Victoria University of Wellington
School of Engineering and Computer Science

SWEN225: Software Design, Assignment 1

Due: 26 July, before midnight

1 Outline

You are to implement a prototype of "Murder Madness" (specification follows below) on a desktop computer.

You will design a solution using CRC cards and UML, and implement a solution using Java. Feel free to optionally 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

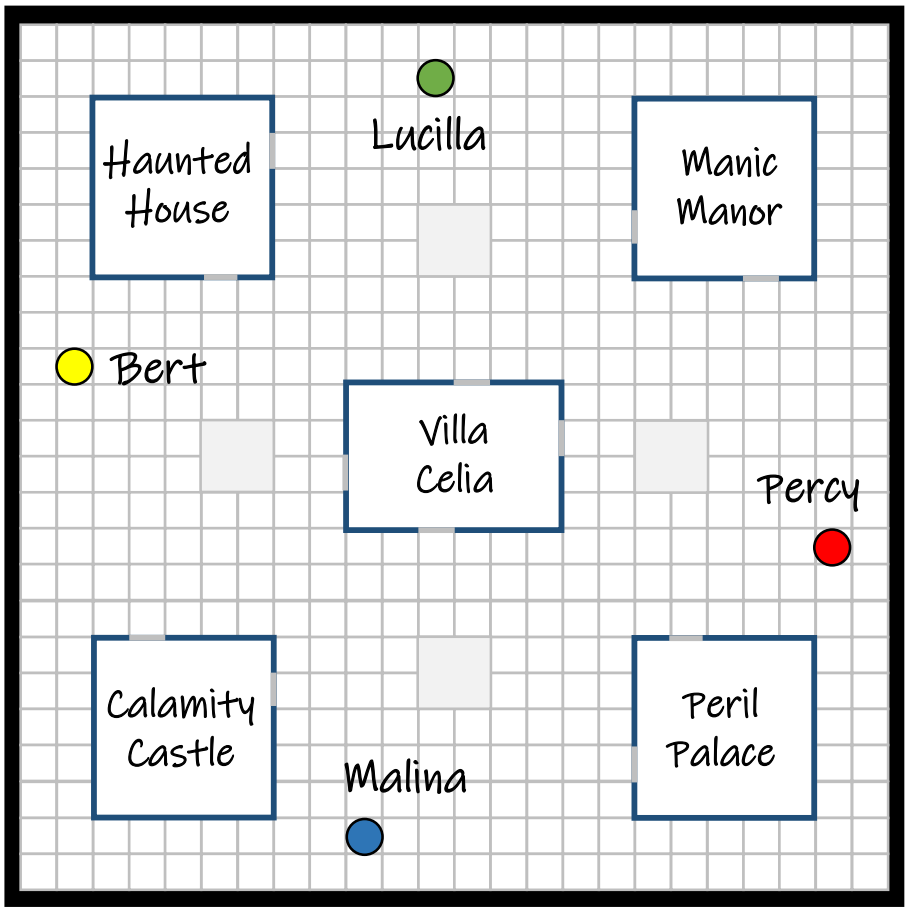
True to life, the following specification may be incomplete or ambiguous in parts and in such cases you will have to make reasonable assumptions which you should then explicitly state in the assignment reflection. You must not publicly ask for clarifications on rules, etc., as identifying and resolving open questions is part of the assignment.

2.1 Objective

The game is a detective challenge played by three to four players who move around a board comprising five estates. The game starts after a murder has happened in one of the estates. The aim is to deduce the murder circumstances, i.e., who the murderer was, what weapon they used and in which estate they committed the murder.

2.2 Board

The "Murder Madness" board consists of five estates located on a 24x24 square area.



The four grey areas are inaccessible to players and player cannot go through estate walls or use a square occupied by another character either. Estates do not have internal structure; they count as a single square.

There are four starting squares, one per character as shown above.

2.3 Characters

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

- Lucilla
- Bert
- Maline
- Percy

Each player assumes the role of one of these characters. If there are less then four players, one character will not be controlled by a player.

2.4 Weapons

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

- Broom
- Scissors
- Knife
- Shovel
- iPad

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

2.5 Rules

Every character, weapon and estate is represented by a card in the game. Before the game starts, one character, one weapon, and one estate card are selected at random. This selection represents the murder circumstances, i.e., the "solution" that players need to figure out during game play.

The remaining weapon, estate and character cards are then combined and distributed at random to players. Some players may end up with more cards than others but only at most one more.

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

When a guess is made, each player – using the order Lucilla, Bert, Malina, Percy, but starting one player after the player making the guess, and cycling around – attempts to refute the guess. A guess is refuted by producing a card that matches one of the suggested murder circumstances (such a card cannot be part of the solution and hence refutes the guess). A refutation card is only revealed to the player that made the guess and if a refutation can be made, it must be made.

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 a *solve attempt* later on (by any player).

A *solve attempt* comprises a character, a weapon, and an estate. If the solve attempt made by a player exactly matches the actual murder circumstances, the player wins, otherwise the player is excluded from making further guesses or solve attempts. This means the player cannot win the game anymore but will continue to refute guesses by others.

2.6 User Interface

Implement an object-oriented program for playing the "Murder Madness" 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`. You may, of course, use the standard output to print a text character-based 2D presentation of the game board using text, location of characters, etc.

1. The program begins by asking how many players wish to participate.
2. One player is selected to start at random.
3. At the start of each turn, the program rolls two (virtual) 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.
4. Once a player has moved, they are presented with the option of making a guess or a solve attempt. All rules of the games must be enforced at all times, e.g., only guesses that involve the estate the current player is in, should be permitted.
5. The program then repeats steps 3–5 for the next player, unless a player has won or all players have been eliminated due to incorrect solve attempts.

Assume the game is being played on a tablet and players hand the tablet back and forth so that only one player at a time can see information meant only for them, e.g., when refutations are made. Your game must print instructions which player is to receive the tablet at each point in time and must not reveal secret information to players who are not privy to that information.

3 Tasks

3.1 Team Work

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

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

Note that there is no expectation that all teams will entirely cover the full specification of the game. We expect simplifications such as not treating each estate as one square but assuming squares exist within estates, pieces being allowed to cross walls, etc. You need to state such simplifications in your reflections, explaining why they have been made. They will cause some mark deductions but we will ensure by scaling grades at the end of the course that no one will be unduly penalised by not having covered the full game specification.

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 adhere to good coding style (see section "Marking Guide").

3.4 Reflection

Finally, you should write an individual reflection that

- makes any clarifications regarding specification ambiguities, specification simplifications made, and user interface design choices.
- recounts how the CRC cards were used to ensure the design is complete and adequate.
- explains how the game logic is implemented by making references to the class diagram.
- documents which (code) contributions you have made and briefly describes any challenges you encountered.

We recommend that every team member gets involved in all phases of the development in order to be able to address all aspects of the reflection. Keep your descriptions concise. A bullet-style phrasing of short sentences is fine and recommended.

We encourage you to be critical of the group work, i.e., remark if any of the above mentioned four aspects were not handled to your satisfaction, and if so, why. Any such remarks will *never* have a negative impact on the group mark. They can, however, give you individual marks for recognising any room of improvement.

4 Submission

The following artefacts need to be *submitted electronically*:

1. **Program code, including source code.** (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. **Reflection**, written in your own words (1000 word limit), as specified above.
   (filename = `reflection.pdf`, 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). You must not use any third-party libraries; the code must compile in a standard, plain Java environment and must run on an ECS machine (JDK build 11.0.10+9), we cannot account for any specifics your local installation may have.

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 reflection must be your own individual work that you produce without collaborating with your team partners. That means you cannot submit a reflection which shares reflection content with that of any of your team partners.

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 and an entry into the academic misconduct register. Note that a person sharing material will be implicated in a plagiarism investigation as well and that it is not allowed to upload VUW assignment descriptions or solutions to VUW assignments to a public repository.

We will be using automated systems such as Turnitin and MOSS to check for plagiarism and with your submission you consent to the use of these systems.

5 Marking Guide

5.1 Individual Reflection [82]

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

- **Grammar and Spelling (5)**. Reflection submissions should be free from spelling and grammar mistakes.
- **Clarifications (12)**. Marks will be awarded for clear justifications of assumptions made to fill in missing or ambiguous game specification information and/or explanations why particular simplifications have been made to the specification and/or elaborations on user interface choices made.
- **CRC Cards Discussion (20)**. Marks will be awarded for how well the individual reflection describes the use of CRC cards.
- **Game Logic (20)**. Marks will be awarded for explaining how the game logic is distributed across the classes by making references to the class diagram.
- **Contributions (25)**. Marks will be awarded for describing (design and code) contributions, any respective challenges encountered, and suggestions for improvement.

5.2 Design [12]

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

- **CRC Cards (5)**. Marks will be awarded for the correct use of the notation and a well-designed distribution of responsibilities.
- **Class Diagram (7)**. 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 [6]

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

- **Correctness (3)**. Marks will be awarded for programs which correctly implement the given specification of "Murder Madness".
- **Robustness (1)**. 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).
- **Coding Style (1)**. 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 (1)**. Marks will be awarded for good use of comments.

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