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

Computer Chess

1.1 Usage scenario

```
Welcome to Minor Carlson Chess!
Please select a gamemode (1 for Player vs Player, 2 for Player vs AI): 1
Please select side for Player 1(White or Black): White
White side has been selected for Player 1.
The game will now initialize.

  +----+----+----+----+----+----+----+----+
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

Player 1, input a move: E2E4

  +----+----+----+----+----+----+----+----+
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

Player 1's pawn has moved from e2 to e4.
It is now Player 2's turn.
...
(Game will continue until one of players make illegal move or checkmate)
Player 1 (White) wins!
Press 1 to go back to menu or 2 to rematch: 2
Game will restart...
```

For our program, the normal usage scenario involves the user loading our program up which leads to the main menu with options for the user to select. After a selection is made then the user will be given a choice to choose which color the player would like to be. Then a board with pieces will be displayed and white goes first. After each move is made a log is generated and an updated board will be created. Moves are exchanged until a player has a checkmate then the game ends. The player gets to choose to play again or return to the menu to select another mode or exit the program.

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

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

- Choosing difficulty of AI
- Computer vs. Computer gamemode
- Timer on the board to see remaining time left for turn

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.

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
- 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(Move* move)

- Takes in the input of the instance of struct move
- 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

Move* computer_move(Player player, Board* board)

- Takes in the input of all the piece types on the board
- Uses the minimax algorithm to decide the computer player’s best move
- Designed to predict the user’s moves six 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

Moves* findAllMoves(Player player, Board* board)

- 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

bool isLegal(Move* move, Board* board)

- The value is true when the move is legal, and false otherwise (prints out the error that the move is illegal when the value is false)
- Takes in the input data of the user (through the main function) and evaluates whether the move is legal or not.
- If the move is not legal, endgame() will occur and the game will end

bool inCheck(Board* board)

- Takes in the input of the pieces in the chessboard's location
- 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 a true value, otherwise the value will be false

void draw(Board *board)

- When it is called upon by the main function, it will print out the 8 by 8 chessboard created through a 2D array
- 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()

- 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

Void recordMove(Move* move, Player player)

- Records the move from that has been made, including the piece type and which side made the move
- 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"

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