CMPSCI 187 / Spring 2015
Tic-Tac-Toe

Due on 2015-02-05 08:30 EST

Marc Liberatore and John Ridgway

Morrill I N375
Section 01 @ 10:00
Section 02 @ 08:30

Contents

Overview

For this assignment, you will implement a specification (a commented Java interface) for the game of Tic-Tac-Toe.

Tic-tac-toe is a simple, deterministic, adversarial two-player game with no hidden knowledge. Traditionally, it is played on a 3 $\times$ 3 board, where the X player (who goes first) and the O player alternate turns, claiming spaces on the board. The first player to claim three spaces in a row (horizontally, vertically, or diagonally) wins the game. The game concludes in a draw if neither player is able to do so.

The game can be generalized to an $n \times n$ (where $n \geq 3$) board. In the general game, n spaces in a row claimed by the same player are required to win. To receive full credit for this assignment, you must implement a generalized $n \times n$ game of Tic-Tac-Toe.

Your implementation will enforce the rules of the game between two players, but it will not implement an AI opponent.[1]

Learning Goals

- Demonstrate mastery of CMPSCI 121 material as applied to a new problem.
- Show ability to break a larger problem down into manageable pieces.
- Read a specification in text and as a Java interface, and demonstrate understanding by implementing it.
- Use existing JUnit test cases.
- Write new JUnit test cases to test specified behaviors.

General Information

Read this entire document. If, after a careful reading, something seems ambiguous or unclear to you, then email cs187help@cs.umass.edu immediately. Do not wait until 5pm the night before the assignment is due to tell us you don't understand something, as our ability to help you will be minimal.

Reminder: Copying partial or whole solutions, obtained from other students or elsewhere, is academic dishonesty. Do not share your code with your classmates, and do not use your classmates' code.

You are responsible for submitting project assignments that compile and are configured correctly. If your project submission does not follow these policies exactly you may receive a grade of zero for this assignment.

Policies

- For some assignments, it will useful for you to write additional class files. Any class file you write that is used by your solution MUST be in the provided `src` directory you export.
- The TAs and instructors are here to help you figure out errors, but we won't do so for you after you submit your solution. When you submit your solution, **be sure to remove all compilation errors from your project**.

[1] We may do this later in the semester, or you can investigate it yourself if you're curious. Start by reading about Minimax.

Any compilation errors in your project will cause the autograder to fail, and you will receive a zero for your submission.

Test Files

In the `test` directory, we provide several JUnit test cases that will help you keep on track while completing the assignment. We recommend you run the tests often and use them to help create a checklist of things to do next. But you should be aware that we deliberately don't provide you the full test suite we use when grading.

We recommend that you think about possible cases and add new `@Test` cases to these files as part of your programming discipline. Simple tests to add will consider questions such as:

- Do your methods taking integers as arguments handle positives, negatives, and zeroes (when those values are valid as input)?
- Does your code handle unusual cases, such as empty or maximally-sized data structures?

More complex tests will be assignment-specific. To build good test cases, think about ways to exercise methods. Work out the correct result for a call of a method with a given set of parameters by hand, then add it as a test case. Note that we will not be looking at your test cases, they are just for your use.

Before submitting, make sure that your program compiles with and passes all of the original tests. If you have errors in these files, it means the structure of the files found in the `src` directory have been altered in a way that will cause your submission to lose some (or all) points.

Problem 1

Import Project into Eclipse

Begin by downloading the starter project and importing it into your workspace. It is very important that you **do not rename** this project as its name is used during the autograding process. If the project is renamed, your assignment will not be graded, and you will receive a zero.

By default, your project should have no errors and contain the following root items:

`src` This is the source folder where all code you are submitting must go. You can change anything you want in this folder (unless otherwise specified in the problem description and in the code we provide), you can add new files, etc.

`support` This folder contains support code that we encourage you to use (and must be used to pass certain tests). You must not change or add anything in this folder. To help ensure that, we suggest that you set the support folder to be read-only. You can do this by right-clicking on it in the package explorer, choosing Properties from the menu, choosing Resource from the list on the left of the pop-up Properties window, unchecking the Permissions check-box for Owner-Write, and clicking the OK button. A dialog box will show with the title "Confirm recursive changes", and you should click on the "Yes" button.

Problem 1 continued on next page. . .

`test` The test folder where all of the public unit tests are available.

`JUnit 4` A library that is used to run the test programs.

`JRE System Library` This is what allows Java to run; it is the location of the Java System Libraries.

If you are missing any of the above or errors are present in the project, seek help immediately so you can get started on the project right away.

What to Do

As with the previous assignment, we have provided several files. The specification for the game is contained in the file `support/tictactoe/TicTacToeGame.java`. You must not edit this file, but you should read it carefully. In it, you will see signatures and descriptions for six methods: `getN()`, `toString()`, `getWinner()`, `getCurrentPlayer()`, `isValidMove(int)`, and `move(int)`. To complete this assignment, you must correctly implement these six methods and a constructor.[2]

A game starts when an instance of a class implementing `TicTacToeGame` is created. Then, `move` is called repeatedly. Each time it is called, the current player claims the space indicated by the argument to `move`. The other methods are used to get other details of the game state: if there is a winner, who the current player is, whether a given move is allowed, and so on. The details are in `support/tictactoe/TicTacToeGame.java`.

Next, open `src/tictactoe/TicTacToe.java`. This file contains a class that implements `TicTacToeGame`. But, it will quickly become apparent that the implementation is both minimal and wrong. You must correct the implementation. Do not change the name of this class, nor that it implements `TicTacToeGame`, nor that the constructor takes a single argument of type **`int`**. Otherwise, you have free reign.

Correcting `TicTacToe` will require you to make decisions about how data representing the game's state will be stored and manipulated. You will likely want to declare and use instance variables. You may also want to declare one or more private methods. We are deliberately leaving the details up to you, though we expect you will need to use one or more arrays to deal with boards of arbitrary size.[3]

As in the previous assignment, you can use the tests in `test/tictactoe/TicTacToeTest.java` to help guide your efforts. But remember that there are other tests, particularly for larger boards, that we will be using to assess your submission. There is also a `TicTacToeRunner` that illustrates the use of the `TicTacToe` class. But it expects a correct implementation of `TicTacToeGame` in `TicTacToe`, and won't function as you'd expect until your implementation in `TicTacToe` is at least somewhat correct.

Grading

The autograder will use a large set of automated tests to grade your assignment. The tests verify that your implementation of `TicTacToeGame` behaves as specified in the assignment.

[2] We are not going to test `toString()`, but you will almost certainly have trouble debugging if you don't implement it.

[3] The solution we wrote for `TicTacToe.java` is about 150 lines long. If you find your code is significantly longer (over 300 lines), you may want to stop and reassess your approach, or come talk to a TA or instructor.

Problem 1 continued on next page...

Approximately 70% of the total points for this assignment will be based upon your implementation behaving correctly with a 3×3 board. The remaining 30% will be for correctly implementing arbitrary ($n \times n$) boards. **We strongly suggest you get the 3×3 case working first**, then turn your attention to boards of arbitrary size.

Export and Submit

When you have completed the changes to your code, you should export an archive file containing the entire Java project. To do this, click on the `tictactoe-student` project in the package explorer. Then choose "File → Export" from the menu. In the window that appears, under "General" choose "Archive File". Then choose "Next" and enter a destination for the output file. Be sure that the project is named **`tictactoe-student`**. Save the exported file with the `zip` extension (any name is fine). Log into Moodle and submit the exported zip file.