Lab 3: Tic-Tac-Toe

Due Date:
11:59PM, Wednesday September 30th
Email zipfile “LastName-Lab3.zip” to cse436ta@gmail.com

Assignment – 10 points

In this lab, you will design and implement the popular kid’s game Tic-Tac-Toe. To start off with, you may want to familiarize yourself with the game and rules: [http://en.wikipedia.org/wiki/Tic_tac_toe](http://en.wikipedia.org/wiki/Tic_tac_toe)

In this lab, you will be responsible for implementing the game with a simple GUI only (i.e. no graphics for pieces or the board). This lab will require you to get comfortable with user actions.

**ATTENTION:** Many of the implementation decisions for the lab are left up to you. We will only cover the basic requirements. You are free to add on as much as you want, provided the basic requirements are met.

Task Overview

Your primary task for this lab will be to allow a user to play Tic-Tac-Toe. You must implement both one-player and two-player versions.

You should start with the two-player version (human versus human). After the first player moves, the human controlled second player must make a move of his own (at which point play reverts back to the first player, etc.).

After you get this part working, you will implement the one player version (human versus computer). In this version, you will design a simplistic artificial intelligence to act as the second player. Your computer player should follow the priority guidelines listed below (it should do step 1 before steps 2, 3, or 4, etc.)

1) If the computer can win in the next step (i.e. it has two in a row, and there exists an empty spot to make it 3 in a row) it should make the winning move.
2) If the computer cannot win in the next step, but the player can, the computer should block the winning spot. If the player can win in more than one way, the computer should block one of the possible winning moves.

3) If neither the player nor the computer can win in the next move, the computer should play a move that makes some sense. If the computer is the first to move, it should place a move into the center tile (unless you wish to use a different strategy). The computer should try to go for two in a row when possible, and block the player when needed.

After every move, a function should be called that checks to see if a player has won. If there is a winner, you should output a message declaring that player one or player two has won. You should then ask the user whether or not to play a new game. If the user wishes to play a new game, you should ask whether the game is to be a one or two player game, and then you should ask who should move first (you must include the option to have a random first move).

Some example prompts:

>>Player one has won. Do you wish to play a new game?
>>Do you wish to play a one or two player game?
>>Who should move first? (1, 2, or Random)

The user should also answer the last two questions when the application first launches. At the start of each game, the piece assigned to each player should be randomized (with a 50% chance the user receives a given piece).

After each move, you should specify who is next to move and also update the game board to display the current layout of the pieces.

The internal structure of the game board is left to you. We recommend either using a matrix (perhaps as an array of arrays), or a single array of 9 positions. The first element of the matrix would refer to spot 1 (the top left), etc. This way the inputted move corresponds perfectly with the element of the array/matrix.

A simple way to define each cell is by using an enumerated data type. Each cell would thus become X if an X is there, O if an O is there, or E if the spot is empty. You may use a different data type if you want.
HINT: Your storyboard file should only contain a subset of buttons, labels, text fields, and switches. The best approach would be to have 9 buttons, each one corresponding to a tile on a Tic-Tac-Toe board. The button’s text should either be blank, or show an X or O. If a user enters an invalid move (i.e. clicks on a non vacant position), you should mention the error to the user and have them repeat a move. However, feel free to use your own design ideas.

Grading

The usual automatic deductions apply.

6 Points – The structure of the game environment works correctly. The user can choose between the required options (new game, who moves first, etc.). The user should be able to move easily. A user shouldn’t be allowed to place a piece on a non-vacant tile.

2 Points – The game correctly detects when a player has won.

2 Points – The “Artificial Intelligence” (versus computer option) behaves as it should.