King's Gambit

Cory Petersen, Humza Syed, Youssef Ibrahim, Sedat Guvercin, Jason Nguyen, Jack Kellaher, Matthew Kelly

Table of Contents

Overview

Introduction
Flow Chart

Elements

Camera Element
Engine Element
Gantry Element

Looking Ahead

Demo
Goals for the Future

Introduction

Inspiration

Chess is a growing field, requires two people

Background

Multidisciplinary
Collaborative

Thought Process

Three main elements
Challenges

End Goal

Create a working prototype

System Flow Chart

Camera

Physical Image → Digital Board

Engine

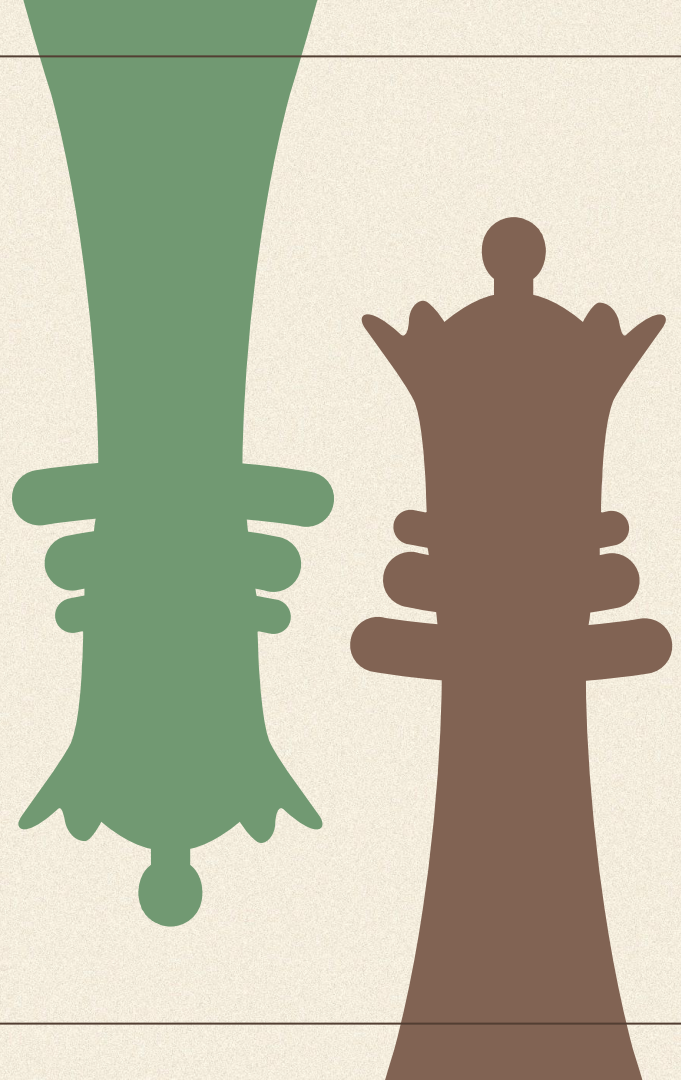
Digital Board → Chess Move

Gantry

Chess Move → Move Piece

Repeat

Reset and Wait for Opponent



01

The Camera

The Chess Robot's Eyes. This is how we know what was played and store the pieces' positions

The Camera

STEP 01

PICTURE

The first and easiest part is just taking a picture of the board

STEP 02

RECOGNITION

Using image processing we identify if there are pieces in certain squares

STEP 03

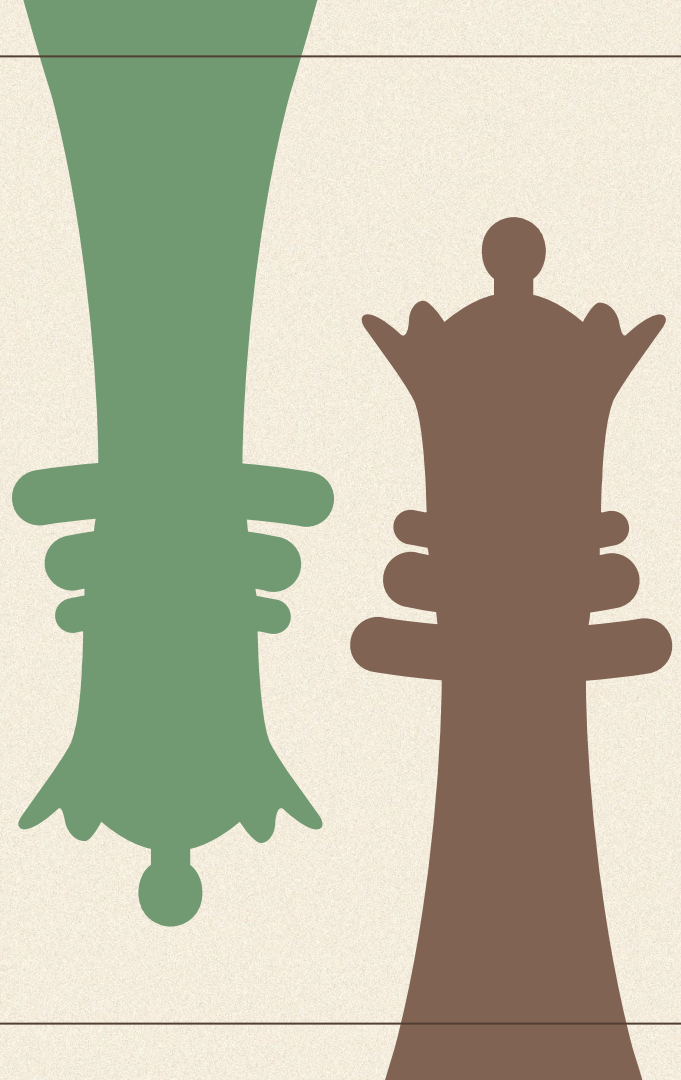
TRANSLATION

After detecting the positions of each piece, we generate a FEN notation

STEP 04

SHARING

Once we have FEN notation we need to share it with the chess engine

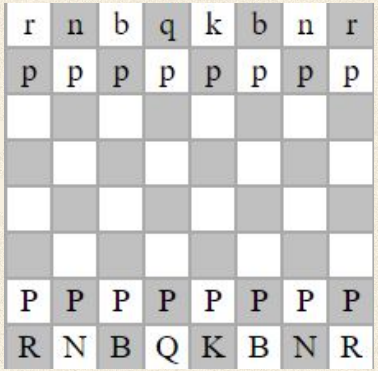


02

The Engine

The Chess Robot's Brain.
Using the final image we
can calculate the next move

STOCKFISH API

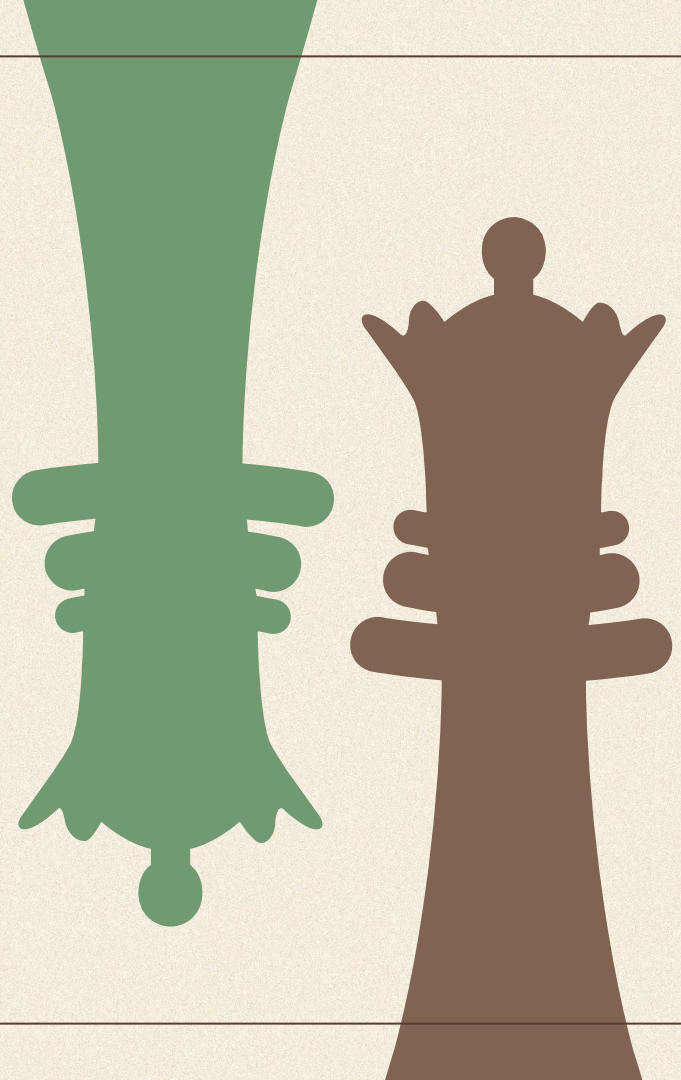


ENGINE

Stockfish, the chess engine, evaluates the chess board and returns the next best move, depending on the set difficulty

DELTAS

In addition to FEN notation, analysis of the board generates “deltas.” Through Arduino, the gantry uses this to calculate where the arm needs to travel



03

The Gantry

The Chess Robot's Arms.
This is how we translate
what the robot thinks and
sees to what it does

Gantry Components

Structure

The chassis fits around a 16” chess board and mitigates unwanted piece interference

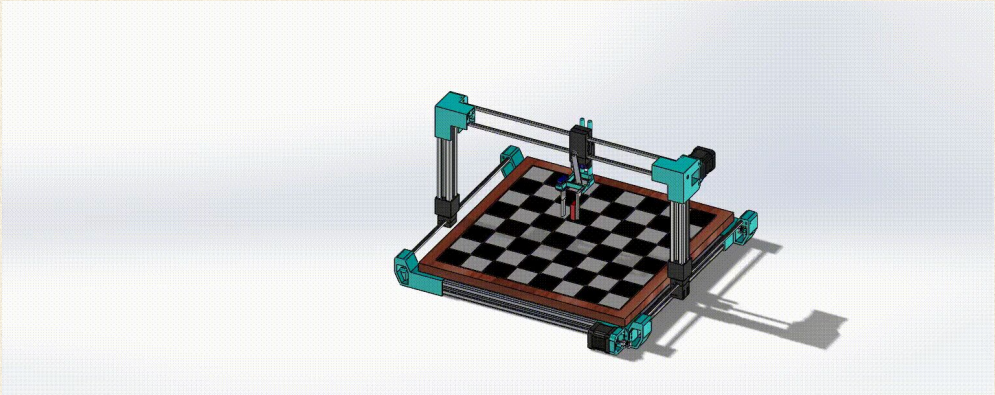
Electronics

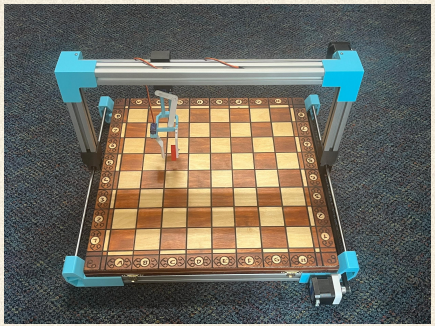
Electronics for directing motor movement are housed and cable-routed along the assembly

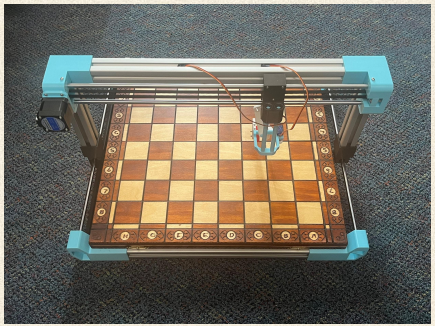
Actuation

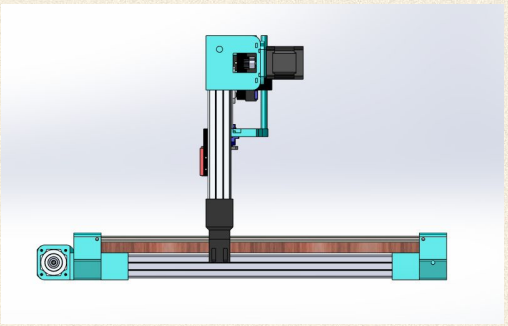
Stepper motors and servo motors actuate the gantry in discrete steps to pick up and move pieces

Design and Improvements

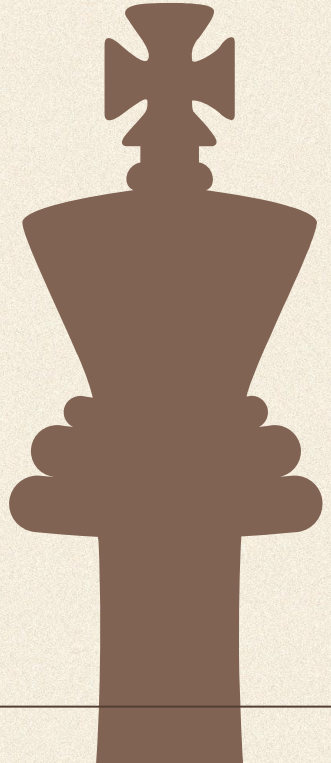


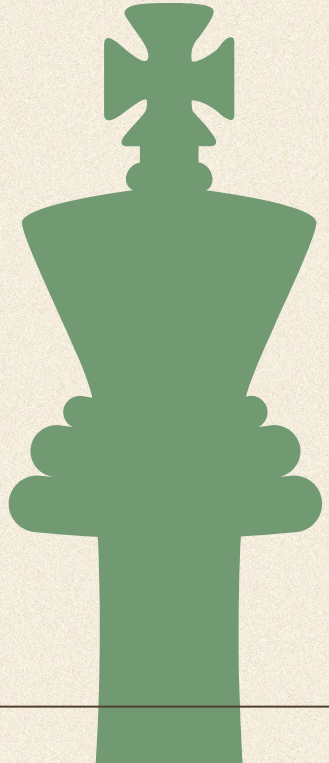


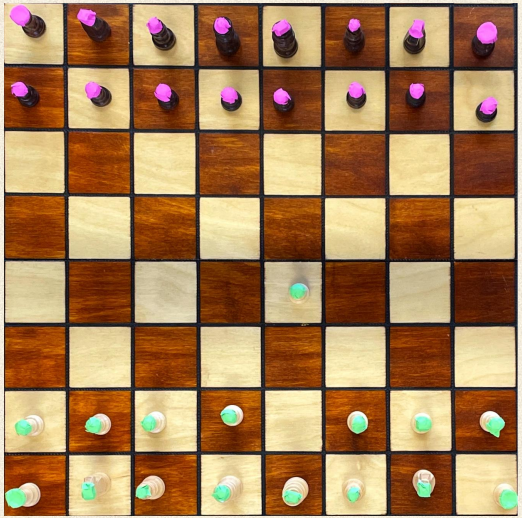




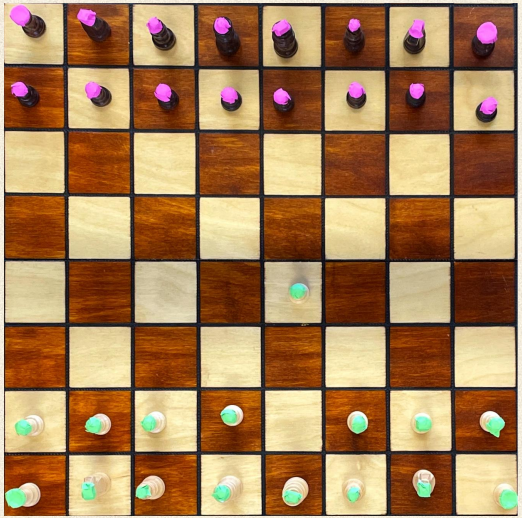
Demo



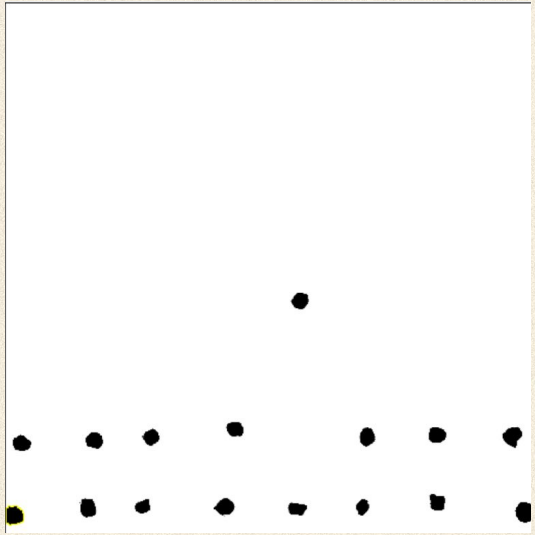




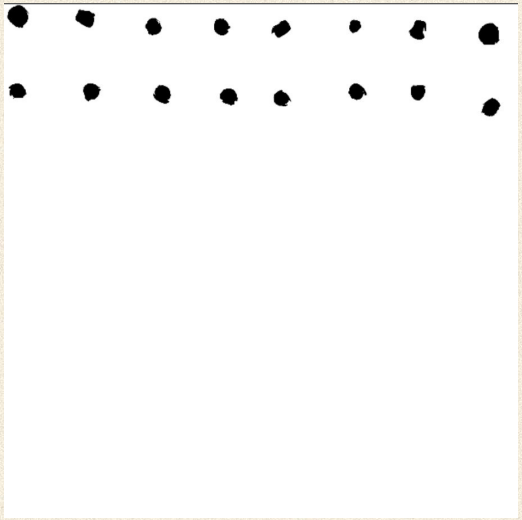
Takes Picture, Crops, and does Perspective Transform



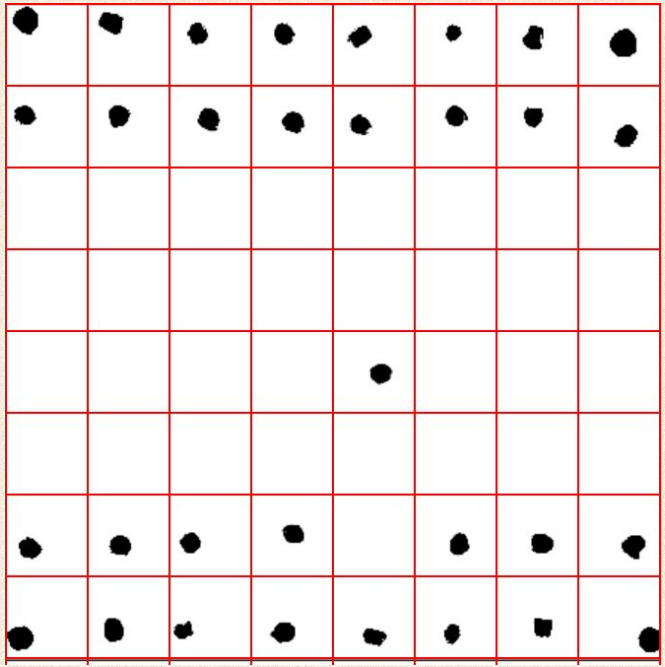
Takes Picture, Crops, and does Perspective Transform



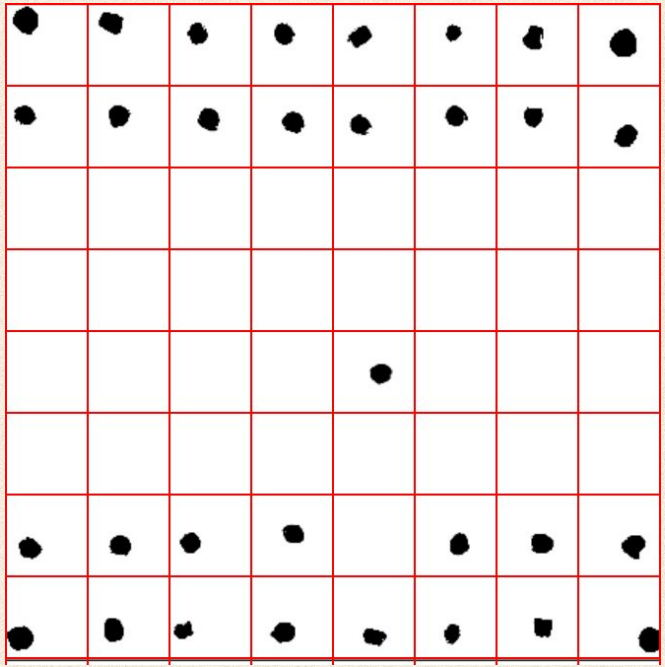
Creates Threshold Mask of User's Pieces



Creates Threshold Mask of CPU's Pieces



Creates 8x8 Grid Based on Current Positions After the First Move



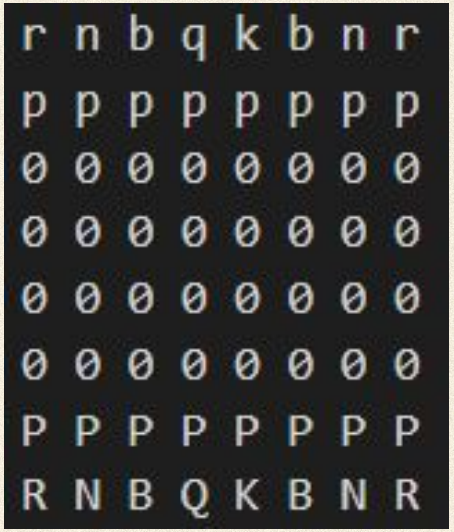
Creates 8x8 Grid Based on Current Positions After the First Move

```
2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 5 0 0 0
0 0 0 0 0 0 0 0
5 5 5 5 0 5 5 5
5 5 5 5 5 5 5 5
```

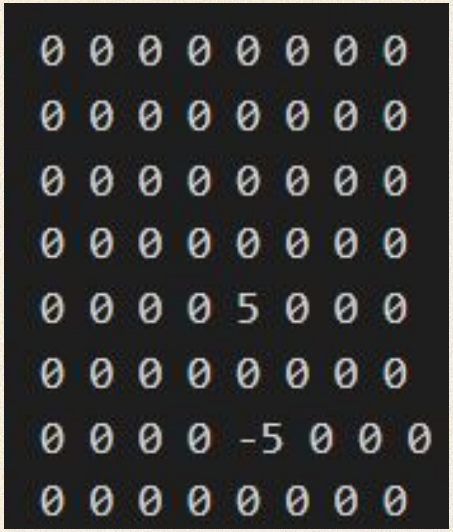
Array of Current Board State

```
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 5 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 -5 0 0 0
0 0 0 0 0 0 0 0
```

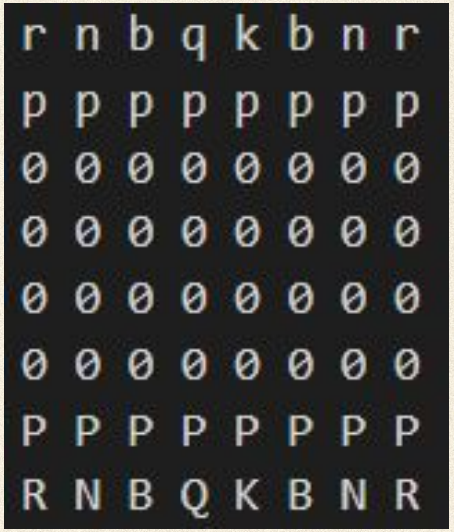
Array of Board Deltas



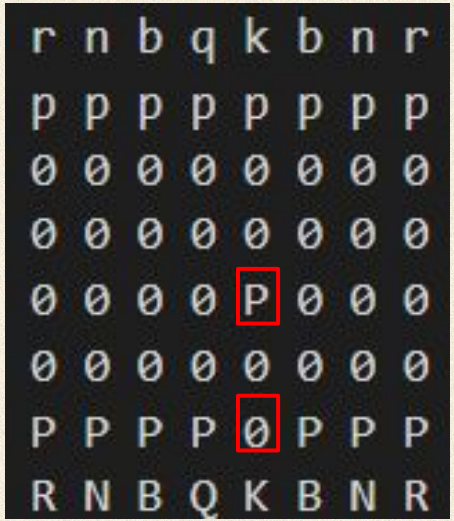
Old Chess Board Array



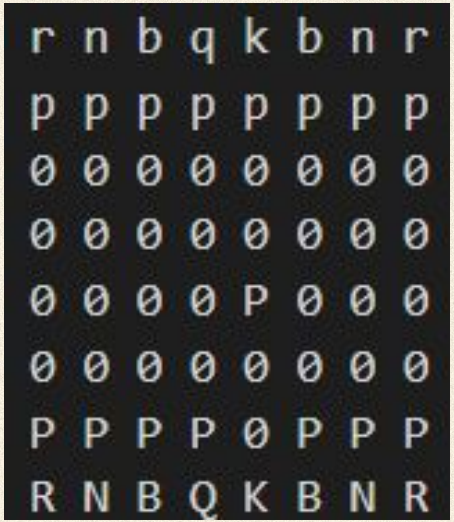
Deltas Array



Old Chess Board Array



New Chess Board Array


r n b q k b n r
p p p p p p p p
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 P 0 0 0
0 0 0 0 0 0 0 0
P P P P 0 P P P
R N B Q K B N R

```
r n b q k b n r
p p p p p p p p
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 P 0 0 0
0 0 0 0 0 0 0 0
P P P P 0 P P P
R N B Q K B N R
```

```
r n b q k b n r
p p p p p p p p
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 P 0 0 0
0 0 0 0 0 0 0 0
P P P P 0 P P P
R N B Q K B N R
```

```
rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR
```

```python
def StockfishComp (fen):
    fenvalidity = stockfish.is_fen_valid(fen)
    if fenvalidity := True:
        #sets the board position as current fen string
        stockfish.set_fen_position(fen)

        #next best move
        NBM = stockfish.get_best_move()
        stockfish.make_moves_from_current_position([NBM])
        newFEN = stockfish.get_fen_position()
    else:
        print('fen not valid')
    return [NBM, newFEN]
```

```
Next Best Move is c7c5
this would be the new FEN
rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR w KQkq - 0 2
```

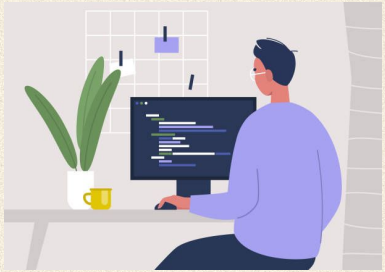
```
Next Best Move is c7c5
this would be the new FEN
rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR w KQkq - 0 2
```

```
Next Best Move is c7c5
this would be the new FEN
rnbqkbnr/pp1ppppp/8/2p5/4P3/8/PPPP1PPP/RNBQKBNR w KQkq - 0 2
```

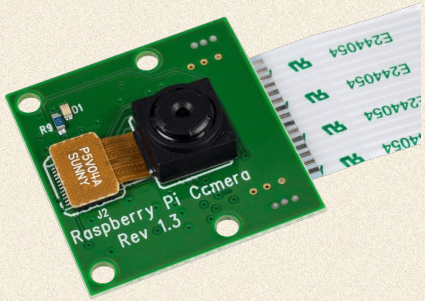
```
r n b q k b n r
p p 0 p p p p p
0 0 0 0 0 0 0 0
0 0 p 0 0 0 0 0
0 0 0 0 P 0 0 0
0 0 0 0 0 0 0 0
P P P P 0 P P P
R N B Q K B N R
```

```
Go to moving peice
Number of tiles to move: 5 left 1 down
New position after move: c7
grab
go to end position
Number of tiles to move: 0 left 2 down
New position after move: c5
```

Future work



Contact Github Developers for Insight



Built-in Camera

Cost Efficiency Optimization

Thank you!
Questions?

Backup Slides

If a Piece is Captured

```
0 0 0 0 0 0 0 0
0 0 0 0 0 3 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 -5 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
if (len(neg_indices) == 1 and len(pos_indices) == 1): # Standard movements (no takes) or white takes black
    #for one negative value and one positive value
    mof[row2][col2] = mof[row1][col1]
    mof[row1][col1] = '0'
elif (len(neg_indices) > 1 and len(pos_indices) == 0): #black takes white
    #for two negative values
    row1, col1 = neg_indices[0]
    row2, col2 = neg_indices[1]
    if (abs(diff[row1][col1])>abs(diff[row2][col2])):
        mof[row1][col1] = mof[row2][col2]
        mof[row2][col2] = '0'
    else:
        mof[row2][col2] = mof[row1][col1]
        mof[row1][col1] = '0'
```

Fen Extras and Castling Rights

```
#checks if white can castle at all
if (mof[7][4] != 'K'): #checks to see if king moved
    value[1] = 'false'
    value[2] = 'false'
else:
    #checks white castle queenside
    if(mof[7][0] != 'R'):#checks to see if shortside rook moved
        value[2] = 'false'
    #checks white castle kingside
    if(mof[7][7] != 'R'):#checks to see if longside rook moved
        value[1] = 'false'

#checks if black can castle at all
if (mof[0][4] != 'k'): #checks for black king
    value[3] = 'false'
    value[4] = 'false'
else:
    #checks white castle queenside
    if(mof[0][0] != 'r'):#checks for shortisde rook
        value[4] = 'false'
    #checks white castle kingside
    if(mof[0][7] != 'r'):#checks for longside rook
        value[3] = 'false'

#active piece
if (value[0] %2 == 0):
    ap = 'b'
else:
    ap = 'w'
#move number
mn = int((value[0] + (value[0]%2))/ 2)
mn = str(mn)
```

```
b KQkq - 0 1
```

```
active peice is black
turn number 2
move number 1
white can castle Kingside
white can castle Queenside
black can castle Kingside
black can castle Queenside
```

```
rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR b KQkq - 0 1
```