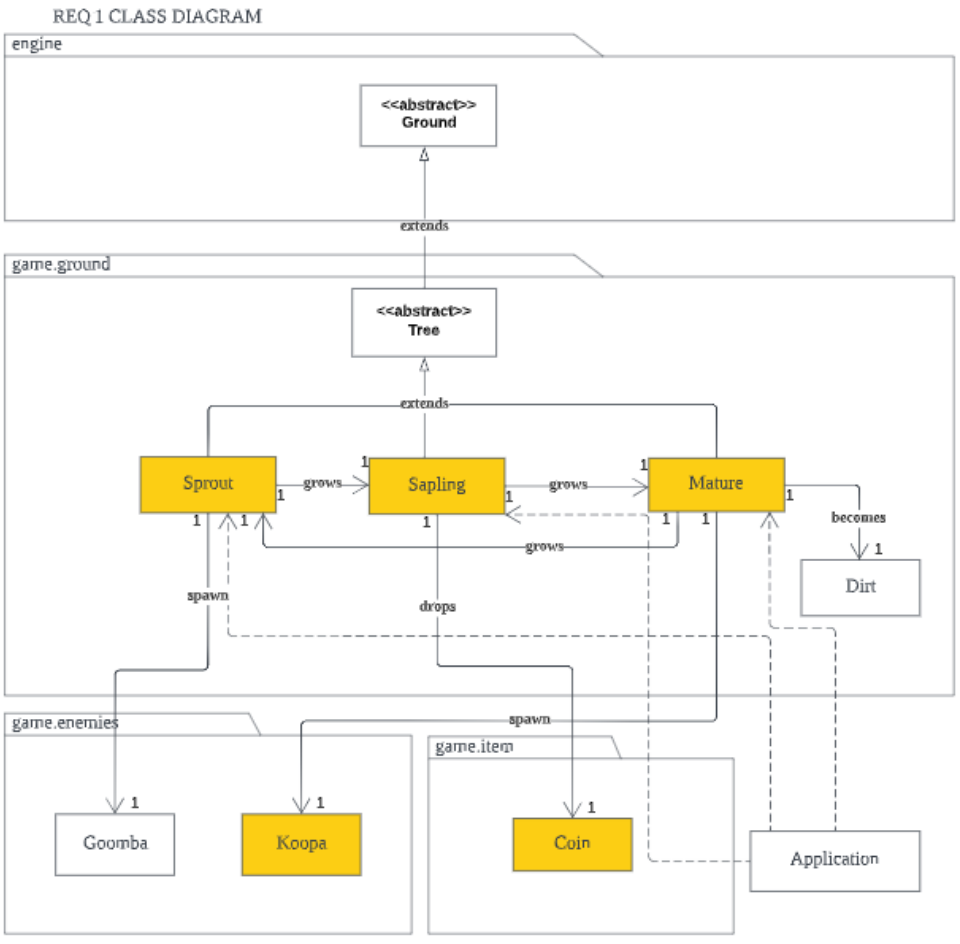
Design Rationale For Assignment 1

LAB 12 Team 4

Seng Wei Han (32229070)

Chua Jun Jie (32022425)

REQ1: Let it grow! 🌳



(The screenshot shown above is the UML designed for requirement 1)

Design Implementation

Tree Class :

1) The purpose of making the Tree class to be an abstract class is because we knew that we will not instantiate the Tree class in any of our code. Such implementation helped us to achieve abstraction.

2) From requirement 1 of the assignment,it is stated that a Tree has three stages and each stage has a unique spawning ability. On the other hand, each spawning stage will also have additional characteristics, for instance each sprout has 10% chance to spawn Goomba.Therefore, instead of doing all the spawning in one Tree class which will cause the Tree class to be so called the God class, I have decided to split the classes into 3 small new classes namely Sprout , Sapling and Mature and make it inherit the Tree class.In each of these new subclasses, it will have their own responsibility so that this design will adhere to the Single Responsibility Principle (SRP). On the other hand, I have also planned to declare an abstract method called tick in Tree class so that any subclasses that inherits the Tree class will need to implement the tick method. By doing this way, we can prevent too many if-else statements in one Tree class so that it will not violate the second principle which is the Open-closed Principle.

3) Since sprout, sapling and mature are the 3 stages that a tree can experience, meaning to say the 3 classes will have the same characteristics for instance, having the same method implementation for the following method :

```
@Override
public boolean canActorEnter(Actor actor)
```

```
@Override
public ActionList allowableActions(Actor actor, Location location, String direction)
```

Thus, it's a better idea to implement those method in the Tree class and allow the other 3 new classes(Sprout,Sapling and Mature) to inherit the method from their parent class. By using this implementation, we can ensure that DRY(Don't Repeat Yourself) Principle will not be violated.

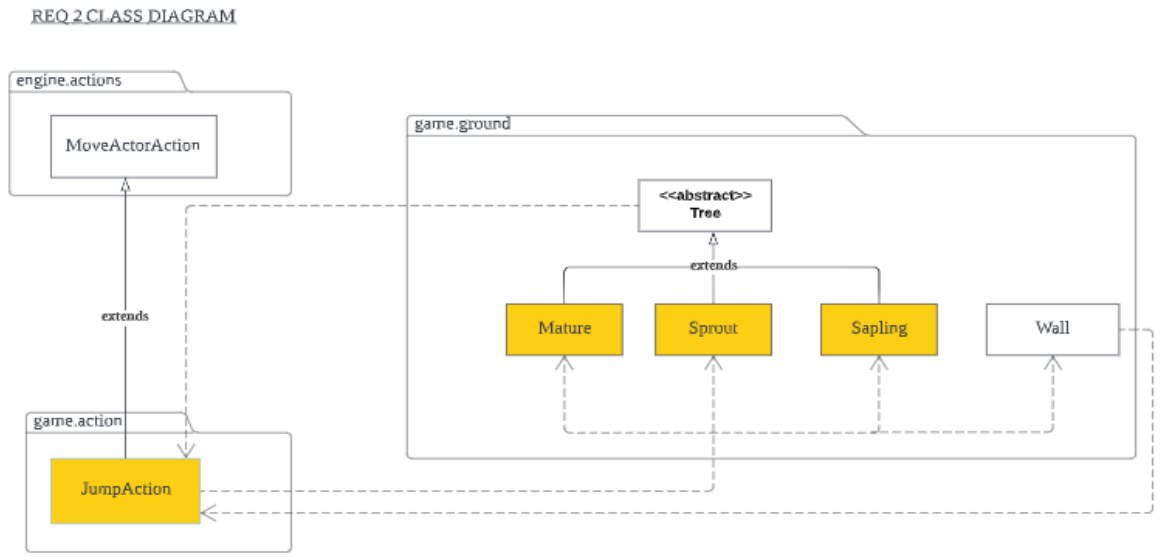
Relationship :

1) The dependency relationship between Application and the three subclasses of Tree is due to the fact that for each new ground that we created, we will need to add into the FancyGroundFactory constructor.

2) The association relationship between Sprout and Goomba , Sprout and Sapling is due to the fact that each sprout can spawn goomba, so we will need to have Goomba as an attribute in the Sprout class and the same goes for Sprout and Sapling.

3) The association relationship between Sapling and Coin , Sapling and Mature is because the requirement states that a sapling can have a chance to drop a coin and a sapling will grow into a mature tree, therefore we will need to have those attributes in the sapling class. Since coins are needed, we will need to create a new class for coins.

REQ2: Jump Up, Super Star! 🌟



(The screenshot shown above is the UML designed for requirement 2)

Design implementation

Jump Action :

1) Since requirement 2 is about jumping features, then we definitely need to have a JumpAction class implementation.This JumpAction class will inherit (extends) the moveActorAction. The ground inside the UML class diagram is basically the ground that does not allow player to enter without using the jump action . Therefore, for each ground that player walked through, we will need to do a checking inside the execute method .The checking needs to be done due to the fact that certain ground does have certain success rate which means if the player failed to jump to that particular ground, it will get inflicted a certain amount of damage by that ground. We need to set an allowable action to the ground that allows the player to use the jumping action. Since Sapling, Sprout and Mature are inheriting the Tree class, we can just set the

allowable actions in Tree class and also for Wall class. This design implementation does satisfy the requirement for the Single Responsibility Principle (SRP) since this class is only responsible for the jumping action of a player . As for the case of the SuperMushroom, we can just check whether a player has the status when it consumes super mushroom, if it does then the player will have a 100% success rate of jumping through those grounds.

Relationship :

1) The dependency relationship between Tree and JumpAction, Wall and JumpAction is because we need to add a new instance of the JumpAction to the actionList inside the allowableActions method.

2) The dependency relationship between JumpAction class and the 4 other ground is due to the checking process inside the execute method where we need to check whether the ground in the player exit surrounding is one of that four ground , if it is, then it will trigger the jump action in the console to allow the player to jump through that ground.