CS312 - Artificial Intelligence Lab
Assignment 5

Group 10

Rudraksh (190030038), Sumedhsingh (190030042)

Likhilesh (190030024)

Introduction

In this assignment, we have to code a bot to win the game of Othello. Given a board configuration and a turn, our bot will return a valid move. The game ends when neither of the players can make a valid move. The player with the maximum number of coins is the winner.

1) Brief Introduction of Algorithms

- **Minimax Algorithm:** Minimax is a decision rule used in artificial intelligence, decision theory, game theory, and statistics for minimizing the possible loss for a worst-case (maximum loss) scenario. This algorithm makes a tree of positions as it explores all possible moves by the opponent. When it reaches the required depth, it evaluates the position using a heuristic function. It evaluates the best move such that it minimizes the best move of the opponent.
- **Alpha Beta Pruning:** Alpha-Beta Pruning minimizes the number of states explored in the search tree made by the

Minimax algorithm and prunes the search tree. It stops evaluating a move when it has been found that the current move is worse than a previously examined move. Such moves need not be explored further.

2) Heuristic Functions considered

- **Positions Heuristic**

  We were able to observe that corners gave an advantage to the player and apart from that more coins implied a better situation. So, we have built this heuristic function that takes both factors into account.

```
int H0(OthelloBoard board, int which)
{
    int corners = 0;
    int value_mul = 5;
    int value_const = 7;
    corners = corners + (board.get(0, 0) == TURN) * value_mul;
    corners = corners + (board.get(value_const, value_const) == TURN) * value_mul;
    corners = corners + (board.get(value_const, 0) == TURN) * value_mul;
    corners = corners + (board.get(0, value_const) == TURN) * value_mul;

    return corners + board.getBlackCount() - board.getRedCount();
}
```

- **Possible Moves Heuristic**

  This heuristic function gives an advantage to the player who has more moves left to play.

```
int H1(OthelloBoard board, int which)
{
    return (int)board.getValidMoves(TURN).size() - (int)board.getValidMoves(flip(BLACK)).size();
}
```

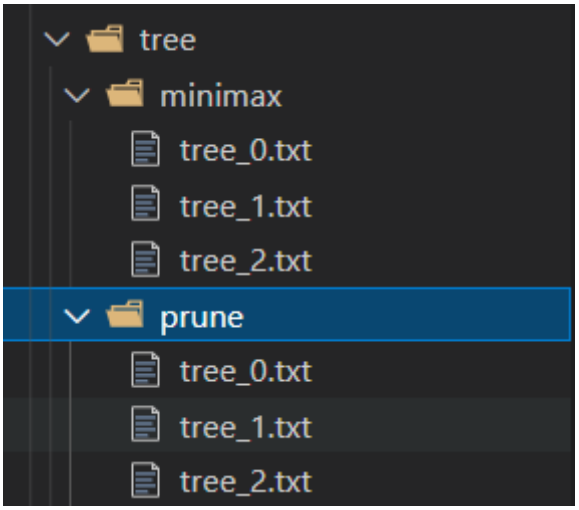
- **Coins Heuristic**

  This heuristic returns the difference between the coins.

```
int H2(OthelloBoard board, int which)
{
    return board.getBlackCount() - board.getRedCount();
}
```

3) Trees to show my particular moves are chosen for any 6 moves given the board configuration.

We have a separate folder “tree” which contains 3 trees (using 3 different heuristics) for each algorithm.



4) Comparison of both Algorithm

- **Based on Space & Time Complexity**

  Alpha Beta Pruning has less space and time complexity compared to the minimax algorithm, as moves that are guaranteed to be worse than previously examined moves are not further explored. As it eliminates the worst states that need to be explored leading it to save space thus decreasing its space complexity. Since fewer states are explored compared to minimax helping it to do its task in less time complexity too. time complexity is roughly O(b) and d space complexity is O(d * b).

- **Based on Winning Criteria**
    - As mentioned, the algorithm has a 2s deadline to play the next move which gives the alpha-beta pruning algorithm an edge over the minimax algorithm as it is able to explore greater depths. If there were no bound-on time then both bots are expected to play equally well
    - The comparison between the bots is made using the same heuristic.
    - we observe that the bot with alpha-beta pruning wins the game as it is able to search till a greater depth.