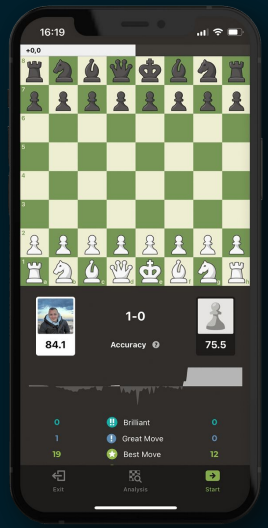
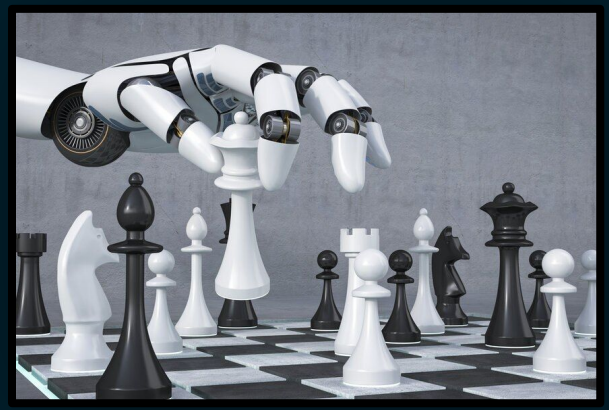
AI Chess Robot with Computer Vision

Spring 2024 Team 33
Zack Alonzo, Jose Flores, and Joshua Hur

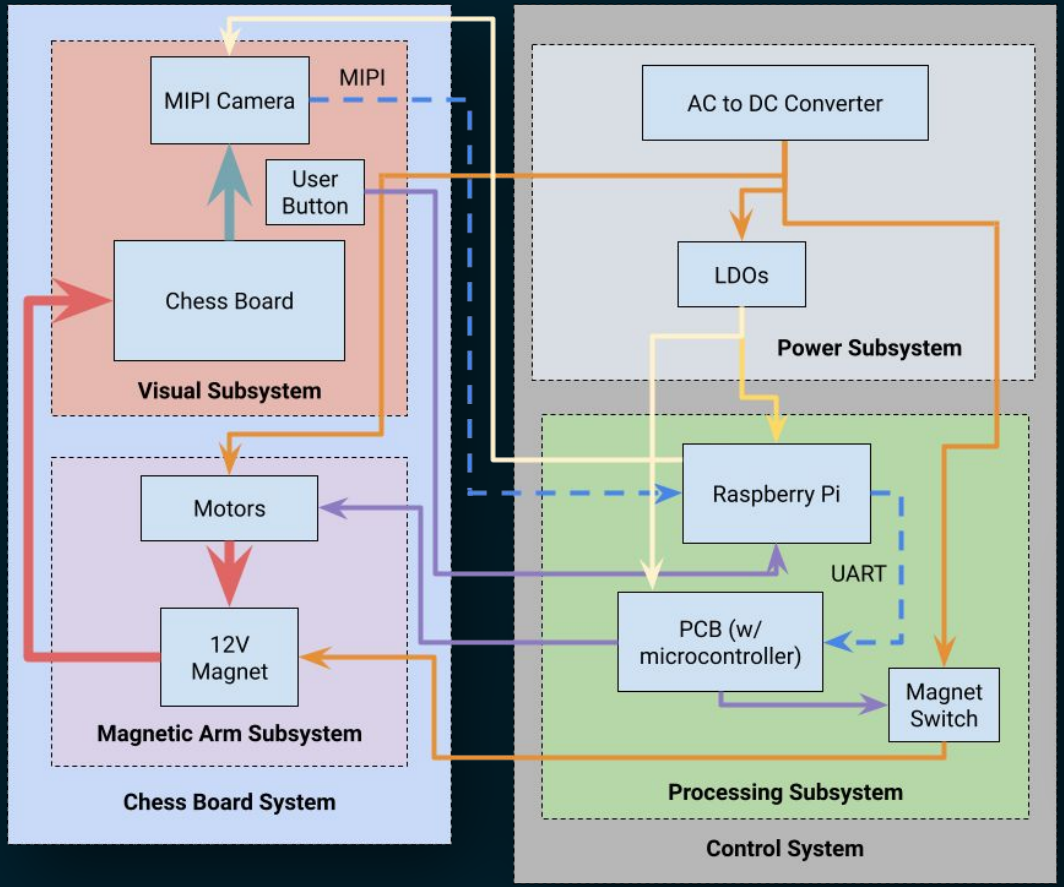
ECE 445 Senior Design Laboratory
University of Illinois at Urbana-Champaign

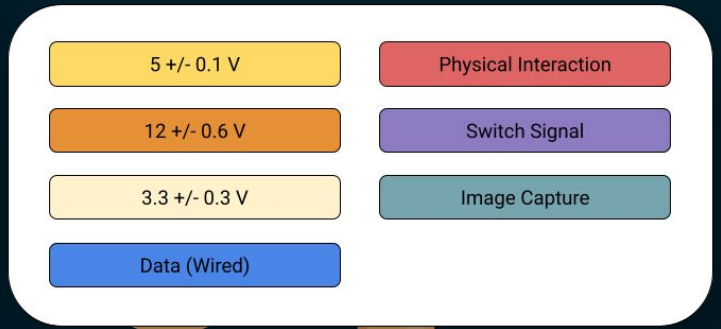
Introduction/Objective





Block Diagram





Requirements and Verification

01

Visual Subsystem

- Raspberry Pi recognizes camera and takes pictures
- Can detect the board with 95 ± 5% accuracy

02

Magnetic Arm Subsystem

- Rail system can move to correct square with 95 ± 5% accuracy
- Holds on to pieces and moves to the correct positions with 95 ± 5% accuracy without grabbing others

03

Processing Subsystem

- Correctly identifies chess piece and position with 95 ± 5% accuracy, as well as cheating
- Plans and executes correct path with 95 ± 5% accuracy

04

Power Subsystem

- 12V± 0.6V (stepper motors, magnet), 5 ± 0.1 V (Raspberry Pi), 3.3 ± 0.1 V (ESP-32)
- Magnet switch (MOSFET) is able to turn on and off the magnet

Code Overview

Computer Vision

Utilizing OpenCV and EasyOCR, we take a picture of the board, extract the lines, loop through each square, and detect letters and the letters' colors on squares that have changed since the previous image.

Chess AI Integration

We update the internal representation of the board with python-chess, determine the best move for the robot to make using Stockfish, and convert this into data we can send to the ESP-32.

Magnet Arm Control

Plan a path using A*, facilitate serial communication between the Raspberry Pi and the ESP-32 microcontroller, and execute the path with the stepper motors and the magnet.








Computer Vision using OpenCV and EasyOCR

Computer vision algorithm correctly identifies chess piece positions and their identity on the board with 95% ± 5% accuracy.

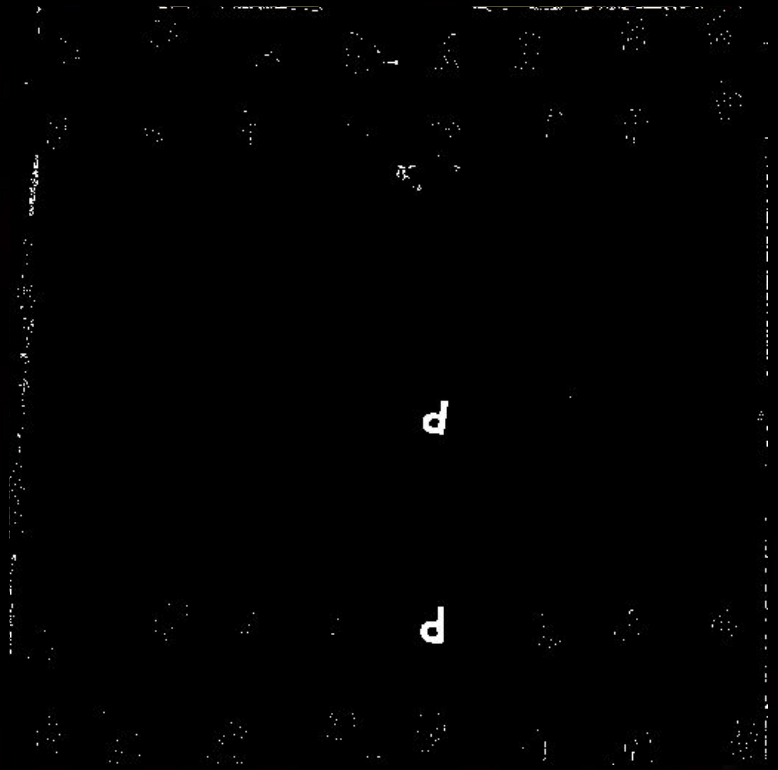
Optical Character Recognition


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

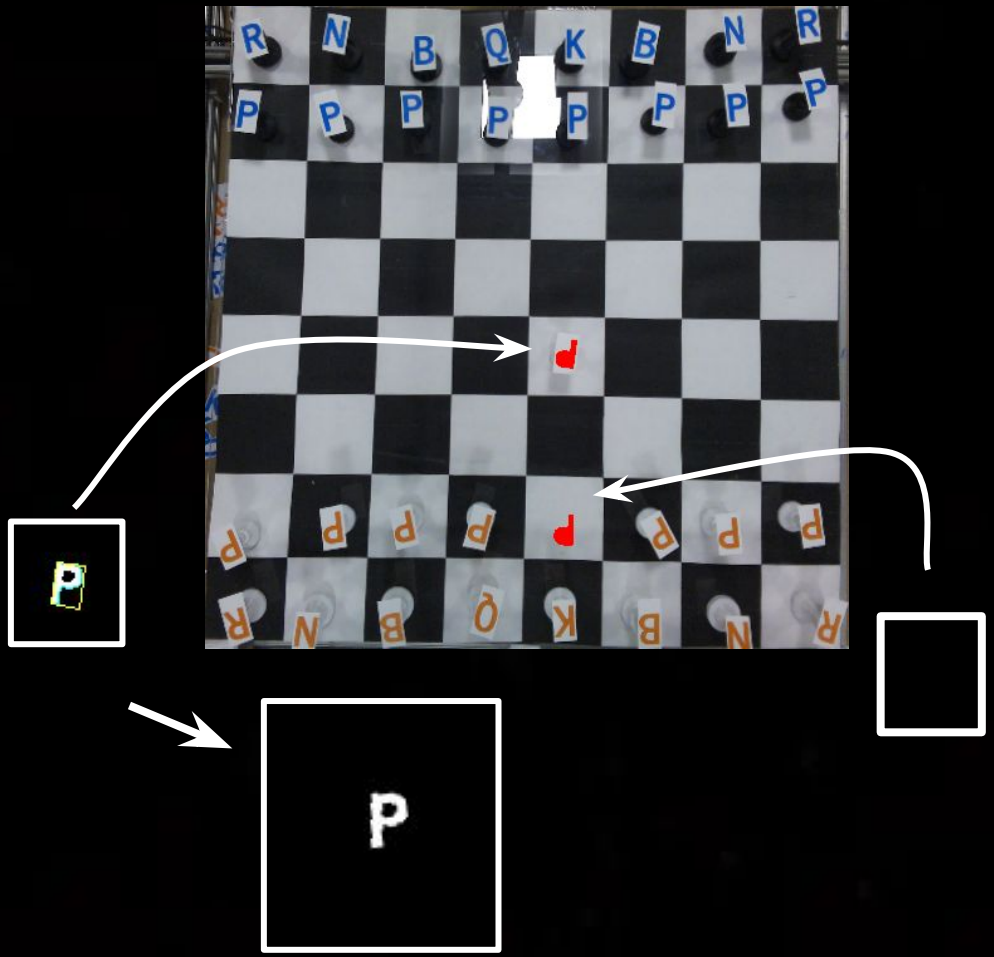
Previous

Current

Noisy



Filtered



```
P 0.9999364624579563 E4
X (remove) 0 E2
```

```
Found move! e2e4
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
```

Chess Robot Response

```
Found move! e7e5
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
rnbqkbnr/pppp1ppp/8/4p3/4P3/8/PPPP1PPP/RNBQKBNR w - - 0 2
```



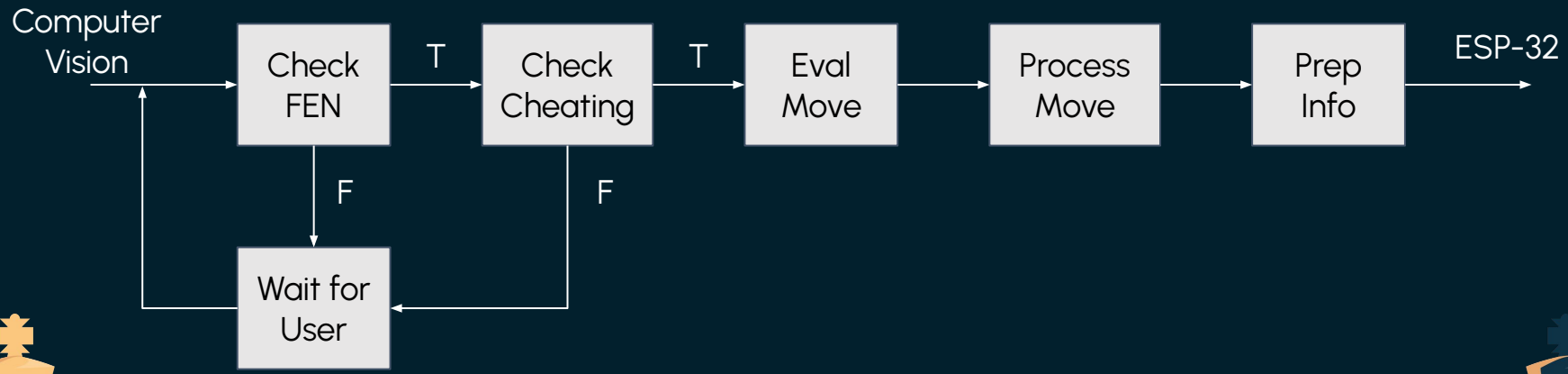





Chess AI Integration with Stockfish and python-chess

Chess AI is implemented in a way that is able to identify when the human player has cheated with 95% ± 5% accuracy.

Computer Vision to Motors Control Flow



Valid FEN Check

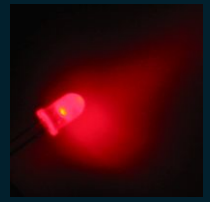
- Test 0:
  - Input: Valid FEN
  - Return: Pass
- Test 1:
  - Input: Invalid FEN
  - Return: Fail

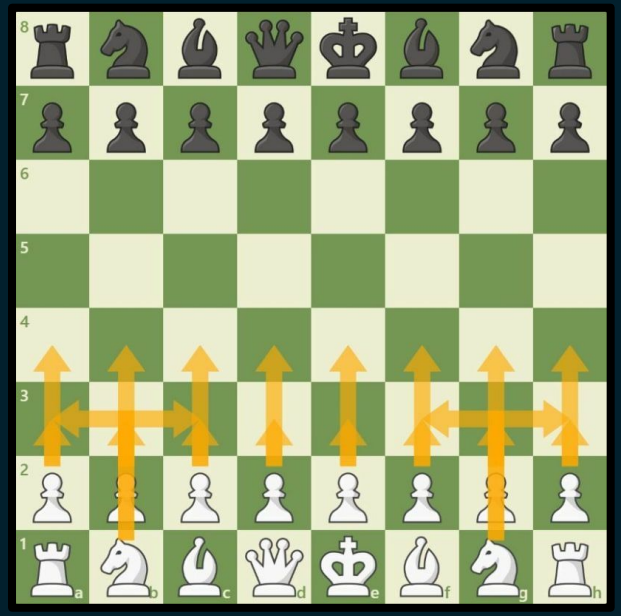
```
# === TESTS: FEN Parsing ===
if which_func == 0:
    is_valid_FEN = False

    # 0: Valid FEN (Success)
    if which_test == 0:
        FEN_string = "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"
        is_valid_FEN = FEN_check(FEN_string)

    # 1: Invalid FEN (Fail)
    if which_test == 1:
        FEN_string = "This should totally fail"
        is_valid_FEN = FEN_check(FEN_string)
```

```
PS C:\ece445> & C:/Users/jhur2/AppData/Local/Mic
You're currently testing: FEN Parsing's Test #0
This inputted FEN is: Valid
PS C:\ece445> & C:/Users/jhur2/AppData/Local/Mic
You're currently testing: FEN Parsing's Test #1
This inputted FEN is: Invalid
```

Cheat Detection

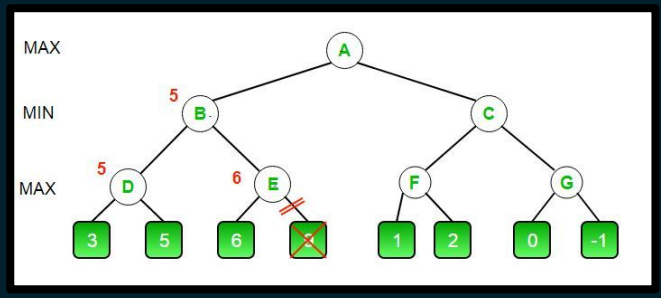
- Receives Prev_FEN and cur_FEN from Computer Vision
- Computes all possible plys from Prev_Fen
- Checks matches with cur_FEN
- If match, then no cheating
- Else, cheating

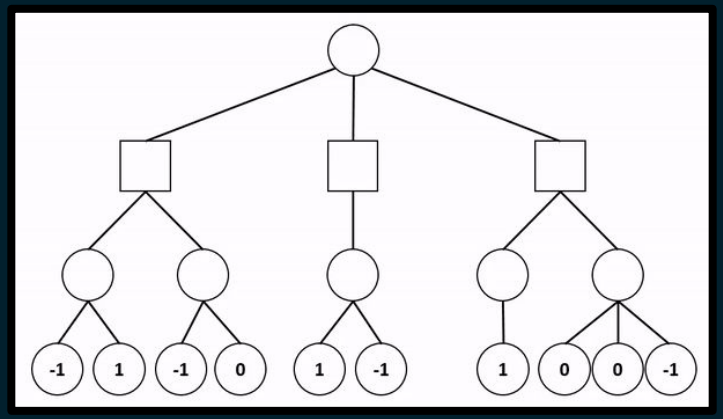




Stockfish

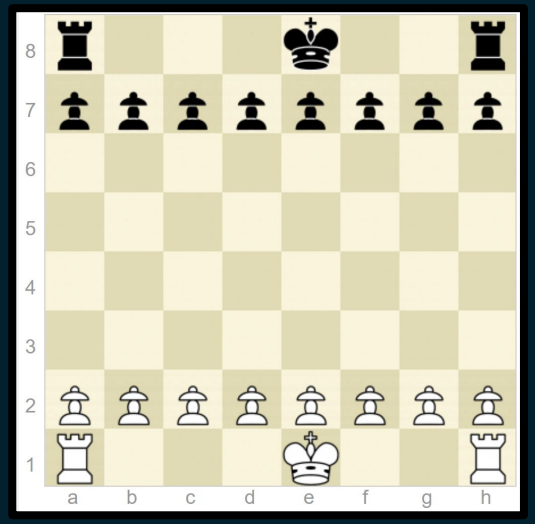
- Open source Chess engine
- Utilizes:
  - Minimax
  - Alpha-Beta Pruning

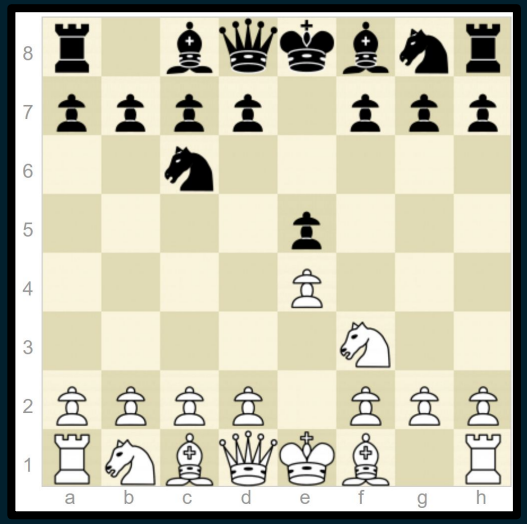




Special Move Check

Move: e1g1





Move: f3e5

```
PS C:\ece445> & C:/Users/jhur2/AppData/Local/Micr
You're currently testing: Special Move's Test #0
The move results in Castling at location: g1
PS C:\ece445> & C:/Users/jhur2/AppData/Local/Micr
You're currently testing: Special Move's Test #1
The move results in Capture at location: e5
```

Formatting Data for ESP32

```
You're currently testing: Processing ESP32 Info's Test #0
[['f 5', 'e 6'], ['e 5'], [0, 1, 0, 1, 0], '8/8/8/4pP2/4P3/8/8/8 w - e6 0 1']
```

```
You're currently testing: Processing ESP32 Info's Test #1
[['f 3', 'e 5'], ['e 5'], [0, 0, 0, 1, 0], 'r1bqkbnr/pppp1ppp/2n5/4p3/4P3/5N2/PPPP1PPP/RNBQKB1R w KQkq - 0 3']
```

- Process and parse data for ESP32
- Provide relevant piece locations and flags





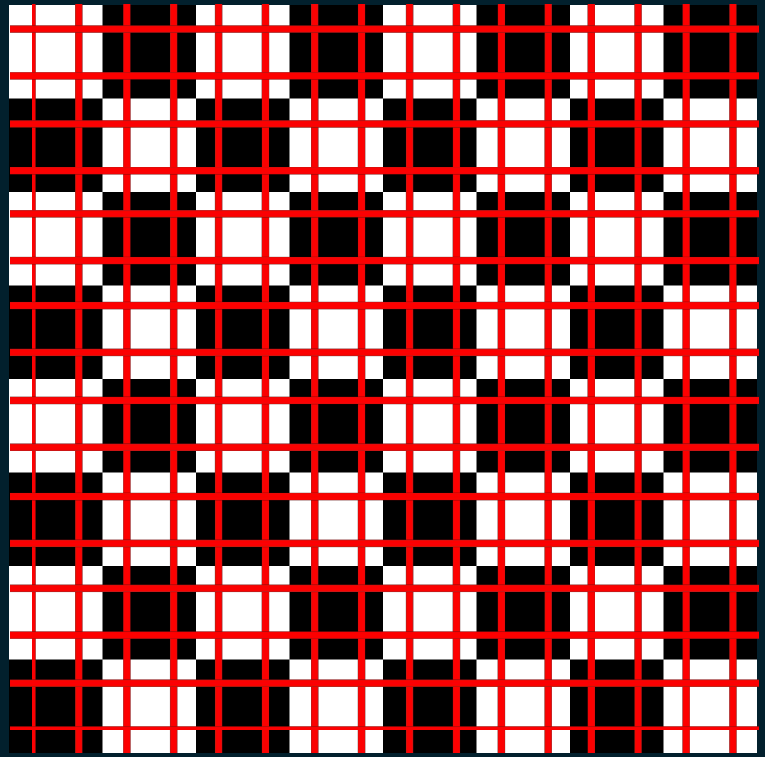



ESP32 Serial Communication and Magnet Arm Control

Rail and magnet system grabs the intended chess piece to the intended location on the chess board with 95% ± 5% accuracy.

Setting up Internal Representation for Motors

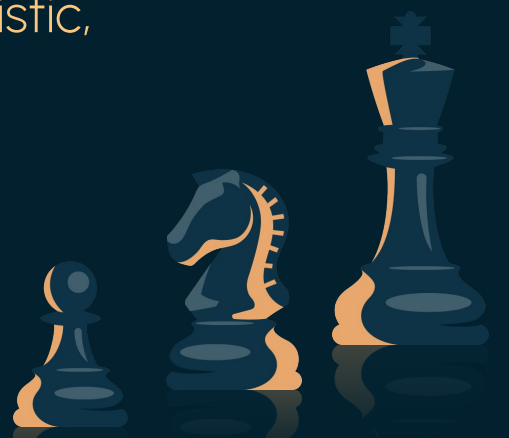
- From the ChessAI, we receive the FEN string that represents the board state before executing the Chess AI's chosen move.
- We abstract this string into a 17x17 array where each entry represents a possible location for the chess piece to move to (including the lines between the squares of the board).



```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

A* Search

- Using the internal representation we constructed earlier we can find the shortest path using A* Search
- Starting index and goal index was given by Chess AI as a chess move (Ex: E4, F6) and translated into array indices.
- Used modified Manhattan Distance as the heuristic,



```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[(7, 15), (6, 15), (6, 14), (5, 14), (5, 13)]
```

Serial Communication With ESP-32

- Transform the path array into a string to send to the ESP32
- Semi-colons mean the end of a coordinate, X will always come first, followed by Y (Ex: (4,6) -> 6;4;)
- ESP32 reads the message until it reads end-line character which is '>'

Example Message

```
[['e 2', 'e 4'], '', [0, 0, 0, 0, 0], 'rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1']
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

```
[(13, 9), (12, 9), (11, 9), (10, 9), (9, 9)]
```

```
9;13;9;12;9;11;9;10;9;9;>
```

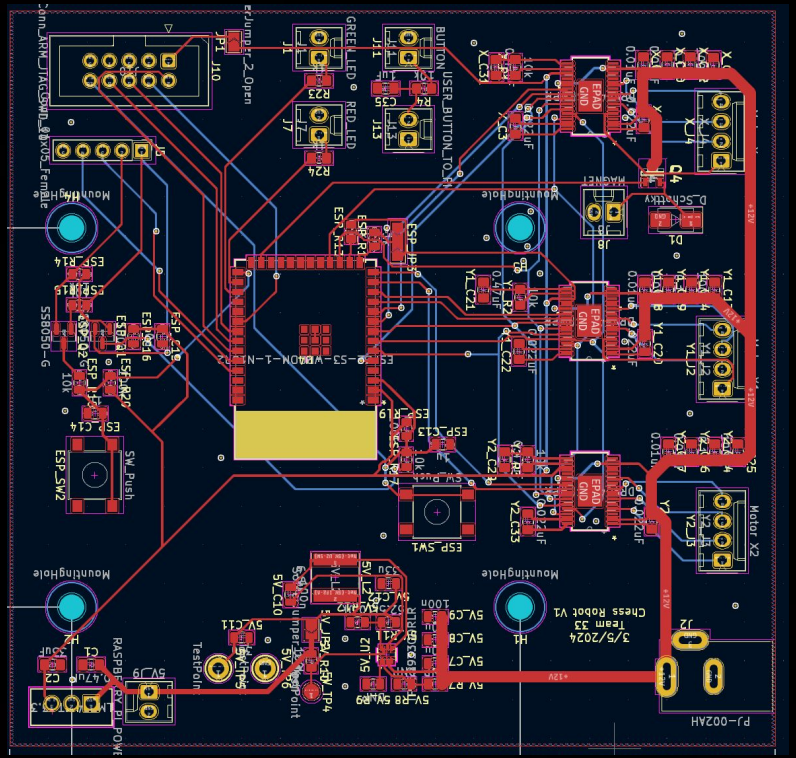
X, Y

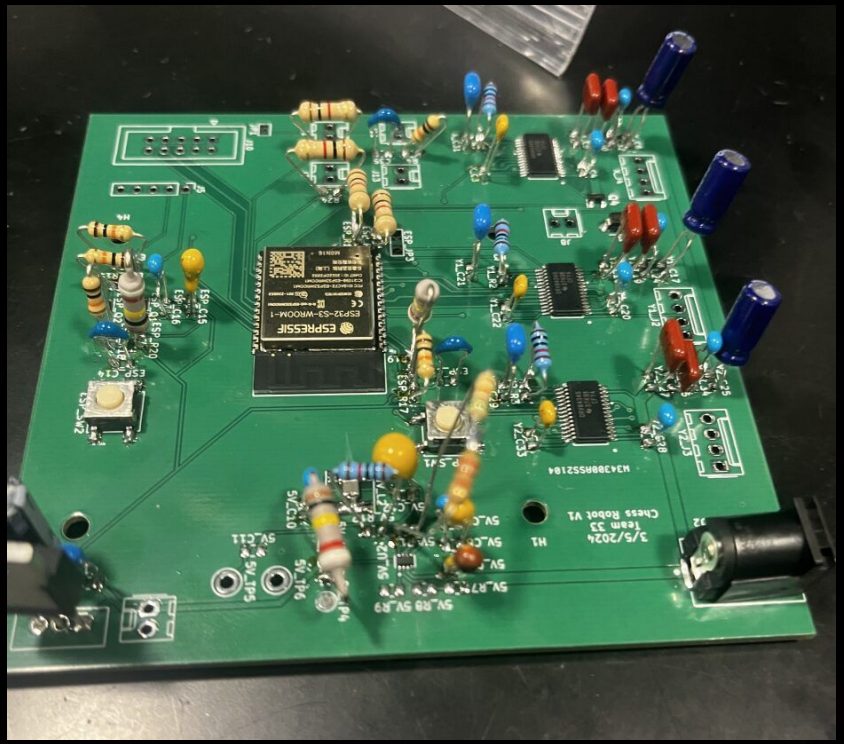
Motor Controller

- After string is received, it is parsed for the necessary information.
- 1 complete rotation is 200 steps, the board is about 1900 steps long in both X and Y
- Gives us about 1900/16 = 118.75 steps per open space
- Using this constant and the coordinates in the message, motors are moved with high precision to the desired location

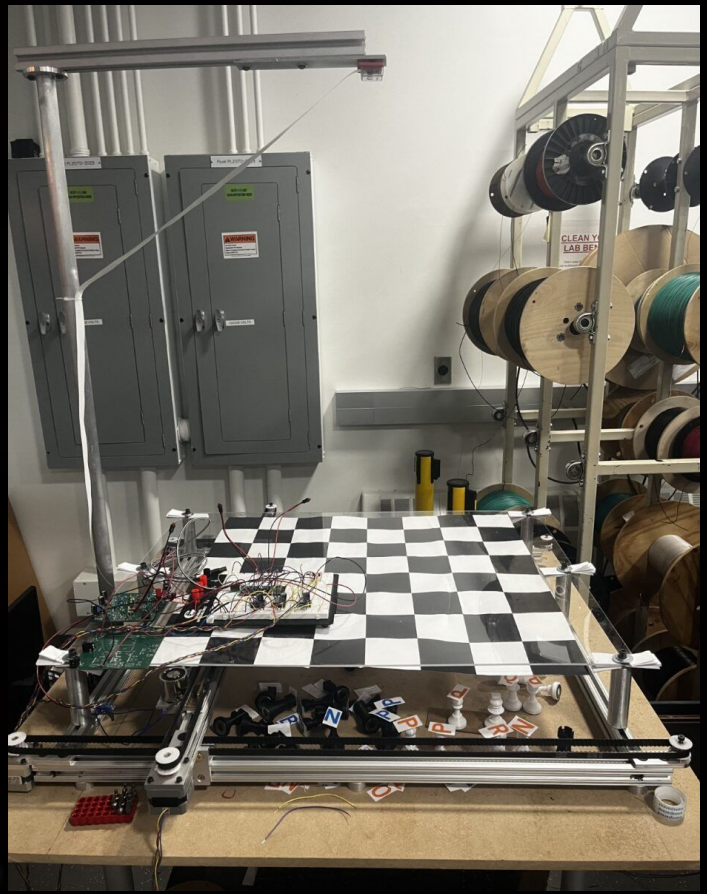
PCB Design and Challenges

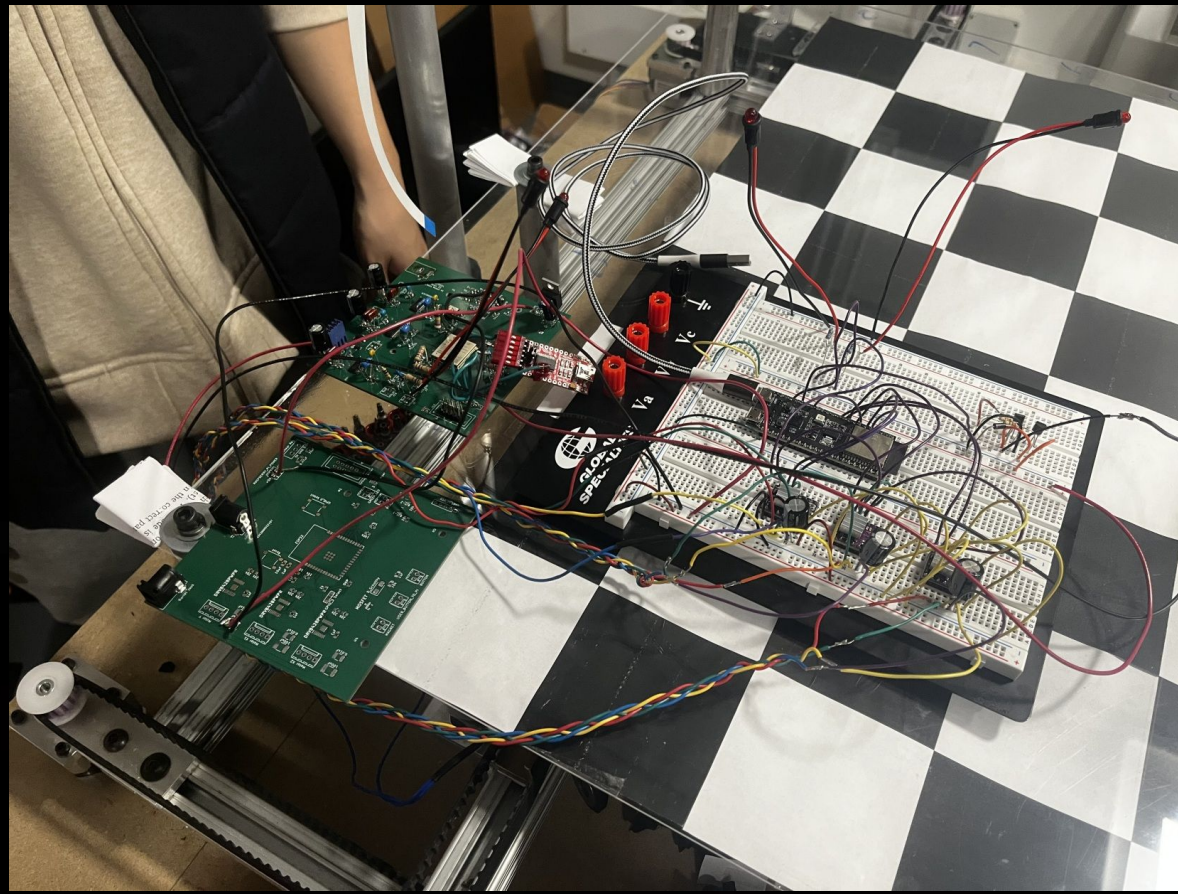
Anticipated Implementation



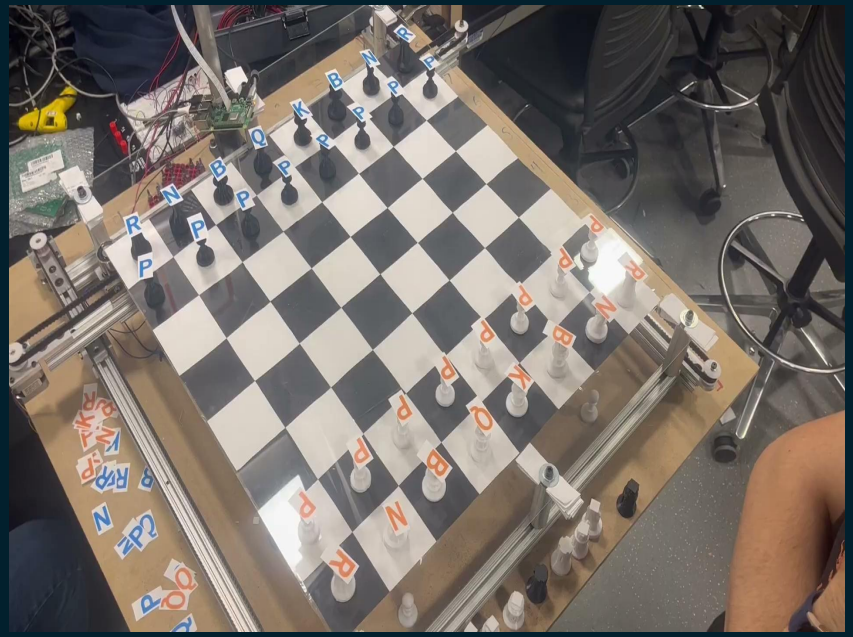


Actual Implementation



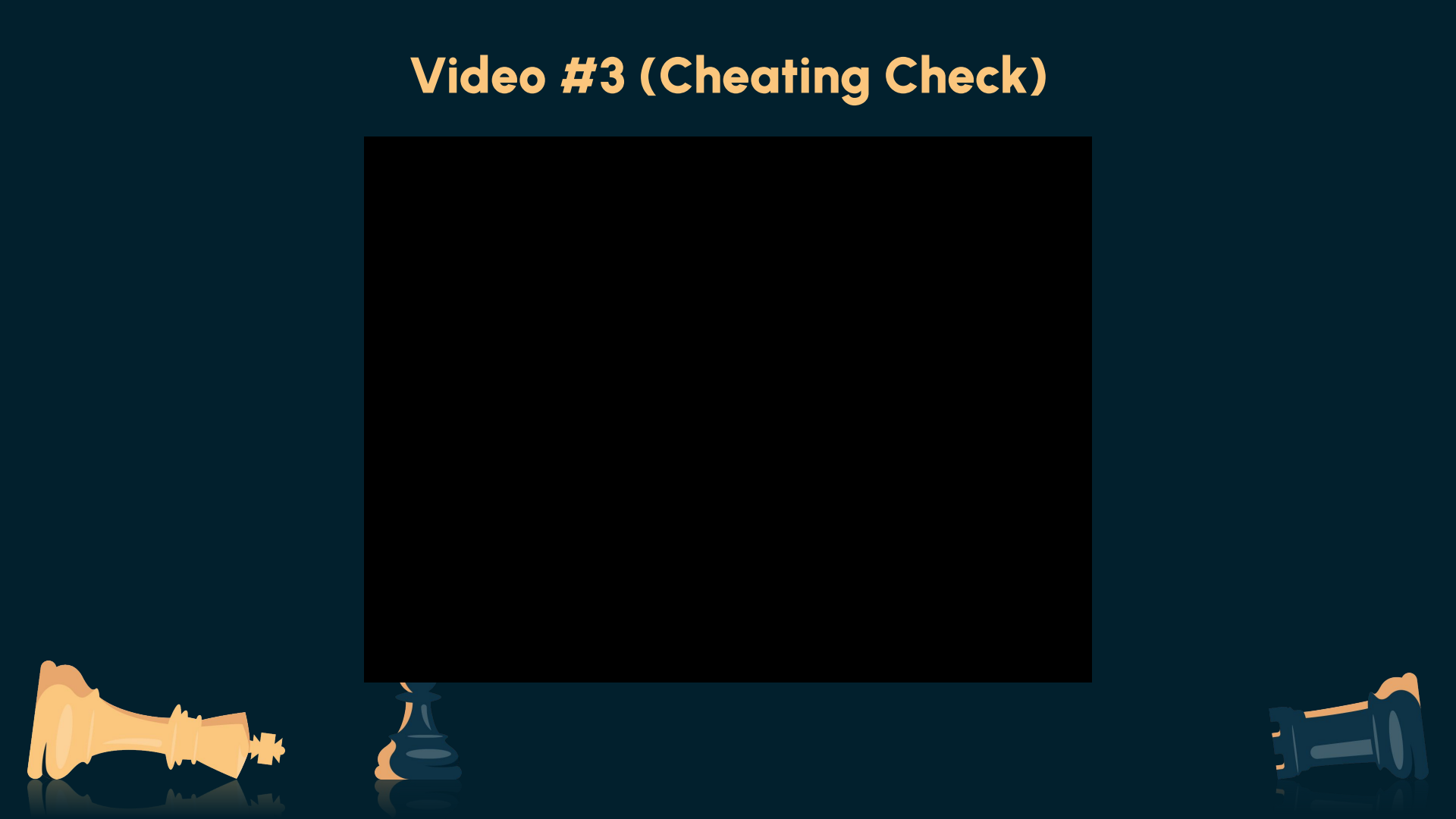


Video #1 (4 Moves)



Video #2 (Capture Move)

Video #3 (Cheating Check)



Conclusion

Future Goals

Fix PCB, ensure en passant, promotion, and castling works with a higher accuracy, implement OCR and Chess AI ourselves, and expand to different games