

Assignment 1

Learning Outcomes & Materials

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.
- ✓ **LO2.** Describe the quality of object-oriented software designs, both in terms of meeting user requirements and the effective application of object-oriented design concepts and principles.
- ✓ **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:

- read and understand UML design documentation for an existing Java system
- propose a design for additional functionality for this system
- create UML class diagrams to document your design using a UML drawing tool such as [diagrams.net](#), [UMLet](#) or [plantuml](#) – you are free to choose which one
- write a design rationale evaluating your proposed design and outlining some alternatives
- implement the features of the system that you designed
- use an integrated development environment to do so
- use git to manage your team's files and documents

The marking scheme for this assignment will reflect these expectations

Learning Materials

The base code will be automatically available in your group's repository (Gitlab).

Repeat this mantra: Design, write code, test, fix design, fix code, repeat

- ⚠ **Note:** You **must NOT follow** demo apps' design decisions; they only show how to use the engine, **NOT** how to design a proper system with object-oriented principles.

Introduction

For the rest of the semester, you will be working on a relatively large software project. You will design and implement new functionalities to an existing system that we will provide to you.



IMPORTANT: A document explaining the FIT2099 Assignment Rules is available here: <https://edstem.org/au/courses/14525/lessons/47389/slides/322550>. Please read it and make sure you understand **BEFORE** you begin the project – you are expected to follow what it says and will almost certainly lose marks if you do not.

In this assignment (Assignment 1), you will be working **individually** to implement the first few features of the game. We highly recommend starting by extracting all of the game features. Then, proceed with designing, testing, and repeating the process as needed. This iterative **design-thinking** approach will help ensure a well-developed and refined system.

Getting Started

The initial codebase will be available in a repository that will be created for you on git.infotech.monash.edu. In the meantime, please go through the assignment support modules on Ed Lesson to familiarise yourself with the game engine that you will use during the assignment.



You do not need to submit an interaction diagram (e.g. sequence diagram or communication diagram) in assignment 1. However, you will need to submit these documents for Assignment 2 and Assignment 3. For assignment 1, you may still create these documents if you find them useful for designing the system.

General background

You will be working on a text-based **“rogue-like”** game. Rogue-like games are named after the first such program: a fantasy game named Rogue. They were very popular in the days before home computers were capable of elaborate graphics and still have a small but dedicated following.



If you would like more information about rogue-like games, a good site is <http://www.roguebasin.com/>. The initial codebase will be available in the repository mentioned above. It includes a folder containing design documents for the system.

Assignment 1 Base code

This is your Assignment 1's base code containing `game` and `engine` packages. Please download and put it inside your assignment's repository.



base-main.zip

Static Factory Game

In this assignment, we will develop the “Static Factory” game, inspired by the game [Lethal Company](#). We may use several similar names (characters, items, locations) and concepts. The purpose of using an actual game’s concepts is to help you visualise the required features, such as watching the video gameplay from the actual game to illustrate features that you may find challenging to comprehend. We also believe using actual game references may **bring fun** while working on the assignments.

All linked game contents, videos, and images belong to the respective owners and are subject to copyright. We mainly use the concepts for educational purposes and provide credit to the original creators accordingly. We may also add, alter, and reduce the original content and features to make them more suitable to the game engine, unit outcomes, and assignments’ time frame.

What’s next?

Below are four slides (REQ1-REQ4) describing the requirements you need to complete. Each requirement includes game features that include background stories, entity descriptions (actors, items, or ground), relationships between entities, and actions between them. We suggest extracting these features into a list and discussing it with the TAs to ensure all features are included (*this discussion is not assessed, but highly recommended*). We also provide a section called "testing instructions", which is a manual gameplay testing to measure feature completeness.

Following these requirement slides, we will outline the deliverables necessary for the Assignment 1 assessment.

REQ1: The Intern of the Static factory

Story

In "Static Factory", you are playing as an Intern hired by the Static factory. Your objective is to collect scraps from abandoned moons, which are valuable for the factory's production system. You are sent to one of the abandoned moons, "Polymorphia", for your first mission.

Requirement

The player ("Intern": @) starts the mission inside their spaceship. They begin with 4 hit points (the health attribute).



The spaceship is represented by several squares of floor (_) surrounded by walls (#) in the middle of the game map. The Intern can walk around the floor of the spaceship but cannot step on the walls.

There are scraps scattered across the map that the Intern needs to pick up: large bolts (+) and metal sheets (%). The Intern has an almost unlimited size inventory to pick up a lot of scraps.

In this requirement (REQ1), ~~these scraps can only be picked up and dropped off by the Intern.~~ the only interactions that the intern can perform to these scraps are picking them up and dropping them off.



Clarification (4th April): For the sake of this REQ1's completeness, pick-up and drop-off items are the only interactions that the intern can do to large bolts and metal sheets. In REQ4, the Intern will have more interactions with other objects, such as consuming fruits and attacking creatures.

Once the Intern has picked up these scraps, the goal is to drop them inside their spaceship.

Testing instructions

You may use the following game map or create your own.



```
"...~~~~.....~~~.....",
"...~~~~.....",
"...~~~.....",
".....",
".....#####.....",
".....#___#.....~",
".....#___#.....~~",
".....##_##.....~~~",
".....~~.....~~~",
".....~~~~.....~~~",
".....~~~~~.....~~",
```



REQ2: The moon's flora


Story

Within the surroundings of the ship, the Intern notices a plant growing out of the ground of the moon. Scanning the plant shows that it is named "Inheritree" and can grow and produce fruits, which could be valuable as raw materials for the factory.






Requirement

The "Inheritree" plant starts as a sapling () when it first appears on the moon. At this stage, it can produce small fruits (), which have a 30% chance of dropping within the plant's surroundings (i.e. one "exit" away, next to the tree), which the Intern can pick up or drop off.

After 5 ticks, the plant can grow to its mature stage (). Once matured, the plant can produce larger fruits () that have a 20% chance of dropping within the plant's surroundings (again, one "exit" away). Like the smaller ones, the Intern can pick up the larger fruits or drop them off.

 Edit (04/04): One tree can only drop one fruit on each tick

Testing instructions

Place some plants on the map in `Application` class to test your feature. Play several turns, more than 5, to see if those plants grow from  to . If it runs correctly, you should be able to see many fruits ( and ) on the map, especially at the trees' surroundings. Since you play as an Intern , try to pick up some fruits (both small and large) and drop them off anywhere on a map.

REQ3: The moon's (hostile) fauna

Story

It turns out the Intern has a new objective: survive.

Requirement

The moon has several craters (`u`) that can spawn large and hostile Huntsman spiders (`8`). Each crater can spawn this creature with a 5% chance at every game turn.

This creature will wander around if the Intern is not within its surroundings (i.e. one "exit" away). However, when the Intern enters the creature's surroundings, the creature will attack the Intern with one of its long legs, dealing 1 damage with 25% accuracy. It cannot attack any other creatures that are hostile to the Intern. The creature has 1 hit point.



Don't forget to display the player's hit point (the health attribute of the player) on the screen!



If the player's hit point reaches 0, make sure to display a message on the screen. You can use the message provided in the `FancyMessage` class.

The creature cannot enter the Intern's spaceship.



Think about how your design can support future extensions to this feature (for example, what if we need to add more hostile creatures?). Can you think of other potential extensions for this feature? Can your design easily accommodate those extensions, or would you need to change a lot of the design to implement those extensions? Not sure, please revisit our weekly content.

Testing instructions

Place some craters `u` on the map in `Application` class. Play several turns to check if hostile Huntsman spiders (`8`) emerge from those craters. Next, move/walk the Intern `@` so it stands next to one of them. Try to attack/get attacked by checking the game logs printed every time you accomplish an action. You should see it only attack the Intern, not other creatures.



For the sake of simplicity & completeness to this requirement, Huntsman Spider won't follow the intern if the intern tries to get away from the attack, so for example if the intern moves right, the enemy won't go right unless the `WanderBehaviour` makes it do so

REQ4: Special scraps

Story

After spending several days looking for scraps for the factory, the intern finds some useful scraps that could help them survive on the abandoned moon, such as metal pipes. But, of course, meeting the factory's scraps quota is the highest priority...



Fun fact

Did you know that a [jar of pickles](#) is a "scrap" in the original game?

Requirement

The metal pipe (!) is one type of a special scrap that the Intern can interact with. If it is in the Intern's inventory, they can use it to attack hostile creatures on the moon, dealing 1 damage with 20% accuracy. Similar to other scraps, the metal pipe can be picked up or dropped off.

The Intern can also punch hostile creatures with their bare hands, dealing 1 damage with 5% accuracy.



The player should be given the option to punch the hostile creatures regardless of whether they are carrying a metal pipe. In other words, if the player carries a metal pipe, they are given two options to attack the hostile creatures: with their bare hands or the metal pipe.

The Intern can only attack a hostile creature if the hostile creature is within the Intern's surroundings.

Next, the fruits produced by the "Inheritree" plant can be consumed by the player. The smaller fruits can heal the player by 1 hit points, while the larger ones can heal the player by 2 hit points. Once consumed, the fruit should be removed from the Intern's inventory. These fruits are also special scraps.



What if there are more special scraps? Can you implement these new scraps easily, given your current design? These scraps would be traded (bought/sold) in the future assignment.

Testing instructions

Find a metal pipe, and pick it up. Find any hostile creature and walk to it. Once you are next to this creature, you should see two attack options: attack with a bare hand or with a metal pipe. Try to attack the enemy and check on the game log if you have successfully dealt damage and the creature attacks back. Then, find two types of fruits and consume them. You should see your hit points increase accordingly.

Submission Instructions



Not following any one of the instructions below will result in penalty being applied to your final submission.



You **do not** need to submit an interaction diagram (e.g. sequence diagram or communication diagram) in **assignment 1**. However, you will need to submit these documents for assignment 2 and assignment 3. For assignment 1, you may still create these documents if you find them useful for designing the system although we cover them in later weeks.



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.

Class Diagrams & Documentation Submission

- As mentioned in the Design & Diagrams section, **you MUST create one design (class) diagram per requirement**. In other words, each requirement must be represented in its own separate diagram. Organising them in one large design diagram or combining requirements based on their packages is not allowed. Doing so may reduce the clarity of your design diagram or could be seen as an attempt at reverse engineering. Penalties will be applied if this occurs. Please keep each requirement diagram distinct and avoid any attempts to merge or combine them.
- You **MUST** save your design documentation, including design diagrams, rationale and report (diagram + rationale combined into a single document), in the "docs/design/{assignmentName}" directory. So, for Assignment 1, **you must save all design documentation in the "docs/design/assignment1" directory**.



TAs cannot be expected to look at other directories, except for "docs/design/{assignmentName}", when marking the design diagrams and rationale.

- All design documentation must be saved in PDF format (or PNG/PDF for diagrams, such as REQ1.png, REQ2.png, and so on).
- A Moodle submission is compulsory, and it must be done before the deadline. Please, compress ALL files (code, documentation, and diagrams) as one zip file, and submit the compressed file.



Disclaimer: Although there are multiple ways to design the game, there are also bad designs and good designs - we will mark your submission not against one design but based on the design principles

Design Diagrams

We expect you to produce **four** *UML class diagrams* following **the FIT2099 Assignment Rules**. These Rules are available on EdLesson.

You should not create one class diagram that shows the entire system. The sample diagram in the base code shows the whole system to help you understand how the `game` works with the `engine`. But, in this assignment, you only need to show the following:

- the new parts,
- the parts you expect to change, and
- enough information to let readers know where your new classes fit into the existing system.

As it is likely that the precise names and signatures of methods will be refactored during development, you do not have to put them in these class diagrams. Instead, the overall responsibilities of the classes need to be documented *somewhere*, as you will need this information to begin implementation. This can be done in a separate text document (`.md` markdown format) or spreadsheet, which you should put inside the `docs` directory. We will not assess this document, but we believe it will be handy during the implementation phase.

Design Rationale



A design rationale should be created for **each** requirement. You can submit a single text-based/PDF document.

To help us understand how your system will work, you must also write a *design rationale* to explain your choices. You must demonstrate how your proposed system will work and **why you chose to do it that way**. Here is where you should write down concepts and theories you've learnt so far (e.g., DRY, coupling and cohesion, etc.). You must consider **the pros and cons** of the design to justify your argument.

The design (which includes *all* the diagrams and text that you create) must clearly show the following:

- what classes will exist in your extended system
- what the roles and responsibilities of any new or significantly modified classes are
- how these classes relate to and interact with the existing system
- how the (existing and new) classes will interact to deliver the required functionality

You are not required to create new documentation for components of the existing system that you *do not* plan to change.



IMPORTANT! We will not accept any Word document because it cannot be opened/displayed in Gitlab.

Implementation

We will assess your design implementation (i.e., Java codes). We will assess all of your codes in the

Gitlab repository. Please ensure you push/merge all of your local Java codes to the `main` branch.

You need to ensure that your game can run without breaking. Please follow the "Testing Instructions" from each requirement to satisfy the completeness of its features.



A reminder: you must not modify the game engine, as stated in the assignment rules document.

Your implementation must adhere to the Google Java coding standards.



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

Write Javadoc comments for *at least* all public and protected 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.

Marking Rubric

Assignment 1 rubric

Prerequisite for marking: Design documents (UML diagrams and design rationale) and implementation need to be submitted for the assignment to go through the marking process. Missing any of these will result in 0 marks.

Overview:

Total: 28 points

- Feature implementation completeness (6 points)
- Implementation quality: design principles (8 points)
- Design rationale (4 marks)
- Integration with the existing system (2 points)
- UML syntax and clarity (2 points)
- Alignment and design consistency (2 points)
- Style & Javadoc (1 point)
- Git usage (2 points)
- Format and directory structure (1 point)

Detailed rubric items:

Feature implementation completeness (6 points)

6 marks - The system runs and perfectly meets **all functional expectations** as per the relevant requirement(s) with no runtime errors. All required classes and relationships are implemented, and the program's behaviour aligns perfectly with the specification.

5 marks - The system runs and meets **all functional expectations (with some minor errors or punctual omissions)** as per the relevant requirement(s). All necessary classes and relationships are included, and the program's behaviour largely matches the specification, except for **one or two minor unexpected behaviours**. No runtime errors occur.

4 marks - The system runs and **partially meets all functional expectations, displaying several minor functional errors or omissions** as per the relevant requirement(s). Despite this, the majority of necessary classes and relationships are included, and **only a few minor unexpected behaviours occur**. No runtime errors are present.

3 marks - The system runs and **all functional expectations were addressed to some extent,**

(with one or two major functional errors) as per the relevant requirement(s). Most important classes and relationships are included but **at least one or two major unexpected behaviors** are observed. A major unexpected behaviour is that which affects considerably the completeness of at least one requirement. No runtime errors occur.

2 marks - The system runs but **several functional expectations were not addressed** as per the relevant requirement(s) (e.g. the notion of Behaviour is not included even though it is a part of the requirement). Alternatively, runtime errors occur during the execution of the system but they can be easily fixed.

1 mark - The system runs but **addresses only some functional expectations and shows major omissions** as per the relevant requirement(s) (important classes or relationships are missing). Alternatively, runtime errors occur during the execution of the system without a clear idea of the cause or cannot be easily fixed.

0 marks - The system runs but it **poorly addresses or doesn't address the functional expectations** as per the relevant requirement(s). Alternatively, the system might not run at all.

Implementation quality: design principles (8 points)

This item applies across all functional expectations as per the relevant requirement(s)

8 marks - The implementation of all functional expectations as per the relevant requirement(s) **flawlessly adheres to good design principles and concepts** (e.g., DRY and SOLID principles), making the design easy to extend and maintain. Any punctual violations are **convincingly** justified in the design rationale (e.g., using singleton to implement a feature). All relevant requirements have been addressed.

7 marks - The implementation follows good design principles **nearly perfectly** making the design is easy to extend and maintain (if some punctual principles are violated, the trade-off is **somewhat** justified in the design rationale. All relevant requirements have been addressed.

6 marks - The implementation involves **one or two minor violations of design principles** (could be easily fixed) across all functional expectations as per the relevant requirement(s) (e.g., some attributes are not set to private without any justification). All relevant requirements have been addressed.

5 marks - The implementation involves **minor violations of design principles in multiple places** (could still be easily fixed) across all functional expectations as per the relevant requirement(s). All relevant requirements have been addressed.

4 marks - The implementation involves **one or two non-severe violations of design principles that could be implemented in a better way** (no trade-offs are convincingly provided in the design rationale) across all functional expectations as per the relevant requirement(s) (e.g. some code repetitions are found in the implementation that would require some refactoring to be fixed). All

relevant requirements have been addressed.

3 marks - The implementation involves **non-severe violations of design principles in multiple places** (no trade-offs are convincingly provided in the design rationale). All relevant requirements have been addressed.

2 marks - The implementation involves **one or two severe violations** of design principles that could be implemented in a better way across all functional expectations as per the relevant requirement(s), e.g. violating the basic principles covered so far. Fixing them would require substantial refactoring. Alternatively, not all relevant requirements have been attempted.

1 mark - The design/implementation can be considered **hacky or various instances of procedural programming are found**. For example, the implementation uses downcasting and 'instanceof' in various cases without a convincing justification. Alternatively, it can also be the case that abstraction is not used, making the design **difficult to extend and maintain**. Alternatively, only some relevant requirements have been attempted.

0 marks - The implementation mainly follows a non-OO paradigm; or the UML class diagram(s) or the implementation is missing. Alternatively, most relevant requirements have not been attempted.

Design rationale (4 points)

4 marks - The design rationale includes a description of **what has been done and why**, focusing on the principles and concepts taught in the unit (such as DRY and SOLID principles) and not in terms of game design. The rationale discusses the **advantages and disadvantages** of the design (pros and cons), **the reasons** for choosing the current design, and **ways in which it can be easily extended** (e.g. my design achieves OCP because if a new character is added in the future ...). All relevant requirements have been addressed.

3 marks - The rationale describes **what has been done and why**, based on the principles and concepts taught in the unit. It also includes **some discussion** of the pros and cons of the design and the reasons for the current design choice but **lacks examples of future extensibility**. All relevant requirements have been addressed.

2 marks - The rationale describes **what has been done and why**, aligning with the principles and concepts taught in the unit. However, it **lacks a discussion on the pros and cons of the design and/or examples of future extensibility**. All relevant requirements have been addressed.

1 mark - The rationale **primarily describes what has been done** without a thorough explanation of the decision-making reasons. For instance, it may mention design principles and concepts without providing convincing explanations, discussion of pros and cons, or examples of how the system can be extended. Alternatively, not all relevant requirements have been attempted.

0 marks - The design rationale is either very challenging to read, is missing, or the submitted work omits more than one relevant requirement.

Integration with the existing system (2 points)

This item applies across all relevant requirements.

2 marks - The implementation **effectively uses the engine classes** (e.g. the submitted work demonstrates that the students understand the difference between actions and behaviours). All relevant requirements have been addressed.

1 mark - The implementation **does not use some engine classes as intended** (e.g. behaviours are created for some actions that do not need it, such as actions performed by the player) or includes **custom classes for functionality that could be implemented with the engine class** (e.g. created a custom ground class instead of using the ground class given in the engine package). Most relevant requirements have been addressed.

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

UML syntax and clarity (2 points)

This item applies across all relevant requirements.

2 marks - Relevant (static and/or dynamic) diagrams are **perfect in terms of syntax**, with no missing multiplicities, correct arrowheads for all relationships, and appropriate usage of realisation for classes implementing interfaces, generalisation for classes or interfaces inheriting others, etc. Diagram formatting is **consistent and clear**. All requested diagrams have been submitted.

1 mark - Diagram(s) contain **some minor syntax errors**, such as missing multiplicities for several associations, inappropriate use of generalisation for classes implementing interfaces, realisation for classes extending others, or classes extending multiple classes, etc. Nonetheless, the design can still be understood by the TA with a little effort. Alternatively, there are several **inconsistencies across diagrams**. All necessary diagrams have been submitted.

0 marks - Diagram(s) are significantly hard to understand or show major inconsistencies in formatting and clarity, OR more than one required diagram is missing, OR they do not resemble a UML diagram as required.

Alignment and design consistency (2 points) - incl. Design rationale, code and UML diagrams

2 marks - The design rationale and UML diagrams are in **perfect alignment** with each other and with the code implementation across all functional expectations, as per the relevant requirement(s). All relevant requirements have been addressed or attempted and they are covered in the implementation, UML diagrams and design rationale.

1 mark - There are **some small inconsistencies** (that can be easily fixed) between the design documents and the implementation, but all relevant requirements have been addressed or attempted, and they are covered in the implementation, UML diagrams and design rationale.

0 marks - There are **major inconsistencies** (which would necessitate significant changes for correction) or **numerous smaller discrepancies** distributed throughout the design documents and the implementation. Alternatively, one or more than one required diagram or implemented requirement is missing, or some submitted design documents do not resemble the required UML diagram format.

Style & Javadoc (1 point)

1 mark - The code is properly documented with Javadoc across all functional expectations as per the relevant requirement(s), 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.). All relevant requirements have been addressed or attempted.

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.

Git usage (2 points)

2 marks - At least **15 meaningful commits** have been logged (with meaningful comments) each with a descriptive commit comment (note: default comments from the web UI don't count.), ideally, one commit per week previous to the submission deadline.

1 mark - Between **7 and 14 meaningful commits** have been logged (with meaningful comments) each with a descriptive commit comment (note: default comments from the web UI don't count.). OR there are more than 10 commits but ALL were done on the week of the submission deadline.

0 marks - There are less than 7 commits.



IMPORTANT: if most of the code has been (in practice) uploaded in one go the whole assignment will not be marked and 0 marks will be awarded.

Project formatting and directory structure (1 point)

1 mark - If the directory structure suggested in the specification has been followed. All the documents are easily found where expected.

0 marks - The suggested directory structure was not followed and some files may be hard to find.

Handover Interview (Compulsory)

COMPLETED - The student can answer all (or at least 2) questions during the handover interview satisfactorily. The responses need to demonstrate that the student understands the various parts of the submitted assignment.

NOT COMPLETED 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 for the full assignment.**

Assignment 1 Q&A Session

Direct link: <https://monash.au.panopto.com/Panopto/Pages/Viewer.aspx?id=7a8b43b1-bf64-4d39-bb74-b14800bdb4b3>

Meeting chat:



[GMT20240404-070552_RecordingnewChat.txt](#)