S1 2022

FIT2099 Object-Oriented Design and Implementation

Assignment 1: Analysis and Design

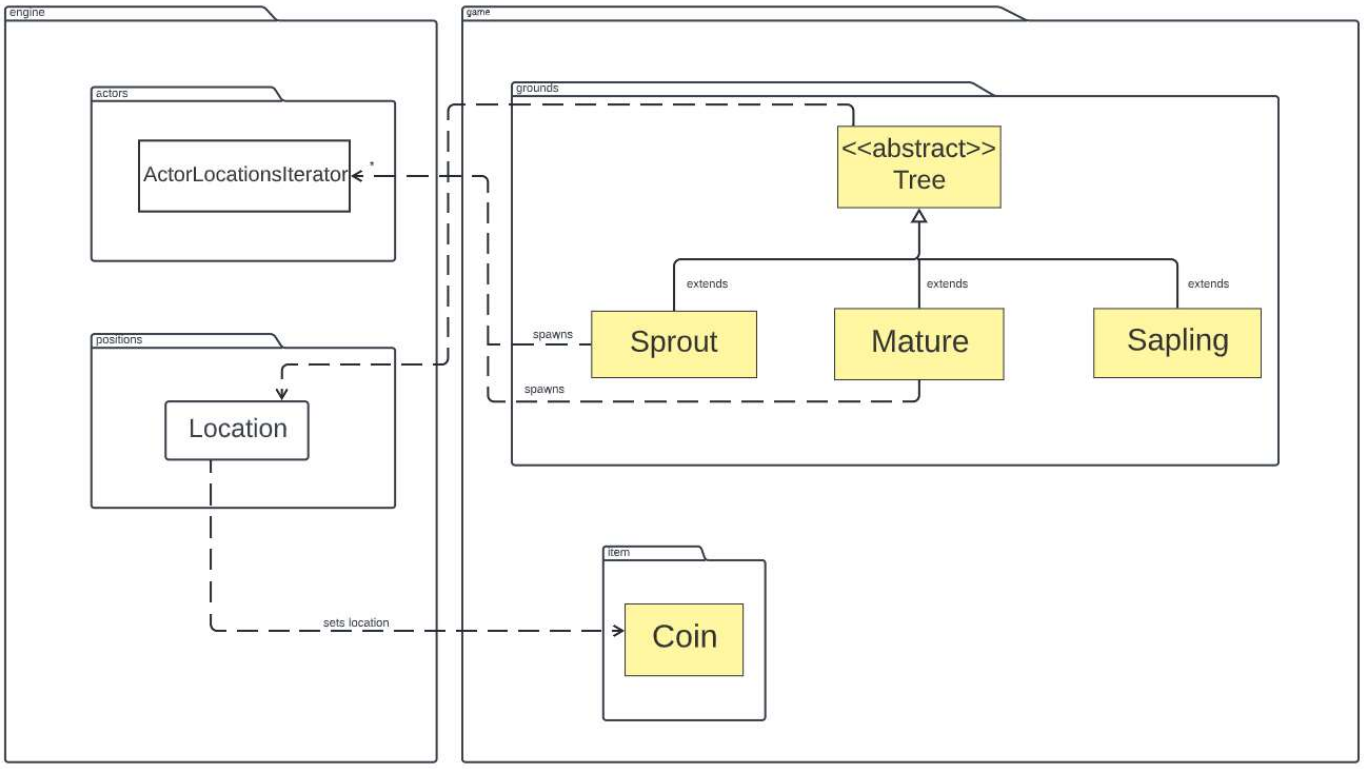
Lab_14Team6:

30041880 Junhao Li
31950124 Ashton Sequeira
29693500 Kenda Wan

Table of Contents

UML and Design Rationale

REQ 1 : Let it Grow ! 🌳

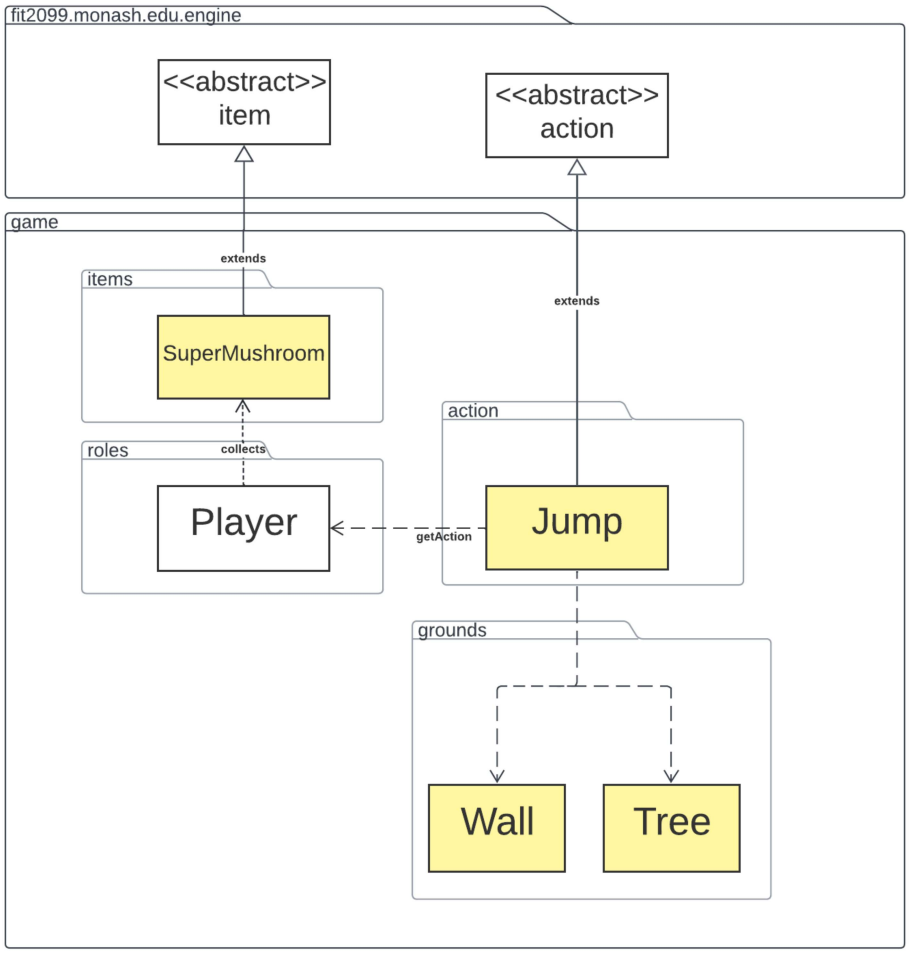


Tree class is made abstract and Sprout, Sapling and Mature extend the Tree class: While Tree extends Ground class, we plan to extend Tree with Sprout, Sapling and Mature, each with their own unique capabilities. This will help us follow the Single Responsibility Principle.

Tree class has a dependence on Location: Since the Sprout, Mature and Sapling Classes extend the Tree class, it can use methods in the Location class. The Sprout can grow to Sapling and the Sapling can grow to Mature and Mature can revert back. This is implemented within the Location class which the Grounds class extends.The setGround method in Location allows us to change the ground type. In the same way, a coin can be dropped by a sapling on every turn(10% probability) using the setItem() method in Location class.

Sprout and Mature has an association with ActorLocationIterator: Sprout and Mature both have probability to spawn Goomba and Koopa respectively in every turn. ActorLocationtIterator has an association with the Actor class. Methods in ActionLocationIterator like add,remove can help us spawn Goomba and Koopa.The contains method in ActionLocationIterator can be used to check if the player is at the location of the Sprout or Mature and can stop it from spawning the enemy at that time.

REQ 2 : Jump Up, SuperStar!

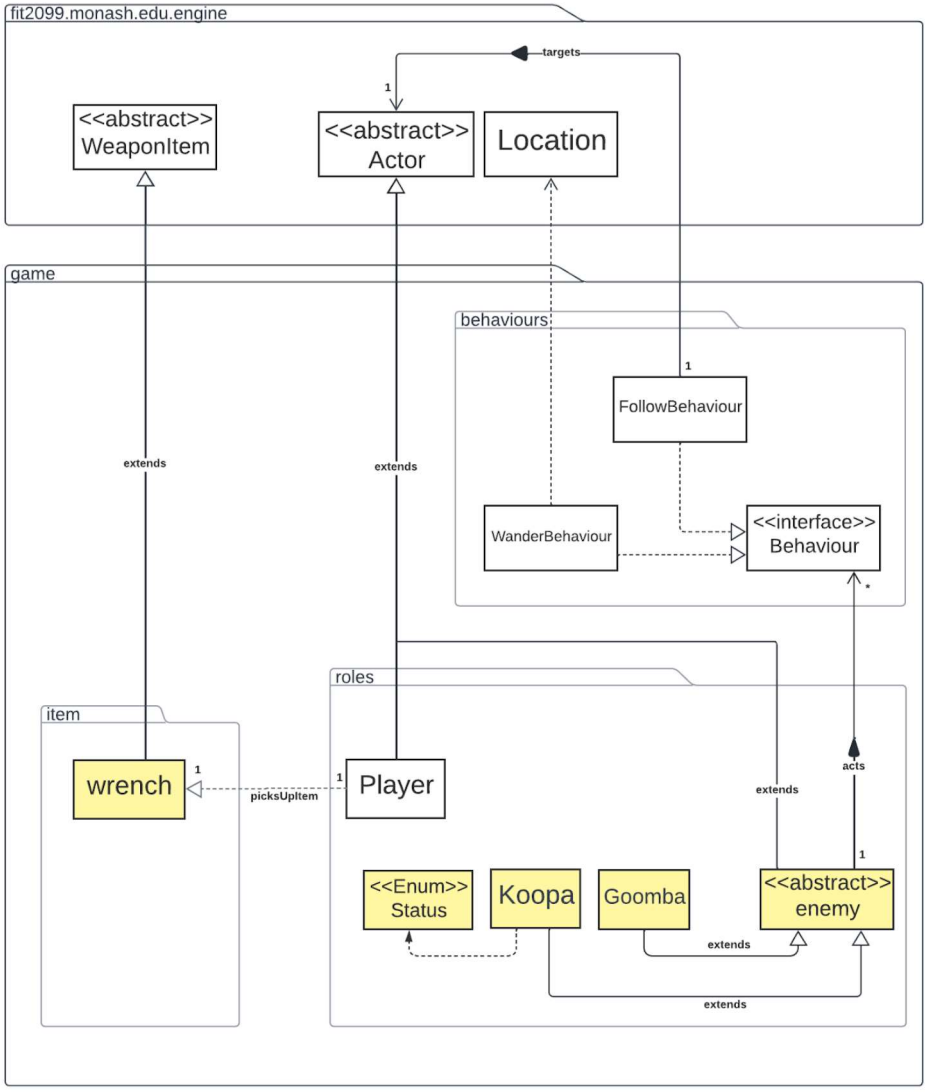


The new SuperMushroom class extends abstract Item class (ultimately, we are implementing a class where SuperMushroom implements an abstract class, however, here will only show the relationship between SuperMushroom and Player). While Item implements Printable and Capable packages it can display the call Item (name, displayChar, portable). Additionally, Item imports Action and Location packages therefore, the SuperMushroom as an item can return ItemAction methods, acquiring Location package tick.

This implementation follows the Single Responsibility principle where these classes mentioned are extended to SuperMushoom and Player, but not modifying those existing classes. Item class is an abstract class, meaning, it holds an open method (no parameters declared) for child classes to extend and implement their own unique sets of methods.

Creating a new Jump class: Inheriting the abstract class Action, which contains methods to call action to hit or increase hp – depending on item. We will need to use the parent class for returning descriptive String methods and calling the hotkey() methods. Additionally returning the parent class's methods to provide a descriptive String method, providing the Jump class's own implementation. The class has an association with the two ground classes: Wall and Tree only, as those classes will inherit the Ground class for GameMap methods. Again, similar to SuperMushroom class, the new Jump class is also implementing a single responsibility Principle by utilising an open method from the parent class, and having its own unique implementations.

REQ 3 : Enemies ☠

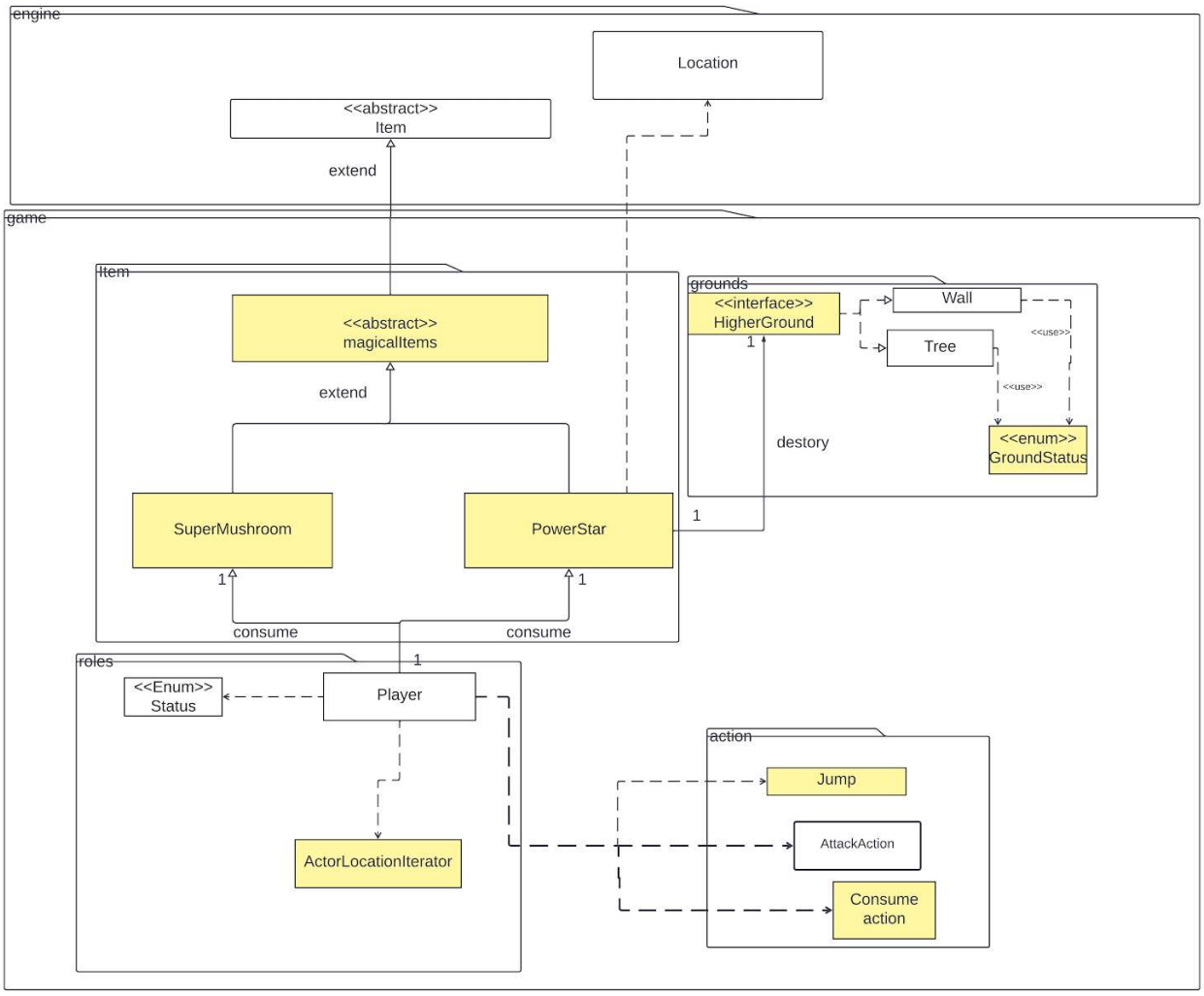


The new Wrench class will inherit the abstract WeaponItem class, which has Item capability to access the parent class of Weapon interface from the weapons package in the engine. This will allow Wrench to also inherit Item class's other packages: Action, Location, Printable and Capable. Retrieving abstract open parameter methods from WeaponItem abstract class such as damage(), verb() and chancetoHit().

We will implement a new Enemies class which is an abstract class with open constructor parameters for **Koopa and Goomba classes** to inherit implementations from. In this case, Koopa and Goomba are each a class that extends the Actor class in Actor packages. The principle of Open-Closed is implemented here, although Koopa and Goomba impact on Player directly, they will be implemented within a behaviours package interconnected with the <<abstract>> enemy class.

The <<enum>> status will be used for all roles within the game, for example, Koopa's dormant state. The <<abstract>> actor class will include methods to activate for Player to use Wrench and attack Koopa, following that, initiate the following behaviour for enemies to player. The <<interface>> behaviours consist of methods to instantiate a new enemy body with new implementation of behaviour methods, this will be re-implemented to suit the Koopa as well.

REQ 4 : Magical Items



Design Rationale:

Abstract class MagicalItems extends Item. So magicalItems are able to inherit item's constructors' value and methods.

SuperMushroom and PowerStar extends MagicalItem. Those concrete classes can use the same methods from MagicalItem and override the methods if those items have different functionalities. Moreover, if there are more magicalItem to be implemented, they can be extended to a magicalItem directly that meets the Open Closed Principle.

Player consumes SuperMushroom and PowerStar . SuperMushroom and PowerStar will change the player <<Enum>> to its particular status after the player consumes Super mushroom or powerStar.

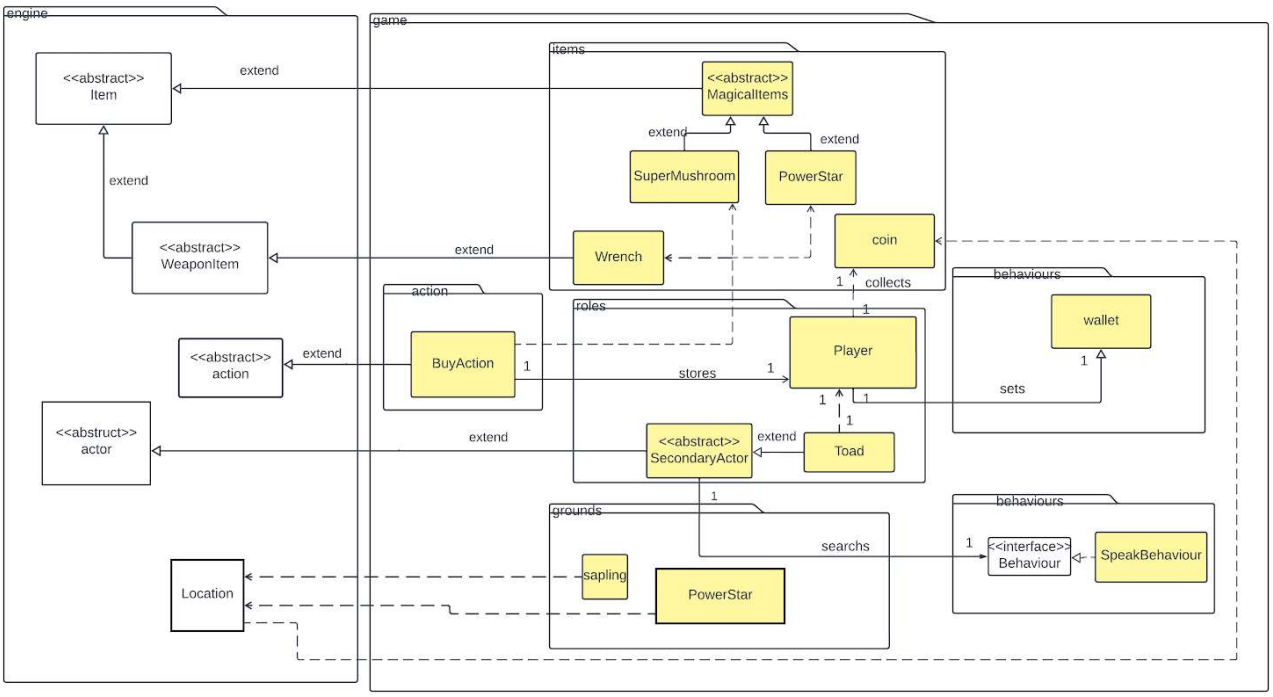
Player strengthens Jump. When the player consumes super mushroom, it can increase the success rate to 100 for the Jump class.

Player checks AttackAction and Player kills the ActorLocationIterator. When the player consumes power star, the player needs one dependency for AttackAction to check if the player has attacked the enemy successfully through the if statement. If this is a successful attack, it needs another dependency for ActorLocationIterator to remove that actor on the map.

PowerStar destroys HigherGround and sets all walls and trees Enum to be walkable . Once the player has power star status, the player can walk on walls and trees by setting the trees and walls status to be walkable. We use a HigherGround interface that satisfies the Dependency Inversion Principle. PowerStar class does not depend on the tree or wall now.

PowerStar sets Location. PowerStar resets the current higher ground to floor by getting the type from location and reset it to dirt. Moreover, PowerStar generates coins on the ground.

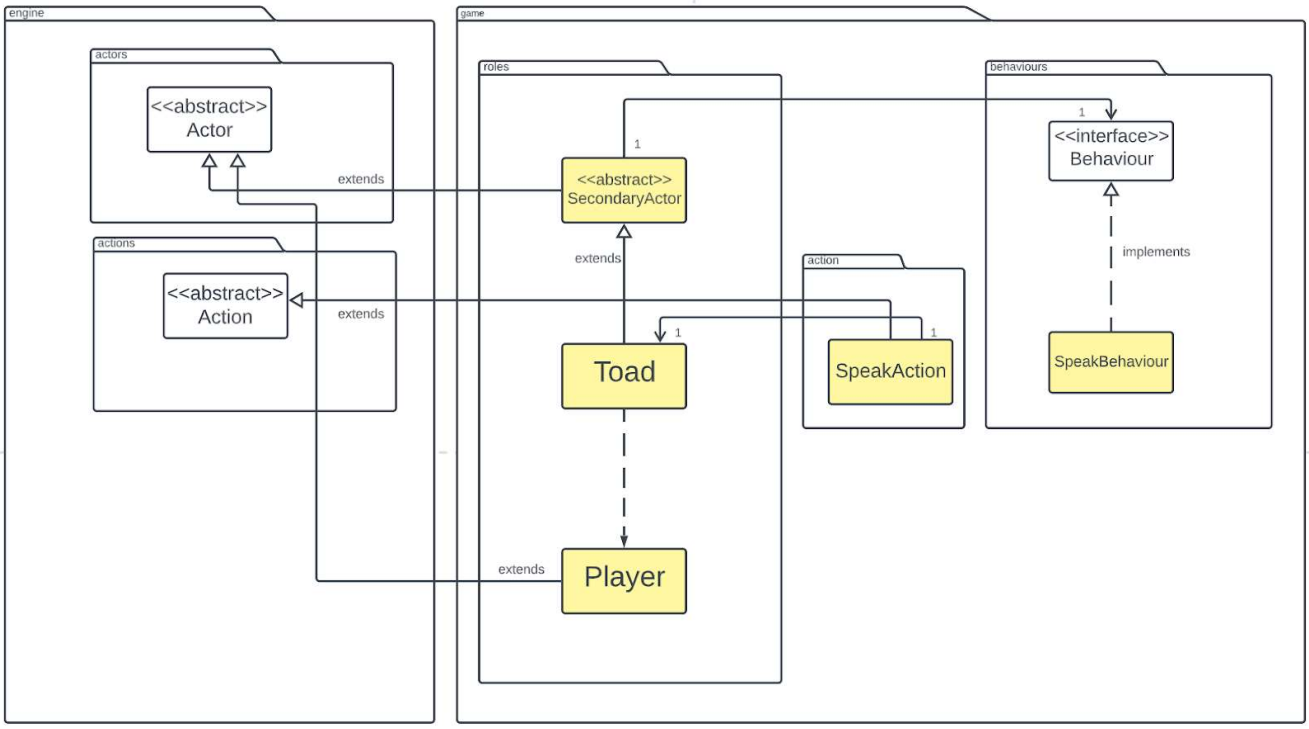
REQ 5 : Trading 💰



MagicalItems extends Item, SuperMushroom and PowerStar extends magicalItem. MagicalItems inherit Item so that MagicalItem has the methods of Item. Hence, when we add more magical items to the game, we can just extend the concrete class from MagicalItems such as SuperMushroom and PowerStar. The MagicalItems class is able to be extended without modifying itself that meets the open-closed principle.
Wrench extends WeaponItem. Wrench is a weapon. When the wrench extends the weaponItem, it can be defined as a weapon Item in the game since it inherits all the methods from weaponItem.
BuyAction extends the action and BuyAction buys Wrench, SuperMushroom, PowerStar. BuyAction inherits the action class. BuyAction interacts with those items in BuyAction's methods, therefore we have dependency between them.

BuyAction associates with Player. The items can be stored in the player's inventory. By doing so, it satisfies the Single Responsibility Principle.
SecondaryActor extends Actor, and Toad extends SecondaryActor. SecondaryActor can be extended for more actors similar to toad if it is required in the future.
SecondaryActor is associated with Interface Behaviour, so that we can implement a unique functionality to secondary actors.
Toad checks Player. This is a dependency that checks if player status is Power star or if the player has wrench already. Therefore, Toad will not say particular hints.
Sapling and PowerStar sets Location to create Coin. Coins are spawn from sapling or ruined high grounds by PowerStar, coins will have a random value.
Player collects coins and increases its balance in the wallet since the player is associated with the wallet. Also, from the BuyAction class can cost the balance of the player depending on how many items the player has consumed.

REQ 6 : Monologue



SecondaryActor extends Actor, and Toad extends SecondaryActor: Since Toad is a friendly actor, we create a subclass extending Actor class named SecondaryActor. Toad extends SecondaryActor. The abstract subclass “Secondary Actor” was created so that we can add more actors like Toad (Actors who are neither player nor enemy) in SecondaryActor in the future. This ensures that we follow the Open-Closed Principle. If we add more secondary actors in the future, SecondaryActor can have methods that are only applicable to the secondary actors.

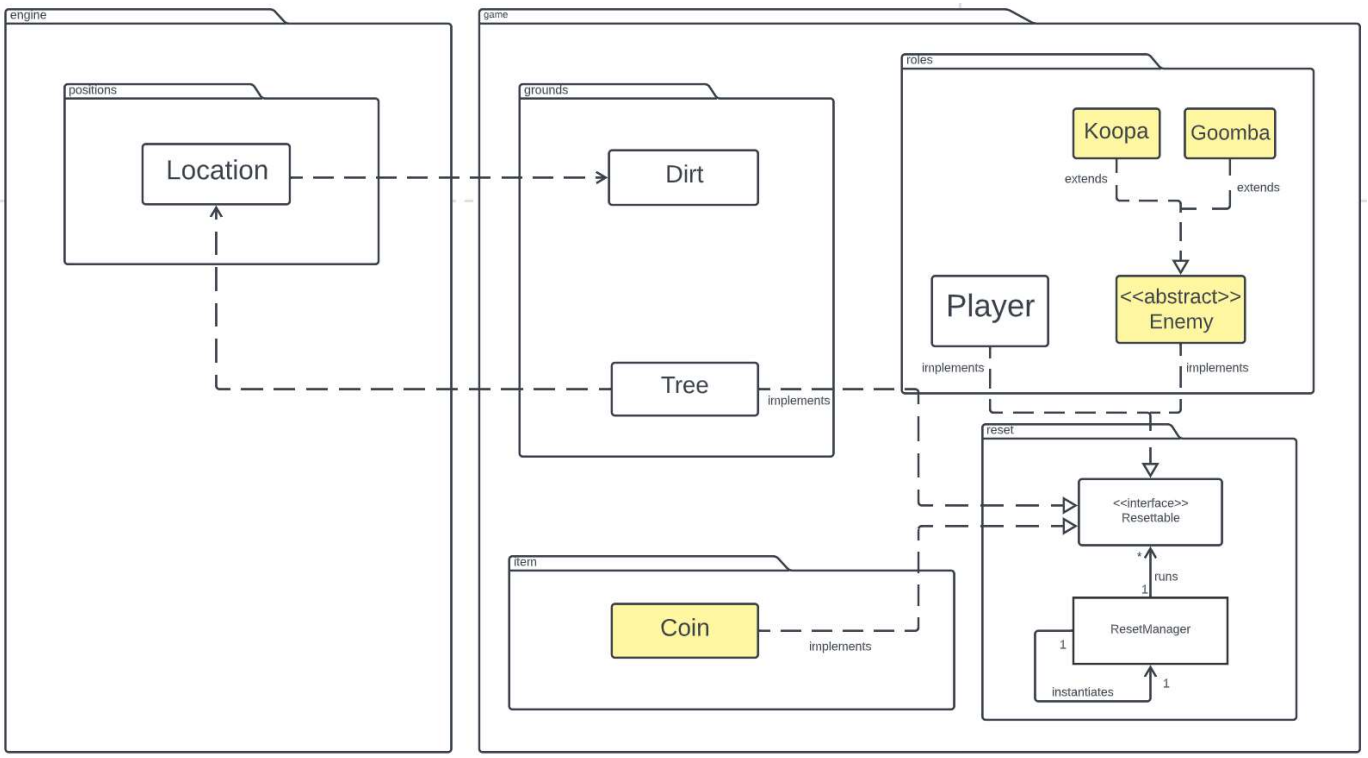
SpeakAction extends Action: Since Toad can speak, a new Action subclass has been created, called SpeakAction which stores the sentences it can say in an Arraylist and can randomly generate one of the four sentences listed in the requirement. Creating a SpeakAction class. The checks for statement(s) that can’t be said will be done using conditional checks.

SecondaryActor is associated with Interface Behaviour, SpeakBehaviour implements the interface Behaviour: A new SpeakBehaviour class has also been created which implements the Behaviour interface. The SpeakBehaviour class checks the Player’s proximity to Toad so that the Toad’s Speak Action is available. SpeakBehaviour might be necessary in the future as there might be more actors that can be put into the game who can do multiple actions like SpeakAction,AttackAction hence making the design of the code better and easier to read.

Creating separate classes for SpeakAction and SpeakBehaviour helps us to adhere to the Single Responsibility Principle.

Toad has a dependence on Player: Since the dialogues said by the Toad are dependent on whether the player has the WeaponItem Wrench and the MagicalItem PowerStar.

REQ 7 : Reset Game ⏱️



Coin, Tree, Player and Enemy class implements Resettable interface: All the requirements that need to be reset have to implement the resettable interface such as Coin (All coins need to be removed from the ground), Tree(Tree has a 50% chance of converting back to dirt),Player(Player status needs to be reset) and Enemy(All enemies need to be killed).

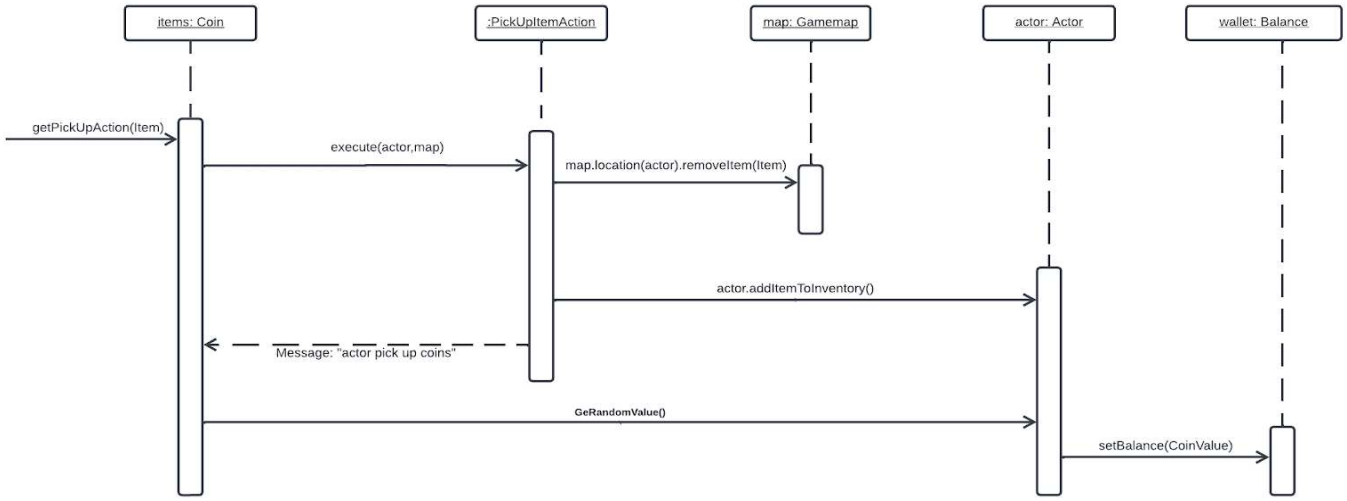
Tree has a dependence with Location and Location has an association with Dirt: Since Tree has a 50 percent chance to convert to dirt, and Tree extends Ground, setGround in Location can convert the Ground type from Tree class to Dirt class which is a dependence.

The ResetManager instantiates itself: The ResetManager class has a list that stores all the requirements that need to be reset. Since a player can reset the game only once, getInstance Method checks if the instance is null, and if true, Reset Manager instantiates itself

Sequence Diagram

Sequence Diagram 1:

Player picking up Coin :

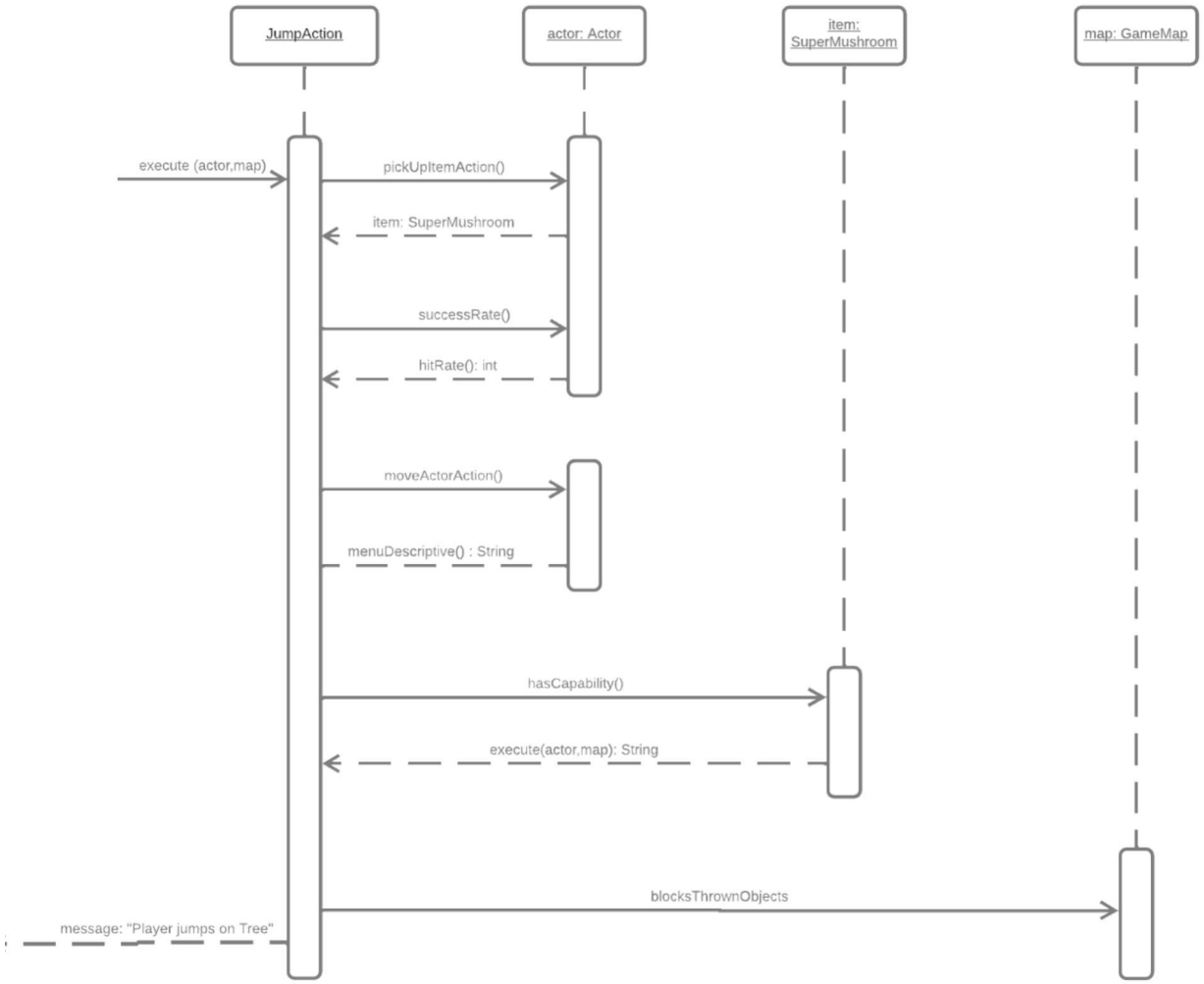


(Junhao Li)

This sequence diagram discusses the process of the actor picking up the coin and converting the coin to credit so that it can be added to the wallet.

Sequence Diagram 2:

Player Jumps on Tree with SuperMushroom :



(29693500 Kenda Wan)

This sequence diagram depicts the step-by-step sequence of execution for when Player consumes a SuperMushroom and activates jumpAction to execute. The target ground object is a Tree.

Work Agreement Breakdown

The work agreement breakdown is located in our shared GIT folder in the docs folder. Here is a view of the committed WBA.

```
WORK BREAKDOWN AGREEMENT

# Task allocation:
The process for allocating tasks was undertaken during the group meeting.

# UML diagrams
Discussed and completed together

# Rationale
Question 1: Ashton Sequeira
Question 2-3 Kenda Wan
Question 4-5 Junhao Li
Question 6-7 Ashton Sequeira

# Sequence diagram
Junhao Li, Kenda Wan

@Communication:
We created a groupchat on FaceBook, we can discuss and update the problems and solutions among our team.

Position assign:
Junhao Li: Diagram developer, Diagram reviewer
Kenda Wan: Diagram developer, Diagram reviewer
Ashton Sequeira: Diagram developer, Diagram reviewer

@Diagram developer:
Designs diagram to satisfy features and stories from the tasks. They are
responsible for their designed diagram that meets the requirements of the tasks appropriately.
They need to provide the reasons why they use Interface, Abstract, Class, and Enum under different
situations. Attends group meetings in any forms to update the progress to team.

@Diagram reviewer:
Reviews and checks the diagram that is designed for quality assurance. The
diagram reviewer needs to review diagram designed by other diagram developer. Hence, a new
perspective will be likely discovered so that the diagram can be improved.
Currently, assigned to: Everyone (Each person responses to different questions)

By signing below, we agree to the WBA stated above:
Junhao Li
Kenda Wan
Ashton Sequeira
```

(This page was intentionally left blank)