Term Project
COSC 3P71

Fahad Arain

INTRODUCTION

Chess is a board game but unlike other board games chess is arguably the most famous board game made around the 6th century AD. Chess is a strategy-based game in which a player controls a total of 16 pieces, there are different moves each piece can make. The main object of chess is to eliminate the opponents king while taking out the other pieces in the process. For this project we implemented a chess game this was done using Java, the chess game that can be played in two different ways when it runs you can either play the game versus another human or you can play the computer (AI). To play the game the player first is asked to enter an input to select the piece that they would like to move, once they have selected the piece the player is then prompted to input the coordinates of the location they would like to move to.

COMPILING, RUN, & OPERATE

When it comes to our project, we have a fair number of classes that are used in compiling the but the Chess class which is our main class is the class that needed to be run first. The main class when run gives the user an option to choose what game mode, either to verse AI or another human and this is crucial option which is why it is to build and run first. When the user picks verses human there is a slightly different process as to how the program is run compared to when the user selects verse AI. When the user selects the PvP it asks the player to make a move and so the pieces classes (king, queen, pawn, rook, bishop, empty, knight, piece) are run along with the player class that identifies who is the current player. Every class is run besides the Node class and the only method that is not used from the main class is the alphabeat(). On the other hand when the user selects the option to play against the AI all the same classes are used as the PvP option only when it is the AI turn to play we also use the node class and the aplhabeta() method. In order to run the code all these classes need to be present and once you have run the Chess class

which is also the main class everything else will fall into place.

IMPLEMENTED SYSTEM

Public Class Chess

CheckGame(): Check game is a method that checks to see if the king can make a move using the method cankingmove() which returns the number of pieces checking the king, it lets the user know if the king can move without dying and if not it will end the game.

canKingMove(): This method checks if the king can move if an only if the method returns 0. The number returned by this method is the amount of pieces that are checking the king this method uses the boardchecker() method to find the number of checks.

boardChecker(): this method has a variable called checkCounter which is updated each time a piece in the board can attack the king which is done using the coordinates of the king and with the makeMove() method that uses the coordinates and iterates through the board and finds if anything is attacking the king.

checkForPromotion(): This method is used to promote the pawn to a different piece once it has reached the opponents end. This is done by checking if the players pawn is either on the end of the board meaning that the 'Y' value is either 7 or 0 and once that condition is met, we also need to see if the piece in question is a pawn. After the two requirements are met we then prompt the user to select a number from 1 to 4 with each number corresponding to a different pieces they can upgrade to.

editBoard(): This method stores the location of both the kings in a separate array with the first index as the X coordinate and the Y as the second. This is also where we update the board when a legal move is made by first making the spot at which the piece left to empty and then replaces the piece at which it is supposed to move. Conditions such as the king's safety, if it is not a pawn at the end of the board, and if the piece it is replacing is a different colour need to meet before the editing of the board can commence.

buildBoard(): this method is a simple method that initializes everything. It starts off by taking a 8x8 2d array and initialing everything to '-'which signifies an empty

spot. Once this is done, we now hard code the pieces into the locations for both players. The array holds classes for each piece at their allocated spots.

getPiece(): This method used to prompt the user to select the piece they would like to move. This is done by first asking them the x coordinate of the piece and then the y coordinate, once they have entered the values, we now have the selected piece.

getMove(): This method follows the getPiece() method because this method now asks the user to enter the desired location of the move they would like to make by taking the x and y coordinates.

movePiece(): This method works together with the getPiece() and getMove() method in order to move a piece. If the makeMove() method returns true meaning that the move is legal it then goes into the editBoard() to confirm the move other wise an error is shown where we tell the user that the move they are attempting to make is illegal and to re-input the coordinates.

alphaBeta(): This method is where the alpha beta pruning takes place which is done for the AI component. This method works with the node class in order to make a tree of possibilities where the maximizing player wants to increase its score while decreasing the opponents. This is done by either getting the max value of the leaf node or vise versa. More information on Alpha Beta is discussed under the Heuristics title.

printBoard(): This method basically sets up board that we see in the terminal tab once the game is run and is used after each move is made to print the new state of the board.

Public Class Node (AI)

Node(): Holds the board state, the current player who is making the move, the board score, and the children which is the tree of possible moves.

expandNode(): This method checks if the possibilities are not there it will run the findMove() method which starts to find the next moves that can be made.

findMove(): This method iterates through out the whole board and finds each piece and using a helper method called generateMoves() works in finding the best move.

generatMoves(): This method uses the piece selected in the findMove() method and runs it through makeMove() which returns if the move is legal and if so this move is then added to the tree of possible moves.

editBoard(): This method is different from the last editBoard as it is not officially making the move but rather creating a copy of the board in which it makes the move thereby simulating a move.

getChildren(): Returns the children(possible moves) and if there aren't any it will create them.

evaluateBoard(): This is the Heuristic method that works together with everything to make the moves for the AI and will be discussed in more details under the Heuristics headline.

Global Method

makeMove(): This method is used by each piece and is unique to each piece and is used to see if the desired move is legal depending on the piece and the path.

Public Class Player

Player(): Holds the player number either player 1 or 2 and what the associated color of the player is.

Public Class Piece

Piece(): Holds the rank meaning the score of the piece in question, the type of piece, and the color of the player.

Public Class Empty

Empty(): Holds the rank the type('-') and also implements the makeMove() method.

Public Class Bishop

Bishop(): rank, type, color.

makeMove(): This method for the bishop checks to see if the absolute value of the coordinates of the new position minus the old coordinates are 0, this is because if we are to move in a diagonal direction the math would always have the difference be 0 because we are subtracting 1 from both the y and x value each time.

checkDiagonal(): This method checks the path of the bishop it each diagonal direction bottom right, top right, bottom left, and top left. This method is used

in the make move method to see if the path is clear.

Public Class King

King(): rank, type, color

makeMove(): checks each position to see if the same color piece is there and if the move is of distance 1.

Public Class Knight

Knight(): rank, type, color

makeMove(): checks if the spot to move doesn't have the same color piece, if not, it has to check if the absolute value of the difference of coordinates from where it was and where it is going multiplied together always results in the value 2 as we always are moving 2 indexes forward and one to the side.

Public Class Pawn

Pawn(): rank, type, color

makeMove(): restricts the pawn form any other movement but forward and move two blocks if it's the first move. If there is a different color piece diagonal to it, it can capture it.

Public Class Rook

Rook(): rank, type, color

makeMove(): this checks if the absolute value of the difference between the desired spot and the current spot is either moving the Y value or X not both and if the path is clear.

Public Class Queen

Queen(): rank, type, color

makeMove(): This method implements the horizontal movements we used in the bishop class and vertical and horizontal we used for the rook while also checking if the not is empty or has the same color piece.

HEURISTIC

The classes that has the most important role to play in the heuristic of this chess game would be the main class "Chess" and "Node". We will first look at the node class which iterates through each piece on the board and then sends each piece to a method called generate moves which iterates each position that certain piece can move to and if its legal it creates a node and adds it to a arraylist called children that holds the moves. The heuristic method that we use is

done in the evaluate board method where we see the score of the positions this certain piece can move and what would be the score of the board when it does this. Each empty space has a rank, and this rank determines the best move with a higher rank resulting in a better position this is all implemented when we build the board by increasing the rank for the empty spaces as we get closer to the middle of the board. Chess is a strategy game in which the best position to be in is to have control over the board hence, as we get closer to the middle, we get a higher rank. These possible moves are also done in a clone of the real board which means that we have simulated each move, but we have not confirmed. Once this arraylist is made of the possible moves we then implement the alpha beta pruning method. Alpha beta pruning starts off with having one player be the alpha player or maximum player and the other be the beta or minimum. The players both start with the worse score for themselves, the alpha starts with negative infinity and beta starts with positive infinite. Each move the player who is either maximize their score or minimize goes through the tree of possibilities and sees after selecting a move with the score associated to it would get them closer to victory while at the same time get the opponent away from winning by traversing the tree and seeing the possibilities in the future they could make after a certain move is made. This method is found in the "Chess" class as alphaBeta().

CONCLUSION

In conclusion we have programmed one of the most famous and old board games using our own style of coding and with this PvP game we have also made a mode for PvAI implementing a AI using the alpha beta pruning strategy. With a friendly UI jumping into the game is as easy as pressing a button. Enjoy the game and good luck!

REFERENCES

- Alpha Beta Pruning slides from class slides
- Chess 101: Who Invented Chess? Learn About the History of Chess and 3 Memorable Chess Games (https://www.masterclass.com/articles/chess-101-who-invented-chess-learn-about-the-history-of-chess-and-3-memorable-chess-games)