0JZDtr9qZJ2UjuIimHNaz1ckl7AJ9sYBamhmBZYk-W8Qhwam1UY_5ePTeS-qeoStxeXhzz-B4ksMmFR6IO7EbggQUFKYWGTWNb0Wer2fZtZwejvMSdT44P8xtHGrOgU9ul59QixNrE.G6sy_nDCdtk05QS8l4jwzPXc8EXGhgQEdq0hxr3Se5s&dib_tag=se&keywords=4%2Bpack%2Bcanvas%2B16%2Bi&qid=1714503325&sprefix=4%2Bpack%2Bcanvas%2B16%2Bi%2Caps%2C119&sr=8-5&th=1
Plexiglass	Already Own	Home Depot

Total Cost: $150.35

The above table includes the costs that our team spent developing the Smart Chessboard. Some of the items in our proposal and CDR reports ended up not being purchased and were unnecessary, so we removed those items. They include the colored acrylic, arduino nano, logic level shifter, and varying screw sizes. We first decided to 3D print our housing instead of using acrylic, and when the 3D printed exterior didn't work, we used plexiglass along with canvas. Using an arduino nano ended up being unnecessary since we were able to connect the LEDs directly to the Raspberry Pi along with the hall effect sensors. We avoided the costs of buying logic level shifters by implementing voltage dividers with resistors we already had from previous electrical engineering labs, which were used to convert the 5v output signal from the multiplexers to a 3.3v signal that is safe for the Raspberry Pi GPIO. We ended up not needing to use screws to secure any of the components of our system, and used double-sided tape and super glue for securing our housing and the breadboards beneath the housing. Some of the components our team purchased ended up not making it into the final implementation, such as the OLED screen, the female usb to dip 5 boards, and the posterboard. I included these in the total cost since we already purchased them, but not including these components would save $22.50. Our team did have some of these items already available to us and some provided to us by the Computer Science department, so if someone had to purchase all the components from scratch, including the components not used in the final implementation, the total estimated cost would be about $243. A Raspberry Pi with an 8gb micro sd card could be obtained for $50, super glue for $5, the multiplexers for $10, resistors for $10, double sided tape for $10, and plexiglass sheets for $8. With these materials already available to us, our final costs were $150.35. This cost is significantly cheaper than the estimated cost in our proposal reaport, which was $232.84. If the Smart Chessboard was mass produced, they would likely use an embedded processor instead of a Rasperry Pi, which would be anywhere between $5 and $15 per unit instead of $40 per unit. The resistors, hall effect sensors, and multiplexers would all be cheaper if purchased in bulk, further reducing the manufacturing price per unit. The estimated cost per unit under mass production would be $180. This is higher than the cost of our prototype due to some of the materials and Raspberry Pi already being available for us.

9 Appendices

- **Product datasheets**

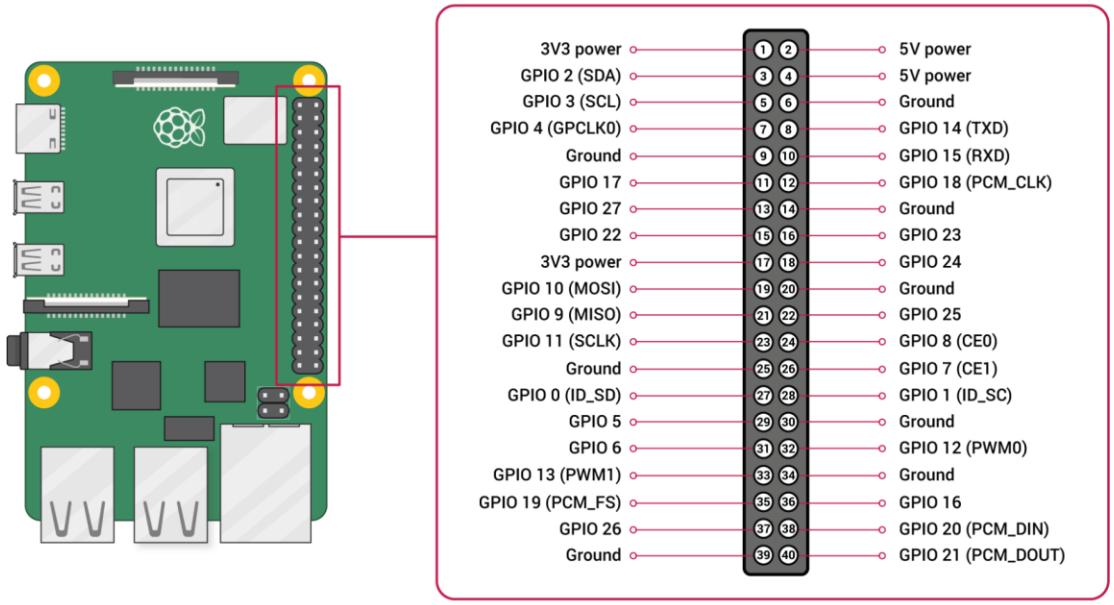
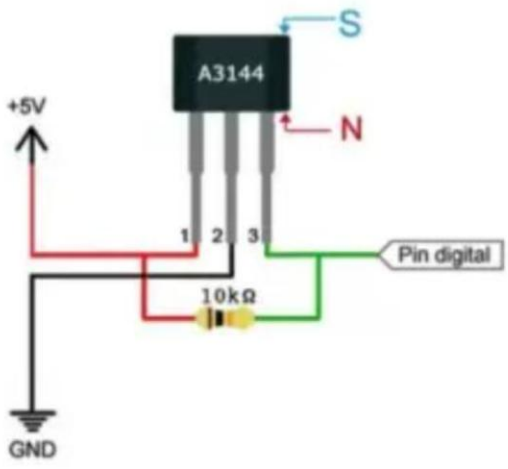


Figure 7: Raspberry Pi GPIO Pinout



A3144 Pinout

No	Pin Name	Description
1	'+5V (Vcc)	Used to power the hall sensor, typically +5V is used
2	Ground	Connect to the ground of the circuit
3	Output	This pin goes high if a magnet is detected. The output voltage is equal to the Operating voltage.

Figure 8: Hall Effect Sensor Schematic and Pinout

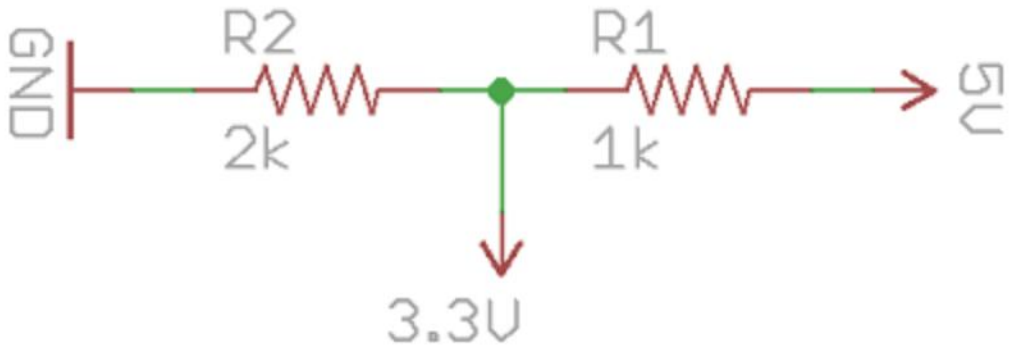


Figure 9: Voltage Divider Schematic

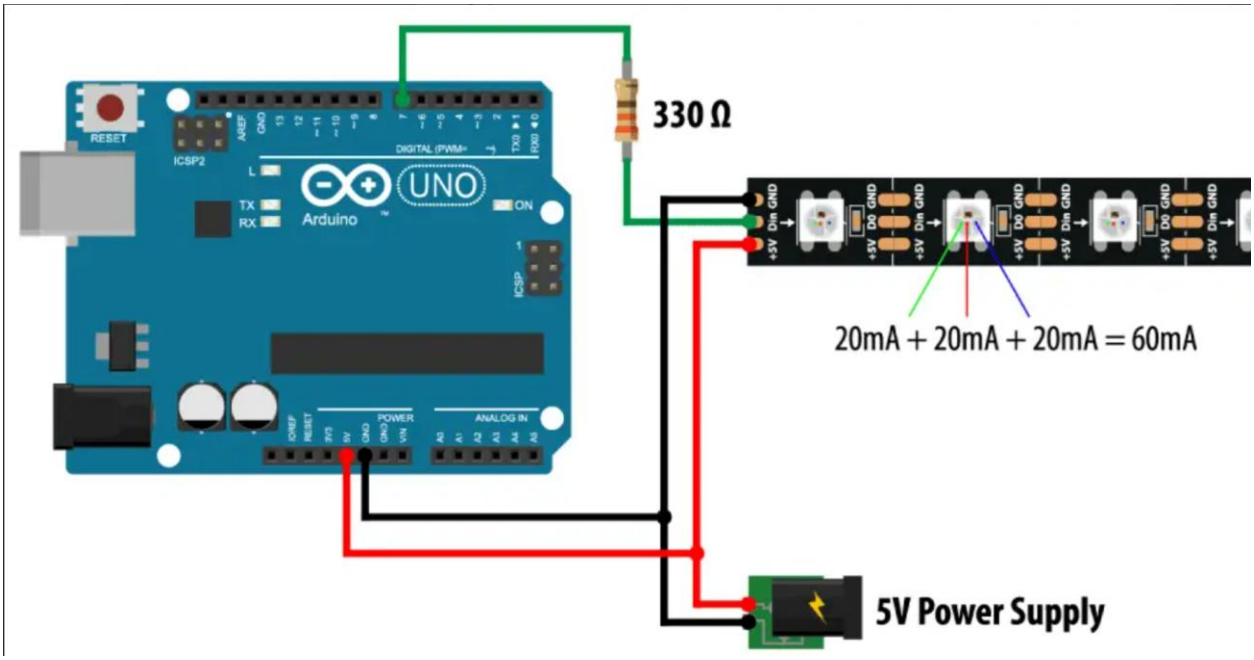


FIgure 10: LED Strip Schematic

10 References

"A3144 Hall Effect Sensor: Datasheet, Circuit and Pinout," *www.utmel.com*. https://www.utmel.com/components/a3144-hall-effect-sensor-datasheet-circuit-and-pinout?id=828

C. com Team (CHESScom), "Chess Board Dimensions | Basics and Guidelines," *Chess.com*, Oct. 04, 2021. https://www.chess.com/article/view/chess-board-dimensions

"CDx4HC251, CDx4HCT251 High-Speed CMOS Logic 8-Input Multiplexer, Three-State 1 Features." Accessed: Mar. 23, 2024. [Online]. Available: https://www.ti.com/lit/ds/symlink/cd54hc251.pdf?ts=1711045698159&ref_url=https%253A%252F%252Fwww.google.com%252F

Dejan, "How To Control WS2812B Individually Addressable LEDs using Arduino," *HowToMechatronics*, Jan. 13, 2018. https://howtomechatronics.com/tutorials/arduino/how-to-control-ws2812b-individually-addressable-leds-using-arduino/

"Home - Stockfish - Open Source Chess Engine," *stockfishchess.org*. https://stockfishchess.org/

"python-chess: a pure Python chess library — python-chess 0.2.0 documentation," *python-chess.readthedocs.io*. https://python-chess.readthedocs.io/en/v0.2.0/#:~:text=python%2Dchess%3A%20a%20pure%20Python%20chess%20library (accessed Mar. 23, 2024).

"Raspberry Pi Documentation - Raspberry Pi Hardware," *www.raspberrypi.com*. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html