AI Checkers Project

Team Members

- Shagun Agrawal sa831 - 1910110363 (33.3%)
- Siddharth Borderwala sb943 - 1910110389 (33.3%)
- Shivang Madhwal sm479 - 1910110371 (33.3%)

Index

- Install Instructions
- Checkers Game
- MiniMax Algorithm
- Analysis

Install Instructions

Node version > 12 should be installed on the system.
Then in the root directory

```
npm install
npm run dev
```

The entire commented code is prenset inside the src/ folder.

Checkers Game

Introduction

Checkers, also called draughts, board game, one of the world's oldest games. Checkers is played by two persons who oppose each other across a board of 64 light and dark squares, the same as a chessboard. The 24 playing pieces are disk-shaped and of contrasting colours (whatever their colours, they are identified as black and white). At the start of the game, each contestant has 12 pieces arranged on the board.

Rules for checkers game

Play consists of advancing a piece diagonally forward to an adjoining vacant square. Black moves first. If an opponent's piece is in such an adjoining vacant square, with a vacant space beyond, it must be captured and removed by jumping over it to the empty square. If this square presents the same situation, successive jumps forward in a straight or zigzag direction must be completed in the same play. When there is more than one way to jump, the player has a choice. When a piece first enters the king row, the opponent's back row, it must be crowned by the opponent, who places another piece of the same colour on it. The piece, now called a king, has the added privilege of moving and jumping backward; if it moved to the last row with a capture, it must continue capturing backward if possible. A win is scored when an opponent's pieces are all captured or blocked so that they cannot move. When neither side can force a victory and the trend of play becomes repetitious, a draw game is declared.
For now we are using the **50-move rule** to declare if the game is draw or not. This rules states that the game is draw if **one of the following conditons** is true

- if neither player has advanced an uncrowned man towards the king-row during the previous 50 moves
- no pieces have been removed from the board during the previous 50 moves.

MiniMax Algorithm

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that the opponent also plays optimally. It is widely used in two player turn-based games like checkers.
In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.
Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Pseudocode

```
function minimax(node, depth, maximizingPlayer) is
    if (depth is 0) or (node is a terminal node) then
        return evaluation of node

    // for Maximizer Player
    if MaximizingPlayer then
        maxEva = negative infinity
        for each child of node do
            eva = minimax(child, depth-1, false)
            maxEva = max(maxEva,eva) //gives Maximum of the values
        return maxEva

    else // for Minimizer player
        minEva = positive infinity
        for each child of node do
            eva= minimax(child, depth-1, true)
            minEva= min(minEva, eva)         //gives minimum of the values
         return minEva
```

Checkers Game using AI

AI using random moves

We iterate over all the pieces of the computer and check for possible moves. If there are any moves that allow the computer to capture the enemy pieces then only one of those moves will be executed. If there is no move that can capture enemy piece then any one of the possible moves will be executed.

Pseudocode

```
list of all possible moves is made
if possible moves has possible capture moves then
    choose any one of the possible capture moves and execute it
else
    choose any one of the possible moves and execute it
```

AI using minimax alogrithm

For using minimax we need a heuristic for evaluating the state of the board so that we can make the best possible move. We have used the sum of the difference between the kings of ai and the player and difference between the normal pieces of ai and the player.

```
evaluate(board) = (aiKings - playerKings)*2 + (aiPieces - playerPieces)
```

Pseudocode

```
board // game board current state
depth // depth in the search tree


minimax(board, depth)
    bestMoveBoard // the final board
    //base condition
    if depth == 0 or isGameWon or isGameDraw
        return evaluate(board) //gives the score to the board

    for every board possible after different moves
        utility = minmax(candidateBoard, depth)
        if calculating for minimisation
            candidateUtility = min(candidateUtility, utility)
        else
            candidateUtility = max(candidateUtility, utility)

        if candidateUtility is equal to utility then
            bestMoveBoard = cadidateBoard
    return bestMoveBoard
```

Optimization

Generating a search tree after every move is a very computationaly heavy task. To optimise the code and reduce wait time to feasible limits we compromise on the full path that leads to finish and use the paths that may lead to victory. So one basic optimisation we made is to reduce the depth of the search tree to 3. This allows us to look 3 steps into the possible future without much computation.

Analysis

From our game we observed that the minimax AI made more valid moves than the random AI like not giving away free pieces and making moves that helped it capture more pieces in the future moves. The minimax AI could determine the possibilties and make the move accordingly to avoid losing the game. It was relatively tough to beat the minimax AI as compared to random AI. The response time for the random AI is faster as compared to minimax AI since it requires very less computational power as compared to minimax AI. As the depth increases in the minimax AI the reponse time will decrease but the difficulty of the game will also increase.