CS 540-1: Introduction to Artificial Intelligence
Homework Assignment # 5

Assigned: Thursday, 10/13
Due: Thursday, 10/27 before class

Hand in your homework:

This homework is a programming assignment. Please compress all your source code into a zip file named **wiscusername_hw5.zip**. You are required to **use only built-in libraries** to complete this assignment. You may define your own classes for this assignment. **Do not include any package statements** in your submission. We expect that main() is in **TicTacToe.java.** TicTacToe.java must also include a header with: **your name, Wisc username, class section, Homework #, submission date, and if handed in late, how many days late it is.**
Go to Learn@UW My Course Dashboard, choose the 540 course, choose Assignments/Dropbox, click on hw5: this is where you submit your 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. A total of two (2) free late days may be used throughout the semester without penalty.

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

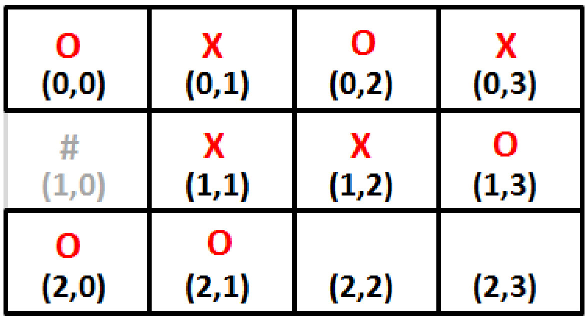
In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1:

For this assignment, you will be implementing alpha-beta pruning in a generalized Tic Tac Toe game.

The board is shown in Figure 1, with the grid positions labeled as (row, column). Importantly, (1,0) is a forbidden position: no one can place a piece on that. Xs are played by the computer, and the human player plays Os. The human always plays first.

Given a current state for the board where the next move is X's, your task is to make the best next move. That is, you need to find the best possible placement of X using an alpha-beta search from the current state down to the leaves. Note that the current state need not be an optimally-played configuration and can have multiple pieces on the board. (See example in Figure 1).



O (0,0)	X (0,1)	O (0,2)	X (0,3)
# (1,0)	X (1,1)	X (1,2)	O (1,3)
O (2,0)	O (2,1)	(2,2)	(2,3)

Figure 1: Game Board Current State

The rules of the game are:

1. That # represents the forbidden position on the board.
2. Three consecutive X's or O's in the same line (horizontal, vertical), or diagonal are considered to be the corresponding player's winning states.
   Note: 4 consecutive X's or O's on the same row are not considered winning states.
3. States where O wins are scored +1.
4. States where X wins are scored -1.
5. States where neither player wins are scored 0.
6. To break ties (if any) between multiple child nodes that have equal game theory values, pick the child in this order:
   (a) (First) (0,0)
   (b) (0,1)
   (c) (0,2)
   (d) (0,3)
   (e) (1,1)
   (f) (1,2)
   (g) (1,3)

(h) (2,0)
(i) (2,1)
(j) (2,2)
(k) (Last) (2,3)

Program Specifications

1. The input configuration for the board is given through command line arguments separated by spaces, in the order mentioned above from (0,0) to (2,3). To represent blank spaces in the board, underscore characters are used. A # character represents a forbidden position on the board.
   Example input (for the board configuration in Figure 1):

   ```
   java TicTacToe O X O X # X X O O O _ _ Y
   ```

   Note: We will not give your program invalid inputs; do not worry about input error-handling.

2. The last command line input is either an upper-case Y or an upper-case N. When this input is Y, your program must print out the alpha-beta search trace, along with their alpha and beta scores **right before returning** from the recursive alpha-beta pruning algorithm. Print a # for the unused grid position (1,0). States do not need to be printed if this input is N.
   Example (sample run with flag = Y):

   ```
   java TicTacToe O X O X # X X O O O _ _ Y
   O X O X
   # X X O
   O O X O
   Alpha: -2 Beta: 2
   O X O X
   # X X O
   O O X _
   Alpha: 0 Beta: 2
   O X O X
   # X X O
   O O _ X
   Alpha: -2 Beta: 0
   O X O X
   # X X O
   O O _ _
   Alpha: -2 Beta: -1
   SOLUTION
   O X O X
   # X X O
   O O _ X
   ```

   Note: Each row of the board is printed on one line, with a space separating the grid values. **Use +2 and -2 for ∞ and $-\infty$ for your implementation of the alpha-beta pruning algorithm.**

3. Your program must print "SOLUTION" on a new line and output the state which represents the best next move for X.
   Example (Sample run with flag = N):

```
java TicTacToe O X X O # _ O X _ _ O _ N
SOLUTION
O X X O
# X O X
_ _ O _
```

Sample Run 2:

```
java TicTacToe O X O X # O X O _ _ _ _ Y
O X O X
# O X O
X O X O
Alpha: -2 Beta: 2
O X O X
# O X O
X O X _
Alpha: 0 Beta: 2
O X O X
# O X O
X O _ X
Alpha: -2 Beta: 0
O X O X
# O X O
X O _ _
Alpha: -2 Beta: -1
O X O X
# O X O
X _ O _
Alpha: -1 Beta: 2
O X O X
# O X O
X X _ O
Alpha: 1 Beta: 2
O X O X
# O X O
X _ _ O
Alpha: 1 Beta: -1
O X O X
# O X O
X _ _ _
Alpha: 1 Beta: 2
O X O X
# O X O
```

```
_ X _ _
Alpha: -2 Beta: 1
O X O X
# O X O
O _ X _
Alpha: -2 Beta: -1
O X O X
# O X O
_ _ X _
Alpha: 1 Beta: -1
O X O X
# O X O
_ _ _ X
Alpha: -2 Beta: -1
O X O X
# O X O
_ _ _ _
Alpha: -2 Beta: -1
SOLUTION
O X O X
# O X O
_ X _ _
```