****

**AI Chess Opponent Final Report**

Karim Zaher

[**1. Summary**](#_heading=h.gjdgxs) **3**

[**2. Introduction/background**](#_heading=h.30j0zll) **3**

[**3. System Design**](#_heading=h.1fob9te) **3**

[3.1 Hardware](#_heading=h.3znysh7) 4

[3.1.1 WS2812B LED Strip](#_heading=h.2et92p0) 4

[3.1.2 Pi Camera](#_heading=h.tyjcwt) 6

[3.1.3 Button, Buzzer, and 3 Feedback LEDs](#_heading=h.3dy6vkm) 6

[3.2 Software](#_heading=h.1t3h5sf) 7

[3.2.1 Chess Board Simulation](#_heading=h.4d34og8) 7

[3.2.2 Physical Board Mapping](#_heading=h.2s8eyo1) 9

[3.2.3 Player Movement Tracking](#_heading=h.17dp8vu) 15

[3.2.4 Opponent Movement Verification](#_heading=h.3rdcrjn) 16

[**4. BOM**](#_heading=h.26in1rg) **17**

[**5. Tentative work schedule**](#_heading=h.lnxbz9) **18**

[**6. Appendix**](#_heading=h.35nkun2) **19**

# 1. Summary

For my project, I will be creating an AI chess opponent. Although plenty of AI chess opponents exist in software nowadays, there are very few that are able to play on a physical chess board. My project focuses on creating an interactive chess board that allows the AI opponent to call out its moves through lights that indicate where the user should move the AI’s chess piece. This system is especially useful in instances where a user cannot interact with other players, either due to a pandemic or due to a lack of internet.

# 2. Introduction/background

When I was younger, I loved playing chess, especially on a physical chessboard. I had a chess application on my phone and computer, however, it did not feel as fun as playing with physical chess pieces. Physical chess boards make the game feel much more intense and immersive than a simulated chess game. The only problem with a physical chessboard is that when there is no other individual that is available to play a game of chess, one cannot play chess on his or her own. If there was a way to play chess with a chess AI on a physical chessboard, people would still be able to feel immersed in the game even when there is no human opponent to play against.

In today’s Covid-19 era, people are unable to be in the same room together. So when an individual wants to practice playing chess on a chessboard with another player, they are unable to. With an AI chess opponent, the individual does not have to worry about making contact with another person and risk getting sick in the process.

# 3. System Design

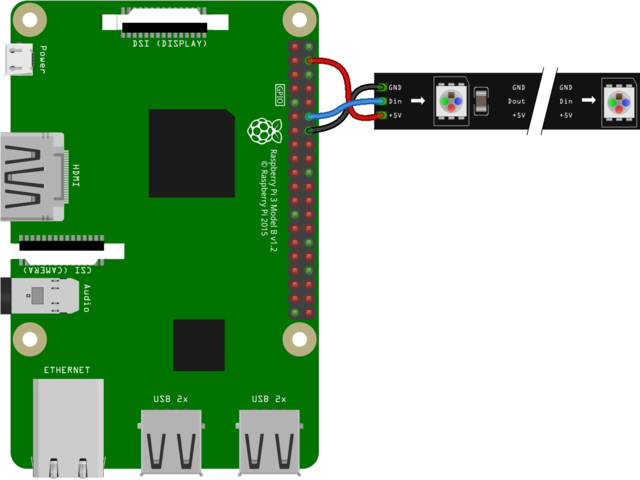
Everything is stored within a large storage compartment that is located underneath the chessboard. This storage compartment is typically used to store chess pieces, however, using it as a storage compartment for my hardware allows the end product to look clean. It also removes the need to use multiple breadboards or extension wires to get things connected to the raspberry pi if the part is located far away from the pi. To prevent any overheating from within the box, I created a massive hole on the side of the box to allow for continuous airflow. I also created another hold to allow wires to run through the compartment without getting in the way of the product. The black chess pieces were also covered in ripped up pieces of labels. This was done because of the Pi Camera’s low resolution which prevented the camera from being able to differentiate between black (chess pieces) and brown (grid color) pixels.

## 3.1 Hardware

Excluding the Raspberry Pi, this section covers the hardware components that are used in this project.

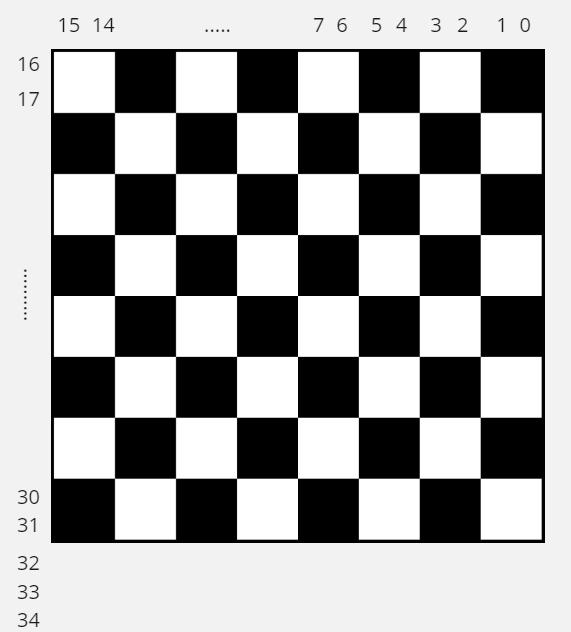
### 3.1.1 WS2812B LED Strip

Initially, I planned on using multiple RGB LED light bulbs to indicate what AI chess piece should be moved and the position of its new location. However, I felt as though the number of wires that I would have to use was terrifying in both the sense of debugging and plain aesthetic. After submitting my one-pager, my project was approved, however, I was told that LED lightbulbs have been a tough challenge before. Seeing as to how massive this project was, I realized that lightbulbs should have to be the least of my concerns, so I went out to find a better alternative. This is when I found out about WS2812B RGB LED strips. These LED strips are not only wired through a PCB, each LED is individually programmable.



While looking at LED strips, I realized that the shortest length I could find was much longer than the chess board itself. This was an issue, especially in terms of design. In addition, only one WS2812B LED strip can be connected to the raspberry pi at a time. I was able to find a WS2812B LED strip that can be cut and reattached at certain parts of the strip. However, they needed to be reconnected somehow. After more research, I found connectors. I purchased L-connectors, 45-degree connectors, so that the cut LED strip could easily curve around the edge of the chess board.

****



In total, there are 35 LEDs used in my project. The first 32 are used to indicate where the AI opponent’s chess piece should go. This is done by using 4 LEDs per grid position. For example, the position (0,0) uses 2 LEDs 14 and 15 on the X-axis and 16 and 17 on the Y-axis. LEDs that light up red shows what piece needs to be moved while LEDs that light up green show where the piece must be moved. The formulas below show how the LED indices are calculated:

* redX = 15 - currentPos\_x \* 2
* redY = 16 + currentPos\_y \* 2
* greenX = 14 - newPos\_x \* 2
* greenY = 17 + newPos\_y \* 2

The remaining 3 LEDs (32, 33, and 34) are feedback LEDs. They let the user know what is going on. Their significance is further explained in Section 3.1.3.

### 3.1.2 Pi Camera

To keep track of the chess pieces, I use a raspberry pi camera module. The camera is attached to an old lamp with a flexible neck. That way, not only do I save on any additional spending on a stand, I also have the ability to quickly and easily adjust the camera’s position until I get an optimal view of the chess board.

****

### 3.1.3 Button, Buzzer, and 3 Feedback LEDs

This product only takes 1 user-controlled input which comes from the button. The button is used to let the Raspberry Pi know three different things depending on what is going on:

1. If it is the start of the game and all three LEDs are on, pressing the button lets the Pi know that the chess board is completely cleared out.
2. If only the last LED on the light strip is on, pressing the button lets the Pi know that all pieces have been placed on the chess board.
3. If only the middle LED is on, which indicates that it is the user’s turn, pressing the button lets the Pi know that the user has moved their piece into a new position.
4. If it is the AI opponent’s turn, pressing the button lets the Pi know that the opponent’s piece has been moved to its designated location.

The buzzer lets the user know that the Pi has moved onto a different stage.



All 3 feedback LEDs enabled

## 3.2 Software

Libraries used:

* Pygame: For simulation and board set-up
* gpiozero: For buzzer and button functionality
* OpenCV: For computer vision and image processing

### 3.2.1 Chess Board Simulation

Using an 8x8 matrix, each piece on the board is kept track of using the Piece class which keeps track of the piece’s color, position, and whether it's the piece’s first move or not (useful for pawns since they move differently on their first move). Then 6 subclasses are used to keep a log of the possible moves that the piece can make.

These subclasses and their possible moveset are:

1. Pawn class:
   1. Hardcoded it so that the pawn can only move up to 2 steps right at the start, then 1 step after that. If there is a piece in its way, no matter the color, it cannot move forward. If there is a black piece in its diagonals, it can take the piece's position.
2. Rook class
   1. For the rook's algorithm, it checks to see whether there were any pieces blocking its x path or its y path. If there was a piece with its opposite color in its way, it could move to that piece's position and capture the piece. If it was a similar color to it instead, it could only move to the position right before that piece's position.
3. Bishop class
   1. The bishop functioned similarly to the rook algorithm, however, it took diagonal movements into account.
4. Knight class
   1. Knights move by taking 1 step in one direction and 2 in another. The direction does not matter as long as it is not off of the board. It cannot move to a spot if there is a piece equal to its color on that spot.
5. Queen class
   1. Combined both the bishop and rook algorithms to create the queen. This saved me a ton of time with having to reimplement something that I have already implemented
6. King class
   1. Can take 1 step in any direction unless there is either a white piece in its way or if it is in the path of an opposing team's movements.

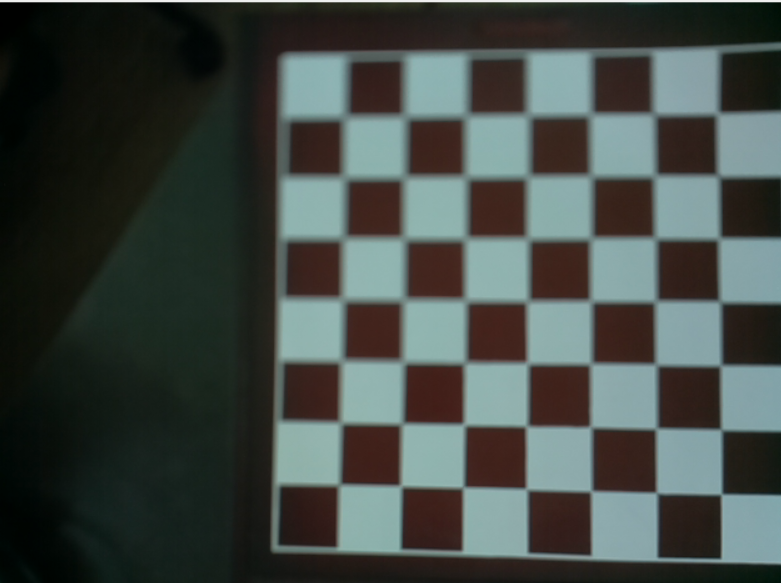
The chess AI algorithm will use the minimax algorithm, which works to minimize the worst case. This algorithm uses a tree to look at every possible scenario within a certain number of rounds which depends on the set depth of the tree. In this case, this project had a set depth of 80. Each node on the tree is then assigned a value based on what pieces could potentially get lost during that round. This is done by assigning each chess piece a weight. Then, the worst case is calculated by adding up all of the weights of the pieces that could be lost. Finally, the minimax algorithm decides what move will cause the least amount of casualties. This causes the AI to play a more defensive role, however, it also makes it tougher to capture the opponent’s king. The aforementioned weights are listed below along with any reason for the set weight:

1. Pawn = 10
   1. Not the most important piece in the game.
2. Knight = 30
   1. Slightly more important than the pawn.
3. Bishop = 30
   1. Slightly more important than the pawn.
4. Rook = 50
   1. I gave this a higher score because I personally use rooks to checkmate my opponents whenever I play.
5. Queen = 650
   1. The queen is definitely the most important piece in the game, however, if she could be sacrificed for a good play, then it is worth losing her.
6. King = 900
   1. The king cannot be lost in any case. Losing the king means losing the game, which is not what the AI is supposed to do. Therefore, it should avoid situations in which the king is at most risk.

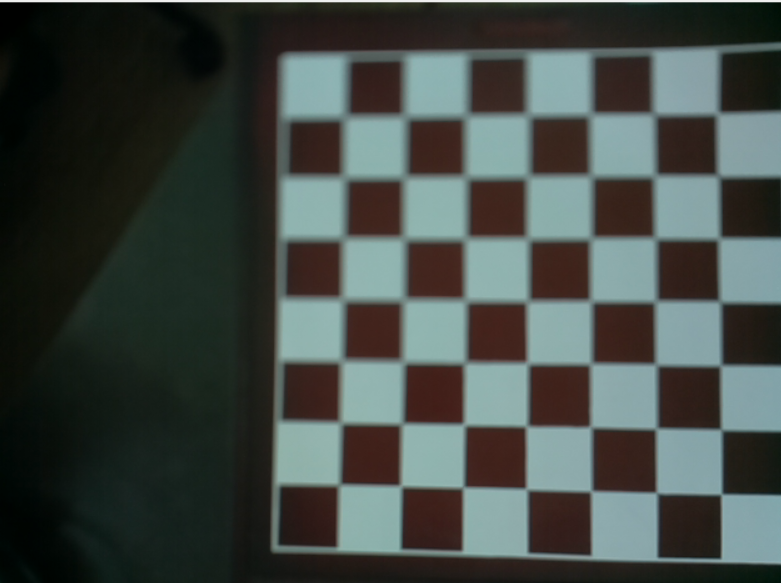
Once an ideal move is found, the system lights up two LEDs based on the position of the piece on the chess location matrix. The move-piece LED will be represented by a red light and the new position will be represented by a green light. If the piece is moved to the incorrect position, the incorrect position will flash a red light and the correct position will flash a green light until the piece is moved.

### 3.2.2 Physical Board Mapping

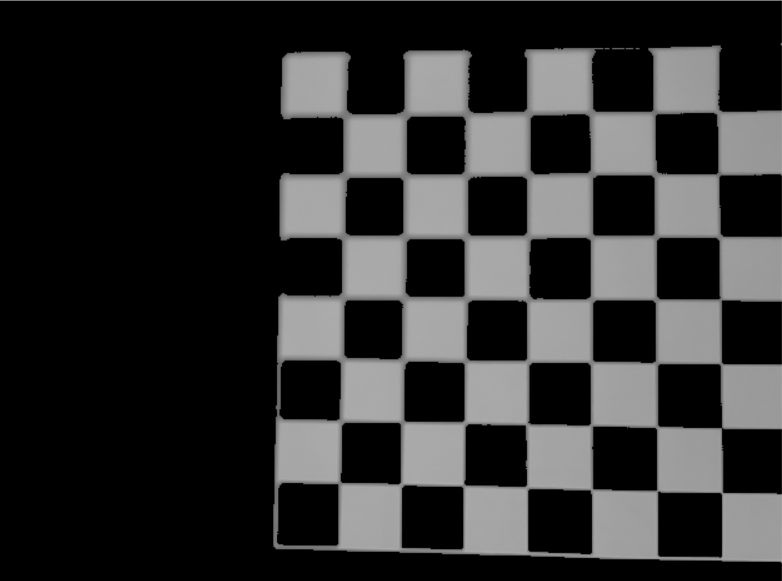
The physical board is mapped using the OpenCV library. First, a picture of the board is taken using the Pi Camera. The image is then flipped so that traversing the image is much quicker during analysis. Next, the image is converted into a binary image -- black and white -- to remove noise from awkward lighting in a room. Since the chess board is a repeating pattern of brown and white squares, the board is much more differentiable as a binary image.



Initial image of the chess board

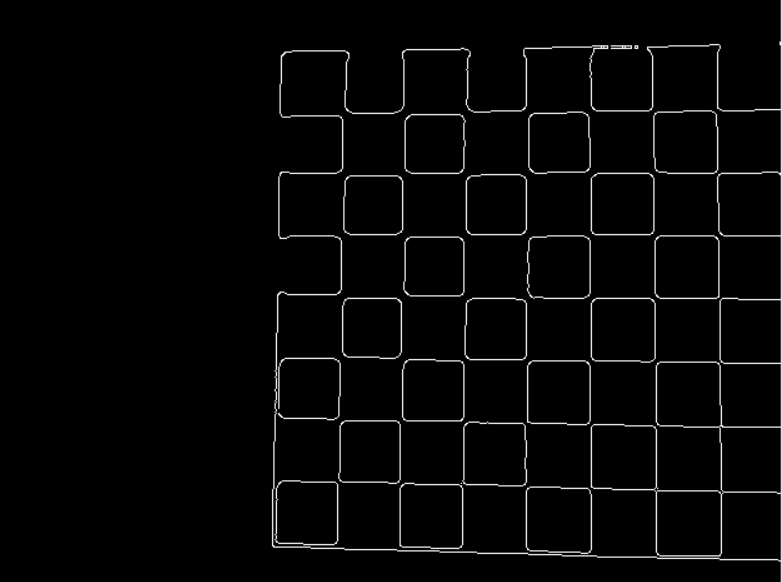


Flipped image of chess board

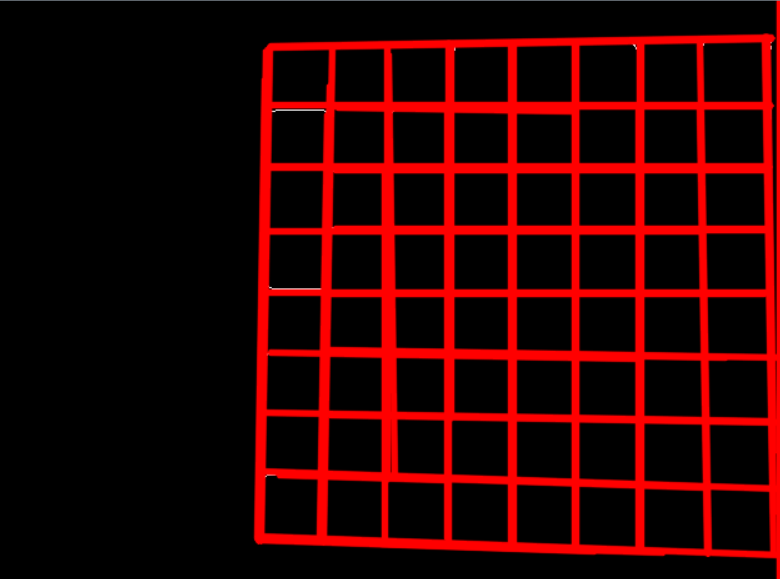


Binary image of the chess board

Next, the binary image is taken and its edges are traced using Canny Edge Detection. Canny edge detection creates a black image with a white outline to indicate detected edges. Finally, Hough Line Transform is performed on the image which detects potential points or lines and draws a solid red line over them. I did this because I could not optimize the Canny Edge Detection algorithm to detect every single edge on the board because the Pi Camera’s low resolution was not able to detect the white outline that separated the outer grids on the chess board from the outer edge of the chess board. This caused multiple gaps to show up on the Canny image. Applying Hough Line Transform filled in these gaps to create a red and black outline of the chess board.



Canny image of the chess board



Hough Line Transform of the Canny image

Once the Hough Line Transform image is obtained, it is traversed to find the coordinates of each line. Starting from the top-left pixel, I make my way across the image until I hit a red pixel. Once that happens, I keep track of the red pixels until I hit a black pixel. When I hit a black pixel, I travel 25 more pixels across. If no other red pixel is detected, I store the position of the last red pixel. That is how I get the vertical lines' position. For the horizontal line, after I hit a red pixel, I keep track of the number of red pixels that I am hitting in a row. If that number goes over 100, it is a horizontal line and I store its position. Repeatedly doing this process gets the position of every single line. Now I know the bounds of each individual grid on a chessboard. In the instance where I ran the demo, I obtained the following coordinates:

X: [11, 60, 115, 170, 224, 277, 331, 386, 440]

Y: [31, 86, 139, 193, 240, 305, 363, 431, 477]

Assuming that the chessboard does not get moved around, I am able to instantly know where to look on an actual image to find a specific grid position. For instance, if I wanted to check the grid position (0, 0), I could examine the positions 11 < x < 60 and 31 < y < 86. In more general terms, to check the grid position (a, b), I could examine positions X[a] < x < X[a+1] and Y[b] < y < Y[b+1].

### 3.2.3 Player Movement Tracking

The technique above is how the player’s movement is tracked throughout the game. An image of the board is taken before the player makes a move. Once the player does make a move, they are requested to press a button. Pressing the button takes another picture of the board. Then the two images are traversed at the same time, one grid at a time. Each pixel contained within a grid is compared to the other image’s grid equivalent grid. The percent change of each pixel between the two images is calculated and stored in an array. Once each grid is checked, an algorithm to find the top 2 largest percent changes in an array is run. These 2 grids represent the piece’s initial position and its updated position. Initially, I created an algorithm that found 2 percent changes greater than 30% and used those 2 grids, however, shadows that the pieces cast created noise in the image that was caught and recorded as percent change. This can be seen in the image below

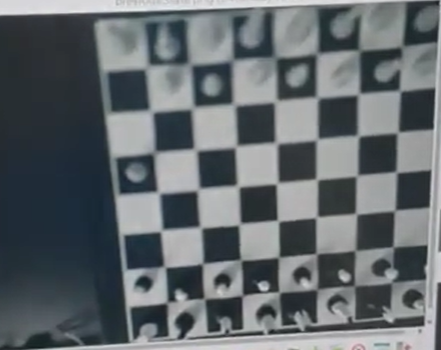
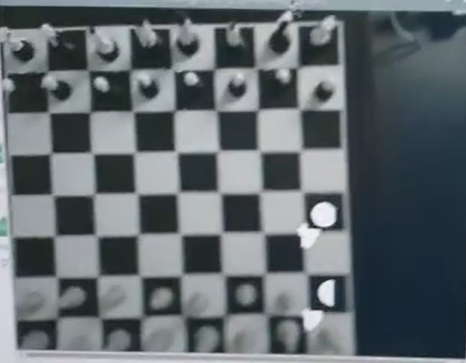


Image of the board after a piece is moved



Detected pixel changes are marked in white. The chess piece’s shadow is also detected.

Finally, the two grid positions are sent to another algorithm that figures out which of the two grid positions contains the piece and which grid position is the new position.

### 3.2.4 Opponent Movement Verification

To make sure that the player is not cheating or misplacing the opponent’s piece, before the LEDs light up to indicate what the AI played, a picture of the board is taken. Once the opponent’s position is marked with the LEDs, the player is instructed to move the piece to its new position and press the button. Pressing the button will temporarily turn off the LEDs then take a picture of the board again. The LEDs are turned off to make sure that the colorful lights that spread onto the board do not make the image taken messier than it needs to be. Once both the ‘before’ and ‘after’ pictures are taken, they are compared. However, instead of traversing through every single grid position, only the new grid position is checked. If there is a noticeable difference detected, the software is updated and the game continues, or else the 3 indicator LEDs flash red to indicate that the piece was either misplaced or not detected. This repeats until the user placed the piece in the correct position.



Only the designated destination grid is checked

# 4. BOM

|  |  |
| --- | --- |
| **Item** | **Price** |
| [WS2812B LED Strip](https://www.amazon.com/gp/product/B01CDTED80/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1) | $10.88 |
| [L-shaped light Connector](https://www.amazon.com/gp/product/B087TB4VCQ/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1) | $9.99 |
| [Pi-Camera](https://www.amazon.com/gp/product/B07SN8HB1R/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1) | $13.99 |
| Raspberry Pi | Already own |
| Camera stand | Already own |
| Chessboard | Already own |
| 5V Battery | Already own |
| Wires and resistors | Already own |
| **Total:** | **$34.86 [Free shipping]** |
| **Arrival Date:** | **03/22/2021 [5-day shipping]** |

**Request for reimbursement:**

No reimbursement is necessary. I will be keeping my own project.

# 5. Tentative work schedule

|  |  |
| --- | --- |
| **Week** | **Plan** |
| Week 1 | * Purchase any additional materials that I may need * Complete hardware wiring * Have a basic understanding of what I need to do on the software side |
| Week 2 | * Implement additional hardware that may have arrived late or needed to be replaced * Begin working on the chess algorithm using a software GUI * Use OpenCV on software GUI to get a better understanding of OpenCV |
| Week 3 | * Polish chess algorithm * Implement physical camera into system using OpenCV and raspberry pi camera |
| Week 4 | * Polish any additional touches |

# 6. Appendix

**Wednesday, March 17th**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [8:14 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1615986849000200)

joined #ai-chess-opponent-project-1member.



[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [8:14 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1615986850000300)

set the channel description: Karim Zaher's project

**Monday, March 29th**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [8:21 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617024096000700)

Chess pieces that I will most likely use on the simulation side of the project: <https://commons.wikimedia.org/wiki/Category:PNG_chess_pieces/Standard_transparent>

**Wednesday, March 31st**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [8:08 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617196135000100)

renamed the channel from “ai-chess-opponent-project” to “ai-chess-opponent-project-1member”



[**liu**](https://app.slack.com/team/U01QZRBL5PY) [8:13 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617196387000300)

was added to #ai-chess-opponent-project-1member by Karim Zaher.



[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [8:57 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617199046006200)

Today, I have decided that I will be using pygame software library for the simulation implementation. By the end of the project, it will be used to mimic the physical board in real-time, however, for now it will be my testing playground to test the chess opponent algorithm before I implement the hardware aspect of the project.

Update: I have created a chess board simulation. I will work on chess piece movement next

Note to self for the next time I work again: Black pieces blend in with brown background. I might have to go with an unconventional chess board color, at least with the simulation

This file can’t be shown because your workspace has passed the free plan’s 5 GB storage limit. [See paid plans](https://21acsce462ter-jxg9676.slack.com/plans)



[**liu**](https://app.slack.com/team/U01QZRBL5PY) [8:58 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617199086007000)

checked - add your portrait

:white_check_mark:**1**

**Sunday, April 4th**

****

[**liu**](https://app.slack.com/team/U01QZRBL5PY) [5:12 PM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617574373000300)

no update

**Monday, April 5th**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [9:19 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617632365015700)

I am sorry for not updating my progress. I thought that I was only supposed to update the slack after I complete a task, not as I complete it. I hope you are pleased with the undocumented progress that I have made though :slightly_smiling_face:

Since my previous update, I have been working on implementing the movement of the pieces by hand. Initially, I just used a chess library that checked and implemented the moves for me, however, it felt like I was cheating. Therefore, I decided to implement it myself. The process was not too difficult, but it was very tedious. The hardest part was testing out every test case I possibly could, however, I used some inspiration from other online chess games and implemented a path-guiding function just to make the testing and debugging process much easier.

Assumption: Users are only allowed to play with the white pieces and all white pieces are correctly placed on the board.

Pawn: Hardcoded it so that the pawn can only move up to 2 steps right at the start, then 1 step after that. If there is a piece in its way, no matter the color, it cannot move forward. If there is a black piece in its diagonals, it can take the piece's position.

Rook: For the rook's algorithm, I made it check to see whether there were any pieces blocking its x path or its y path. If there was a black piece in its way, it could move to the black piece's position. If it was a white piece instead, it could only move to the position right before the white piece's position.

Bishop: The bishop functioned similarly to the rook algorithm, however, it took diagonal movements into account.

Queen: Combined both the bishop and rook algorithms to create the queen. This saved me a ton of time with having to reimplement something that I have already implemented

King: Can take 1 step in any direction unless there is either a white piece in its way or if it is in the path of an opposing team's movements.

Knight: Knights move by taking 1 step in one direction and 2 in another. The direction does not matter as long as it is not off of the board. I also had to make sure that there is no white piece in the spots that it can move in.

Design update: I modified it so that the board UI is no longer brown, but rather a light shade of gray. My physical chess board will still brown because I prefer a classical chess board :smile:

[9:22](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617632529016700)

[Currently working on sending my recorded demo on here, but seems as though the file might be too big. I will likely include a google drive link to the video if I am unable to directly send it on here]



[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [9:32 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617633146017100)

<https://drive.google.com/file/d/1ckcljCaYKDgarXg1WVZc8mZSSV4SE7bT/view?usp=sharing>

[9:32](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617633166017700)

Please let me know if you are unable to view the link

**Friday, April 9th**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [8:55 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1617976509010700)

Currently working on completing the minimax algorithm to make the chess AI smarter. The algorithm checks every possible combination of the next D possible movements (D=depth of the tree that is created in the process. Initially, I had the depth at 5, but I have been playing around with the value to find the most optimized value that makes the AI an extremely tough opponent without making the system very slow. I will test 50 next). It makes its decision based on the lowest weight that it finds using Dijkstra's algorithm. The weight is initialized based on how many pieces are in danger of being long. Each piece has an assigned weight to it.

I will likely play around with these values during the optimization phase of my project, but for now, I have decided to assign the following weights to each piece:

Pawn = 10 (not the most important piece in the game)

Knight = 30 (Slightly more important than the pawn)

Bishop = 30 (Slightly more important than the pawn)

Rook = 50 (I gave this a higher score because I personally use rooks to checkmate my opponents whenever I play. I will likely experiment with this value during the optimization phase)

Queen = 90 (The queen is definitely the most important piece in the game, however, if she could be sacrificed for a good play, then it is worth losing her)

King = >1000 (The king cannot be lost in any case. Losing the king means losing the game, which is not what the AI is supposed to do. Therefore, it should avoid situations in which the king is at most risk. I will also play with this value during the optimization phase, however, the weight will likely be at least 10x the value of the second highest weight.)

There are instances where the game will not allow me to move my pieces to a place that I am allowed to move them. Debugging this issue is going to be my next focus as I also begin to move onto the hardware phase.

**Monday, April 12th**

****

[**liu**](https://app.slack.com/team/U01QZRBL5PY) [6:37 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618227420002000)

You are essentially using an existing library as your project. Do you have any problem solving element in this project? Regarding update of slack, it has been mentioned many times about technical updates. There is no room for misunderstanding.

[6:37](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618227477002800)

CKP 1: SW simulation, no HW prototype -

:raised_hands:1



[**liu**](https://app.slack.com/team/U01QZRBL5PY) [8:30 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618234203003200)

HW depends on SW, so HW will start this week

:white_check_mark:**1**

**Tuesday, April 13th**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [9:29 PM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618367384005000)

Today I began learning openCV so that I am able to use computer vision to keep track of my chessboard with my Raspberry Pi camera. I also began trying to set up my LED light strip that I will place along the chessboard. I started by cutting out 3 LEDs from the light strip to test everything out before I commit to cutting 16 LED strips. As seen from the demo video below, the test was successful. Tomorrow, on Wednesday, I will start working on connecting my LED light strip to the software simulation that I have been working on. I will make it so that whenever the AI opponent makes a move, 4 LEDs light up. 2 will light up in red to indicate what chess piece will need to be moved, and 2 others will light up in green to indicate the new position of that chess piece.

Demo [please let me know if the link does not work]:

<https://drive.google.com/file/d/1k0kOFkiWi1ebimEDVj0FeSKn7eIvVTI6/view?usp=sharing>

**Tuesday, April 20th**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [10:04 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618931086011800)

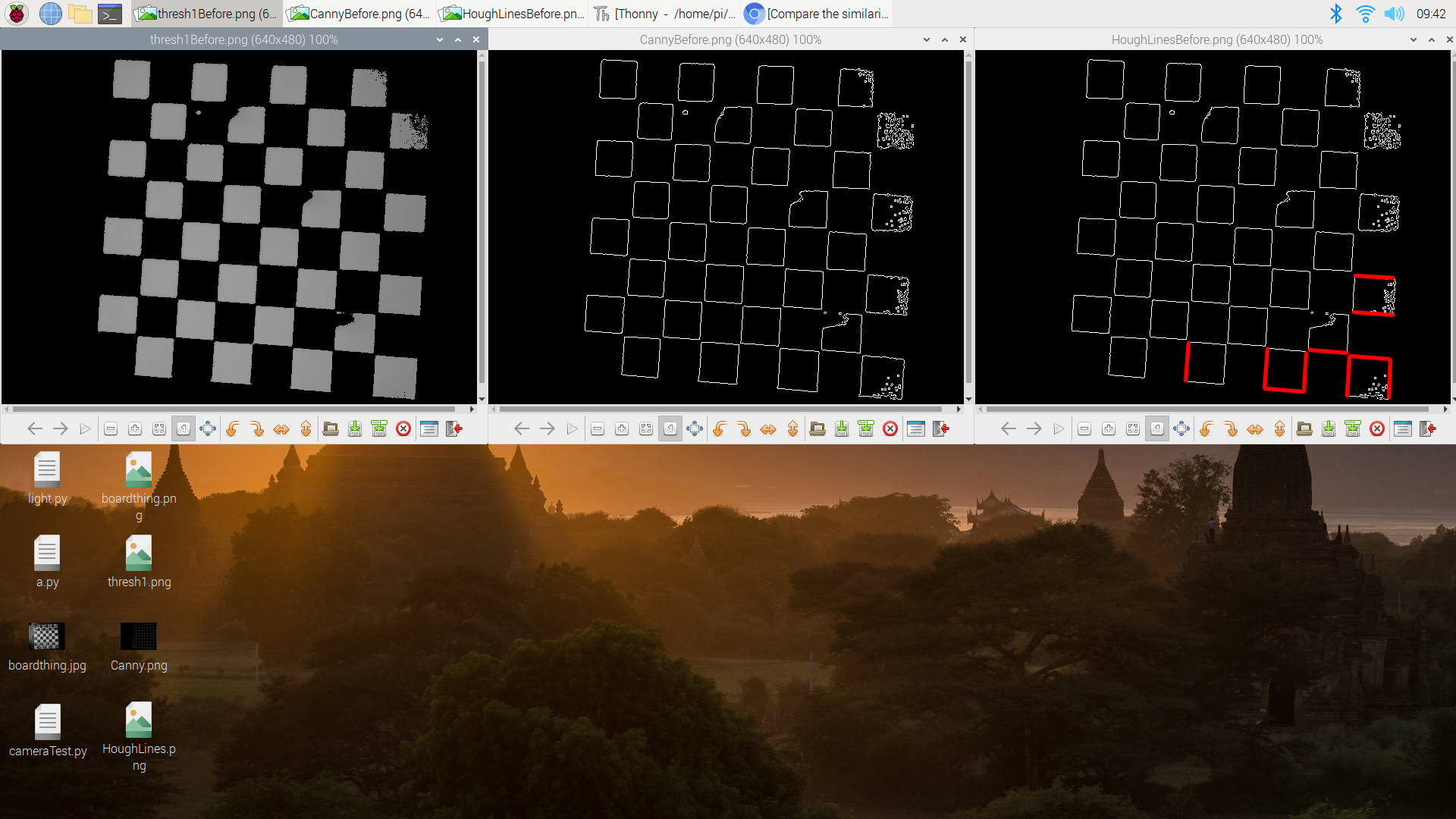
After countless days of learning the basics of OpenCV, I found an explanation on drawing bounding boxes based off of an image. Upon this discovery, I set up my raspberry pi camera over the chess board and began to work. This is the end result:

When people use my product, the first step will be to clear the board. Once the board is cleared, they will indicate that it is cleared by pressing a button. Once the button is pressed, the raspberry pi camera takes an image of the board. The image is then converted into a somewhat binary photo where pixels with a value less than 105 are turned into zeros, and values above that value remain the same. Then, I use an edge detection function called Canny to create a new image with the edges sketched out. The problem with that, however, was that not every single edge could be detected. I tried moving the board around, adjusting the camera, and even played around with the lighting, but nothing worked to fix the problem. After doing some more research, i discovered the Hough Line Transform, which detects edges by connecting points of similar colors based on a set of parameters. I still kept running into some errors, however, after hours and hours of optimization testing, I was finally able to create something that I can use. Using my optimized result, I will now be able to map out my physical board.

I am one step closer to finishing the computer vision aspect of my project. Once I am done, I will be able to add the smaller details such as the LED strips (which is already 85% done. I just have to stick it to my chessboard), the LCD display(this shouldn't take that long either. The hardest part will probably be trying to figure out where to place it so that the user is able to read it easily), and any additional features that I wish to add.

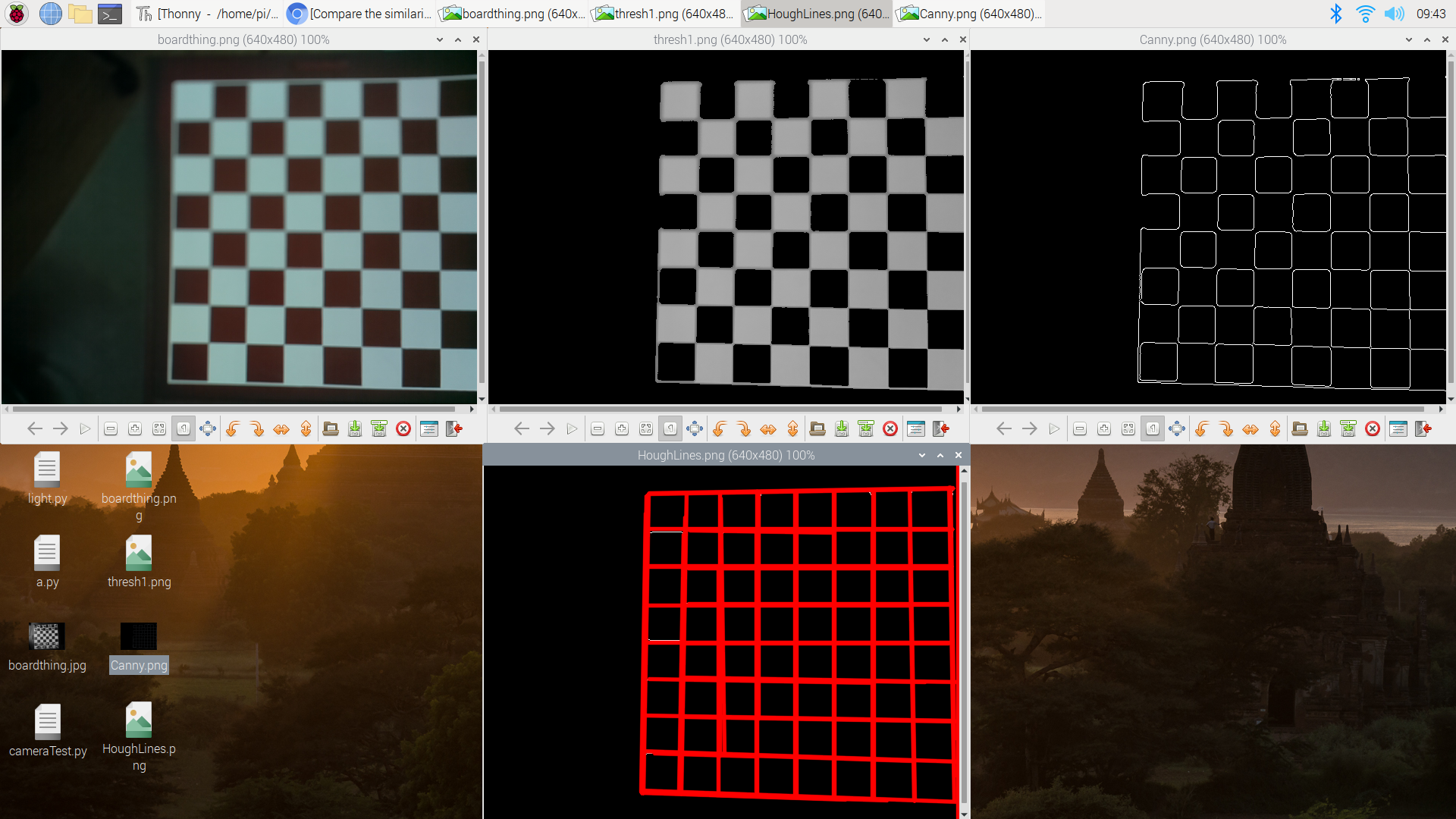
[10:04](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618931099012000)

Before optimization



[10:05](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618931117012400)

After optimization:



[10:07](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1618931253014400)

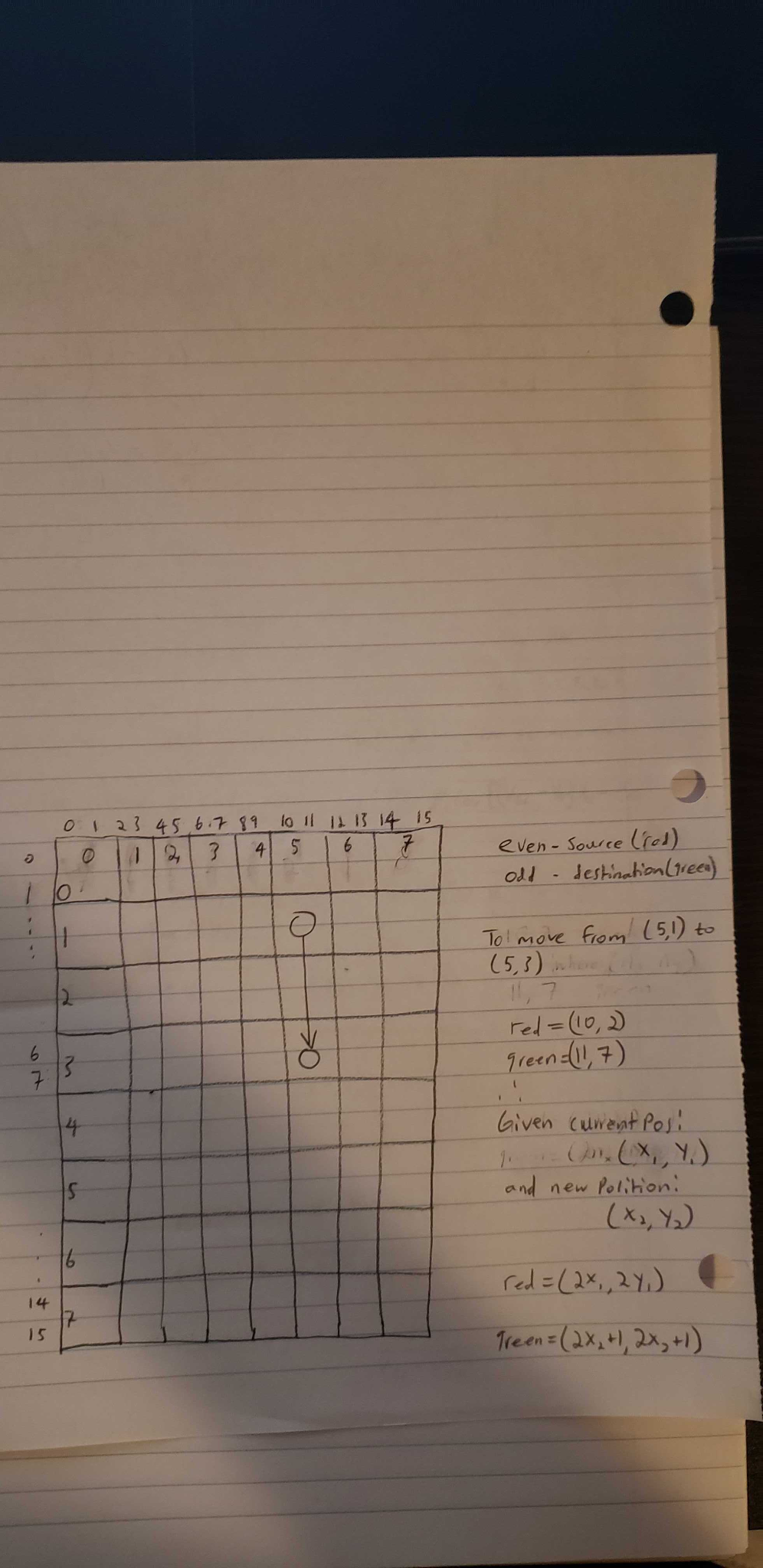
I apologize for not completing the hardware aspects first like I said I would. I was very stressed about the computer vision aspect since it is a very large part of my project, so I wanted to complete it first. I have never used OpenCV prior to this project, so I am very proud of myself. I am enjoying this project so thank you for giving us a final project where we are able to teach ourselves new things :slightly_smiling_face:

**Thursday, April 22nd**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [11:42 PM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619152963000200)

Today, I mapped the LED lights to the board. When the AI opponent makes a move, the move is reflected on the physical board using the LED lights. The LED lights will let the user know where to move the opponent piece on the board. Red LEDs will indicate the piece's current position and green LEDs will indicate the new position that the piece needs to be moved to. Once the user has moved the piece to the new location, they will be asked to press a button to continue. When pressed,, the computer vision algorithm is run to make sure that the opponent piece is moved to the right position. If the position isn't right, the lights will flash twice to let the user know that it is in the wrong position. It will also prevent the player from continuing the game until the user fixes the position



[11:44](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619153044000500)

I have already implemented the LEDs and the button. Now I just have to complete the position-checking algorithm

[11:45](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619153148000600)

After I finish creating this algorithm, all I will have left to do is adapt the algorithm so that it supports the user's plays and uploads the player's move to the system

**Friday, April 23rd**

****

[**liu**](https://app.slack.com/team/U01QZRBL5PY) [7:59 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619182755000300)

checked. good to see your progress

**Tuesday, April 27th**

****

[**liu**](https://app.slack.com/team/U01QZRBL5PY) [7:37 AM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619527065000900)

No further action? I am curious about what the final demo would look like ....



[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [2:38 PM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619552339022900)

I have basically completed my project. I just have to place the component in a place where they are not in the way, and I also need to stick my LED strip onto the chessboard. I will include a full summary of what I have done since the last time below:

[2:39](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619552346023100)

The board coordinates-finding algorithm works as follows: I begin by taking the image of the chessboard's Hough Line Transform (the red and black grid-like image) and parsing through it. Starting from the top-left pixel, I make my way across the image until I hit a red pixel. Once that happens, I keep track of the red pixels until I hit a black pixel. When I hit a black pixel, I travel at least 10 pixels across. If no other red pixel is detected, I store the position of the last red pixel. That is how I get the vertical lines' position. For the horizontal line, after I hit a red pixel, I keep track of the number of red pixels that I am hitting in a row. If that number goes over 85, it is a horizontal line and I store its position. Repeatedly doing this process gets me the position of every single line. Now I know the bounds of each individual grid on a chessboard. Assuming that the chessboard does not get moved around, I am able to instantly know where to look on an actual image to find a specific grid position. This also allows me to know exactly where to look to make sure that a user is placing the opponent's pieces in the right position.

The position-tracking algorithm works in a similar fashion. Prior to the user pressing the button to indicate that they have played their turn, an image of the board is taken. This image is called "previousState". Then, after the player moves their piece and presses the button, another picture is taken called "currentState". Next, using the coordinates collected from the algorithm described above, the position-tracking algorithm checks each individual grid on the board of the previousState and the currentState images, comparing the pixels of each. If there is a difference between a pixel on a grid, the algorithm keeps track of it. Finally, it calculated the percent change of the pixels on the grid and stores it in an array. The algorithm goes through each grid, starting at grid (0,0) and making its way to grid (7,7). Once it has the percent change of each grid, the algorithm grabs the 2 highest percent changes and sends the two grid positions back to the main program. The main program then determines what piece must be moved to what position.

The AI opponent position tracking algorithm works like the last algorithm, however, instead of checking every single grid, it only checks the grid where the opponent's piece must be moved. If it the percent change is over 20%, the game continues.

Here is a progress video: <https://drive.google.com/file/d/1qSpfjU27ORer20BtjN84JBsG4VodTJ6E/view?usp=sharing>

**Yesterday**

****

[**Karim Zaher**](https://app.slack.com/team/U01RQSR338U) [4:26 PM](https://21acsce462ter-jxg9676.slack.com/archives/C01S8HQN2CQ/p1619645169001500)

The link below is for my final demo. I tried to keep it as short as possible. I will likely upload everything to the drive tomorrow:

<https://drive.google.com/file/d/1qpQnD1VEGjHDVN--GiVEoKMnJMoGdRKj/view?usp=sharing>