CHESS DOCUMENTATION

BY SAMUEL JOHNSTON

June 2021

Table of Contents

Note: You can click on the text above to go to that section.

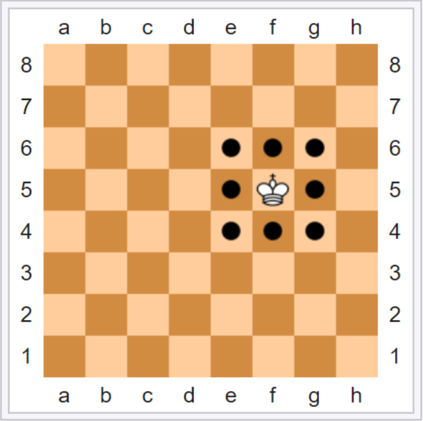
Stage 1 - Define & Understand

Overall Concept

My concept for this project is to utilise python and the tkinter module to create a window based two-player chess game. My goal for this project is to produce a fully functioning chess board, where each player takes turns to move a piece. The program should only show and carry out valid moves for each player, and should stop if a player reaches a checkmate or draw. Since the program is a two-player game, there won't be a need to produce a computer chess engine.

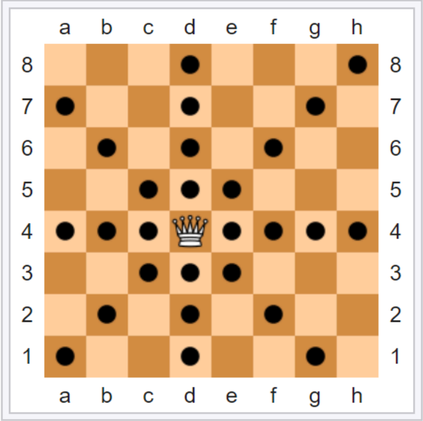
How to Play Chess

Chess is a two player game, one player with the white pieces and the other with the black pieces. Each player then takes turns moving one of their pieces (the person with the white pieces always has the first move). A move consists of moving a single piece to either an empty square or to take an enemy piece. There are 6 types of pieces, the king, queen, bishop, knight, rook and pawn, which all have different move sets. They are as the following:



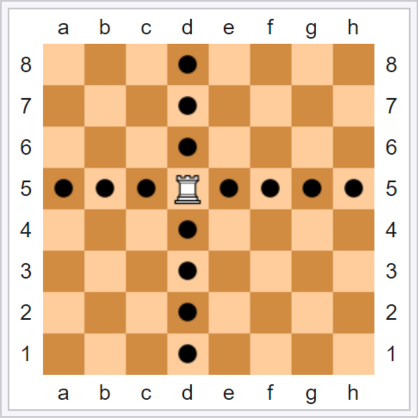
The King:

Kings move one square in any direction, as long as that square is not attacked by an enemy piece. Additionally, kings are able to make a special move, called castling.



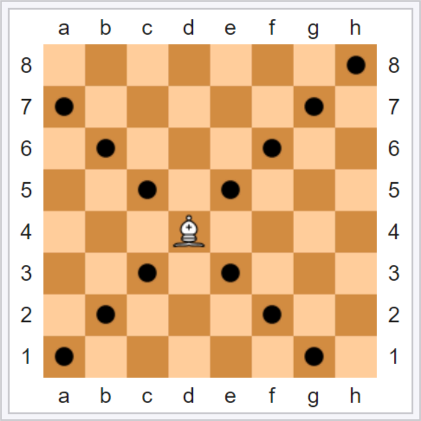
The Queen:

Queens move diagonally, horizontally, or vertically any number of squares. They are unable to jump over pieces.



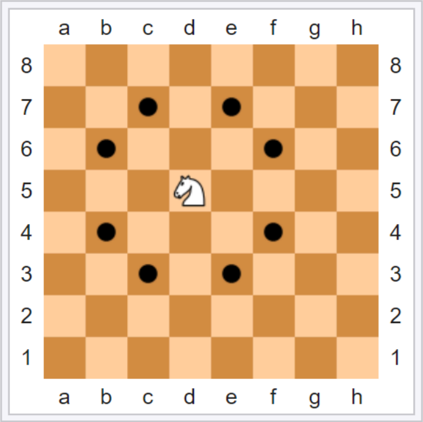
The Rook (Castle):

Rooks move horizontally or vertically any number of squares. They are unable to jump over pieces. Rooks move when the king castles.



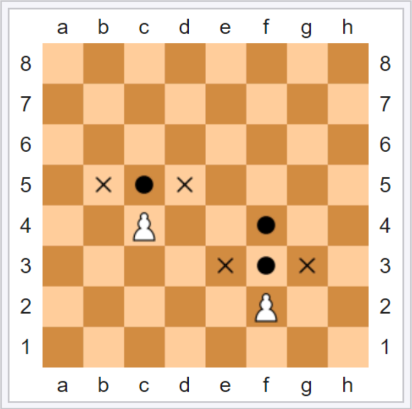
The Bishop:

Bishops move diagonally any number of squares. They are unable to jump over pieces.



The Knight:

Knights move in an ‘L’ shape’: two squares in a horizontal or vertical direction, then move one square horizontally or vertically. They are the only piece able to jump over other pieces.

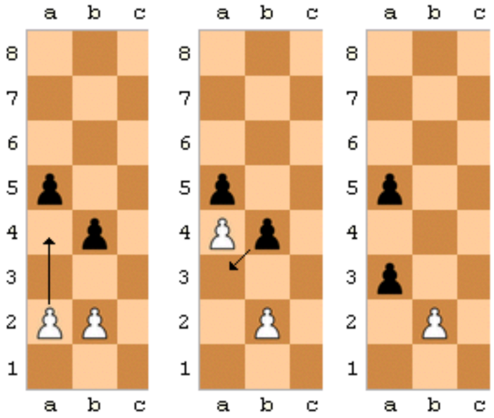


The Pawn:

Pawns move vertically forward one square, with the option to move two squares if they have not yet moved. Pawns are the only piece to capture different to how they move. Pawns capture one square diagonally in a forward direction. Pawns are unable to move backward on captures or moves. Upon reaching the other side of the board a pawn promotes into any other piece, except for a king. Additionally, pawns can make a special move called En Passant.

All Images from: https://en.wikipedia.org/wiki/Chess

In addition to the standard moves for each piece, there are three ‘special moves’:



En Passant:

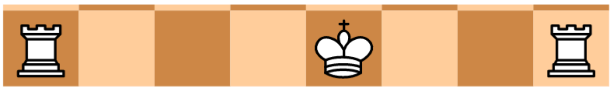
En Passant is a special pawn capture that can only occur immediately after a pawn makes a move of two squares from its starting square. A pawn may capture the just-moved pawn "as it passes" through the first square. The result is the same as if the pawn had advanced only one square and a pawn had captured it normally.

The requirements for an En Passant to be legal:

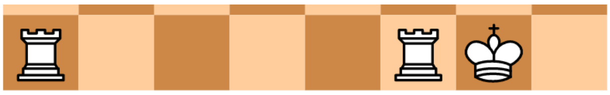
1. The capturing pawn must have advanced exactly three ranks to perform this move, that is the pawn has moved three squares forward.
2. The captured pawn must have moved two squares in one move, landing right next to the capturing pawn.
3. The En Passant capture must be performed on the turn immediately after the pawn being captured moves. If the player does not capture En Passant on that turn, they no longer can do it later.

Short Castling:

Before

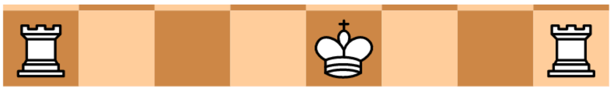


After

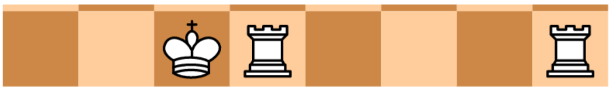


Long Castling:

Before



After



Castling:

Castling consists of moving the king two squares towards a rook on the first (or eighth for black) rank, then moving the rook to the square that the king crossed. Castling can be performed in one of two directions, kingside (abbreviated to ‘short’) or queenside (abbreviated to ‘long’).

The requirements for Castling to be legal:

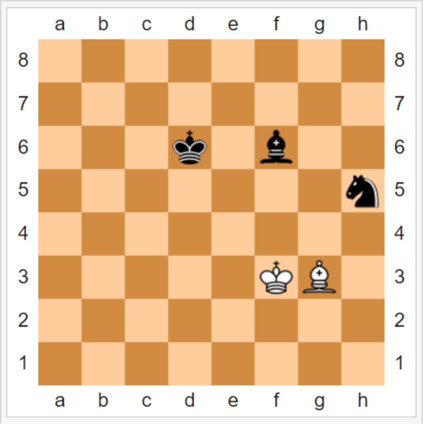
1. The castling must be kingside or queenside.
2. Neither the king nor the chosen rook have been previously moved.
3. There are no pieces between the king and the chosen rook.
4. The king is not currently in check.
5. The king does not pass through a square that is attacked by an enemy piece.
6. The king does not end up in check.

Promotion:

Once a pawn reaches the eighth rank (the end of the board), the pawn can be replaced by the choice of a bishop, knight, rook, or queen.

The Goal of Chess

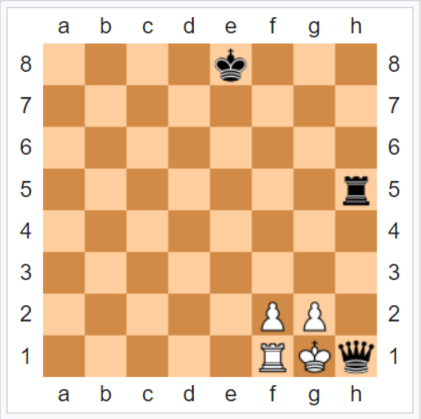
The ultimate goal of chess is to place your opponent's king in checkmate. Although not all chess matches end in checkmate, there are other outcomes such as draws, stalemates, flagging, or resigning.



Check:

A check is a move that results in a player's king to be under immediate threat of capture by an opposing piece. The player whose king is in check must make a legal move that gets their king out of check; if there are no legal moves, the king is checkmated and the player loses (see next section). As seen on the board to the left, black's king is in check by white's bishop. There are three ways to get out of check:

1. Capture the checking piece. This could either be with another piece or even the king, so long as the king doesn't place itself in check after the capture. In addition to this, a piece that is pinned (that is moving the piece would result in a check) against the king can not capture the checking piece.
2. Move the king to a square that is not in check. As mentioned above, the king cannot castle out of check.
3. Block the check with another piece as long as the piece is not pinned (as mentioned in 1.)



Checkmate:

If a player's king is in check and there are no legal moves to get out of check, then it is checkmate. The board to the left depicts a scenario where white's king is in checkmate; black cannot make a move that places the king out of check. When a player's king in checkmate, the game ends and the opponent (the player who isn't checkmated) wins.

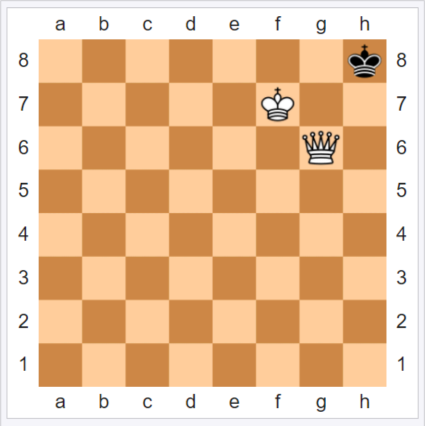
Resigning:

At any time either player may concede (forfeit) the game to the opponent, this is called resigning. This is performed either verbally or by 'tipping' their king over.

Draws:

There are many types of draws. These include:

1. Draw by agreement. A player may offer a draw at any stage of a game; the opponent is not obligated to accept, but if they do so it is called a draw. When both players agree to a draw, the game ends with no winner.
2. Stalemate. Occurs when it is the players turn, their king is not in check, and they have no available valid moves (see next section).
3. Threefold repetition. A player may claim a draw by threefold repetition if the same position has been reached three times. This usually occurs when both players move back and forth one of their pieces for several moves.
4. In addition to these three, there are also the draw by the fifty-move rule, draw by the five-fold repetition rule, and draw by the seventy-five-move rule.



Stalemate:

A stalemate is the result when the current player has no valid moves and is not in check. As mentioned above, a stalemate will result in a draw. The board to the left depicts a scenario where it is black's turn to move; black's king cannot move and is not in check, stalemate.

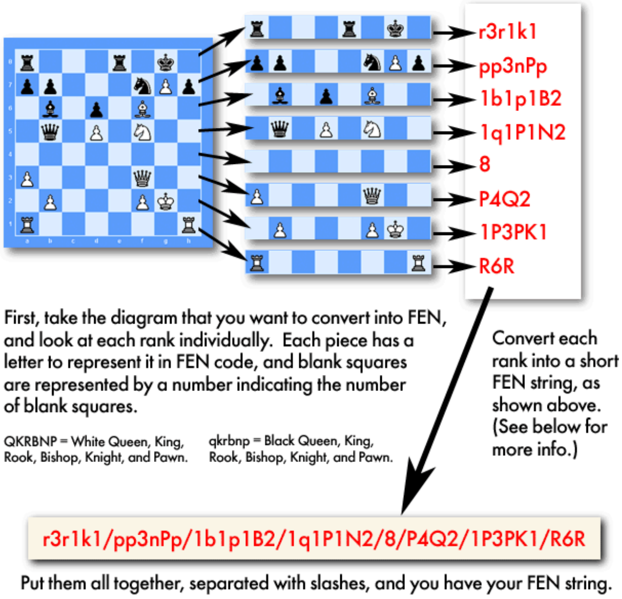
Flagging:

Sometimes chess matches are played under timed conditions using a clock, where both players have a set amount of time (in some cases bonuses may be allowed). If the time remaining reaches zero by one of the players, then the game ends and they lose; this is called flagging.

Forsyth-Edwards Notation (FEN)

The Forsyth-Edwards Notation, or FEN for short, is the standard notation used for describing a particular position in a game of chess. FEN notation allows for a particular game position to be recorded in one line of ASCII characters. A FEN record contains six fields separated by a space. They are as the following:

1. The piece arrangement (with white at the bottom). The pieces for each rank, starting with the top rank moving left to right, are described with a character.



First, take the diagram that you want to convert into FEN, and look at each rank individually. Each piece has a letter to represent it in FEN code, and blank squares are represented by a number indicating the number of blank squares.

QKRBNP = White Queen, King, Rook, Bishop, Knight, and Pawn.

qkrbnp = Black Queen, King, Rook, Bishop, Knight, and Pawn.

Convert each rank into a short FEN string, as shown above. (See below for more info.)

r3r1k1/pp3nPp/1b1p1B2/1q1P1N2/8/P4Q2/1P3PK1/R6R

Put them all together, separated with slashes, and you have your FEN string.

P = Pawn	N = Knight	B = Bishop
R = Rook	Q = Queen	K = King

Uppercase characters are designated as white pieces and lowercase pieces are designated as black pieces. Empty squares are represented by a numerical value that displays how many empty squares there are in a row. A forward slash is used to indicate the next rank, moving from the top rank down to the bottom rank. As seen in the diagram to the left, this process is repeated until all eight ranks are represented by a string.

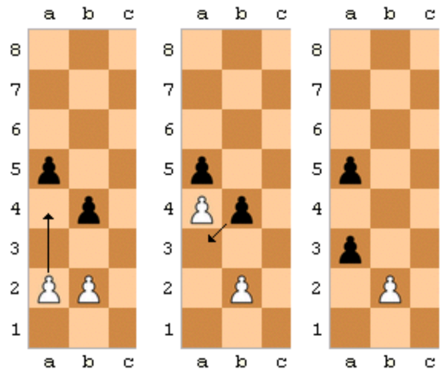
2. Player's move. “w” = white's turn to move, “b” = black's turn to move.

3. Castling availability. As mentioned previously in the section about castling, a player may lose their privileges to castle their king. This is represented with a:

K = White can castle kingside	Q = White can castle queenside
k = Black can castle kingside	q = Black can castle queenside

If they don't have castle privileges, a ‘-’ is used instead.

4. En passant target square. The en passant target square is the location directly behind a pawn that has just moved two squares in the last move. As seen in the diagram to the right, the en passant target square would be a3. If there's no en passant target square, a ‘-’ is used instead.

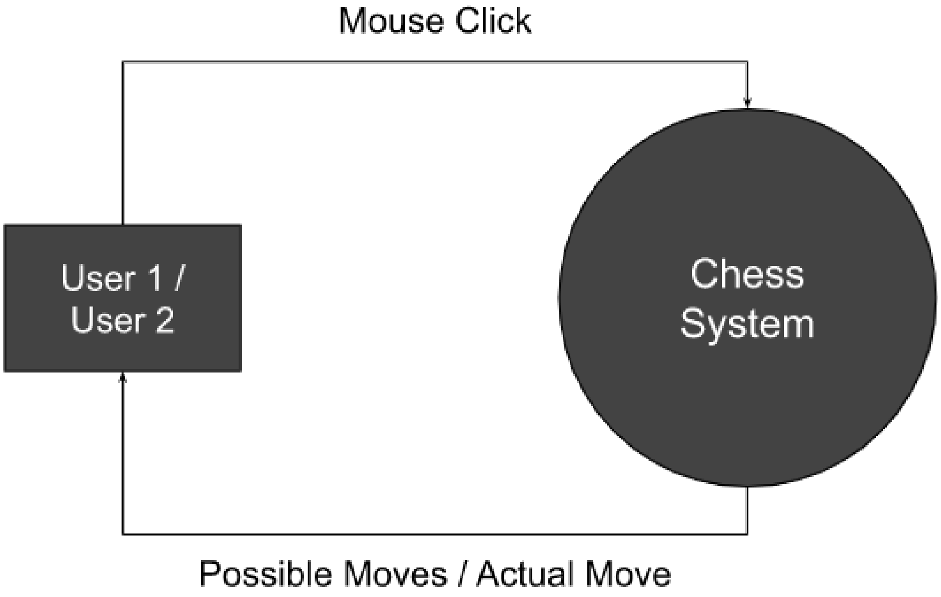


5. The halfmove clock (used for the fifty-move rule)
6. The fullmove number (number of full moves, starting at 1 and increments after black's move)

For example, the initial starting position in FEN notation would be:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

Context Diagram



Sources

Wikipedia 2021, *Chess*, viewed 28 May 2021, <https://en.wikipedia.org/wiki/Chess>.

Chess.com 2021, *How to Play Chess | Rules + 7 Steps to Begin*, viewed 28 May 2021, <https://www.chess.com/learn-how-to-play-chess>.

Wikipedia 2021, *Rules of chess*, viewed 28 May 2021, <https://en.wikipedia.org/wiki/Rules_of_chess>.

iChess.net 2017, *How the Chess Pieces Move: Learn Now and Fast!*, viewed 29 May 2021, <https://www.ichess.net/blog/chess-pieces-moves/>.

Wikipedia 2021, *Forsyth–Edwards Notation*, viewed 29 May 2021, <https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation>.

Chessgames.com 2021, *FEN Made Easy*, viewed 29 May 2021, <https://www.chessgames.com/fenhelp.html>.

Wikipedia 2021, *Check (chess)*, viewed 30 May 2021, <https://en.wikipedia.org/wiki/Check_(chess)#Types_of_checks>.

Wikipedia 2021, *Stalemate*, viewed 30 May 2021, <https://en.wikipedia.org/wiki/Stalemate>.

Wikipedia 2021, *En Passant*, viewed 30 May 2021, <https://en.wikipedia.org/wiki/En_passant>.

Wikipedia 2021, *Castling*, viewed 30 May 2021, <https://en.wikipedia.org/wiki/Castling>.

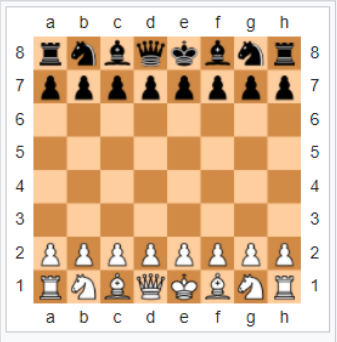
Stage 2 - Plan & Design

Success Criteria

Overall, for the project to be successful the project should met the following criteria:

1. Board & Piece Display

The program should display a whole chess board with both player's pieces (black and white) on it. The board should have the correct dimensions (8 x 8) and the pieces should be easily differentiated (pawns look like pawns, bishops look like bishops, etc). The board and pieces combined should look something like this (depending on the board position) to the right. In addition to this, the program should allow the user to input a FEN string to start the match at a certain position.



2. Piece Moves

The program should allow the user to select and deselect their own pieces via mouse clicks. Only one piece should be selected at any time, and the program should display all legal moves the selected piece has through a dot displaying the move. These moves should also include any legal castling moves or en passant moves. The program should then allow the user to click on the dot (the move) and perform it on the board, or click the piece again to deselect it.

3. Board Evaluation

After each move, the program should correctly evaluate the board and display any checks, checkmates, or draws. This will involve checking the board for pieces attacking a player's king (and taking into consideration pinned pieces) which will feed into the next player's moves. For example, if a player moves a piece to check the opponent's king, the program should display "check" and the opponent should only see the legal moves to stop the check when selecting their pieces. In addition to this, if a player reaches the position of a checkmate or draw, the board should also display this and end the game.

IPO Table

The table below shows a very simple high-level overview of the program's inputs, processes and outputs.

Input	Processes	Output
Clicked Piece (Select)	1. Select & highlight piece 2. Generate legal moves for selected piece	Show generated moves
Clicked Piece (Deselect)	1. Unselect & unhighlight piece	Hide generated moves
Move	1. Move selected piece to clicked location 2. Remove any enemy piece 3. Switch player turn	Updated board

Data Dictionary

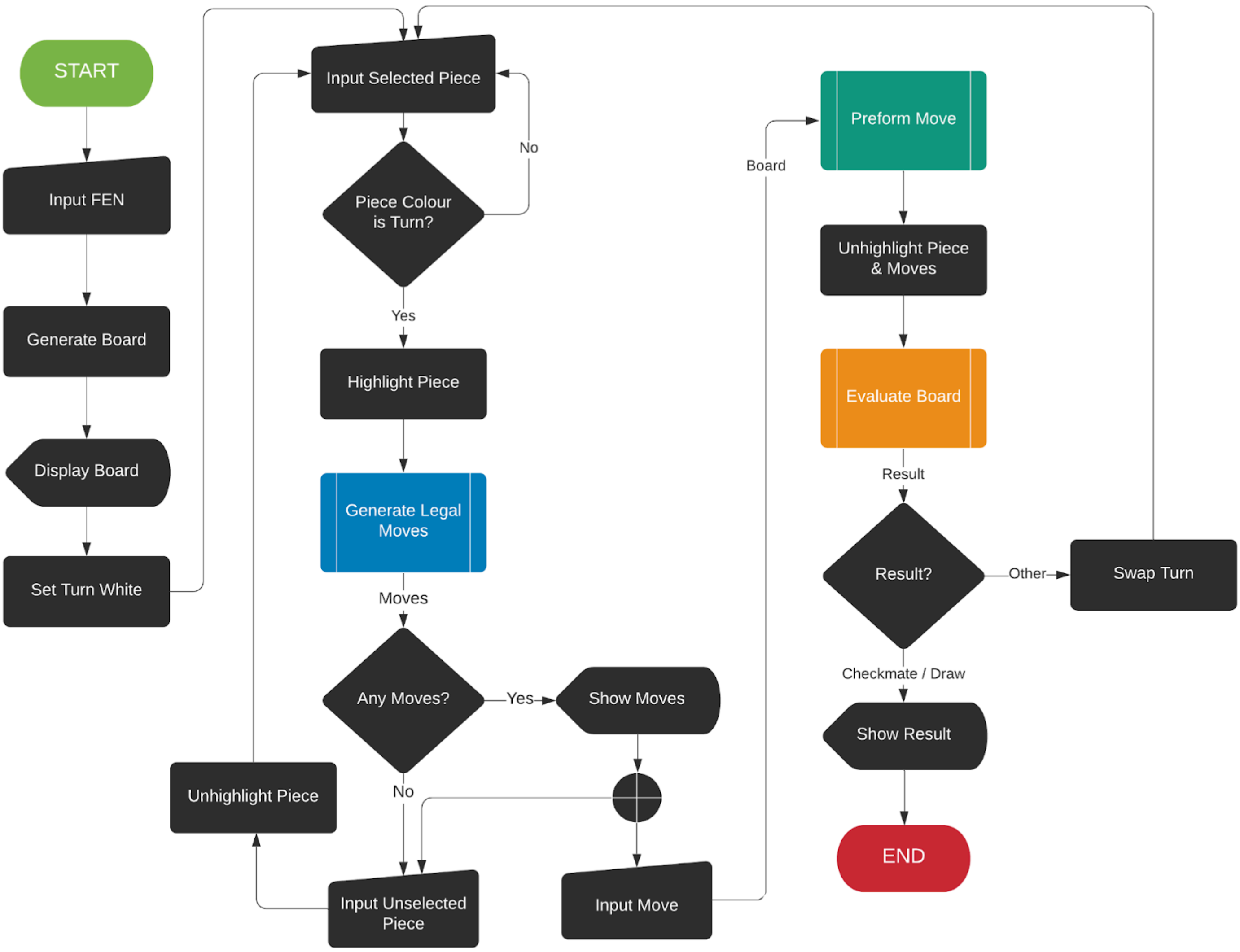
In addition to the variables, I am planning to use class attributes to help code the program. The table below shows the planned data variables going to be used throughout the program.

Item	Type	Size	Description
Fen	String	-	Board starting position in FEN notation.
Board	Array / List	64 (8 lists of 8 elements)	Holds all the variables (such as values, positions, etc) for the pieces on the board.
Squares	Array / List	64 (8 lists of 8 elements)	Holds all the images for displaying the piece on the board.
Current Turn	String	1	Current player's turn, either 'w' for white of 'b' for black.
Piece	Class	Number of pieces	Holds all the variables for each piece as class attributes.
Piece.Value	Class Attribute; String	1	Piece's value, either 'K' for King, 'B' for bishop, etc.
Piece.Colour	Class Attribute; String	1	Piece's colour, either 'w' for white or 'b' for black.
Piece.Rank	Class Attribute; Integer	1	Piece's rank location, 1 int ranging from 0-7.
Piece.File	Class Attribute; Integer	1	Piece's file location, 1 int ranging from 0-7.
Piece.Selected	Class Attribute; Boolean	-	Determines if a piece is selected, either 'True' for being selected or 'False' for not being selected.
Piece.Moves	Class Attribute; Array / List	Number of moves	Holds all the piece's moves, list holding all the moves corresponding rank and file.

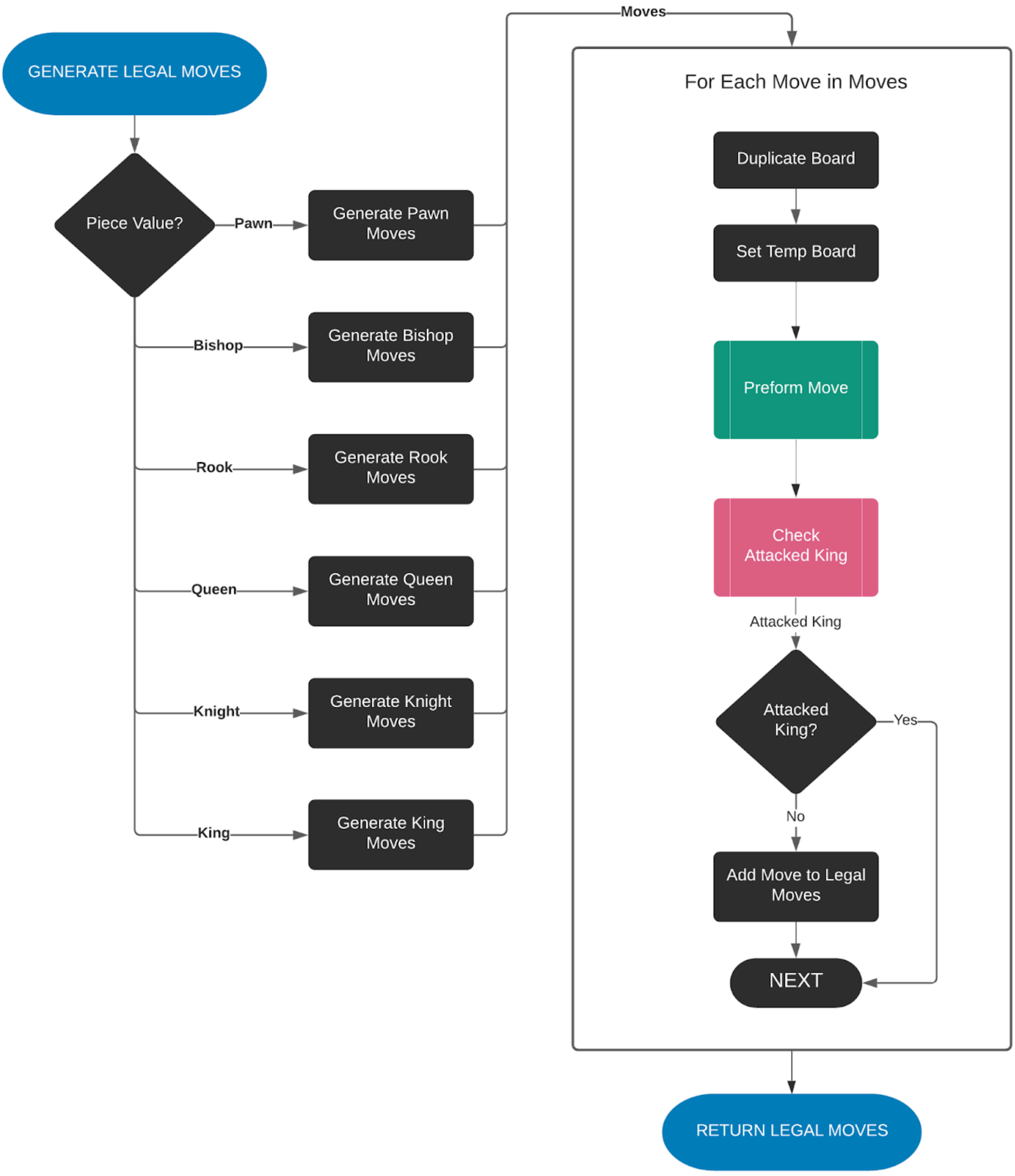
Flowcharts

The first flowchart shows an overarching perspective (level 0) of the planned processes within the program. Additionally, there are 4 subroutines which are shown by the charts subsequently below. Although, these flowcharts may change during stage 3 (implementing) as other processes may be added or altered (ie. special moves such as castling or en passant)

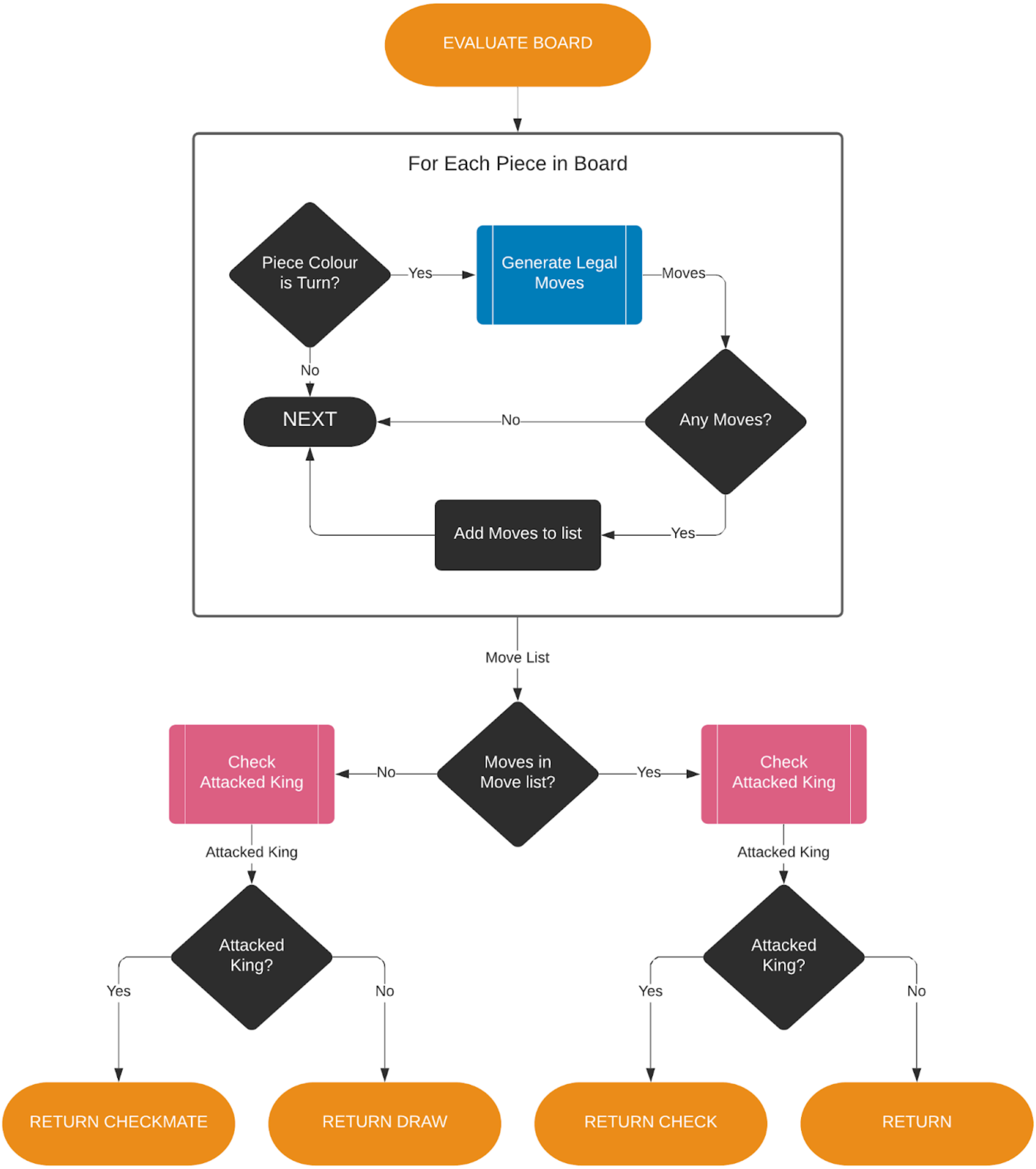
Main Flowchart



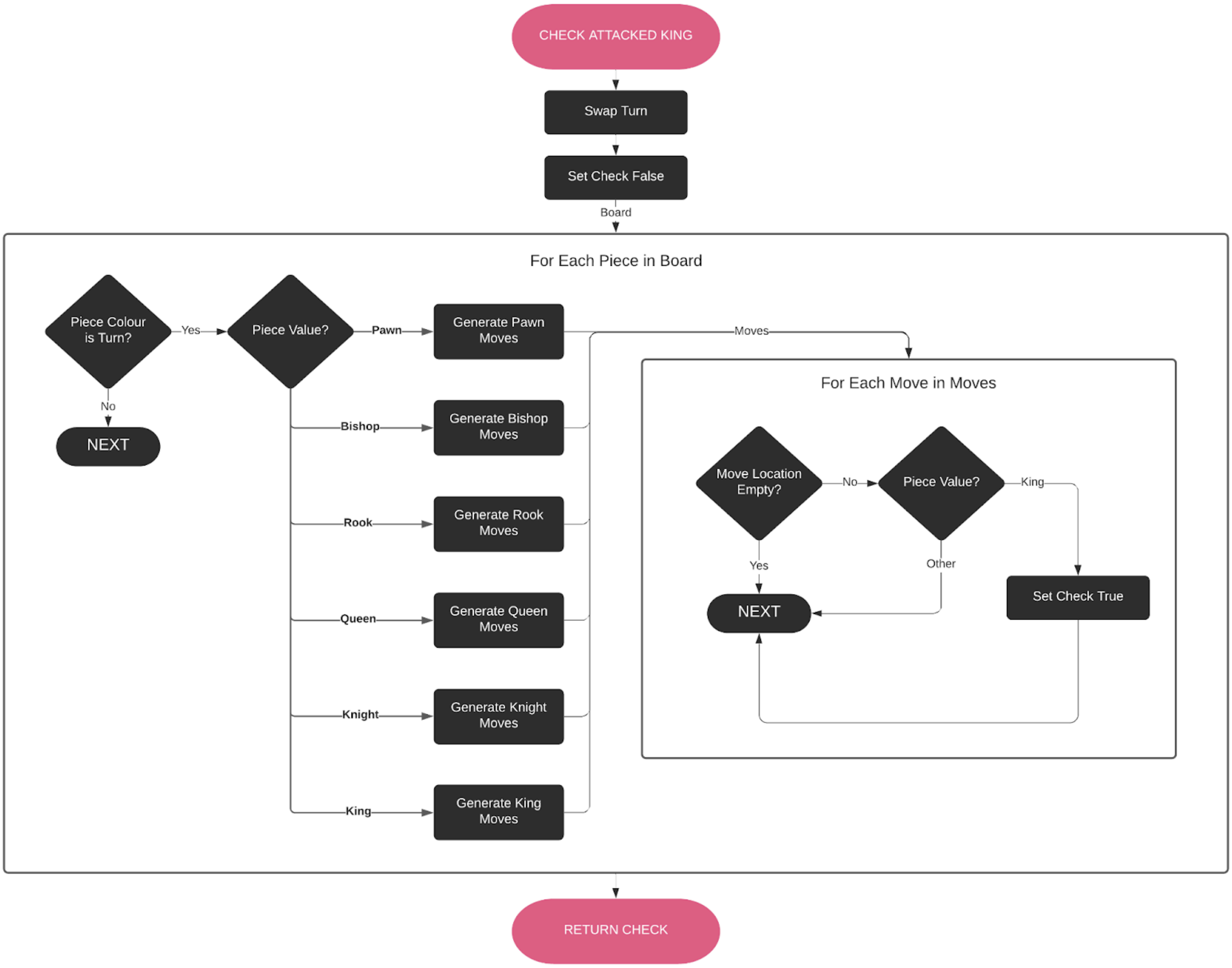
Subroutine Flowchart: Generate Legal Moves



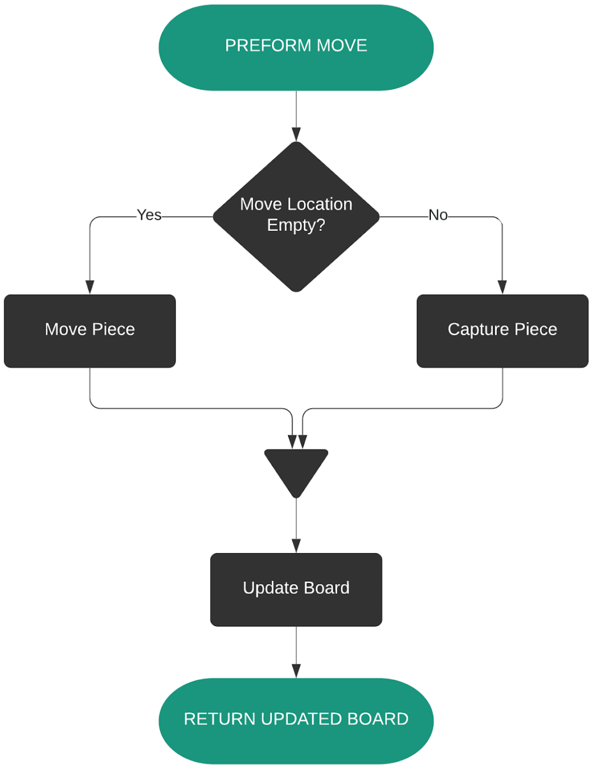
Subroutine Flowchart: Evaluate Board



Subroutine Flowchart: Check Attacked King



Subroutine Flowchart: Preform Move



Stage 3 - Implementing

Program Files

Please see the submitted zip file for the program (there are two, one for mac and another for windows). If for some reason you're unable to access it, I've provided some extra links just in case.

Google Drive Link:
https://drive.google.com/drive/folders/1naTfl9T7bq-iEZyP6l93MzxVzcgkYqwV?usp=sharing

Github Repository Link:
https://github.com/SurferSamuel/2021-Computing-Task-2

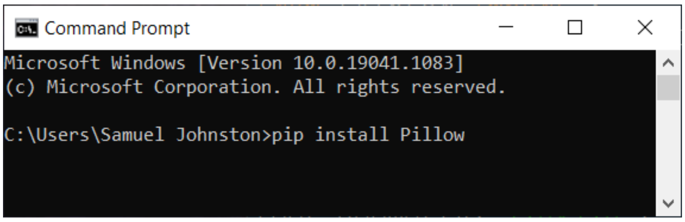
Running the Script

When running the script for the first time, you most likely will encounter the following error:

"ModuleNotFoundError: No module named 'PIL'"

This is because the program uses a non-standard python module, called 'Pillow'. This module allows the program to utilise transparent images and layering which is critical for the UI to operate.

In order to fix this error you need to install the 'Pillow' module. To do so, type "pip install Pillow" into command prompt (windows) or terminal (mac). After the module has been installed, the program should be able to be run without any errors.



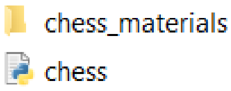
```
Microsoft Windows [Version 10.0.19041.1083]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Samuel Johnston>pip install Pillow
```

In addition to this, after pressing start you may encounter the following error:

"FileNotFoundError: No such file or directory: 'chess_materials/bR.png'"

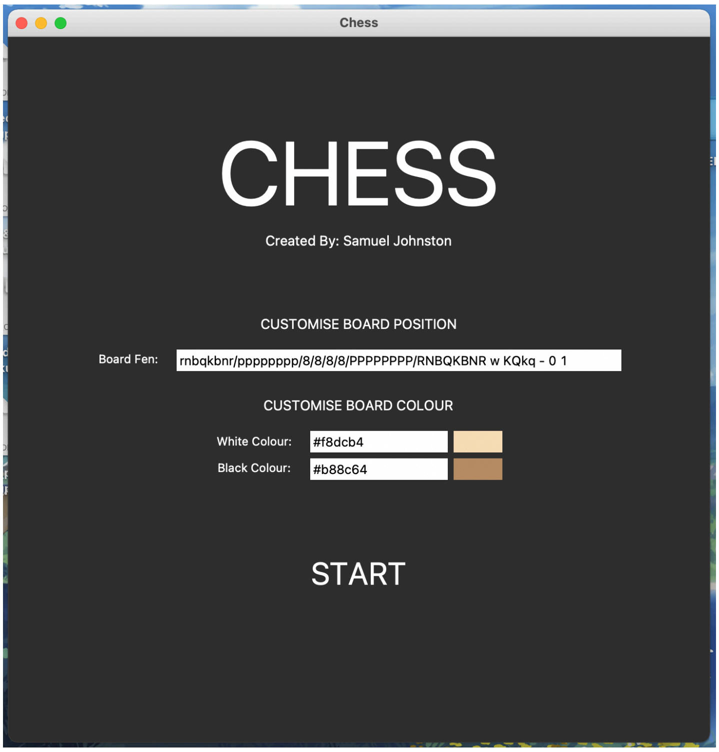
Ensure the 'chess_materials' folder is in the same directory as the program script - otherwise the script will not be able to find the required materials.


chess_materials
chess

Using the Program

When launching the program, users are confronted with an editor screen. There are two areas that can be edited, the board starting position and colour. As seen to the right, by default there are already values inside the text boxes - launching the game without changing these will result in the game starting normally.

The user may input a custom FEN to initiate the board at a different starting position. Handy generators, like this [one](), can create a random position FEN, or you could make your own one up. In saying this, an invalid FEN may prove to not set up the board correctly or even result in an error.



CHESS

Created By: Samuel Johnston

CUSTOMISE BOARD POSITION

Board Fen: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

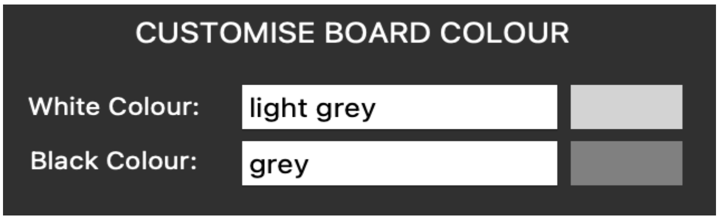
CUSTOMISE BOARD COLOUR

White Colour: #f8dcb4

Black Colour: #b88c64

START

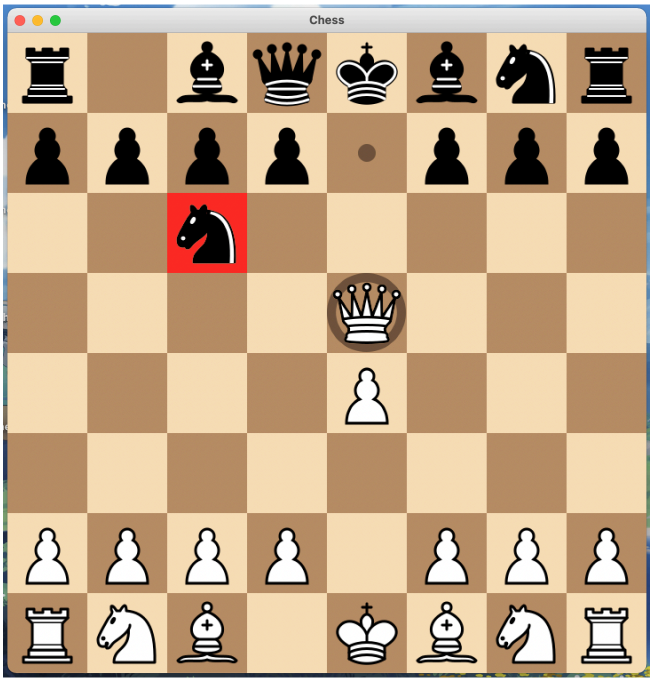
In the second section, the user may change the aesthetic colour of the board. In addition to hex codes, typed colours can also be used to customise the board. The tile to the right updates to show the colour. If no colour or an invalid colour is present, the program will default to the colour white.



CUSTOMISE BOARD COLOUR

White Colour: light grey

Black Colour: grey



When ready the user may click START to begin the match. As per the rules, the player with the white pieces starts first. Each player will then take turns moving each of their pieces.

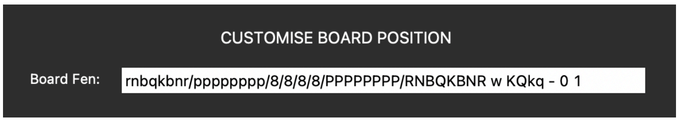
To move a piece, the user can click on the desired piece and the program will show all the legal moves the selected piece has (this includes castling and en passant). They can then click on the square they want to move to (shown by a dot to move or a circle to capture). If the player decides not to move the piece, they can click on the piece again to deselect the piece.

Stage 4 - Testing & Evaluating

As referenced in stage 2, for the project to be deemed successful the program should meet all three aspects of the success criteria. As such, the program has successfully been able to meet all aspects of the success criteria, as demonstrated below:

1. Program Aesthetics and Composition (Board & Piece Display)

As seen by the image in stage 3 (the page before this one) and below, the program successfully meets the criteria for the aesthetics and composition. The program displays the chess board and its pieces distinctly, and allows the player to initialise the board at a custom position through inputting a FEN string (as shown to the right). In saying this, an invalid FEN may not set up the board correctly, or in some cases even result in an error.

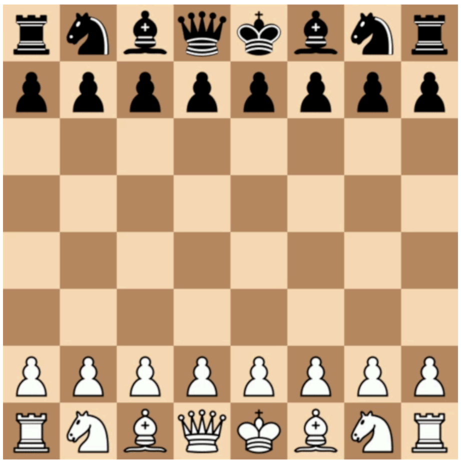


CUSTOMISE BOARD POSITION

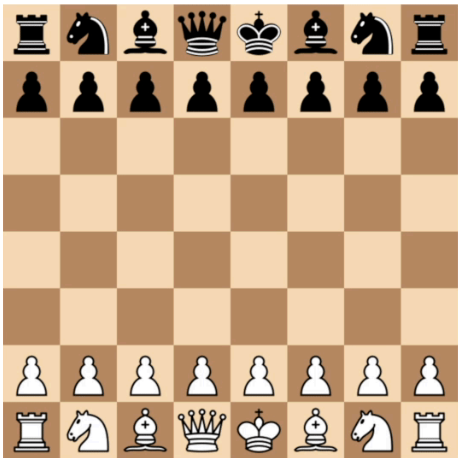
Board Fen: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

2. Piece Movement (Moves)

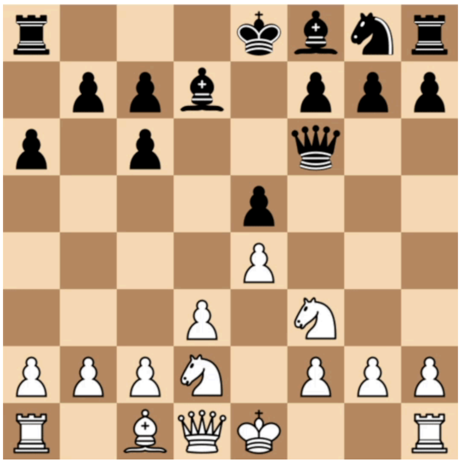
As seen by the animated gifs below, the program allows the player to select and deselect their pieces, showing the legal moves for each piece represented by a small dot.

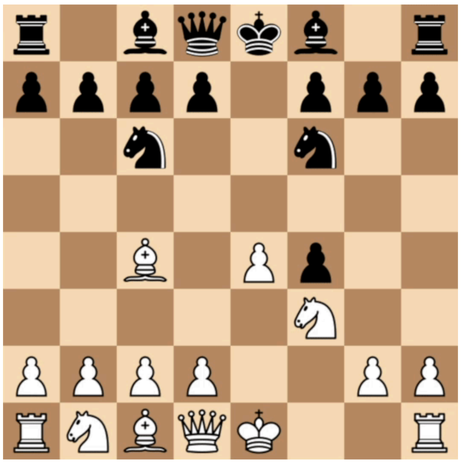


The player can then click on the dot to move the selected piece. This could either be moving a piece to an empty square or capturing an enemy piece.

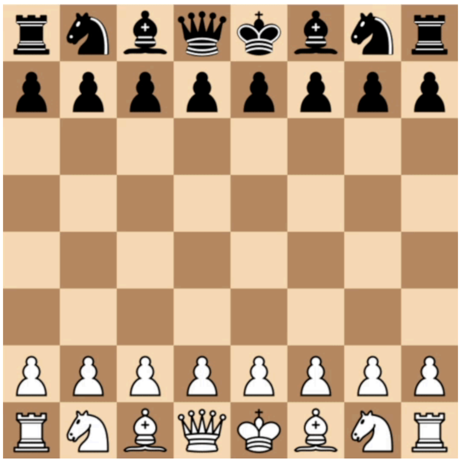


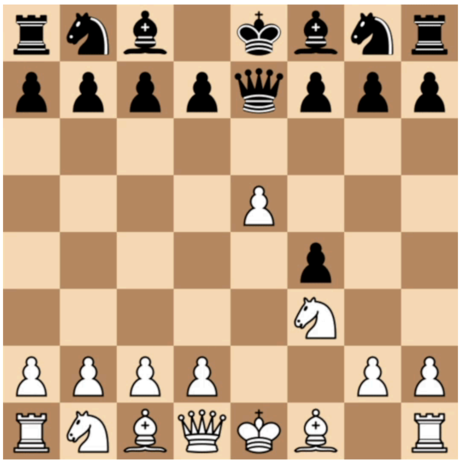
In addition to moving and capturing pieces, the program successfully allows the player to castle their king to safety (provided castling is a legal move). If the player moves their rook they will lose castling privileges for that specific side, and if the player moves their king they will lose all castling privileges. Additionally, if the king is in check, goes through check, or goes into check, the program will not allow them to castle their king.



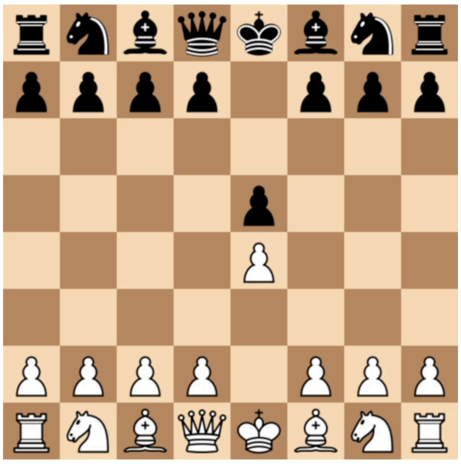


The program successfully allows the player to perform an *en passant* move (provided the move is legal). If the player does not perform the en passant move straight after the opponent's pawn double move, the player loses the privileges to en passant capture at a later time.



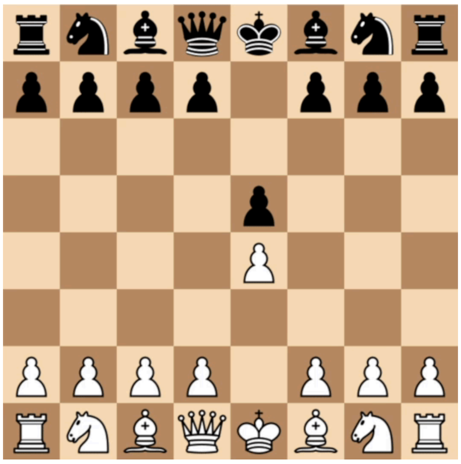


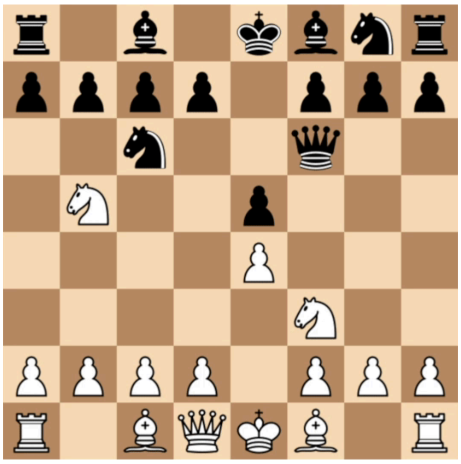
Also, the program successfully accounts for pinned pieces to the player’s king. If a player clicks on a pinned piece, no moves will be shown as there are no legal moves for that specific piece.



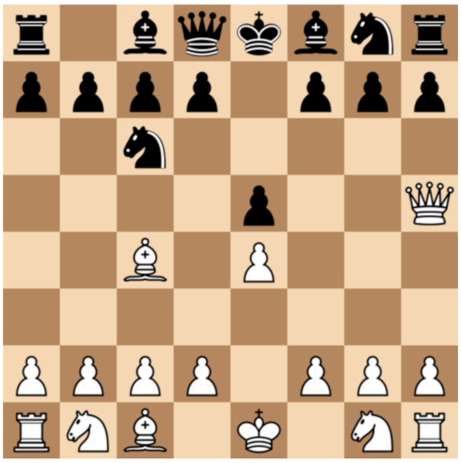
3. Board Evaluation (Checks, Checkmates and Draws)

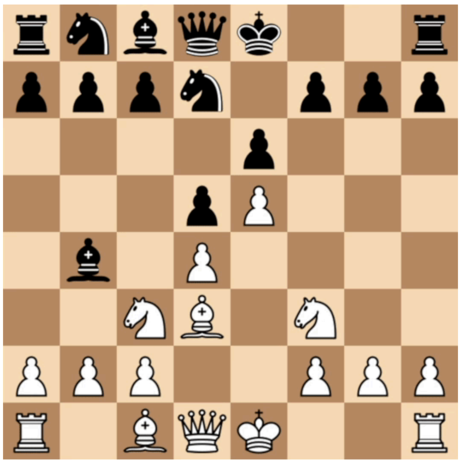
If a player moves/captures a piece and places the opponent's king in check, the program will temporarily display “check”. Thereafter, the opponent will only see moves that will either block the check, take the attacking piece, or position the king out of check.



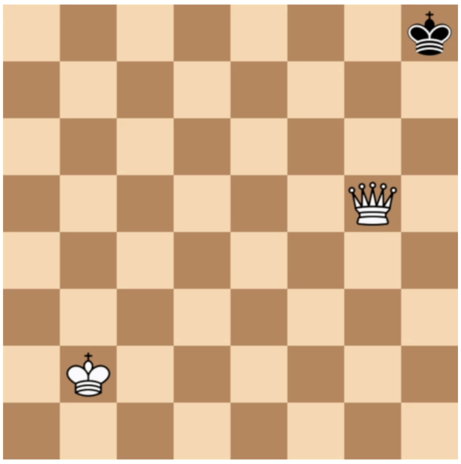


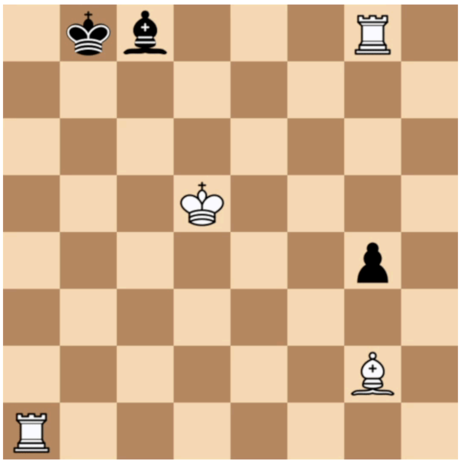
If a player is placed in check and they have no legal moves left (checkmate), the program will display “[player] wins!” (does not display “check” as well). The program will then stop running as the match has ended.





In addition to checks and checkmates, if a player is not in check and does not have any legal moves (draw by stalemate), the program will display “stalemate”. The program will then stop running as the match has ended.





Program Evaluation

Overall, I am very pleased with the final result of the program. I achieved what I wanted for the project, ensuring that the program has met all the success criteria requirements. In saying this, the project did exceed the 20-hour limit due to the complex nature of chess.

For a large portion of this project, a lot of time and effort was put towards implementing each piece's moves. Initially it was quite challenging using classes for each of the pieces - this project was the first time I have ever used classes in python. But after some online research and coding experience I felt much more comfortable using classes, which also allowed me to reap the benefits of my own hard-work. By using classes it became substantially easier to store the necessary values as well as to code the interacts with each of the pieces (ie capturing etc.). In the long run, classes proved to be very fruitful and assisted in many ways to help code the program.

In addition to this, a lot of work was put towards handling the processes behind evaluating the board's position. A large challenge I faced throughout the development of the program was implementing a process that could determine if a move is legal or not. Although the process was planned with a flowchart, it was quite difficult to implement into code. Initially, the program would loop infinitely which would freeze the program until it eventually terminated itself. After some tweaks and fixes the code appeared to work, but when tested the calculating times would exponentiate with each move to the point where it would take a solid 5-10 seconds before it showed the legal moves when a piece is clicked. In the end, I restructured the code and it ended up working much more smoothly and efficiently.

All in all, the project went according to plan. The program has a fully functioning chess board where each player performs a move until an eventual checkmate or draw. All the pieces' moves are incorporated, including special moves such as castling and en passant. It is safe to say that the program succeeded in achieving its desired purpose (overall concept from stage 1).

Stage 5 - Maintaining

In addition to the tests performed in the previous stage (stage 4), the program was tested on mac and windows computers. When testing on the windows device (originally coded on a mac device), button and text sizes did not match the original and were out-of-proportion. Hence, some additional tweaking was required to fix up the windows version.

Although this project successfully met all criteria, there are still some things that I would have liked to add to further improve the program. In its current state the program doesn't have too much functionality, users can only customise the board position and colour. I think in the future it would be very beneficial to incorporate a "save" button to allow users to save board positions. At the moment if the player closes the window, the board and its position is lost. By having some sort of save button it could allow players to continue playing a certain match at a later time. For instance, this could work by converting the board back into a FEN string which is saved into a .txt file. When the program is launched thereafter, an option may appear to select the FEN from the previous match. Although this may sound like a tedious task, the program already has a FEN-to-board process, which when reversed can convert the board back into a FEN.

In addition to this, some key components could be added to the program - a timer. Timer's are an essential component of a chess match, by implementing one it will further improve the program's capabilities. Including tools the user can use to configure the timer would additionally improve the functionality of the program. This could be shown on the editor menu when the program is launched, allowing players to tweak the amount of time they each have. Also, on a more minor scale, a resign button could be added as well as a button to play again after a match has ended.

Journal & Schedule

Schedule

I've decided to add a schedule to keep on track for finishing the program on time. Although the due date isn't until 21st of July (start of term 3), my goal is to finish coding the program by the end of term 2 (25th of June).

Task	Goal Date	Completed	Time Spent	Explanation (if not completed)
Core Elements				
Create Board & Pieces	10/6/21	8/6/21	~6 hours	-
Select Pieces & Show Moves	17/6/21	20/6/21	~18 hours	Had to restructure the board mid-way through.
Legal Moves & Evaluation *(checks, checkmates and stalemates)*	21/6/21	22/6/21	~12 hours	Ran into some minor issues along the way.
Special Moves *(castling and en passant)*	25/6/21	25/6/21	~4 hours	-
Other Elements				
FEN Editor UI	21/6/21	24/6/21	~4 hours	Got held back with other elements
Save Button Function	25/6/21	n/a	-	Content with the progress made so far, it didn't feel necessary to complete as much time has already been spent (also it was well over the time limit of 20 hrs).

Journal

Date: 7/6/21

Finished research and completed documentation on the first stages, started to code the board.

Date: 8/6/21

Finished coding the board and implementing the pieces. At the moment the program is just a board with images of pieces on it.

Date: 10/6/21

You can now select a piece and the background behind it will light up. At the moment you can click on either player's pieces (regardless of who's turn it is). After fixing the turn issue, my next goal is to implement moves of each type of piece.

Date: 13/6/21

For the past few days I have been working on the pawn moves. I have added all its moves, the double move at the start, the single move forward, and the capture move. I have yet to add the en passant move - I will do this at a later date.

Date: 15/6/21

Instead of coding moves for each piece type (rook, bishop and queen), I have figured out that I can split it into two different aspects. One for straight moves and the other for diagonal moves. The rook is just straight moves, the bishop is just diagonal moves, and the queen is both straight and diagonal moves. I've finished coding these moves and they all work, my next goal is to work on the knight moves.

Date: 16/6/21

When working on the knight moves, I found that how I've set up the board has made coding some of the moves quite difficult. At the moment the board is split into a 64-long array, with each square being a number between 0-63. I've found that some offsets for the moves can produce incorrect moves - for example, if a knight move is off the board it shouldn't appear, but with the current board format the program thinks there actually is a move - which is incorrect. So today I have decided to restructure the board, from a 64-long array into a double array (8-long array with 8-long array's). This way it is much easier to determine whether a move is on the board or not. Although, by doing so I had to go back and re-code the previous moves to work with the new board structure. This took quite a bit of time and effort, but I think it'll help a lot when coding the other piece's moves.

Date: 17/6/21

Finished coding the knight moves. It was much easier to code with the new board format.

Date: 18/6/21

Finished coding the king moves, but I have yet to implement a check system. At the moment the king acts like any other piece, you can capture the king... once the evaluation system is implemented this should be all fixed up.

Date: 20/6/21

Previously the moves were highlighted by a yellow square, today I've fixed these so a dot is shown instead. I did have to do some research into how to layer transparent images over each other, which I found that I needed to use a python module named PIL (Python-Imaging-Library) to keep the transparency when layering the images. After implementing the new dots the moves look much more clearer now.

Date: 21/6/21

Started to code the evaluation system. I ran into quite a few issues where the process would loop infinitely. After much tweaking, the process seemed to work, except in some cases the process would remove moves that were legal and keep moves that weren’t legal - I wasn’t quite sure why this was happening.

Date: 22/6/21

Finished coding the evaluation system. The program now only shows the legal moves and at the moment prints ‘check’, ‘checkmate’ or ‘stalemate’ in the console. I did manage to find and fix the issues that were happening yesterday, but for some reason the process took quite some time (I would click the piece and sometimes it would take up to 5 seconds before the moves were shown). After some digging through the code, I found what was causing it and decided to restructure the code so it worked more efficiently.

Date: 23/6/21

I made a start on the UI and implemented castling. This was easier than I originally thought as the evaluation system came in very handy and did most of the work.

Date: 24/6/21

Today I finished the UI. You now can change the starting board position and even the colour of the board. I know the changing board colour wasn’t necessary, I just felt like adding it.

Date: 25/6/21

Finished coding en passant moves. The program now has all the necessary moves implemented. In addition to this, I did some extra testing and all the moves seem to be working well.

Date: 27/6/21

When testing the program on my main computer (windows) I found that the program did seem to look the same. All the button and text sizes did not match the original and were out-of-proportion. I had to do some minor adjustments so the program looked similar to that on my mac device.