CS 540-1: Introduction to Artificial Intelligence
Homework Assignment # 4

Assigned: Thursday, 9/27
Due: Thursday, 10/4 before class

Hand in your homework:

This homework is a programming assignment. Please hand in the Java program `RingTreblecross.java`, completing the code skeleton we have provided. 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. Go to UW Canvas, choose your CS540-1 course, choose Assignment, click on Homework 4: this is where you submit your file.

Question 1: (n, k)-Ring Treblecross

For this assignment, you will be implementing the minimax algorithm in order to play the game of (n, k)-Ring Treblecross.

(n, k)-Ring Treblecross is a variant of tic-tac-toe with the following key differences:

1. It is played on a ring-shaped board with n positions.
2. Both players place X pieces on the board.
3. The game is won when a player achieves k OR MORE consecutive X pieces on the board. See below for an example of (12,3)-Ring Treblecross.

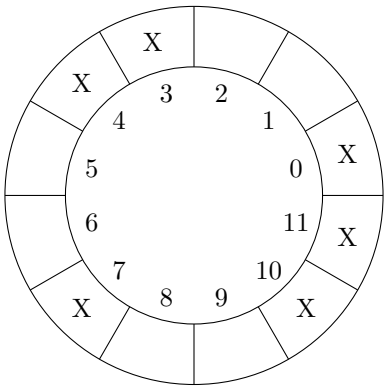

Figure 1: A completed game of Ring Treblecross for $k = 3$.

In your implementation, both n, the size of the board, and k, the number of contiguous pieces that need to be placed for the game to be won, will be specified as command line parameters to your program.

Given a current state for the board where the next move is yours, your task is to make the best next move. That is, you need to find the best possible placement of X using a minimax 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.

The rules of the game are as follows:

1. The game is played by two players who alternate turns placing an X on the board at some unoccupied square.
2. A player wins by achieving k OR MORE consecutive pieces on the board.

Your mission, should you choose to accept it...

1. To complete this assignment, it is sufficient to complete the code template marked by "TO DO" in the provided source file.
2. You are free to define any helper methods you would like. However, you should not define any additional member variables or class variables as they will likely only increase the chances of unexpected bugs in your implementation.
3. In particular, you **must not** modify any other method bodies other than the 4 methods we have marked in the source.

Program Specification

1. Command line arguments specify the following: the board size n, the parameter k of contiguous pieces defining the least winning condition of the game, an operating flag which will determine the output of the program, and a variable-length list of integers defining the configuration of the board. For example:

   ```
   java RingTreblecross 6 3 0 1 5
   ```

   will begin a game on a board of size 6 where 3 contiguous pieces OR MORE are required to win the game. The operating flag `0` causes the program to execute an interactive game where you can play against your code. The first three arguments are required. The remaining (optional and variable length) arguments `1 5` specify the initial board configuration. As you can see positions `1` and `5` are occupied by X pieces. And you are prompted to input your next move.

   ```
   Initial :  |   | X |   |   |   | X |
   Choose Next Move:
   ```

   Inputting `0` is a winning move here, and if you enter it, the program will terminate. Otherwise, the opponent will move and you will be prompted again for your move, until the game is over.

2. A state where MAX player wins is scored +1; A state where MIN player wins is scored -1.
3. We require the code to return a list of successor from `getSuccessors` method in a specific order. In particular, the successor must be sorted by the index of the "filled-in" square. For example, the initial state

   ```
   |   | X |   |   |   | X |
   ```

should produce a list of successors in this order:

```
0: | X | X |   |   |   | X |
1: |   | X | X |   |   | X |
2: |   | X |   | X |   | X |
3: |   | X |   |   | X | X |
```

This is because successor 0 filled in the empty square at index 0 in the initial state, successor 1 filled in the square at index 2, and so on.

4. It is possible that multiple successors of a state have the same score. For tie breaking, the `maxValueAndBestSuccessor` and `minValueAndBestSuccessor` methods should use the first successor with the best score from getSuccessors method as the best successor.

5. The following operating flags are defined:

   (a) `0`: Play an interactive game against the algorithmic opponent beginning at the specified position. Human moves first. The code template will call your code when it is your code's turn to move. See above for an example.

   (b) `1`: Print all the successors of the board position specified on the command line as well as their game-theoretic score. Your code is ALWAYS THE MAX PLAYER. For example, `java RingTreblecross 3 3 1`, on a correct implementation, should print the following output:

   ```
   Min Node  | X |   |   |  Score:  1
   Min Node  |   | X |   |  Score:  1
   Min Node  |   |   | X |  Score:  1
   ```

   (c) `2`: Print a trace of an optimal sequence of moves beginning from the board position specified at the command line. For example, `java RingTreblecross 7 3 2`, on a correct implementation, should print the following output:

   ```
   Max Node  |   |   |   |   |   |   |   |  Score: -1
   Min Node  | X |   |   |   |   |   |   |  Score: -1
   Max Node  | X |   |   | X |   |   |   |  Score: -1
   Min Node  | X | X |   | X |   |   |   |  Score: -1
   Max Node  | X | X | X | X |   |   |   |  Score: -1
   ```

   (d) `3`: Print the best successor of the board position specified on the command-line. For example, `java RingTreblecross 10 3 3 5`, on a correct implementation, should output:

   ```
   Min Node  | X |   |   |   |   | X |   |   |   |   |  Score:  1
   ```

6. Note that all the code to parse the command line arguments has been written for you. We will not test your code on invalid inputs. Do not worry about input error-handling.

7. The board is represented in the provided code skeleton as of type `boolean[]`, an array of boolean values. Think of the squares of a board of length n as indexed 0 to $n-1$, reading from left to right. The i-th element in the boolean array represents the i-th squre of the board and contains the value `true` if the corresponding board position contains an X.