CSCI 561: Foundations of Artificial Intelligence

Homework #1: Adversarial Search

Due on February 6 2017, at 11:59pm PST

Introduction

In this project, you will write a program to determine the minimax value for given positions of the *Reversi* game, using the Alpha-Beta pruning algorithm with positional weight evaluation functions.

Rules of the Game

The rules of the Reversi game can be found at http://en.wikipedia.org/wiki/Reversi and interactive examples can be found at http://www.samsoft.org.uk/reversi/. In the Othello version of this game, the game begins with four pieces (two black, two white) placed right in the middle of an 8x8 grid, with the same-colored pieces on a diagonal with each other (see the left side of Figure 1). A move of a player can be either a *valid move* or *pass move*.

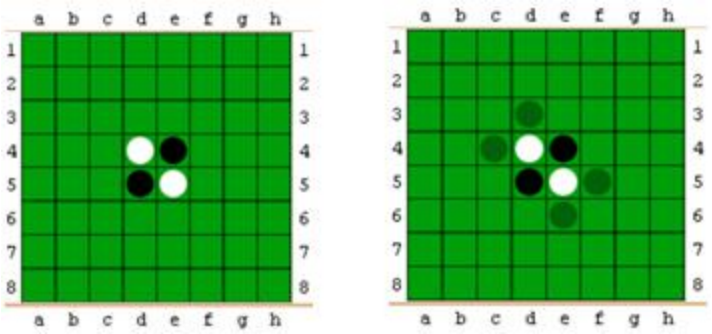


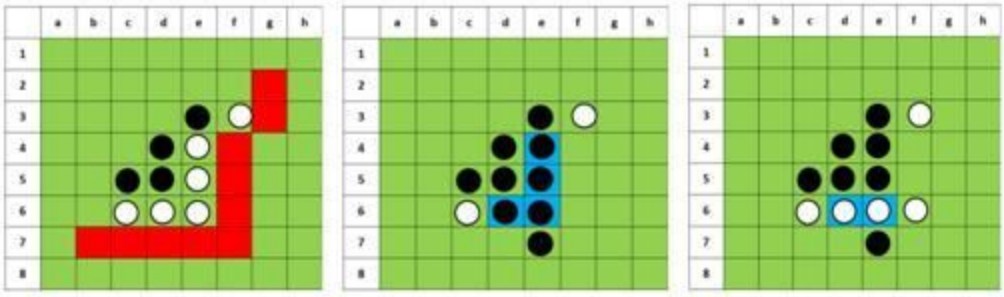
Figure 1

Valid moves and Flips

Assume that the current position is as shown in the left side of Figure 1 and that Black is the current player to move (the valid moves for White are similarly defined). Black can place a black piece on the board only in such a grid cell that there exists at least one straight (horizontal, vertical, or diagonal) line of white pieces between the new black

piece and another existing black piece. The right side of Figure 1 shows all the valid moves available to the Black player at the position as shown in the left side.

Once the new black piece is placed, *all* the line of white pieces that are between the new black piece and another existing black piece are flipped into black pieces. As an illustration, in the left side of Figure 2, the valid Black moves are shown as red cells. After placing the piece at e7, for example, Black flips two lines of white pieces (i.e. the line



e4-e5-e6 and the line d6). All flipped pieces are shown in blue cells in the middle part of Figure 2. Then, the game goes into the White player's turn, and the right side of Figure 2 illustrates the resulting game state after White plays f6 (which flips d6 and e6 into black).

Figure 2

As another example of Flips, in the left side of Figure 3, the valid White moves are shown as red cells. After placing a new White piece at d3, all captured pieces are shown as blue cells in Figure 3 (middle). Then as shown in the right side of Figure 3, Black can use her own turn to flip some white pieces back by playing b6.

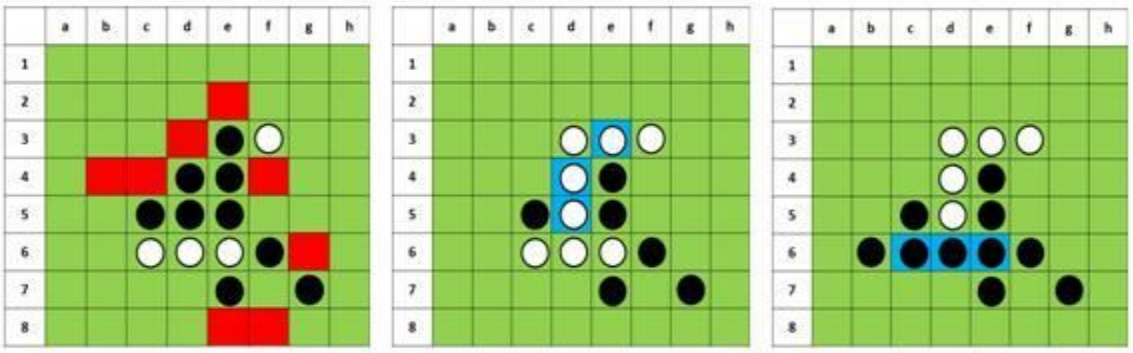


Figure 3

Pass Move and End Game

If one player cannot make a valid move, she must play a *pass* move, which makes no change on the board, and the game goes to the other player's turn. A player cannot pass when she has at least one valid move. When neither player can make a valid move, the game ends. This occurs when the grid has filled up or when neither player can legally place a piece in any of the remaining cells.

Your Task

In this assignment, you will implement the Alpha-Beta algorithm (Figure 5.7, AIMA 3rd edition) to determine the depth-bounded minimax values of given game positions. Your program will take the input from the file *input.txt*, and print out its output to the file *output.txt*. Each input of your program contains a game position (including the board state and the player to move) and a search cut-off depth *D*, and your program should output the corresponding information after running an Alpha-Beta search of depth *D*. That is, the leaf nodes of the corresponding game tree should be either a game position after exactly *D* moves (alternating between Black and White) or an end-game position after less than *D* moves. A leaf node is evaluated by the following evaluation function:

Evaluation function: positional weights

In this evaluation function, each cell i of the board has a certain strategic value W_i. For example, the corners have higher strategic values than other cells. The map of the cell values is shown in the left side of Figure 4.

	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

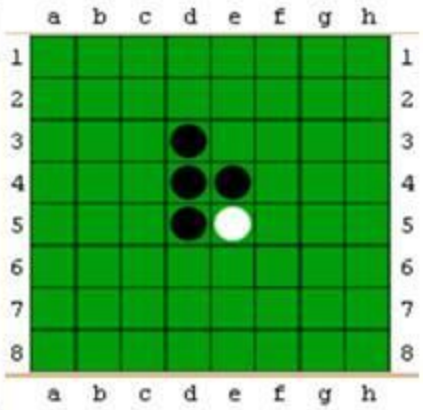


Figure 4

Given these "weights", the evaluation function of a given game position s (with respect to a specific player) can be computed by

$$E(s) = \sum_{i \in \{player's\ cells\}} W_i - \sum_{j \in \{opponent's\ cells\}} W_j$$

For example, the game position in the right side of Figure 4 is evaluated, with respect to Black, as E(s) = (4+0+0+0) - (0) = 4; while it is evaluated with respect to White as E(s) = (0) - (4+0+0+0) = -4.

Note: The leaf-node values are always calculated by this evaluation function, *even though it is an end-game position*. Although this may not be a good estimation for the end-game nodes (and is a deviation from the "official" Reversi rules), you should comply with this rule for simplicity (so that you do not need to worry about possible ordering complications between terminal utility values and evaluation values at non-terminal nodes).

Tie breaking and expansion order

Ties between the legal moves are broken by handling the moves in positional order as shown in the right side of Figure 5 (that is, first favor cells in upper rows, and in the same row favoring cells in the left side). For example, if you need to expand the moves in the position of Figure 5 (left), you should expand them in the order of d3, c4, f5, e6.

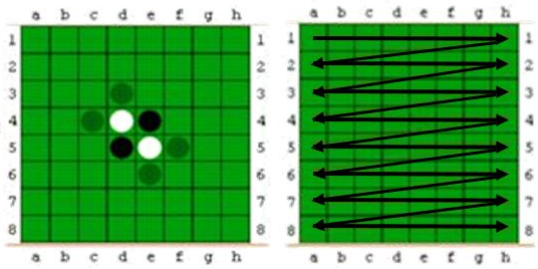


Figure 5

File Formats

Input format:

<player to move>

<search cut-off depth>

<current board status>

where the board status is denoted by * as blank cell, X as black piece, and O as white piece.

Output format:

<next state>

<traverse log>

where the traverse log requires 5 columns. Each column is separated by ",". The five

columns are node, depth, minimax value, alpha, beta.

Example Test Case

As an example, the following input instance asks to compute a depth-2 alpha-beta search from the starting game position:

```
X
2
********
********
********
***OX***
***XO***
********
********
********
```

and the corresponding output should be:

```
********
********
***X****
***XX***
***XO***
********
********
********
Node,Depth,Value,Alpha,Beta
root,0,-Infinity,-Infinity,Infinity
d3,1,Infinity,-Infinity,Infinity
c3,2,-3,-Infinity,Infinity
d3,1,-3,-Infinity,-3
e3,2,0,-Infinity,-3
d3,1,-3,-Infinity,-3
c5,2,0.0,-Infinity,-3
d3,1,-3,-Infinity,-3
root,0,-3,-3,Infinity
c4,1,Infinity,-3,Infinity
c3,2,-3,-3,Infinity
c4,1,-3,-3,-3
root,0,-3,-3,Infinity
f5,1,Infinity,-3,Infinity
```

```
f4,2,0,-3,Infinity
f5,1,0,-3,0
d6,2,0,-3,0
f5,1,0,-3,0
f6,2,-3,-3,0
f5,1,-3,-3,-3
root,0,-3,-3,Infinity
e6,1,Infinity,-3,Infinity
f4,2,0,-3,Infinity
e6,1,0,-3,0
d6,2,0,-3,0
e6,1,0,-3,0
f6,2,-3,-3,0
e6,1,-3,-3,-3
root,0,-3,-3,Infinity
```

Homework Rules

1. **Please follow the instructions carefully. Any deviations from the instructions may lead your grade to be zero for the assignment.** If you have any questions, please use the discussion board. Do not assume anything that is not explicitly stated.
2. For **all** the test cases of this homework, you should implement the Alpha-Beta algorithm. Importantly, you should **not** implement the Minimax Search algorithm.
3. You must use **Python 2.7** to implement your code. You are allowed to use standard libraries only. You have to implement any other functions or methods by yourself.
4. You need to create a file named "hw1cs561s2017.py". The command to run your program would be as follows: (When you submit the homework on labs.vocareum.com, the following commands will be executed.)

   python hw1cs561s2017.py

5. Create a file named "output.txt" and print the outputs there. For each test case, the grading script will paste a "input.txt" file in your work folder, runs your program (which reads input.txt), and check the generated output.txt file from your code with the correct output. (It will replace the files automatically, so you do NOT need to do anything for that part.)
5. You will use labs.vocareum.com to submit your code. Please refer to http://help.vocareum.com/article/30-getting-started-students to get started with the system. Please only upload your code to the "/work" directory. Don't create any subfolder or upload any other files.
6. If we are unable to execute your code successfully, you will not receive any credit.
7. **For your final submission, please do not print any logs on the console other than the required output.**

8. When you press "Submit" on Vocareum, your code will be run against the grading script and the sample input/output files. You will receive feedback on where your code is making errors, if any. You can click "Submit" to test new versions of your code as many times as you like up until the deadline. Only your last submission will be graded. The sample input/output files are designed to cover many basic situations of the problem and rules of the output log, so please utilize them as much as you can.
9. Your program should handle all test cases within a reasonable time (not more than a few seconds for each sample test case). The complexity of the grading test cases will be similar to or less than the five example test cases.
10. You are strongly recommended to submit at least two hours ahead of the deadline, as the submission system around the deadline might be slow and possibly cause late submission, and **late submissions will not be graded**.