U08186 (Advanced Object-Oriented Programming) Coursework for 2014

This coursework has two parts. For Part I, you will write a GUI for the strategy game of FourPlay using the Model View Controller design pattern. This is divided into two submission points. Part IA requires the Model and is worth 25% of the module marks. Part IB requires the Views and Controller and is worth 20% of the module marks. Part II is an exercise in design patterns and is worth 5% of the module marks.

Learning Outcomes

This coursework will assess the following learning outcomes.

- Create a software artefact by applying the methodologies of advanced object-oriented programming to a requirements specification
- Select and apply a design pattern from one of the major design patterns to solve a given problem
- Consult on-line code libraries to find the classes and methods most appropriate for solving a particular problem
- Create appropriate documentation that clearly communicates the intended behaviour of a program

Submission and Feedback Timetable

All three submissions should be uploaded to Moodle in the form of a Word or PDF document, with code pasted from several files where appropriate.

Thurs Week 7, 11pm Submit Part IA

Thurs Week 8, 11pm Submit Part II

Fri Week 8, 9am Present solution to Part II. Get spoken feedback

Fri Week 8, 11am Get spoken and written feedback to Part IA

Fri Week 9, 11am Get written feedback to Part II

Mon Week 11, 11pm Submit Part IB

Fri Week 12, 9am Complete Module Evaluation survey for discussion in lecture

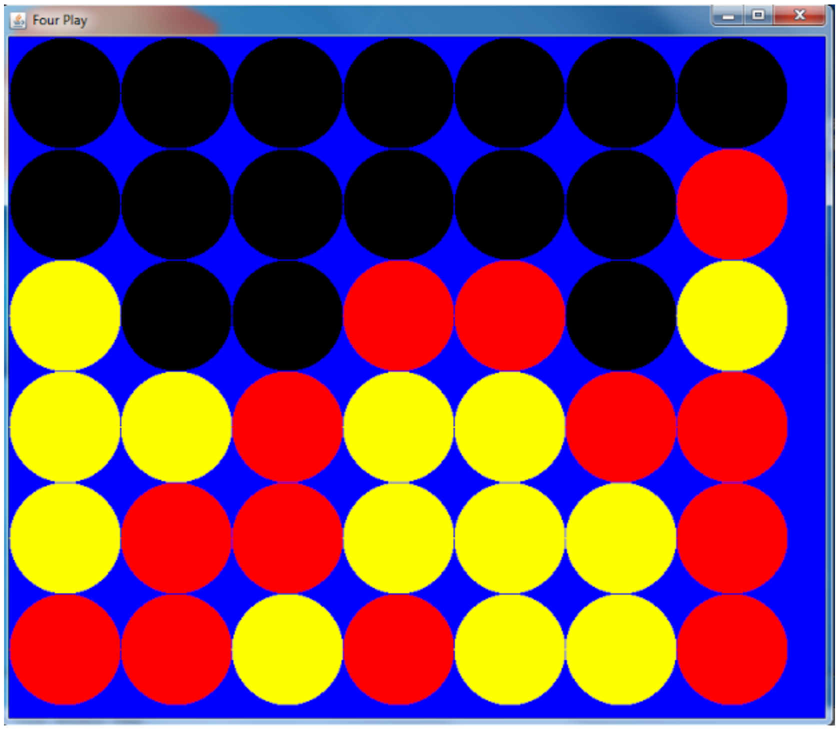
Fri Week 12, 11am Get spoken and written feedback to Part IB.

Part I
The FourPlay Strategy Game

For this part, you will write a program in Java using Model View Controller (MVC), submitting first the Model (specified, unit tested and translated into C#) and then the Views and Controller. Be warned that getting a Java GUI to work is worth only a small number of the available marks since this coursework assesses what you have learned on the module. See the marking scheme for further detail.

Description of the Game

FourPlay is a strategy game in which two players take turns to drop discs of two different colours down columns. The objective of the game for each player is to end up with four discs of his own colour horizontally, vertically or diagonally. For example, below, the red player has just won the game by building up a line of four red discs starting from the bottom-left.



Description of the GUI

A player adds a disc to a column by clicking on the column somewhere between the top and where the disc would land. As well as the view shown above, called the Game Board, there should also be a second view, ie a separate window or separate panel of the same window, called the Score Board, that displays the number of games won by each player. It should have two buttons: one to reset the score and another to end a game. Both should be initially disabled. Because it is an advantage to go first, the identity of the player that does so must switch from game to game.

Marking Scheme for Part IA

5% Model, implementing, for the highest marks, all the required functionality with an interface designed to be convenient for the Controller and View to use. It should have no references to those two classes and contain no GUI code. To receive marks at all, and to ensure high code quality, it should implement a separate method that finds out whether there is a winning line of four discs of the same colour between two given points.

5% specification of Model in either JML or Spec#, including invariants for the class as well as pre and post conditions for each method. This will be marked according to how many of the relevant conditions are included and the correctness of the JML / Spec#. Partial credit will be available for describing them in English. Some statements may be unprovable due to the limitations of JML / Spec# even when specified correctly. In such cases, there will be no negative affect on your marks.

5% unit testing of the Model in JUnit. There should be three tests, significantly different from each other, and none related to the screenshot. Each test should examine the state of the board after a sequence of interactions initiated by the players.

5% translation of the Model to C#. The mark given to this will equal to mark given to the Java version of the Model.

5% use of the code quality practices described in Lecture 1, plus light relevant commenting and correct formatting.

Marking Scheme for Part IB

5% Controller, which must forward only valid requests to the Model, querying the Model if necessary to find out if the request is valid, and must also enable / disable the two reset buttons. It must have no GUI code, though it may send messages to the View. It will be marked with respect to these requirements.

5% Game Board View, with highest marks for a functional attractive View with that communicates with the Model in the correct way and has good quality code, as defined in the marking scheme for Part IA. It is not necessary to animate the falling of the discs.

5% Score Board View, marked in the same way as the Game Board View.

5% enabling the computer to replace one of the players so that the remaining player plays against the computer, which implements some rudimentary artificial intelligence. This will be marked according to the level of functionality and code quality.

Part II
Design Pattern Exercises

For this part, you need to find examples of four different design patterns, from internet sources, library books etc. If you find it difficult to find an example, you can make up your own example. You cannot use an example from the lecture notes. Include the class diagrams in your report.

For each design pattern, explain carefully the conditions for a design to satisfy the design pattern and then give the values to which the existentially quantified variables are bound to show that it does. See the slide Checking that AttackDemo Conforms to Strategy for an example. Since Strategy has already been done for you, you need to choose one of the remaining twelve design patterns. Each pattern is worth 1% of the module mark.

You also need to translate one of the classes from one of your class diagrams into Java. If all the examples you have found already have code then select a diagram with an inheritance hierarchy, add another subclass and translate that into Java. This is worth 1% of the module mark.

You will talk about the design patterns you have found in the lecture. Please note that it is not necessary to produce slides and the original PDF/Word document you submitted will be sufficient as a visual aid. The module leader will provide this.