Victoria University of Wellington
School of Engineering and Computer Science

SWEN222: Software Design

Assignment 2

Due: Sunday 23th August @ Midnight

1 Introduction

In this assignment, you will build a Graphical User Interface for the cluedo game. As with the previous assignment, you have the option to:

1. **Working on your own.** In such case, all work must be your own.
2. **Working in a pair.** In such case, you may work together with another partner on the project (this could be the same partner as before, though does not have to be). You must register your intent to do this using the team signup system:

   `http://ecs.victoria.ac.nz/cgi-bin/teamsignup`

NOTE: In marking the assignment, no distinction will be made between students who worked in pairs and those who did not. Therefore, there is a clear advantage for students who chose to work in pairs. This is because the workload will be shared between them. We wish to encourage you to work in pairs, as this will help develop good teamwork skills and better prepare you for the group project.

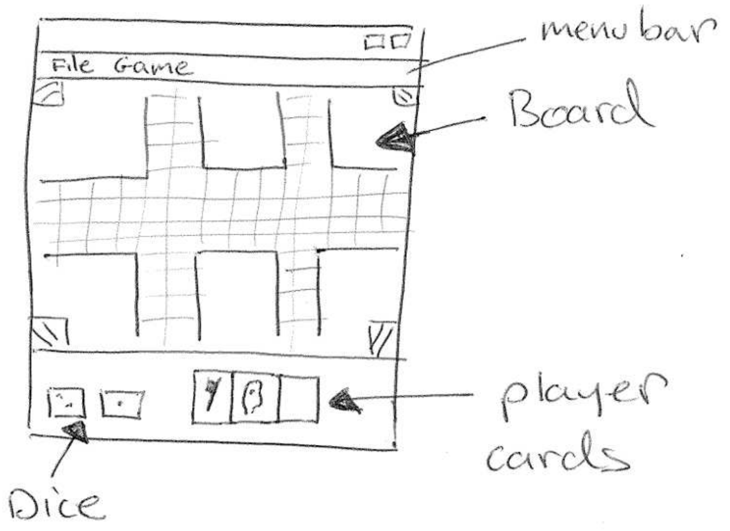
2 Requirements

In this lab, you are required to use at least the following Swing components, although you may use more:

- `JMenuBar` and `JMenuItem`. A menu bar for your cluedo game must be provided.
- `JButton`. Buttons should be used in your GUI. For example, you might provide a button to start the next turn and/or roll the dice.
- `Canvas`. Your GUI should use a `Canvas` to draw the current state of the game, including the cluedo board itself and the position of all players on that board. The location of all weapons should also be visible, as should those cards held by the current player.
- `JTextField`. These should be used to allow each player to enter their name.

- `JRadioButton`. Radio buttons should be used when entering the details of a player. In this case, the available characters (e.g. Miss Scarlet, Professor Plum, etc) should be displayed using radio boxes, thus allowing the user to choose the one they wish. Characters which have already been chosen should not be selectable (and ideally, should be grayed out).
- `JDialog`. Dialogs should be used in a number of places. For example, when the user attempts to close the main window, a dialog should check that they wanted to do this. Similarly, when the game starts, a dialog should prompt the user to input the name of each player, and their token.
- `MouseListener`. A mouse listener should be used to received events from the mouse. For example, the user should be able to move to a square by clicking on it (provided he/she has rolled a sufficiently high number).

An example layout illustrating the main GUI window is given below:



Your GUI window does not have to be exactly the same as this, although it should be similar.

3 Fancy Stuff

There are a number of ways in which you can make your GUI more interesting, and extra marks will be awarded for this. These include:

- **Hovers**. In this case, hovering the mouse over various items on the board should produce little pop-up messages. For example, hovering over a players token will show the players name.
- **Short-Cut Keys**. Modern GUIs provide short-cut keys enabling experienced users to operate the program more efficient. For example, a short-cut key could be used to start the next turn, and to access menu items. A `KeyListener` is needed to implement this.
- **Animations**. Providing an animation of certain events in the game will make it more fun. For example, when a player's token is being moved to another square, you might animate the motion rather than moving it there immediately. Similarly, you could have the tokens themselves be animated to perform different actions when different events happen to them (e.g. when going through a secret passage).

- **Resizeable Display**. A more complicated problem is to make the main window resizeable, along with all of the items being displayed (e.g. the board). In this case, the size of items on the board will depend upon the size of the main window. The `ChessView` program used in SWEN221 provides a good example of how to do this.

4 What to Do

If you have not used Swing before, you may find the following tutorial helpful:

`http://download.oracle.com/javase/tutorial/ui/index.html`

5 Submission

Your program code should be submitted electronically via the *online submission system*, linked from the course homepage. Your submitted code should be packaged into a jar file, including the source code (see the export-to-jar tutorial linked from the course homepage). Your program should include appropriate Javadoc comments, and provide JUnit tests where appropriate. In particular, all existing JUnit tests from Assignment 1 should still pass.

In addition to the source code, various pieces of design documentation are required. These should be submitted as either PDF or PNG files; other formats will not be accepted. The required documents are:

1. Sequence diagram illustrating what happens when the user clicks on the board.
2. An object diagram illustrating the structure of the main GUI window.
3. A two-page report written in your own words giving an overview of the design for your graphical user interface. This should include brief discussions on how the GUI is organised and, in particular, how the main display (e.g. the board and other graphical components) is drawn. Finally, you should also describe the sequence of events that occurs when a user clicks on the board area.

NOTE: The written report must be your own work. That means you cannot submit a report which is identical to that of your other team member(s). Furthermore, it is not acceptable to simply copy material from other sources, such as the internet or text books.

6 Assessment

This assignment will be marked as a letter grade (A+ ... E), based primarily on the following criteria:

6.1 Individual Report [15 marks]

For the individual report, marks will be awarded on an *individual basis* as follows:

- **Grammar and Spelling (4 marks)**. Reports should be free from typographical and related errors.
- **Design Discussion (6 marks)**. Marks will be awarded for how well the individual report conveys aspects of the program's design. Marks will also be awarded for how well-written and easy to follow it is.
- **Discussion of Click Sequence (5 marks)**. Marks will be awarded for how well the individual report describes the sequence of events that occurs when the user clicks on the board area.

6.2 Group Design [15 marks]

For the design documents, marks will be awarded on a *group basis* as follows:

- **Object Diagram (5 marks)**. Marks will be awarded for correct use of the following: *inheritance*, *multiplicity*, *associations*, *classes* and *attributes.* Marks will also be awarded for *clarity.*
- **Sequence Diagram (10 marks)**. Marks will be awarded for clarity and presentation of the sequence, as well as correct use of the notation. Marks will also be awarded for how well the diagram conveys the prescribed event.

6.3 Group Implementation [40 marks]

For the main implementation, marks will be awarded on a *group basis* as follows

- **Execution (5 marks)**. Marks will be awarded for programs which execute correctly without crashing. This includes appropriate handling of invalid input (e.g. text entered when a number is expected, etc).
- **Swing Components (14 marks)**. Marks will be awarded for programs which make appropriate use of the required Swing components, as set out in §2.
- **Fancy Stuff (11 marks)**. Marks will be awarded for good use of additional features which improve the overall quality of the Graphical User Interface.
- **Code Style (5 marks)**. Marks will be awarded for overall code style which includes: good use of naming for methods, fields, classes and variables method naming, field naming, classes and variables; good division of work into methods, so as to avoid long and complex chunks of code; consistent formatting (e.g. indentation, etc).
- **Commenting (5 marks)**. Marks will be awarded for good use of JavaDoc comments on all public classes, interfaces and methods. Marks will also be awarded for good use of internal (i.e. non-javadoc) comments. Typically, these are found within a method body describing some aspect of how it works. Comments should make sense and correctly describe what is happening.