CSD311: Artificial Intelligence
Assignment - 2

Done by:
A Aadhavan
Ridham Bhagat
Samarth Gupta

Checkers:

Checkers is a game played by two players. One plays with light colored pieces and the other plays with dark colored pieces. Each player takes turns moving one of their pieces diagonally. A player may capture a piece by jumping over it and onto the next square. At the end of the game, the player with more pieces on the board wins.

Our approach:

Our take on the game of checkers is one where the player always makes the first move. In response to the player's move, the computer makes a move, and so on and so forth until the game concludes.

We have provided two algorithms that help the computer determine what move to make in response to a player's move. They are:

1. Min-Max algorithm
2. Random algorithm

The strategies and how they are implemented are described below.

Min-Max Algorithm:

There are two players involved in the game, denoted by MIN and MAX. The player MAX tries to maximise their score/utility, while MIN tries to minimise MAX's score/utility. MIN and MAX try to act opposite of each other.

The decision tree we use assumes both parties are competent and making the best decision at any given turn.

Algorithm:

1. Generate the entire game tree starting with the current position of the game all the way upto the terminal states.
2. Determine the scores/utility of all the terminal states.
3. Determine the utility of each of the internal nodes, by looking at the scores of nodes connected to it from the layer right below.

- If it is a MAX turn, pick the highest score from the nodes below.
- If it is a MIN turn, pick the lowest score from the nodes below.

4. Follow this algorithm until the root of the tree. This is a MAX turn and MAX should choose the highest value among the nodes connected to it from the layer below.

Random algorithm:

The goal of this algorithm is to make an AI that plays random moves. Moves that have no rime or reason behind them.

Algorithm:

1. Generate the entire game tree, similar to how the game tree was generated in the Min-Max algorithm.
2. Determine the scores of all the terminal states.
   This is where the two algorithms diverge.
3. Determine the utility of each of the internal nodes by randomly selecting a score from the nodes connected to it from the layer below.
4. Follow this algorithm until the root of the tree. In this way a random move was selected.

Win Condition:

The winner of the game is one who can make the last move; that is, no move is available to the opponent on his turn to play, either because all his pieces have been captured or his remaining pieces are all blocked.

Comparing the Two Strategies:

The Min-Max algorithm definitely gives rise to a much more formidable opponent. It always chooses the most optimal move in any given situation and is much harder to beat when compared to the AI running a random algorithm.

The random algorithm is on average a much easier opponent to play against. It plays with no intelligent strategy, and its moves have no overarching goal in mind. This makes it a much worse opponent than the AI operating on the Min-Max algorithm.

In terms of efficiency, in the ideal case, the random algorithm would be much more efficient as it would require no decision trees. Our implementation on the other hand runs both algorithms in the same amount of time (O(b^d) , b is branching factor and d is max depth).

The inefficiency of the random function arose due to us being crunched for time. This is something that can easily be remedied in the future.