CS 314, Lab 5 - Report

Shriram Ghadge (180010015), Rishit Saiya (180010027)

March 15, 2021

1 Abstract

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 Algorithms & Heuristics

2.1 Algorithms

Below are the algorithms that were used during in the code.

2.1.1 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 explores all possible moves by the opponent. When it reaches the required depth, it assesses the position using a heuristic/evaluation function. It evaluates the best move such that it minimizes the best move of the opponent in the course of game.

2.1.2 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.

2.2 Heuristics

Below are the Heuristics that were used during in the code.

2.2.1 Coin Parity

This heuristic returns the lead of the player with respect to their opponent.

Algorithm 1 coinParity(position)

```
procedure coinParity(position)
    def coinParity(position):
    if position.turn == BLACK then
        return board.getBlackCount() - board.getRedCount()
    else
        return board.getRedCount() - board.getBlackCount()
```

2.2.2 Mobility

If we have more choices, it is likely to be the case that our best choice is better. This heuristic tries to improve our freedom to make a variety of moves with respect to our opponent.

Algorithm 2 mobility(position)

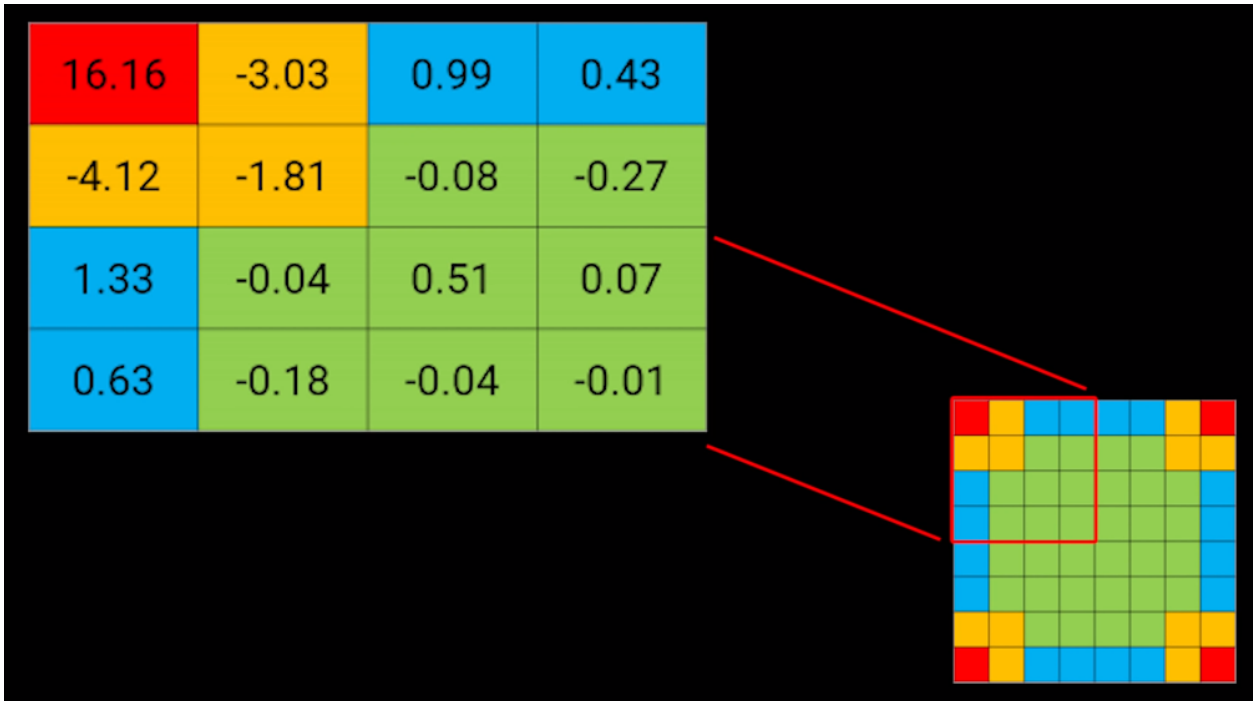
```
procedure mobility(position)
    def mobility(position):
    myMoves = position.getValidMoves(position.turn).size()
    oppMoves = position.getValidMoves(position.turn.opponent).size()
    return myMoves - oppMoves
```

2.3 Corners Captured

We see that the number of corners occupied improves the chances of winning substantially. This heuristic returns the difference in the corners occupied alongwith as well as edge elements. The reference weights for the heuristics were taken from the Figure 1. The main idea of algorithms remains same.

Algorithm 3 cornersCaptured(position)

```
procedure cornersCaptured(position)
    def cornersCaptured(position):
    myCorners, oppCorners = 0, 0
    for corner in position.corners() do
        if corner == position.turn then myCorners++
        if corner == position.turn then oppCorners++
    return (myCorners - oppCorners)
```



16.16	-3.03	0.99	0.43
-4.12	-1.81	-0.08	-0.27
1.33	-0.04	0.51	0.07
0.63	-0.18	-0.04	-0.01

Figure 1: Corners & Neighbors Captured Heuristics

PS: With the reference of Figure 1, we are not only concentrating upon the effect due to the corner cells, but also the effect due to neighbors of the corner cells. The probability of winning increases when we don't select the neighbors of the corner cells. Keeping that in mind, the weights in the heuristics were arranged and accordingly applied in algorithms. Reference

3 Trees moves for a given the board configuration

3.1 Minimax Algorithm

- Tree 1: [Coin Parity heuristic] → trees/minimax/tree_1.txt
- Tree 2: [Corners Captured heuristic] → trees/minimax/tree_2.txt
- Tree 3: [Mobility heuristic] → trees/minimax/tree_3.txt

3.2 Alpha Beta Pruning

- Tree 1: [Coin Parity heuristic] → trees/alphaBeta/tree_1.txt
- Tree 2: [Corners Captured heuristic] → trees/alphaBeta/tree_2.txt

- Tree 3: [Mobility heuristic] $\rightarrow$ trees/alphaBeta/tree_3.txt

4 Comparison across Minimax and Alpha-Beta Pruning Algorithms

In ideal case, both algorithms should perform ideally well given that Alpha-Beta Pruning is an optimized version of the Minimax Algorithm.

4.1 Winning Criteria

The time constraint (2 sec) to play the next move gives the Alpha Beta bot the advantage of exploring greater depths compared to the Minimax Bot. When time constraints are relaxed, both the bots are ideally expected to play equally well.

The comparison between the bots is made using the same heuristic as comparing their performances with different heuristics would be more constructive of the nature of the heuristic rather than that of the algorithm. 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.

4.2 Space and Time Complexity

The Alpha-Beta Pruning Algorithm has lesser time and space complexity as it moves that are guaranteed to be worse than previously examined moves, are not further explored. Thus by eliminating worse states that need not be explored, the space complexity is reduced. Since, lesser states are explored, in comparison to Minimax, the time complexity is also lesser in case of Alpha-Beta Pruning Algorithm.