Assignment 1: Design

Learning Outcomes

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

✓ **LO1.** Iteratively construct object-oriented designs for small to medium-size software systems, and describe these designs using standard software engineering notations including UML class diagrams (in conceptual and concrete forms), UML interaction diagrams and, if applicable, UML state diagrams;

✓ **LO2.** Evaluate the quality of object-oriented software designs, both in terms of meeting user requirements and in terms of good design principles, using appropriate domain vocabulary to do so;

✓ **LO5**. Use software engineering tools, including UML drawing tools, integrated development environments, and revision control, to create, edit, and manage artifacts created during the development process.

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 and interaction diagrams to document your design, using a UML drawing tool such as UMLet or Visual Paradigm – you are free to choose which one
- write a design rationale evaluating your proposed design and outlining some alternatives
- use git to manage your team's files and documents

The marking scheme for this assignment will reflect these expectations

Learning Materials

Please download this .zip file to familiarise yourself with the game engine, documentation, and more. The base code will be available in your group's repository (Gitlab) in the following weeks. So, please be patient. :)

fit2099-assignment.zip

⚠ 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.

It is intentional that a .zip file above doesn't have a `game` package. We don't want you jumping to the code straightaway. Design first, then code, fix them up, repeat 🔁

Introduction

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

✓ **IMPORTANT**: A document explaining the FIT2099 Assignment Rules has been uploaded to the Assessments section in Moodle. Please read it and make sure you understand it **BEFORE** you begin the project – you are expected to follow what it says and will almost certainly lose marks if you do not.

Getting Started

The initial codebase will be available in a repository that will be created for you on **git.infotech.monash.edu** by Week 5. In the meantime, design documents for the system are available for you under the Assessments section in Moodle. Download these and familiarise yourself with the design.

For each requirement, you will see `Scenario` and `Implementation Expecations` subheadings. Implementation expectations will be used in Assignment 2 later, but the provided information might be useful for you to grasp the feature as a whole.

Aiming to simulate the real-world conditions you may experience in the industry, you **will be allocated to a design and development team**. We won't create groups with students from different lab classes as this makes it less clear who's supposed to mark you.

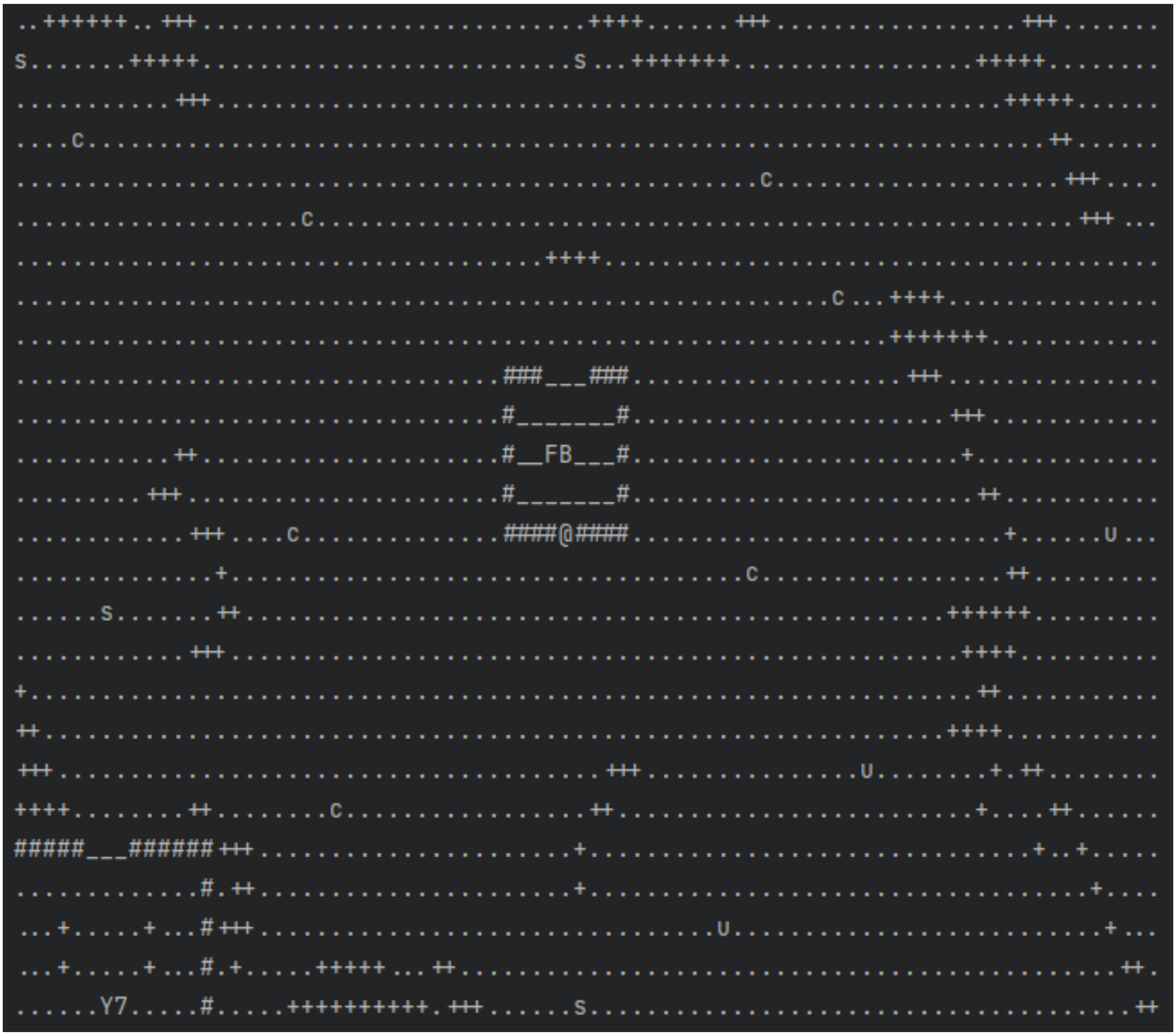
Requirements vs Group size

If you are a group of 3, you must complete all of the provided scenarios. If you are working in pair (2 people), You don't need to complete **REQ6: Monologue**. Despite there might be some dependencies with these two features from other requirements, you still need to include them in your submission.

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.

i 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.



```
..++++++..+++...........................++++......+++.................+++.......
S.......+++++..........................S...+++++++.................+++++........
...........+++.......................................................+++++......
....C...................................................................++......
....................................................C....................+++....
....................C.....................................................+++...
.....................................++++.......................................
.........................................................C...++++...............
.............................................................+++++++............
..................................###___###...................+++...............
..................................#_______#......................+++............
...........++.....................#__FB___#.......................+.............
.........+++......................#_______#........................++...........
............+++....C..............####@####..........................+......U...
..............+....................................C.................++.........
......S.......++.................................................++++++.........
............+++...................................................++++..........
+..................................................................++...........
++...............................................................++++...........
+++......................................+++...............U........+.++........
++++........++........C.................++.........................+....++......
#####___######+++......................+...............................+..+.....
.............#.++......................+...................................+....
...+.....+...#+++................................U..........................+...
...+.....+...#.+.....+++++...++..............................................++.
......Y7.....#.....++++++++++.+++......S......................................++
```

In this assignment, we are inspired by the Super Mario game. 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 game. It also helps you to immerse yourself in the world.

Lastly, we believe that using actual game references can **bring fun** while working on the assignments. All of the 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, reduce the original content and features to make them more suitable to the game engine, unit outcomes, and assignments' time frame.

In this game, you will play as **Mario** (`m`) to explore the magical land of Mushroom Kingdom!

REQ1: Let it grow! ▯

Scenario

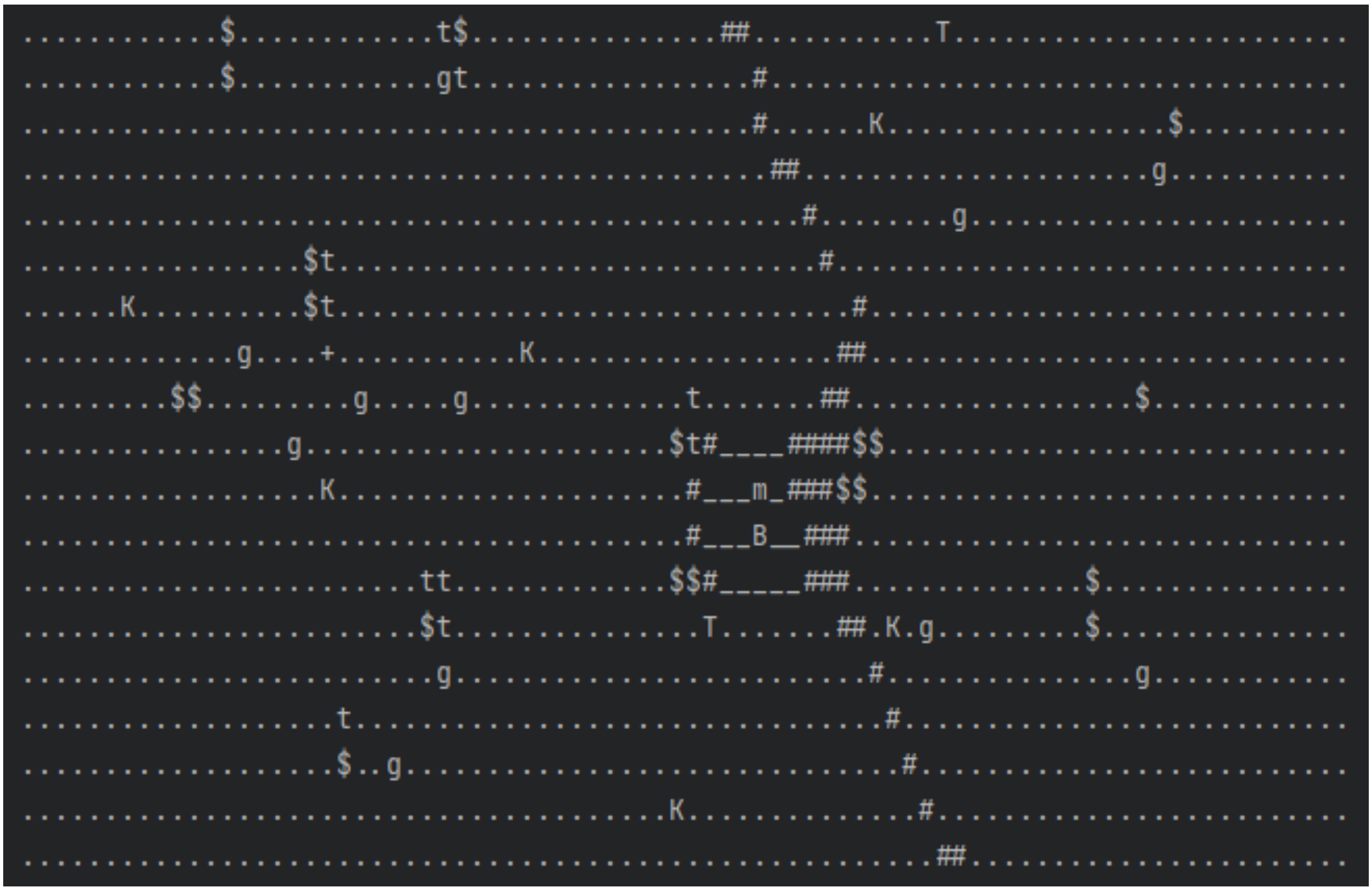
First, we want to enrich the world with some greens (World 1-1). A Tree has three stages/cycles and each stage has a unique spawning ability. It takes 10 turns to grow into the next stage.

- Sprout (`+`)
  - It has a 10% chance to spawn Goomba on its position in every turn. If any actor stands on it, it cannot spawn Goomba.
  - It takes 10 turns to grow into a small tree/Sapling (`t`)
- Sapling (`t`):
  - It has a 10% chance to drop a coin ($20) on its location in every turn.
  - It takes another 10 turns to grow into a tall tree/Mature(`T`)
- Mature (`T`):
  - It has a 15% chance to spawn Koopa in every turn. If an actor stands on it, it cannot spawn Koopa.
  - For every 5 turns, It can grow a new sprout (`+`) in one of the fertile squares randomly. If there is no available fertile square, it will stop spawning sprouts. At the moment, the only fertile ground is Dirt.
  - It has 20% to wither and die (becomes Dirt) in every turn.

✓ The generated coins should stay in the same position even though the Tree grows.

Implementation Expectations (A2)

- Place some sprouts randomly on the map during the instantiation.
- The game must run normally without throwing any errors or exceptions.
- Picking up a coin should increase the Player's wallet balance.
- After running several turns (~40), it should approximately look like this picture -- not too crowded.



```
............$............t$...............##...........T........................
............$............gt.................#...................................
............................................#......K.................$..........
.............................................##.....................g...........
...............................................#........g.......................
.................$t.............................#...............................
......K..........$t...............................#.............................
.............g....+...........K..................##.............................
.........$$.........g.....g.............t.......##.................$............
................g......................$t#____####$$............................
..................K.....................#___m_###$$.............................
........................................#___B__###..............................
........................tt.............$$#_____###..............$...............
........................$t...............T.......##.K.g.........$...............
.........................g.........................#...............g............
...................t................................#...........................
...................$..g..............................#..........................
.......................................K..............#.........................
.......................................................##.......................
```

Legend: `$` coin, `+`sprout, `t` sapling, `T` tree, `g` Goomba, `K` Koopa, `m` Mario, `B` Toad, `#`wall, `.` dirt, `_` floor.

REQ2: Jump Up, Super Star! ▯

Scenario

A Super Mario game will not be complete without a **"jump"** feature. When the actor is standing next to the high ground, the actor can jump onto it. Going up requires a jump, but going down doesn't require it (the actor can walk normally). Each jump has a certain success rate. If the jump is unsuccessful, the actor will fall and get hurt. The success rate and fall damage are determined by its destination ground as listed below:

- Wall: 80% success rate, 20 fall damage
- Tree:
  - Sprout(`+`): 90% success rate, 10 fall damage
  - Sapling(`t`): 80% success rate, 20 fall damage
  - Mature(`T`): 70% success rate, 30 fall damage

✓ We might expand the height and depth of the ground. So, please prepare your design to accommodate this feature.

For instance,

> Mario tries to jump from Dirt to Wall. Unfortunately, the jump fails, Mario **stays** on the current ground (e.g., Dirt), and he receives 20 damage (from the Wall). The next attempt is a success and Mario is standing on the Wall now. If Mario tries to move to a Tree next to his current position, he must jump again. Otherwise, he can walk down to Dirt, Floor, and other grounds freely.

When Mario consumes **Super Mushroom** ▯ (see Magical Items), he can jump freely with a 100% success rate and no fall damage.

How about the enemy? ☠ The enemy cannot jump at all, but it can walk down to the lower ground. For instance, the enemy that is standing on the Tree can walk down to the Dirt freely. But, it cannot jump to other high grounds (any kind of Trees and Wall).

Implementation Expectations (A2)

We provide several expectations to consider your game as complete:

- The player will have a jump action when standing next to high grounds.
- For better game experience, prints the coordinate of the current high ground. For example, `Wall(54, 10)` where 54 and 10 are x and is y coordinates respectively.

- A successful jump will put the player on the target ground and print a message in the console.
- If the player fails to jump up, it stays on the original ground, receives a certain amount of damage and prints a message along with the amount of damage received.
- After a successful jump, the player cannot simply walk to other high grounds (must use jump action).
- The player can walk down to the lower ground that has one level difference.
- If the player tries to jump down (more than one level) and fails, the player will be on the target ground and receive the amount of damage.
- Ensure that the commands list is clean. For example, the player must not see the current ground's jump action in the command list (e.g., if the player stands on the Wall, it must not print "jump to Wall").
- Print appropriate and fun messages in the console for every successful or failed jump attempt.

REQ3: Enemies ☠

Scenario

The game will be incomplete without any enemies. There are two enemies at the current stage: Goomba and Koopa Troopa. Once the enemy is engaged in a fight (the Player attacks the enemy or **the enemy attacks you**), it will follow the Player. The unconscious enemy must be removed from the map. All enemies cannot enter the Floor (`_`).

1. Goomba

> Goombas /ˈɡuːmbə/, known in Japan as Kuribo, are **a fictional mushroom-like species from Nintendo's Mario franchise**. Goomba is a basic enemy. In Japan, Goombas are called "Kuribō", which loosely translates as "chestnut person".[7]

- It starts with 20 HP
- It attacks with a kick that deals 10 damage with 50% hit rate.
- In every turn, it has a 10% chance to be removed from the map (suicide). The main purpose is to clean up the map.

2. Koopa

> Koopa Troopas, commonly shortened to Koopas or Troopas, known in Japan as Nokonoko, are reptilian mini-troopers of the Koopa Troop from the Mario franchise. When defeated, Koopas may flee from or retreat inside their shells, which can usually be used as weapons.

- It starts with 100 HP
- When defeated, it will not be removed from the map. Instead, it will go to a dormant state (`D`) and stay on the ground (cannot attack nor move).
- Mario needs a **Wrench** (80% hit rate and 50 damage), which is the only weapon to destroy Koopa's shell.
- Destroying its shell will drop a Super Mushroom.

Implementation Expectations (A2)

- You should see some Goombas spawn from the sprouts (not all of them)
- Goomba should not fully populate the map because it can self-destruct
- Try to attack Koopa until it is unconscious. It will hide inside its shell, and so the display character should change to `D`. You must not have an attack action to it anymore.

- Koopa that goes to dormant state (i.e., becomes `D`), cannot follow, attack, or wander around.
- Grab a Wrench from Toad (or put it on the map in the beginning), and stand next to hiding Koopa (`D`). Here, you should see an action to hit and destroy the shell. Once executed, it will be removed from the map and you should see a Super Mushroom.

REQ4: Magical Items □

Scenario

1. Super Mushroom □

Anyone that consumes Super Mushroom `^` will experience the following features:

- increase max HP by 50
- the display character evolves to the uppercase letter (e.g., from `m` to `M`).
- it can jump freely with a 100% success rate and no fall damage.

The effect will last until it receives any damage (e.g., hit by the enemy). Once the effect wears off, the display character returns to normal (lowercase), but the maximum HP stays.

2. Power Star □

Power Star `*` cannot stay in the game forever. It will fade away and be removed from the game within 10 turns (regardless it is on the ground or in the actor's inventory). Anyone that consumes a Power Star will be healed by 200 hit points and will become invincible. The invincible effect will remove the fading duration, and it lasts for another 10 turns.

- **Higher Grounds.** The actor does not need to jump to higher level ground (can walk normally). If the actor steps on high ground, it will automatically destroy (convert) ground to Dirt.
- **Convert to coins.** For every destroyed ground, it drops a Coin ($5).
- **Immunity.** All enemy attacks become useless (0 damage).
- **Attacking enemies.** When active, a successful attack will instantly kill enemies.

Implementation Expectations (A2)

- Place Super Mushroom and Power Star on the same ground when you instantiate the Player.
- For the Power Star, you should see the number of remaining turns (e.g., "Mario consumes Power Star - 10 turns remaining") before it is removed from the game.
- Consume Super Mushroom and try to jump to the nearest wall and trees.
- Try to get hit by the enemy, and so the display character should be converted back to original
- Consume the Power Star (if it still exists, otherwise restart the game), and you shouldn't see any jump action anymore when standing near the walls or trees.
- While the effect is active, you should see the text "Mario is INVINCIBLE!" in the console.
- Whenever you walk on these grounds (e.g., wall and tress), they should drop a coin.

- Try to get attacked by the enemy and your HP should not decrease.
- Try to attack the enemy, if successful (not miss), you should kill it immediately (regardless it is Koopa or Goomba).

REQ5: Trading 🪙

Scenario

The coin(`$`) is the currency that player uses to trade with items. A coin has an integer value that determines the actual value of the money. Coins will spawn randomly from the Sapling (`t`). The collected coins can be traded with Toad for the following items:

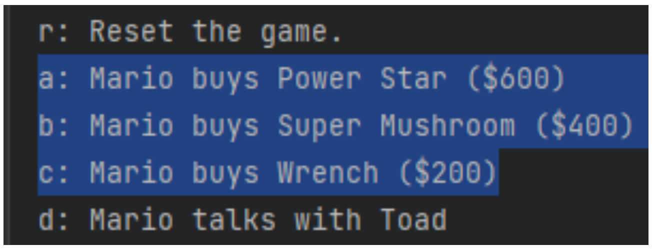
1. Wrench: $200
2. Super Mushroom: $400
3. Power Star: $600

Toad (`B`) is a friendly actor, and he stands in the middle of the map (surrounded by brick Walls).

✓ It is recommended that you have a wallet system that can store the total balance of your money.

Implementation Expectations (A2)

- You should see multiple actions to buy an item



```
r: Reset the game.
a: Mario buys Power Star ($600)
b: Mario buys Super Mushroom ($400)
c: Mario buys Wrench ($200)
d: Mario talks with Toad
```

- When you buy Power Star or Super Mushroom, you cannot drop these items.
- You can consume Super Mushroom anytime, but not for Power Star (which has fading duration).
- If you don't have sufficient money, it must print "You don't have enough coins!" and so you cannot get the item.

REQ6: Monologue 🍄

This requirement is only for group of 3. This feature is optional for group of 2.

Scenario

Toad (`O`) is a friendly actor, and he stands in the middle of the map (surrounded by brick Walls). You will have an action to speak with Toad. Toad will speak one sentence at a time randomly:

1. "You might need a wrench to smash Koopa's hard shells."
2. "You better get back to finding the Power Stars."
3. "The Princess is depending on you! You are our only hope."
4. "Being imprisoned in these walls can drive a fungus crazy :("

Once the player holds a **Wrench**, he won't' say the first sentence. When the Power Star effect is active, the second monologue must not be printed in the console. Other than these two scenarios, he will randomly pick a monologue above.

Implementation Expectations (A2)

- When you interact with Toad, you should see a random sentence printed on the console.
- Get wrench, and try to talk with Toad several times. It must not print the first sentence.
- Consume Power Star, and try to talk with Toad several times. It must not print the second sentence.

REQ7: Reset Game□□

Scenario

Sometimes, the game can become overwhelming and it is hard to walk around (too many Koopas!). Mario has an action to reset the game at any time. If it is executed, it will do the following:

- Trees have a 50% chance to be converted back to Dirt
- All enemies are killed.
- Reset player status (e.g., from Super Mushroom and Power Star)
- Heal player to maximum
- Remove all coins on the ground (Super Mushrooms and Power Stars may stay).

Mario can only do this **once** throughout the entire game.

Implementation Expectations (A2)

- Play the game for several turns until you see all enemies and items are on the map.
- Execute the reset (change the hotkey to `r`) and it should clear up all enemies and coins.
- Some of the trees are converted back to dirt.
- All destroyed walls should stay as dirt.
- The reset command should not be available in the console anymore once it is executed.

Submission Instructions

Design Diagrams

You are expected to produce the following three design artefacts in Assignment 1:

- Class diagrams (distinguish between existing classes and new classes with colours)
- Interaction diagrams (sequence diagrams or communication diagrams)
- A design rationale

You will implement your design later, but for Assignment 1, you are not required to write any code. Instead, you must produce preliminary *design documentation* to explain how you will add the specified new functionality to the system. The new functionality is specified in the **Project Requirements** section.

We expect you to produce *UML class diagrams* and *UML interaction diagrams* (i.e. sequence diagrams or communication diagrams) following **the FIT2099 Assignment Rules**. These Rules are available on EdLesson.

Your class diagrams do not have to show the entire system. You only need to show 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 this class diagram. However, the overall responsibilities of the class need to be documented *somewhere*, as you will need this information to begin implementation. This can be done in a separate text document if you prefer.

To help us understand how your system will work, you must also write a *design rationale* to explain your choices. You must demonstrate both how your proposed system will work and ***why* you chose to do it that way**. You may consider using **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:

- 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.

Work Breakdown Agreement

We require you to create a simple Work Breakdown Agreement (WBA) to let us know how you plan to divide the work between your team members. There is more information on **WBAs in the FIT2099 Assignment Rules** (in Moodle). 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.

Summary

In sum, you must put your design documents and work breakdown agreement (in **PDF** format) in the design-docs folder of your Monash GitLab repository.

The due date for this assignment is at the top of the first page. We will mark a snapshot of your repository as it was at the due time. This means that you will need to notify your marker if you finished late and let them know which version they should check out.

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 working day late. Details about special considerations can be found in the Unit Information section in Moodle.

It is ALL team members’ responsibility to ensure that the correct versions of the documentation are present in the repository by the due date and time. Once all teammates have agreed on a final Assignment 1 submission, do not make further commits to the master branch of the repository until the due date has passed unless your teammate agrees. If you want to continue to make changes to the repository for some reason, create another branch.

Marking Criteria

This assignment will be marked on:

Design completeness. Does your design support the functionality we have specified?

Design quality. Does your design take into account the programming and design principles we have discussed in lectures? Look in lecture slides, and check the Object-Oriented Fundamentals documents for principles like

Practicality. Can your design be implemented as it is described in your submission?

Following the brief. Does your design comply with the constraints we have placed upon it — for instance, does your design leave the engine untouched, as required?

Documentation quality. Does your design documentation clearly communicate your proposed changes to the system? This can include:

- UML notation correctness, appropriateness, and consistency
- consistency between artefacts
- clarity of writing
- level of detail (this should be sufficient but not overwhelming)

Explanation. Can you adequately explain your design and the reasoning behind it, both in your rationale and in response to interview questions from your marker? Here are the marking criteria:

1. Clearly communicated the information – using suitable language, tone and pace.
2. Used questioning techniques to encourage views and opinions.
3. Used active listening to confirm understanding.
4. Provide possible suggestions and comments for improvement.

Marking Protocol and Rubric

For this assignment, you have been asked to prepare design documents. Here are some important instructions for completing the submission.

1. Try to put together all your design documents (UML diagrams) into a single Word or PDF instead of creating various documents.
2. Your WBA can be in a different document.
3. Put both documents into a docs or design-docs folder in the GitLab project space created for your team. Commit the design document and your WBA documents as many times you need if you update them.
4. Also, upload the same documents to Moodle (at least one team member can do this) before the deadline.
5. This means you upload the same copy both to GitLab and Moodle, but you will still be able to keep modifying them for assignments 2 and 3 in GitLab in the future.
6. Good use of git
   - All team members should check work in (at least one of the two files each)
   - Add commit comments are good (not just the GitLab defaults)

Rubric (20 Marks)

Q1: Use of git and gitlab - 2 marks maximum

2 marks: Minimum standard for 2 marks is multiple commits from all partners with helpful commit comments (note: default comments from the web UI don't count.)

1 mark: Minimum standard for half a mark is that all partners committed to git, with at least one committing. If you've managed to submit the project but only as a single bulk upload before the deadline would be one mark.

0 mark: Submitting both files as attachments to the release notes would be marked as zero.

Q2: Presentation - 4 marks maximum

All subtasks here are qualitative only and are about overall presentation quality.

4 marks: all diagrams appear complete and readable, and UML standards have been followed where applicable: inheritance/implementation/association arrowheads are correct. Design rationale is lucid and matches the submitted documentation. **The design is easy to understand.**

3 marks: High distinction. It's *almost* perfect, but there's a flaw or two: perhaps using the wrong

arrowheads, neglected to include multiplicities, messed up the layout, or the grammar in the design rationale is poor. **Still, with a little work the design can be understood by the TA.**

2 marks: Credit. One diagram is good, although the syntax might have some flaws; others might be inadequate or even missing. The design rationale is okay, but seems to leave a few questions unanswered. **The design is unclear.**

1 mark: Bare fail. Diagrams are barely comprehensible and critical design requirements might have been left out or misrepresented. The design rationale doesn't make much sense. **There are big gaps in the students' knowledge, and further consultation will be required.**

0 mark: Clear fail. Some attempt has been made at a class diagram, but it has been done very badly. A design rationale document may exist but is very unclear. The UML readings and lectures have not been followed. **Consultation is required.**

Q3: Design quality

Q3.1: Completeness - 3 marks maximum

3 marks if everything's been modelled well enough and ***all*** **functionality will be able to be supported (i.e., all the requirements)**. It's okay if the design isn't good from the point of view of design principles.

2 marks: ***most*** **functionality will be able to be supported**, and none of the major entities (e.g. Enemies, Souls, Weapons etc.) are missing.

1 mark: **at least one major piece of functionality will be able to be supported**, or several minor items have been attempted but the design is incomplete or unclear.

0 mark: a design has been attempted but the description is so vague or poorly-depicted that **it is unclear how the required functionality will be supported**.

Q3.2 Ease of comprehension - 2 marks maximum

This applies to the totality of the submitted work.

2 marks: The design rationale and diagrams are **understandable**, and nothing significant is missing.

1 mark: The design rationale and diagrams are **somewhat understandable**, but other parts are either missing or incomprehensible. Perhaps diagrams are inconsistent.

0 marks: The submission **is not understandable** at all. Perhaps it is very incomplete and/or internally inconsistent (e.g. the partners did not communicate during development).

Q3.3 Good design - 3 marks maximum

3 marks: Makes sense in the context of the existing system and reuses it wherever possible, including by inheriting existing engine classes. The design will be **straightforward to implement and easy**

to extend in Assignment 3.

2 marks: The design is **average**. The design is not bad, and should be implementable, but is lacking in some way: perhaps the interaction diagram expects unlikely functionality of classes, or something big has been insufficiently described.

1 mark: **Barely adequate**. Perhaps the design is more of a guideline than a plan, or perhaps it's hacky. This is the most that a design can get if it involves alteration to engine classes -- because that's hacky.

0 mark: **Inadequate design**, some attempt has been made but there is no clear link with the existing engine. It is unclear how the design addressed the requirements.

Q3.4 Design is well-justified - 3 marks maximum

3 marks: The design rationale **makes good points** about all (or nearly all) significant design decisions that are reflected in the other design documents.

2 marks: Says something sensible about most design decisions. Alternatively, the design rationale is good, but is **only partially consistent** with the design described by other documents.

1 mark: Very incomplete, or inconsistent with the design submitted, or vocabulary relating to design is not being used appropriately. (For example, the student might assert things about "minimizing dependencies" or "maximizing cohesion" without any context or evidence.)

0 marks: **no design rationale is provided.**

Interview - 3 marks maximum (individually allocated)

All students in the team should attend the marking interview which will happen in the lab immediately after the deadline. The TA will ask each student three or more questions about the submission. These marks will be awarded accordingly:

3 marks: If all questions are responded adequately. The responses demonstrate that the student understands the various parts of the assignment.

2 marks: If all the questions but one is responded adequately and sensibly. The responses still demonstrate some knowledge about the student's own work.

1 mark: If two or more of the questions are not responded adequately and sensibly. The remaining question(s) is/are partly responded but it is unclear whether the student understands their own work.

> ⚠ **IMPORTANT:** If **two students** in the team are awarded this mark, this would automatically flag this assignment as a potential case of plagiarism for the whole team and it will be further investigated using a similarity check software and zero marks can be awarded as a result.

0 marks: The student cannot justify their own assignment, none of the responses is sensible**.**

IMPORTANT: If **one student** in the team are awarded this mark, this would automatically flag this assignment as a potential case of plagiarism for the whole team and it will be further investigated using a similarity check software and **zero marks will be immediately awarded for the team**.

If you completed the assignment by yourselves there is nothing to worry about during these interviews. It will be an opportunity to receive some feedback to consider in assignment 2, which involves implementing your own design. You will receive additional feedback once the marks are awarded. Remember, this is a team task so you should be aware about how the parts designed or coded by others 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.

WBAs

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