# User Manual

Minor Carlson

Version 1.0

Team 21:
Anthony Flores
Chenyu Wang
Davis Nguyen
Nathan Bae
Timothy Chan

# Table of Contents

# Glossary

## Chess Pieces:

**Pawn:** The Pawn can only move forward. It can only attack diagonally. On its first move it can either move forward once or twice. Once it reaches the completely other side of the board it is able to transform into either a Bishop, Knight, Queen, or Rook.

**Rook:** The Rook can move vertically or horizontally for as many spaces it wants given there isn't any piece in the way. It can attack any piece horizontal or vertical.

**Knight:** The Knight can move in L shape so either two moves vertically and one move horizontally or two moves horizontally and one move vertically. It is able to jump over pieces. It can attack any piece it lands on.

**Bishop:** The Bishop can move diagonally as many spaces as it wants. It can not jump over pieces and it attacks diagonally as well.

**Queen:** The Queen can move diagonally, vertically, or horizontally as many spaces as it wants. It can not jump over pieces and it attacks diagonally, vertically, or horizontally.

**King:** The King can move diagonally, vertically, or horizontally but for only one space. It can not jump over pieces and it attacks diagonally, vertically, or horizontally.

## Chess Moves and Outcomes:

**Castling:** Is a special move where the Ring and Rook simultaneously move towards each other. The king moves two squares towards the rook and the rook moves right next to the king on the opposite side. Castling can only be done when the King and Rook have yet to move and there are no pieces between them.

**Enpassant:** this special move involves the pawns. This move consists of a pawn being able to capture a pawn right after the pawn has taken two moves to avoid capture. The capture must be done right after the pawn's two space evasion. The capturing pawn can diagonally attack the opposing pawn at the square right behind the opposing pawn.

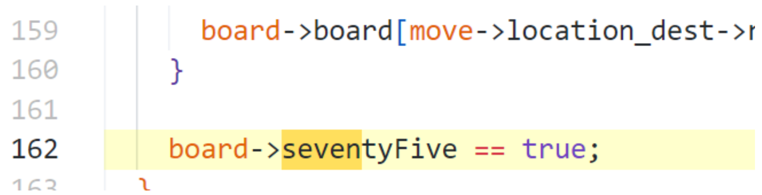
**Check:** When the king is being threatened it is called check. When an opposing piece has the opportunity to capture the King in the next turn if no preventative action is taken this is called

“Check”. Check forces the threatened player to take action to prevent his king from being captured in the next turn.

**Checkmate:** When in Check the defending player has no protective action available to save their King.

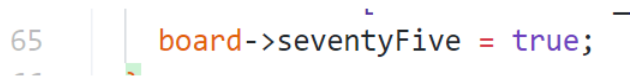
**Draws:**

**Stalemate:** This occurs when a player has no possible moves and is not in Check.
Impossibility of Checkmate: This occurs when there aren't enough pieces in each of the players to produce a Checkmate.
**75-move-rule:** If no pawn moves or capture occurs within 75 moves, it is an automatic draw.

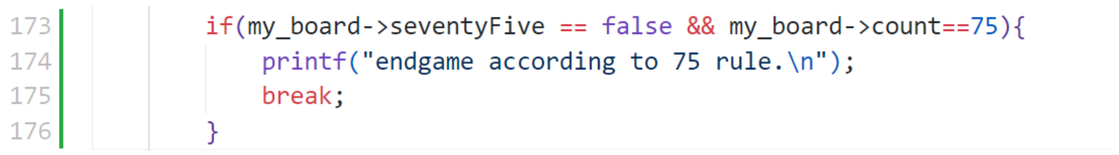


```
        board->board[move->location_dest->
    }

    board->seventyFive == true;
```



```
    board->seventyFive = true;
```



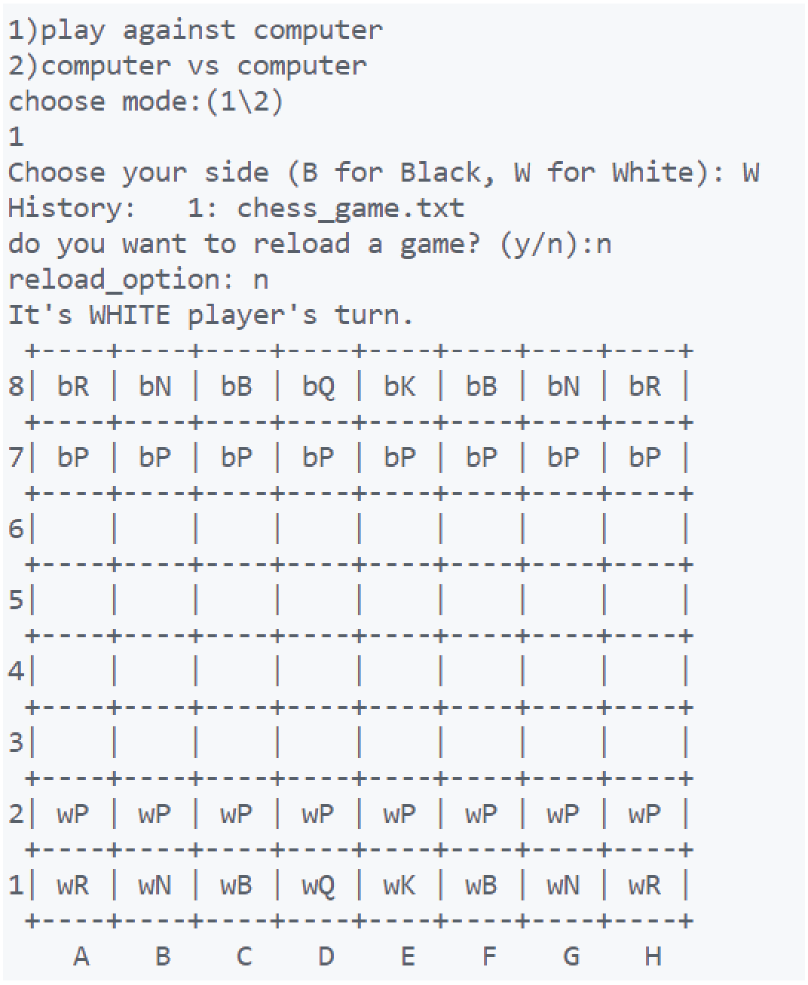
```
            if(my_board->seventyFive == false && my_board->count==75){
                printf("endgame according to 75 rule.\n");
                break;
            }
```

(code for implementing 75 rule)

**Draw agreement:** Players are allowed to agree on a draw at any point in the game.

# Computer Chess

## 1.1 Usage scenario



```
1)play against computer
2)computer vs computer
choose mode:(1\2)
1
Choose your side (B for Black, W for White): W
History:   1: chess_game.txt
do you want to reload a game? (y/n):n
reload_option: n
It's WHITE player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP | bP | bP | bP | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP | wP | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
```

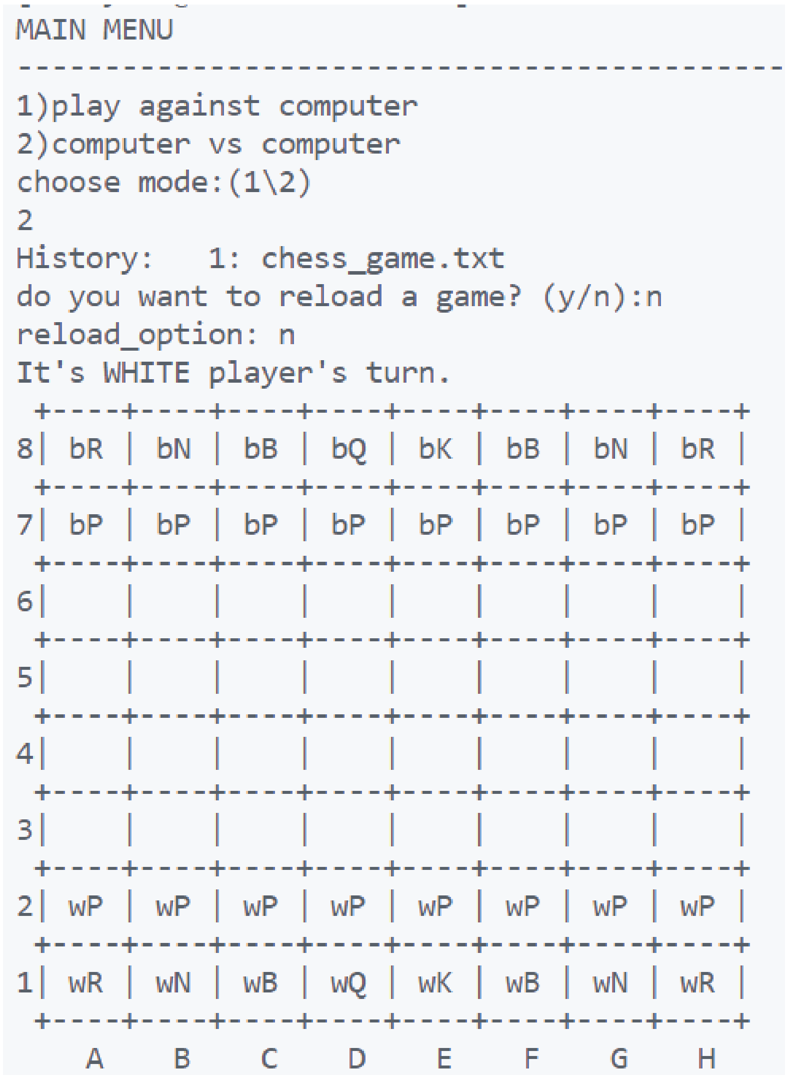
**Fig1. Starting Menu**

First, when entering the game, there will be a main menu with several choices:

- player against computer  & computer vs computer
- Human player chooses side (Black or White)
- Reload game using log file “chess_game.txt”

After all the choices are made, the program will draw the board and let the player input moves.

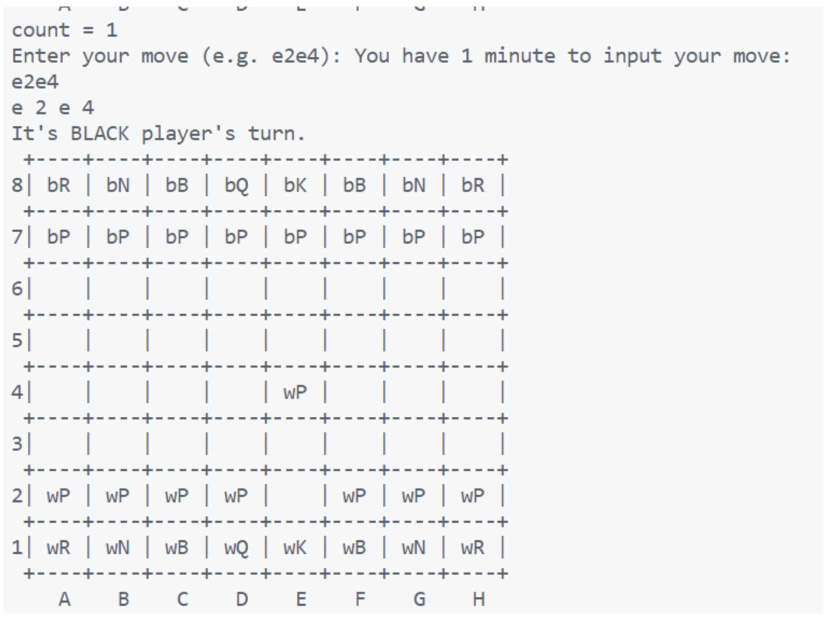
If the user choose computer vs computer, there will be no choosing side option:



```
MAIN MENU
-------------------------------------------
1)play against computer
2)computer vs computer
choose mode:(1\2)
2
History:   1: chess_game.txt
do you want to reload a game? (y/n):n
reload_option: n
It's WHITE player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP | bP | bP | bP | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP | wP | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
```

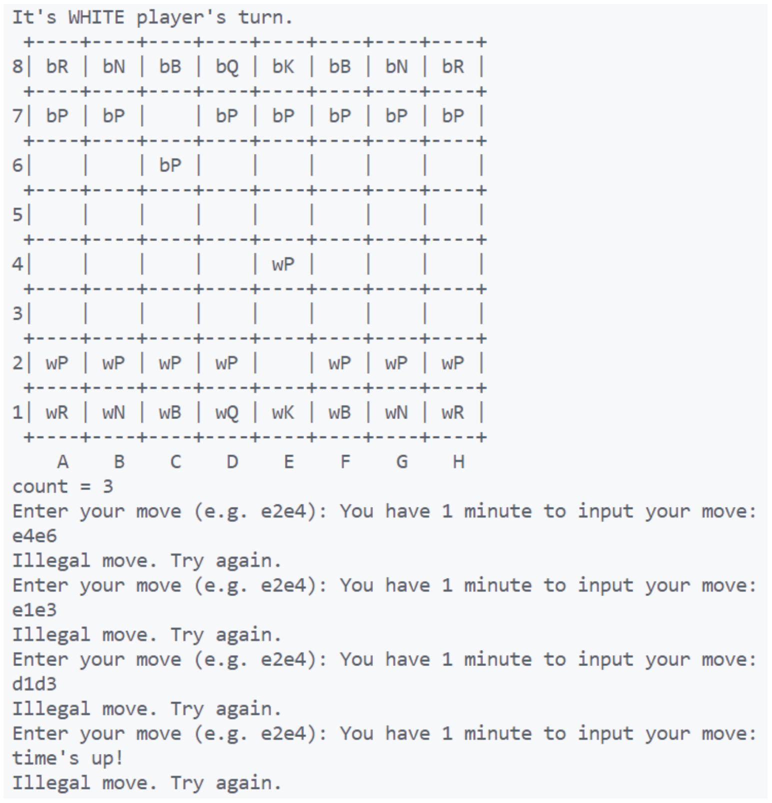
**Fig2. User chooses computer vs computer**

If the user inputs an illegal move, in case the user inputs a type, the program will warn the user, which will be considered as losing the game in the tournament.



```
count = 1
Enter your move (e.g. e2e4): You have 1 minute to input your move:
e2e4
e 2 e 4
It's BLACK player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP | bP | bP | bP | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    |    |    | wP |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP |    | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
   A    B    C    D    E    F    G    H
```

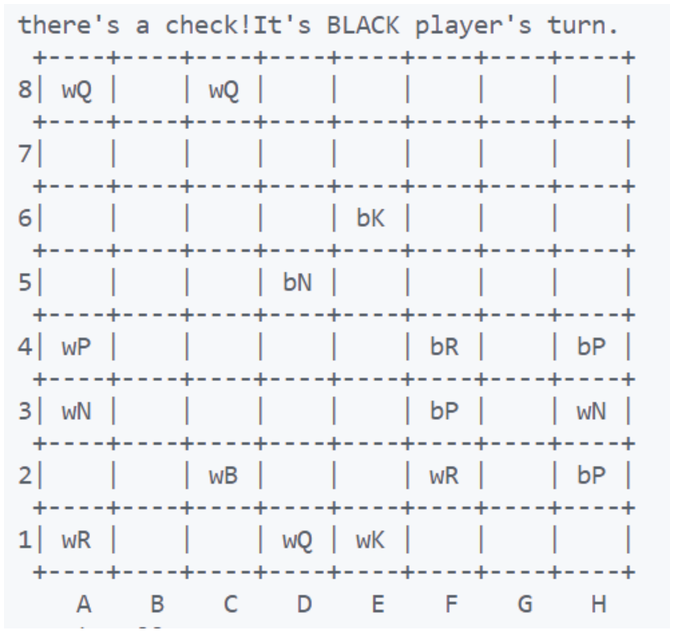
**Fig3 User inputs legal moves**



```
It's WHITE player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP | bP |    | bP | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    |    | bP |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    |    |    | wP |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP |    | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
   A    B    C    D    E    F    G    H
count = 3
Enter your move (e.g. e2e4): You have 1 minute to input your move:
e4e6
Illegal move. Try again.
Enter your move (e.g. e2e4): You have 1 minute to input your move:
e1e3
Illegal move. Try again.
Enter your move (e.g. e2e4): You have 1 minute to input your move:
d1d3
Illegal move. Try again.
Enter your move (e.g. e2e4): You have 1 minute to input your move:
time's up!
Illegal move. Try again.
```

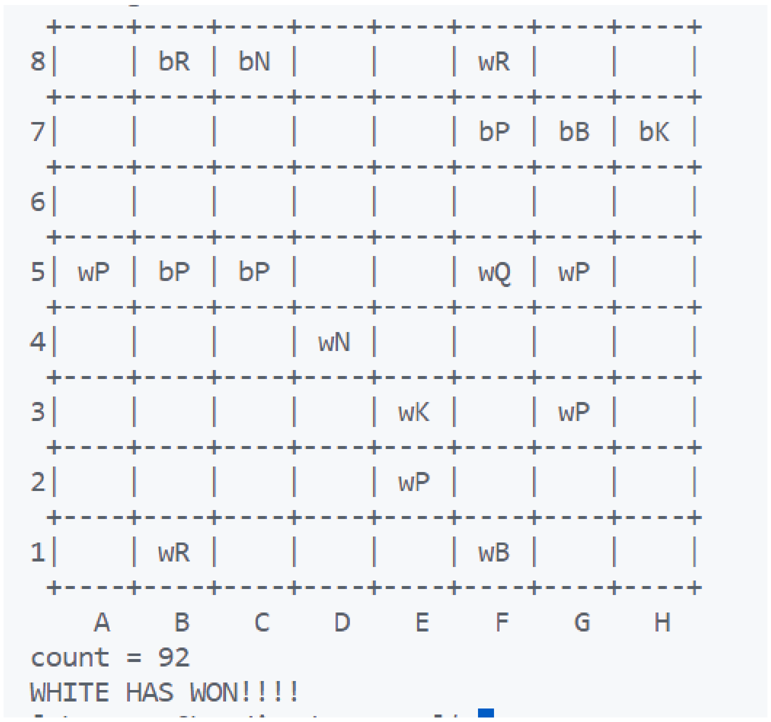
**Fig 4. User inputs illegal moves or reaches the time limit (The four errors: pawn takes 2 steps after the first move; King moves against the rule; Queen moves when the way is blocked)**

It will be prompted when there’s a check or one player wins the game:



```
there's a check!It's BLACK player's turn.
 +----+----+----+----+----+----+----+----+
8| wQ |    | wQ |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
7|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
6|    |    |    |    | bK |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    | bN |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4| wP |    |    |    |    | bR |    | bP |
 +----+----+----+----+----+----+----+----+
3| wN |    |    |    |    | bP |    | wN |
 +----+----+----+----+----+----+----+----+
2|    |    | wB |    |    | wR |    | bP |
 +----+----+----+----+----+----+----+----+
1| wR |    |    | wQ | wK |    |    |    |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
```

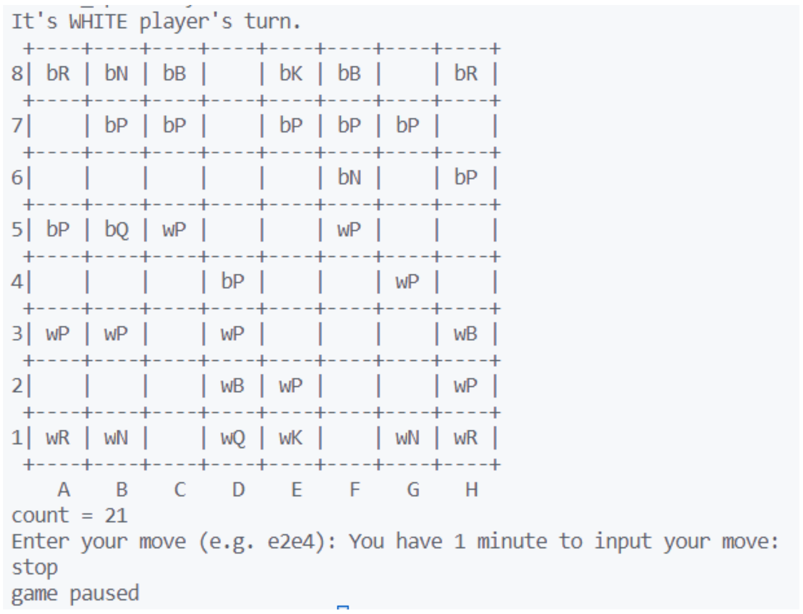
**Fig 5. Check**



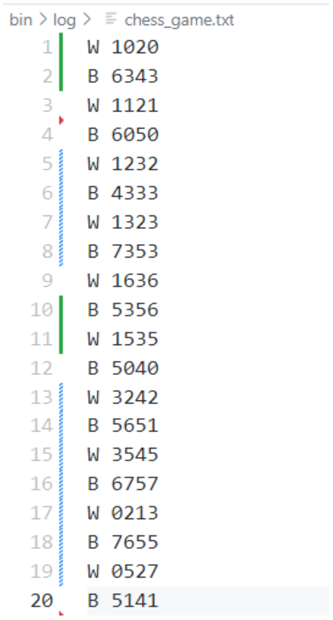
```
 +----+----+----+----+----+----+----+----+
8|    | bR | bN |    |    | wR |    |    |
 +----+----+----+----+----+----+----+----+
7|    |    |    |    |    | bP | bB | bK |
 +----+----+----+----+----+----+----+----+
6|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5| wP | bP | bP |    |    | wQ | wP |    |
 +----+----+----+----+----+----+----+----+
4|    |    |    | wN |    |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    | wK |    | wP |    |
 +----+----+----+----+----+----+----+----+
2|    |    |    |    | wP |    |    |    |
 +----+----+----+----+----+----+----+----+
1|    | wR |    |    |    | wB |    |    |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
count = 92
WHITE HAS WON!!!!
```

**Fig 6. Endgame Prompt**

When the user chooses player vs computer, the user could input “stop” to pause the game, and the log file will be saved:



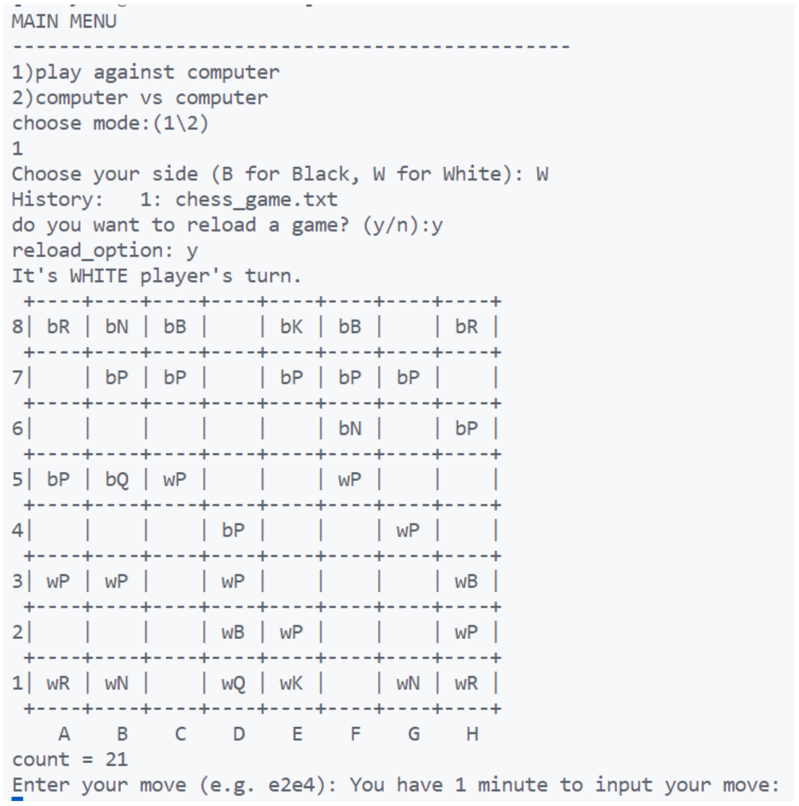
```
It's WHITE player's turn.
  +----+----+----+----+----+----+----+----+
8| bR | bN | bB |    | bK | bB |    | bR |
  +----+----+----+----+----+----+----+----+
7|    | bP | bP |    | bP | bP | bP |    |
  +----+----+----+----+----+----+----+----+
6|    |    |    |    |    | bN |    | bP |
  +----+----+----+----+----+----+----+----+
5| bP | bQ | wP |    |    | wP |    |    |
  +----+----+----+----+----+----+----+----+
4|    |    |    | bP |    |    | wP |    |
  +----+----+----+----+----+----+----+----+
3| wP | wP |    | wP |    |    |    | wB |
  +----+----+----+----+----+----+----+----+
2|    |    |    | wB | wP |    |    | wP |
  +----+----+----+----+----+----+----+----+
1| wR | wN |    | wQ | wK |    | wN | wR |
  +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
count = 21
Enter your move (e.g. e2e4): You have 1 minute to input your move:
stop
game paused
```



```
bin > log > chess_game.txt
W 1020
B 6343
W 1121
B 6050
W 1232
B 4333
W 1323
B 7353
W 1636
B 5356
W 1535
B 5040
W 3242
B 5651
W 3545
B 6757
W 0213
B 7655
W 0527
B 5141
```

**Fig 7. Pause the game** **fig 8. Log file**

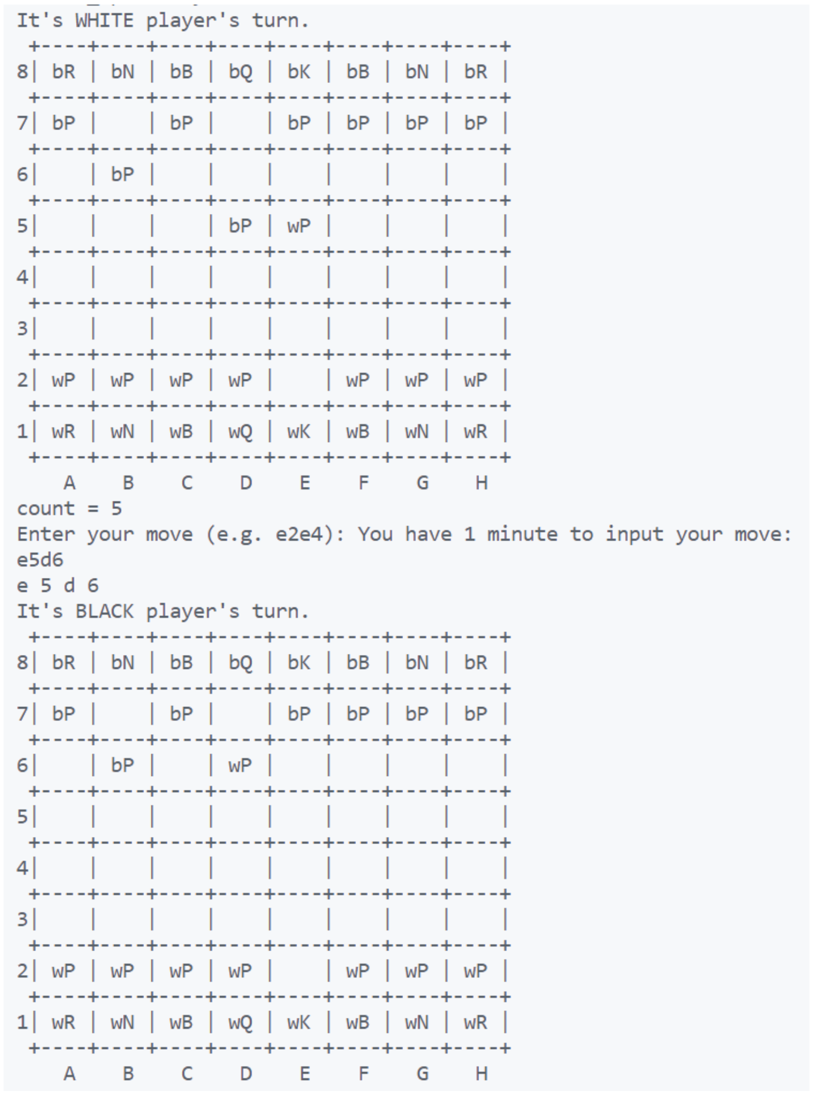
When starting the game again, the user could choose to reload the game:



```
MAIN MENU
------------------------------------------------
1)play against computer
2)computer vs computer
choose mode:(1\2)
1
Choose your side (B for Black, W for White): W
History:    1: chess_game.txt
do you want to reload a game? (y/n):y
reload_option: y
It's WHITE player's turn.
  +----+----+----+----+----+----+----+----+
8| bR | bN | bB |    | bK | bB |    | bR |
  +----+----+----+----+----+----+----+----+
7|    | bP | bP |    | bP | bP | bP |    |
  +----+----+----+----+----+----+----+----+
6|    |    |    |    |    | bN |    | bP |
  +----+----+----+----+----+----+----+----+
5| bP | bQ | wP |    |    | wP |    |    |
  +----+----+----+----+----+----+----+----+
4|    |    |    | bP |    |    | wP |    |
  +----+----+----+----+----+----+----+----+
3| wP | wP |    | wP |    |    |    | wB |
  +----+----+----+----+----+----+----+----+
2|    |    |    | wB | wP |    |    | wP |
  +----+----+----+----+----+----+----+----+
1| wR | wN |    | wQ | wK |    | wN | wR |
  +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
count = 21
Enter your move (e.g. e2e4): You have 1 minute to input your move:
```

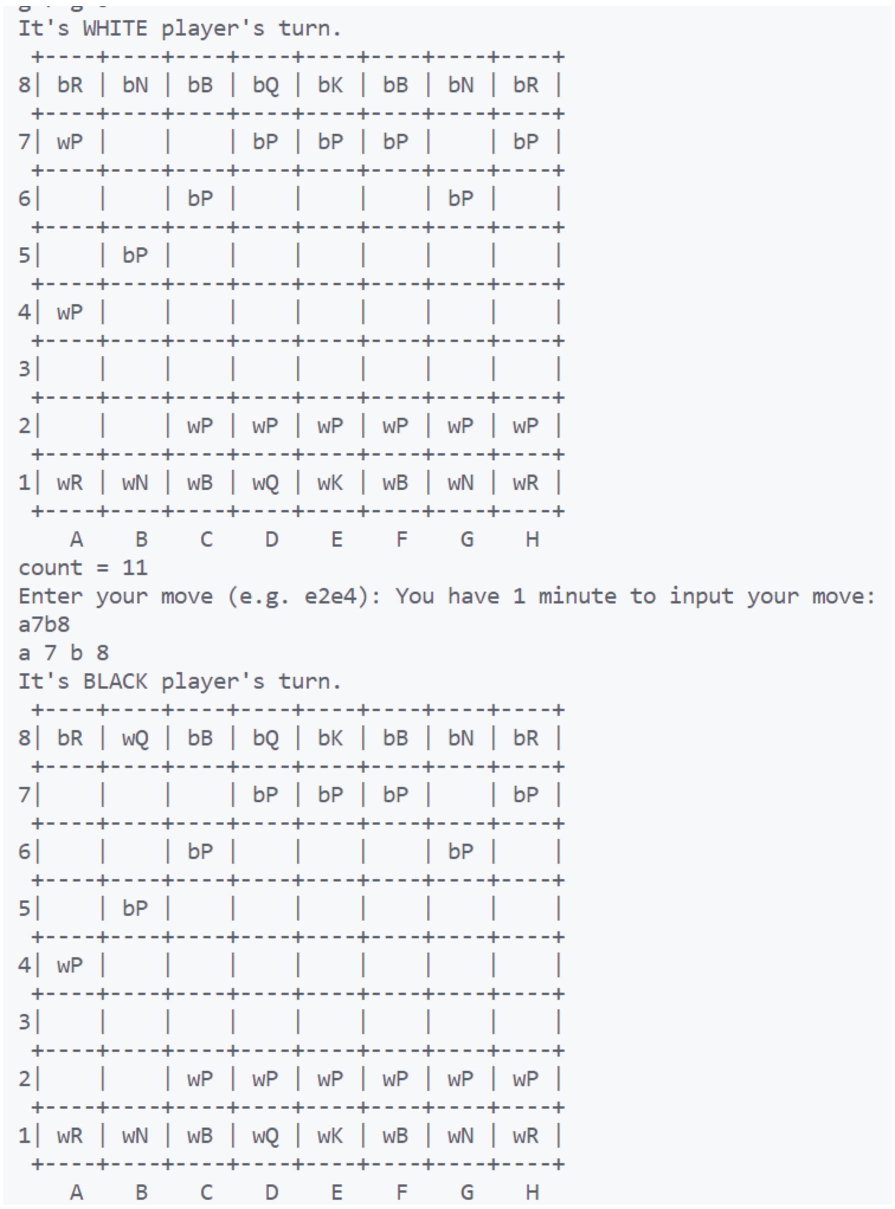
**Fig 9. Reload a paused game**

There are some special moves like Enpassant, Castling and Pawn promotion, which are shown in the following figures.



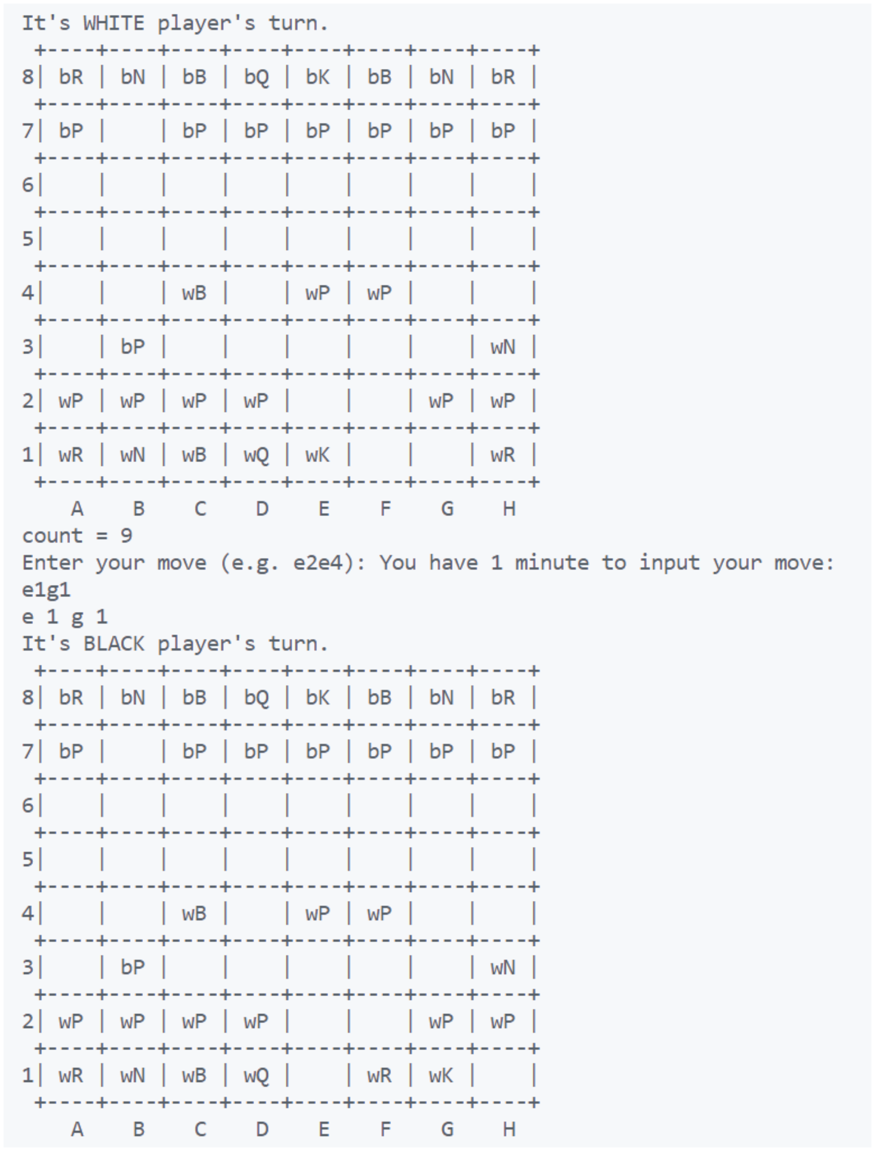
```
It's WHITE player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP |    | bP |    | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    | bP |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    | bP | wP |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP |    | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
count = 5
Enter your move (e.g. e2e4): You have 1 minute to input your move:
e5d6
e 5 d 6
It's BLACK player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP |    | bP |    | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    | bP |    | wP |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP |    | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
```

**Fig 10. Enpassant**



```
It's WHITE player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| wP |    |    | bP | bP | bP |    | bP |
 +----+----+----+----+----+----+----+----+
6|    |    | bP |    |    |    | bP |    |
 +----+----+----+----+----+----+----+----+
5|    | bP |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4| wP |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2|    |    | wP | wP | wP | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
count = 11
Enter your move (e.g. e2e4): You have 1 minute to input your move:
a7b8
a 7 b 8
It's BLACK player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | wQ | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7|    |    |    | bP | bP | bP |    | bP |
 +----+----+----+----+----+----+----+----+
6|    |    | bP |    |    |    | bP |    |
 +----+----+----+----+----+----+----+----+
5|    | bP |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4| wP |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
3|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
2|    |    | wP | wP | wP | wP | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK | wB | wN | wR |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
```

**Fig 11. Pawn Promotion**
**(the black pawn at A7 eats the white Knight at B8 and is promoted to a Queen)**



```
It's WHITE player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP |    | bP | bP | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    | wB |    | wP | wP |    |    |
 +----+----+----+----+----+----+----+----+
3|    | bP |    |    |    |    |    | wN |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP |    |    | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ | wK |    |    | wR |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
count = 9
Enter your move (e.g. e2e4): You have 1 minute to input your move:
e1g1
e 1 g 1
It's BLACK player's turn.
 +----+----+----+----+----+----+----+----+
8| bR | bN | bB | bQ | bK | bB | bN | bR |
 +----+----+----+----+----+----+----+----+
7| bP |    | bP | bP | bP | bP | bP | bP |
 +----+----+----+----+----+----+----+----+
6|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
5|    |    |    |    |    |    |    |    |
 +----+----+----+----+----+----+----+----+
4|    |    | wB |    | wP | wP |    |    |
 +----+----+----+----+----+----+----+----+
3|    | bP |    |    |    |    |    | wN |
 +----+----+----+----+----+----+----+----+
2| wP | wP | wP | wP |    |    | wP | wP |
 +----+----+----+----+----+----+----+----+
1| wR | wN | wB | wQ |    | wR | wK |    |
 +----+----+----+----+----+----+----+----+
    A    B    C    D    E    F    G    H
```

**Fig 12 Castling**

## 1.2 Goals

Our main goal for this project is to create a fully functional chess program that includes the required basic functions and integrating the advanced options that are desirable if time permits. This will incorporate a multitude of software engineering concepts. These include documenting the program itself, designing the program, writing the code, and debugging the code. Additionally, this project will provide a way to work as a team with other group members. Students will need to successfully cooperate with one another in order to complete this project.

## 1.3 Features

The most basic features that will be included in this program include:

- Follows the official rules of chess
- User interface where the use can see the board and type in where to move
- Allows user to decide which side to play (white or black) and to see the board
- Supports human player vs player or human player vs computer
- Provides a log file of all actions after the game is finished
- User has a one minute timer to input a move or it will resort in the game ending and the opposing side being announced the winner

The advanced feature that will be included in this program include:

- Advanced AI
- Computer vs. Computer gamemode
- Could reload the game and set up the board if the target log file have contents

# Installation

## 2.1 System requirements

A functional computer (Either Windows or Mac) that is capable of running Linux. The machine can run this program in both a 32-bit and 64-bit operating system. This program requires little processing power to function. Additionally, an insignificant amount of hard drive space is required. It should be compatible with other computers running a similar program to be able to function against each other.

## 2.2 Setup and configuration

For the setup of the chess program, simply download the tar.gz file and choose the location of where you want the program to be placed. Then simply unpack the tar.gz and run the executable file in order to open up the program. To run the program please type the following into the terminal: “bin/chess”.

## 2.3 Uninstalling

To uninstall the program, simply delete the unpacked folder and its contents.

# Chess Program Functions and Features

## int main()

- Takes input of which game mode was chosen
- After deciding the gamemode, it will initialize the game
- There are 3 game modes (Player vs. Player, Player vs Computer, Computer vs. Computer)
- It will setup the chessboard and asks user which side they want (Black or White)
- Will call upon the draw function to create the chess board.
- The opposite side will be assigned to the computer
- After the game is finished, will create a log file of all moves chosen, including the chess piece type. Log file will be stored in the log folder and be titled as "chess_game.txt"
- Once the player wins or loses, will call upon the endgame function to end the game.

## void move_piece(Move* move, Board *board)

Input:

- move: A pointer to the move to be made.
- board: A pointer to the current board state.

Process:

- This function uses the input information to move a certain piece to a certain location, if there's capture involved, the capture will be automatically done and recorded in the log file
- Covers how pieces will move in special cases (en passant, castling, pawn promotion)

## Move* computer_move(Player player, Board* board)

- Takes in the input of all the piece types on the board
- Uses a complex algorithm to decide the computer player's best move
- Designed to predict the user's moves several steps ahead
- Will output the computer player's best move after evaluation
- Invoked in the main function when it is the computer player's turn to move

## void generatemoves(Player player, Board* board, Moves *p_list)

- Takes in the input of all the location of the pieces on the board and what piece type, including their color, is moving
- Returns an instance of struct Move, containing an array of all valid possible moves for each piece on the board for specified player color

### void gentlemoves(int srcrank, int srcfile, int destrank, int destfile, struct Moves *p_list)

- This is where defensive moves can be initialized and stored

### void attackmoves(int srcrank, int srcfile, int destrank, int destfile, struct Moves *p_list, struct Board *position)

- This is where offensive moves can be initialized and stored

### void castlingmoves(int srcrank, int srcfile, int destrank, int destfile, struct Moves *p_list, struct Board *position)

- This is where castling moves can be initialized and stored

### int get_user_move(struct Move *move)

Input:

- Takes the input of the user as a 4 character array like “e2e4”.

Process:

- It breaks it down into the source rank and file and the destination rank and file
- And then the move_piece function to make the move.
- Prompts the user to enter a move in algebraic notation, parses the input and stores the move in the move struct.
- Has a one minute timer for the user to input their move
- If the user does not input a move within the one minute, the player that ran out of time will lose

Output:

- Returns an integer value indicating whether the move was successfully parsed and stored in the move struct.
- Returns 1 if successful.
- Returns 0 if the move is invalid.

### isLegal()

Input:

struct Board *board: a pointer to a Board struct representing the current chess board state.

- struct Move *move: a pointer to a Move struct containing the details of the move to be checked.
- struct Moves *moves: a pointer to a Moves struct containing the list of legal moves for the current player.
- enum Player playerColor: the color of the player making the move.

Process:

- Checks if the given move is legal for the given player, based on the current board state and the list of legal moves for the player.

Output:

- A boolean value of true if the move is legal, and false otherwise.

## generateLegal()

Input:

- struct Board *board: a pointer to a Board struct representing the current chess board state.
- struct Moves *moves: a pointer to a Moves struct that will be filled with the list of legal moves for the current player.

## extern void generateLegal(struct Board *board, struct Moves *moves, enum Player playerColor);

Input:

- board: A pointer to the current board state.
- moves: A pointer to a struct to store the legal moves.
- playerColor: The color of the player whose turn it is.

Process:

- Generates all legal moves for the player whose turn it is and stores them in the moves struct.

## int inCheck(Board* board, Player playerColor)

Input:

- struct Board *board: a pointer to a Board struct representing the current chess board state.
- enum Player playerColor: the color of the player to be checked for check.

Process:

- Prints out warning to computer or to player if previous move resulted in a check
  Input: Board with current status
- This function takes the input data of the previous move and evaluates whether that move resulted in a check
- It will print a warning on the screen indicating that a check has happened and you must resolve the check
- If the board is incheck, it will return the number of checks

Outputs:

- An integer value of 1 if the player is in check, and 0 otherwise.
- struct Board *board: a pointer to a Board struct representing the current chess board state.
- struct Moves *moves: a pointer to a Moves struct containing the list of legal moves for the current player.

- enum Player playerColor: the color of the player to check for game over.

## isGameOver()

Input:

- struct Board *board: a pointer to a Board struct representing the current chess board state.
- struct Moves *moves: a pointer to a Moves struct containing the list of legal moves for the current player.
- enum Player playerColor: the color of the player to check for game over.

Process:

- Checks if the game is over for the given player, based on whether they have any legal moves left.

Output:

- A boolean value of true if the game is over for the given player, and false otherwise.

## void draw(Board *board)

Inputs:

- When it is called upon by the main function, it will print out the 8 by 8 chessboard created through a 2D array
- struct Board *board: a pointer to a Board struct representing the current chess board state.
- It will also print out all the piece types and their location on the board
- Uses a "for loop" to print out the information

## void endgame()

Input:

- enum Player currentPlayer: the color of the player who lost the game.

Process:

- Will be invoked in the main function when the game ends or in function isLegal if an illegal move was performed
- Announces the result of the game and tells the user who wins the game
- Prints our white is the winner or black is the winner

## void recordMove(struct Move *move, enum Player curr_player, FILE *fp)

Inputs:

- struct Move *move: A pointer to a Move struct representing the move to be recorded.
- enum Player curr_player: An enum Player representing the current player.
- FILE *fp: A file pointer to the log file to which the move will be recorded.

Process:

- Will print a message that will notify the user that the move has been recorded
- If the move can not be recorded due to an error, it will print “incorrect player information”

Outputs:

- Records the move to the specified log file.

## int piece_value(enum PieceType t_piece)

- Gives each piece type a value so the computer ai can know how important each chess piece type is
- This will allow the ai to make intelligent decisions when playing against the player

## int evaluate_move(struct Move *move, struct Board *board)

- This will evaluate the overall value of each move
- The value calculated is from the difference between the piece moving and the value of the piece it is trying to capture

Input:

- struct Move *move: a pointer to a Move struct containing the details of the move to be evaluated.
- struct Board *board: a pointer to a Board struct representing the current chess board state.

Outputs:

- An integer value representing the score of the move.

## struct Move *select_best_move(struct Moves *p_list, struct Board *board)

- After the ai evaluates what move is the best to perform, this function allows the best move to be returned to the main file
- Function also accounts for predictability so it does not always choose the best move. (Will choose between several of the best moves)
- Invoked in the main file when it is the computer’s turn to move

## int reload(FILE *log, struct Board* board)

Inputs:

- FILE *log: A file pointer to the log file from which to reload the game state.
- struct Board *board: A pointer to a Board struct representing the current state of the chess board.

Process:

- Used to store and record log files
- Can be be used to restore a game state if the user never finishes the game
- Reads all moves to see if they can be stored in the log file (Will notify the user if there is an error with storing the moves)

Outputs:

- Returns an integer value indicating the success of the reload operation. If the operation was successful, the function returns 0. Otherwise, it returns -1.

## void sighandler(int signum)

- Used to notify the user if they run out of time during their turn

## void print_moves(struct Moves *moves, struct Board *position)

Inputs:

- struct Moves *moves: A pointer to a Moves struct representing the list of moves to be printed.
- struct Board *position: A pointer to a Board struct representing the current state of the chess board.

Process:

- When invoked, the function will print out all the possible moves that the current player has
- Prints the details of the move in the format of "<piece name> from <source location> to <destination location>" (to console)

Outputs:

- Prints the list of moves to the console.

## void initialize(struct Board *board)

- struct Board *board: a pointer to a Board struct representing the current chess board state.
- When invoked by the main function, it will print out the chessboard, setting up with all the chess piece types in the original locations

## debug()

Input:

- struct Board *board: a pointer to a Board struct representing the current chess board state.

Process:

- Prints debugging information about the current board state.

Output:

- An integer value representing the number of pieces on the board.

## Function: int straight_moves_count(struct Board *position, int dir, int rank, int file)

Inputs:

- struct Board *position: A pointer to a Board struct representing the current state of the chess board.
- int dir: An integer representing the direction of the move (either 1 or -1).
- int rank: An integer representing the rank of the starting position of the move.
- int file: An integer representing the file of the starting position of the move.

## extern int print_files();

Process:

- Prints the names of all saved games in the current directory.

Outputs:

- Returns an integer value indicating the number of saved games.

# Back matter

## • Copyright

## • Error messages

Illegal Moves

- "There is no piece at selected location"
- "That piece cannot move to selected position"
- "That piece is not yours"

## • Index

## • References

http://www.wachusettchess.org/ChessGlossary.pdf
https://cdn.shptrn.com/media/mfg/1725/media_document/8976/Imp_ChessR.pdf?1401227342