S1 2022

FIT2099 Object-Oriented Design and Implementation

Assignment 2: Develop

Lab_14Team6:

30041880 Junhao Li
31950124 Ashton Sequeira
29693500 Kenda Wan (Special Consideration)(Will Submit Later along with Team Task)

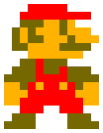
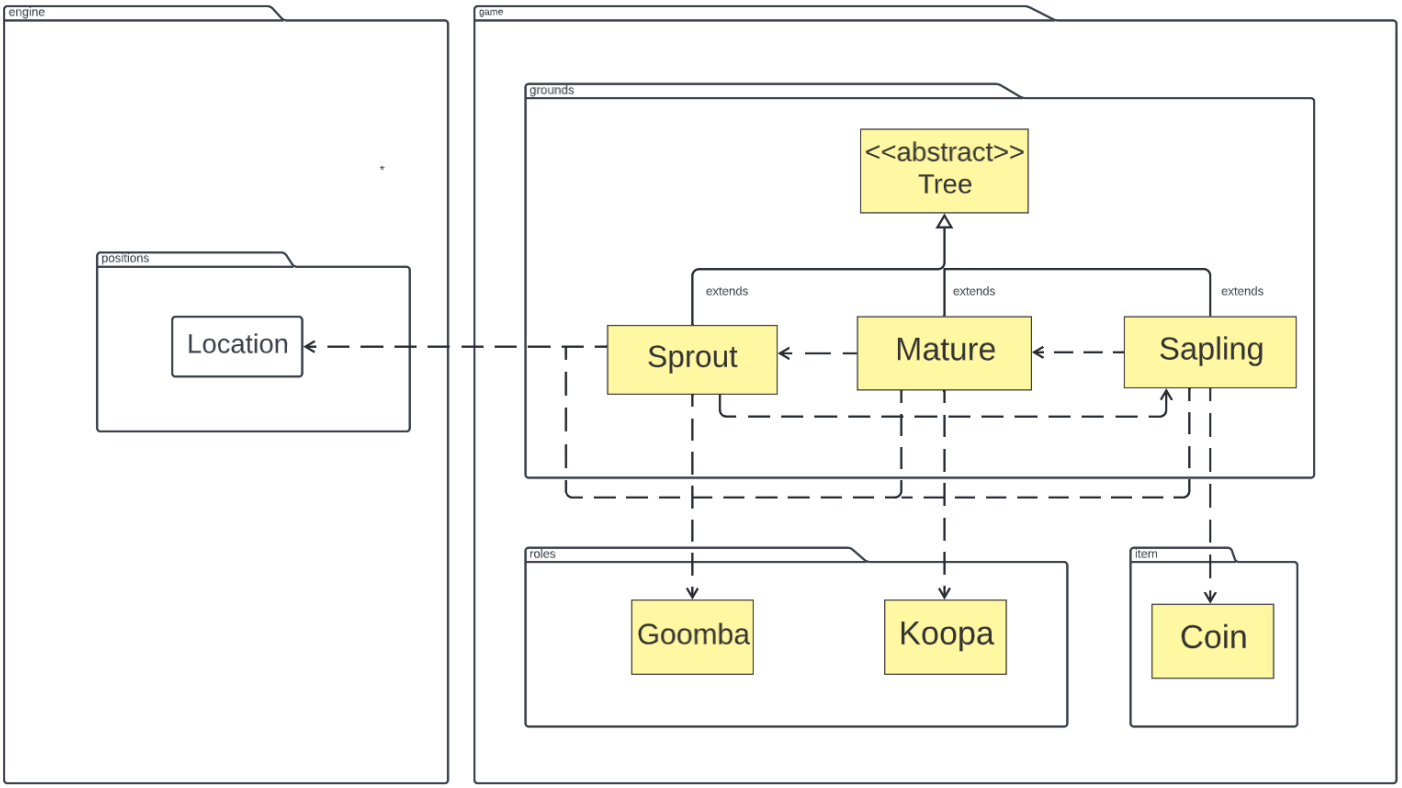


Table of Contents

UML and Design Rationale

REQ 1 : Let it Grow !🌳



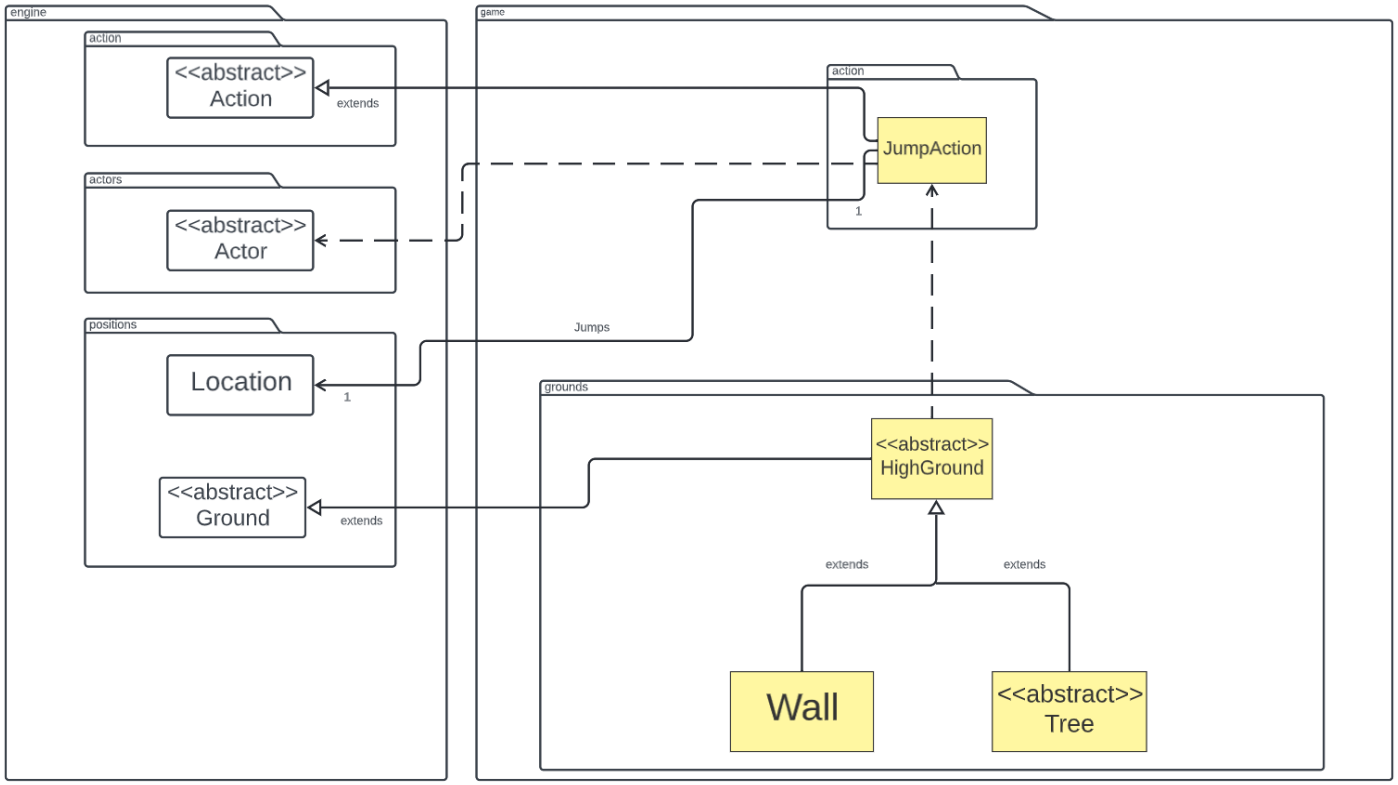
Tree class is made abstract and Sprout, Sapling and Mature extend the Tree class: While abstract Tree class extends abstract HighGround class which extends abstract Ground Class.Sprout,Sapling and Mature extend the abstract Tree Class.These classes extend the Tree class as we can implement and override methods and each subclass can have it's own properties without violating the Liskov Substitution Principle. This will help us follow the Single Responsibility Principle as we are not overburdening the Tree class with all the methods.

Sprout has dependence on Sapling and Goomba,Sapling has dependence on Mature and Coin, and Mature has dependence on Sprout and Koopa:.The Sprout can grow to Sapling and the Sapling can grow to Mature and Mature can grow Sprout around when the area around is fertile.Sprout can spawn Goomba,Sapling can drop a Coin and Mature can spawn Koopa. This can be done by overriding the tick methods in Sprout,Sapling and Mature

Sprout,Sapling and Mature has a dependence with Location.

The older implementation of implementing actor with ActorLocationIterator method did not work during execution and therefore we used methods from the Location for better implementation.Improved the UML for task one based on the interviewer's suggestion and added more detailed dependencies

REQ 2 : Jump Up, SuperStar! 🌟



Abstract HigherGround class created that extends Ground. Wall and Tree extends HighGround: Since you can Jump only on HighGround class has been created. This will make it easier to design and implement Jump as we have to override methods only in HighGround class and also helps in better design as we dont have to repeatedly override methods in Wall and Tree class. It also follows dependency inversion priciple as there are no dependencies with any of the concrete classes with each other but instead relies on abstraction. It adheres to the Liskov Substitution Principle as grounds which come under subclasses of HighGrounds will be the only grounds that allows the player to jump.

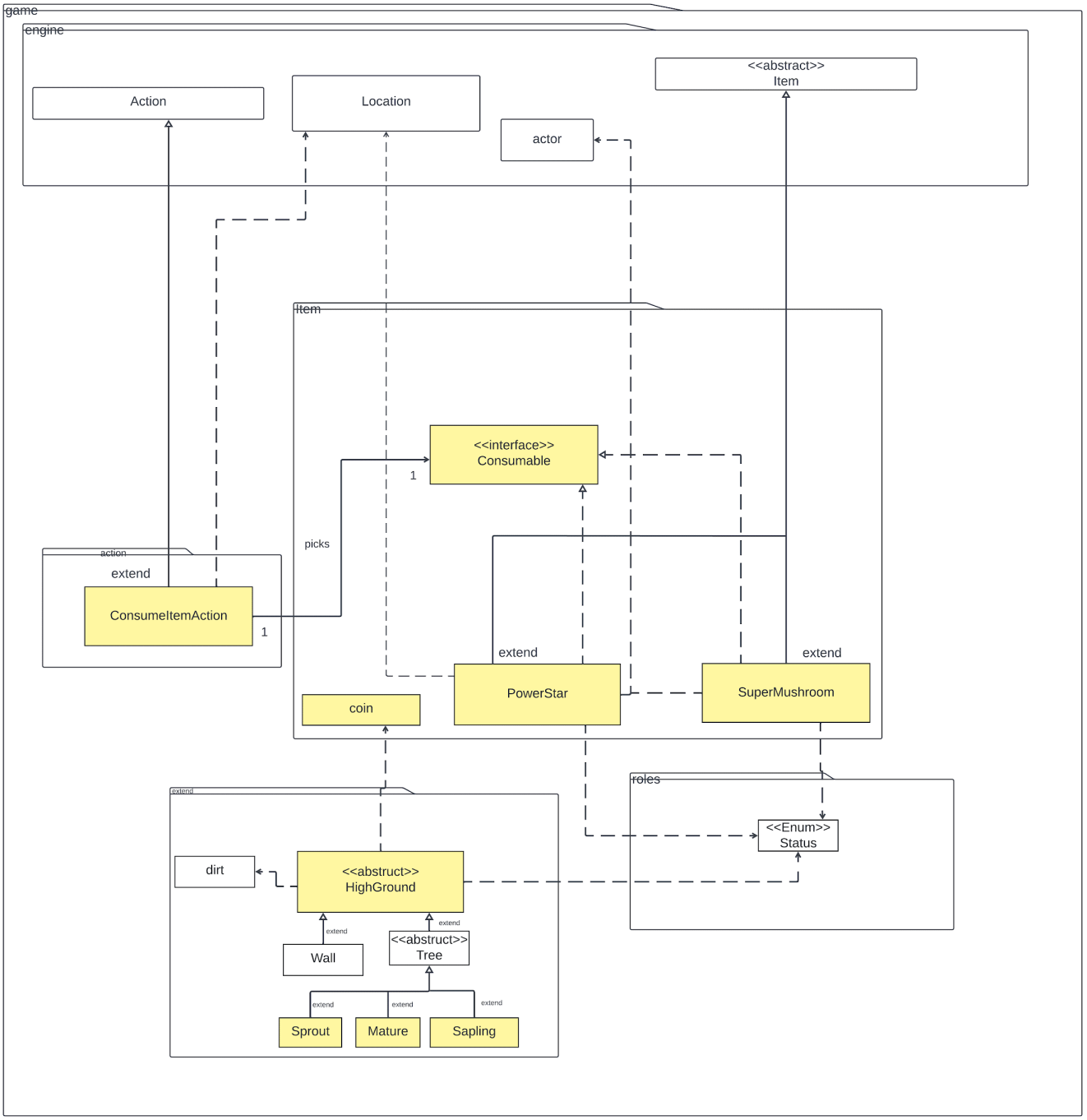
Creating a JumpAction class that extends Action class: We override the execute method and check for the type of High Ground and implement conditions for the success rate of the jumps.

JumpAction has an association with Location and a dependence with Actor.

In the newer implementation, we used HighGround for JumpAction instead of linking Wall and Tree to it as it would be a better design decision as explained above. The check for SuperMushroom is done using an enum so it is not necessary to show it in the UML diagram. JumpAction has a dependence with Actor instead of Player due to easier code implementation. But canActorEnter return false so that actors can enter it without a JumpAction

FIT2099 Assignment 2: Develop

REQ 4 : Magical Items 🍄



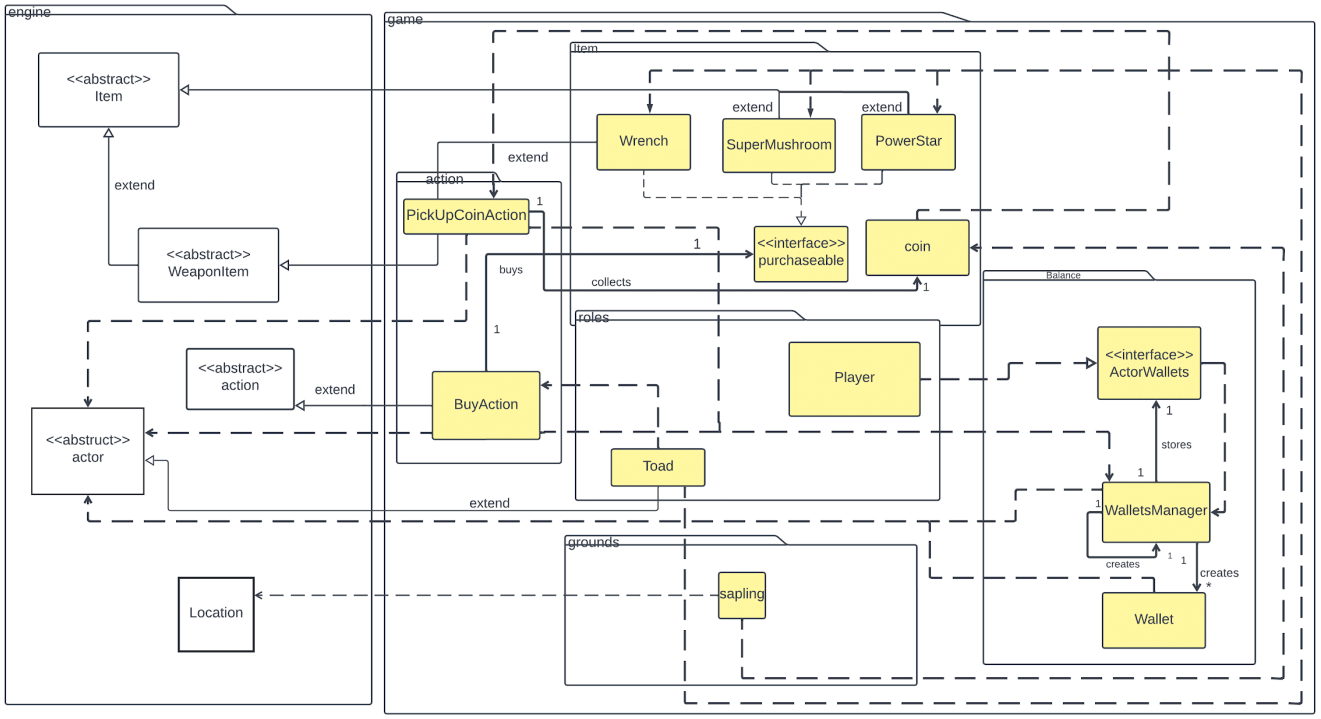
Design Rationale:

SuperMushroom and PowerStar implement Consumable. Those concrete classes can inherit the same methods from Consumable and override the methods if those items have different functionalities. Moreover, if there are more Consumable items to be implemented, they can be implemented directly that meets the Open Closed Principle. Since powerstar and super mushroom have implemented Consumable interface, therefore they can be downcasted from Item to “Consumable” item. Hence, ConsumableItemAction can recognise which items can be consumed by the actor. ConsumableItemAction will check if the inventory has the item, yet not it will generate a new item in the inventory.

Actor consumes SuperMushroom and PowerStar. SuperMushroom and PowerStar will change the actor's <<Enum>> status to its particular status after the actor consumes Super mushroom or powerStar.

walls and trees extend HigherGround. Once the actor has power star status, the player can walk on walls and trees by setting the HigherGround's canActorEnter to be true. We use a HigherGround abstract class that satisfies the LISKOVE substitution principle(LSP). So that walls and trees will have the functionality of the high ground that generates the coin after the actor breaks the high ground. Therefore, the same methods can only be implemented in the HigherGround once.

REQ 5 : Trading 💰



SuperMushroom and PowerStar extends Item. SuperMushroom and PowerStar inherit Item so that SuperMushroom and PowerStar have the methods of Item. Hence, when we add more items to the game, we can just extend the concrete class from Items such as SuperMushroom and PowerStar. The Items 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.

SuperMushroom, PowerStar, and Wrench implements purchasable that allows BuyAction to buy them. Those items can be purchased by implementing the purchasable. According to the Open-close principle, it is able to extend for more items to be purchasable. Since BuyAction can implement those items that have implemented purchasable items directly. BuyAction meets the SRP principle so that one class has one function.

BuyAction extends the action. BuyAction inherits the action class. BuyAction interacts with those items in Buy Actions methods, therefore we have dependency for Toad.

Toad extends Actor. Referring to LSP, Toad inherits everything from the actor. The toad has dependency between purchasable items. There are dependencies for Power Star, Wrench, super mushroom. Since, from the AllowableAction of each item that will check if the actor is near toad. Then return the price of items to the player.

Sapling creates Coins around sapling. Coins are spawned by sapling, coins will generate 5 dollars.

PickUpCoinAction collects coins and increases its balance in the Wallet from the WalletManager. WalletManager is using the static factory method that would store the actors and wallets in the hashmap. Since there is an ActorWallet interface. Implementing this to the player, it can add a method from the player's constructor. So that Player can be included in the WalletsManager. Therefore, it satisfies the open-closed

principle. If more actors need to have a wallet as an extension, they can just implement this interface. Then WalletsManager can store their particular wallet in the hashmap.

Work Agreement Breakdown

The work agreement breakdown is located in our shared GIT folder in the docs folder.

```
WORK BREAKDOWN AGREEMENT

Task allocation: The process for allocating tasks was undertaken during the group meeting.
Requirements 1-7:
JavaDoc:
Q1~Q2 Ashton Sequeira
Q3: Kenda Wan (Special Consideration:12th May 11:55 pm)
Q4~Q5 Junhao Li
Q6: Team Task(Ashton Sequeira,Kenda Wan,Junhao Li)(Special Consideration:12th May 11:55 pm)
Q7: Kenda Wan (Special Consideration:12th May 11:55 pm)
Kenda responses to generate the Javadoc once everything has been completed.

#UML and Rationale
Q1~Q2 Ashton Sequeira
Q3:Kenda Wan (Special Consideration:12th May 11:55 pm)
Q4~Q5 Junhao Li
Q6:Team Task(Ashton Sequeira,Kenda Wan, Junhao Li) (Special Consideration:12th May 11:55 pm)
Q7:Kenda Wan (Special Consideration:12th May 11:55 pm)

#Code
Ashton Sequeira: Q1~Q2 abstract Tree,Sprout,Sapling,Mature,abstract HighGround,JumpAction
Junhao Li: Q4~Q5 Player,SuperMushroom, PowerStar,Wrench, Coin, BuyAction, Building Wallet
System, Consumable.
Kenda: Q3,Q7 (Special Consideration:12th May 11:55 pm)
Team Task: Q6 (Special Consideration:12th May 11:55 pm)

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)