Project 4 - Tic Tac Toe

CS121 Computer Science I

Overview

The *TicTacToe* interface defines methods for a class that manages game state information and logic for a game of tic-tac-toe. *TicTacToeGame* implements the *TicTacToe* interface.

The *TicTacToeTester* test class is provided for confirming correct functionality of the *TicTacToeGame* class.

TicTacToeGUI is a graphical user interface (GUI) for interactively playing a game against the computer, just because it's fun.

Objectives

- Write a class that uses 1D and 2D arrays to keep track of games of Tic Tac Toe and implements a testable interface.

Getting Started

- Create a new folder in VS Code named "TicTacToe".
- Download and import interface *TicTacToe.java* and classes *TicTacToeTester.java* and *TicTacToeGUI.java*. (all ZIPped)
- Create a new class *TicTacToeGame.java* that implements *TicTacToe.java*.
  ```
  public class TicTacToeGame implements TicTacToe { }
  ```

Development Strategy

Before writing a single line of code, read **all** project specifications and **all** given source files carefully and completely. Pay special attention to the javadocs in interface *TicTacToe* to understand what is expected from each method.

As soon as you have created the class *TicTacToeGame*, run the test class *TicTacToeTester* to confirm that it compiles and runs. At first, most tests are expected to fail. When you have completed *TicTacToeGame*, all tests should pass.

While *TicTacToeTester* will be the definitive judge of your *TicTacToeGame*'s correct implementation, it would be a very valuable exercise for you to **write your own driver class** to interact with your game class under the conditions you want to test.

Example ad hoc driver testing strategy:

1. Create a new `TicTacToeGame`
    a. Call `gameOver()`, `getGameState()`, `getGameGrid()`, and `getMoves()` methods and see what you get.
    b. Can you call `choose()` with any player at any position as the first move?
2. Make a first choice as player X.
    a. Call all the game info methods, again, and see what you get.
    b. Can the same player make two choices in a row?
    c. Can the other player claim the spot the first player claimed?
3. Repeat all of 2. but start as player O.
4. Build up games with valid moves by each player and test that you recognize a win in each row, column, and diagonal as soon as it happens.
    a. Confirm that no moves can be made after a win has occurred.
5. Build up a tie game and confirm that you recognize a tie.
    a. Confirm that no moves can be made after a tie.
6. Confirm that `newGame()` resets everything back to the starting conditions.

Specification

TicTacToe

This interface defines required methods for your *TicTacToeGame* class. The required functionality for each method is described in its javadoc. Do **not** modify this interface.

TicTacToeGame

This is the class you must write. It must correctly implement the *TicTacToe* interface to enforce all rules of the game to pass all tests in *TicTacToeTester*. You must use arrays only (no ArrayLists) for storing and working with game data - a 2D **BoardChoice** array for the game grid, and a 1D **Point** (from java.awt) array for the history of moves.

Important: The correctness of your *TicTacToeGame* implementation will be determined by passing *TicTacToeTester* - not by whether a game can be played in the GUI.

TicTacToeTester

This driver class confirms *TicTacToeGame* correctly implements the *TicTacToe* interface. Do **not** modify this class.

TicTacToeGUI

This driver class is provided as an example GUI application for playing your *TicTacToeGame*. You do **not** need to edit anything in this class, but because this class is given only as an

example front-end for interacting with a game, you are allowed to modify it if you wish. It will not be used to verify correct functionality of your *TicTacToeGame* in any way.

- When the GUI program is first launched, it is ready for a new game. All board positions are OPEN and no moves have been made.
- The user goes first as player X. The computer (player O) will have a turn after each player turn until there is a winner or all spaces have been selected.
- A turn ends when the player has chosen an available space. (Selection of an already-occupied space does nothing.)
- On the computer's turn, a random, unoccupied space is chosen.
- After every turn, the game grid is analyzed to see if either the player or computer has won with 3-in-a-row - across, down, or diagonally - or if the last space has been claimed with no winner.
- When a game has ended with a winner or tie, the history of moves and the final result are displayed in the GUI's `JTextArea`.
  It is not possible to continue interacting with a game after a game-over condition has been reported.
- At any time, if the *New Game* button is pressed, the game grid, moves array, and displayed history of moves are reset and the game is waiting for the user to make their first move.

Testing

Test your program frequently throughout development to quickly catch errors. Run the following tests from the command line in the folder you intend to submit **on onyx** and make certain they pass before submitting:

```
# Check that all code compiles
javac *.java
# Check that TicTacToeTester runs and all tests pass
java TicTacToeTester
```

Documentation

If you haven't already done so, add **javadoc comments** to each class and all methods that are not overriding one of the *TicTacToe* interface methods.

- Javadoc *class* comments should include, at minimum, a meaningful description of the class and an @author tag.
  - The class javadoc comment is placed before the class header.
  - The class description must be the first content in the javadoc, before any tags.

    - Your class comment must include the @author tag at the end of the comment. This will list you as the author of your software when you create your documentation.
- Javadoc *method* comments should be placed before ALL methods (including constructors) and include a meaningful description and any appropriate tags.
    - Methods that were defined in a well-documented interface (usually identified by an @Override tag before the method header) do not require a new javadoc comment since they inherit the javadoc from the interface.
    - The method description must be the first content in the javadoc, before any tags.

Include a plain-text file called **README.md** that describes your program and how to use it. Expected formatting and content are described in README_TEMPLATE.md. See README_EXAMPLE.md for an example.