Assignment 3: Further Design and Implementation

Learning Outcomes

In this assignment, you will design and implement some new game functionality. 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;

✓ **LO3**. Implement object-oriented designs in an object-oriented programming language (i.e., Java), using object-oriented programming constructs such as classes, inheritance, abstract classes, and generics as appropriate;

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

To demonstrate your ability, you will be expected to:

- design and implement further extensions to the system
- use an integrated development environment to do so
- update your UML class diagrams and interaction diagrams as required, to ensure that they match your implementation
- use git to manage your team's files and documents

The marking scheme for this assignment reflects these expectations.

Project Requirements

✓ **All the requirements stated in the Assignment 1 and 2 specifications still apply.** You must document your designs as you did for Assignments 1 and 2.

In this assignment, you will design and implement **three (3)** fixed requirements and **either one** (for Group of 2) or two flexible requirements **(Group of 3)**. You will develop four or five new requirements to complete the game.

Please note that FLEXIBLE requirements **ARE NOT OPTIONAL**, so you **MUST** include them.

1. Fixed Requirements

You must **design and implement three (3) fixed requirements**, regardless number of people in your team. Those fixed requirements are REQ1, REQ2, and REQ3.

2. Flexible Requirements

We hope that you enjoy working on the assignment so far! We also hope that you are comfortable working with the engine code. Since this unit is about design, we give you the opportunity to do a bit of creative work! In this part, **you have two choices**:

- **Structured Mode:** we decide the features that you must work on (no hassle).
- **Creative Mode:** you can go wild with your imagination. In other words, you can be creative in designing and implementing your own requirements. However, **you must strictly follow rules and standards** that we set so that we can grade your work fairly (see table image).

✓ You must clearly declare your choice in the `README.md` document (at the root of your repository) whether you choose **Structured Mode** or **Creative Mode**.

So, a **Group of 3** must design and implement **two (2) flexible requirements**, while a **Group of 2** must design and implement only **one (1) flexible requirement**.

2a. Structured Mode

If you decide to use our features, please complete **REQ4** and **REQ5**. If you are in a group of 2, you may choose either **REQ4** or **REQ5.**

✗ You must **NOT** design and implement two features that are labelled as "Structured Mode" if you choose **Creative Mode**!

2b. Creative Mode

If you prefer to create your own features, we will compensate for your creativity and extra work with a bonus mark. The total original group score becomes **105%.** In other words, you will get an extra 5% mark. For instance, you will get 82.95% if the original mark is 79% (that's an HD!). However, the following rules apply:

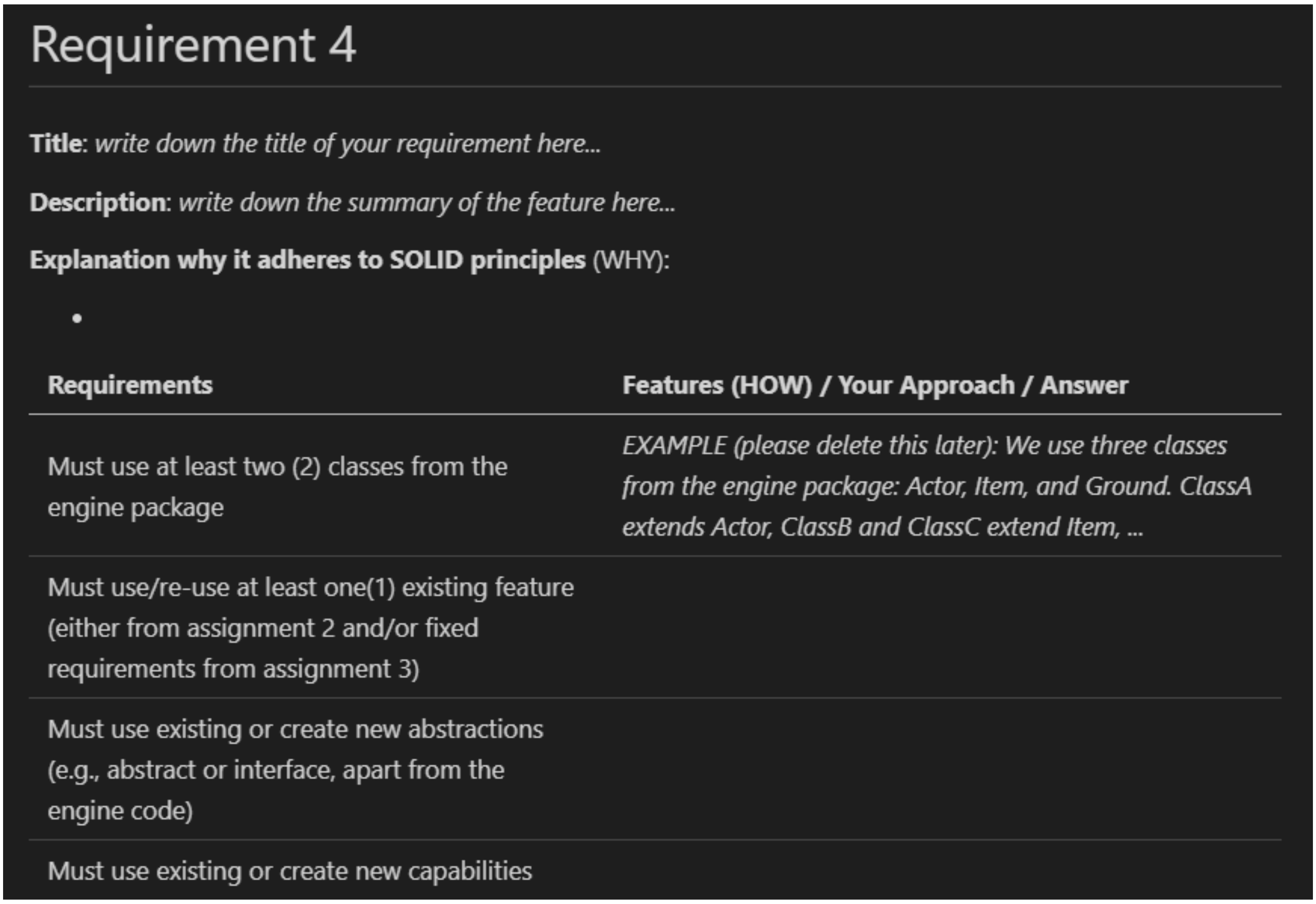
- You must use the **MINIMUM REQUIREMENTS** table to meet our standards. The description must be specific with the appropriate design reasoning.
- You must attach this information (description and tables) to the **README.md document** at the root of your repository. If you are a group of three, you will have two separate tables (2 complex features).
- You must ensure new features do not affect the existing system (old features).
- Your feature(s) must be new and original (you cannot use the major ideas from the Structured Mode features).
- You must explain why each feature illustrates the application of SOLID principles.
- For simplicity, you are not allowed to combine features from Structured Mode with this Creative Mode. For example, you cannot pick one feature from Structured Mode and then create a new unique feature.

If your requirements violate any of the rules above, you will **NOT** get a bonus mark and we will mark you normally. Don't worry, no penalties apply.

You **MUST** discuss your requirements' proposal with your TAs to get approval **in Week 10** (Your TAs need to record this information). Otherwise, NO bonus mark will be given.

Following that, you must present the features proposal with the provided **MINIMUM REQUIREMENTS template table** in your discussion.

MINIMUM REQUIREMENTS table for ONE feature that we can consider complex:


Requirement 4

Title: *write down the title of your requirement here...*

Description: *write down the summary of the feature here...*

Explanation why it adheres to SOLID principles (WHY):

-

Requirements	Features (HOW) / Your Approach / Answer
Must use at least two (2) classes from the engine package	*EXAMPLE (please delete this later): We use three classes from the engine package: Actor, Item, and Ground. ClassA extends Actor, ClassB and ClassC extend Item, ...*
Must use/re-use at least one(1) existing feature (either from assignment 2 and/or fixed requirements from assignment 3)	
Must use existing or create new abstractions (e.g., abstract or interface, apart from the engine code)	
Must use existing or create new capabilities	

Please download a Markdown file below:

creative-mode-table.md

Examples

Here are several general examples to give you some ideas. You may pick **one** or **combine two or more** of the following ideas to be considered one **COMPLEX** feature/requirement (i.e., REQ4). Otherwise, feel free to create your own feature as long as you can complete the table above :). Please repeat as required.

- When Mario dies, reduce their life (3 hearts) and restart the game automatically.
- Create new enemies that have unique abilities and weapons. Place them somewhere in the second map or third map (create a new map).
- Fog of war feature to increase the tension/difficulty of the game, where the enemies are invisible from the map until the player is within a certain radius.
- A system to generate random locations to place enemies and trees every time the game restarts.
- Create new allies and enemies (e.g. Luigi, Wario, Waluigi etc.).
- Upgrade bottle charges by consuming a new item or buying the service from Toad to upgrade it.

- A patrol behaviour (moves at the set path, back and forth).
- Blink to a location that is several steps away within one turn.
- Additional and extra buff effect (burned, poisoned, frostbite) where it can inflict or deal damage per turn or reduce the Player's maximum hit points.
- More unique services and items from Toad such as increased strength, intelligence, stamina, agility, and more.
- Add Yoshi as your adventure partner!
- Menu to select the main player to be Luigi or Mario, and each character should have unique traits.
- And more...

REQ1: Lava zone

Scenario

1. New Map

It is time to expand our game space! We will create another map that is a bit smaller but much more challenging than the previous map. Creating a new map means that you should not just decrease the size of the current map. Place some random blazing fire grounds (Lava `L`) that will inflict 15 damage per turn when the player steps on them. Enemies cannot step on this lava.

> All actors' activities in both maps should run normally and be printed in the log. You have the freedom to design how does the second map look like as long as you include all required features.

2. Teleportation (Warp Pipe) `C`

Mario can teleport to the other map back and forth through a warp pipe `C`. However, the warp pipe is blocked by a Piranha Plant. Once you killed Piranha Plant, you can stand on the pipe (i.e., using jump) and use it to teleport to another pipe on the second map. You can travel back to the last/previous pipe that teleported you before.

Implementation Expectations

- After creating a second map (Lava Zone), place one Warp Pipe in the top-left corner.
- Place several warp pipes on the first map, scattered around randomly.
- When we play the game for the first time, we only see warp pipes. Piranha Plants will show up in the following turn.
- Go to one of the Piranha Plant, and kill it. You should not be able to see teleportation action until you kill the plant. Once you have killed it, you can jump to the warp pipe. Here, you'll see an option to "Teleport to Lava Zone"(second map).
- If you choose this option, you should be moved to the second map, and now you are standing on top of a warp pipe that you've created on the second map. And you will have an option to teleport back to the previous pipe.
- Doing so should bring you back to the exact position of the pipe in the first map. Assumption: When you are teleporting, you may instantly kill Piranha Plant that is on a Warp Pipe.
- Find another pipe, kill the Piranha Plant, and teleport. Again, it should bring you to the second map (on the pipe at the top-left corner), and teleporting back will bring you to the last pipe, not the first pipe.

REQ2: More allies and enemies! □☠

Scenario

1. Princess Peach `P` □□

Princess Peach is held captive by Bowser! Place her somewhere on a Dirt in the Lava zone, and she stands next to Bowser and cannot move, attack, or be attacked. Once you have defeated Bowser and obtained a key, you can interact with her to end the game with a victory message!

2. Bowser `B` □😈

Bowser will stand still, waiting for Mario on the second map. Once Mario stands next to him, Bowser will attack and follow Mario! He will attack whenever possible. Whenever Bowser attacks, a fire will be dropped on the ground that lasts for three turns. That fire will deal 20 damage per turn to anyone. When the Bowser is killed, it will drop a key to unlock Princess Peach's handcuffs.

It has the following stats:

- 500 hit points
- "punch" attack with a 50% hit rate and 80 damage.

Resetting the game (`r` command) will move Bowser back to the original position, heal it to maximum, and it will stand there until Mario is within Bowser's attack range.

3. Piranha Plant `Y` □□

As mentioned in the previous requirement, Piranha Plant `Y` will spawn at the second turn of the game (note: the first turn is when you start the game for the first time). This plant will attack Mario when he stands next to it. Piranha Plant cannot move around. It will not follow anyone when it is engaged in a fight. It has the following stats:

- 150 hit points
- "chomps" attack with 50% hit rate and 90 damage.

Once the player kills it, the corresponding `WarpPipe` will not spawn Piranha Plant again until the player resets the game (`r` command). Resetting will increase alive/existing Piranha Plants hit points by an additional 50 hit points and heal to the maximum.

4. Flying Koopa `F` □□

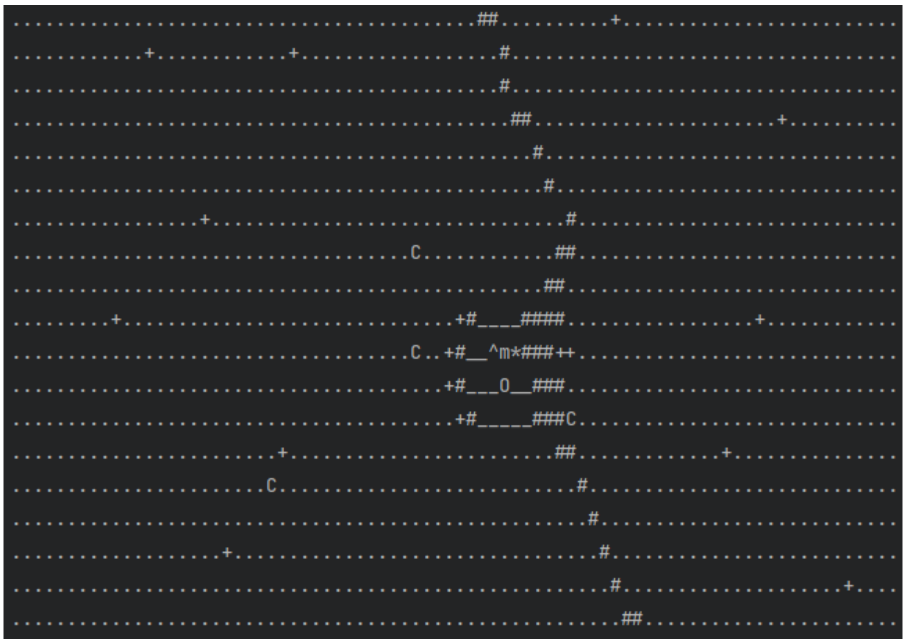
Flying Koopa `F` has pretty much similar characteristics as the original Koopa, except it has a bigger health bar (150 hit points). Furthermore, it can walk (fly) over the trees and walls when it wanders around. It also applies when Flying Koopa follows Mario. Like Koopa, it will go to a dormant state whenever it is unconscious until Mario brings a wrench to crack the shell (killing it)! Cracking its shell drops a Super Mushroom. The mature tree has a 50% chance of spawning either normal Koopa or Flying Koopa.

HINT: Please be aware of the LSP violation here. :)

Implementation Expectations

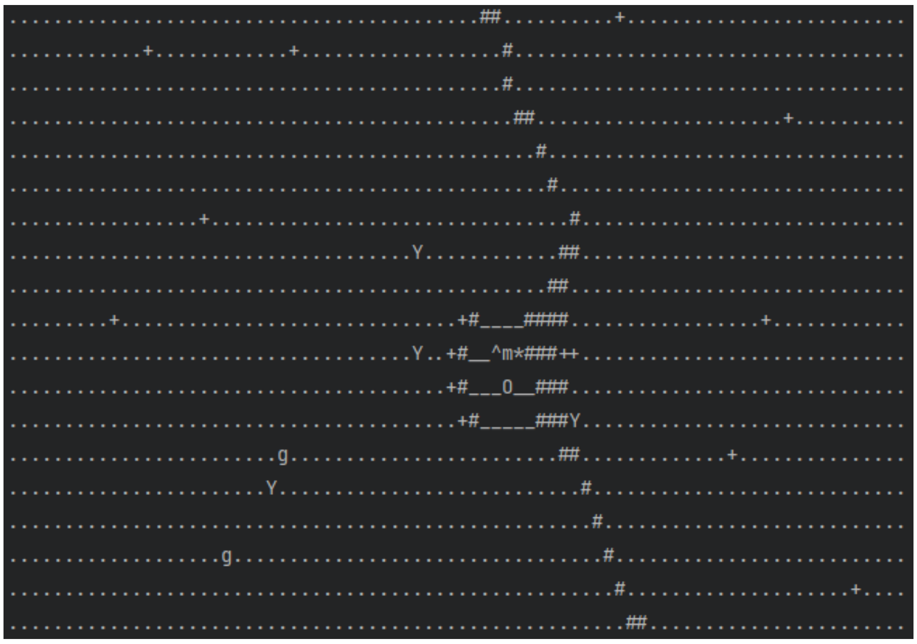
- Place **Princess Peach** somewhere on the second map, near Bowser.
- In the first map, after killing Piranha Plant, step onto the Wrap Pipe, and teleport to the second map.

Turn 1 (when the game runs for the first time):



```
..........................................##..........+.........................
............+............+..................#...................................
............................................#...................................
.............................................##......................+..........
...............................................#................................
................................................#...............................
.................+................................#.............................
....................................C............##.............................
................................................##..............................
.........+..............................+#____####.................+............
....................................C..+#__^m*###++.............................
.......................................+#___O__###..............................
........................................+#_____###C.............................
........................+........................##.............+...............
.......................C...........................#............................
....................................................#...........................
...................+.................................#..........................
......................................................#....................+....
.......................................................##.......................
```

Turn 2:



```
..........................................##..........+.........................
............+............+..................#...................................
............................................#...................................
.............................................##......................+..........
...............................................#................................
................................................#...............................
.................+................................#.............................
....................................Y............##.............................
................................................##..............................
.........+..............................+#____####.................+............
....................................Y..+#__^m*###++.............................
.......................................+#___0__###..............................
........................................+#_____###Y.............................
........................g........................##.............+...............
.......................Y...........................#............................
....................................................#...........................
...................g.................................#..........................
......................................................#....................+....
.......................................................##.......................
```

- At first, Bowser does nothing until Mario approaches him (standing in Bowser's adjacent squares).
- After Bowser lands an attack, Mario needs to step out from that ground so that we can see a fire symbol `v` on that ground. Note: Bowser is not immune to its fire (you are welcome to make Bowser immune to its fire, but not against Fire Flower damage).
- Bowser will follow Mario and attack whenever possible. Mario may receive two kinds of damage within one turn: normal hit and fire damage.
- After defeating Bowser, we should be able to see a key `k`, pick it up, and there should be an action from Princess Peach to unlock and end the game. You are welcome to expand the ending game scenario (e.g., bring back Peach to the Mushroom Kingdom)

REQ3: Magical fountain ▯

Scenario

1. Bottle `b` ▯

Let's add a magical bottle that can contain unlimited magical waters. Mario will have this bottle in its inventory at the beginning of the game (before it starts). This bottle will be a permanent item that cannot be dropped or picked up. This bottle can be filled with water, and Mario can drink/consume the water inside the bottle. Each water will give a unique effect depending on its original source (see Fountains below).

2. Fountains ▯

Mario can refill the bottle when he stands on the fountain. A fountain produces an endless amount of water. Each water from a fountain provides different effects that will help Mario in his journey.

2a. Health Fountain `H` ▯ Drinking this water will increase the drinker's hit points/healing by 50 hit points.

2b. Power Fountain `A` ▯ When the water is consumed, it increases the drinker's base/intrinsic attack damage by 15.

> Note: When attacking an enemy while having a weapon in the inventory (equipping), the hit rate and attack damage will be based on that weapon. For example, the Mario has 200 base attack damage, but it holds a Wrench that has 80% hit rate and 50 damage. When Mario attack Goomba, the damage will be 50 points.

Optional Challenges ▯

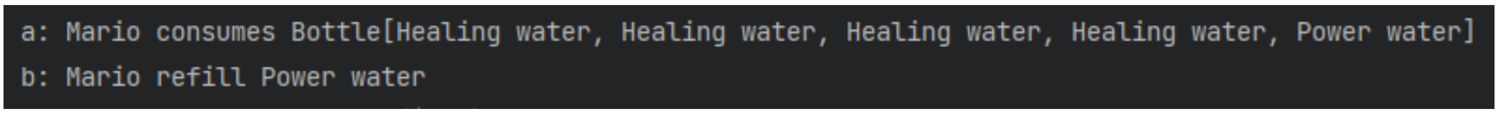
- All enemies who stand next to a fountain can drink the water and gain its effect straightaway from the fountain. Ensure that you find a balance in behaviour priorities (i.e., when to drink, follow, attack, and wander around).
- Mario doesn't have a bottle in his inventory at the start of the game. Instead, Mario will obtain it from Toad. However, it has no price, and you shouldn't use purchase or trade action.
- A fountain has **limited** water. For the sake of simplicity, all fountain has a maximum of 10 water capacity. In the console, it should print: `Mario refills bottle from Health Fountain (10/10)`, where `(10/10)` means there are 10 out of 10 waters inside of this fountain. Once all water is either drank or used to fill up a bottle, the water will be replenished to the maximum in the next five turns. (Note: you may decide how much water per sip or fill. For example, a sip/drink will automatically consume 5 water slots, and filling up to a bottle only uses one water slot).

FURTHER CLARIFICATION: That "five turns" timer will run when the waters in the Fountain are fully exhausted. For example, a Fountain has 10 water slots, and then an enemy drinks 10 times -- if in one turn it drinks 5 slots, then it will be fully exhausted in the second turn. Once it is empty, the recharge timer starts. After 5 turns, it becomes full again (10 water slots).

✓ Completing **ALL of these** (FULLY FUNCTIONAL - NOT JUST AN ATTEMPT) extra features will get an additional **+1 mark in Criteria 2 (Functionality)**. If you already reach full mark in this criteria, we will add that to your total mark. You also need to ensure that doing this additional features adhere to the good object-oriented design principles. It means these additional features will influence design criterias/mark in the assessment. Tell your interviewer if you've implemented these optional challenges.

Implementation Expectations

- Place two fountains (Health Fountain and Power Fountain) near the starting point. We do this for marking purposes; you may put them far away later.
- When Mario is standing on top of the fountain ground, you should see an action to fill the bottle with its water.
- The bottle should be filled up like a stack (HINT: it uses the stack, not implementing it): First in, Last out. So, this bottle can be filled up with any kind of water, depending on its stack order. Push means to fill in the water to the bottle, and pop means drinking the water.
- Mario may fill up and drink the water as much as possible (until you look like this).



```
a: Mario consumes Bottle[Healing water, Healing water, Healing water, Healing water, Power water]
b: Mario refill Power water
```

REQ4: Flowers ▯ (Structured Mode)

You must **NOT** design and implement this feature if you decide on doing **Creative Mode**!

Scenario

1. Fire Flower `f` ▯

For every growing stage of the tree (Sprout -> Sapling and Sapling -> Mature), it has a 50% chance to spawn (or grow) a Fire Flower on its location. Mario can consume this fire flower to use Fire Attack. There could be multiple Fire Flowers in one location.

2. Fire attack `v` ▯

Once the actor consumes the Fire Flower, it can use fire attack action on the enemy. Attacking will drop a fire `v` at the target's ground. This "fire attack" effect will last for 20 turns (i.e., it is pretty similar to being INVINCIBLE for 10 turns, please print a different text in the implementation). The fire will stay on the ground for three turns. The fire will deal 20 damage per turn.

Implementation Expectations

- After playing several turns, you'll notice that some places on the map shows `f` (Fire Flowers are growing).
- When Mario is standing in this location, we will see an action (or some actions) to consume the Fire Flower.
- After consuming it, whenever Mario attacks the enemy, it will drop some fire (`v`), similar to Bowser's attack.
- Trying to attack with Fire Attack should have "with fire" in the menu description:

```
b: Mario attacks Piranha Plant at West with fire!
```

- Mario is not immune to this fire. But, feel free to allow such immunity in the implementation.

REQ5: Speaking (Structured Mode)

✖ You must **NOT** design and implement this feature if you decide on doing **Creative Mode**!

Scenario

Each listed character will talk at every "even" or second turn (alternating). It means, they don't talk all the time. They will pick random sentences and a list of monologues by actors:

- Princess Peach
  - "Dear Mario, I'll be waiting for you..."
  - "Never gonna give you up!"
  - "Release me, or I will kick you!"
- Toad: Monologue, Assignment 1
- Bowser
  - "What was that sound? Oh, just a fire."
  - "Princess Peach! You are formally invited... to the creation of my new kingdom!"
  - "Never gonna let you down!"
  - "Wrrrrrrrrrrrrrrrryyyyyyyyyyyyyy!!!!"
- Goombas
  - "Mugga mugga!"
  - "Ugha ugha... (Never gonna run around and desert you...)"
  - "Ooga-Chaka Ooga-Ooga!"
- All Koopas
  - "Never gonna make you cry!"
  - "Koopi koopi koopii~!"
- Flying Koopa (addition):
  - "Pam pam pam!".
- Piranha Plants
  - "Slsstssthshs~! (Never gonna say goodbye~)"
  - "Ohmnom nom nom nom."

It should print in the following format:

```
[actor]: "[monologuesentence]"
```

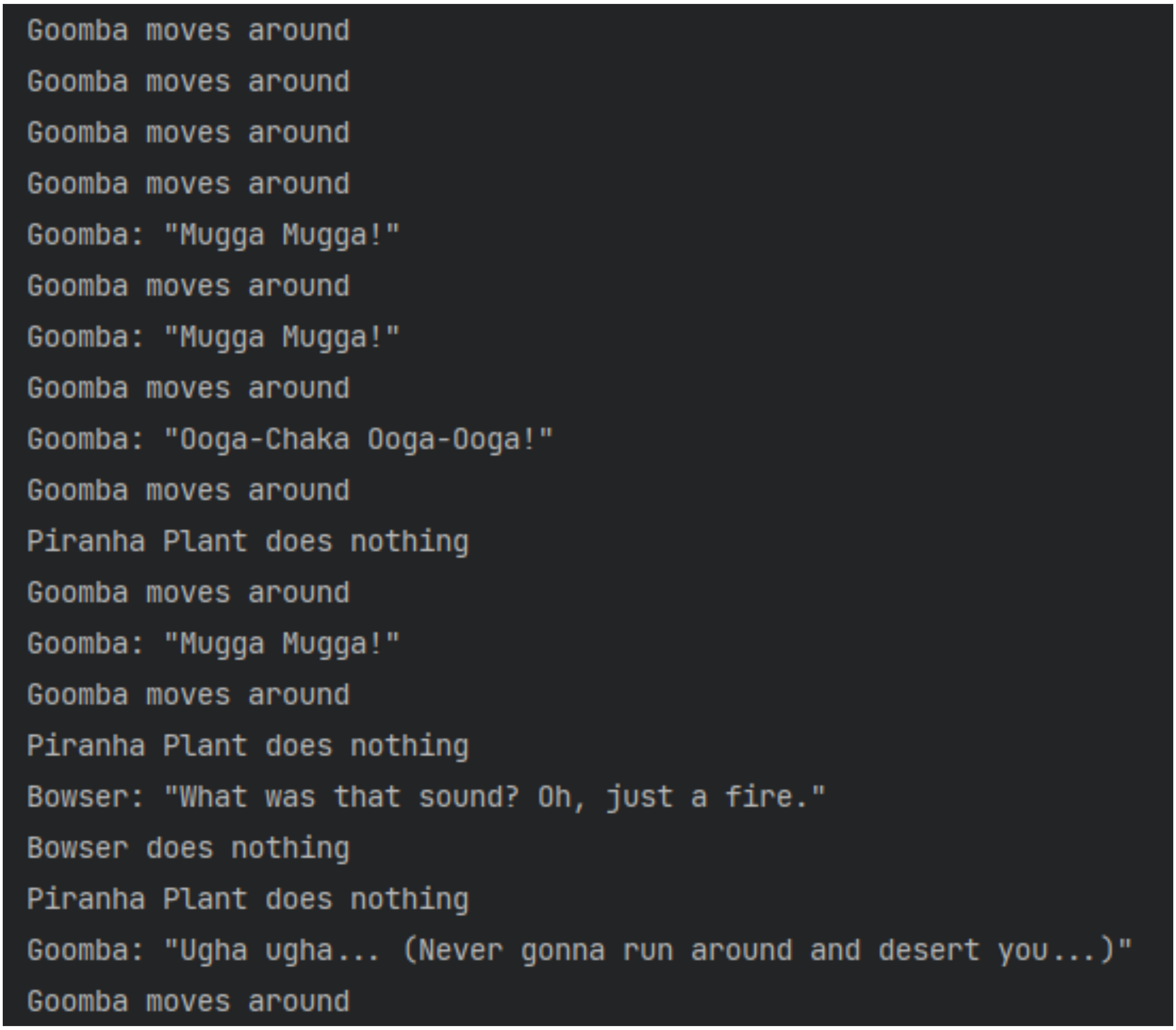
For example:

```
Toad: "You might need a wrench to smash Koopa's hard shells."
Princess Peach: "Bowser's Castle... I think this is the first time I've come here on my own two fee
```

If you can think of any better monologues, please feel free to modify or add them. 😃

Implementation Expectations

- Ensure that your design doesn't extend from concrete classes that could lead to a tight coupling problem.
- Ensure that the sentence is not using a simple string that will be printed by the I/O straightaway. Instead, try to create a class and use that instance (e.g., something like `Monologue`? Try to think about how to use it).
- After playing several turns, you should see something like this in the console. Sometimes they say something; sometimes they don't (due to speaking at alternate turns, depending on when the actor was instantiated).



```
Goomba moves around
Goomba moves around
Goomba moves around
Goomba moves around
Goomba: "Mugga Mugga!"
Goomba moves around
Goomba: "Mugga Mugga!"
Goomba moves around
Goomba: "Ooga-Chaka Ooga-Ooga!"
Goomba moves around
Piranha Plant does nothing
Goomba moves around
Goomba: "Mugga Mugga!"
Goomba moves around
Piranha Plant does nothing
Bowser: "What was that sound? Oh, just a fire."
Bowser does nothing
Piranha Plant does nothing
Goomba: "Ugha ugha... (Never gonna run around and desert you...)"
Goomba moves around
```

Assignment Expectations

Design is important

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

If you would like informal feedback on your design at any time, please consult your TA in a lab or attend any consultation session.

You might find that some aspects of your existing design need to change to support the new functionality. You might also think of a better approach to some of the requirements you have already implemented — your understanding of the requirements and codebase will have improved as you have progressed. You might also spot places where your existing code (and thus design) can benefit from refactoring.

If you want to update your design, you may do so; if you decide to do this, update your design documents to match the code and write a brief explanation of your changes and the reasons behind them. This will help your marker understand the thinking behind your code.

Coding and commenting standards

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

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

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

Submission Instructions

The due date for this assignment is at the top of this document, your local time. We will mark your Assignment 3 on the state of the “master” branch of your Monash GitLab repository at that time. If you have done any work in other branches, make sure you merge it into master before the due time. Additionally, please, make sure at least one team member uploads a copy of the repository to Moodle. This step is compulsory.

> ⚠ **Do not create a new copy of your work for Assignment 3.** Continue working on the same files, in the same directory structure (you might like to add a tag to your final Assignment 2 commit before starting on Assignment 3, so you can find that version easily). *https://git-scm.com/book/en/v2/Git-Basics-Tagging*

As we said above, you may update your design and implementation if you find that your Assignment 2 solution is not suitable for extension or if you think of a better approach during Assignment 3.

You must update your Work Breakdown Agreement to include the work necessary for the Assignment 3 requirements. We will take your Work Breakdown Agreement into account when marking if there seems to be a major discrepancy in the quality of different parts of the submission or if the code is missing major sections. If you choose to reallocate tasks, keep your WBA up to date.

In summary, we expect you to produce:

- Updated or **new UML class diagrams** with new features.
- At least one **NEW interaction diagram** from new features.
- Updated Design Report in the `.md` or `.pdf` format.
- Updated WBA document.
- A playable game with newly added features (code).

Marking Criteria

This assignment will be marked based on:

- **Functional completeness**
- **Design quality**
    - adherence to design principles discussed in lectures
    - UML notation consistency and appropriateness
    - consistency between design artefacts – clarity of writing.
    - level of detail (this should be sufficient but not overwhelming)
- **Code quality**
    - readability
    - adherence to Java coding standards
    - quality of comments
    - maintainability (application of the principles discussed in lectures)
- **Correct use of GitLab**

Marks may also be **deducted** for:

- late submission
- inadequate individual contribution to the project
- academic integrity breaches

Protocol and Rubric

Submission

For this assignment, you have been asked to extend the functionality beyond those you designed and implemented in Assignments 1 and 2. You are free to change your design, of course. Here are some important instructions for completing the submission.

1. You can update your design documents directly on GitLab.
2. Make sure the implemented code is in the main branch on GitLab.
3. Make sure you haven't modified the engine.
4. Before the final deadline, download a copy of your repository and upload it to Moodle (at least one team member can do this).

Rubric (20 marks)

Q1: Use of git and GitLab - 1 marks maximum

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

0.5 mark: Minimum standard for one mark is that all partners committed to git, with at least 2 commits.

0 mark: If the project has been submitted but only as a single bulk upload before the deadline would be a mark of zero. Commits without meaningful comments can also contribute to a mark of zero.

Q2: Functionality - 4 marks maximum

4 marks: all the NEW functionality indicated in Assignment 3 requirements (REQ1, REQ2, etc... in assignment specifications) has been implemented and all the items work well. The application doesn't behave incorrectly.

- 3.5 marks: implementing all the functionality indicated in the requirements (assignment specifications) has been attempted but something is either incorrect or incomplete.

3 marks: the implementation of most functionality has been attempted and most of it works, but may be unstable or buggy. No more than **1 functionality heading** (according to the specifications: REQ1, REQ2, etc.) hasn't been attempted/implemented. This means all the remaining functionality headings have been covered.

2 marks: No more than **2 functionality headings** (according to the specifications: REQ1, REQ2, etc.)

hasn't been attempted/implemented. This means all the remaining functionality headings have been covered.

1 mark: No more than **3 functionality headings** (according to the specifications: REQ1, REQ2, etc.) hasn't been attempted/implemented. This means all the remaining functionality headings have been covered.

0 marks: it is unclear how the functionalities have been implemented/covered or the application doesn't compile.

Q3: Style and readability - 2 marks maximum

Things to complete:

- Meaningful and descriptive names for classes, interfaces, attributes, methods, parameters, local variables, enums etc.
- Adherence to Java style guide
  - Lower case names for packages
  - CamelCase names for classes
  - dromedaryCase names for methods, attributes, parameters, variables
  - UPPERCASE names for constants, including enums
- Javadoc
  - For all public methods; optional for protected/package/private
  - Should be buildable; don't give full marks if it doesn't build (it's usually a failure to HTMLize < and > signs in the comments)
  - Should describe what the method does and what its parameters are

2 mark: All nontrivial methods documented with Javadoc, adheres to standard Java naming conventions this is: (-- upper case names for classes, camelCase for methods, attributes, variables, and parameters, lowercase for packages), comments are useful and match code. Variable and method names are informative.

1 marks: Commenting and Javadoc patchy but adequate, mostly adheres to standard Java naming conventions (see above).

0 marks: Uncommented or very nearly so.

Q4: Supporting design documentation - 3 marks maximum

> Note: this rubric item is focused on the match between the design and the implementation and the correctness of the design in terms of notation (the UML diagrams). This should exist for new parts of the code. If the design of Assignment 2 functionality has had to change, the documentation should be kept up to date.

⚠ Note that we do expect at least one interaction diagram in Assignment 3 (a sequence diagram or a communication diagram).

3 marks: The **design documentation is consistent with the implemented system** and all UML notation is correct. The design rationale reflects this consistency.

2 marks: The design documentation is mostly consistent with the implemented system but **some discrepancies** exist. Notation is mostly good but may have some problems. The design and the documentation matches somehow the design rationale.

1 mark: Some aspects of design are consistent with the system. Notation may be poor. The design rationale may not be completely aligned to the design documentation.

0 marks: The design needed to be updated but it wasn't or the implementation doesn’t match the design.

Q5: Design and implementation of new system components - 4 marks maximum

> *Note: this rubric item is focused on the quality both of the design and its implementation. Design documents should cover the entire specification, even if not all of it was implemented. Important: on top of the rubric definitions below,* ***one mark will be deducted per specification heading being omitted in the design/implementation (out of the X headings in the assignment specifications document). At most two marks will be deducted if more than one specification heading has been omitted (those not even attempted).***

> Consider the implementation of or attention to the following constructs: encapsulation, abstraction, interface quality, connascence within the system, classes that seem to be trying to do too much, and thinking in terms of "sending messages to objects" or "passing data between methods".

4 marks: All the functionality is well-designed and implemented and nothing is missing. The design is good. This means: core Object-Oriented principles (i.,e, SOLID principles, inheritance, encapsulation) were considered when implementing the design.

3 marks: Design is good, but it is more complicated than it needs to be: e.g. unnecessary dependencies or connascence, methods added when functionality could have been implemented using existing methods, repeated code, etc.

2 marks: Very complicated approaches to simple problems

1 mark: A large part of the design is unworkable.

0 marks: The design is unimplementable as it stands or has not been done.

Q6: Integration with the existing system - 3 marks maximum

> **Important:** on top of the rubric definitions below, one mark will be deducted per specification heading being omitted in the design/implementation. At most two marks will be deducted if more than one specification

heading has been omitted.

> "Using the engine classes well" means using them where appropriate and not using them where it's not appropriate.

Please check in particular for:

- Good use of base classes
- Reuse of logic, e.g. in behaviours, instead of cut and paste
  - Some new behaviours and actions might need refactoring so that code can be shared, e.g. if reusing direction finding in HuntBehaviour to create "fleeing" logic
- Correct use of interfaces in the interfaces package.

> For the purposes of this exercise, "existing game classes" includes, besides classes included in the engine, also the classes created in Assignment 2. Obviously, you're allowed to change those, provided they don't break everything.

3 marks: **Sensible use** of provided code (the engine and classes created in Assignment 2). The team understands how to use the engine, e.g. the distinction between Action and Behaviour, how to use the interfaces packages, etc.

- 2.5 marks: Mostly uses provided code in the way it was intended, but something seems to be missing or is problematic.

2 marks: Makes some good use of provided code, but **some unexpected use of the engine is encountered**, for example, in ways that affect the correctness of behaviour.

1 mark: The engine provided has been used in odd ways or **changed in minor ways**. If a change to the engine was reverted before submission it is ok, if it affected only whitespace, or if the change was done by a staff member it is also ok. Some poor use of existing functionality is present: needlessly extending an Item for items that could have been left as Item instances, unjustified super() calls, shadowing base class variables, etc.

0 marks: **The engine code has been changed** in substantial and non-approved ways.

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