Software Quality Assurance and Testing, NEU, 2018

Laboratory: Test-Driven Development and Refactoring

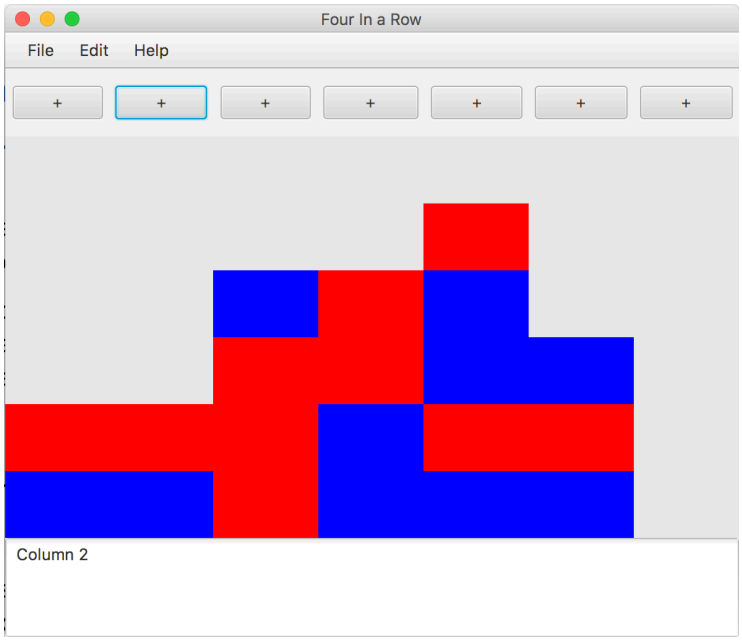
In this laboratory, you will use Test-Driven Development (TDD) to develop a set of tests for a game of Four in a Row. You are provided with the existing code.

Your task will be to make some changes to the code to separate out the game logic from the GUI Controller code. You will then develop some new features based on example stories[1].

Develop the code using the TDD approach. This means following the Red-Green-Refactor stages. You will probably use Red and Green the most. You will start with some Refactoring steps to tidy up the existing code that you are given.

1 Your Task

You are going to work on a game called Four in a Row. A basic version of the game has been created. You are going modify the existing code and add some tests to check that modification. You will then use TDD to add new features to the game.



Firstly, download the existing game from https://github.com/digidol/FourInARow-JavaFX. The game is built with maven, which we discussed in the session about Continuous Integration (CI).

The README.md file, https://github.com/digidol/FourInARow-JavaFX/README.md, covers some information about the build options. The build process can be used to compile the software, run the tests, generate a code-coverage report, create a runnable jar file and generate Javadoc pages.

Use the Maven build file to control the build. Your chosen IDE may help with this.

[1] A story is a short statement about a function the system should do. It is not as formal as a requirement, but it is a similar idea.

The project from GitHub has the following structure:

- **pom.xml** – the Maven build file. This contains the instructions to tell Maven what it does to build the application and run the tests.
- **src** – a folder that contains the source code for the game and a file of tests for one of the classes.

The project also includes a LICENCE file and README.md file.

Access the project and import it into your preferred IDE – see the notes from Laboratory 1 about how to do that for Eclipse and IntelliJ. You can also run this from the command line – you will need to have your command line path setup so that you can run the mvn command, which is the Maven command line program.

We will talk about the structure of the application in the laboratory.

1.1.1 Changes to the system

The first task is to refactor the `FourInARowController` class. It has a mixture of code to manage the GUI and code that represents the game logic.

Create a new class, e.g. `FourInARowGame`, that will represent the board and the game logic. Then, look for parts of the code in `FourInARowController` that are managing the board or the game logic.

For example, the `initialiseBoard()` method and part of the `handleButtonAction()` method could be moved from `FourInARowController` to `FourInARowGame`.

The `FourInARowController` would be changed to use the `FourInARowGame`.

As you move those methods, add some tests.

1.1.2 Adding new functionality

Once you have changed the system to use a Game class, you can look at some new features.

On the last page of this document, you will see a list of stories. A story is a short description of part of the requirements. Stories are used to remind teams to talk about what is required.

In this laboratory, discuss these stories with your classmates. What do you think needs to be done for each story? When you have discussed the stories with others, choose the ones that you want to develop.

On your own, develop the stories that you have chosen. Develop them using the TDD approach, remember Red-Green-Refactor. Use JUnit to write the tests.

1.1.3 Scope of your work

When you are working on this laboratory, focus your tests on the game logic.

You can add other classes into the application if you decide they will help your design.

You are not expected to test the GUI and you are not expected to modify the GUI code.

2 Workshop Workbook

The workbook will be marked. It contributes 20% of the marks for this course. In this laboratory, I want you to get experience of using Test-Driven Development and Refactoring.

> **IMPORTANT:**
> In your answers, I want to see your ability to discuss the issues. I do not need to see very long descriptions – the workbook will provide guidance on expected length of your answers. I do want to see the discussion in your own words. The workbook must use your words, not those of other people in the class or anything from the Internet or books.
>
> **If the workbook contains text that has been copied from somewhere else, you will receive a penalty on your score.**

For the code, I am checking to see that you have done a reasonable number of tests for the given time. There is no set number of tests. I want to see that you have made sensible choices about refactoring the code and adding tests for the code that you are developing.

The marks will be for the following categories:

- **Code – 50%.** Mark is based on good choice of tests and the names and structure of the tests. There will be some marks for the restructuring of the code to provide a game class.
- **Discussion in the Workbook – 50%.** Mark is based on the comments you make in your workbook in response to the questions.

3 Estimated Time

We have spent some time on this in the Laboratory. You will continue the work in your own time over the next 12 days. In addition to the time spent in the Laboratory, it is suggested that you spend up to 8 more hours on the work and the workbook. You might not need all of that time. Don't spend longer than that number of hours.

4 Submission

Create a Zip file with the following contents:

1. Your project directory. This should include the source code and the tests as well as the pom.xml file. Please don't submit the target directory. I can rebuild that by running pom.xml on your project.
2. A copy of your workbook, as a PDF or Word document.

This should be uploaded to Blackboard by **8pm on Wednesday 3rd October.**

> **IMPORTANT:**
> Check the file you are submitting to make sure it has all of the files that you have been asked to provide. If the files are missing, it will have an impact on your mark.

4.1 Questions

If you have any questions about this work, please speak to Neil or ask your question on Slack or the Blackboard Forum. I will try to check the Forum at least once a day, but I might forget. If I have not answered a question, please email me to tell me that there is a question on the forum. My email is nst@aber.ac.uk.

5 Stories for the Four in A Row game

This section lists a number of different stories. They are numbered for the purpose of this exercise. A team using stories on an agile project may not number the stories.

There is no order to this list of stories. Some stories will depend on others. To satisfy some of these stories, we may need to add some new features.

Some of the stories are already implemented, e.g. stories 2 and 5, or some are partially implemented, e.g. story 8. In your work, show that these stories are still working after you have made your changes.

1. After a player has introduced a square, the game should check if the player has won the game. A player can win in one of three ways:
    1. Four consecutive squares arranged vertically.
    2. Four consecutive squares arranged diagonally
    3. Four consecutive squares arranged horizontally.
2. It should not be possible for a player to have two consecutive turns.
3. A player can undo the last move.
4. A player can redo the last move that was undone.
5. It should not be possible to add more squares into a column than there are empty spaces.
6. Add a feature where you can play a computer player. The computer player should take its turn. You don't need to use any artificial intelligence. You can just come up with some workable rules on how to choose which move to play next.
7. The application should reset the board and show that the board is empty, ready for a new game.
8. Add option to allow a computer player or a two-player game.
9. At the end of the game, it should be possible to see a full list of the moves. These can be shown in the message area at the bottom of the window.

NOTE: Everyone should attempt stories 1.1 (checking vertically) and 1.3 (checking horizontally). Then, choose other stories that you will work on in the available time.