Victoria University of Wellington
School of Engineering and Computer Science

SWEN225: Software Design, Assignment 1

Due: Tuesday, 6 August, before midnight

1 Outline

You are to implement a prototype of program allowing people to play a version of the board game Cluedo on a desktop computer. If you have not heard of the game Cluedo before, don't worry, a specification detailing what you need to do is provided and you may always consult the game manual or the Wikipedia description of Cluedo.

You will design a solution using CRC cards and UML, and implement a solution using Java. Feel free to use the Umple tool for designing models and generating code. You may fully stay within Umple, or depart from the original Umple ethos of never editing the generated code by instead using the generated code as a starting point for your implementation.

Hint: If letting Umple assist you for the implementation part appeals to you, note that you will have to develop a deeper acquaintance with Umple than Lab 1 was designed to achieve.

2 Specification

What follows is a specification for a simplified version of Cluedo. Please bear this in mind whenever referring to the original manual or other sources. True to life, the specification may be incomplete or ambiguous in parts and in such cases you will have to make reasonable assumptions which you should then explicitly state in the assignment report.

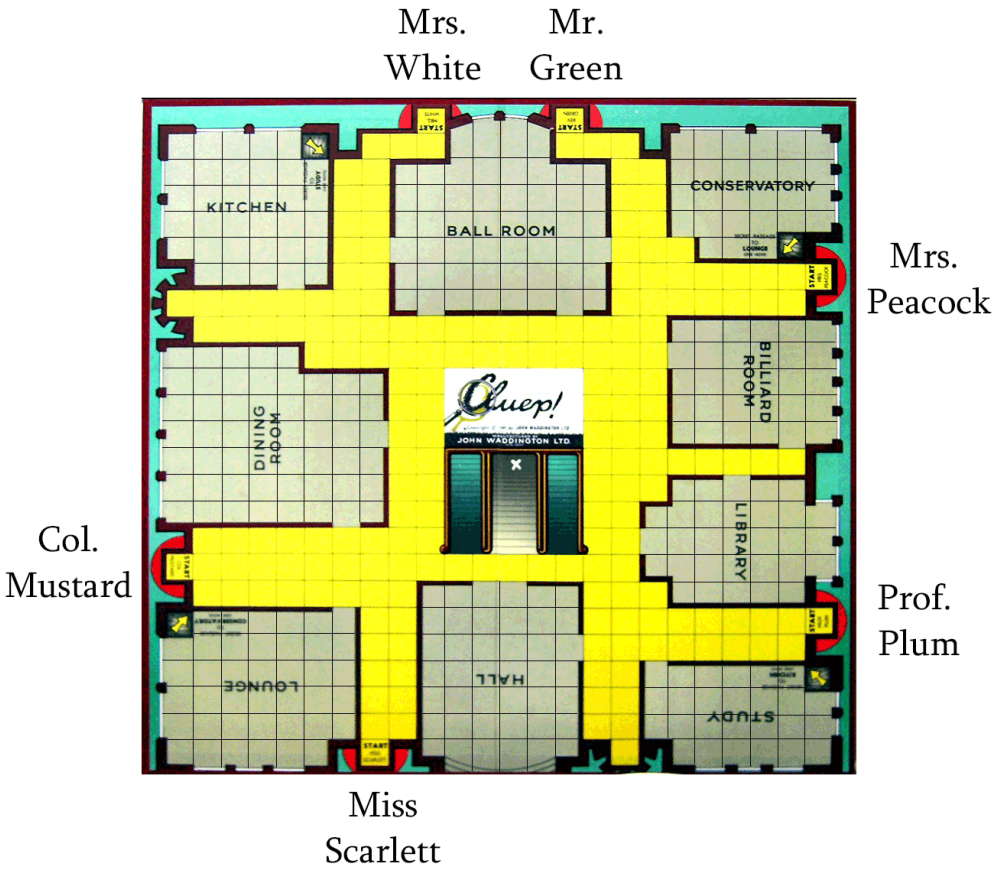
2.1 Objective

The game is a murder mystery played by three to six players who move around a board comprising nine rooms. The aim is to deduce the murder circumstances, i.e., who the murderer was, what weapon they used and in which room they committed the murder.

The Cluedo board consists of nine rooms laid out in a circular fashion. The centre space (the "Cellar") is inaccessible to players.

2.2 Board

The board is divided into a grid of 24x25 squares of which some are not accessible and some (those within rooms) may optionally be treated as a single location.



Players move on this grid and the prototype does not need to honour the fact that rooms are technically not divided by grid cells in the original game. The prototype furthermore does not need to account for the "secret passages" (stairwell connections which connect rooms in opposite corners).

There are six starting squares located at the perimeter of the board. Each starting square indicates which character starts at that position.

2.3 Characters

There are six characters, one of which (randomly selected for each game play) is the murderer:

- Miss Scarlett
- Colonel Mustard
- Mrs. White
- Mr. Green
- Mrs. Peacock
- Professor Plum

Each player assumes the role of one of these characters. If there are less then six players, some of the characters will be unused.

2.4 Weapons

There are six weapons in the game, one of which is the murder weapon:

- Candlestick
- Dagger
- Lead Pipe
- Revolver
- Rope
- Spanner

Each weapon is initially placed in a room chosen at random, such that no two weapons are in the same room.

2.5 Rules

Every character, weapon and room is represented by a card in the game. Before the game starts, one character, one weapon, and one room card are blindly selected at random. This selection represents the murder circumstances, i.e., the "solution" that players need to figure out during game play. The respective cards go into an envelope to be hidden from view.

The remaining weapon, room and character cards are then combined into one stack, shuffled and then dealt facedown evenly to players. Some players may end up with more cards than others.

Players then take it in turns to roll two dice and move their character token a corresponding number (sum of the dice values) of squares. Diagonal movement is not allowed and no space may be used twice during one turn. When a player enters a room, they do not need to use any remaining moves they have left. They may then hypothesise about the murder circumstances by making a *suggestion* which comprises the room they are in, a character and a weapon. If the character and weapon named in the suggestion are not in that room yet, they are now moved into the room.

When a suggestion is made, each player, in a clockwise fashion, starting from the current player, attempts to refute the suggestion. A suggestion is refuted by producing a card that matches one of the named murder circumstances (as such a card cannot be in the solution envelope). A refutation card is only shown to the player that made the suggestion. If a player has multiple refutation cards, it is their choice which one they pick. If no player can produce a refutation, the named murder circumstances are a potential solution candidate that may or may not be used to make an *accusation* later on (by any player).

An *accusation* comprises a character, a weapon, and a room (which can be any room, not just the one the player making the accusation may be in). If the accusation made by a player exactly matches the actual murder circumstances (only the accusing player is allowed to see the solution) the player wins, otherwise the player is excluded from making further suggestions or accusations. This means the player will continue to refute suggestions by others but cannot win the game anymore.

2.6 User Interface

Implement an object-oriented program for playing the Cluedo game. The game interface should be simple and must be text-based. Only text-based input and output is permitted, i.e., all input/output must occur via `System.in` and `System.out`.

1. The program begins by asking how many players wish to participate.
2. At the start of each turn, the program rolls the two dice to determine the move distance of the player who's turn it is by using the sum of the dice values. The current player then moves their token to a desired spot on the grid.
3. Once a player has moved, they are presented with the option of making a suggestion or an accusation. All rules of the games must be enforced at all times, e.g., only suggestions that involve the room the current player is in, should be permitted.
4. The program then repeats steps 2–4 for the next player, unless a player has won or all players have been eliminated from making accusations.

3 Tasks

3.1 Team Work

For this assignment, you have two work in teams of two. You must register your intent to do this using the team signup system.

NOTE: Working in pairs is a good preparation for the group project in which you will work in even larger teams.

3.2 Design

Before you start implementing, you should formulate a design for your program. Use *CRC cards* and a *UML class diagram* to create your design.Validate your design by playing through use case scenarios and keep tweaking it until you are satisfied that it will support an implementation.

3.3 Implementation

Once you are happy with your design, you should begin the implementation which should follow the design you created before. If, during the implementation, you discover that the design requires revision, update the design documentation to reflect the lessons learned. Your code should feature Javadoc comments, adhere to good coding style, and include test cases (see section "Marking Guide").

3.4 Report

Finally, you should write a report that states any significant assumptions you may have had to make, and elaborates on the design.

4 Submission

The following artefacts need to be *submitted electronically*:

1. **Program code, including source code, and Javadoc files.** (filename = `code.jar`)
2. **CRC Cards**, covering the main classes in the system. (filename = `crc-cards.*`)
3. **UML Class Diagram**, featuring at least the main classes with their key attributes/operations and their relationships. (filename = `class-diagram.*`)
4. **Report**, written in your own words (500 word limit) stating any assumptions you have made and providing a rationale for the design. (filename = `report.pdf`, needs to be submitted to its dedicated assignment on the submission system)

Your submitted code must be packaged into a `code.jar` file, including the source code (see for instance this "Exporting to Jar" guide) and including Javadoc documentation. Note that for the Javadoc documentation5 to make sense, your source code should include appropriate Javadoc comments. The code should furthermore provide a suite of JUnit test cases. You must not use any third-party libraries; the code must compile in a standard, plain Java environment.

The other submission artefacts must adhere to the naming scheme detailed above and must be provided as PDF or PNG files; other formats will not be accepted.

NOTE: The written report must be your own work. That means you cannot submit a report which is identical to that of your team partner.

Furthermore, for all submitted work, we cannot acknowledge material copied from other sources, such as the internet or textbooks. If you include any material that you have not authored yourself then you need to state this clearly with a disclaimer and the material will then be discounted for marking purposes. This may or may not impact on your mark, depending on the nature of the material used. If you fail to include such a disclaimer for copied material, unintentionally or not, you will be subject to a plagiarism investigation which may result in the loss of all marks.

We will be using automated systems to check for plagiarism and with your submission you consent to us using these systems.

5 Marking Guide

5.1 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 spelling mistakes and other presentation errors.
- **Assumptions Made (4 marks)**. Marks will be awarded for clear justifications of assumptions made to fill in missing or ambiguous game specification information.
- **Design Discussion (7 marks)**. Marks will be awarded for how well the individual report describes and justifies design decisions made.

5.2 Design [30 marks]

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

- **CRC Cards (10 marks)**. Marks will be awarded for the correct use of the notation and a well-designed distribution of responsibilities.
- **Class Diagram (20 marks)**. Marks will be awarded for adequate use of the following: *inheritance*, *associations*, *multiplicities*, *classes*, *operations*, and *attributes*. Marks will also be awarded for adequate use of object-oriented principles and elegant designs.

5.3 Code [35 marks]

For the submitted code, marks will be awarded on a *team basis* as follows:

- **Correctness (10 marks)**. Marks will be awarded for programs which correctly implement the given specification of Cluedo.
- **Robustness (5 marks)**. Marks will be awarded for programs which execute without crashing and continue to operate correctly even when faced with exceptional situations. This includes appropriate handling of invalid input (e.g. text entered when a number is expected).
- **User Experience (5 marks)**. Marks will be awarded for the text-based interface which includes ease of use and clarity of presentation.
- **Coding Style (5 marks)**. Marks will be awarded for overall coding style which includes: appropriate naming of classes, fields, methods, and variables; good division of work into methods, in order to avoid long and complex chunks of code; consistent formatting (e.g., indentation, etc).
- **Documentation (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 providing a high-level view on the purpose of code segments.
- **Testing (5 marks)**. Marks will be awarded for the inclusion of sensible JUnit tests. These should cover the basic functionality of the program and it is expected that *at least 20 test cases* would be included.

A SUBSTANTIAL AMOUNT OF MARKS WILL BE DEDUCTED FOR NOT COMPLYING TO THE RESTRICTION TO PURELY TEXT-BASED IMPLEMENTATIONS.