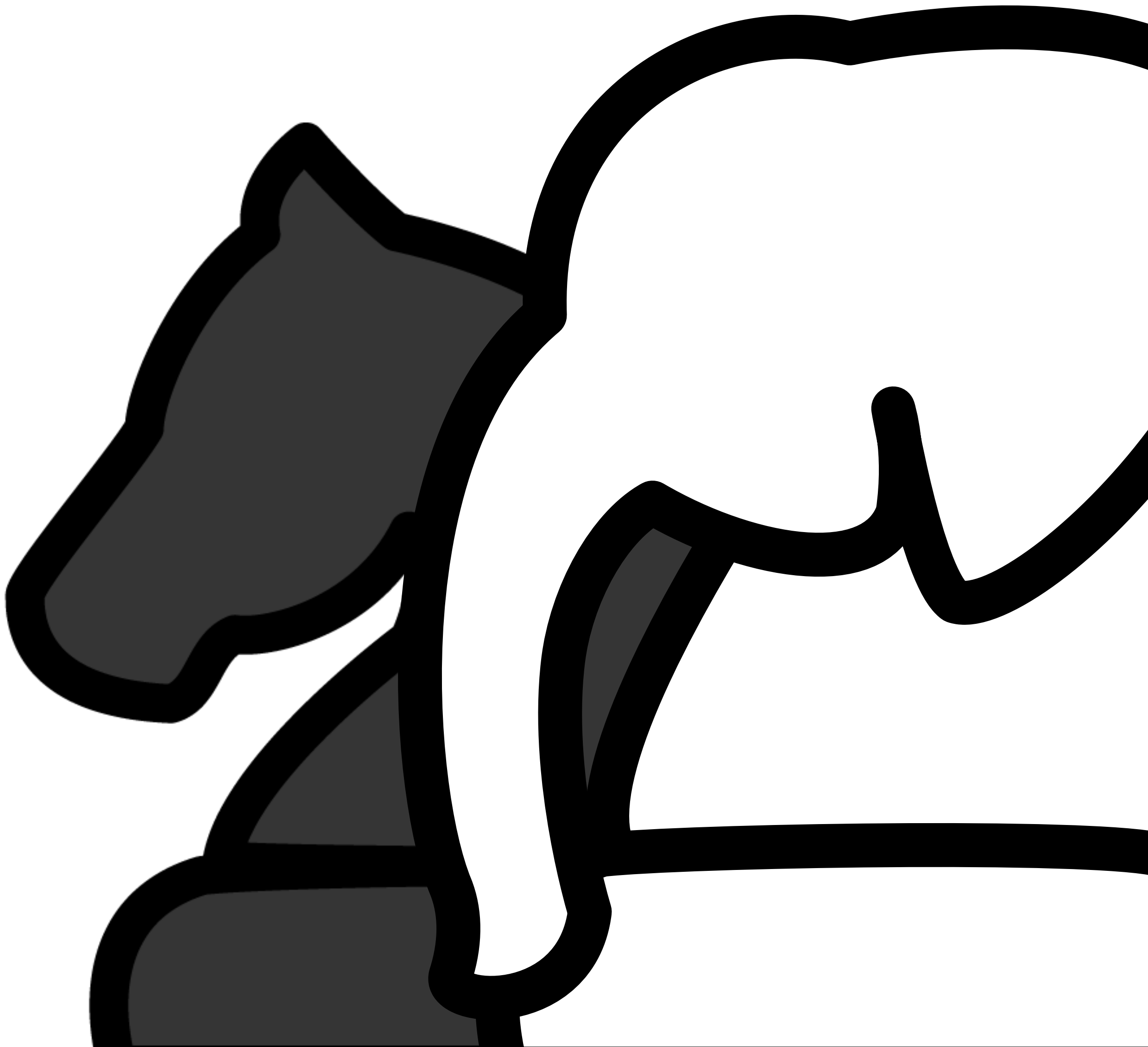
Tjatoer!



Contents

Prologue 3

Chapter 1: Introduction to Chess and Chess Variants

Rules and Pieces 4

Algebraic Chess Notation 7

A Little About Chess Variants 13

Chapter 2: Overview of Tjatoer!

New Board and Rules 14

The Pieces 17

Chapter 3: What I Have Done

Graphical Interface 23

Gameplay 25

Engine 26

Chapter 4: What Needs to be Implemented

Graphical Interface 30

Gameplay 40

Engine 43

Chapter 5: Summary

Goals 51

Optional Additions 52

Prologue

I started this project on a whim, thinking this would be a good way to pass time. But now I am obsessed with it, here I am bashing my head to a keyboard for the past weeks to no avail. I think it has so much potential, yet, my ideas are not on par with my skills and since I am a mild perfectionist, I need someone else to help me finish the project. I am more inclined towards the graphical side of things with little to no experience of Python or coding in general, and with this document, I wish to provide helpful guidelines in improving this project.

In this document, you will find a brief explanation of the game of chess and in case you are not familiar with it, and then Tjatoer! as its variant, moreover, in chapter 3, there will be an overview of what has been done in the project. And in chapter 4 there will be an exposition on the things that I would like to be implemented in the program, but couldn't because of my lack of skill. And finally, chapter 5 will give you a summarization in the form of a to-do list for convenience.

With hopeful regards,

Cheesewaffle

Chapter 1: Introduction to Chess and Chess Variants

If you are already familiar with the game of chess, its notation, rules, and pieces, you can skip this part, as this chapter is solely dedicated to giving a brief explanation of the game and an explanation of the notation used in recording the game

Rules and Pieces

The Flow of the Game

The chessboard must be positioned so that a light square should be on the bottom right corner of the board. White always move first, A chess move is made after both white and black has made a move, the number of the move is written before a pair of moves, one for white and one for black, for example, if white makes his first move by moving their knight to f3 then black responds with moving their knight to f6, then the move would be written as 1. Nf3 Nf6. Then white's move after that, say for example pawn to e5, would be recorded as 2. e5, so the whole game thus far is written as "1. Nf3 Nf6 2. e5".

How the Pieces Move

A piece is moved to either an unoccupied square or one occupied by an opponent's piece, which is captured and removed from play. With the sole exception of en passant, all pieces capture by moving to the square that the opponent's piece occupies. Moving is compulsory; a player may not skip a turn, even when having to move is detrimental.

Each piece has its way of moving. In the diagrams, the dots mark the squares to which the piece can move if there are no intervening piece(s) of either color (except the knight, which leaps over any intervening pieces). All pieces except the pawn can capture an enemy piece if it is located on a square to which they would be able to move if the square was unoccupied.

- **King**
  moves one square in any direction. There is also a special move called castling that involves moving the king and a rook. The king is the most valuable piece — attacks on the king must be immediately countered, and if this is impossible, immediate loss of the game ensues (see Check and checkmate below).
- **Rook**
  Can move any number of squares along a rank or file, but cannot leap over other pieces. Along with the king, a rook is involved during the king's castling move.
- **Bishop**
  Can move any number of squares diagonally, but cannot leap over other pieces.
- **Queen**

Combines the power of a rook and bishop and can move any number of squares along a rank, file, or diagonal, but cannot leap over other pieces.

- **Knight**
  Moves to any of the closest squares that are not on the same rank, file, or diagonal. (Thus, the move forms an "L"-shape: two squares vertically and one square horizontally, or two squares horizontally and one square vertically.) The knight is the only piece that can leap over other pieces.
- **Pawn**
  Can move forward to the unoccupied square immediately in front of it on the same file, or its first move, it can advance two squares along with the same file, provided both squares are unoccupied (black dots in the diagram). A pawn can capture an opponent's piece on a square diagonally in front of it by moving to that square (black crosses). A pawn has two special moves: the en passant capture and promotion.

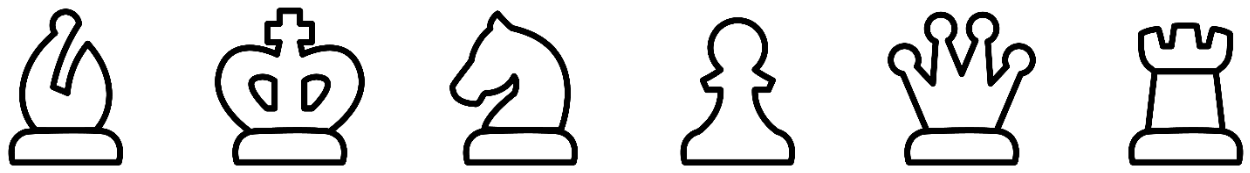


Figure 1.1: My design of the regular chess pieces

To better understand the moves, let us look at some diagrams, (yellow circles in pawn moves are valid only in Tjatoer!)

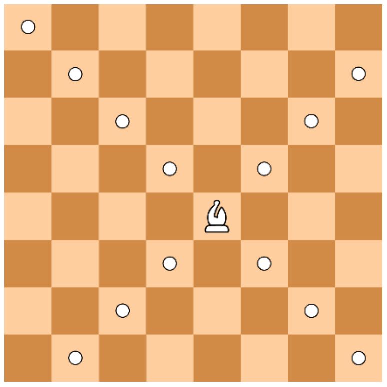


Figure 1.2: Bishop moves

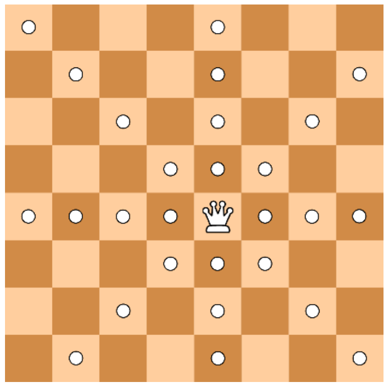


Figure 1.3: Queen moves

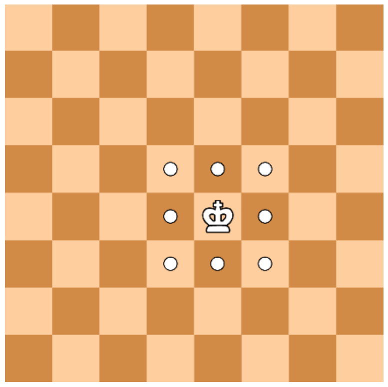


Figure 1.4: King moves

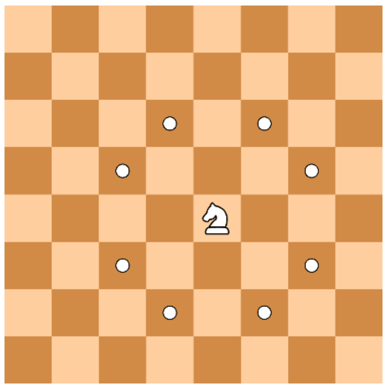


Figure 1.5: Knight moves

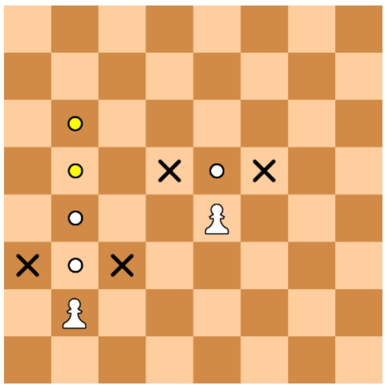


Figure 1.6: Pawn moves

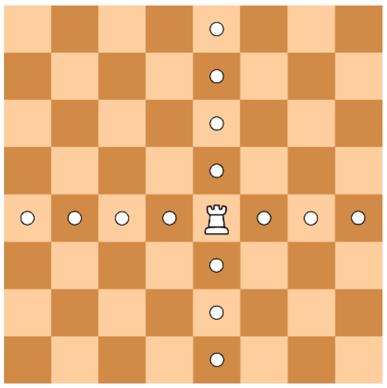


Figure 1.7: Rook moves

Check and Checkmate

When a king is under immediate attack, it is said to be in check. A move in response to a check is legal only if it results in a position where the king is no longer in check. This can involve capturing the checking piece; interposing a piece between the checking piece and the king (which is possible only if the attacking piece is a queen, rook, or bishop and there is a square between it and the king); or moving the king to a square where it is not under attack. Castling is not a permissible response to a check.

The object of the game is to checkmate the opponent; this occurs when the opponent's king is in check, and there is no legal way to get it out of check. It is never legal for a player to make a move that puts or leaves the player's king in check. In casual games, it is common to announce "check" when putting the opponent's king in check, but this is not required by the rules of chess and is not usually done in tournaments.

Castling and En Passant

This has no diagrams since it is not relevant to Tjatoer! (you'll see why later). Once in every game, each king can make a special move, known as castling. Castling consists of moving the king two squares along with the player's first rank toward a rook on the same rank and then placing the rook on the last square that the king crossed.

Castling is permissible if the following conditions are met:

- Neither the king nor the rook has previously moved during the game.
- There are no pieces between the king and the rook.
- The king is not in check, and will not pass through or land on any square attacked by an enemy piece.

Castling is still permitted if the rook is under attack, or if the rook crosses an attacked square.

When a pawn makes a two-step advance from its starting position and there is an opponent's pawn on a square next to the destination square on an adjacent file, then the opponent's pawn can capture it en passant ("in passing"), moving to the square the pawn passed over. This can be done only on the turn immediately following the enemy pawn's two-square advance; otherwise, the right to do so is forfeited.

Algebraic Chess Notation

Back in the 1980s, a revolutionary way of recording chess games was popularized, back then, the previous way of recording games is to use the descriptive notation (I won't explain it here, I don't fully grasp it either) but it has since been replaced with the algebraic notation, and that's the method I'm going to explain here.

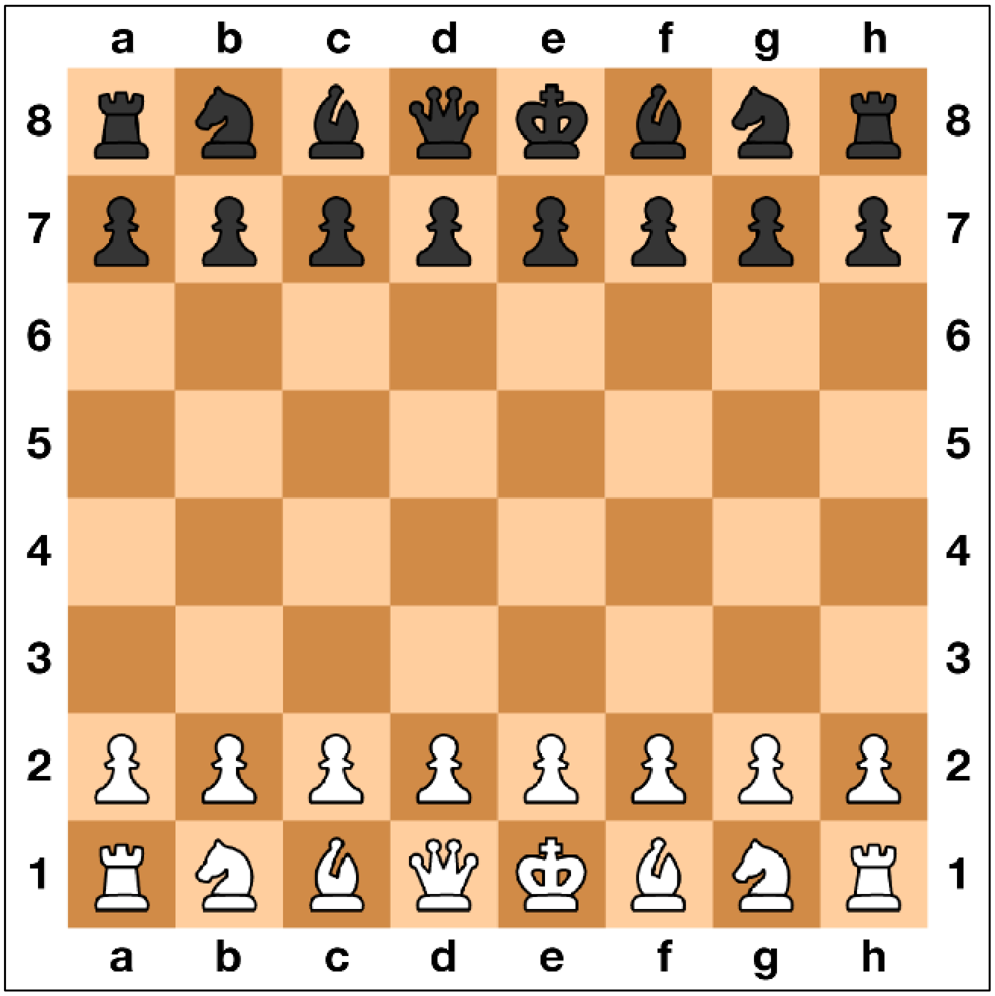


Figure 1.8: Initial piece setup for chess

Before we move into move recording, we need to acknowledge the board and initial piece setup. Each square of the chessboard is identified by a unique coordinate pair, a letter, and a number, from White's point of view. The vertical columns of squares, called files, are labeled a through h from White's left (the queenside) to right (the kingside). The horizontal rows of squares, called ranks, are numbered 1 to 8 starting from White's side of the board. Thus each square has a unique identification of file letters followed by rank number. For example, the initial square of White's king is designated as "e1".

Each piece type (other than pawns) is identified by an uppercase letter. English-speaking players use the letters K for the king, Q for a queen, R for a rook, B for a bishop, and N for a knight (since K is already used).

Move

Now for the moves, each move of a piece is indicated by the piece's uppercase letter, plus the coordinate of the destination square. For example, Be5 (move a bishop to e5), Nf3 (move a knight to f3). For pawn moves, a letter indicating pawn is not used, only the destination square is given. For example, c5 (move a pawn to c5).

Capture

When a piece makes a capture, an "x" is inserted immediately before the destination square. For example, Bxe5 (bishop captures the piece on e5). When a pawn makes a capture, the file from which the pawn departed is used to identify the pawn. For example, exd5 (pawn on the e-file captures the piece on d5).

Check, Double Check, and Checkmate and Draws

A move that places the opponent's king in check usually has the symbol "+" appended. Double-check is usually indicated the same as a check but is sometimes represented specifically as "++". Checkmate after moves is represented by the symbol "#" in standard FIDE notation and PGN. As for draws, there are ways to achieve it, though I will only be discussing the ways that are relevant to Tjatoer! namely:

- **Stalemate**

  If the player to move has no legal move, but is not in check, the position is a stalemate, and the game is drawn.
- **Dead position**

  If neither player is able to checkmate the other by any legal sequence of moves, the game is drawn. For example, if only the kings are on the board, all other pieces having been captured, checkmate is impossible, and the game is drawn by this rule. On the other hand, if both players still have a knight, there is a highly unlikely yet theoretical possibility of a checkmate, so this rule does not apply. The dead position rule supersedes the previous rule which referred to "insufficient material", extending it to include other positions where checkmate is impossible, such as blocked pawn endings where the pawns cannot be attacked.

- **Threefold repetition**

  This most commonly occurs when neither side is able to avoid repeating moves without incurring a disadvantage. In this situation, either player can claim a draw; this requires the players to keep a valid written record of the game so that the claim can be verified by the arbiter if challenged. The three occurrences of the position need not occur on consecutive moves for a claim to be valid. The addition of the fivefold repetition rule in 2014 requires the arbiter to intervene immediately and declare the game a draw after five occurrences of the same position, consecutive or otherwise, without requiring a claim by either player. FIDE rules make no mention of perpetual check; this is merely a specific type of draw by threefold repetition.
- **Fifty-move rule**

  If during the previous 50 moves no pawn has been moved and no capture has been made, either player can claim a draw. The addition of the seventy-five-move rule in 2014 requires the arbiter to intervene and immediately declare the game drawn after 75 moves without a pawn move or capture, without requiring a claim by either player. There are several known endgames where it is possible to force a mate but it requires more than 50 moves before a pawn move or capture is made; examples include some endgames with two knights against a pawn and some pawnless endgames such as queen against two bishops. Historically, FIDE has sometimes revised the fifty-move rule to make exceptions for these endgames, but these have since been repealed. Some correspondence chess organizations do not enforce the fifty-move rule.

Castling, En Passant, Pawn Promotion and other Exceptions

Again, there won't be any examples here since castling and en passant are not relevant to Tjatoer! Castling is indicated by the special notations 0-0 (for kingside castling) and 0-0-0 (queenside castling). En passant captures are indicated by specifying the capturing pawn's file of departure, the "x", the destination square (not the square of the captured pawn), and (optionally) the suffix "e.p." indicating the capture was en passant. For example, exd6 e.p.

When a pawn promotes, in Portable Game Notation (PGN), as well as in Chess Life and English Wikipedia, pawn promotion is indicated by the equals sign (e8=Q).

When two (or more) identical pieces can move to the same square, the moving piece is uniquely identified by specifying the piece's letter, followed by (in descending order of preference):

- the file of departure (if they differ);
- the rank of departure (if the files are the same but the ranks differ);
- both the file and rank of departure (if neither alone is sufficient to identify the piece, which occurs only in rare cases where a player has three or more identical pieces able to reach the same square, as a result of one or more pawns having promoted).

The notation 1–0 after moves indicates that White won, 0–1 indicates that Black won, and ½–½ indicates a draw. To better understand chess notation, let us look at an example game. If it's too small to see, it's probably for aesthetic and layout purposes, help yourself and zoom in.

Example Game

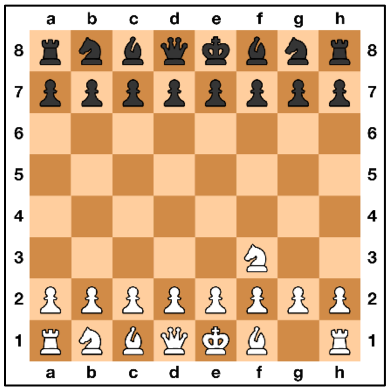


Figure 1.9

The position after 1. Nf3

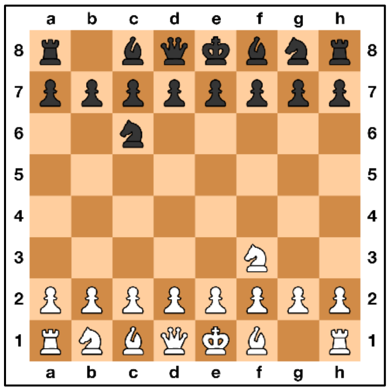


Figure 1.10

After 1. Nf3 black responds with 1...Nc6, thus the whole move is written as 1. Nf3 Nc6

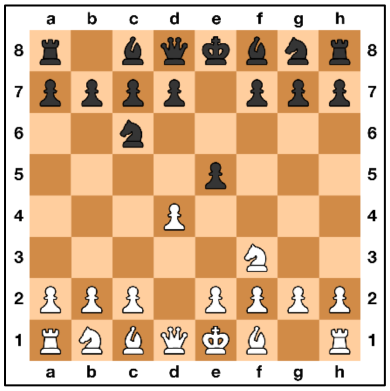


Figure 1.11

The position after 1. Nf3 Nc6 2. d4 e5

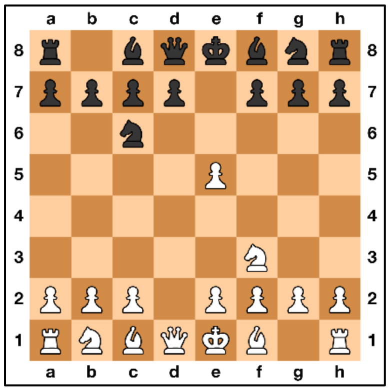


Figure 1.12

After 1. Nf3 Nc6 d4 e5, white captures black's e5 pawn, it is written as 3. dxe5 since it is the third move

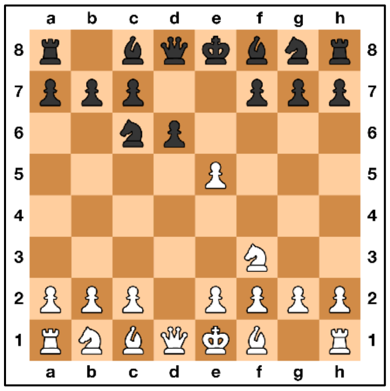


Figure 1.13

The position after 1. Nf3 Nc6 2. d4 e5 3. dxe5 d6, now it is white's turn, and both of their knights can move to the d2 square

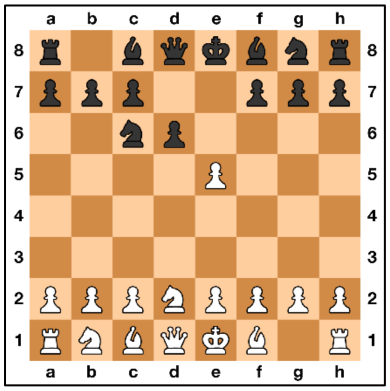


Figure 1.14

One of white's knights has moved to the d2 square, and since the knight that moved was from f3, the move is written as 4.Nfd2 (note that if the knight that was on f3 was on b3 instead, the move will be N3d2 since both knights are on the same file in that case)

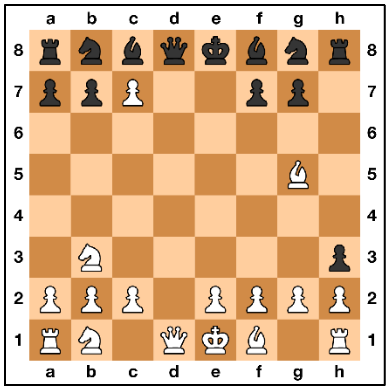


Figure 1.15

Fast forward a little bit, the position after 1. Nf3 Nc6 2. d4 e5 3. dxe5 d6 4. Nfd2 h6 5. exd6 Nb8 6. dxc7 h5 7. Nb3 h4 8. Bg5 h3

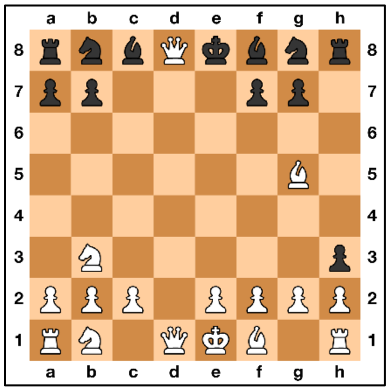


Figure 1.16

Final example, after 1. Nf3 Nc6 2. d4 e5 3. dxe5 d6 4. Nfd2 h6 5. exd6 Nb8 6. dxc7 h5 7. Nb3 h4 8. Bg5 h3 (black's final move) then 9. cxd8=Q# 1-0 is checkmate.

in case you didn't get that last one, it's the c pawn capturing the queen on d8 (cxd8), then promoting to a queen (=Q), resulting in a checkmate (#), and a win for white (1-0), so the whole move is (cxd8=Q# 1-0). Thus the whole game can be written as 1. Nf3 Nc6 2. d4 e5 3. dxe5 d6 4. Nfd2 h6 5. exd6 Nb8 6. dxc7 h5 7. Nb3 h4 8. Bg5 h3 9. cxd8=Q# 1-0.

A Little About Chess Variants

A chess variant is a game "related to, derived from, or inspired by chess". Such variants can differ from chess in many different ways. "International" or "Western" chess itself is one of a family of games that have related origins and could be considered variants of each other. Chess developed from chaturanga, from which other members of this family, such as shatranj, Tamerlane chess, shogi, and xiangqi also evolved.

Many chess variants are designed to be played with the equipment of regular chess. Most variants have a similar public-domain status as their parent game, but some have been made into commercial proprietary games. Just as in traditional chess, chess variants can be played over the board, by correspondence, or by computer. Some internet chess servers facilitate the play of some variants in addition to orthodox chess.

In the context of chess problems, chess variants are called heterodox chess or fairy chess. Fairy chess variants tend to be created for problem composition rather than actual play.

There are thousands of known chess variants. The Classified Encyclopedia of Chess Variants catalogs around two thousand, with the preface noting that with creating a chess variant being relatively trivial—many were considered insufficiently notable for inclusion.

Chapter 2: Overview of Tjatoer!

Chess with big boards is not new, there is a variant called Terachess which is very similar to Tjatoer! which has its own set of unique pieces. the problem is, there are not many options in playing these variants, finding a human to play chess variants is hard enough, what's even harder is to find an engine capable of playing such variants. But now, I want to focus on only one variant, and especially an engine that can play such a variant.

New Board and Rules

Now that we've got that out of the way, let us talk about Tjatoer! It is pronounced tʃatur (Chatoor) which means chess in Indonesian, though it is spelled in archaic forms of spelling for novelty's sake, I guess. I scaled up the board up from 8x8 to a whopping 16x16, enabling both players to have more space, and of course allowing the way for more powerful pieces.

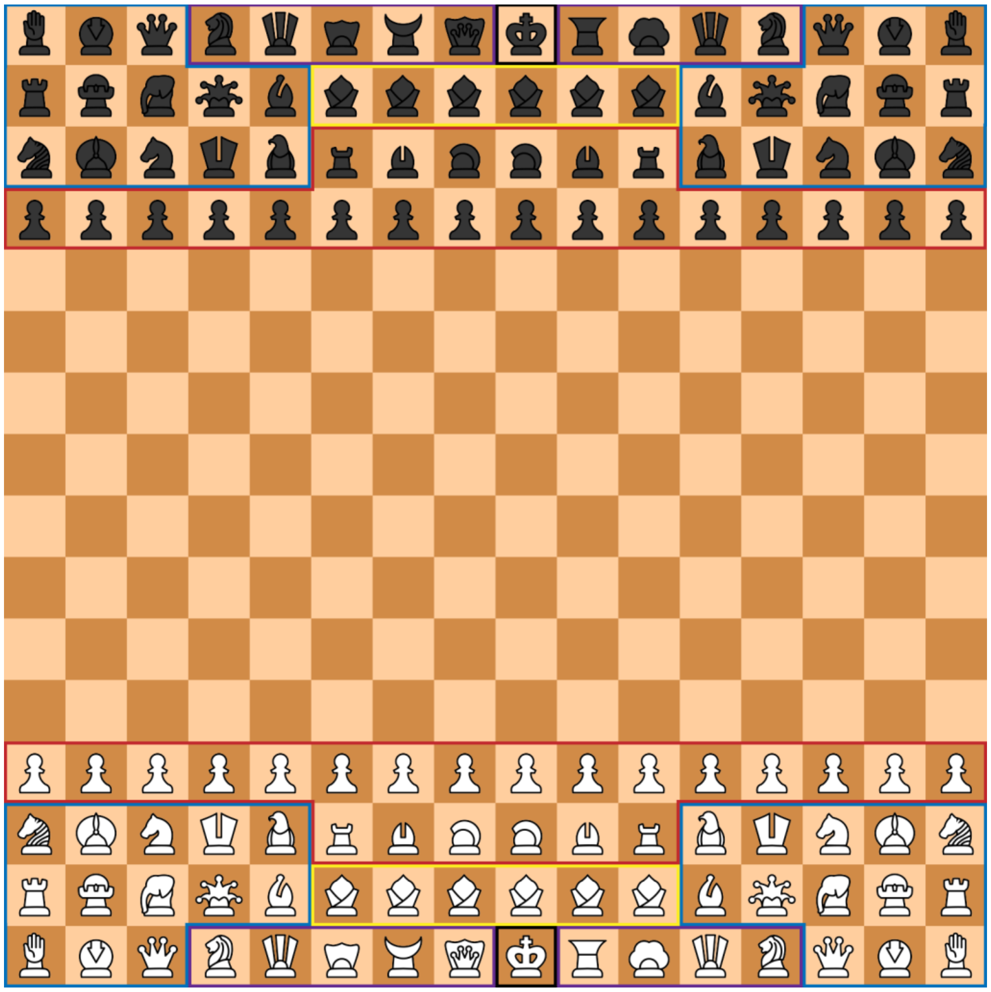


Figure 2.1: Initial piece setup in Tjatoer! (I won't even bother putting notation marks because it will make it unbelievably small)

You may notice that I put colored outline boxes on the board, this won't show in the actual program but I wanted to categorize the pieces in the game since there is so many, namely:

- Purple outline box (the King's Court)
  This is the king's court, it is where the most powerful pieces sit, from left to right they are, the queen, xiphodon, viscount, oknha ferz, lance, esquire, and duchess
- Yellow outline box (the King's Guards)
  The piece in the king's guard can promote to one of the pieces in the king's court once they reached the last rank, they only have one type, the man
- Blue outline box (the Queen's court)
  These are the normal pieces, not as powerful as the king's court but more powerful than the king's guards and infantry, from left to right downwards they are the zebra, imam, knight, archbishop, hawk, rook, chariot, gajah, jester, bishop, yishi, and upasaka
- Red outline box (the Infantry)
  The pieces in the infantry are weak, but they can promote to a specific piece, or in the case of the pawn, can promote to any piece from the queen's court, their first row consist of pawns as for the second row from left to right there are the shrook, thalia, and warrior
- Black outline box (the king, no explanation needed)

Regarding those pieces, I took some creative liberty and added:

- **Archbishop**
  Moves and captures like a bishop and knight, notated by the letter A
- **Chariot**
  Moves and captures like a gajah and rook, notated by the letter C
- **Duchess**
  Moves and captures like a hawk and bishop, notated by the letter D
- **Esquire**
  Moves and captures like a hawk and rook, notated by the letter E
- **Ferz**
  Moves and captures like a knight and queen, notated by the letter F
- **Gajah**
  Moves and captures like a knight but longer, won't be obstructed by enemy or ally piece, notated by the letter G
- **Hawk**
  Moves and captures like a gajah and a knight, notated by the letter H
- **Imam**

Moves and captures like a rook and knight, notated by the letter I

- **Jester**
  Moves and captures like a gajah and bishop, notated by the letter J
- **Lance**
  Moves and captures like a gajah, knight, and queen, notated by the letter L
- **Man**
  Moves and captures like a king and knight, notated by the letter M
- **Oknha**
  Moves and captures like a gajah and queen, notated by the letter O
- **Shrook**
  Moves and captures a maximum of two squares orthogonally, notated by the letter S
- **Thalia**
  Moves and captures a maximum of two squares diagonally, notated by the letter T
- **Upasaka**
  Moves and captures like a knight and shrook, but only one square backward, notated by the letter U
- **Viscount**
  Moves and captures one square diagonally then moves and captures any distance orthogonally, notated by the letter V
- **Warrior**
  Moves like a king but can only capture 2 squares forward, Notated by the letter W
- **Xiphodon**
  Moves and captures one square orthogonally then moves and captures any distance diagonally, notated by the letter X
- **Yishi**
  Moves and captures like a knight and thalia, but only one square diagonally backward, notated by the letter Y
- **Zebra**
  Moves and captures like a knight but can do it continuously, notated by the letter Z

Neatly, the total types of pieces in this game is 26, one for each letter of the alphabet, In addition to those new pieces, I also tweaked some rules to befit the game, such as:

- The threefold repetition rule is now the fivefold repetition rule;

- All the pieces of regular chess move the same way except for the Pawn;
- Baring the enemy king (capturing all other pieces and leave a sole king) will be considered a win;
- Since the board is now bigger, the now files range a-p and the ranks range 1-16;
- The 50-move rule is now the 100-move rule;
- The man promotes on the last rank to the pieces in the king's court;
- The pawn promotes on the last rank to the pieces in the queen's court;
- The shrook promotes to a rook on the last rank;
- The thalia promotes to a bishop on the last rank;
- The warrior promotes to a knight on the last rank;
- The Pawn would also be able to move up to 4 squares on their initial move as opposed to the 2 squares in regular chess;
- There is no castling, nor en passant (this is why you don't need to understand it).

To better imagine what these would apply, let us look at some piece designs and diagrams

The Pieces

I have made two sets of piece designs for Tjatoer! Alfaerie (designed by David Howe) and Simple (Which is fully designed by me). We will mainly use the simple set on the diagrams.

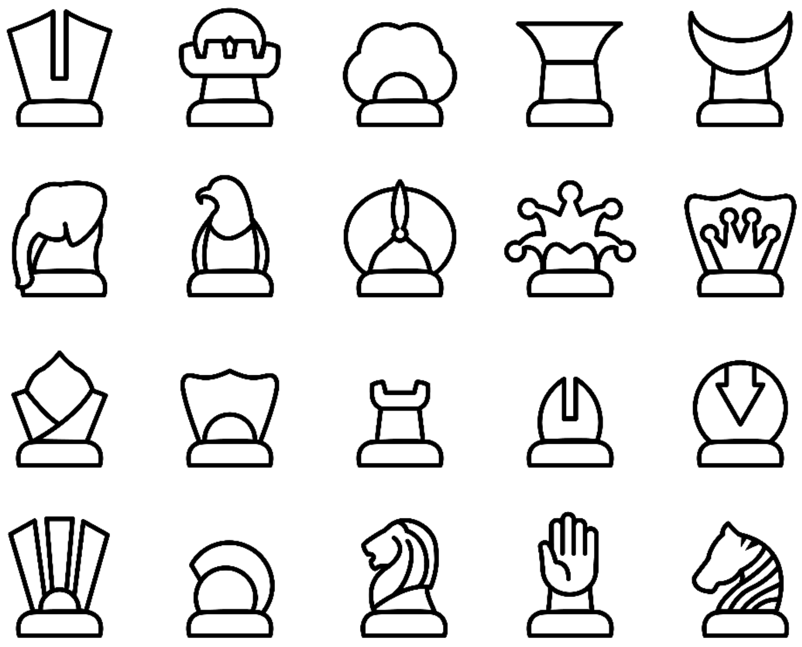


Figure 2.2: (From left to right, downwards) The simple style textures for the Archbishop, Chariot, Duchess, Esquire, Ferz, Gajah, Hawk, Imam, Jester, Lance, Man, Oknha, Shrook, Thalia, Upasaka, Viscount, Warrior, Xiphodon, Yishi, Zebra

The archbishop is depicted as a cardinal headpiece in the simple style of pieces. one way to counter this piece is to attack it from the orthogonal direction, or with a piece with gajah moves. This piece has a value of 825

The chariot is depicted as some sort of a lighthouse, with elements of the rook and imam in the simple style of pieces. one way to counter this piece is to attack it from of the four diagonals direction, or with a piece with knight moves. This piece has a value of 875

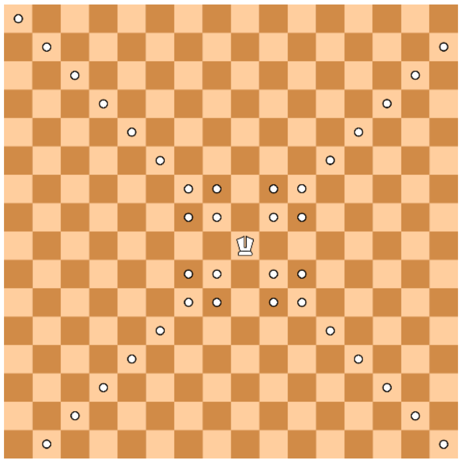


Figure 2.3: Archbishop moves

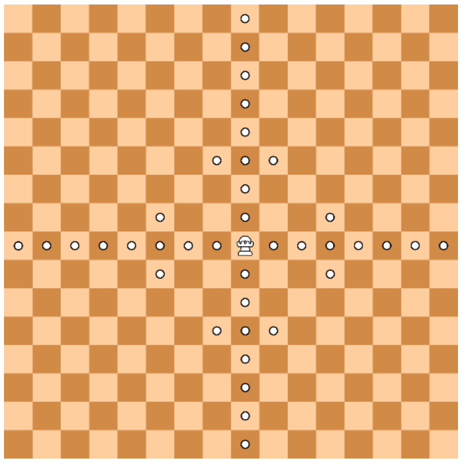


Figure 2.4: Chariot moves

The duchess is depicted as a Russian traditional hat, in the simple style of pieces. one way to counter this piece is to attack it from of the four orthogonal directions. This piece has a value of 900

The esquire is depicted as an African traditional hat, in the simple style of pieces. one way to counter this piece is to attack it from of the four diagonal directions. This piece has a value of 1075

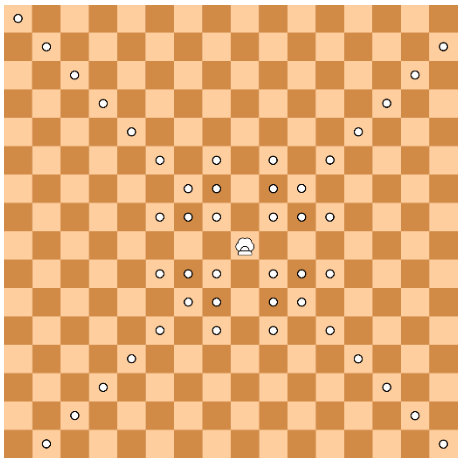


Figure 2.5: Duchess moves

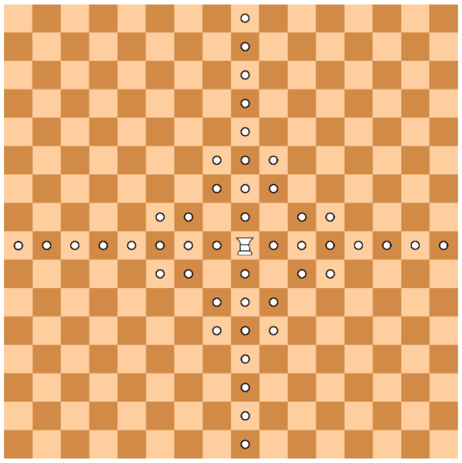


Figure 2.6: Esquire moves

The ferz is depicted as an Indonesian, west-Sumatran traditional headdress in the simple style of pieces. one way to counter this piece is to attack it with a piece with gajah moves. This piece has a value of 1250

The gajah is depicted as an elephant in the simple style of pieces. there are a lot of ways to counter this piece. This piece is color bounded (can only go to its original square color, if it starts on a light square then it can only go to light squares). This piece has a value of 250

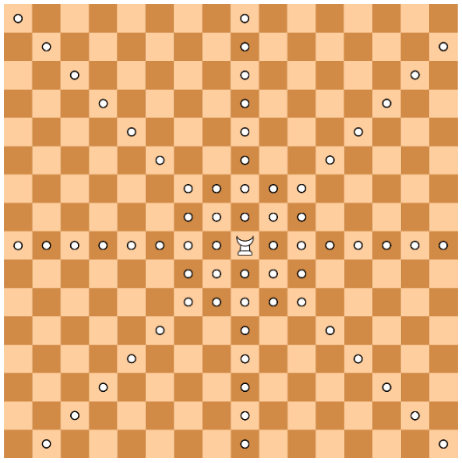


Figure 2.7: Ferz moves

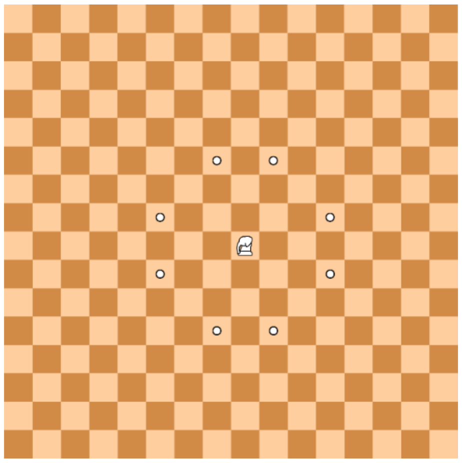


Figure 2.8: Gajah moves

The hawk is depicted as a hawk in the simple style of pieces. there are a lot of ways to counter this piece since it is a basic piece. This piece has a value of 450

The imam is depicted as a turban headpiece in the simple style of pieces. one way to counter this piece is to attack it from one of the four diagonal directions, or with a piece with gajah moves. This piece has a value of 925.

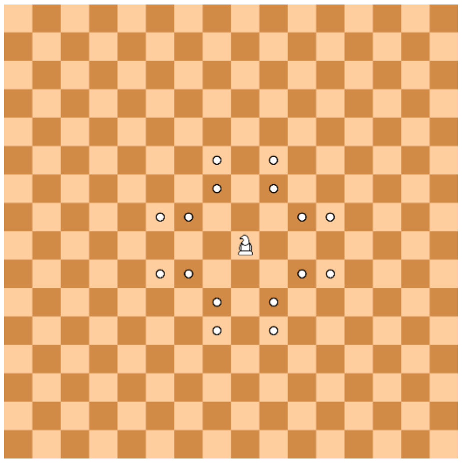


Figure 2.9: Hawk Moves

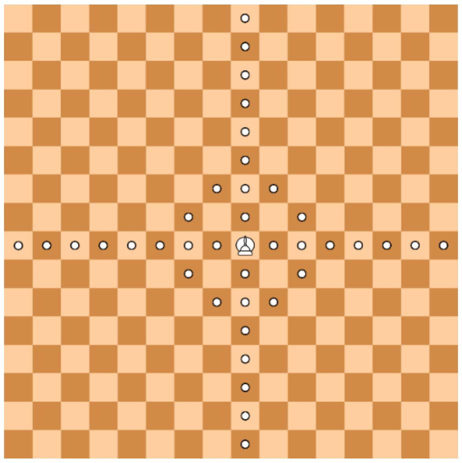


Figure 2.10: Imam moves

The jester is depicted as the hat of a court jester in the simple style of pieces. one way to counter this piece is to attack it from of the four orthogonal directions, or with a piece with knight moves. This piece has a value of 775

The lance is depicted as the elements of the oknha and queen in the simple style of pieces. This piece is the strongest and there is no real way of countering it, it can maybe be trapped by a combination of pieces. This piece has a value of 1500

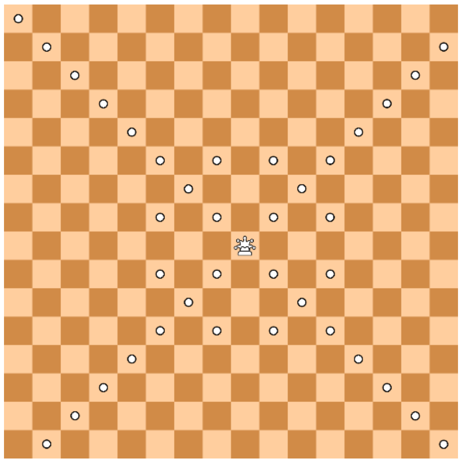


Figure 2.11: Jester moves

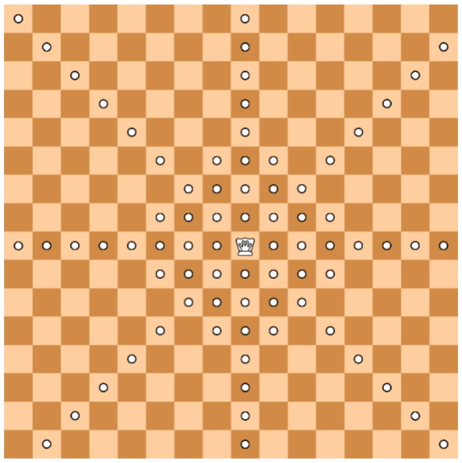


Figure 2.12: Lance moves

The man is depicted as an Indonesian, Acehnese traditional hat in the simple style of pieces. this piece can promote the more powerful pieces. one strategy to play this piece is to surround the king with this piece. This piece has a value of 400

The oknha is depicted as an Indonesian traditional Javanese hat in the simple style of pieces. one way to counter this piece is to attack it with a piece with knight moves. This piece has a value of 1200

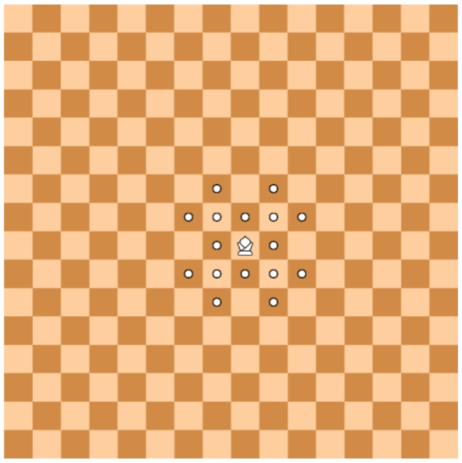


Figure 2.13: Man moves

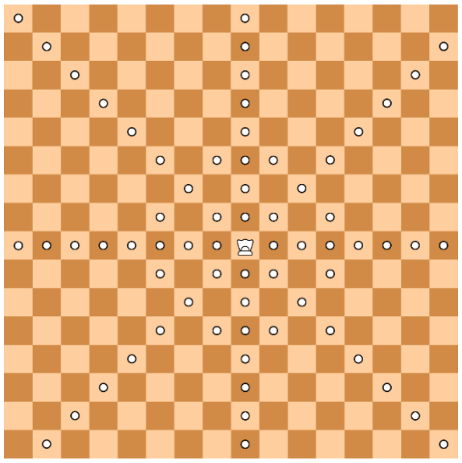


Figure 2.14: Oknha move

The shrook is short for short rook, it is depicted as a mini rook in the simple style of pieces, there are many ways to counter this piece since it is very weak. This piece has a value of 150.

The thalia is depicted as a mini bishop in the simple style of pieces, there are many ways to counter this piece since it is very weak. This piece has a value of 150.

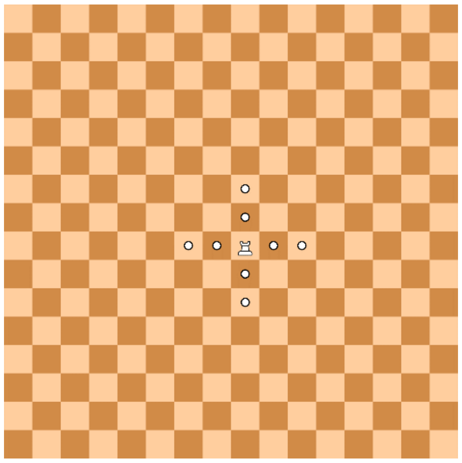


Figure 2.15: Shrook moves

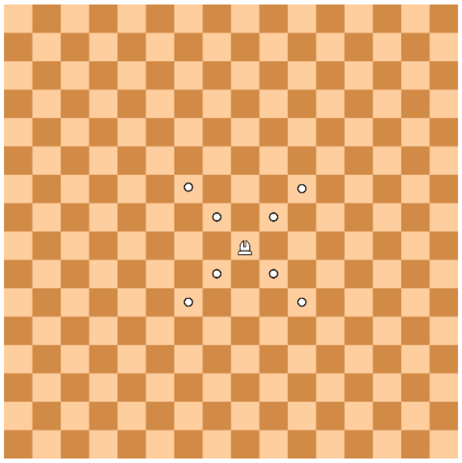


Figure 2.16: Thalia moves

The upasaka is depicted as a circle with an arrow (avatar reference) in the simple style of pieces, there are many ways to counter this piece since it is very weak. This piece has a value of 175.

The viscount is depicted as an 18th-century collar design in the simple style of pieces, the way to counter this piece is surprisingly easy, any piece with rook or bishop moves can do it from afar. This piece has a value of 840.

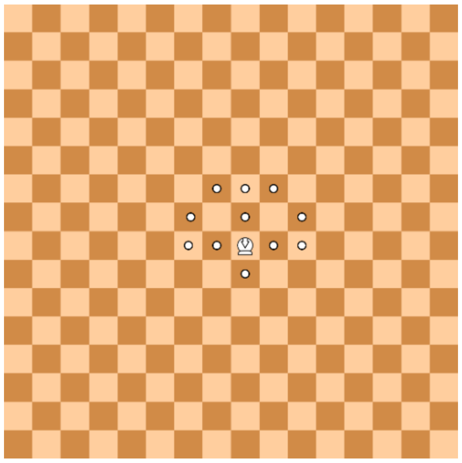


Figure 2.17: Upasaka Moves

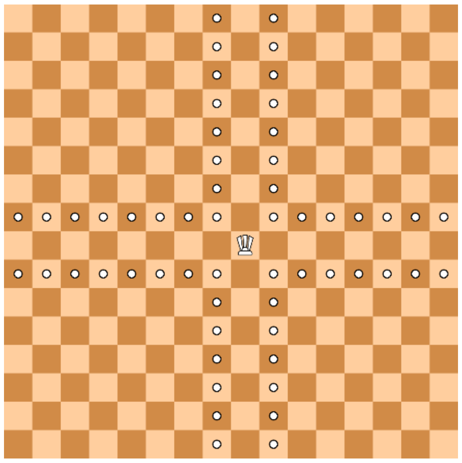


Figure 2.18: Viscount moves

The warrior is depicted as a roman headpiece in the simple style of pieces, there are many ways to counter this piece since it is very weak. This piece has a value of 165.

The xiphodon is depicted as a lion in the simple style of pieces, the way to counter this piece is surprisingly easy, any piece with rook or bishop moves can do it from afar. This piece has a value of 610.

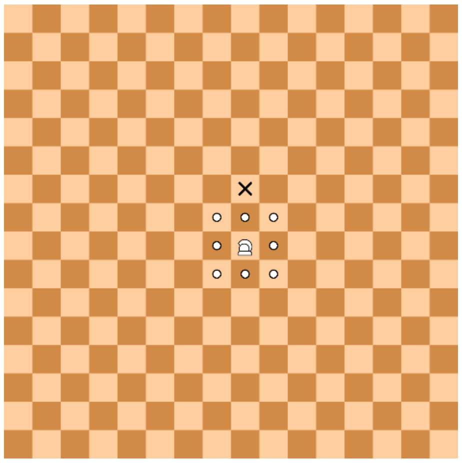


Figure 2.19: Warrior moves

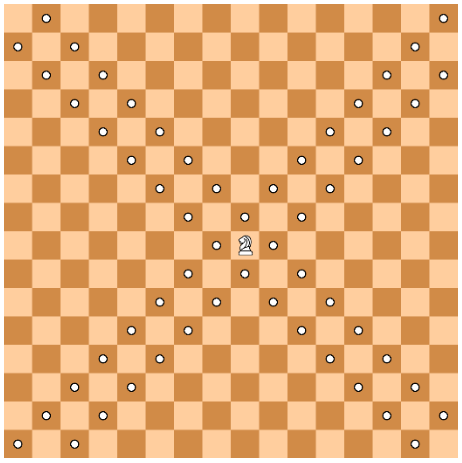


Figure 2.20: Xiphodon moves

The yishi is depicted as a hand in the simple style of pieces, there are many ways to counter this piece since it is very weak. This piece has a value of 175.

The zebra is depicted as a zebra in the simple style of pieces, the way to counter this piece is to attack it with a range piece since it cannot move in a straight line. This piece has a value of 315.

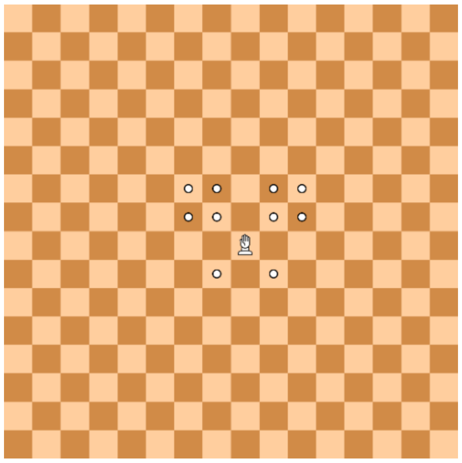


Figure 2.21: Yishi moves

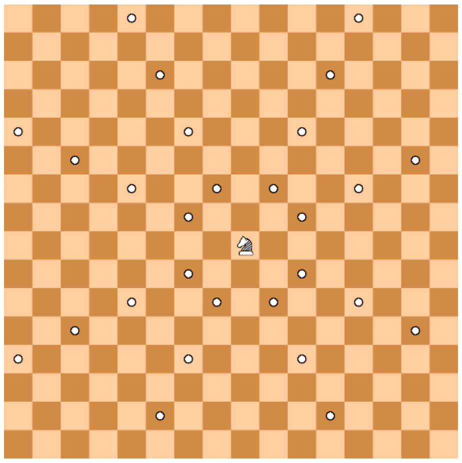


Figure 2.22: Zebra moves

Chapter 3: What I Have Done

Now onto the main part of this project, I have said that there are few options in playing chess variants, and chess variant engines are even more tacky and impractical. So, I took it into my own hands to make Tjatoer! into a real thing. I'm just going to flat-out say that I am in no way good at coding, I am even confident in saying I'm very bad at it! Still, I've found this video series tutorial on youtube by Eddie Sharick on how to make a chess game and chess engine in python. And I followed it through. To my horror, the video series ended abruptly, and I am left with only half of the things that could be done, this is why I need help from someone more knowledgeable to finish the project.

But first, let's go over what I have done so far. The diagrams in the next chapters will be interchanged, mostly I will use chess diagrams for examples because of their simplicity. In the project folder, you'll find 3 python files (ChessMain, ChesEngine, and SmartMoveFinder) and 3 subfolders (Sounds, Fonts, and Textures). To run the program, you'll need Python 3 of course, and Pygame installed. Once it is there simply run the ChessMain file.

Graphical Interface

Let's start with the actual GUI of the program, there aren't many intuitive options the user can make in without going into the code. This makes the program not very user-friendly.

Board and Pieces

The main window of the program is where everything happens. As you can see, Figure 3.1 is what you get immediately after you run the program, not very intuitive, but does the job.

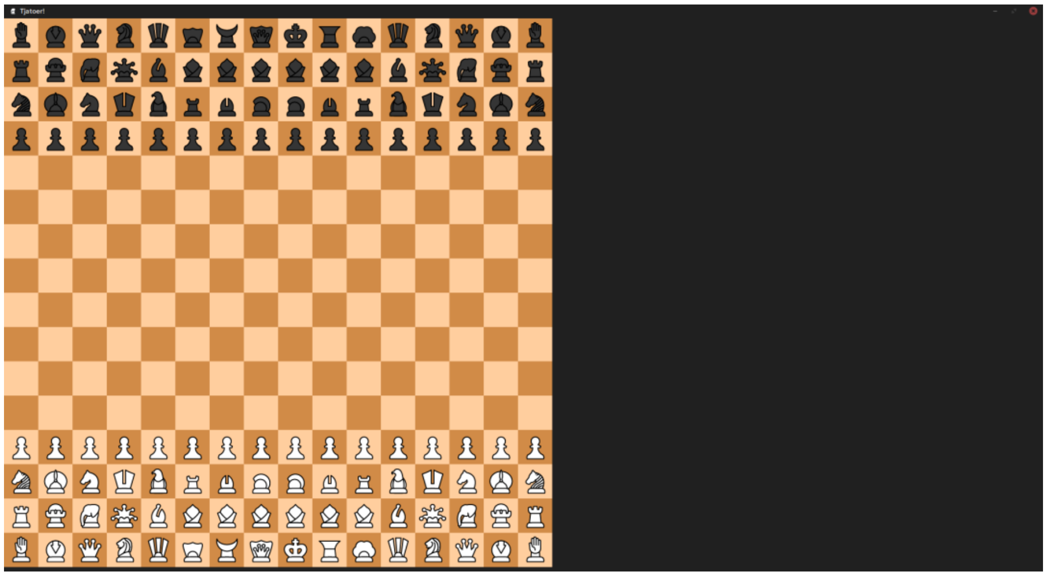
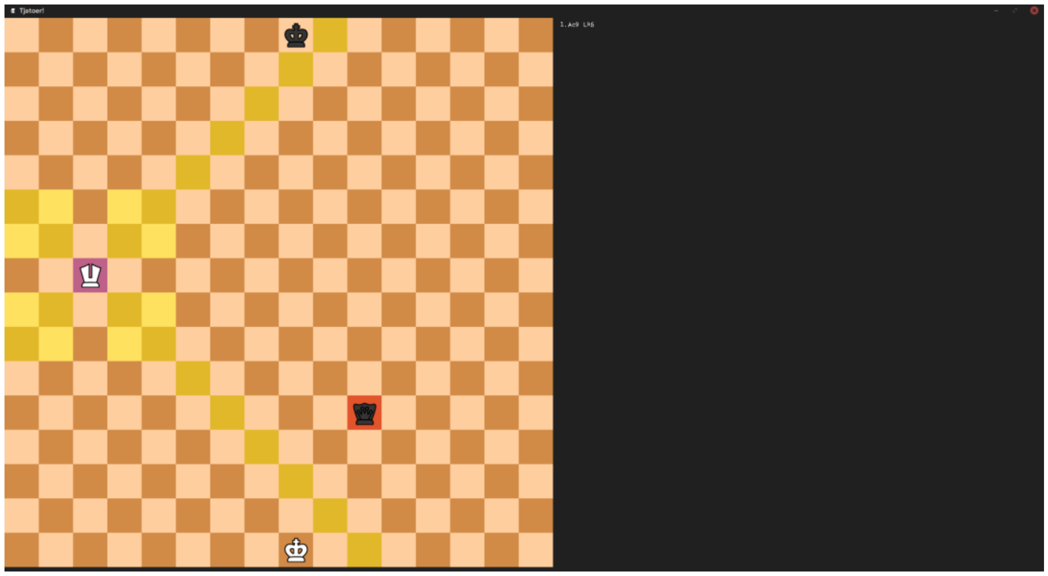


Figure 3.1: The main window

Valid Moves Highlight and Attached Move Log

As shown below, in Figure 3.2, the legal moves a piece can make are highlighted in yellow, the last pieced move by the opponent is highlighted in red and the selected piece is highlighted in purple. The huge black rectangle that is always attached to the board is used for recording moves, although it has some weaknesses.



```
1.Ac9 Lk6
```

Figure 3.2: The main window

As seen in Figure 3.3 below, the game is far from over, yet the move log is already full and can't display more moves. And since you can't scroll, subsequent moves simply cut off at the end of the window.

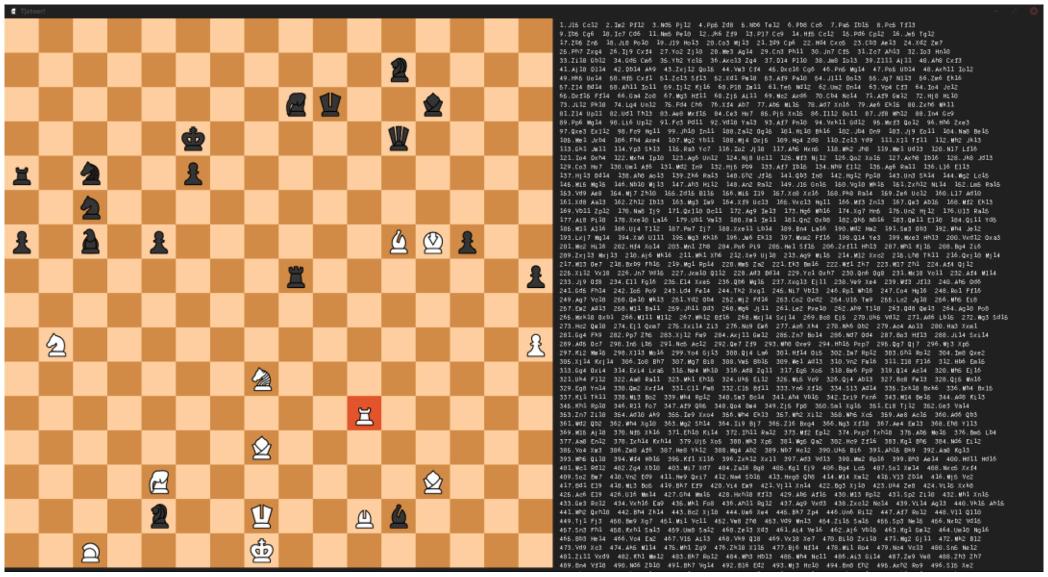
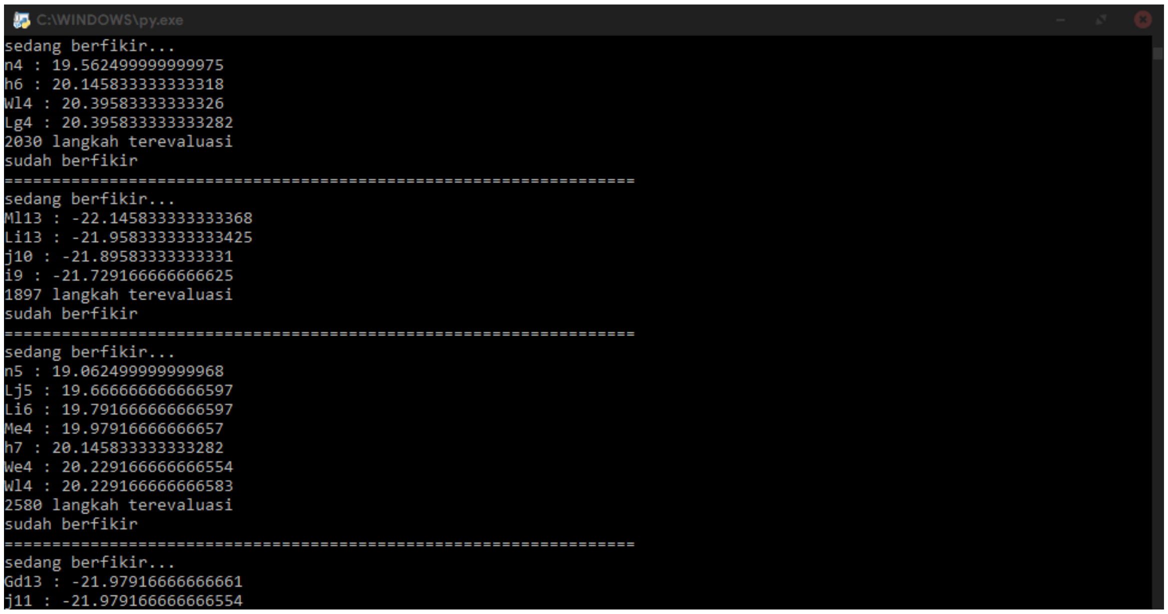


Figure 3.3: The main window

Since the move log is attached to the board and the window can't be resized nor scrolled through, the amount of moves that can be displayed is capped at 496 moves with the current settings.

Engine Evaluation (in the Console)

In figure 3.4 below is the simple display of the console, "sedang berfikir" (is thinking) followed by the best move evaluation, then "... langkah terevaluasi", (... moves evaluated), and finally sudah berfikir (done thinking). Again, the display is very simple and still very crude



```
sedang berfikir...
n4 : 19.562499999999975
h6 : 20.145833333333318
Wl4 : 20.39583333333326
Lg4 : 20.395833333333282
2030 langkah terevaluasi
sudah berfikir
==================================================================
sedang berfikir...
Ml13 : -22.145833333333368
Li13 : -21.958333333333425
j10 : -21.89583333333331
i9 : -21.729166666666625
1897 langkah terevaluasi
sudah berfikir
==================================================================
sedang berfikir...
n5 : 19.062499999999968
Lj5 : 19.666666666666597
Li6 : 19.791666666666597
Me4 : 19.97916666666657
h7 : 20.145833333333282
We4 : 20.229166666666554
Wl4 : 20.229166666666583
2580 langkah terevaluasi
sudah berfikir
==================================================================
sedang berfikir...
Gd13 : -21.97916666666661
j11 : -21.979166666666554
```

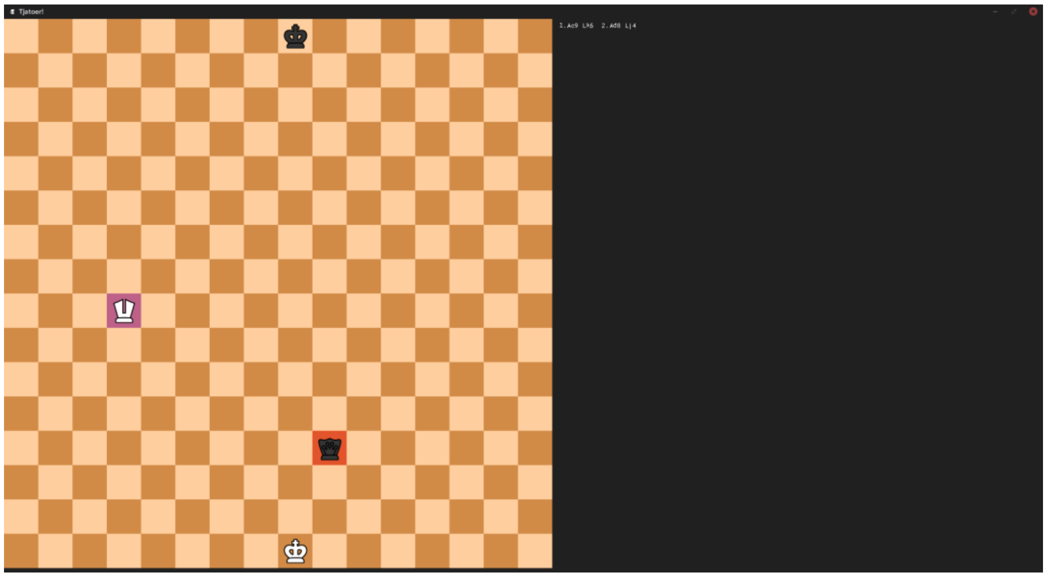
Figure 3.4: The program's console

Gameplay

Onto the actual mechanics of the game. Apart from the pieces and valid moves to play the game properly, I have implemented checks, checkmate, and stalemate, so you could say two humans can play with the program with no problem, but there are still loads of features I need help with to add. Most of the game mechanics are coded in the ChessEngine file. Little things are I also added an undo button (z) and a reset game button (r), I also added sounds for the start of the game, moves, captures, and game overs (win/lose/draw). again, these are very crude and could be improved.

Check, Checkmate and Stalemate

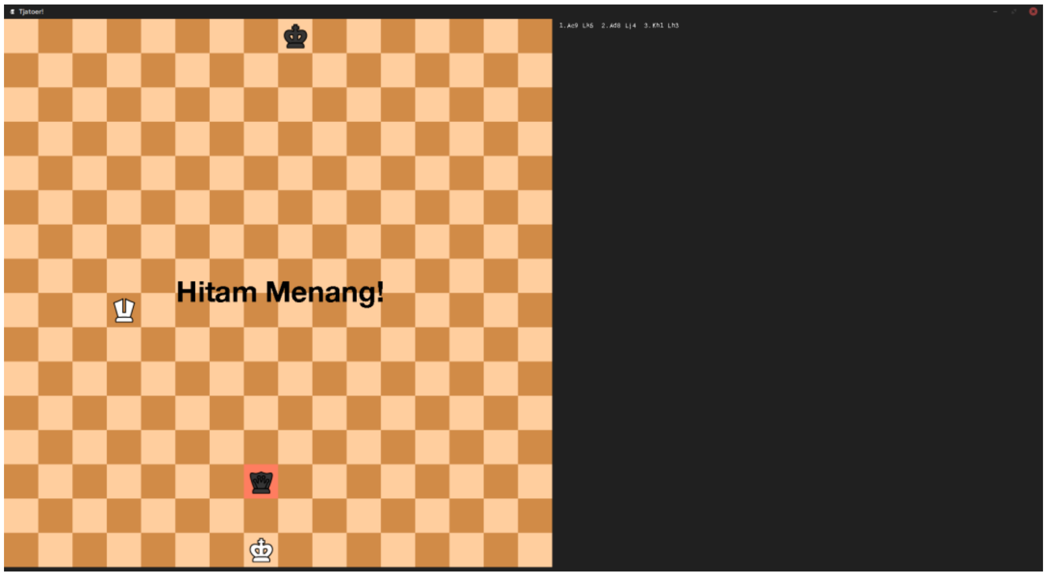
As shown below, in Figure 3.5, the selected archbishop can't prevent the king from being in check, thus it has no valid moves. the program checks for checks by silently evaluating all valid moves, then generating all valid opponent moves, if that opponent move attacks the king, then the first move is not valid thus the king is in check, this method is of course very ineffective and needs to be improved.



```
1.Ac9 Lh5  2.Ad8 Lj4
```

Figure 3.5: The main window

As shown below, in Figure 3.6, that is the basic checkmate text, “Hitam Menang!” (Black Wins!). For stalemates, the text would simply say “Remis” (Draw). Stalemating is currently the only way to draw and checkmating is currently the only way to win.



```
1.Ac9 Lh5  2.Ad8 Lj4  3.Kh1 Lh3
```

Hitam Menang!

Figure 3.6: The main window

Engine

Now we delve into the heart of the project, the engine. The engine is most crucial since this program has no online multiplayer capabilities, so if you don’t have another person in the room playing with you, you’ll be stuck recreating Pixar’s Gery’s Game by playing by yourself. The problem is, the engine currently is very abhorrently weak and inefficient, depth 2 (I will talk about depths later) takes a couple of minutes just to make a move. there are ways to make it reasonably smarter, but frankly, I don't have the necessary coding skills to do so (this can also be said regarding all aspects of this project). The file responsible to control the engine is the SmartMoveFinder file.

Before we get into what little I have done with this slow engine, let me explain aspects of a chess engine. Strong regular chess engines like Stockfish and Alpha-Zero and a bunch more chess engines in general use a communication protocol called UCI Universal Chess Interface, developers use UCI as a way to hook up any engine to any GUI. Since Tjatoer! is considered a fairy-chess variant (chess variant with pieces outside of regular chess) we can't use regular Stockfish as its engine. There are some fairy chess engines like Fairy-Max or Sjaak II but each of those doesn't use UCI and has limitations like piece types and board sizes. Although, if you can somehow implement up one of these fairy engines to Tjatoer! that will astronomically ease your job of course.

Now, regarding this engine I've lazily built, the main point of it is that it evaluates the game based on scores, draws are valued as score 0, and of course, winning is the highest score a player can get, with checkmate (and hopefully other ways to win) being 1000 points for white and -1000 points for black, in other words, white is trying to get the score as high as possible, while black is doing the opposite, trying to get the score as low as possible. Aside from the basic win-lose-draw scores, more aspects contribute to the game score that I will explain below. The depth of the engine determines how many moves ahead the engine evaluates the game, since a chess move is made when both players make a move, depth 2 for example is only 1 chess move ahead, depth 6 in 3 chess moves ahead, and so on. The game of course starts with the score set to 0.

Piece Relative Value (Material Balance)

Logic dictates that if a player has more pieces than the opponent, they are more likely to win (obviously for example if black only has a king and white still has their king and queen, white would win easily with the right technique).

Notation	P	B	N	R	Q	K
Value	1	3	3	5	9	∞

Table 3.1: Regular chess piece values

How many pieces a player has is called their material, for example, if we use the regular chess piece values, if white captures one of black's rook with their bishop, they are up 5 points of material than black. Moreover, if black recaptures white's bishop, white is still up 2 points of material than black since a bishop is only worth 3 points while a rook is worth 5. I mentioned that I have assigned specific points to each piece in Tjatoer! so the engine can better evaluate which player is more likely to win. (Piece notation is explained in chapters 1 and 2, the value of the king is 0 in the SmartMoveFinder file since it cannot be taken off the board obviously). Below are the piece values in Tjatoer!

Notation	A	B	C	D	E	F	G	H	I	J	K	L	M
Value	825	325	875	900	1075	1250	250	450	925	775	∞	1500	400
Notation	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Value	300	1200	100	950	500	150	150	175	840	165	610	175	315

Table 3.2: Tjatoer! piece values

These values are of course not final, I loosely assigned the values based on alpha-zero's valuing system for the regular chess pieces and added the values for the pieces with move combinations of other basic pieces. So, if you can assign better values based on better evaluations be my guest.

Positional Scoring

Apart from the value of the piece itself, the position of the piece matters too. For example, a knight on the corner of the board with only 2 squares to go to is worth reasonably less than a knight on the center of the board with all 8 squares available to go to, bishops are more powerful on long diagonals and as such should worth more if they are on long diagonals, rooks are worth more on open files and ranks as oppose to crowded, closed ones, pawns will try to get as far up the board to promote, etc.

As of right now, the only 4 positional scoring system is in Tjatoer!

- Skorpion (pawn score)
  Divided into bP and wP (bP is just the reverse of wP as pawns are dependent on direction) used for the pawn and man, the further they go up the board (to promote) and the closer they are to the middle of the board, their score will become higher.
- Skorkuda (knight score)
  Used for the knight, man, gajah, hawk, upasaka, warrior, yishi, zebra, archbishop, jester, imam, chariot, ferz, oknha, duchess, esquire, and lance, all these pieces will try to get to the center of the board as it is there that the score is highest
- Skorgajah (bishop score, not to get confused, gajah is the word for bishop in indonesian)
  Used for bishop, xiphodon, thalia, queen, archbishop, jester, duchess, ferz, oknha, and lance, all these pieces will try to get on the largest diagonal of the board as it is there that the score is highest
- Skorbenteng (rook score)

Divided into bR and wR (bR is just the reverse of wR), used for the rook, viscount, shrook, queen, imam, chariot, ferz, oknha, esquire, and lance, these pieces will try to get to the opponent's pawn rank (4th rank of each side, this is why it is dependent on which side the piece belongs to), as it is there that the score is highest

All these scores are still very simple, if you have any improvements, that will be appreciated. With all that being said, if the pieces use more than one positional valuing, like for example the archbishop that uses both skorkuda and skorgajah, then it will proportionally divide between the two of them (half of the positional score will come from skorkuda, and the other half from skorgajah). The engine evaluates the true value of the piece with the formula **Relative Value*0.01 + Positional Value*0.25 = True Value.** The engine then adds all the pieces' true values each time the board changes (when one of the players makes a move or during evaluation) and determines which player is at an advantage.

Negamax Alpha-Beta Pruning

I am going, to be honest, I don't fully grasp this concept yet, so here is a crappy image I got from the chess programming wiki (which may or may not be a helpful source with the improvement of this project). But, from what I can gather, basically if the engine evaluates a branch of moves with the initial move giving an evaluation of 6 for example, any moves that give an evaluation below 6 after that initial move will not be considered (pruned).

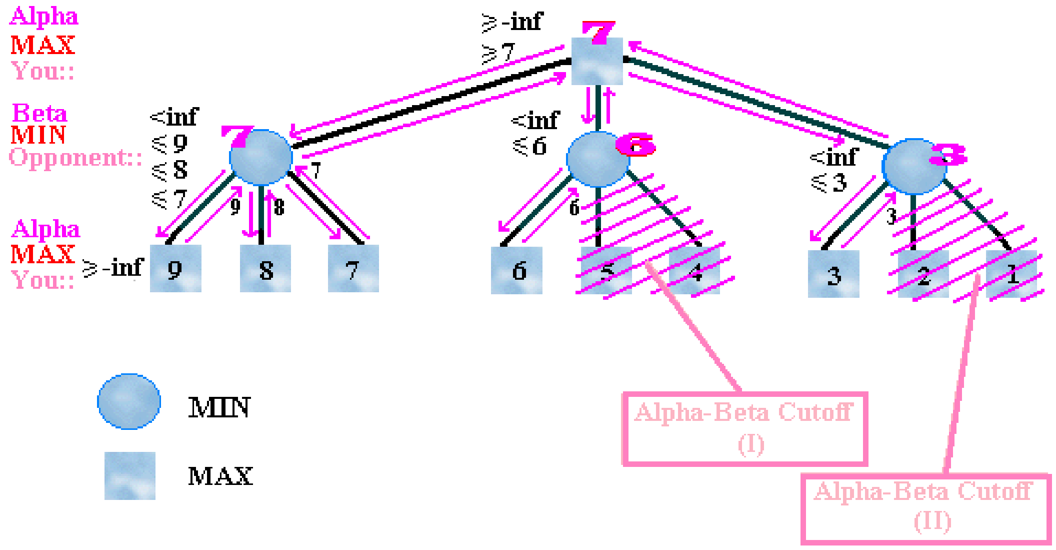


Figure 3.7: Alpha-Beta pruning in a nutshell (maybe

Chapter 4: What Needs to be Implemented

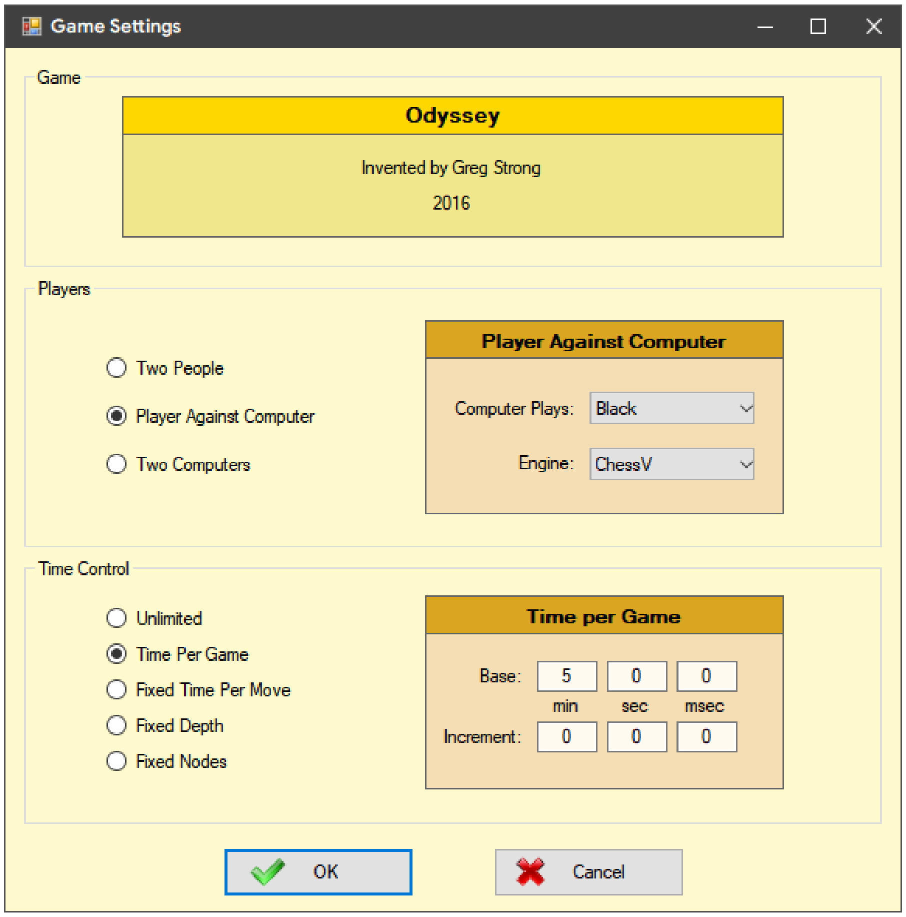
This chapter will be an exposition to everything that I think needs to be implemented before I could call this project complete. Moreover, this chapter would include examples I took from existing chess programs and chess websites as a reference to design and mechanics.

Graphical Interface

Onto the features that I lack the skills to implement, we will start with some GUI improvements. You don't have to match these perfectly, but you can use these as a reference. I will of course provide images to better convey my thoughts. All resources that could be implemented are provided in the project folder and their location will be mentioned as I go.

Start Menu

As seen below in Figure 4.1, this start menu is taken from the chess program ChessV. The start menu is very simple yet intuitive, first, it has a proper title of the game, then it has three options for game modes (human vs human, human vs engine, and engine vs engine), and finally, it has time control options.



Game

Odyssey

Invented by Greg Strong

2016

Players

( ) Two People [✓ Player Against Computer] ( ) Two Computers

Player Against Computer

Computer Plays: Black

Engine: ChessV

Time Control

( ) Unlimited [✓ Time Per Game] ( ) Fixed Time Per Move ( ) Fixed Depth ( ) Fixed Nodes

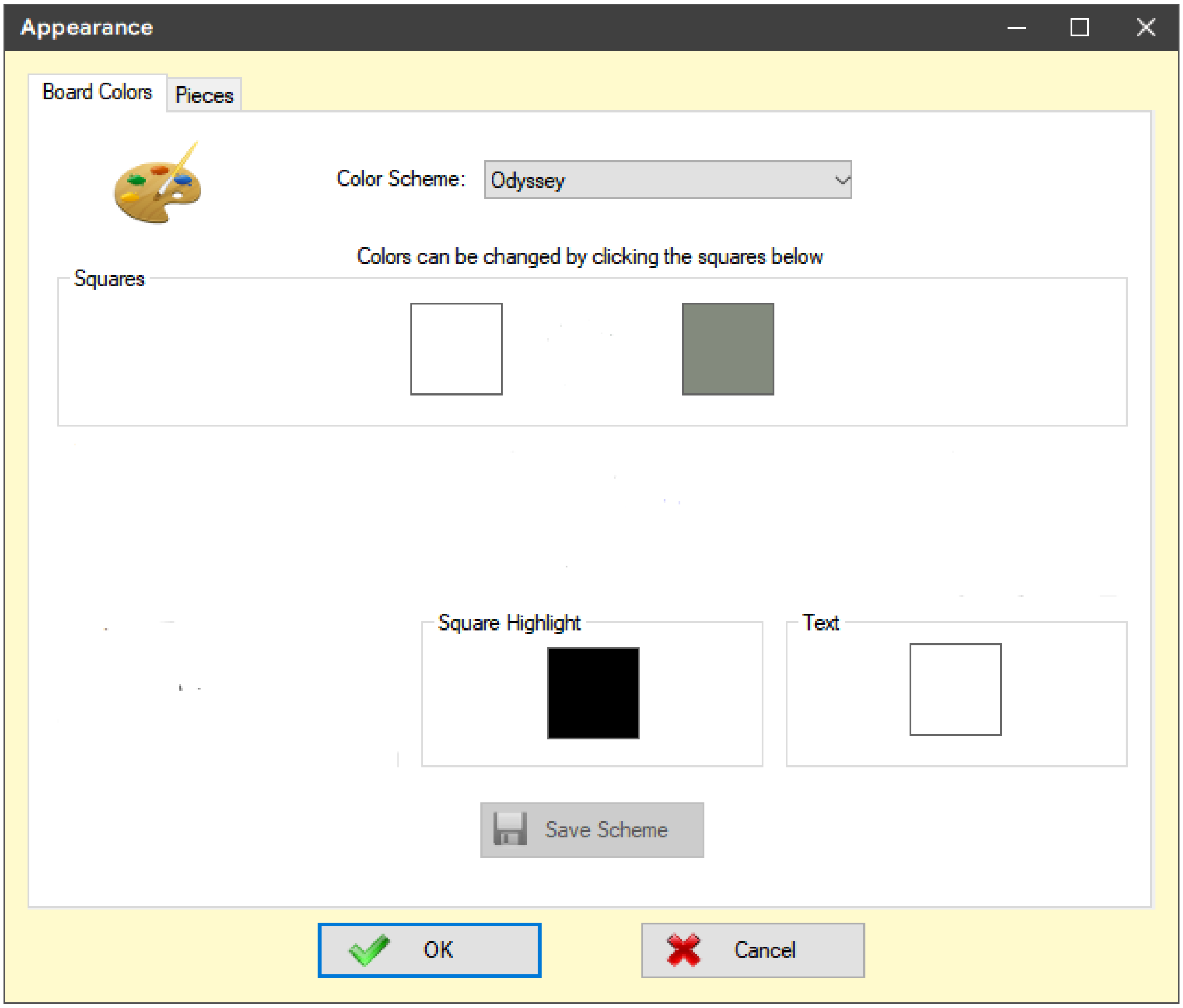
Time per Game

	min	sec	msec
Base:	5	0	0
Increment:	0	0	0

Figure 4.1: Start menu (taken from ChessV chess program)

Options and Help Window

Again, with ChessV as an example, in Figure 4.2 below the user can choose the square colors, square highlight color (though here there is only 1 square highlight color whereas in Tjatoer! there are 3, purple for selected pieces, red for the last piece moved, and yellow for valid moves), and the text color.

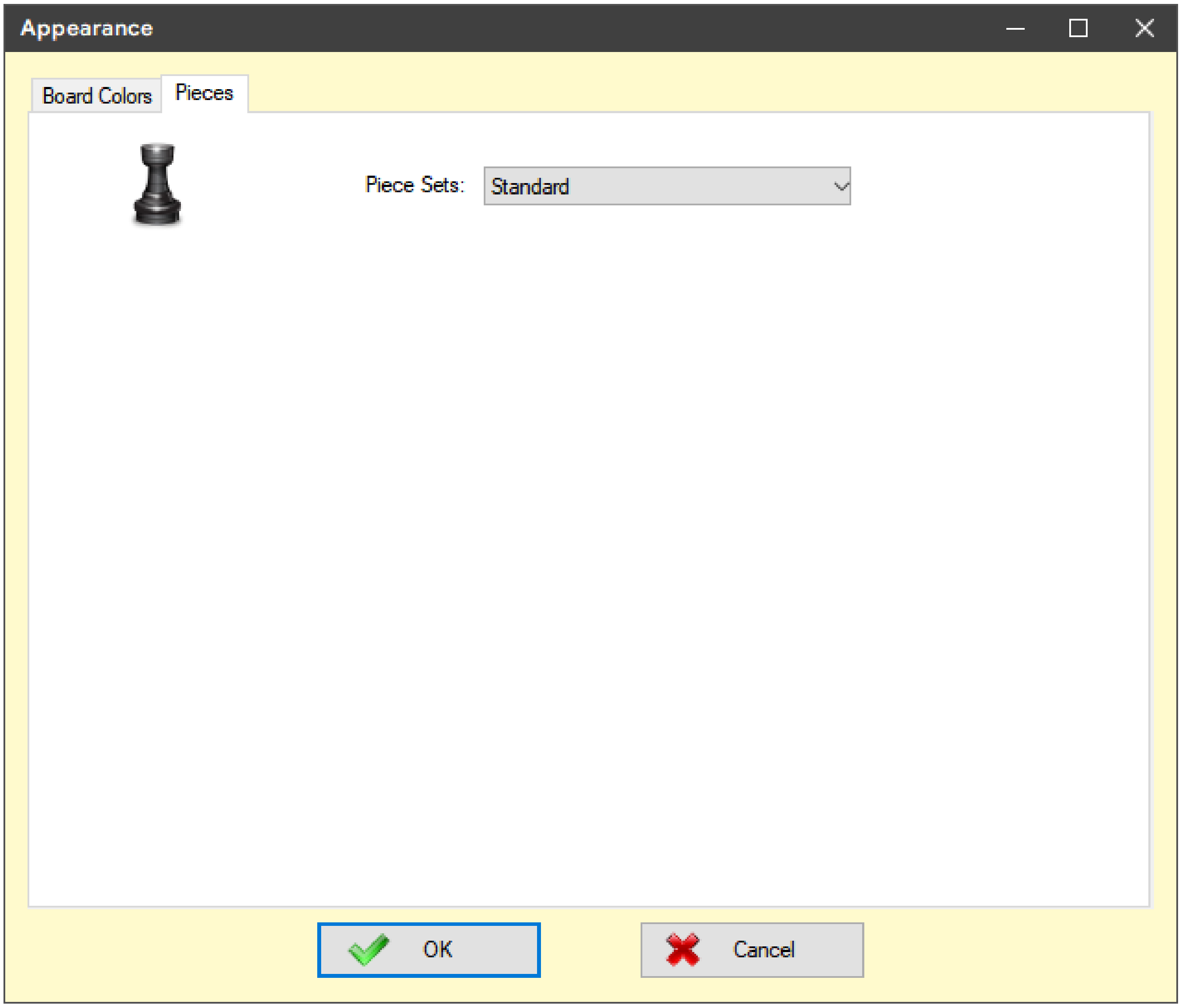


Color Scheme: Odyssey

Colors can be changed by clicking the squares below

Figure 4.2: the first page of ChessV's appearance options window

As seen below in figure 4.3 the user can choose what piece sets that will be displayed (currently as I have mentioned Tjatoer! only two sets of pieces, but there is certainly more to come). But, this second page in my opinion is a waste of space and can be put in the blank space of the first page, still, I wanted to give an example of how changing piece sets can be implemented as an option in the program.



Piece Sets: Standard

Figure 4.3: the second page of the ChessV's appearance options window

Shown below in Figure 4.4 is a window that pops up whenever the user right-clicks on a piece. It is helpful as it shows the piece's name, how the piece moves, and its value. An added bonus is its board visibility, this is how many squares the piece can possibly get to, for example, a bishop is bound to one color this means it can only go to 32 of the 64 squares in an 8x8 board.

As for how the pieces move, the diagrams that can be used are located in the folder of the project, specifically in the Tjatoer!/Textures/Piece Move Diagrams folder. Though since the example shows an 8x8 board and the diagrams I will provide will show a 16x16 board, I suggest it is best to enlarge the pictures so that the user can see the diagram better. And for piece icons, you can use the same images used for pieces in the program itself

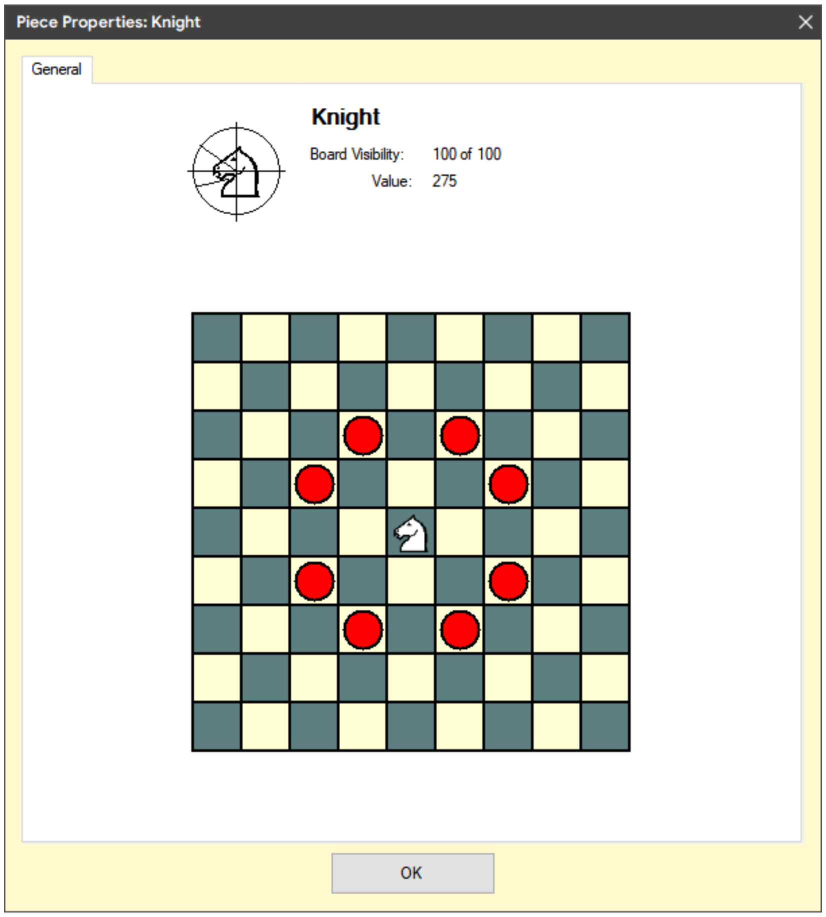


Figure 4.4: Help window in ChessV

To access these windows and options, I would like to give an example of [WinBoard](), another fairy chess program. But in Winboard I want to focus on the implementation of a menu bar above the board to hold the options, I think this is a great way to position the options so that the user can get to it without much of a hassle, not every menu and submenu I think should be on the final product but I think the layout of the whole thing eases the user in navigating the program. Furthermore, I will provide examples of each individual menu and its submenus and provide thoughts on which menus that I think could be implemented. To start the example, see figure 4.5 below.

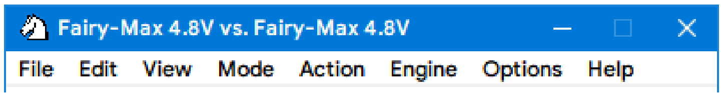

Fairy-Max 4.8V vs. Fairy-Max 4.8V
File Edit View Mode Action Engine Options Help

Figure 4.5: Winboard's main window title bar along with the menus

Keep in mind the keyboard shortcuts that are shown on the right of the options I'm about to discuss. Let's start with the help menu, as shown below in Figure 4.6, there is not much I can say about this menu since I think the method used to open the help window in

ChessV is a better implementation than in Winboard, but still, you can always implement more ways to access a feature in a program. All the “Help Content”, “Help Index”, and “How to use Help” can be put under one help category. As for the about option “About Winboard” doesn’t need to be implemented.

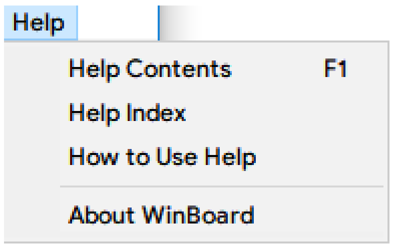


Figure 4.6: Winboard's help menu

Now we get into the file menu, I think one of the most important options to implement here is the “New Game” feature as it is integral to the program, it is already implemented as the reset feature. The “Save as Diagram” option would be great so the user can save the current state of the board as a picture for example. The “Save Game” and “Save Position” options could be useful as well to save the game in PGN or FEN format (I will not explain into these, look it up). Other options like “New Shuffle Game” and “New Variant” along with the other disabled, greyed options out are irrelevant since Tjatoer! only has one variant. And lastly, another most important option to implement here is the “Quit” option for obvious reasons.

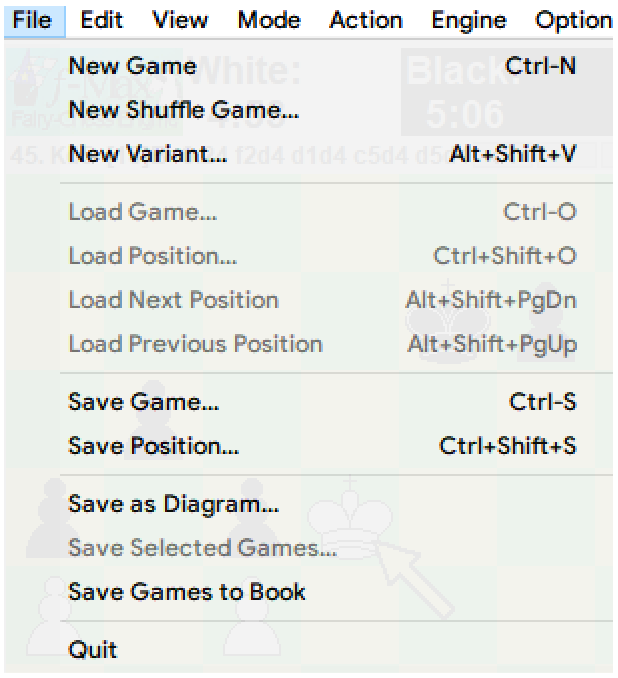


Figure 4.7: Winboard's file menu

After that, we have the engine menu in Figure 4.8, this menu is there because winboard has the capability to play with multiple fairy chess engines, but since Tjatoer will only have 1 dedicated engine, the engine menu doesn't have to be implemented.

Moving on, we have the view menu in Figure 4.9, this is the most crucial menu to implement as it is full of features that I want to be implemented. First, we have “Flip View” or the board flipper, I will explain this feature later in this document. “Swap Clocks” not really important because even though I’m planning to implement time controls in this program, swapping clocks is not as important in my opinion. Next, we have “Engine Output” and “Move History” and “Evaluation Graph, these will show up as a window and will be discussed more thoroughly in the document. Then, “Tags”, “Comments”, and “Game List Tags” and all the greyed-out options are not features that are important enough for me to implement. “Colors”, “Board Themes” and “Fonts” on the other hand are important features, though I think putting it into one appearance options window as we have already discussed is a good way to be more efficient. To add to that, I think it's best to put the button to open the appearance options window in the actual options menu that will be discussed later instead of here in the view menu.

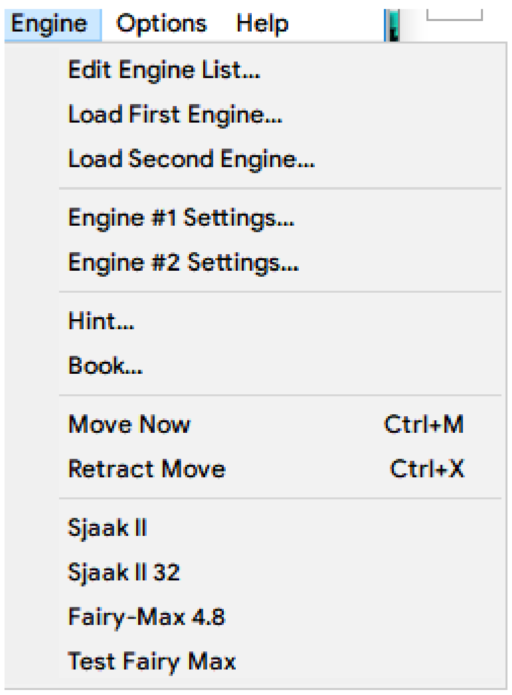


Figure 4.8: Winboard's engine menu

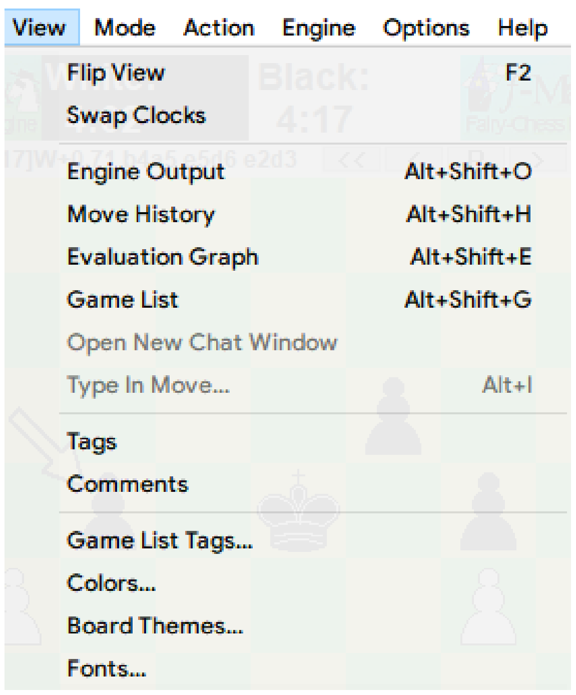


Figure 4.9: Winboard's view menu

Next, we have the edit menu in Figure 4.10, not much that is important here as well. All the “Copy...”, “Paste...”, “Edit...” all the other greyed-out options are only important if the program has a PGN and/or FEN reader function. Few things to note are the “Truncate Game” option there so the user can quickly stop the game if two engines are playing to save it as a diagram for example. And the “Backward”, and “Forward” options are also important as they are the ways of implementing the undo and redo (another feature that should be implemented) features. And lastly, The “Back to Start” and the “Forward to End” feature are important to implement so the user won't have to spam redo or undo when they want to review a game at the end of a game. Although the undo, redo, back to start, and forward to end could be located in the action menu which we will discuss later, instead of here in the edit menu. So in other words, to reduce clutter, the edit menu does not have to be implemented

As for the action menu in Figure 4.11, the only option I think are necessary to implement is the “Draw” and “Resign” option, draw immediately ends the game in a draw of course and resign ends the game and declares the one who resigns as a loser. “Call Flag” and “Abort” and other greyed-out options do not need to be implemented because, for example, the call flag is a trivial option used in time control between two players and abort is simply a way to end the game before it even starts. The action menu is also where the undo, redo, back to start, and forward to end option could be put.

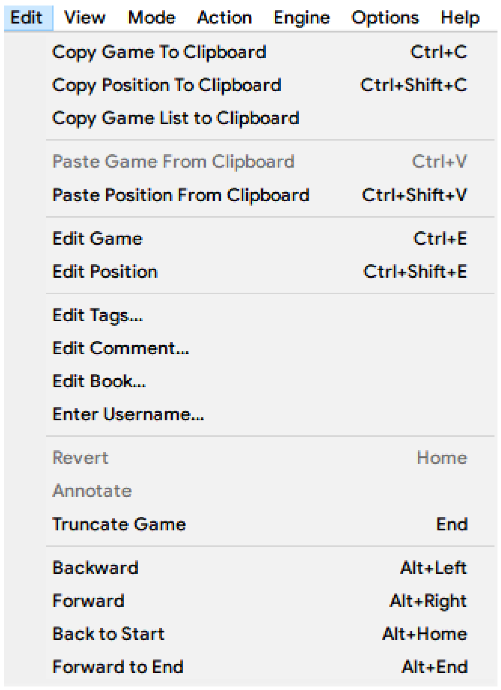

Edit View Mode Action Engine Options Help
Copy Game To Clipboard Ctrl+C
Copy Position To Clipboard Ctrl+Shift+C
Copy Game List to Clipboard
Paste Game From Clipboard Ctrl+V
Paste Position From Clipboard Ctrl+Shift+V
Edit Game Ctrl+E
Edit Position Ctrl+Shift+E
Edit Tags...
Edit Comment...
Edit Book...
Enter Username...
Revert Home
Annotate
Truncate Game End
Backward Alt+Left
Forward Alt+Right
Back to Start Alt+Home
Forward to End Alt+End

Figure 4.10: Winboard's edit menu

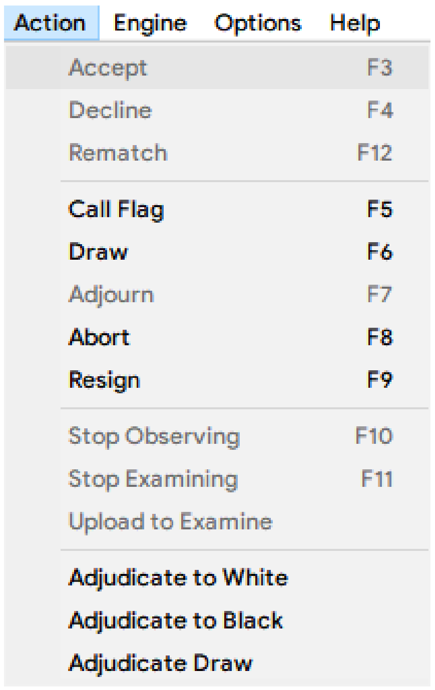


Figure 4.11: Winboard's action menu

Next, we have the mode menu in Figure 4.12, this is another important menu as it gives the user the option to change the game mode while in the program without needing to go back to the main menu. Not all of the options need to be implemented though, in my opinion only the “Machine White” (when the engine plays white), “Machine Black” (when the engine plays black), and “Pause” feature. The “Two Machines” feature is not important as the user can just enable both machine white and machine black to play the engine vs engine mode. The enabling of these options shouldn’t reset the game, any of these options should be able to be selected at any point of the game, for example when playing manually (human vs human) up to a certain position, then the machine white option is enabled, the engine will continue playing white in the position, and the same principle will happen with black and if both are enabled, the engine will play both white and black.

And lastly, we have the options menu in Figure 4.13. the only options that I think are important to implement here are the “Mute all Sounds” option, of course, to mute all the game sounds, and the “Time Control” to configure the time controls, again, these two options should be able to be configured at any point of the game without having to reset the game. The “Sounds” option is not important as Tjatoer! only has one set of sounds. And finally, the “General” option should be the button to open the appearance options window.

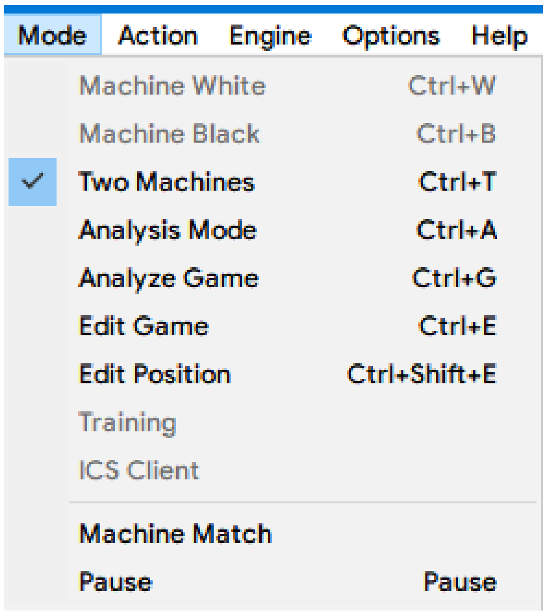


Figure 4.12: Winboard's mode menu

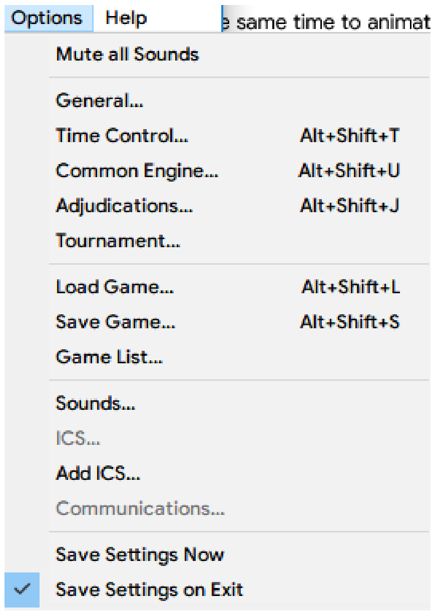


Figure 4.13: Winboard's options menu

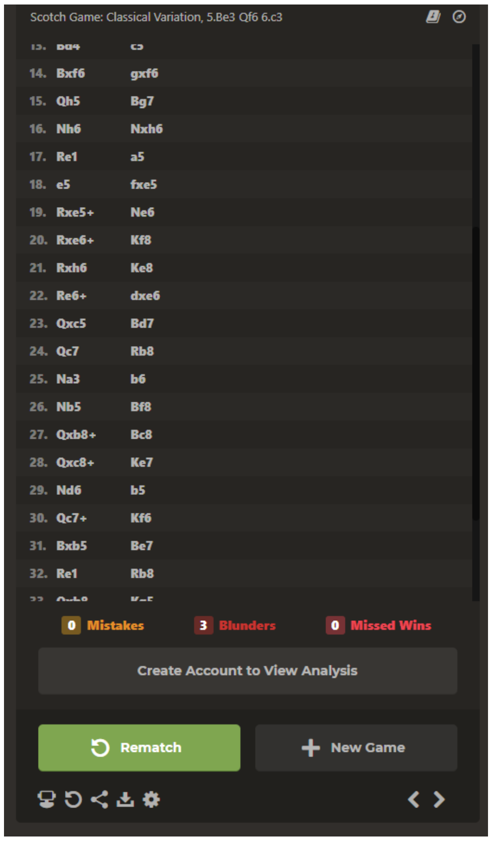
Animation

The video series tutorial on which I based this project does touch the subject of animation, but it is very crude, and I would like to implement it differently since in the series the animation has a set time duration and this made the animation a bit not proportional. What I mean by this is for example a pawn that moves 1 square forward and a queen that moves 8 squares across a rank will take the same time to animate, thus the pawn will seem to move much slower since it has to cover less distance, and on the other hand the queen will animate faster as it has to move 8 squares in the same amount of time.

A good reference on how to implement animation would be chess.com's piece animation. There the pieces seem to move at the same velocity no matter how far or near their destination is and that is the animation I would want to implement. To add to that, the implementation of dragging pieces to move them negates the need for animation. so, to sum up, the animation should be proportional and only play when either you make the move by double-clicking (the current way of moving the pieces) or when it's moved by an engine, thus the animation won't play when the user moves the pieces by dragging it.

Separated Move Log

As seen below in Figure 4.14, to the right we can see chess.com's move display log. It displays one move per line and can be scrolled indefinitely. Each pair of moves is perfectly aligned with the ones below and above it with the first one being white's move and the second one being black's move of course. But while I think displaying one move per is adequately good, maybe the implementation can display two or maybe three moves (that are still aligned with the ones above and below it) per line since as we have discussed, a game of Tjatoer! can last hundreds of moves. Finally, the current program doesn't have the notation for checks and checkmates yet, it also can't differentiate when two pieces can go to the same square, so those three should be implemented.



Scotch Game: Classical Variation, 5.Be3 Qf6 6.c3

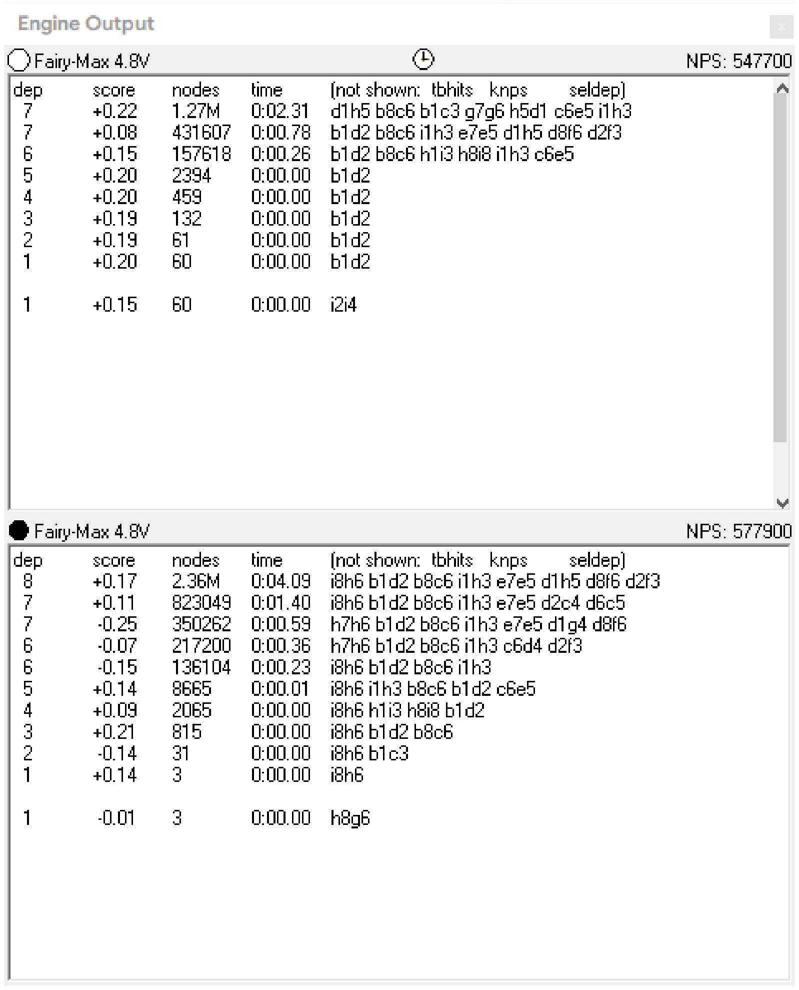
14.	Bxf6	gxf6
15.	Qh5	Bg7
16.	Nh6	Nxh6
17.	Re1	a5
18.	e5	fxe5
19.	Rxe5+	Ne6
20.	Rxe6+	Kf8
21.	Rxh6	Ke8
22.	Re6+	dxe6
23.	Qxc5	Bd7
24.	Qc7	Rb8
25.	Na3	b6
26.	Nb5	Bf8
27.	Qxb8+	Bc8
28.	Qxc8+	Ke7
29.	Nd6	b5
30.	Qc7+	Kf6
31.	Bxb5	Be7
32.	Re1	Rb8

0 Mistakes 3 Blunders 0 Missed Wins

Figure 4.14: Chess.com's move log display

Engine Evaluation Display

As shown below in Figure 4.15, the Winboard program has a proper engine evaluation display. It shows the depth of the evaluation, and the score result of the evaluation, the time elapsed to make the evaluation for each depth (though on depth 10 the engine can make the evaluation instantly), and the nodes (amount of moves evaluated).



```
○ Fairy-Max 4.8V                                              NPS: 547700
dep    score    nodes    time      (not shown:  tbhits   knps      seldep)
 7     +0.22    1.27M    0:02.31   d1h5 b8c6 b1c3 g7g6 h5d1 c6e5 i1h3
 7     +0.08    431607   0:00.78   b1d2 b8c6 i1h3 e7e5 d1h5 d8f6 d2f3
 6     +0.15    157618   0:00.26   b1d2 b8c6 h1i3 h8i8 i1h3 c6e5
 5     +0.20    2394     0:00.00   b1d2
 4     +0.20    459      0:00.00   b1d2
 3     +0.19    132      0:00.00   b1d2
 2     +0.19    61       0:00.00   b1d2
 1     +0.20    60       0:00.00   b1d2

 1     +0.15    60       0:00.00   i2i4

● Fairy-Max 4.8V                                              NPS: 577900
dep    score    nodes    time      (not shown:  tbhits   knps      seldep)
 8     +0.17    2.36M    0:04.09   i8h6 b1d2 b8c6 i1h3 e7e5 d1h5 d8f6 d2f3
 7     +0.11    823049   0:01.40   i8h6 b1d2 b8c6 i1h3 e7e5 d2c4 d6c5
 7      -0.25   350262   0:00.59   h7h6 b1d2 b8c6 i1h3 e7e5 d1g4 d8f6
 6      -0.07   217200   0:00.36   h7h6 b1d2 b8c6 i1h3 c6d4 d2f3
 6      -0.15   136104   0:00.23   i8h6 b1d2 b8c6 i1h3
 5     +0.14    8665     0:00.01   i8h6 i1h3 b8c6 b1d2 c6e5
 4     +0.09    2065     0:00.00   i8h6 h1i3 h8i8 b1d2
 3     +0.21    815      0:00.00   i8h6 b1d2 b8c6
 2      -0.14   31       0:00.00   i8h6 b1c3
 1     +0.14    3        0:00.00   i8h6

 1      -0.01   3        0:00.00   h8g6
```

Figure 4.15: Engine evaluation display in WinBoard

This is not important nor mandatory to implement but it is worth noticing that below in Figure 4.16 you can also see the evaluation history, its hard to see on the graph but what it does is makes bar graphs for each evaluation score for each move that is played in the game, if the bar goes down, that means black is at an advantage, if the bar goes up, then that means white is at an advantage, the evaluation bar after black's move is shown darker and the bar after white's move is shown as lighter. I advise you to try Winboard for yourself to better understand this.

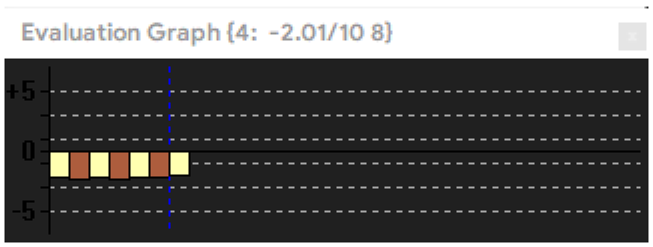


Figure 4.16: Evaluation graph in Winboard

Board Flipper

Below in Figure 4.17 is chess.com's analysis page, The board flipper is probably the easiest feature to explain and hopefully easier to implement, it basically flips the board so that the user can see and play from black's perspective. I have mentioned that this option can be put inside one of the menus, but below, you can see that the options button and the flip board button can be small and unnoticeable but still available for the user. Giving the users multiple options to access a feature wouldn't hurt.

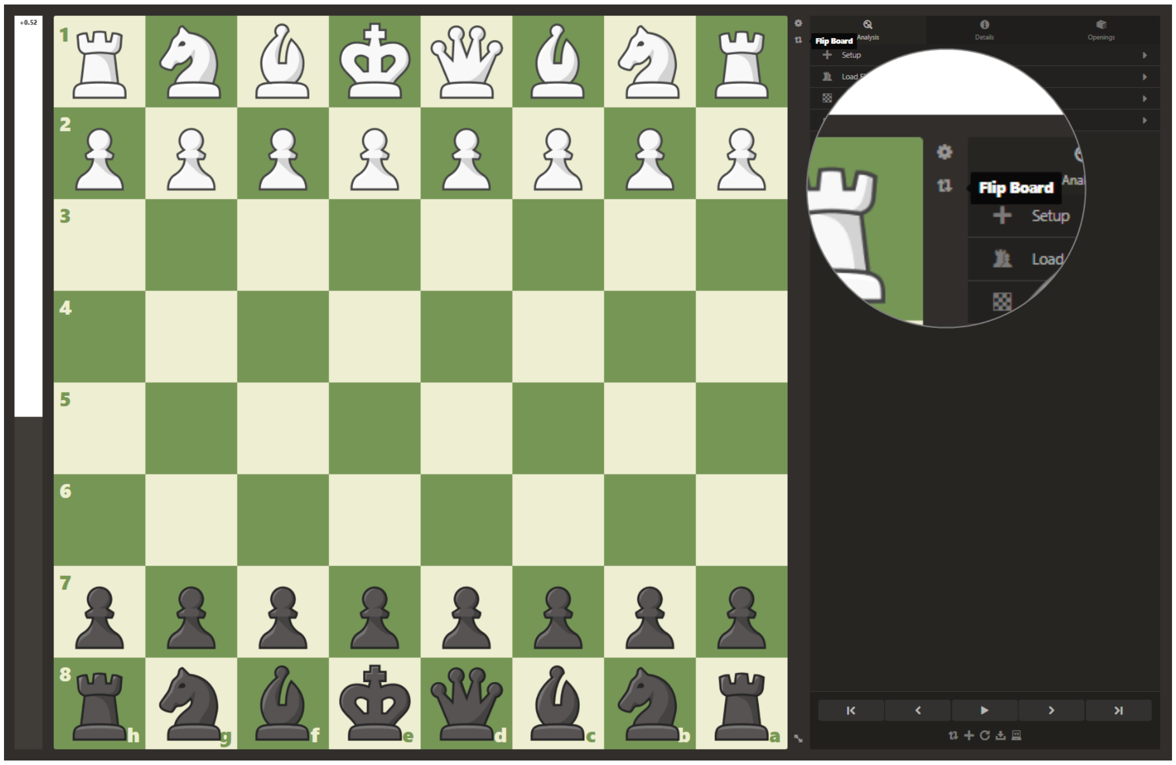


Figure 4.17: Chess.com's options and flip board button

Gameplay

Now we get into the actual features, rules, and mechanics of the game. Regarding this topic, here are some of my ideas of how to implement these features, again, I will leave it up to you to find a way to implement these features and make them more efficient, but I hope my explanation can give a rough idea of how these features can be implemented.

More Ways to Draw and Win

Let's start with fivefold repetition as a way to draw the game, in order to implement this rule, I guess there must be a way to store the previous board states so that the program can keep track of repeated positions since the repeated position is fivefold repetition does not have to happen back-to-back. One way to reduce the data usage is to empty the stored board states whenever a piece is captured or whenever a pawn moves since previous positions can't occur when there are one less piece on the board and pawns can't move backward.

Next is the 100-move draw, the rule states that the game can be considered a draw when 100 moves have been played after the last capture or pawn move. One way to implement this is to, of course, keep track of the last pawn move or capture. the other way is to have a countdown of 100 moves that of course resets whenever a piece gets captured or a pawn is moved.

As for baring the king, the way to implement this I can think of is that the program must check if one side only has their king left, if that is true, then the game can conclude that the other side wins.

A draw can also occur when there is insufficient material on the board, for example, if the game consists of a knight and a king vs a king, the program should just end the game in a draw since there is no way of checkmating king with just a knight. Regarding Tjatoer! this is irrelevant since leaving a sole king is considered a win.

Drag and Drop to Move Pieces

Currently, the only way to move the pieces is to select the piece, then click on one of the valid destination squares of the piece, this double-click method works fine as it is but I think it would be more pleasing to the eyes of the user can simply drag their piece to their destination square and drop it there. The highlight of the valid destination squares will still be shown and if the player tries to drag and drop the piece to an invalid square or outside the board, the piece will just snap back to its original square. Again, chess.com's drag and drop mechanic to move the pieces is a good reference to refer to.

Pre-moving and Time Control

As shown below in Figure 4.18, pre-moving is when you make your move before the opponent has taken their turn. while it's the opponent's turn you can make a move to automatically be played as soon as they make their move. To do this, simply move the piece as if it were your turn. The move is being highlighted in red in the example.

In the picture, the move black just made, d5, is highlighted in yellow. Black then made a pre-move before white moved, which is highlighted in red, Nc6. The knight move is not yet on the board, but will play automatically the instant white moves, no matter what white does! If white takes the pawn, for example, with exd5, Nc6 would be a very bad move. If the opponent's move makes the pre-move illegal (such as putting the king in check), then the pre-move will be canceled, and you will have to think of something else. To undo a pre-move, simply right-click or tap on the pre-moved piece.

As for time controls, it's very simple, it's just a countdown clock of say for example 10 minutes, if a player's clock runs down to 0, then that player loses. Black's clock will start counting down exactly after white's first move, then after black moves, their clock will pause and white's clock will then start to count down, and when white makes their second move black's clock will start counting down again and so on.

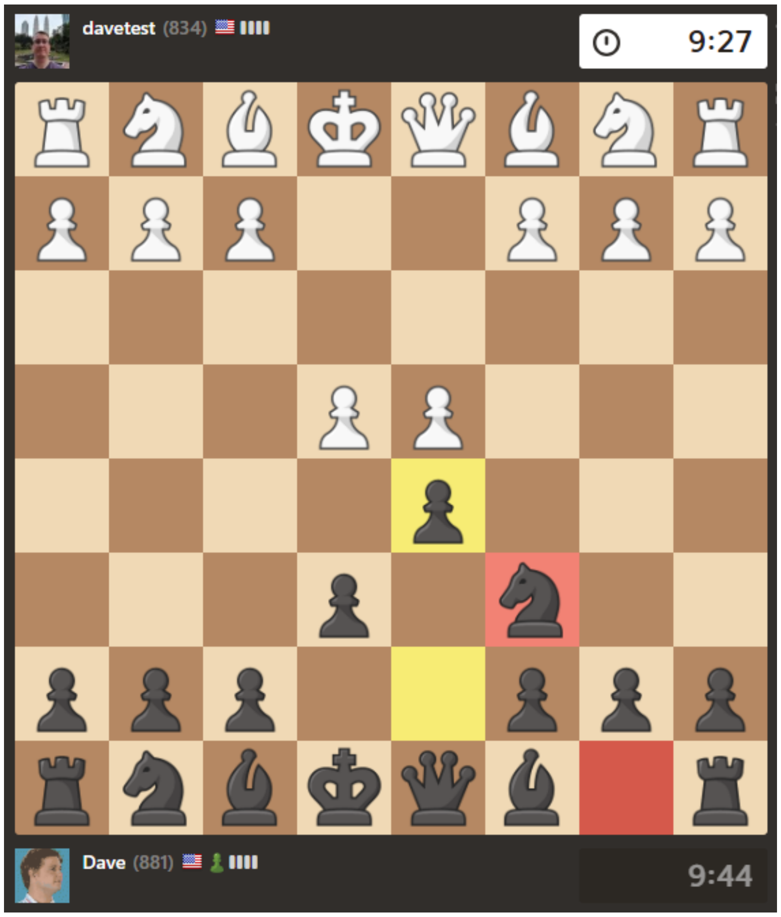


Figure 4.18: Chess.com's pre-move and time control feature

Pawn and Man Promotion Options

As you can see below in Figure 4.19, that is the chess.com pawn promotion mechanic, the black king has just moved to c6 and its time for the white pawn to promote, as soon as the pawn reaches the d8 square, an option appears with the icon of the pieces the pawn can promote to, the user then clicks the piece that they want to promote their pawn to.

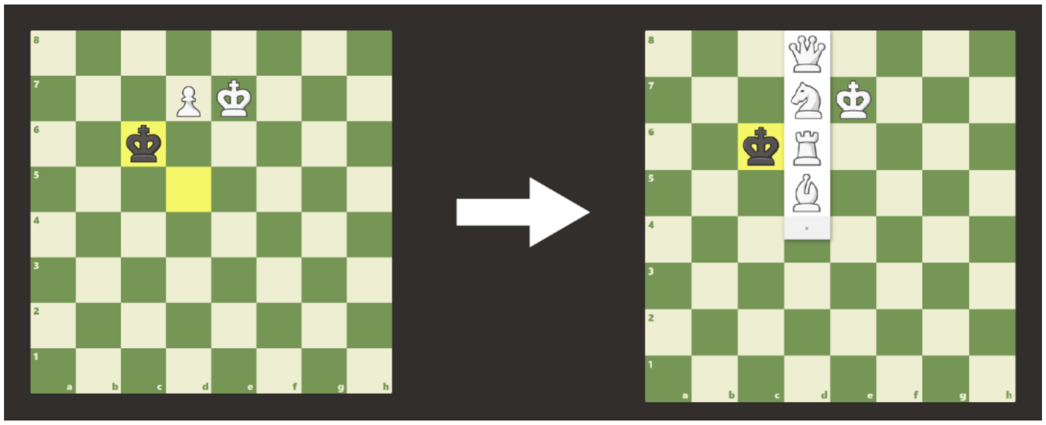


Figure 4.19: Chess.com's pawn promotion mechanics

Note that in Tjatoer! 2 pieces can promote, the man and the pawn. The pawn can promote to one of the pieces from the queen's court, and the man can promote to one of the pieces from the king's court. Currently, both of these can only promote one type of piece, queen for the pawn and lance for the man.

Engine

And now, we delve into the most complex thing of this project so far, it would help if you have some experience in machine learning although I would explain everything anyway, but having the prior knowledge would help a ton. Python is a very archaic language and not the best for making AIs, so I'll give you the freedom of choosing any other languages as you see fit. With that being said, I think a good goal to set the engine is for it to be able to reach depth 8-12 without slowing down. To reiterate, depth is how many moves ahead the engine can think or evaluate, depth 2 is a whole chess move so depth 10 is 5 chess moves ahead. Currently, the program's engine can go to depth 2, but it takes minutes just for it to make a single move, in comparison, other fairy-chess engines like ChessV and Fairy-Max can reach depth 5 almost instantly.

Move Ordering

The first thing we will get into is move ordering. What I mean by this is currently the engine randomizes which move it wants to evaluate first, in other words, it picks a random move to evaluate, after it finishes the evaluation of that move it will move to another random move, and so on. But it would be more efficient if we evaluate the possible moves that have the better probability of being the best move, like captures and checks. The program already has a list of valid moves that are used to determine where a piece can and cannot go, all we need to do is to put captures and checks or any other moves that have a probability of being the best move first before any other move, if there is no checks or captures, only then the engine would randomize which passive moves it would make first.

Advanced Positional Scoring

Instead of just counting what and where pieces are on the board, the engine could consider the dynamic between pieces as well. There is no way to concretely explain how to implement this since I would need to explain the whole concept of positional chess, I would suggest researching positional chess on your own, but some ideas include having the engine consider

- Pawn structures such as
    - Pawn chain
      A pawn chain is simply a chain of pawns that are defended by each other except for the one in the very back (the base of the pawn chain)
    - Isolated pawns or weak pawns
      Isolated pawns or weak pawns are pawns that are at the base of a pawn chain (capturing the base of a pawn chain can create a chain reaction in weakening the position) or have no pawns or pieces protecting it
    - Connected pass pawns
      Pass pawns are pawns that are not blocked or have passed the enemy pawns, in other words, these pawns would eventually promote if not blocked or captured. These pawns are surprisingly powerful and annoying in the endgame especially when they're in a chain (connected)
    - Doubled pawns
      Doubled pawns are two or more pawns that are on the same file as a result of captures, these pawns have limited mobility and considered as a weakness, so the engine should avoid having them
- Bad and good bishops
  A good bishop is a bishop that is not blocked by its ally pieces, on the other hand, a bad bishop is a bishop that is blocked by its ally pieces. this principle should also be applied to other pieces in Tjatoer! that can move diagonally like a bishop.
- Batteries
  In regular chess, a battery is when a queen and a bishop are on the same diagonal line, or similarly a queen and a rook that is on the same file or rank. In Tjatoer! many types of pieces have the same moves so a shah and a chariot can make a battery for example
- Connected pieces

Connected pieces are two pieces that protect each other, again, this principle should apply to other pieces in Tjatoer! that has similar moves, for example, a hawk and a knight or a hawk and a gajah can protect each other

- King safety
  The engine should be able to determine the safety of its king (is it exposed to open files or ranks? is it pinned? is it prone to checks or even threatened to be checkmated? And other aspects of the king safety).

With all that being said let's look at an example position, in Figure 4.20 below, the material balance is equal but black has an advantage if we consider advanced positional values,

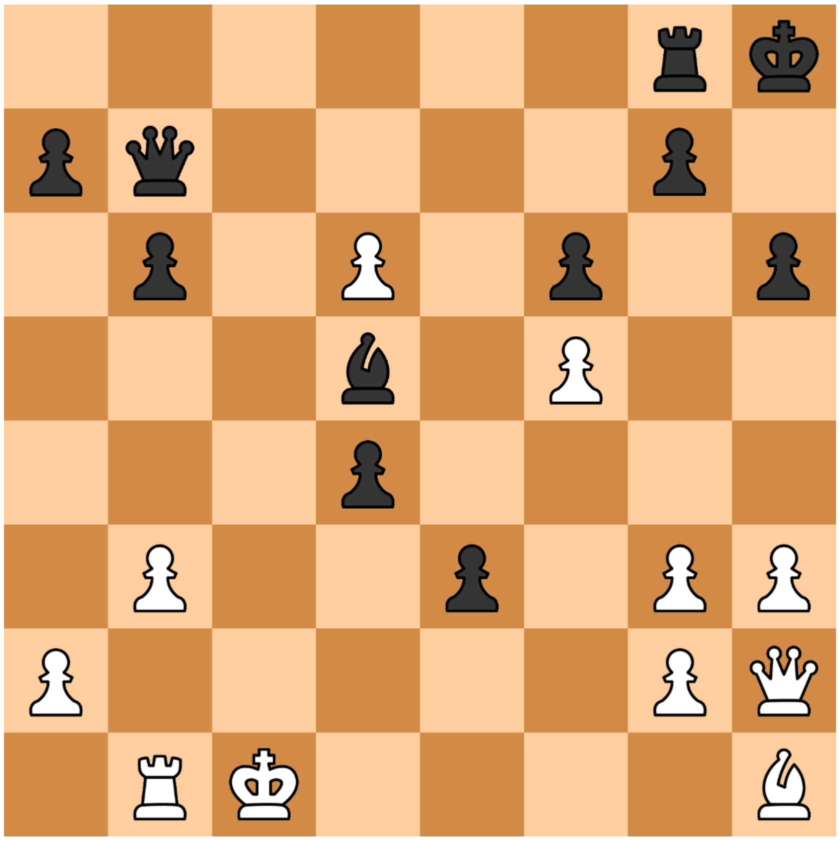


Figure 4.20: Example position

If we use the current positional scoring on the position above, the engine will say it's about equal. Both sides have bishops and queens on or near to the longest diagonal they can get to, from white's perspective he has 2 pawns on the second rank, 3 on the third rank, 1 on the fifth, and one on the sixth rank. From black's perspective, they also have 3 pawns on the second rank (seventh rank from white's perspective), 3 pawns on the third rank (sixth rank from white's perspective), 1 on the fifth rank (fourth rank from white's perspective), and one on the sixth rank (third rank from white's perspective). Both rooks are also symmetrically positioned.

But if we analyze the position with more advanced positional valuing, you'll see that white's queen and bishop can't go anywhere and all of white's pawns are on light squares which will possibly block their bishop, their pawns on the g file are doubled, their pawns on d6 and f5 are isolated, and their king is exposed, in fact, if it were black to move they can check the king in four different ways (Qc6+, Qc7+, Qc8+, Rc8+). Meanwhile, the black bishop is more active, all of their pawns are on dark squares which won't constrain the bishop, the queen and bishop form a very powerful battery, their pass pawns are connected, and the king is safe, thus we can say that black is better here.

Capture Evaluation

Next, we have capture evaluation, we need a different mechanic that deals with evaluating captures that are not bound by evaluation depth. I will explain the reason for this, see Figure 4.21 below.

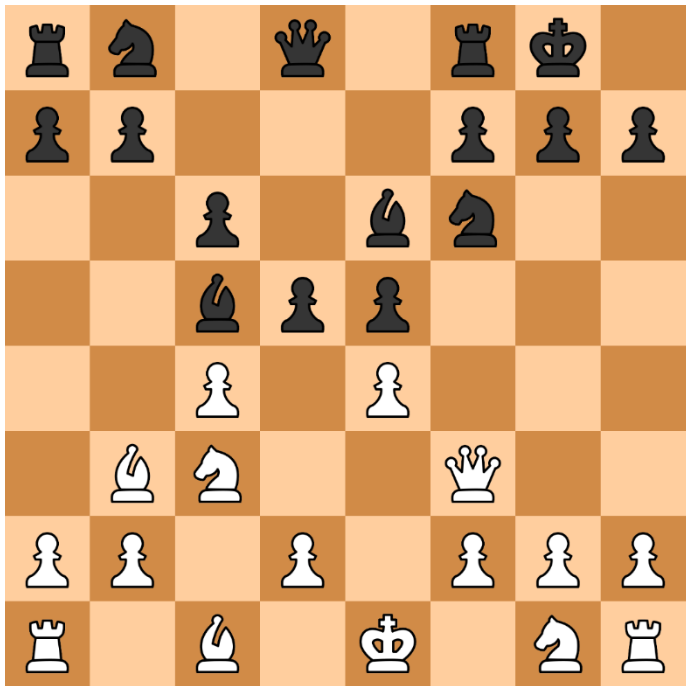


Figure 4.21: Example position, white to move

Assume both sides are engines, black has just moved their pawn to d5 (0. d5, since this is an example, the pawn moving to d5 is considered the 0th move), the question is, does this move lead to a loss in material for black? The answer to this question is yes, but, if the engine is set to a depth of 8, it would answer no, why? Well let's analyze, it's white's move and white can capture the pawn on d5:

- 1. exd5 cxd5 (depth 2)

The material is equal, white captured the d5 pawn with their pawn on e3, gaining 1 point, then black recaptures white's pawn with their pawn on c6, also gaining 1 point, so so far so good.

- 1. exd5 cxd5 2. cxd5 Bxd5 (depth 4)

Again, white captures the pawn on d5 this time with their pawn on c3, gaining 1 point, then black recaptures the pawn this time with their bishop, also gaining 1 point, thus far black still thinks that playing 0. d5 is safe

- 1. exd5 cxd5 2. cxd5 Bxd5 3. Bxd5 Nxd5 (depth 6)

White then captures black's bishop with their own bishop gaining 3 points, but black can simply recapture white's bishop with their knight on f6, gaining back those 3 points they lost, the material balance is still equal.

- 1. exd5 cxd5 2. cxd5 Bxd5 3. Bxd5 Nxd5 4. Nxd5 Qxd5 (depth 8)

Now white captures black knight with their own knight on c3, gaining 3 points, then white recaptures white's knight with their queen, regaining the three points they lost, thus the material balance is equal. We have reached the critical depth 8, remember, if the engine was set to depth 8, it would stop its evaluation on 4...Qxd5 and would not go any further, but had it evaluated just one more move, the engine playing black would now realize that black is now losing

- 1. exd5 cxd5 2. cxd5 Bxd5 3. Bxd5 Nxd5 4. Nxd5 Qxd5 5. Qxd5

And black just lost their queen, the remainder of the game would be easy for white because they will be up 9 points. If black didn't play 4...Qxd5, then white would still be up 3 points since white just captured black's knight on 4. Nxd5. Thus, playing 0. d5 would lead to a loss in material for black.

From this example, we can conclude that the engine must evaluate all captures that are possible until that there are no captures left to ensure that it won't lose any material, regardless of the depth it is set to.

Transposition Table

Yet another way to improve the efficiency of the engine is by implementing a transposition table. A transposition table stores all the positions that have been evaluated, since there are multiple ways to reach a position. To understand it better, let's look at an example position below in Figure 4.22

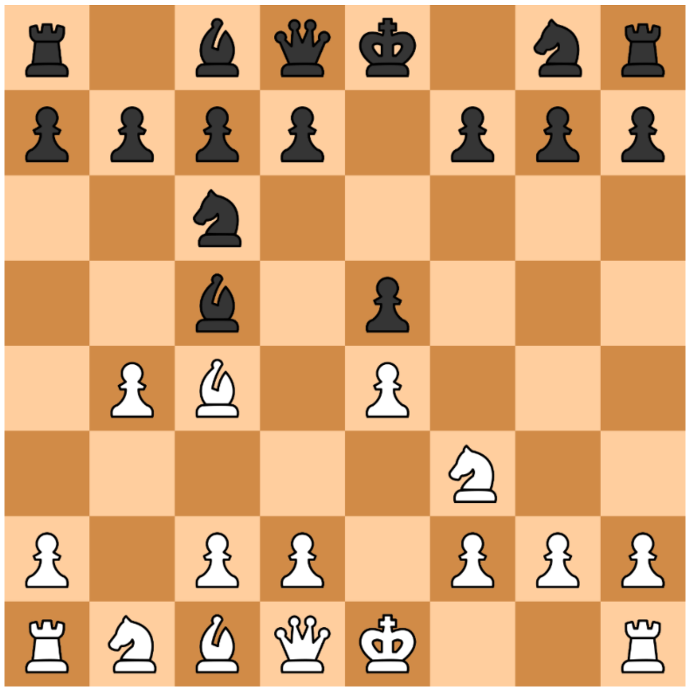


Figure 4.22: Example position, the Evans Gambit

The position above is the first four moves of the common chess opening, the Evans Gambit. The position can be reached in multiple ways, simply shuffle the order in which the moves are played, and you'll reach the same position, for example:

- 1. e4 e5 2. Nf3 Nc6 3. Bc4 Bc5 4. b4

This position is mostly reached through the line above, but there are other ways to reach it:

- 1. Nf3 Nc6 2. e4 e5 3. Bc4 Bc5 4. b4
- 1. e4 e5 2. Bc4 Bc5 3. Nf3 Nc6 4. b4
- 1. e4 e5 2. Bc4 Nc6 3. Nf3 Bc5 4. b4

Those three lines above are some of the valid ways to reach the Evans Gambit position, there are tons of more ways to reach it, but be careful, not all ways are safe to play, for example:

1. 1. e4 e5 2. b4 Nc6 3. Nf3 Bc5 4. Bc4

That line gets to the same position but after 4. Bc4 but white can simply capture the bishop and win a piece. Another example

2. 1. b4 Nc6 2. Nf3 e5 3. e4 Bc5 4. Bc4

On paper, that line also takes you to the Evans Gambit, but in this case, if black is smart enough, instead of playing 2...e5, they could just take the undefended pawn by playing

2...Nxb4, therefore the engine must assume that the opponent is smart enough to see that and avoid that line altogether.

Those are just examples of how a set of moves can be shuffled to reach the same position, regardless of the safety of the order of moves of which to reach a position, the engine mustn't evaluate the same position over and over again. So, to improve the efficiency of the engine, the engine must have the positions that it has evaluated stored in a database where it can refer to, so it won't have to reevaluate a position. Again, the chess programming wiki would have a better in-depth technical explanation of this, so I suggest going there.

Opening Variety and Endgame Tablebase

the final features that I'm planning to add, for now, I was also planning to add an opening database so that the engine doesn't have to put much effort into evaluation in the early stages of the game, but that would take time and effort and is already adequately touched on the advanced positional scoring section, the only thing I would like to point out is that when using the mentioned advanced positional scoring, since the opening stages of the game usually don't have any checks or captures, the engine would eventually find the same best first few moves, to add variety to the engine's play may be the engine could make a list of, for example, top five moves that it has found and then picks one of those moves randomly to increase its variety in playing,

as for the endgame tablebase, it is a little bit complicated. In chess and chess variants, endgames are sometimes winning but the strategy for a checkmate sometimes takes more than 20 moves, so the engine with only depth 10 for example would just randomly shuffle the pieces then suffer a 100 move draw. For simplicity sake for now we will **define the endgame as a point where there are less than 5 pieces on the board,** a king and a piece vs king and two pieces (K + 1 vs K + 2) and king and a piece vs king and a piece (K + 1 vs K + 1) there is no king vs king and three or two pieces (K vs K + 3 or K vs K + 2) since those counts as a win. from what I can gather, tablebases are generated by working backward from the final mating/drawing positions, "retracting" moves rather than playing them forward. First, every possible final position is exhaustively listed. Then from each, we "retract" (take back) moves that could have been played. In doing so, we know a possible outcome of the resulting position (by playing forward to the final position again). Do this for all positions, and we know the ideal outcome of all of them. Now, what is a retraction? Pieces move backward, so everything moves as in normal chess except pawns. Pieces can uncapture other pieces, leaving an extra piece on the board. Notably, pieces can unpromote into pawns. Implementing transposition tables before endgame tablebases are important because transposition tables can be used for retracting.

For example we want to go to any position (for example position N), to any checkmate position (for example position M), to find the best way from N → M we will start at M, then rewind/retract for example 10 moves to position E (the more the better, since we can better utilize the transposition table), the engine would then store every possible variations of position E with optimizations from the transposition table, same principle applies but backward, and therefore every pathways leading from E → M, then from E we do the same thing, retracting moves to the position D, then storing all the variations of D with transposition table optimization, therefore storing all the pathways from D → E, and we keep doing that (from positon D we go back to position C, then B, then A, then Z, then Y and so on) until we run into position N (initial position), and once we run into position N, the engine would now have a clear way to go from N → M, note that not every K + 1 vs K + 2 and K + 1 vs K + 1 positions lead to checkmate, some are draws but we won't have to worry since there is a 100 move draw rule.

Of course, there will be more efficient algorithms out there to evaluate endgames but for the implementation, we have 2 ways of doing this, either:

- Evaluate all possible positions of all possible combination of K + 1 vs K + 2 and K + 1 vs K + 1 pieces (such as KRR vs KQ, KQ vs KN, etc.) and storing it into a single huge database, for comparison, regular chess in the modern era with advanced engines have only figured out 7-piece endgames, we're doing 5-piece endgames here, granted the data that is needed reduces significantly and exponentially for every piece that is substracted, the data for storing all possibilities 5-piece endgames should be some tiny fraction of 7-piece endgames, plus we are eliminating all the K vs K + 3 or K vs K + 2 combination. This will be practical to use during the game thus the engine can play in time control (we will have to do this only one time) but will take some time and processing power to do it.
- Or we can just do it in-game, evaluate all possible positions when a 5-piece combination arises during a game, this will still take a long time and the engine won't have enough time to do time control, but it doesn't take up much data storage space.

Chapter 5: Summary

I realize that all the things I said here and in previous chapters are easier said than done, but I would still appreciate all improvement anyone can make in this project. This chapter contains lists for future references to anyone who wishes to improve this project.

Goals

To sum up and for easy reference in the future, here is the to-do list of the features I would like to be implemented:

- A start menu with game mode options to play against a human, against the engine as either color or to watch 2 engines play;
- Add an appearance options/general options window containing the options to change board colors, square highlight colors, the font and font color, and piece sets;
- A help window that appears when a user right-clicks on a piece or from the options;
- Add a redo move feature;
- Add a "Forward to End" and "Back to start" so the user doesn't have to spam the redo and undo when reviewing a game;
- A board flipper, way to flip the board so the player can both play from white and black's perspective;
- A separate move log window that can be scrolled indefinitely thus has no maximum number of moves it can display;
- Polish up the move recording feature, add the notation for checks, checkmate, and/or when two pieces can go to the same square
- A separate engine evaluation window, in other words, display the already made engine evaluation in the console to a more polished dedicated window;
- Add an immediate draw option;
- Add a resign feature (the gameover.mp3 file in the Sounds folder should play when this feature is used);
- Add a pause feature when two engines are playing or when playing with time control;
- Add a way to access all of the above options and features in the form of menus above the board such as:
    - the file menu (containing start new game and quit);
    - the help menu (containing the button to open the help window);
    - the view menu (containing the board flipper, the button to open the move log window, the engine output window);
    - the action menu (containing the draw, resign, undo, redo, back to start, and forward to end option);

    - the mode menu (containing the machine white, machine black, and pause feature);
    - the options menu (containing the mute all sounds, time control, and general options/appearance options window);
- Proportional and smooth animation of pieces moving;
- Implement baring the king as a way to win;
- Implement the 100-move rule to draw;
- Implement fivefold repetition draw;
- Making it so that the user can drag the pieces to move them;
- Implement pre-moving;
- Implement basic time controls;
- Implement an option for pawn promotion so that the user can choose which of the pieces (the pieces in the queen's court) to promote to;
- Implement an option for man promotion so that the user can choose which of the pieces (the pieces in the king's court) to promote to;
- A typical chess player can think up to 4-6 moves ahead so I guess a good goal for the engine is for it to reach depth 8-12 without slowing down significantly, for the engine what needs to be implemented include:
    - Implement move ordering in the engine;
    - Implement a specific capture evaluation mechanic for the engine;
    - Implement a transposition table in the engine;
    - Creating an endgame tablebase for the engine;
    - Creating a more advanced positional scoring for the engine.

I will leave it up to you, whatever the way to achieve these goals, but I would appreciate it if you keep track of the general changes you made (no need to be detailed) so that I can refer it in the future.

Optional Additions

Some more optional features I would add for a more user-friendly experience include:

- Changing the taskbar icon on Windows OS so that it looks like the actual window icon as opposed to the default python icon;
- Implementing a script where the program automatically installs the pygame module and other necessary components if the user's computer doesn't have it installed yet so that the whole program can be given in a zip file and run immediately out of the box;
- Change the way the program check for checks since the current one is very ineffective;

- Implementing a material balance window so that the user can see what pieces are already off the board (captured), the window also provides the sum of the material balance of the board at that state;
- A save game as diagram feature to basically take a screenshot of the current state of the board in the file menu above the board;
- An engine graph history window is another way to display the engine's evaluation, this could be accessed in the view menu above the board;
- Implement increment time for the time controls;
- an analysis board builder option so that the user can set a custom position and get an engine evaluation or continue to play it against the engine for example;
- More on the previous one, PGN and FEN input capabilities (I won't explain what those are here so maybe google it);
- A save game feature where a player can save a game, they just played in the form of a PGN file (again if you don't know what a PGN is I advise you to google it).

To end this chapter and document, I would apologize because some of the code I have written is written in my native language. Also, the features and improvements of this program do not have to be written in Python, but I would appreciate it if it did, and all the features I have mentioned are just a few of many features that can be added, so any feature that is implemented but isn't mentioned here are welcomed. I would also appreciate it if the final product of the code contains some comments or annotations to help me better understand the code. But all in all, thank you.