UMEÅ UNIVERSITET **May 24, 2020**
Department of Computing Science
Assignment 1

Assignment1

Artificial Intelligence - Methods and Applications

Othello

Name	Hui Zhang
Username	mrc19hzg
Email	mrc19hzg@cs.umu.se

Graders
Ola Ringdahl & Juan Carlos Nieves & Lennart Steinvall

1 Introduction

In this assignment, we are required to write an Othello engine, a program that computes moves for the game Othello. Basically, what I have done in this assignment is to write functions and algorithms to take an Othello position represented by a string, analyze this position by using the heuristic evaluation method and return a recommended move for the player who has this move within a given time limit by applying Alpha-Beta pruning algorithm.

2 How to run

Since we have the provided shell script called othello.sh. We can run use that to run the program. The shell script takes three arguments, a description of the position, a time limit, and a flag do_compile to indicate whether to run or compile.

- 1. The first argument is an ascii string of length 65. The first character is W or B, and indicates which player is to move. The remaining characters should be E(for empty), O(for white markers), X(for black markers). Theses characters describe the board.
- 2. The second parameter gives a time limit in seconds. The time limit is used to guide the depth of search, when time is up, the search will end at the deepest depth it has reached and then return a best move it has found.
- 3. The third parameter do_compile decides if the code should compile or not. If 0, the program will run, if 1 it will be compiled but not run.

We also have the shell script for testing the program. Run the ***'othellostart'*** script with three parameters, indicating which programs should play against each other and the time limit for them:

./othellostart white_player black_player time_limit

Where *white_player* and *black_player* are shell scripts for white and black player respectively. *time_limit* is the time limit for the search algorithm in seconds. For example, if I want my program to be the white player and play against the naive program and set 4 seconds for the time limit, I should write the following command in a Linux terminal:

./othellostart /path_to_myProgram/othello.sh ./othello_naive 4

3 Implementation

In this part, I will explain how did I implement the othello engine in details. Which includes the description of the algorithm for makeMove() method. Heuristic evaluation algorithm, and alpha-beta pruning algorithm, and how to end the search when time is up.

3.1 makeMove()

makeMove() is a method of the OthelloPosition class. Which takes an Othello action and returns the position resulting from making this action in the current position and change the player to move next.

It makes writing the makeMove method very simple and straight forward with the help of helper code. Basically, we need to check whether this action is a pass move first, if yes, change the player to the next one and return the current position. If not, check whether this action is a move in 8 different directions(check northwest, north, northeast, west, east, southwest, south, southeast by using the Position.checkNorthEast() and other direction checking method in other directions.) one by one. If this action is a move in this direction, flip the opponent's color to the current player's color in every square along this direction until it reaches the current player's own square.

3.2 Heuristic

I got the idea to write my static heuristic function for evaluating position from Kartik Kukreja's blog.[1] I took the board weights matrix from his heuristic function. Each cell has a static weight, which is used to calculate the total board score for a position. Figure.1 shows the weight for every cell.

```
public static final int[][] BOARD_WEIGHT = {
        {20, -3, 11,  8,  8, 11, -3, 20},
        {-3, -7, -4,  1,  1, -4, -7, -3},
        {11, -4,  2,  2,  2,  2, -4, 11},
        { 8,  1,  2,  1,  1,  2,  1,  8},
        { 8,  1,  2,  1,  1,  2,  1,  8},
        {11, -4,  2,  2,  2,  2, -4, 11},
        {-3, -7, -4,  1,  1, -4, -7, -3},
        {20, -3, 11,  8,  8, 11, -3, 20}};
```

Figure 1: Board weights

I implemented the OthelloEvaluator interface in the StaticHeurisitc class. I have two heuristic functions in the StaticHeuristic class.

One is to use the board weights matrix, if the cell is marked by white player, add this cell's weight to the evaluation value, if the cell is marked by black player, subtract it from the evaluation value, if the cell is empty, then jump to check the next cell. After loop through all 64 cells, we will get a final evaluation value for this position.

Another one is to evaluate the position by counting how many moves that the two players have respectively. Let the number of my player's moves subtract the number of my opponent player's moves, then return the value.

Sum up the two values that we got from these two different heuristic functions. Use the summary as our final evaluation value of the current Othello position.

3.3 AlphaBetaPruning Algorithm

I used the alpha-beta pruning algorithm as my searching algorithm for best move. The pseudo code for alpha-beta algorithm is shown in Figure.6

```
function AlphaBeta(search tree node s)
    return MaxValue(s, −∞, +∞)
end function

function MaxValue(search tree node s, α, β))
    if s is a leaf then
        return utility(s)
    end if
    value ← −∞
    for each child t of s do
        value ← max(value, MinValue(t, α, β))
        α ← max(α, value)
        if α ≥ β then
            break (* β cutoff *)
        end if
    end for
    return value
end function

function MinValue(search tree node s, α, β))
    if s is a leaf then
        return utility(s)
    end if
    value ← +∞
    for each child t of s do
        value ← min(value, MaxValue(t, α, β))
        β ← min(β, value)
        if α ≥ β then
            break (* α cutoff *)
        end if
    end for
    return value
end function
```

Figure 2: Alpha-Beta Algorithm [2]

3.4 How to stop when time's up

In the main method, check the current system time at where the program starts to run, and add the time limit(times 1000 to transform the unit from second to millisecond) from the user input to it. And use this value as the time limit for our program.

In the AlphaBetaPruning class, we have a method called evaluate, which takes an Othello position and returns an Othello action. When we call the evaluate method, this method will call maxValue or minValue method within the class according to which player is playing, maxValue for maxPlayer, and minValue for the other. minValue and maxValue are recursive functions. I wrote a checkTime method, and the checkTime method will be called by minValue and maxValue methods every time when them start to run, and checkTime method will compare the current system time with the time limit we have set in the main method for our program. When the current system time exceeds the time limit, checkTime will throw a TimeoutException.

Ok, let's go back to the main method of our program. Here I used a for loop to let the alpha-beta algorithm to try to find the deepest depth it can reach within the given time limit. When the program catches the TimeoutException, break the for loop, print the best move it has just found.

4 Results

I have test the program to play as white and black against the naive player. And tried to set different time limit for it. Here are some results from the test.

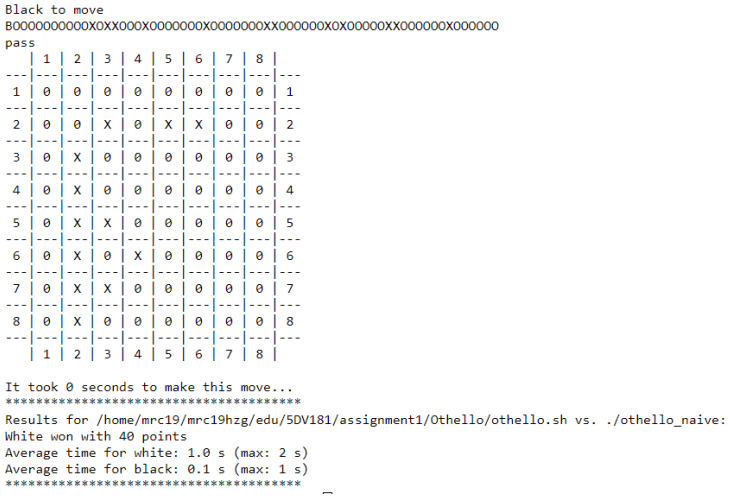

Figure 3: My program plays as white player, with 2s time limit

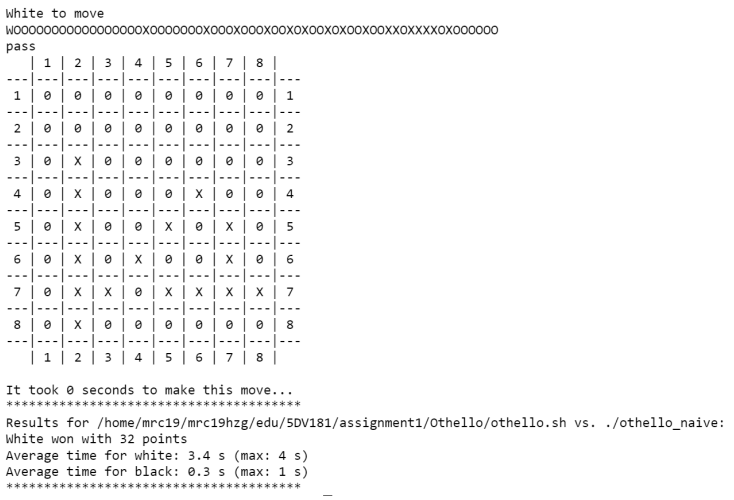

Figure 4: My program plays as white player, with 4s time limit

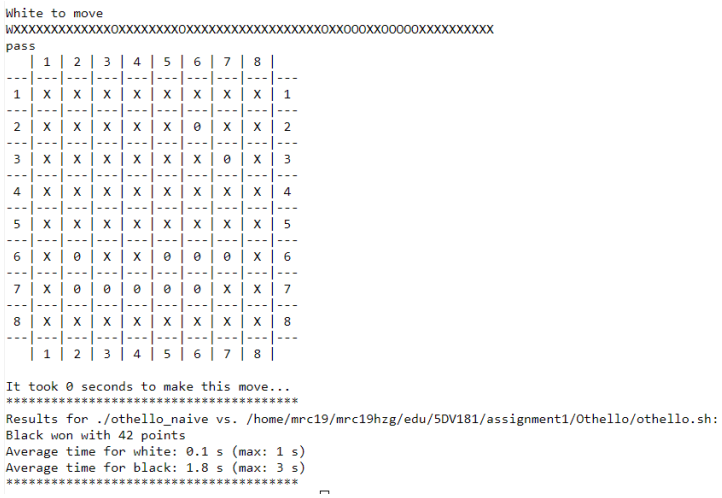

Figure 5: My program plays as black player, with 3s time limit

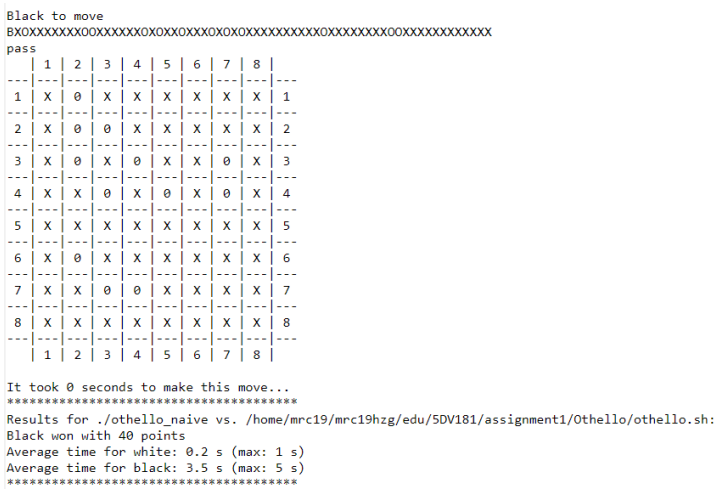

Figure 6: My program plays as black player, with 5s time limit

5 Issues

Honestly, I do have a lot of problems when I did this assignment. Firstly, I don't have a computer science background. I don't have much programming experience. I was even struggling with the coding assignment for the AI 1 course, which was thought to be much easier than this one. I've only learned C, and has a very basic understanding of that. To finish this Othello assignment, I start to learn Java from zero, I just found it is very hard to handle all these courses and learning some stuff that I was supposed to know before those courses by myself at the same time. Anyway, I finally finished this one, and I will do the assignment 2 after the final exams of the courses which I take this semester.

And another problem I would like to mention is one that I encountered when I test the program, Figure.7 shows the exception I've got.

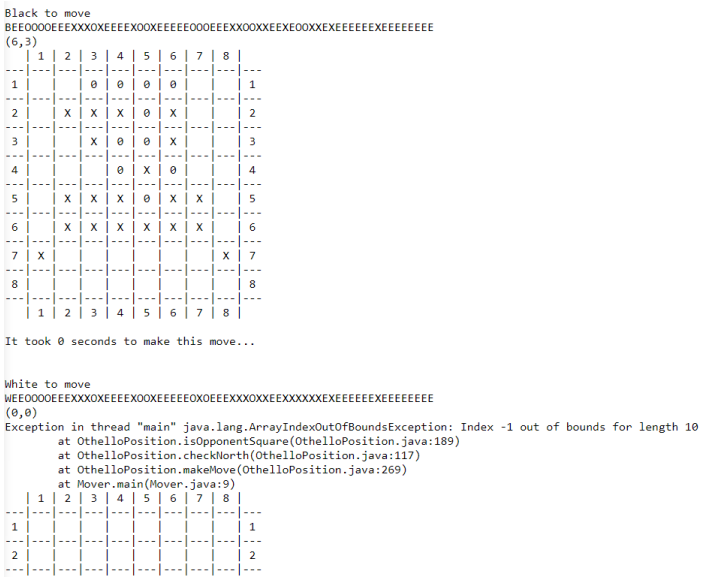

Figure 7: ArrayIndexOutofBoundException

I got this exception because my program returned an illegal action (0,0). I do handle the pass move in the minValue and maxValue method, but I still got this problem. From my understanding, if I've checked the pass move in the evaluate method of AlphaBetaPruning class, I should get a pass move instead of (0,0) if this is a pass move. Well, just to solve the problem is easy, I just need to set the action (0,0) as the pass move in the main method, then the program will run correctly. But I just still don't understand why will I get (0,0) action even though I've already checked it in the evaluate method.

References

[1] https://kartikkukreja.wordpress.com/2013/03/30/heuristic-function-for-reversiothello/ retrived on 2020.05.10

[2] Ola Ringdahl's lecture slide of AI: Methods and applications, Lecture 2: Adversarial Search