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 (a module above Assignment 1)**. Please read it and make sure you understand **BEFORE** you begin the project for your own benefits (i.e., get good grades and better learning experience).

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.

WINTER SURVIVAL

In this assignment, we will develop a winter survival game, inspired by games such as The Long Dark and Subnautica: Below Zero. 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 that using actual game references can make working on the assignment more enjoyable!

NOTE: 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 for the game engine, unit outcomes, and the 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 them with the TAs to ensure all features are included (*this discussion is not assessed, but highly recommended*).

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

REQ1: The Explorer

In the game that you are developing, the player plays as the “Explorer” (`▫`), a character who is left behind to survive in the wilderness during the cold and dark winter. They start with 100 hitpoints (health).

Throughout the game, the “Explorer” has a hydration level, which starts at 20 and decreases by 1 at each turn, and a warmth level, which starts at 30 and decreases by 1 at each turn. If the hydration level or the warmth level reaches 0, then the “Explorer” becomes unconscious and the game ends.

Don’t forget to display the “Explorer”’s statistics, i.e., their hitpoints, hydration and warmth level at each turn!

At the start of the game, the “Explorer” carries several items, including a bedroll (`=`) and a bottle (`o`).

The “Explorer” can sleep on the bedroll, which results in 6-10 turns being skipped. The number of turns being skipped can be randomly chosen. Throughout their sleep, their hydration and warmth levels will stop decreasing. Note that the bedroll must be dropped on the ground before an option for the “Explorer” to sleep appears on the console.

The “Explorer” can drink from the bottle 5 times. Drinking from the bottle reduces the thirst level by 4 each time.

REQ2: The Beasts of the Woods

There are several animals in the game that the “Explorer” can interact with, including a deer (`d`), a bear (`B`), and a wolf (`e`).

Deers start with 50 hitpoints. They can wander around the game map.

Wolves and bears can also wander around. Wolves start with 100 hitpoints, and they can bite another actor, dealing 50 damage points with a 50% chance of hitting (hit rate). Bears start with 200 hitpoints, and they can claw another actor, dealing 75 damage points with an 80% chance of hitting.

The “Explorer” can attack any animals within their surroundings. You can test this by attacking any animals within the “Explorer”’s surroundings using their bare fist.

REQ3: Flora

There are several trees in the game, including wild apple trees (`T`), hazelnut trees (`A`), and yew berry trees (`Y`).

For now, each tree can drop an item within its surroundings.

> ⚠ Note, however, that you cannot assume all plants will produce something. Some plants may not drop anything.

Wild apple trees can drop apples (`a`) every 3 turns. Apples can be consumed by the "Explorer", healing them by 3 points. Additionally, eating an apple reduces the thirst level by 2.

Hazelnut trees can drop hazelnuts (`n`) every 10 turns. Hazelnuts can be consumed by the "Explorer", increasing their maximum health by 1 point.

Yew berry trees can drop yew berries (`x`) every 5 turns. Yew berries can be consumed by the "Explorer", killing the "Explorer" immediately, ending the game.

REQ4: Taming the Wildlife

In this requirement, we give you the theme of the functionality that you need to implement, and you can decide the details by yourself!

The functionality that you need to implement is **Animal taming**. For example, predators, such as bears and wolves, can be tamed by providing them with food. Once tamed, they will follow the player and fight alongside them (attacking other predators that are hostile to the player).

You can expand this functionality further, e.g., the player may also tame deers. Once tamed, they can collect the dropped fruits and give them to the player.

⚠ Note that the difficulty should be of a similar level as mentioned above, i.e., taming the animals must result in **at least three additional behaviours**, e.g., collecting dropped fruits, fighting alongside the player, etc. **Having no effect after the animal is tamed will be considered not completing the HD requirement.**

⚠ It is also important to apply the SOLID principles in the implementation. The application must then be explained clearly in the design rationale. Implementing the requirement without any application of the SOLID principles will be considered as not completing the HD requirement.

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.

AI & Generative AI tools may be used in GUIDED ways within this assessment / task as per the guidelines provided.
In this task, AI can be used as specified for one or more parts of the assessment task as per the 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 txt, pdf or word file titled FoundationModelsUsed added to your "docs" folder in GIT); 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.

Inappropriate AI use and/or AI use without acknowledgement will be considered a breach of academic integrity.

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, with a **maximum word limit of 1,000 words** per requirement. However, aim for concise and focused explanations (at least ~500 words). You can submit a single text-based/PDF document.

To help us understand how your system will work, you must also write **four** *design rationales* to explain your choices (one for each requirement). 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.

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 your mark being capped at 50 (Pass) maximum.

Directory Structure and File Organisation

To ensure a smooth and fair marking process, please follow the directory structure outlined in the assignment specification. This helps your marker easily locate all required documents without confusion. **If files are placed outside the expected locations, markers are not obliged to search for them. Please note that re-marking requests related to misplaced or missing files in incorrect folders will not be considered.** Organising your submission as specified not only supports your own success but also respects the time and effort of those assessing your work.

Moodle Submission Requirements

Submission via Moodle is **mandatory** for this assignment. Please ensure that you complete the Moodle submission **on time** to avoid any penalties for late submissions. To do this, you will need to **export your project from GitLab as a ZIP file** and upload that ZIP file to the appropriate assignment link in Moodle.

Make sure that the version you submit to Moodle is **identical** to the one in your GitLab repository at the time of submission. The **timestamp recorded in Moodle** will be used as the official reference for submission time, including any late submission considerations.

Timely and accurate submission helps maintain consistency and fairness in assessment for all students. If you have any difficulties, we recommend reaching out well before the deadline.

Overview:

Total: 30 points

- Feature implementation completeness (6 points)
- Implementation quality: design principles (10 points)
- Design rationale (5 marks)
- Alignment and design consistency (3 points)
- Integration with the existing system (2 points)
- UML syntax and clarity (2 points)
- Style & Javadoc (1 point)
- Git usage (1 points)

Detailed rubric items:

Feature Implementation Completeness (6 points)

This criterion applies across all functional expectations, including the HD requirement if attempted.

6 marks – Exceptional Implementation (HD)

- The HD requirement is fully implemented, going **beyond expectations** in terms of complexity or sophistication.
- All other functional expectations are **completely and correctly implemented** as per the specification.
- The program is **free of runtime errors**, and all behaviour aligns perfectly with the documented requirements.
- The implementation reflects **deep understanding** of OO programming in Java, with a structure that is **robust, intuitive, and maintainable**.
- No unnecessary code duplication or incomplete logic is present.

5 marks – Excellent Implementation (HD)

- The HD requirement is **mostly implemented**, with **minor gaps** or simplifications.
- All other functional expectations are **fully implemented** and behave as expected.
- No runtime errors are present, and the system shows **strong structural correctness**.
- Behaviour is consistent across all required features, though some **edge cases** may be simplified or omitted.

Distinction (D_low) [4 marks]

4 marks – Fully Functional Implementation (HD Attempted)

- The system runs without errors and satisfies **all core functional requirements** as specified.
- OR The HD requirement was **attempted**, though not fully realised or may lack some behaviours.
- All required **classes, methods, and relationships** are present.
- The implementation behaves as expected in all standard use cases, with **no significant omissions**.

Credit/Pass (C) [3 marks]

3 marks – Mostly Functional Implementation (HD Not Attempted)

- The system runs successfully and meets **most functional expectations**, with minor errors or omissions.
- All necessary classes and relationships are included.
- Several **minor omissions or logic errors** may be present.
- The HD requirement was **not attempted** or was attempted but it wasn't correctly

implemented.
- One or two **unexpected behaviours** occur but do not severely impact functionality.

Fail (F) [0–2 marks]

2 marks – Incomplete or Flawed Implementation

- The system runs and most necessary classes and relationships exist, though some may be underdeveloped or may **behave incorrectly**.
- One or two **major functional errors** affect expected behaviour.
- Runtime errors may occur but are **easily fixable**.
- The HD requirement was **not attempted** or was attempted but it wasn't correctly implemented.

1 marks – Significantly Incomplete Implementation

- The system runs but several core expectations are **missing or faulty**.
- Runtime errors occur and are **hard to trace or resolve**.
- A key feature may be entirely missing.
- The HD requirement was **not attempted** or was attempted but it wasn't correctly implemented.

0 marks – Non-functional or Absent Implementation

- The system runs but meets **few functional expectations** OR the system does **not run** at all, or
- It runs but **fails to address** any core expectations.
- The HD and most requirements were **not attempted**.
- The **design rationale** is missing or **UMLdiagrams are missing**.

Implementation Quality: Design Principles (10 points)

This criterion applies across all functional expectations, including the HD requirement if attempted.

High Distinction (HD) [8–10 marks]

10 marks – Exceptional Design (HD)

- The HD requirement is implemented and demonstrates **outstanding adherence** to **DRY** and **SOLID** principles.
- The design is **flawless**, with **clear abstraction, modularity, and extensibility**.
- Design decisions reflect **creative, effective, and maintainable solutions**.

9 marks – Excellent Design (HD)

- The HD requirement is fully addressed.
- The rest of the design is **flawless** and demonstrates perfect use of DRY and SOLID.

- Code is **modular, reusable**, and adheres to clean design practices.

8 marks – Strong Design (HD)

- The HD requirement is implemented but includes **minor violations** of design principles.
- The rest of the system follows DRY and SOLID with **no issues**.
- Any minor design concerns are **clearly identified and partially justified**.

Distinction (D) [6–7 marks]

7 marks – Very Good Design (HD Attempted)

- All functional expectations are implemented; the HD requirement was attempted.
- DRY and SOLID principles are **mostly respected**.
- Any violations are **justified** in the rationale.
- Design is **extensible and maintainable**.

6 marks – Good Design (HD Not Attempted)

- All functional expectations are met (excluding HD).
- Design **largely follows** DRY and SOLID, with **minimal violations**.
- Trade-offs are **somewhat justified**.

Credit (C) [5 marks]

5 marks – Satisfactory Design

- All relevant requirements (excluding HD) are implemented.
- One or two **minor violations** (e.g., improper encapsulation) exist.
- Design is **understandable** and **moderately maintainable**.

Pass (P) [4 marks]

4 marks – Adequate Design

- Minor violations of DRY/SOLID principles occur **in multiple places**.
- Design shows **basic understanding** but lacks consistency or rigor.

Fail (F) [0–3 marks]

3 marks – Flawed but Recoverable Design

- Design has **noticeable violations** (e.g., code repetition, weak cohesion).
- Refactoring would be needed to meet good design standards.

2 marks – Weak Design

- Multiple design violations present.

- Trade-offs are **not explained**; code is **not easily maintainable**.

1 mark – Poor Design or Incomplete Implementation

- Some requirements are **not addressed**.
- Basic OO principles are **frequently violated**.

0 marks – Unacceptable Design

- Most requirements not attempted.
- The design is **procedural**, or the **design rationale** is missing or **UMLdiagrams/implementation are missing**.

Design Rationale (5 points)

This criterion evaluates the clarity, depth, and reasoning behind the design choices, with focus on DRY, KISS, and SOLID.

5 marks – Exemplary Rationale (HD Requirement Addressed)

- Clear, structured, in-depth rationale showing strong use of DRY, KISS, and SOLID.
- HD requirement is addressed and **explained in detail**.
- Includes pros/cons, hypothetical examples, maintainability foresight, and **comparison of alternatives**.
- All requirements addressed with strong justification.

4 marks – Comprehensive and Well-Reasoned Rationale

- Strong reasoning for most decisions using DRY, KISS, SOLID.
- Pros/cons and maintainability are clearly discussed.
- Comparison of alternatives included (may be brief).
- All relevant requirements are addressed.

3 marks – Adequate Rationale with Some Depth

- Describes what and why with **some principle references**.
- Examples or trade-offs may be generic or underdeveloped.
- Comparison present but shallow.
- All requirements addressed. HD requirement not included.

2 marks – Basic or Limited Rationale

- Surface-level explanation of decisions.
- Design principles **briefly referenced**.
- Trade-offs and future impact **not well covered**.

1 mark – Minimal or Incomplete Rationale

- Lists what was done without explaining why.
- Mentions principles with **vague or no justification**.
- Omits trade-offs and future considerations.

0 marks – Missing or Incoherent Rationale

- Missing, unclear, or unrelated to principles.
- Requirements and HD component **not addressed**.

Alignment and design consistency (3 points) - incl. Design rationale, code and UML diagrams

3 marks (HD Requirement Addressed)

- 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. To get these marks, **the HD requirement is addressed effectively.**

2 marks

- The design rationale and UML diagrams are in **alignment to a great extent** with some **minor inconsistencies**, across all requirement(s). All relevant requirements have been addressed or attempted and they are covered in the implementation, UML diagrams and design rationale. The HD requirement may have been addressed to some extent.

1 mark

- There are **several 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. The HD requirement was not addressed.

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 the design rationale or implemented requirement is missing, or some submitted design documents do not resemble the required UML diagram format. The HD requirement was not addressed.

Integration with the existing system (2 points)

This item applies across all relevant requirements.

2 marks - Effective use of the engine

- 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. To get these marks, **the HD requirement is addressed effectively.**

1 mark - Ineffective use of the engine.

- 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 design rationale OR the implementation is missing

UML syntax and clarity (2 points)

This item applies across all relevant requirements.

2 mark

- 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. To get these marks, **the HD requirement is addressed effectively.**

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 or the design rationale is missing, OR they do not resemble a UML diagram as required.

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. To get these marks, **the HD requirement is addressed effectively.**

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 (1 point)

1 mark

- 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.

0.5 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 OR commit messages do not correspond to the changes in the code.

IMPORTANT: if most of the code has been (in practice) uploaded in one go the marks for the whole assignment will be capped to 50 (P) marks maximum.

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) or having most commit messages not correspond to the changes in the code, and/or failing the handover interview would automatically flag this as a potential case of plagiarism. This may be further investigated using a similarity check software, and **zero marks would be awarded for the entire assignment.**