REPORT

190020021 (Kushagra Khatwani) , 190020039 (Suryavijoy Kar)

---

1. Brief description of algorithms:

MiniMax Algorithm:

Minimax algorithm is a backtracking algorithm. It is used for making decisions and has extensive application in game theory. As the name suggests, it has two components, a maximizer and a minimizer. The maximizer tries to maximize the score and the minimizer tries to minimize it. The basic idea of the algorithm is to explore the different possible states by constructing a tree of fixed depth. Upon reaching the depth, the value of each state is calculated using a heuristic and is added to the score. Depending on whether it is a maximizer or minimizer, the state is chosen to maximize or minimize the score.

AlphaBeta Pruning:

AlphaBeta Pruning is an optimization over minimax algorithm. It tries to reduce the number of states explored. Thus allowing more states to be explored in the same time interval. The idea behind the algorithm is to check if certain moves result in states that definitely perform worse than previous moves, and eliminate those states. Thus, reducing the exploration space.

2. Heuristic functions considered:

The following paper is referred for deciding heuristics: An Analysis of Heuristics in Othello - Vaishnavi Sannidhanam and Muthukaruppan Annamalai

1. Mobility

```
int numOppMoves = board.getValidMoves(other(this->turn)).size();
int numSelfMoves = board.getValidMoves(this->turn).size();

if(numOppMoves + numSelfMoves != 0)
{
  return 100*(numSelfMoves - numOppMoves)/(numOppMoves + numSelfMoves);
}
```

```
else
  return 0;
```

2. CoinParity

```
if(this->turn == BLACK)
  return (board.getBlackCount() - board.getRedCount());
return (board.getRedCount() - board.getBlackCount());
```

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

1. MinMax:

```
Loading libOthello...
Loading bot...
Loading bot...
Exploring : (2, 3),       depth = 4,       No. children - 5
Child - 0 ==> (1, 2) :    Heuristic Value: 0
Child - 1 ==> (2, 2) :    Heuristic Value: 0
Child - 2 ==> (3, 2) :    Heuristic Value: 0
Child - 3 ==> (4, 2) :    Heuristic Value: 0
Child - 4 ==> (5, 2) :    Heuristic Value: 0
Exploring : (1, 2),       depth = 3,       No. children - 6
Child - 0 ==> (1, 3) :    Heuristic Value: 9
Child - 1 ==> (1, 5) :    Heuristic Value: 9
Child - 2 ==> (2, 5) :    Heuristic Value: 9
Child - 3 ==> (3, 5) :    Heuristic Value: 9
Child - 4 ==> (4, 5) :    Heuristic Value: 9
Child - 5 ==> (5, 5) :    Heuristic Value: 9
Exploring : (1, 3),       depth = 2,       No. children - 6
Child - 0 ==> (0, 2) :    Heuristic Value: 7
Child - 1 ==> (1, 4) :    Heuristic Value: 7
Child - 2 ==> (2, 2) :    Heuristic Value: 7
Child - 3 ==> (3, 2) :    Heuristic Value: 7
Child - 4 ==> (4, 2) :    Heuristic Value: 7
Child - 5 ==> (5, 2) :    Heuristic Value: 7
Exploring : (0, 2),       depth = 1,       No. children - 7
```

Child - 0 ==> (0, 1) : Heuristic Value: 16
Child - 1 ==> (0, 3) : Heuristic Value: 16
Child - 2 ==> (1, 5) : Heuristic Value: 16
Child - 3 ==> (2, 5) : Heuristic Value: 16
Child - 4 ==> (3, 5) : Heuristic Value: 16
Child - 5 ==> (4, 5) : Heuristic Value: 16
Child - 6 ==> (5, 5) : Heuristic Value: 16

2. AlphaBeta:

Loading libOthello...
Loading bot...
Loading bot...
Exploring : (0, 3), depth = 0, No. children - 6
Child - 0 ==> (0, 4) : Heuristic Value: 14
Child - 1 ==> (1, 4) : Heuristic Value: 14
Child - 2 ==> (2, 2) : Heuristic Value: 14
Child - 3 ==> (3, 2) : Heuristic Value: 14
Child - 4 ==> (4, 2) : Heuristic Value: 14
Child - 5 ==> (5, 2) : Heuristic Value: 14
Exploring : (1, 5), depth = 0, No. children - 7
Child - 0 ==> (1, 4) : Heuristic Value: 0
Child - 1 ==> (2, 2) : Heuristic Value: 0
Child - 2 ==> (3, 2) : Heuristic Value: 0
Child - 3 ==> (3, 5) : Heuristic Value: 0
Child - 4 ==> (4, 2) : Heuristic Value: 0
Child - 5 ==> (5, 2) : Heuristic Value: 0
Child - 6 ==> (5, 3) : Heuristic Value: 0
Exploring : (2, 5), depth = 0, No. children - 6
Child - 0 ==> (1, 4) : Heuristic Value: -9
Child - 1 ==> (2, 2) : Heuristic Value: -9
Child - 2 ==> (3, 5) : Heuristic Value: -9
Child - 3 ==> (4, 2) : Heuristic Value: -9
Child - 4 ==> (4, 5) : Heuristic Value: -9
Child - 5 ==> (5, 3) : Heuristic Value: -9
Exploring : (3, 5), depth = 0, No. children - 6
Child - 0 ==> (2, 2) : Heuristic Value: 20
Child - 1 ==> (2, 6) : Heuristic Value: 20
Child - 2 ==> (4, 2) : Heuristic Value: 20
Child - 3 ==> (4, 5) : Heuristic Value: 20
Child - 4 ==> (4, 6) : Heuristic Value: 20
Child - 5 ==> (5, 3) : Heuristic Value: 20

4. Compare the two algorithms and justify which is better in terms of

1. Space and Time complexity

As alpha-beta pruning eliminates certain states which will definitely give worse scores, it reduces the search space. As a result, **the time complexity and space complexity of the alpha-beta pruning is less than the minimax algorithm**. The efficiency of pruning the tree is largely affected by the move ordering in the game. When better moves are generated earlier, the cutoffs are greater in number. Better moves can be generated by considering better heuristic functions and sorting the children according to the heuristic functions.

2. Winning criteria

We observe that both the algorithms perform equally well when competing against each other. When the heuristic is the same for both, the algorithm which plays first wins most of the time. The difference is created only based on the heuristic.