CS 540: Introduction to Artificial Intelligence
Homework Assignment # 6

Assigned: 11/1
Due: 11/9 at 11:59 PM

Hand in your homework:

If a homework has programming questions, please hand in the Java program. If a homework has written questions, please hand in a PDF file. Regardless, please zip all your files into hwX.zip where X is the homework number. Go to UW Canvas, choose your CS540 course, choose Assignment, click on Homework X: this is where you submit your zip file.

Late Policy:

All assignments are due at the beginning of class on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 9:30 a.m., and it is handed in between Wednesday 9:30 a.m. and Thursday 9:30 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: Reversi Tiny [100 points]

In this question you will implement game tree search for the game "Reversi Tiny" on a 4x4 board.

Reversi Tiny is a strategy board game for two players, played on a 4x4 board. There are 16 disks, which are dark on one side (player 1's color) and light on the other (player 2's color). At each turn, any disks of the opponent's color that are in one or more straight lines (horizontal, vertical or diagonal) and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. The object of the game is to try to end up with the most disks of your color at the end of the game.

Rules:

A game begins with four disks placed in a square in the middle of the grid, two pieces with the light side up, the other two pieces with the dark side up, with same-colored disks on a diagonal with each other.

```
-----------------
|   |   |   |   |
-----------------
|   | 2 | 1 |   |
-----------------
|   | 1 | 2 |   |
-----------------
|   |   |   |   |
-----------------
```

Note: dark and light is indicated by 1 and 2, respectively.

The dark player moves first and must place a piece with the dark side up on the board, in a position that there exists at least one straight occupied line (horizontal, vertical or diagonal) between the new piece and another dark piece, with one or more contiguous light pieces between them. Dark has the following options indicated by X:

```
-----------------
|   | X |   |   |
-----------------
| X | 2 | 1 |   |
-----------------
|   | 1 | 2 | X |
-----------------
|   |   | X |   |
-----------------
```

If dark decides to put a piece in the leftmost location, the board appears as following:

```
-----------------
|   |   |   |   |
-----------------
| 1 | 1 | 1 |   |
-----------------
|   | 1 | 2 |   |
-----------------
```

```
|   |   |   |   |
-----------------
```

Likewise, light plays next and lay down a light piece, causing a dark piece to flip. Below is possible options for light:

```
-----------------
| X |   | X |   |
-----------------
| 1 | 1 | 1 |   |
-----------------
| X | 1 | 2 |   |
-----------------
|   |   |   |   |
-----------------
```

If light chooses top-right option and reverses one dark piece, the board appears as following:

```
-----------------
|   |   | 2 |   |
-----------------
| 1 | 1 | 2 |   |
-----------------
|   | 1 | 2 |   |
-----------------
|   |   |   |   |
-----------------
```

Players take alternate turns. If one player can not make any valid move, play passes back to the other player (hint: think how to represent the successor in this case). Also, one move may form more than one lines. For example, sometimes a horizontal line and a diagonal line can be formed simultaneously. All opponent's pieces on those lines must be flipped. The game ends when neither player can move. This occurs when the grid has filled up or when neither player can legally place a piece in any of the remaining squares. Then the player with the most disks on the board wins. The game is a tie if both players have the same number of disks. You may want to play the game here `https://www.mathsisfun.com/games/reversi-tiny.html` to get familiar with the rules. For more background on Reversi see `https://en.wikipedia.org/wiki/Reversi`. Note we will be using the simplified rules above, not the riles in Wikipedia.

Write a program **Reversi.java** with the following command line format:

```
$java Reversi FLAG player board
```

where "FLAG" is an integer that specifies the output of the program (see below). "player" denotes who plays at this board, and takes value in 1 or 2. "board" is a string of length 16 that specifies the board in the natural reading order (left to right, top to bottom). Each position in the string takes one of three values: 0=empty, 1=dark, 2=light. Note: for this game, a state is actually the (player, board) pair. For example, with the initial state and FLAG=100, the command line would be

```
$java Reversi 100 1 0000021001200000
```

(Part a, 20 points) When FLAG=100, print out all the successor boards of the given state. The successors should be printed in the natural reading order of the new piece. That is, a successor with the new piece at the upper-left corner should be printed before a successor whose new piece is at the lower-right corner. Each successor board should be printed as a 16-character string on a single line. If the player has no move for this board but it is not a terminal node, you should simply print the board itself. If the board is a terminal node, you should simply produce no output at all. For example,

```
$java Reversi 100 1 2222222222222221
$java Reversi 100 1 0000021001200000
0100011001200000
0000111001200000
0000021001110000
0000021001100010
```

(Part b, 20 points) When FLAG=200, verify if the board is a terminal node. Recall this is true only when neither players can move. If the board is a non-terminal node, print "non-terminal" (without the quotation marks). If the board is a terminal node, print the game theoretic value for it: 1 if dark wins, -1 if light wins, 0 if it is a tie.

```
$java Reversi 200 1 0000021001200000
non-terminal
$java Reversi 200 1 2222222222222221
-1
```

(Part c, 20 points) Implement the Minimax algorithm on slide 14. When FLAG=300, on line one print the game theoretic value for the given (player, board) pair; on line two print the number of states explored by your algorithm. This should be the sum of the number of Max-Value() and Min-Value() calls, and one way to implement it is to have a global counter variable: and you increment the counter as the first line in both Max-Value() and Min-Value(). Note the board now may or may not be a non-terminal node.

```
$java Reversi 300 1 2222222222222221
-1
1
$java Reversi 300 1 0000021001200000
-1
224821
$java Reversi 300 2 0000111001200000
-1
56205
$java Reversi 300 1 0111121111111200
1
6
```

(Part d, 20 points) Note the algorithm does not track the actual move for the player. We ask you to implement it as follows. First, make sure your successors are generated in the natural reading order as in Part a. Then,

when you perform either the max() or min() operations, record the first successor that achieve that optimal value and treat it as the move. That is, even if there are later successors which achieve the same optimal value, those later successors are not chosen as the move. When FLAG=400, print the move (the chosen successor board) for the given (player, board) pair. If the player has no move for the board but the board is a non-terminal, recall the successor board should be the same board, and you should print it. If the board is a terminal node, you should simply produce no output at all.

```
$java Reversi 400 1 2222222222222221
$java Reversi 400 2 2222222222222221
$java Reversi 400 1 0000021001200000
0100011001200000
$java Reversi 400 2 0000111001200000
2000121001200000
$java Reversi 400 1 0111121111111200
1111111111111200
```

(Part e, 10 points) Implement the alpha-beta pruning algorithm on slide 20, but only activate alpha-beta pruning when FLAG $\geq$ 500. When FLAG=500, do the same thing as FLAG=300. You should get the same game theoretic value, but the number of states explored may be smaller.

```
$java Reversi 500 1 2222222222222221
-1
1
$java Reversi 500 1 0000021001200000
-1
7208
$java Reversi 500 2 0000111001200000
-1
482
$java Reversi 500 1 0111121111111200
1
6
```

(Part f, 10 points) When FLAG=600, do the same thing as FLAG=400 but with alpha-beta pruning. Do you get the same moves?

```
$java Reversi 600 1 2222222222222221
$java Reversi 600 2 2222222222222221
$java Reversi 600 1 0000021001200000
0100011001200000
$java Reversi 600 2 0000111001200000
2000121001200000
$java Reversi 600 1 0111121111111200
1111111111111200
```

(Part g, no points) Congratulations! You can now run command-line interface for Reversi Tiny, which simply provides a user interface to allow you to play the game interactively with your code. Note the program requires that you don't change the provided type signatures and methods in Reversi.java.

To run, move provided CLIReversi.java into the same directory where your Reversi.java is located. Then, compile & run CLIReversi.java, choose which player (1 or 2) you play as, and who plays first. When it is your human's turn, enter a single number 1–16 for the board position to put your new piece.

Finally, we provide a few additional examples to help you develop your code.

```
Extra Example 1:
Part A)
$java Reversi 100 1 2100021001200000
2100111001200000
2100021001110000
2100021001100010

$java Reversi 100 2 2100111001200000
2220112001200000
2100211022200000

$java Reversi 100 1 2100211022200000
2100211021201000
2100211021200100
2100211022100010
2100211022100001

$java Reversi 100 2 2100211022100010
2220221022100010
2102212022100010
2100222222100010
2100211022220010
2100221022200012

$java Reversi 100 1 2220221022100010
2220221021101010

Part B)
$java Reversi 200 2 2220221021101010
non-terminal

$java Reversi 200 1 2221222222221010
non-terminal

$java Reversi 200 2 2221222122211011
non-terminal
```

```
$java Reversi 200 1 2221222122111111
0

Part C)
$java Reversi 300 1 2100021001200000
-1
12789

$java Reversi 300 2 2100111001200000
-1
5521

$java Reversi 300 1 2100211022200000
-1
2760

$java Reversi 300 2 2100211022100010
-1
969

$java Reversi 300 1 2220221022100010
-1
71

Part D)
$java Reversi 400 1 2100021001200000
2100111001200000

$java Reversi 400 2 2100111001200000
2220112001200000

$java Reversi 400 1 2100211022200000
2100211021201000

$java Reversi 400 2 2100211022100010
2220221022100010

$java Reversi 400 1 2220221022100010
2220221021101010

Part E)
$java Reversi 500 1 2100021001200000
-1
284
```

```
$java Reversi 500 2 2100111001200000
-1
243

$java Reversi 500 1 2100211022200000
-1
124

$java Reversi 500 2 2100211022100010
-1
226

$java Reversi 500 1 2220221022100010
-1
71

Part F)
$java Reversi 600 1 2100021001200000
2100111001200000

$java Reversi 600 2 2100111001200000
2220112001200000

$java Reversi 600 1 2100211022200000
2100211021201000

$java Reversi 600 2 2100211022100010
2220221022100010

$java Reversi 600 1 2220221022100010
2220221021101010

Extra Example 2:
Part A)
$java Reversi 100 1 2220221021101010
2220221021101010

$java Reversi 100 2 2220221021101010
2220222221101010
2220222022221010
2220221022101210
2220221021201012

$java Reversi 100 1 2220222221101010
```

```
2221221221101010

$java Reversi 100 2 2221221221101010
2221222222221010
2221221222201210
2221221221201012

$java Reversi 100 1 2221222222221010
2221222122211011

$java Reversi 100 2 2221222122211011
2221222122211011

$java Reversi 100 1 2221222122211011
2221222122111111

$java Reversi 100 2 2221222122111111 (Note: produce no output)


$java Reversi 100 1 2221222122111111 (Note: produce no output)


Part B)
$java Reversi 200 1 2221222222221010
non-terminal

$java Reversi 200 2 2221222122211011
non-terminal

$java Reversi 200 1 2221222122211011
non-terminal

$java Reversi 200 2 2221222122111111
0

$java Reversi 200 1 2221222122111111
0

Part C)
$java Reversi 300 2 2220221021101010
-1
70

$java Reversi 300 1 2221222222221010
0
```

```
4

$java Reversi 300 2 2221222122211011
0
3

$java Reversi 300 1 2221222122111111
0
1

Part D)
$java Reversi 400 2 2220221021101010
2220222221101010

$java Reversi 400 1 2221222222221010
2221222122211011

$java Reversi 400 2 2221222122211011
2221222122211011

$java Reversi 400 1 2221222122111111 (Note: produce no output)


Part E)
$java Reversi 500 2 2220221021101010
-1
70

$java Reversi 500 1 2221222222221010
0
4

$java Reversi 500 2 2221222122211011
0
3

$java Reversi 500 1 2221222122111111
0
1

Part F)
$java Reversi 600 2 2220221021101010
2220222221101010

$java Reversi 600 1 2221222222221010
```

```
2221222122211011

$java Reversi 600 2 2221222122211011
2221222122211011

$java Reversi 600 1 2221222122111111 (Note: produce no output)
```