CS 314, Lab 5 - Report

Shriram Ghadge (180010015), Rishit Saiya (180010027)

March 13, 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

In AI, 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

The main idea of 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)

```
1: procedure COINPARITY(position)
2:     def coinParity(position):
3:     if position.turn == BLACK then
4:         return board.getBlackCount() - board.getRedCount()
5:     else
6:         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)

```
1: procedure MOBILITY(position)
2:     def mobility(position):
3:     myMoves = position.getValidMoves(position.turn).size()
4:     oppMoves = position.getValidMoves(position.turn.opponent).size()
5:     return myMoves - oppMoves
```

2.3 Corners Captured

We see that the number of corners occupied improves the chances of winning drastically. This heuristic returns the difference in the corners occupied.

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

Algorithm 3 cornersCaptured(position)

```
1: procedure CORNERSCAPTURED(position)
2:     def cornersCaptured(position):
3:     myCorners, oppCorners = 0, 0
4:     for corner in position.corners() do
5:         if corner == position.turn then myCorners++
6:         if corner == position.turn then oppCorners++
7:     return (myCorners - oppCorners)
```

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] → 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 just an optimized version of the Minimax algorithm.

4.1 Winning Criteria

The two second constraint to play the next move gives the Alpha Beta bot the advantage of exploring greater depths compared to the Mini-max Bot. When unbounded by time constraints, both the bots are expected to play equally well. The two bots are compared using the same heuristic as comparing their performances with different heuristics would be more reflective of the nature of the heuristic rather than that of the algorithm. Simulations reveal 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.

4.2 Space and Time Complexity

The Alpha-Beta pruning algorithm has lesser space and time complexity as moves that are guaranteed to be worse than previously examined moves, are not further explored. By eliminating worse states that need not be explored, the space complexity is reduced. Since, lesser states are explored, in comparison to Mini-max, the time complexity is also lesser.