CS 214:Lab-5 Report

Hrishikesh Karande(210010020) , Shivesh Pandey(210020044)

17 March 2023

1 Introduction

In this assignment we had to code a bot to play the game of Othello in an optimal way, in order to win the game. With a given a board configuration and a turn, 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.

2 Overview of Algorithms

2.1 Minimax Algorithm

Minimax is a decision rule used in decision theory, game theory, statistics, and philosophy for minimizing the possible loss for a worst-case (maximum loss) scenario. This algorithm makes a tree of positions as it investigates all possible moves by the opponent. When it reaches the required depth, it leaves the position using a heuristic/evaluation function. It reckons the best move such that it minimizes the most profitable move of the opponent in the course of the game.

2.2 Alpha-Beta Pruning

Alpha-Beta Pruning seeks to minimize the number of positions explored in the search tree by the Minimax algorithm. It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. So, such moves need not be evaluated further and hence it prunes the search tree such that the outcome of the algorithm remains unchanged in the algorithm.

Condition for Pruning $\beta \leq \alpha$

2.3 Pseudocode

Minimax algorithm:

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Alpha-Beta Pruning:

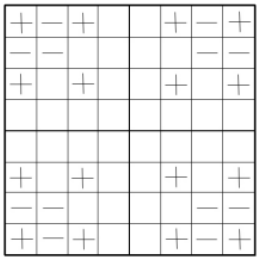
```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if value ≥ β then
                break (* β cutoff *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if value ≤ α then
                break (* α cutoff *)
        return value
```

3 Heuristics

We are using a single heuristics which cover the following features internally:

1. **Corner captivity:Capturing these corners would ensure stability in the region,as the corner discs can't be flipped.Stability is what determines the final outcome to quite a large extent.On contrary selecting the places directly adjacent to corners are usually bad to be the first to take, since your opponent gets a good chance of taking corners.**
2. **Edge Captivity:After the corners the edges are second in value to take, since the can only be flipped from two separate directions. Also, they provide a great basis for future moves of flipping discs towards the interior of the board.**
3. **Interior Board:the interior of the board can be seen as a miniature board of itself.**

All the above mentioned heuristics are not equally likely they follow the order $3 < 2 < 1$ pointing in the direction of better heuristics. Following this order we design a board with weights just like rewards . The favourable moves will give us positive rewards and the unfavourable ones give negative rewards (penalties).



```
{ 100, -10, 11,  6,  6, 11, -10, 100 },
{ -10, -20,  1,  2,  2,  1, -20, -10 },
{  10,   1,  5,  4,  4,  5,   1,  10 },
{   6,   2,  4,  2,  2,  4,   2,   6 },
{   6,   2,  4,  2,  2,  4,   2,   6 },
{  10,   1,  5,  4,  4,  5,   1,  10 },
{ -10, -20,  1,  2,  2,  1, -20, -10 },
{ 100, -10, 11,  6,  6, 11, -10, 100 }
```

4 Comparison between Alpha-Beta Pruning and Minimax algorithm:

Alpha-Beta Pruning is the Optimal version of Minimax algorithm, that prunes some nodes without having any effect on the value of root node. Therefore under the given time constraints it can search for more depth than Minimax and hence can be more accurate. However since it doesn't have any effect on actual value,when no time constraints are present both give same result.
So, further trials show that the two bots play nearly equally well and

that there is a general trend of the winning bot being the one that starts the game first. This is the affect of heuristics over the winning chances of the bot.

5 Time and Space Complexity:

Since both the algorithm mimic the way DFS works on the tree their time complexity is similar to that of DFS. That is Time Complexity $= O(b^d)$ **and Space Complexity** $= O(bd)$
Alpha-Beta pruning has same worst case time complexity as Minimax algorithm but since it prunes some of the nodes it saves time and space. In short the later is better in case when we can pruning of nodes is possible.