Chess Project Technical Documentation

Jesse Masciarelli 6243109

COSC 3P71

December 20th, 2021

Brock University

Contents:

A Brief Introduction

The Game of Chess

This project is an implementation of the classic game of chess – a 2 player game where each player has 16 pieces on an 8x8 board, with the goal of capturing the opponents King piece to win. Different pieces have different possible move sets, and a variety of special rules make the game both a challenge to play and program.

The implementation of this project allows the game to be played in the standard output, either as 2 humans playing against each other, or as 1 human playing against the computer. The detailed instructions are included in the Instructions/README section of this report.

Main Components

The main components, which can also be referred to as the main goals of this project which will be covered in this report include:

- Implementing a chess game that follows the basic rules of chess, by creating a board to be populated with pieces that are able to determine their move sets
- Making the chess game playable by two humans competing with each other
- Making the chess game playable by 1 human vs the computer
- Allowing the human to control the difficulty of the AI
- Creating a heuristic function that allows the AI to make “smart” moves

Instructions/README

Standard Procedure

Within the top layer of the submission folder, there is an executable jar file called FinalProject.jar which can be used to run the project. If for any reasons there are issues with the executable jar file, then the project can be compiled and run through the following path to the main class: *Chess -> src -> game -> ChessGame*. ChessGame is the main class of the project.

The game will be played in the standard terminal output, and upon running, the user will be presented with a chess board and clear instructions on how to start the game in either mode.

First, you will select the game mode: As the instructions say, you can type 0 and press enter to start a human vs human chess game, or type 1 for human vs computer.

Once you’ve chosen the game mode, if human vs human, player 1 will be prompted to make the first move. All moves made by humans will be done in 2 steps: First the player will select the piece they’d like to move using the alpha numeric coordinate of that piece on the board. Once

you've pressed enter and confirmed a legal piece selection, you may then choose the destination of your move using the same syntax.

Move Example: If a player wants to move their pawn from 'A6' to 'A4' on the board, they will first enter A6 as the first choice in their move and they will click enter. If this is a legal piece for them to move, then they can choose the destination by typing A4 and pressing enter. The move will then be made, and the board will be presented in it's altered state.

In a human vs human game, the exact same process will be presented to player 2, and the game will alternate turns and moves until the game is over.

In a human vs computer game, the human player will first be prompted to choose the desired depth of the AI algorithm. The larger the depth chosen, the "smarter" the AI will be, though it is recommended to use a depth of 2-4 for a good balance, as larger depths take far longer for the AI to compute its moves. After choosing the depth, the human will then make human turns identically to as in the human vs human mode and wait after making their turn for the computer to compute its move choice and make that move. The human does not need to do anything other than wait after they've made their move in human vs computer mode.

Special move cases (piece promotion, check detection, etc.) will be computed automatically by the program and will output anything that happens to the user. These special cases will be explained further in the "Overview of the System" section of the report.

You are now ready to play a game of chess in either mode. Enjoy.

Altering The Starting Board

While the standard procedure is enough to run the program and play the game of chess, you may want to make alterations to the starting board set-up. If this is case, the following will detail how to make changes to add/remove/rearrange pieces on the starting board before the game starts.

Changes to the starting board will be made in the Board.java class, within the game folder in the project. Specifically, pieces are created and added to the board within the initialize() method in the Board class. Any pieces you wish to add can be added using the following line within the existing pieces being created, making changes to the underlined portions as desired:

gameBoard[*x coord*][*y coord*] = new *Piece Type*(new int[] {*x coord*, *y coord*},*player number*);

Note that the x coordinate corresponds to the x coordinates shown on the game board, but the y coordinate must correspond to the integer value of the letter shown on the game board. For example, a piece located at space "B3" will actually have x-y coordinates as [3,1]. 3 is the x coordinate which comes first, and B is equivalent to 1, as A is 0, B is 1, C is 2 etc. Piece type can be any of Pawn, Rook, Knight, Bishop, or Queen. Player number can be either 1 or 2.

An Overview of the System

Key functions

The program utilizes numerous key functions to allow the game of chess to run properly, and for the best understanding of every function and their purpose, see the comments within the project code. However, I can detail some of the major key functions and their uses below:

- **Board.Display():** Perhaps the most important part of the entire chess program is being able to translate the stored array of pieces into a visual representation that the human player can see and interact with. That translation occurs with the .display() function.

- **ChessGame.minimaxAB():** This is the main artificial intelligence algorithm that powers the computers moves and allows it to test and score all of the possible board states that arise. MinimaxAB not only uses the minimax algorithm, but it implements alpha-beta pruning to reduce the computation time of finding that ideal move. To score the potential moves, minimaxAB() uses the following heuristic function method.

- **Heuristic.evaluateBoard():** This function is the main method behind evaluating board states for the minimax algorithm. It computes a board score based on 3 different heuristic evaluation methods (explained in the Heuristic section of the report) and returns it to the calling algorithm. This function integrates piece checks, and calls to helper methods to act as the decision maker for the minimax algorithm.

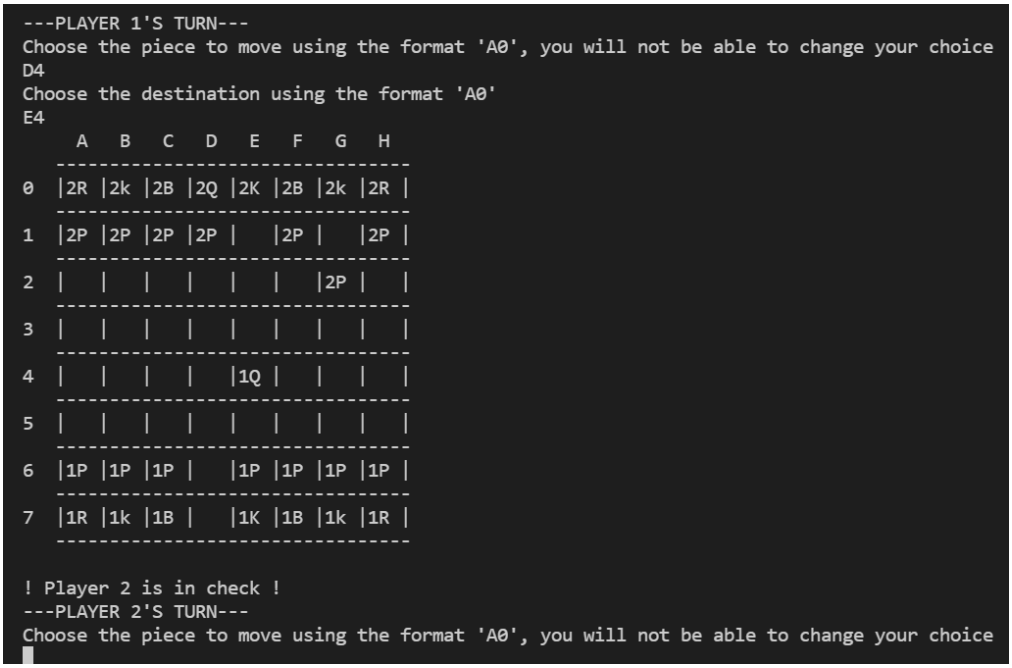
- **Piece.genPossibleMoves():** This method, in which each extension of the abstract Piece class has its own version of, is essential to being able to play the chess game and move the pieces around the board. Given a certain board state, the genPossibleMoves methods will allow the given piece type to create a pseudo board of true/false values that dictate where it can move. This true/false board is used to ensure that players choose legal pieces and make legal moves and is essential to the successful operation of the program.

- **ChessGame.makemove():** This method is what allows the game to progress and allows the board to change states each turn. Once a player is able to choose and dictate their desired move, the program must ensure that that move is carried out correctly to present the correct board to the next player and keep the chess game correct. The makeMove() function makes sure to read from the existing board, before creating a new board with a given move transformation made and is essential to the players being able to continue a game for any number of moves.

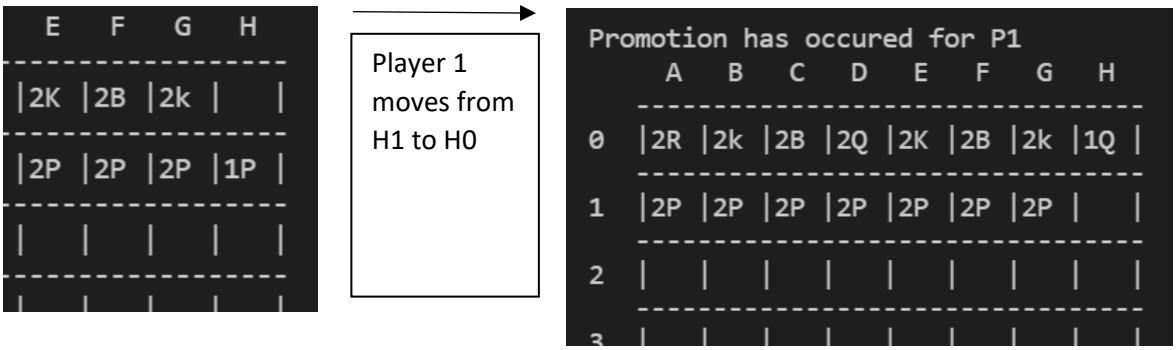
System Features:

Many of the features of the system are made obvious by running and using the application, though there are some notable features and improvements made that may go unnoticed. Below, some of the interesting system features are highlighted and demonstrated.

- **Check detection:** When either player's king ends up in 'check' (their king can be hit by an enemy), the players will be notified that a check has occurred and will let you know which king is in check. As you can see in the example game below, player 1 has just moved their queen into a position which can hit the player 2 king, and before player 2 is notified of their turn, they are informed that their king is now in check.



- **Piece promotion**: An important part of the game of chess is piece promotion, though this doesn't happen regularly in every game so it is a feature that may be overlooked. Piece promotion occurs when a player is able to move their pawn to reach the opposite end of the board. If they manage to do this, they are able to upgrade that pawn to any piece of their choice. In the case of my piece promotion implementation, all pawn promotions become queens for simplicity. The piece promotion occurs automatically when either player reaches this state in a game and the new board state will be automatically displayed in order to keep the flow of the game. Players will also be notified that promotion has occurred. An example of piece promotion occurring can be seen in the example scenario below.



- **Adjustable AI strength:** It is a feature of the program that it allows the human player to choose the search depth of minimax algorithm, and thus the difficulty of the AI in the game. Similar to playing a game on easy or hard mode, the user is prompted before a human vs computer game starts to give the desired depth. This freedom allows a human to play against "different" computer opponents, and also gives control over how long an AI will take to compute its turn.

- **Pawn Movement:** The pawn is a unique piece in chess because it has a different set of moves depending on where it is on the board, and it attacks in a way that differs from its regular movement patterns. Within this implementation, the pawns obey the rule that allows them to move 2 spaces (as opposed to the typical 1 space) on their first turn only. This pawn implementation also allows the pawn to attack diagonally, which is the standard for pawn attacks in chess.

Heuristic Evaluation

The heuristic function that has been implemented to score the state of the game board is broken down into 3 main evaluation components. The combination of these components allow the AI minimax algorithm to make the best possible move, determined by the collective score obtained by these 3 evaluations. The main principles behind these evaluation methods were inspired by articles in a chess programming forum, cited at the end of this report. The 3 evaluation methods and their function are:

Material Advantage

The material advantage evaluation is the simplest of the 3 evaluation methods and gives preference to moves which will result in the AI having a "stronger" set of pieces on the board than the opponent. The way that this evaluation works is by assigning score weights to each type of chess piece, and then totalling the weights for each player, adding them to the score if

they are friendly pieces and subtracting from the score if the are opposing pieces. After experimentation and research, the weight values assigned within the heuristic are as follows:

Pawns: +2 score per friendly pawn, -2 score per opposing pawn
Rooks: +12 score per friendly rook, -12 score per opposing rook
Knights: +6 score per friendly knight, -6 score per opposing knight
Bishops: +6 score per friendly bishop, -6 score per opposing bishop
Queens: +18 score per friendly queen, -18 score per opposing queen

For example, if player 1 has 3 pawns and 1 knight their positive material advantage would be 12 points. If player 2 had 2 pawns, then player 1 would have a material disadvantage of 4 points. This means that in total, player 1 has a material net advantage of 8 points.

Centre control

Centre control is the second method of heuristic evaluation implemented and is defined as "a chess strategy to control or occupy the center and extended center of the board, which gains space and allows pieces fast access of most of the board areas, and more movement and possibilities for attack and defense." (1) To evaluate the central advantage on a given board, the heuristic scans the most dominant central rows on the board and increases/decreases the score based on the pieces that occupy those spaces. After experimentation and research, the following score breakdown was given to the specified board rows:

Row 4 and 5 (The 2 centre rows):
+4 score per friendly piece in these rows, -4 score per opposing piece in these rows

Row 3 and 6 (The 2 rows outside of, but closest to centre):
+2 score per friendly piece in these rows, -2 score per opposing piece in these rows

Using this centre control evaluation will allow the AI to claim advantageous positions on the board and will act in the interest of early offense against the opponent. On the flipside it will also aim to prevent the enemy from gaining these central positions, which will give the AI more control over the long-term outcome of the game.

King safety

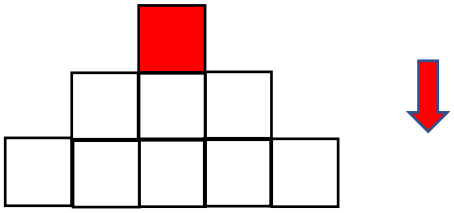
The third and final method of evaluation integrated into the overall heuristic is the degree of king safety that each player holds. King safety is concerned with the area on the board that surrounds the crucial king pieces, commonly referred to as the king zone. One common definition of the king zone is "squares to which enemy King can move plus two or three additional squares facing enemy position" (1). For the heuristic in this project, after experimentation the king zone has been defined slightly differently to provide for more

protection from incoming attacking pieces. The king safety evaluation occurs using a pseudo pyramid formation coming out from the king's position, scoring for the first row of 3 and the next row of 5 spaces in front of the king as follows:

3 spaces in front of the king (on kings attacking side)
+4 score per friendly piece in these spaces, -4 score per opposing piece in these spaces

5 spaces in next row in front of the king (on kings attacking side)
+2 score per friendly piece in these spaces, -2 score per opposing piece in these spaces

The definition of the king zone is easily understood in the following diagram, where the king is represented by the red space and the arrow shows the direction that the king attacks in:



Taking king safety into account during the evaluation process will ensure that the AI makes moves that represent the end goal of the game, to protect their king and attack the opponents. This overall goal evaluation, paired with the other 2 previously mentioned evaluation methods create a well-rounded heuristic that provides good competition for any human chess player.

Reference:

(1) *Chess Programming Wiki: Evaluation,* 2021, https://www.chessprogramming.org/Evaluation