Victoria University of Wellington
School of Engineering and Computer Science

SWEN221: Software Development

Lab Handout — CONNECT

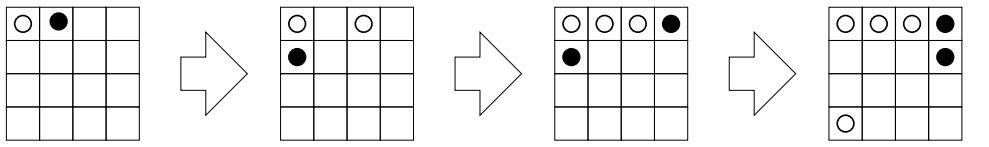
The purpose of this lab is to perform *white-box testing* for a simple program implementing the fictional board game CONNECT. The given implementation contains a number of hidden bugs, and you must write test cases to uncover these. The Emma tool for reporting code coverage should be used to help with this. Before the end of the lab, you should submit your solutions via the *online submission system* which will automatically mark it. You may submit as many times as you like in order to improve your mark and the final deadline will be Friday @ 23:59.

1 CONNECT

Sam and Mel purchased the board game CONNECT from their local games store. The rules are:

> "The game of CONNECT is played on a board divided into 16 squares, arranged in a 4x4 format. There are two players: white and black. Each player starts with 8 tokens in their respective colour (i.e. white or black). Initially, the board is empty. Then, each player takes it in turn to place one of their tokens on the board. The first player to get four of their tokens in a straight line is the winner (note, diagonal lines don't count). A player's token is "captured" when it becomes sandwiched between two of the other player's tokens; captured tokens are removed from the board and cannot be used again (note, diagonal sandwiches do not count). The game continues until one player is the winner."

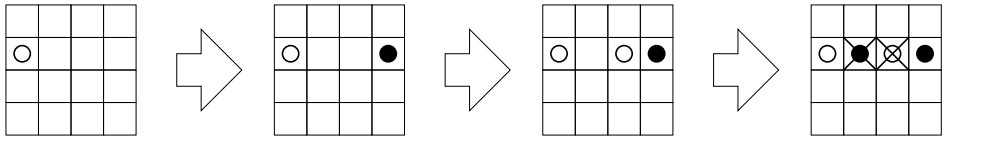
Sam and Mel are playing CONNECT. The following is the sequence of turns taken so far:



(after 1st turn) (after 2nd turn) (after 3rd turn) (after 4th turn)

Here, Mel (white) placed the first piece to begin the game. After the 2nd turn, we see that Sam's piece was captured by Mel using a *short capture* move. After the 3rd turn, we see that Sam has blocked Mel from winning. After the 4th turn, we see that Mel has captured another of Sam's pieces, this time using a *long capture* move.

Mutual Capture. An interesting feature of CONNECT is that it allows for a *mutual capture* move. This is when both players capture a piece from their opponent at the same time. The following illustrates:



(after 1st move) (after 2nd move) (after 3rd move) (during 4th move)

In the final move above, black places his token between the two white tokens, and his/her token is immediately captured by white. At the same time, one of the white tokens is sandwiched between two black tokens and is also captured.

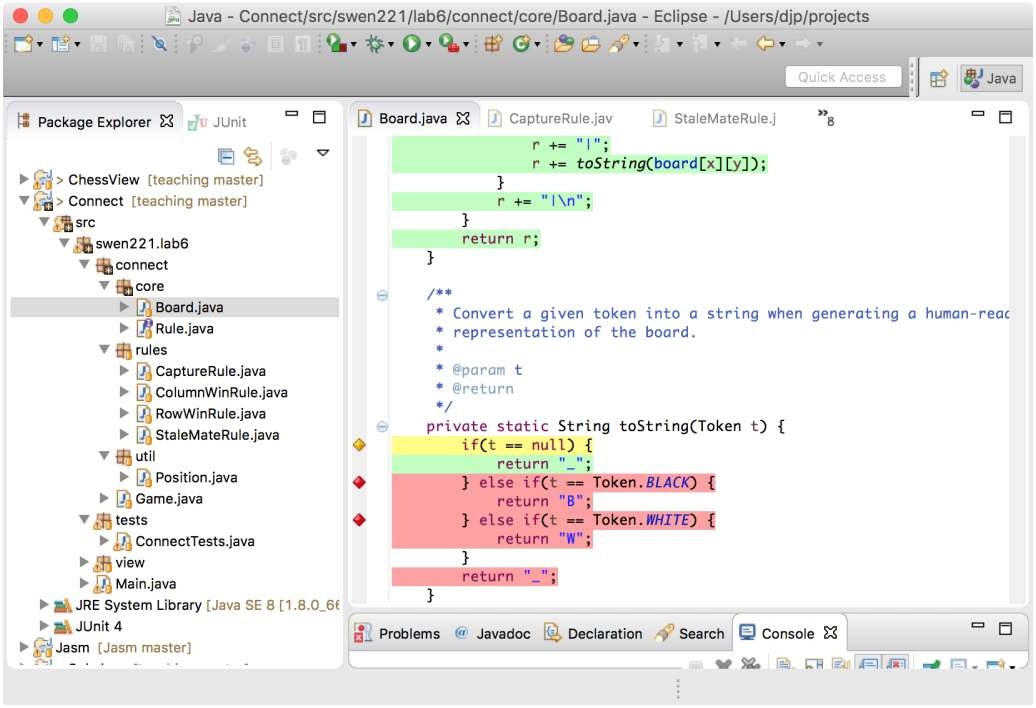


Figure 1: Illustrating the Emma tool being run under Eclipse. Green lines indicate those have been covered by the tests, whilst red lines are those which were not covered. Finally, yellow lines indicates partial coverage which typically occurs for conditionals where e.g. only one side of the conditional is taken.

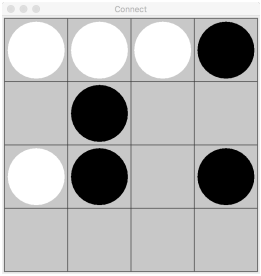
2 Emma

Emma is a *code* coverage tool: for a given test it will tell you how much of your code is covered by that test. Emma is already installed in Eclipse and should be easy to use.

To find the coverage for a test, right click on a JUnit test file, and select "coverage as" and then "JUnit test". The JUnit tests will be run as normal, but in the lower window pane there are test coverage results for the whole program. You should navigate through the package/class hierarchy to find the class or methods you are interested in. You can see pecentage coverage results. By clicking a class or method name, that class will be opened in the main code window and test coverage will be shown graphically: lines highlighted green are covered by the executed tests, lines highlighted red are not covered, lines highlighted yellow are partially covered — that is, some statements on that line are covered and some aren't. The screenshot below shows what this looks like.

3 Getting Started

To get started, download the file `connect.jar` from course website and import them into Eclipse. This provides an implementation of the CONNECT game which has a number of hidden problems. A very simple *Graphical User Interface (GUI)* is provided for playing a game of CONNECT:



To use the GUIw, run the class `swen221.connect.Main` as a "Java Application". With the GUI, you can place tokens by clicking on a square and placing either a white token, or a black token.

4 What to do

The given implementation of CONNECT contains a number of subtle problems which have been specifically chosen as they are hard to spot. Whilst you could simply "eye-ball" the code in detail to find these problems, you will likely this difficult and time-consuming. Instead, the goal of this lab is to gain experience writing test cases to identify (and then eliminate) these problems. This is simulating a *white-box* testing environment where you have access to the source code and can exploit this knowledge to develop test cases with good *code coverage*.

You will notice that the provided tests class `swen221.connect.tests.ConnectTests` contains only *one* test case. Your job is to write more test cases based on the rules of the game and the implementation. If your test cases obtain *good code coverage* you should be able to easily identify and fix the problems. When you submit your code, it will be run against a *hidden* set of test cases and the scored reported. *However, you will not be given information about why any particular test failed.* Instead, you must think carefully about what should and should not be allowed in the game and develop test cases accordingly. The Emma tool can be used to help here, as it will tell you how much code coverage is obtained with your test cases.

HINT: You do not need to test the class `GraphicalUserInterface` (as it is hard to do this).

HINT: Using Emma, identify regions of code not covered by your tests. Then, add tests for them!

HINT: Be sure to test for *invalid* moves as these should be detected and reported.

HINT: The documentation provided for different methods indicates what they should and should not do.

HINT: If you find code which cannot be reached, then ask: *where should it be used?*

Submission

Your lab solution should be submitted electronically via the *online submission system*, linked from the course homepage. The required files are:

```
swen221/connect/core/Board.java
swen221/connect/core/Rule.java
swen221/connect/core/Game.java
swen221/connect/rules/CaptureRule.java
swen221/connect/rules/ColumnWinRule.java
```

```
swen221/connect/rules/RowWinRule.java
swen221/connect/rules/StaleMateRule.java
swen221/connect/util/Position.java
```

You must ensure your submission meets the following requirements (which are needed for the automatic marking script):

1. **Your submission is packaged into a jar file, including the source code**. *Note, the jar file does not need to be executable*. See the following Eclipse tutorials for more on this:

   `http://ecs.victoria.ac.nz/Support/TechNoteEclipseTutorials`

2. **The names of all classes, methods and packages remain unchanged**. That is, you may add new classes and/or new methods and you may modify the body of existing methods. However, you may not change the name of any existing class, method or package. *This is to ensure the automatic marking script can test your code*.

3. **All JUnit test files supplied for the assignment remain unchanged.** Specifically, you cannot alter the way in which your code is tested as the marking script relies on this. This does not prohibit you from adding new tests, as you can still create additional JUnit test files. *This is to ensure the automatic marking script can test your code*.

4. **You have removed any debugging code that produces output, or otherwise affects the computation.** *This ensures the output seen by the automatic marking script does not include spurious information*.

Note: Failure to meet these requirements could result in your submission being reject by the submission system and/or zero marks being awarded.