Final Chess Project Report

Brock university | St. Catharines, Ontario

cosc 3p71 Introduction to artificial Intelligence: Final project

Azeel Jivraj & Micah Rose-Mighty

2021

**Introduction and Implementation Overview**

For the final project of the term we decided to work in a group of two. Our group members are Micah Rose-Mighty and Azeel Jivraj, both students currently enrolled in the COSC 3P71 Fall 2021 course. For the final project of the term we were tasked to implement a fully functional chess program that respects all the rules of chess including the movement of pieces, pawn promotion, check, checkmate, castling and en-passant. Our chess program was implemented in Java using the IntelliJ IDE. Our chess program includes numerous classes that pertain to the many different elements needed to run this sort of chess program. The main class of our chess program is the ChessGame class. This class initializes the variables necessary to run the chess program and also implements the main functionalities of our chess program. This class receives input from either the black or white chess player in the form of a coordinate to determine which piece is selected and eventually, how it is moved. This class also handles the production of a chess board and all the chess pieces in the correct location onto the console. The ChessGame class keeps track of natural order of the game of chess and prompts the respective player to select a move based upon the status of the board and the previous play, if any. Many of the other prompts made to the console are also handled in the ChessGame class including the announcement of a winner, once a player’s King is in check or has selected a move leading to the King being in check, when an illegal move has been selected, when the incorrect format of a coordinate has been entered and when a player selects an opposing or non-existent piece. The main class also sends a message to the console, prior to the chess game, that asks the user to enter a 1 if they would like to play human vs human chess or enter a 2 if they would like to play human vs AI chess. Another feature of the main class is the conversion of the coordinates chosen by the respective players. This is done using a method called convertInput which takes the player’s coordinates and converts it into a space on the chess board. If the move is valid, the board is then reproduced by the main class onto the console with the updated move and the corresponding prompt is then given to the console. The majority of the rules of chess are also implemented in the main class. Pawn promotion is handled by a method called pawnPromotion which informs the player that a pawn has reached the end of the opposite side of the board and the option of pawn promotion can be taken. Castling, en-passant and the initial pawn jump rule are also handled within the main class because they are considered special moves that can be taken in the game of chess that significantly impact the outcome of the game and pieces on the board. Within our chess program there is a Space class that represents an individual space on the chess board with given coordinates. This Space class determines if there is a chess piece on that space, and if so, what colour and type that piece is. Also, within the space class are the methods to set and remove a piece from a space on the board. This works directly with our Board class which implements our 8x8 2-D array that resembles the board used in chess. This class creates the board and places the pieces in the correct location given the status of the board. The selection of the initial piece is also handled within the board class given the standard setup of the game of chess. The Board class also adds up the score of both respective players and finds the difference, which determines who is winning the game at the time. This board evaluation will then be used to assist the AI in selecting the optimal move. There also exists an abstract class called Piece within our chess program that gives each respective chess piece a type, value and colour that corresponds with the known rules and board setup of the game of chess. The Piece class also verifies if the chosen move for each piece is valid, if a piece has moved and if an initial pawn jump is legal or has occurred. The AI portion of the chess program is facilitated using the resources of many of the classes included within our chess program. Within the main ChessGame class, the user is given the choice of playing against another human or against the AI. In the event that the AI is chosen, additional classes that facilitate the operation of our implemented AI are activated. The main AI class is a class called Move which is in charge of the AI’s evaluation of the board and eventual move decision based upon the result of our alpha-beta minimax algorithm. This Move class is assisted by another class called chessMove that stores the starting and ending coordinate of a chess move and the evaluation of that move. The details of our implemented AI system will be discussed further in subsequent sections. The remaining classes deal with the intricacies of each of the pieces in the game of chess. The classes are called Pawn, Queen, King, Knight, Bishop and Rook each pertaining to the respective pieces known to be used in the game of chess. Each individual chess piece class extends the abstract class Piece as it has very similar functionalities and attributes being that each chess piece is in fact a piece. Within each of the 6 classes implemented to represent the different pieces of chess there exists a method called validPath, that is different for each of the pieces, that returns true if the path selected by the player is valid given the rules of chess and the status of the board. One piece class that differs from the rest is the King class. This class has a method called isChecked that checks if the King of either team is in check by any opposing piece. This method is a helper method used in the isCheckMate method of the main class that returns true if the King of either team is in Checkmate. All the previously mentioned methods and classes work together and enable our fully functional chess program to run and satisfy all requirements exceptionally.

**Instructions for Compilation and Operation**

Our chess program was implemented in Java using the IntelliJ IDE and it is advised that the most recent stable environment is used when compiling, running and operating the program. Also, it is advised to use the development kit (JDK) openjdk-15 or higher due to the implementation of the program. Once all that is in place simply click run and you will see a sentence appear on the console that says:



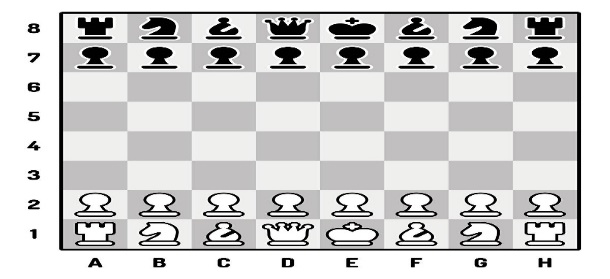
Entering a 1 will activate the human vs. human option of our chess program and entering a 2 will activate the human vs AI option. Whether the player vs. player option or player vs. computer option is selected, the game operation remains the same.

Once a game option (and depth value when applicable) is selected you will see the following prompt:

A screenshot of a computer

Description automatically generated with medium confidence

Here, you can see our initial representation of a chess board and all the pieces using various characters that resemble the distinct attributes of a chess board at the beginning of a chess match. The vacant spaces on the chess board are represented by “[ ]” and the spaces filled with a certain piece are denoted with a code that tells the player the colour and type of each piece. The pieces that begin with the number 1 are the black pieces and the pieces beginning with the number 0 are white pieces. The codes for each type of chess piece goes as follows: P = Pawn, R = Rook, Kn = Knight, B = Bishop, Q = Queen and K = King. The program starts with the chess board and all pieces placed in the standard starting position and the Initial player (user) is prompted to enter the index of the piece desired to be moved. As mentioned in the prior section, the input for piece selection and movement must be done in the form of coordinates. Our coordinate system matches that of a standard chess board meaning that letters A to H represent all columns and numbers 1 to 8 represent all rows. This concept can be easily visualized here:



The coordinate system within the chess program follows the exact methodology of the diagram seen above but an example will be shown to further clarify the concept. Within the image of the console on the previous page we see at the bottom of the image a prompt for the White player (standard player with first move in chess) to enter the index of the desired piece. The prompt also gives an example coordinate to use as reference when entering other coordinates. For the purpose of this example, we will be moving the piece on the example coordinate B2 (White’s second pawn from the left) up one space to B3. The result is seen in the image below:

Text

Description automatically generated

Upon selecting a valid piece, the console output’s the name of the type of piece selected and asks the player to enter a “x” if they would like to reselect a different piece. To select a destination space for that selected piece use the same coordinate format used initially. Once the move is validated and completed we see a reproduction of the previous board appear but with the required updates (Pawn moved from B2 to B3). Finally, the black player is prompted to enter the index of their desired piece in the same coordinate format as previously mentioned and thus the game has begun as pieces are sequentially selected and moved by the appropriate player until a winner has been found. In the case that the user selects the option of a human vs AI game, the user will also be prompted to enter an integer value for the depth of the alpha-beta minimax search tree. This depth value will act as a difficulty meter for the chess game because as the depth value increases, the moves made by the AI will become increasingly better since the AI will choose the optimal outcome of more moves. A depth of 5 is recommended and although that may be a large depth when making a search tree for the potential moves of a chess game, the AI’s moves will be rapidly evaluated and executed by our implemented AI system, thanks to alpha-beta pruning, simulating the moves of a skilled chess player.

**AI Description**

Our AI functionality begins in the main class called ChessGame in which the user is prompted to enter either a 1 or a 2. The entrance of a 1 will allow the user to play a human vs. human chess game with another user, while the entrance of 2 will allow the user to play against our implemented AI. Regardless of the selection, the white player (user) is chosen to move first in standard chess fashion and our AI is implemented to always play as the black player, allowing the user to play first and the AI to react to this play. When the player vs. computer option is selected, the user will be asked to enter a depth value. This will be the depth value used when formulating our alpha-beta minimax search tree that eventually determines the AI’s move selection. The main class that is in control of the AI’s decision is our Move class. Within this move class, there is a method also called Move that takes in a board, depth value, alpha value, beta value and a player value (typically 1 since the AI is the black player) and returns an optimal chess move for the player given the current board evaluation. The actual move selection is handled in our alpha-beta minimax method called alphaBeta which takes the same parameters used in the Move method but also takes in another parameter that resembles the current optimal chess move. Our alpha-beta minimax method uses the pre-determined values of each piece to determine what move to make based upon the board evaluation. This method creates an alpha-beta minimax search tree of the user-inputted depth (typically 5) that contains all possible actions and reactions within the current chess game. The AI then evaluates the optimal move to take in the shortest amount of moves and that is the move selected. The optimal move would be the move that leads to the highest score for that player, that could be the move leading to the capture of the highest opposing piece, avoiding the capture of a black (AI) piece, protecting the black king among other highly valued chess moves. Typically, a search of this magnitude would take extremely long to be facilitated by the computer due to the infinite moves and countermoves available in a chess game, but the addition of the alpha-beta property to the minimax algorithm makes the AI move selection happen faster. Once a good move has been found the remaining moves that are always worse than the good move are not considered. Our AI is implemented to satisfy the exact requirements of our final project and our implemented heuristic is a simple yet powerful evaluation. This is only possible by using alpha-beta pruning as many irrelevant branches of our search tree will not be traversed, allowing our AI to derive the optimal solution in reduced time. Our recommended maximum tree depth of 5 enables us to have a simple heuristic as it is believed that within 5 moves at least 1 opposing piece can be captured and the move towards the optimal capture will most likely be the move selected by the AI.