CS312:Artificial Intelligence Laboratory

Lab 5 Report Group 24

Utkarsh Prakash - 180030042 Manjeet Kapil - 180010021

1. Introduction

In this assignment, we were expected to code a bot to play the game of Othello in an optimal way, in order to win the game.

2. Minimax Algorithm

The minimax algorithm searches the game tree till depth k in a depth-first manner from left to right. It applies the minimax rule to determine the value of the root node. Once the algorithm reaches the terminal node, the algorithm applies the evaluation function (heuristic function) instead of making a recursive call. Since, at first we don't know the optimal value of k so we explore the game tree in an iterative deepening way (DFID).

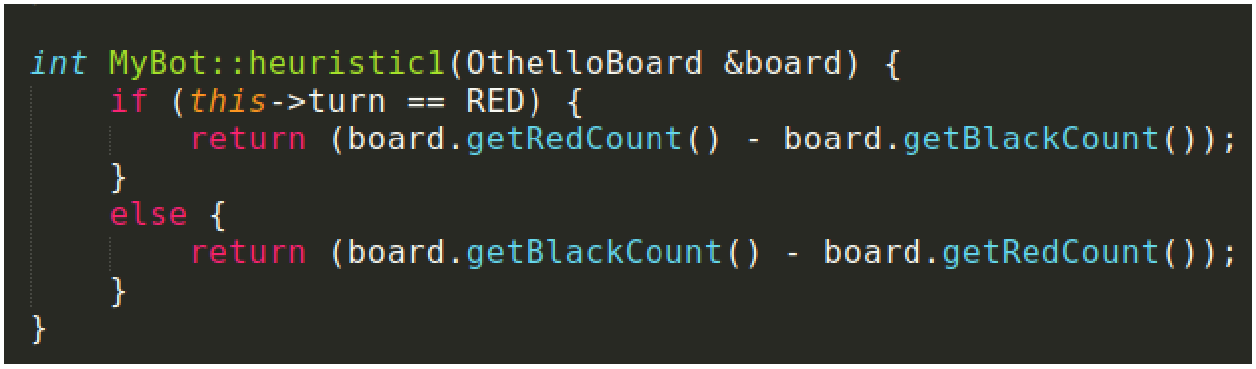
3. Alpha Beta Pruning

Let's say during exploration of successors of a MAX one node evaluates to +infinity. Since one cannot hope to improve upon, it does not make sense to search any further. For pruning to happen, it is not necessary that a winning move has been found. It can also happen that during exploration it becomes clear that a particular child of a node cannot offer to improve upon the value delivered by a sibling. In that case, that node need not be explored further. This way we can minimize the number of positions explored in the game tree by the Minimax algorithm.

4. Heuristics

- **Heuristic 1**

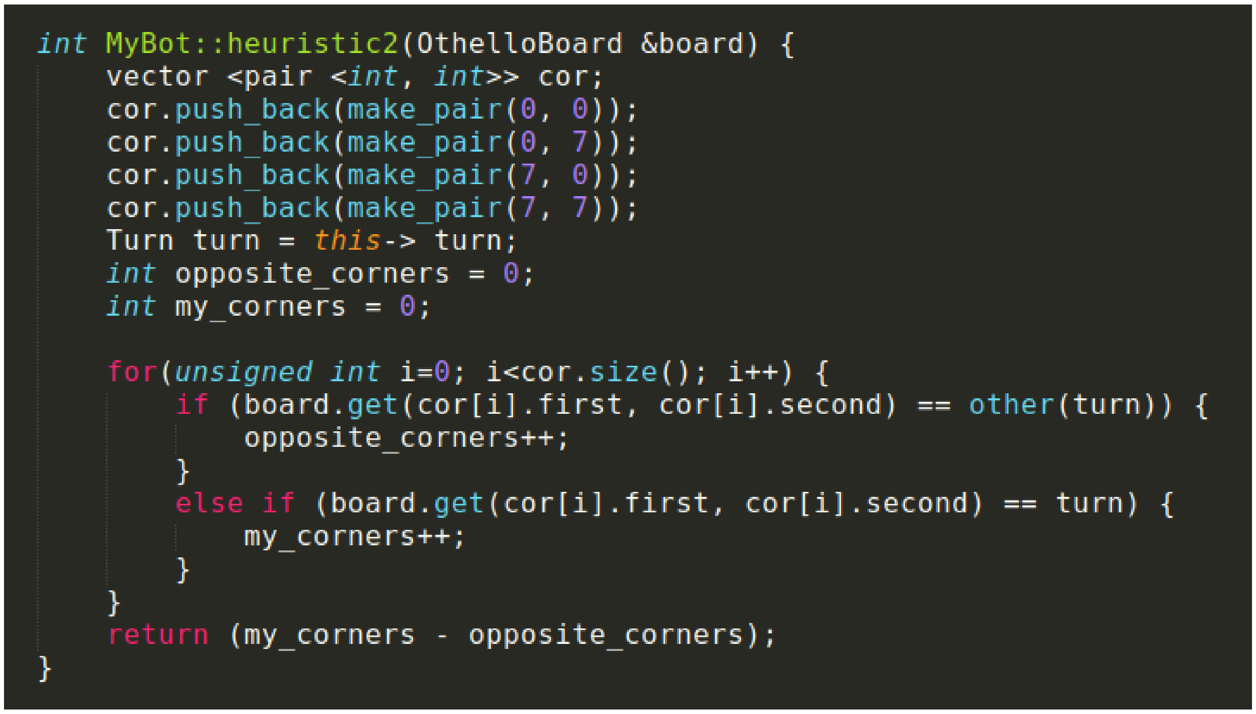
In this heuristic we are calculating the difference between the number of RED coins on the board and the BLACK coins on the board. Hence, this heuristic determines whether the player is leading or falling behind in comparison to his/her opponent.



```
int MyBot::heuristic1(OthelloBoard &board) {
    if (this->turn == RED) {
        return (board.getRedCount() - board.getBlackCount());
    }
    else {
        return (board.getBlackCount() - board.getRedCount());
    }
}
```

- **Heuristic 2**

In the game of Othello, the corner positions are one of strongest positions. The player who occupies more number of corner positions has a higher chance of winning. Hence, this heuristic calculates the difference between the number of corners occupied by me and my opponent.



```
int MyBot::heuristic2(OthelloBoard &board) {
    vector <pair <int, int>> cor;
    cor.push_back(make_pair(0, 0));
    cor.push_back(make_pair(0, 7));
    cor.push_back(make_pair(7, 0));
    cor.push_back(make_pair(7, 7));
    Turn turn = this-> turn;
    int opposite_corners = 0;
    int my_corners = 0;

    for(unsigned int i=0; i<cor.size(); i++) {
        if (board.get(cor[i].first, cor[i].second) == other(turn)) {
            opposite_corners++;
        }
        else if (board.get(cor[i].first, cor[i].second) == turn) {
            my_corners++;
        }
    }
    return (my_corners - opposite_corners);
}
```

- **Heuristic 3**

The player who has a higher number of available positions/moves than his/her opponent has higher chances of winning as it is more likely that the move is better. Hence, this heuristic calculates the difference between the number of available moves for me and my opponent.

```
int MyBot::heuristic3(OthelloBoard &board) {
    return (board.getValidMoves(this->turn).size() - board.getValidMoves(other(this->turn)).size());
}
```

5. Trees to show my particular moves are chosen for at least 6 moves given the board configuration.

We have printed the trees in respective files as given below.

Minimax Algorithm:

- Tree1(Heuristic1) : trees/minimax_tree_1.txt
- Tree2(Heuristic2) : trees/minimax_tree_2.txt
- Tree3(Heuristic3) : trees/minimax_tree_3.txt

Alpha Beta Pruning Algorithm:

- Tree1(Heuristic1) : trees/AB_tree_1.txt
- Tree2(Heuristic2) : trees/AB_tree_2.txt
- Tree3(Heuristic3) : trees/AB_tree_3.txt

6. Comparison of Minimax and Alpha-Beta Pruning

NOTE: Comparison of both the bots are done using the same heuristic as using different heuristic for comparison can bias our results.

Space and Time Complexity:

The space and time complexity of Alpha-Beta pruning are observed to be lower than that of the Minimax algorithm. This is because the Alpha-Beta pruning algorithm doesn't explore the states which are worse than the previously examined moves. Due to exploration of a lesser number of states the space complexity is reduced as fewer number of states are stored into memory. Moreover, due to lesser exploration we also reduce the time complexity of the Alpha-Beta Pruning algorithm.

Winning Criteria

Each bot is given 2 seconds to decide the next move. Since the time complexity of Alpha-beta pruning algorithm is less, hence it can explore states at greater depths than the Minimax bot. Therefore, under time constraints the performance of Alpha-Beta bot is better than the Minimax bot. When we play the game using the flag -t then there is no time constraint on the bots to make a decision. Under these circumstances of unbounded times, we observe after several simulations both the bots perform equally well and the bot which starts first has a slight edge over the other bot.

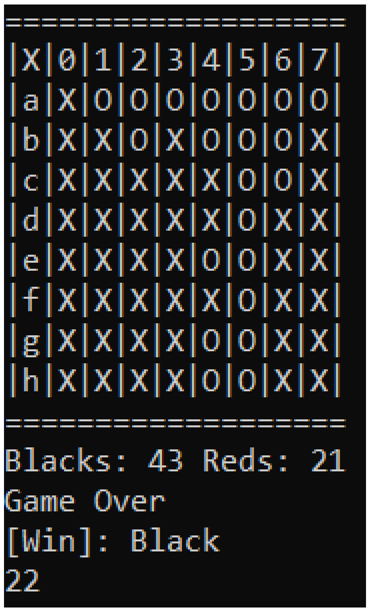
7. Observations

NOTE: The given observations are for running the Alpha-Beta bot against Minimax bot.

Heuristic	Time Taken	Winner Leads By
Heuristic 1	1 min 10 sec	13
Heuristic 2	47 sec	44
Heuristic 3	1 min 12 sec	22

We can observe empirically that heuristic 2 is the best heuristic.

To visualize the othello board use the flag -v to run the program. The final output is shown below:



```
===================
|X|0|1|2|3|4|5|6|7|
|a|X|O|O|O|O|O|O|O|
|b|X|X|O|X|O|O|O|X|
|c|X|X|X|X|X|O|O|X|
|d|X|X|X|X|X|O|X|X|
|e|X|X|X|X|O|O|X|X|
|f|X|X|X|X|X|O|X|X|
|g|X|X|X|X|O|O|X|X|
|h|X|X|X|X|O|O|X|X|
===================
Blacks: 43 Reds: 21
Game Over
[Win]: Black
22
```