Chess AI – Using Mini-Max And Alpha Beta Pruning

- *INTERFACE:* The interface is desgined using Jframes (that is Java Frames) and JPanels. The UserInterface class is responsible for the user interface. What this class does is, it extends Mouse listeners so as to detect the motion of the mouse as you will be moving the mouse to move the pieces around the chess board.
- *EVALUATION FUNCTION:* The Evaluation Function used for the chess AI is a simple one as described in the chessprogramming.org(https://www.chessprogramming.org/Simplified_Evaluation_Function) . We are basically using piece square tables which will just first give each piece a particular favourable(+) or unfavourable (-) value depending upon the place where it is standing. This was due to the place at which it is standing. More over each piece itself has a particular value independent of the place where it stands (rateAttack() and ratePieceAvailability() functions). Now since this still nor sounds perfect like relatively both the sides may have similar arrangement and similar pieces. So to distinguish among them what we do is we check the moves available to each side and for each possible move we give 5 points (the factor 5 comes from the fact since each move of our chess AI is represented as an array of 5 characters so just using the length function to get the string length of the total possible moves so in short giving each move a value of 5 points). Now the thing is how am i considering the case when either checkmate or stalemate will occur, both the cases are handled in Rating.java file from lines 141-147.
- *MINI-MAX AND ALPHA-BETA PRUNING:* In the Chess AI built we haven't made any changes in the basic mini-max algo and alpha beta pruning. But we have written an another documentation about what changes we could have made in the minimax and alpha beta pruning so as to improve performance.
  - *MINI-MAX*: The mini-max algorithm is an algorithm that belongs to the class of algorithms called backtracking algorithms. In this type of algorithms we basically start finding solutions for all possible cases

and on our way whenever a particular possible candidate fails to satisfy our constraints(here, the safety of king) it is removed from the solution space eventually we lead to a set of possible soultions. Now in mini-max algo we go like backtracking assuming that the opponent player also chooses the best possible. Mini-Max algorithm can be assumed like a tree where alternate depths are either of the maximizer or of minimizer. Like we will have a maximizer layer followed by a minimizer layer and that followed by a maximizer layer again and so on. Similarly the other way round. This maximizer and minimizer here represent nothing but the two players. And as said above both the players will try to maximize their score that is they will try to minimize the other person’s score. Therefore applying maximizer and then minimizer. And since we are evaluating all the cases the mini-max tree will grow enormously fast, so to increase the performance Alpha-Beta pruning was introduced. That means in mini-max algorihtm going one depth down means evaluating all the possible moves for each configuration present at that level, and then for each of these possible configurations and possible moves we will evaluate each one of them using the evaluation function as described above and then it will be followed by either choosing the maximum value(maximzer) or the minimum vlaue(minimizer).

- *ALPHA-BETA PRUNING:* Alpha beta pruning is an improvement over the mini max algorithm. In this we somehow make our algo to remember the previous obtained result and compare it with the range of solution we will get for the current branch and if they are totally out of the grid then we can leave the current branch that is prune it. This is the essence of alpha-beta pruning. Let say i was currently at the maximizer node that is the computer and i have to maximize my score. And lets say this node has two branches with the root of these branches as 5 and the second one is unknown. And since i have to maximize my node i can say that this node will either be 5(if the other node’s value comes out to be lesser than 5) or greater than 5(if the other node’s value comes out to be greater than 5). Now consider that other node has certain number of children and in between while evaluating the node’s children we find the value to be 3. And since our original node was maximizer then this node is going to be a minimizer that is tries to minimize the score of the computer and since one of its children has a value of 3 that means this node is gonna to have a value either equal to 3 or lesser than 3(in case other children give a value lesser than 3). But since we originally found that our required node can have a value of at least 5 then why it will go for a value of 3 or lesser than that. That basically means that we can prune other branches of the required node's child. This eventually leads to the decrease in required computation time of the cpu that is as a programmer we are happy :p

- *CODE EXPLANATION*: The main code lies in the AlphaBetaChess.java file. The code begins with creating an instance of the UserInterface adn doing all the stuff related to the UserInterface. The interface is built like it will first ask the user whether he wants to play frist or wants the AI to take the first go. Depending on whatever the user choose our built chess engine will call a function alphabeta once the turn of pc arrives. This function is basically for the alpha beta pruning and the minimax algorithm. The flipBoard function is nothing (as the name suggests will flip all the pieces to their counter piece that is all blacks will go white and all white will go black) but just flips the board. This helps in making the code shorter and easy to read as by doing this we don't have any need to write separate function for both black and white. MakeMove function is responsibel for making a move on the chess board. SortMoves is basically a function that will sort my moves according to the score they will lead to so as to improve cpu utilization (one of the advancements we thought). PossibleMoves is a function responsible for making an array of all the possible moves in the current state of the chess board. Corresponding to this function we hav some another function like possibleR, possibleP, possibleK, possibleB, possibleQ, possibleA responsible for listing the possible moves of rook, pawn, knight, bishop, queen and king respectively. KingSafe function will basically tell whether in the current configuration whether my king is safe or not. Rating.java and UserInterface.java files are already explained.

- *"What About the two modes we said"* : Here is the explanation, on heavily studying the way we play chess adnd watching some videos about how normal people percieve chess we arrived at a conclusion that an average human will consider at max 3-4 steps forward in the chess game. That the human and superhuman modes merely reduce down to just changing the depth of the minimax algorithm. So in order to implement superhuman mode we will keep the mini max algorithm’s depth to be 10 (not more than this due to the computational limits) and for the human mode we decide it to be kept at 4.