AI - LAB

Assignment-5 Report

# GAME PLAYING - OTHELLO

Arvind Kumar M - 200020008

Tarun Saini - 200010051

## Contents

## 1 Introduction

Othello or Reversi is a strategy board game for two players, played on an 8×8 uncheckered board.Players take turns placing disks on the board with their assigned color facing up.
During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color.
Player with more number of disks at the end is the winner.

Given a board configuration of othello, we have to return an optimal move which ensures the probability of winning using **minmax algorithm** and **alphabeta pruning** with **k-ply search** .

## 2 Minimax Algorithm

### 2.1 Definition

Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
Since there is a time constraint of 2 seconds for each move, we are using k-ply search with **3 depth**. Once it reaches the depth, it returns the heuristic for the last node.

### 2.2 Pseudo code:

```
def minimax(node, depth, player):

    if depth is 0 then

        return heuristic(node)

    if player is maximizing then

        best_val = -infinity

        for each child_node of  node do

            val= minimax(child, depth-1, minimizing)

            best_val= max(best_val,val)
```

```
else
    best_val = infinity
    for each child_node of node do
        val= minimax(child, depth-1, maximizing)
        best_val= min(best_val,val)
return best_val
```

## 3 Alpha Beta Pruning

### 3.1 Definition

Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm. Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.The main condition which required for alpha-beta pruning is $\alpha \geq \beta$.
We are applying k-ply search till **depth 4**.

### 3.2 Pseudo code:

```
def alphabeta(node, depth, player, alpha, beta):
    if depth is 0 then
        return heuristic(node)
    if player is maximizing then
        best_val = -infinity
        for each child_node of node do
            val= minimax(child, depth-1, minimizing)
            best_val= max(best_val,val)
            alpha= max(best_val,alpha)
            if alpha $\geq$ beta then
        break
  else
    best_val = infinity
    for each child_node of node do
      val= minimax(child, depth-1, maximizing)
      best_val= min(best_val,val)
      beta= min(best_val,beta)
      if alpha $\geq$ beta then
        break
  return best_val
```

## 4 Heuristic Function

### 4.1 Definition

We are calculating heuristic based on four conditions:

- **Lead :** Assigning positive value for difference between the total number of disks of the player and opponent.
- **Valid moves:** Assigning positive value for difference between number of valid moves between player and opponent.
- **Corner position :** Assigning corner value as 1000 for player as it increases the chances of winning.
- **Next to corner :** The position surrounded by corner cells are assigned as -500 as it increases the chances of winning for opponent.

### 4.2 Pseudo code:

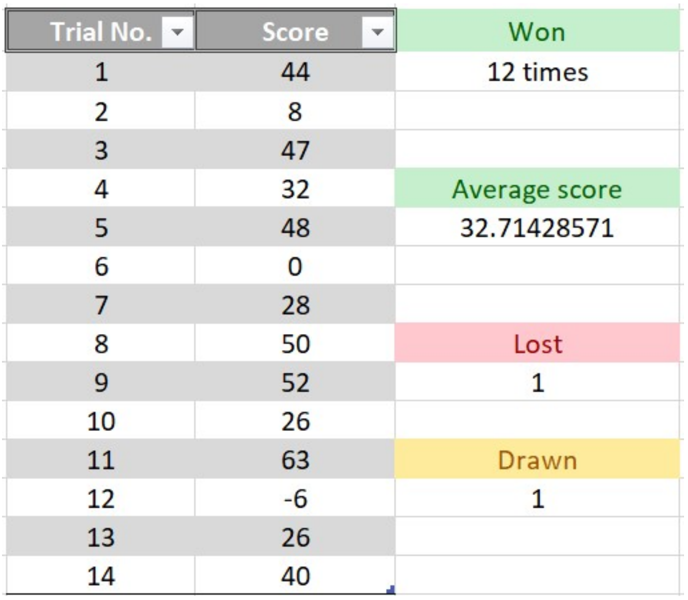
```
def heuristic():
  heur = 0
  heur += no_of_dics_diff
```

```
diff_in_moves = mypossibleMoves - opponentMoves
heur += 5*(diff_in_moves)
if position is corner do
    heur += 1000
if position is next to corner do
    heur -= 500

  return heur
```

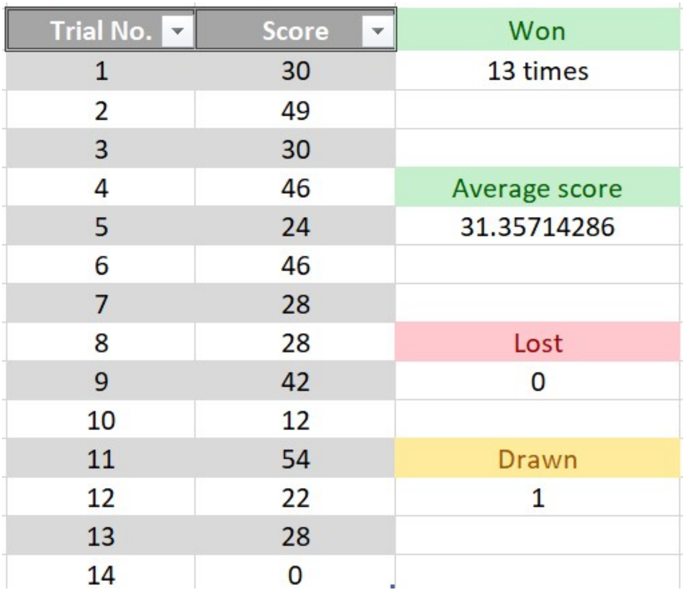
## 5 Results

### 5.1 Minimax vs RandomBot



| Trial No. | Score | Won |
|---|---|---|
| 1 | 44 | 12 times |
| 2 | 8 | |
| 3 | 47 | |
| 4 | 32 | Average score |
| 5 | 48 | 32.71428571 |
| 6 | 0 | |
| 7 | 28 | |
| 8 | 50 | Lost |
| 9 | 52 | 1 |
| 10 | 26 | |
| 11 | 63 | Drawn |
| 12 | -6 | 1 |
| 13 | 26 | |
| 14 | 40 | |

### 5.2 AlphaBeta vs RandomBot



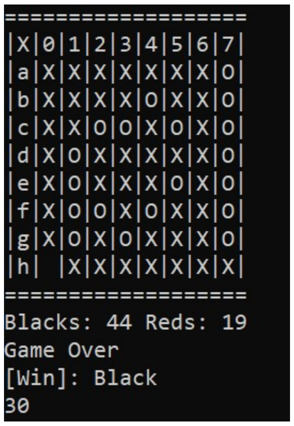
| Trial No. | Score | |
|---|---|---|
| 1 | 30 | Won |
| 2 | 49 | 13 times |
| 3 | 30 | |
| 4 | 46 | Average score |
| 5 | 24 | 31.35714286 |
| 6 | 46 | |
| 7 | 28 | |
| 8 | 28 | Lost |
| 9 | 42 | 0 |
| 10 | 12 | |
| 11 | 54 | Drawn |
| 12 | 22 | 1 |
| 13 | 28 | |
| 14 | 0 | |

# 6 Minimax vs AlphaBeta

## 6.1 Results

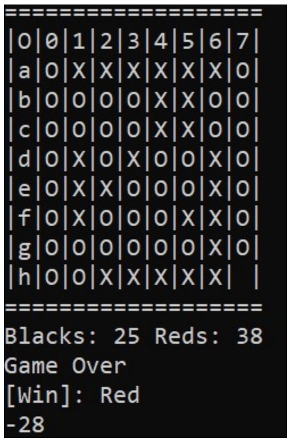
With Minimax bot, AlphaBeta bot is winning all the time (i.e. while playing first or second).

**AlphaBeta bot playing first (BLACK)**



```
===================
|X|0|1|2|3|4|5|6|7|
|a|X|X|X|X|X|X|X|O|
|b|X|X|X|X|O|X|X|O|
|c|X|X|O|O|X|O|X|O|
|d|X|O|X|X|X|X|X|O|
|e|X|O|X|X|X|O|X|O|
|f|X|O|O|X|O|X|X|O|
|g|X|O|X|O|X|X|X|O|
|h| |X|X|X|X|X|X|X|
===================
Blacks: 44 Reds: 19
Game Over
[Win]: Black
30
```

**AlphaBeta bot playing second (RED)**



```
===================
|O|0|1|2|3|4|5|6|7|
|a|O|X|X|X|X|X|X|O|
|b|O|O|O|O|X|X|O|O|
|c|O|O|O|O|X|X|O|O|
|d|O|X|O|X|O|O|X|O|
|e|O|X|X|O|O|O|X|O|
|f|O|X|O|O|O|X|X|O|
|g|O|O|O|O|O|O|X|O|
|h|O|O|X|X|X|X|X| |
===================
Blacks: 25 Reds: 38
Game Over
[Win]: Red
-28
```

## 6.2 Comparision

- When there is no time constraint, both algortihtm performs ideally same. But here we are given with a time constraint of 2 seconds for each move. So Minimax algorithm can only search upto depth 3. Alpha beta algorithm can prune some subtrees and can search upto depth 4 in some cases, thereby improving chances of winning.
- This is the reason being minimax is losing to Alpha beta bot even it starts first.
- In terms of **space and time complexity** both are exponential in terms of branching factor. So we are applying k-ply search for both with k values 3 and 4 respectively for minimax and alpha beta.
- **Winning criteria :** When played against random bot both performs very good but alpha beta won many times compared to minimax bot.
- So when there is time constraint, alphabeta pruning algorithm performs well with k-ply search.