CSCI561 – Introduction to Artificial Intelligence
Instructor: Dr. K. Narayanaswamy
Assignment 3 – Adversarial Search
Reversi
Due: 10/25/2013 11:59:59pm

1. Introduction

In this project you will write a program to play the Reversi game using Minimax and Alpha-Beta pruning algorithms with two different evaluation functions:

1) number of pieces

2) positional weights.

The rules of the Reversi game can be found e.g. in http://en.wikipedia.org/wiki/Reversi [1]. The interactive examples can be found e.g. http://www.samsoft.org.uk/reversi/[2]. In the Othello version of this game, the game begins with four disks placed in a square in the middle of the grid, two facing light-up, two pieces with the dark side up, with same-colored disks on a diagonal with each other. However, in this assignment, the starting position will be specified in the input file.

2. Tasks

In this assignment, you will write a program to implement the following algorithms for both min and max players.

2.1 Minimax using number of pieces as an evaluation function ;

2.2 Alpha-Beta pruning using number of pieces as an evaluation function ;

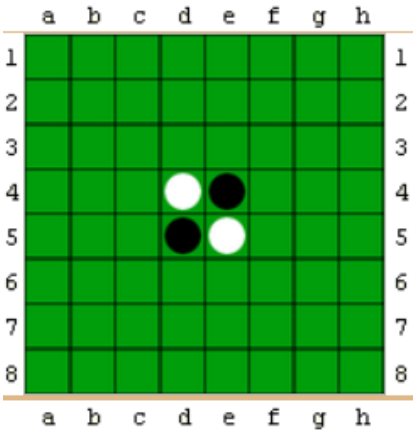
2.3 Alpha-Beta pruning using positional weights as an evaluation function ;

3. Specifications and Illustrative Examples

3.1 Players

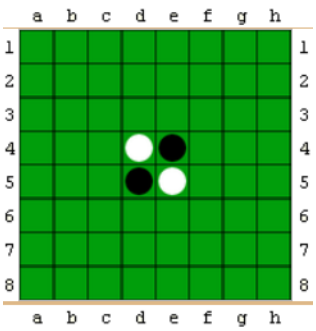
Black = max player, White = min player

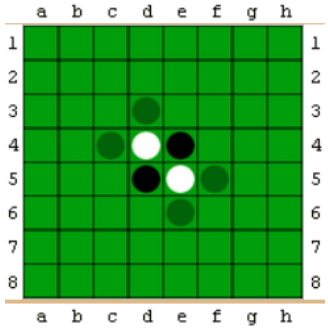
Assume that Max player always makes the first move from the configuration provided.



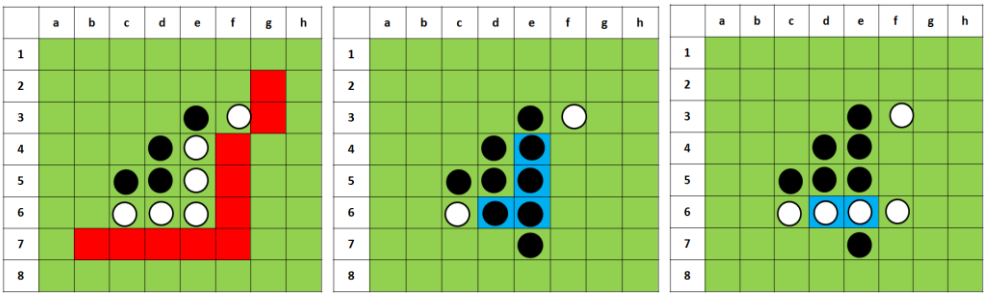
3.2 Legal moves and Flips/Captures

Let us assume that the current position is as shown in the right image below and that the current turn is Black's. Black must place a piece with the black side up on the board, in such a position that there exists at least one straight (horizontal, vertical, or diagonal) occupied line between the new piece and another black piece, with one or more contiguous light pieces between them. The right image shows all the legal moves available to the Black player (see the translucent circles below that highlight the legal Black moves). If one player cannot make a valid move, play passes back to the other player.

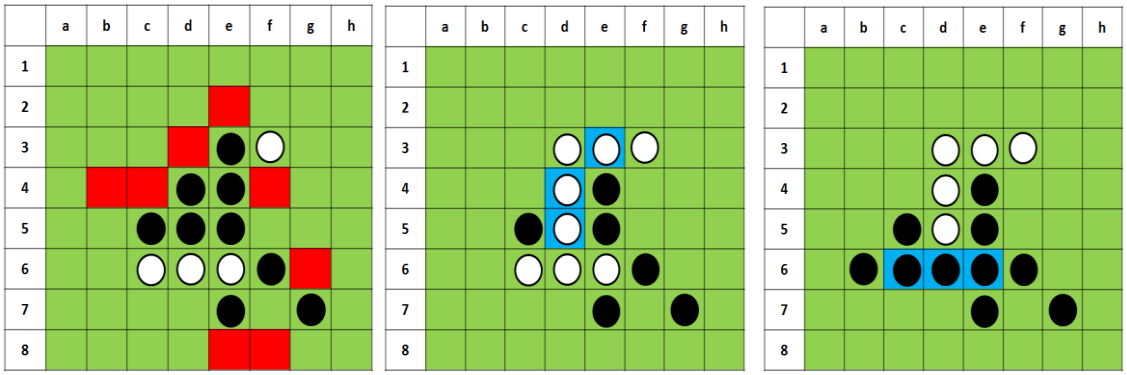




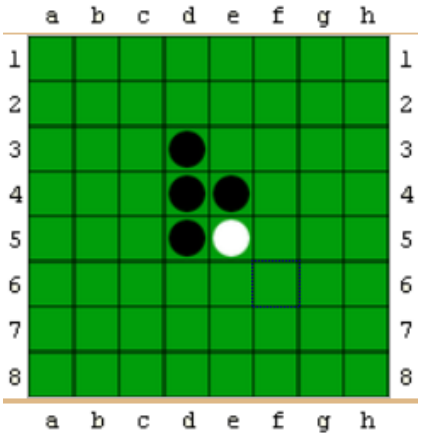
Example of Flips/Captures: In the left image below, the legal Black moves are shown as red cells. After placing the piece at e7, Black turns over (i.e., flips or captures) all White pieces lying on a straight line between the newly added piece and any anchoring Black pieces[1]. All reversed pieces are shown in blue cell in the center image. In the right image, White can reverse those pieces back on his turn if those Black pieces lie on a straight line between the new White piece and any anchoring White pieces.



Example of Flips/Captures: In the left image, the available legal White moves are shown as red cells. After placing a new White piece at d3, all captured pieces are shown in blue cell in the center image. In the right image, Black can use his own turn to reverse those pieces back if he so chooses. However, in this example, Black makes a different move, essentially choosing to flip other pieces.



3.3 Evaluation function: number of pieces



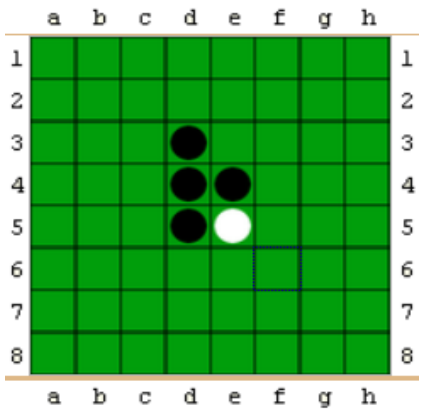
The evaluation function of number of pieces can be computed by

E(s) = #black - #white

For example, the board in the picture above has evaluated value: E(s) = 4-1 = 3

3.4 Evaluation function: positional weights

	a	b	c	d	e	f	g	h
1	99	-8	8	6	6	8	-8	99
2	-8	-24	-4	-3	-3	-4	-24	-8
3	8	-4	7	4	4	7	-4	8
4	6	-3	4	0	0	4	-3	6
5	6	-3	4	0	0	4	-3	6
6	8	-4	7	4	4	7	-4	8
7	-8	-24	-4	-3	-3	-4	-24	-8
8	99	-8	8	6	6	8	-8	99



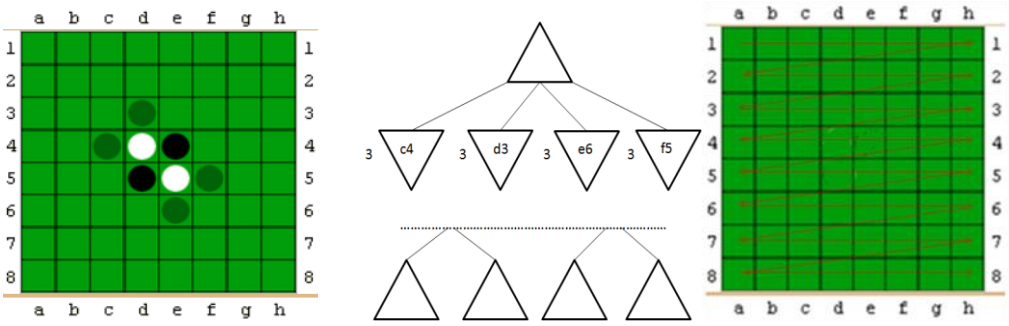
Each position on the board has different strategic value. For example, the corners have higher strategic values than others. This evaluation function assigns different values for each position.

The evaluation function of positional weights can be computed by

E(s) = Weight_black - Weight_white

For example, the board in the picture above has evaluated value: E(s) = (4+0+0+0) - (0) = 4

3.5 Tie Breaking and Expand order



Ties between nodes are broken by selecting the node that is first in positional order on the right figure above. For example, if all legal moves (c4, d3, e6, f5) have the same evaluated values, the program must pick the d3 according to the tie breaker rule on the right figure.

Your traverse order must be in the positional order also. For example, your program will traverse on d3, c4, f5, e6 branch in order.

3.6 Ending the Game

The game ends when all squares are filled in or when both players have no play left.

3.7 Board size

The board size is fixed. It has 8 rows and 8 columns. The number of cells is 64.

4. Input

There are three inputs for your programs;

4.1 Which task to perform: there are three possible values

1 ➔ Minimax using number of pieces as an evaluation function

2 ➔ Alpha-Beta pruning using number of pieces as an evaluation function

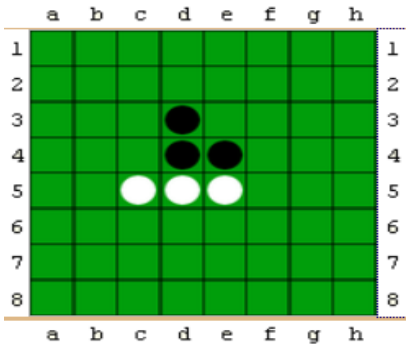
3 ➔ Alpha-Beta pruning using positional weights as an evaluation function

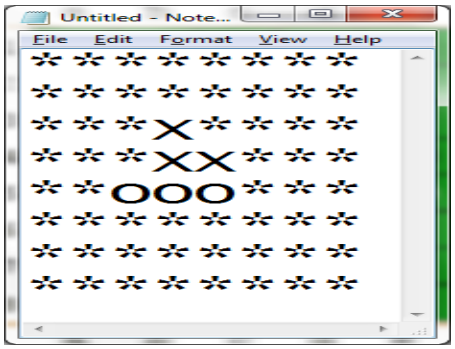
4.2 The cutting off depth

More details in AIMA section 5.4.2

4.3The initial board configuration:

* represents an empty space. X represents a black piece. O represents a white piece. For example, if the initial board configuration is in the left image, the input file contain texts in the right image.

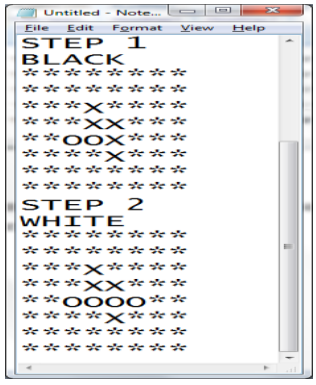




5. Output

In this assignment, we are interested in both how your search algorithms traverse the tree and moves. There are two output files that your program will produce.

5.1 Moves: List the moves at each step in order.



5.2 Logs of the first step: Show complete traverse logs of the first step. Please see the complete examples in output1_tlog_t1.txt, output1_tlog _t2.txt, output1_ tlog_t3.txt, output2_tlog_t1.txt, output2_tlog_t2.txt, output2_tlog_t3.txt.

6 Program Specifications

6.1 Your program must be written in either Java or C++.

6.2 Your program MUST compile and run on aludra.usc.edu

6.3 Write your own code. Files will be compared and any cheating will be reported.

6.4 Your program name must be "reversi" (without quotation mark).

7. Execution Details

Your program will be tested on aludra.usc.edu on unseen input files that meet the input file specifications. Your program will receive 5 arguments, 3 for inputs and 2 for outputs.

7.1 C/C++

The grader will execute this command:

reversi -t <task> -d < cutting_off_depth> -i <input_file> -op <output_path> -ol <output_log>

Example:

reversi -t 1 -d 3 -i input1.txt -op output1_moves_minimax.txt -ol output1_tlog_ minimax.txt

7.2 JAVA

The grader will execute this command:

java reversi -t <task> -d < cutting_off_depth> -i <input_file> -op <output_path> -ol <output_log>

Example:

java reversi -t 1 -d 3 -i input1.txt -op output1_moves_minimax.txt -ol output1_tlog_ minimax.txt

7.3 Arguments:

7.3.1 <task> : there are 3 possible values 1, 2 and 3.

7.3.2 < cutting_off_depth >: the cutting-off depth

7.3.4 <input_file>: location of an input file.

7.3.5 <output_path>: location of an path output.

7.3.6 <output_log>: location of an traverse log output.

Thus, you should interpret the example:

reversi -t 1 -d 3 -i input1.txt -op output1_moves_minimax.txt -ol output1_tlog_ minimax.txt

The first task is chosen. The cutting-off depth is 3. The location of the input file is same as your program and its name is input1.txt. The location of path output file is same as your program and its name is output1_moves_minimax.txt. Finally, the location of traverse log output file is same as your program and its name is output1_tlog_ minimax.txt.

8.1 Programming (90 pts):

Your program must implement the three search algorithms.

8.1.1 Correct outputs for task 1: 30 points

8.1.2 Correct outputs for task 2: 25 points

8.1.3 Correct outputs for task 3: 25 points

8.1.4 Your analysis of similarities/differences result between task1 and task2 and task2 and task3 evaluation functions. Your explanation must be included as part of readme.txt: 10 points

8.2 Readme.txt (10 pts):

8.2.1 A brief description of the program structure and any other issues you think will be helpful for the grader to understand your program. (5 pts)

8.2.2 Instructions on how to compile and execute your code. (5 pts)

8.2.3 Please include your name, student ID and email address on the top.

8.2.4 You must submit a program in order to get any credit for the Readme.txt. In short, if you submit ONLY a Readme.txt file you will get 0.

8.2.5 Remember to also include the explanation of outputs (Part 8.1.4)

9. Submission Guidelines

Your program files will all be submitted via blackboard. You MUST follow the guidelines below. Failure to do so will incur a -25 point penalty.

9.1 Compress and zip ALL your homework files (this includes the Readme.txt and all source files) into one .zip file. Note that only .zip file extensions are allowed. Other compression extensions such as .tar, rar, or 7z will NOT be accepted.

9.2 Name your zip file as follows: firstname_lastname.zip. For example, John_Smith.zip would be a correct file name.

9.3 To submit your assignment, simply select the appropriate assignment link from the Assignments subsection of the course blackboard website. Upload your zip file and click submit (clicking send is not enough).

Please make sure ALL source files are included in your zip file when submitted. Errors in submission will be assessed –25 points. A program that does not compile as submitted will be given 0 points.

Only your FINAL submission will be graded.

For policies on late submissions, please see the Syllabus from the course home page.

10. References

1. http://en.wikipedia.org/wiki/Reversi

2. http://www.samsoft.org.uk/reversi/