Assignment 2: Develop

Learning Outcomes

This assignment is intended to develop and assess the following unit learning outcomes:

✓ **LO1**. Iteratively apply object-oriented design principles to design small to medium-size software systems, using standard software engineering notations, namely UML class diagrams and UML interaction diagrams;

✓ **LO3**. Apply object-oriented programming constructs, such as abstraction, information hiding, inheritance, and polymorphism, to implement object-oriented designs using a programming language (namely, Java);

✓ **LO5**. Apply principles of software engineering practice to create object-oriented systems with peers using tools including integrated development environments (IDEs), UML drawing tools, and version control systems.

To demonstrate your ability, you will be expected to:

- implement the features of the system that you designed in Assignment 1
- use an integrated development environment to do so
- update your UML class diagrams and interaction diagrams as required to ensure that they match your implementation
- justify any design changes in an updated Design Rationale. Make it clear by having a new section that is called "Changes in Assignment 2".
- use git to manage your team’s codes, files and documents

The marking scheme for this assignment reflects these expectations.

Introduction

This assignment follows on from Assignment 1. In that assignment, you designed some extensions to the functionality of a roguelike game. In Assignment 2, you will implement those extensions.

IMPORTANT: You may change your design if you find that you need to. It is normal for the design to evolve alongside the codebase as the developers' understanding of the requirements improves. If your design changes, you should document the changes. This includes your Design Rationale as well as your diagrams.

In Assignment 3, you will be given further requirements to design and implement.

What to do?

Project requirements

In this assignment, you must implement all the functional requirements stated in the Assignment 1 specification.

Heavy penalty will apply for editing (add, modify, delete) any classes in the game engine as it breaks the purpose of the whole learning.

Design is important

You need to submit **an interaction diagram for each requirement** (in total, you need to submit 5 interaction diagrams) in addition to other documents that you submitted in assignment 1 (along with the implementation).

One of the primary aims of this unit is for you to learn the fundamentals of object-oriented design. In order to get a high mark in this assignment, it will not be sufficient for your code to “work”. It must also adhere to the design principles covered in workshops, and in the required readings on Moodle.

To make this completely clear: **code that functions but is not well-designed according to the principles you have seen in workshops and readings, will not be enough to get you a good mark**.

Updating your design

As you implement your design, you may find that it has problems. You might also think of a better approach to some of the problems you’ve faced — your understanding of the problem will improve as you progress. **If you want to update your design, you may do so**; just make sure that you update your design documents so that they match the code. It is also a good idea to write a brief explanation of your changes and the reasons behind them, to help your marker understand the thinking behind your code.

Coding and commenting standards

Please refer to the Implementation Expectations (Assignment 2) section of each requirement in Assignment 1 Specification.

You must adhere to the Google Java coding standards that were posted on Moodle earlier in the semester.

Google Java coding standards: https://google.github.io/styleguide/javaguide.html#s5-naming

Write Javadoc comments for *at least* all public methods and attributes in your classes.

You will be marked on your adherence to the standards, Javadoc, and general commenting guidelines that were posted to EdStem earlier in the semester.

⚠ To ensure that your work adheres to good coding practices in this unit, we encourage you to minimise the use of `instanceof` and/or downcasting, as they are considered code smells. It's important to note that if there are any instances where you need to use them, please provide appropriate justifications in code comments or design rationale. We believe learning how to properly utilise polymorphism is crucial in addressing this code smell, and we are committed to teaching you how to do so effectively.

Submission Instructions

Within this class, you are welcome to use foundation models (ChatGPT, GPT, DALL-E, Stable Diffusion, Midjourney, GitHub Copilot, and anything after) in a totally unrestricted fashion, for any purpose, at no penalty. However, you should note that all large language models still have a tendency to make up incorrect facts and fake citations, code generation models have a tendency to produce inaccurate outputs, and image generation models can occasionally come up with highly offensive products. You will be responsible for any inaccurate, biased, offensive, or otherwise unethical content you submit regardless of whether it originally comes from you or a foundation model. If you use a foundation model, its contribution must be acknowledged in the submission (in the form of a separate file titled FoundationModelsUsed.txt); you will be penalised for using a foundation model without acknowledgement.
Having said all these disclaimers, the university's policy on plagiarism still applies to any uncited or improperly cited use of work by other human beings, or submission of work by other human beings as your own.
Moreover, missing contribution logs/GIT commits and/or failing any handover interview will be treated as a potential breach of academic integrity which will be further investigated.

We will mark your assignment based on the latest commit in the `main` branch in your GitLab repository by the due date, **not** in the `master` branch.

You are expected to produce the following artefacts in Assignment 2:

- Implementation (the code) of the system designed in Assignment 1.
- Contribution Log (mandatory, useful for markers)
- Revised UML design documents (based on feedback received on Assignment 1 or if you consider changes were needed)

Additionally, you will be asked to complete the following forms:

- Peer Assessment Form (mandatory, useful for markers and for you to reflect on your group dynamics)
- Feedback Request Form (to be completed in order to upload your files to Moodle, see instructions here).

You can **update** the artefacts previously marked in Assignment 1 since the **code must align with the UML diagrams**.

- Class diagrams: distinguish between existing classes and new classes with different colours.
- Interaction diagrams: sequence diagrams or communication diagrams.
- Design rationale: why this approach is good? what's the alternative?

You need to submit **an interaction diagram for each requirement** (in total, you need to submit 5 interaction diagrams) in addition to other documents that you submitted in assignment 1 (along with the implementation).

We will mark your Assignment 2 on the state of the `main` branch of your Monash GitLab repository by the due date at your local time (see above). If you've done any work in other branches, make sure you merge it into main before the due time.

The FIT2099 Assignment Rules apply to this assignment. Please read this document and make sure you comply, especially as regards the Work Breakdown Agreement.

You do not need to create a new copy of your work for Assignment 2. Git lets us easily roll back to any previous version of your repository, so keeping a separate "Assignment 1" folder only creates unnecessary clutter. If you like, you can add a tag to your final Assignment 1 commit before starting on Assignment 2, so you can find that version easily.

You may update your design if you find that you need to. We expect consistency between your design documents and implementation. Regarding the contributions document, please use the prepared tab to separate the contribution from the previous assignment. We will take your contribution logs into account when marking.

It is highly recommended that you explicitly Indicate in the **design rationale** what changed in the design between Assignment 1 and Assignment 2. Explaining the changes would help you organise your ideas to explain how the design changed during your handover interview. A quick summary of changes with some explanation regarding why you applied such changes would be really valuable.

Contribution Log

We require you to have a document to know how you plan to divide the work between team members and simultaneously capture individual contributions. We have prepared a template you can copy to your university's Google Drive and share with your team. Please follow the detailed instructions **here.** If you have done it in the previous assignment, great! You don't need to copy & replace that link again. You just need to fill your tasks in the `Assignment 2` tab, inside that template, and work.

⚠ Since your markers need to access this information, make sure that you create a share link and put it in the README of your repository. Failure to do so will result in 0 mark.

In this matter, everyone must contribute a **FAIR amount of code AND documentation**. It means you cannot work only on the code or only on documentation. In other words, your team **MUST NOT** split the work by artefacts: a person working on a "class diagram", another on a diagram alone, and another working on design rationale alone. We will give a heavy penalty to your team as a whole.

You will meet up with your team ideally once a week for about 1 hour for team discussion and another 1 or 2 hours for integration and resolving conflicts (if any).

Peer Assessment Form

You should also complete the peer assessment form, which you can find here: https://forms.gle/mTK8WM6LZ345y8aQ7, to help us understand and evaluate your (and other team members') contributions to the assignment better.

About Late Submissions

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy, late submissions will be penalised at 10% per day late (including weekends). Special considerations are granted individually.

It is all team members' responsibility to ensure that the correct versions of the documentation and code are present in the repository by the due date and time.

Marking Rubric

Marking Protocol & Rubric

For this assignment, you have been asked to prepare the implementation of the design you submitted for Assignment 1. You are free to change your design, of course. Here are some important instructions for completing the submission.

- You can update your design documents directly on GitLab.
- Make sure the implemented code is in the main branch on GitLab.
- Make sure you haven't modified the engine.
- Make sure that the link to your contribution log is in the README file in your repository.

Before the final deadline, download a copy of your repository and upload it to Moodle (each team member should do this). If you forget to do this it is ok, don't panic, just upload it as soon as possible.

When using git:

- All team members **must** commit their work (not relying on one member to commit everything)
- Add commit comments that are effective in describing the changes (not just the GitLab defaults)

You should also complete the peer assessment form, which you can find here: https://forms.gle/mTK8WM6LZ345y8aQ7, to help us understand and evaluate your (and other team members') contributions to the assignment better.

Rubric (100 marks)

Each requirement is worth 20 marks (REQ1-5: 20 marks each, and scaled depending on your group size). It can be broken down as follows:

Please note that since REQ5 is an HD requirement, it is team's responsibility to ensure that everyone can contribute to this task. If there's clear evidence that an individual doesn't contribute to this requirement, the mark won't be awarded to that student. For example, Team Member 3 only worked on REQ 4 , and didn't join any meeting, discussions, or work on REQ5 (evidence is collective from peer review, contribution logs, and interviews). Team member 3 will get 0 in REQ5 but will be marked normally in REQ4. This is subject to the markers' decision. Yet, this may not apply if there's sufficient balance of workload in the team.

Features Completeness (3 marks)

3 marks - Meeting all expectations as per the requirement. (e.g., all important classes and relationships are included in the diagram)

2 marks - Some classes or relationships are missing, but the requirement can still be implemented (e.g. Behaviour is not included in the UML class diagram even though it is a part of the requirement)

1 mark - Most of the classes or relationships are missing

0 marks - The UML class diagram or the implementation is missing

Implementation quality: design principles (8 marks)

8 marks - The implementation follows good design principles (if some principles are violated, the trade-off is explained in the rationale, e.g. using singleton to implement a feature), the design is easy to extend

7 marks - The implementation follows good design principles, and the design is easy to extend. However, trade-offs are not mentioned or explained in the rationale

6 marks - The implementation involves a minor violation of design principles (could be easily fixed), (e.g. some attributes are not set to private without any justification)

5 marks - The implementation involves a minor violation of design principles in multiple places (could still be easily fixed), (e.g. some attributes are not set to private without any justification)

4 marks - The implementation involves a non-severe violation of design principles that could be implemented in a better way (not a trade-off), (e.g. some code repetitions are found in the implementation)

3 marks - The implementation involves a non-severe violation of design principles that could be implemented in a better way in multiple places (not a trade-off), e.g. some code repetitions are found in the implementation (DRY)

2 marks - the implementation involves a severe violation of design principles that could be implemented in a better way, e.g. violating the SOLID principles

1 mark - The design/implementation can be considered hacky. For example, the implementation uses downcasting and instanceof in various cases. It can also be the case that abstraction is not used, making the design difficult to extend.

0 marks - The UML class diagram or the implementation is missing

Integration with the existing system (4 marks)

4 marks - The implementation effectively uses the engine classes (e.g. the submitted work demonstrates that the students understand the difference between actions and behaviours)

3 marks - The implementation has a minor error in regards to the engine usage, e.g. using System.out.println to print out messages to the console instead of using the provided Display class - using instanceof instead of capabilities is another example

2 marks - The implementation includes custom methods for functionality that could be implemented with the methods accessible through the engine (e.g. directly using x & y to get the 1-block away surrounding location instead of using the Exit class with location.getExits())

1 mark - The implementation includes custom classes for functionality that could be implemented with the engine class, e.g. creating a custom ground class instead of using the ground class given in the engine package

0 marks - The engine has been modified in a minor or significant way, OR the UML class diagram or the implementation is missing

Alignment with design (3 marks) - Improving from previous A1 feedback incl. Design rationale and UML diagrams (incl. Class and sequence)

3 marks - The design rationale and the UML diagrams (including class and sequence diagram) perfectly match the implementation, AND The design rationale should contain a brief description of what has been done and why it has been done that way (NOT in terms of game design, e.g. I designed it that way to make the game easier to play). The explanation relates to principles taught in the unit, e.g. DRY, SOLID, etc. The rationale also includes some discussion of the pros and cons of the design, why the current design is chosen, and how it can be easily extended (e.g. my design achieves OCP because if a new character is added in the future)

2.5 marks - The design rationale and the UML class diagram perfectly match the implementation, OR the rationale contains a brief description of what has been done and why it has been done that way (NOT in terms of game design, e.g. I designed it that way to make the game easier to play). The explanation relates to principles taught in the unit, e.g. DRY, SOLID, etc. The rationale also includes some discussion of the pros and cons of the design, why the current design was chosen, but it does not have any examples of how it can be extended in the future

2 marks - The design documents have some mismatches with the implementation, OR the rationale contains a brief description of what has been done and why it has been done that way (NOT in terms of game design, e.g. I designed it that way to make the game easier to play). The explanation relates to principles taught in the unit, e.g. DRY, SOLID, etc. However, it does not have any discussion on the pros and cons of the current design & no examples of how it can be extended in the future

1.5 marks - The design documents have some mismatches with the implementation OR the rationale describes what has been done with some mentions of principles (e.g. my design follows SRP) with insufficient explanation

1 mark - Most of the design documents do not match with the implementation, OR The rationale describes what has been done without in-depth explanation (class A has an association with class B because A will have an attribute of type B)

0.5 marks - Most of the design documents do not match with the implementation, OR the rationale describes what has been done with justification in terms of game design, or the rationale only mentions principles without any explanation

0 marks - The design rationale or the UML class diagram is missing for the requirement

UML syntax (1 mark) - sequence diagram and revised class diagrams

1 mark - The presentation and syntax of the UML diagram are perfect (no missing multiplicities, correct arrowhead for all relationships, correct relationship, e.g. realisation for classes implementing interfaces, generalisation for classes inheriting other classes, generalisation for interfaces inheriting other interfaces, etc.). The design is easy to understand in terms of syntax.

0.5 marks - The presentation and syntax of the UML diagrams are almost perfect, but there are minor syntax errors (e.g. perhaps using the wrong arrowheads, neglecting to include multiplicities, messing up the layout, or the UML syntax is incorrect in some parts.). Still, with a little work, the design can be understood by the TA in terms of UML syntax.

0 marks - the diagram for the requirement is missing or does not resemble a UML diagram as required.

Style & Javadoc (1 mark)

1 mark - The code is properly documented with Javadoc, including class-level and method-level documentation. The Google Java Style guide is followed properly (e.g. package names are written in lowercase, attributes and variables names are written in lowerCamelCase, class names are written in UpperCamelCase, etc.)

0.5 marks - Some classes and methods are missing Javadoc documentation, OR some classes do not follow the Google Java style guide

0 marks - Most classes and methods are missing Javadoc documentation, OR The Google Java style guide is not followed, OR the UML class diagram or the implementation is missing

Individual Contribution & Handover Interview (100%)

i This individual contribution is used to weigh the marks awarded at a group level (see above) for cases where individual students did not contribute to the group task fairly (see details under Individual Contribution, below) or did not show evidence that they understand their own work during the handover interview (see details under Handover Interview, below). If you contributed to the group task and demonstrate that you understand your own work you should get (100%) the full marks awarded at a group level as per the rubric above.

Individual Contribution (80%)

80% - The student did sufficient contributions to the team, i.e., multiple **commits** (+5) with helpful commit comments (note: default comments from the web UI don't count.), ideally, one commit per week previous to the submission deadline; and multiple **contribution log entries** (more than 3 entries) with meaningful explanations that can be understandable by a new team member (e.g. the TA). The student shows a reasonable quality of contributions, such as a fair understanding of the OOP knowledge in programming or designing UML diagrams; If applicable, the student contributes to BOTH design and code. There are positive reviews or zero complaints from other team members (via **peer assessment**);

50% - Peer assessment, contributions logs, and git logs show there are insufficient contributions from an individual; the student received a negative review from other team members which usually happened due to last-minute work, difficult communication outside of labs, or any other kind of difficulties (evidence are collective from peer assessment, emails, contribution logs, and git logs). The student committed to git, with at least one commit and comment OR a single commit found without any comment or various commits that are not meaningful and that do not explain the purpose. Some contribution log entries (at most 2 entries) with meaningful explanations that can be understandable by a new team member (e.g. the TA).

0% - There is no evidence of contribution to the team from Git logs or contribution logs. Submitting files as attachments to the release notes would be marked as zero. If applicable, the student only writes comments (inline or Javadoc) without any appropriate programming contribution. Clear evidence of academic misconduct of an individual would be marked as zero.

Handover Interview (20%)

In the handover interview, students will: 1) run their application and demonstrate each implemented feature to their TA; 2) explain the code and UML diagrams for each feature; and 3) answer any questions from the TA regarding the work they completed.

20% If the student can answer all questions during the handover interview satisfactorily. The responses demonstrate that the student understands the various parts of the assignment both parts created by the student and those created by other team members.

10% If all the questions but one is responded to adequately and sensibly. The responses still demonstrate some knowledge about the student's own work but may miss understanding about parts of the work created by other team members.

0% If two or more questions are not responded to adequately and sensibly. The remaining question(s) is/are partly responded to, but it is unclear whether the student understands their own work.

IMPORTANT: Failing to have meaningful commits (i.e. showing that the task was progressively completed) and/or failing the handover interview would automatically flag this as a potential case of plagiarism, it will be further investigated using a similarity check software, and **zero marks would be awarded.**

Contribution

We will assume all team members equally contributed to the assignment (i.e. 50-50% for a team of two or 33.33% each, for a team of three).

IMPORTANT: Any inquiry (e.g. potential conflict within a team) should be submitted via the emails below (not your AdminTA, Lecturer nor CE). Emails sent in other ways will not be processed in time.

FIT2099.Clayton-x@monash.edu if you are based in Clayton
FIT2099.Malaysia-x@monash.edu if you are based in Malaysia