Assignment 1A - UML & Sequence Diagrams
MA Lab11 Team 1

Group Members:

1. How Yu Chern
2. Eugene Kah Chun Fan

Work Breakdown Agreement (WBA)

Version 2
Last Updated: 6/4/2022

Group Members Signature:
1) How Yu Chern - I accept this WBA
2) Eugene Kah Chun Fan - I accept this WBA

Task Breakdown and Summary:

- Each person in charge of a requirement (REQ) will be in charge of it's respective UML Diagram and Design Rational components. Each member's components will be combined to create the complete version of the UML Diagram, and Design Rational for all 6 Requirements. This will be done in each member's own time until the deadline. Each person's work will be reviewed by other members of the group, to ensure that it is complete and up to date.
- Each person will pick one interesting feature in one of the 6 requirements, and will create a sequence diagram out of it.
- For Group Tasks, members will meet up and collaborate, working together to complete the groups tasks of the requirements.
- This WBA, and all 6 Requirement's Diagrams and Design Rational will combined into a single PDF for submission.

Tasks for How Yu Chern
Review By: Eugene Fan
Deadline: 9/4/2022

Tasks:
REQ1: Add Trees to Terrain
REQ2: Jump Ability

Tasks for Eugene Fan
Review By: How Yu Chern
Deadline: 9/4/2022

Tasks:
REQ3: Enemies
REQ4: Magical Items

Group Tasks

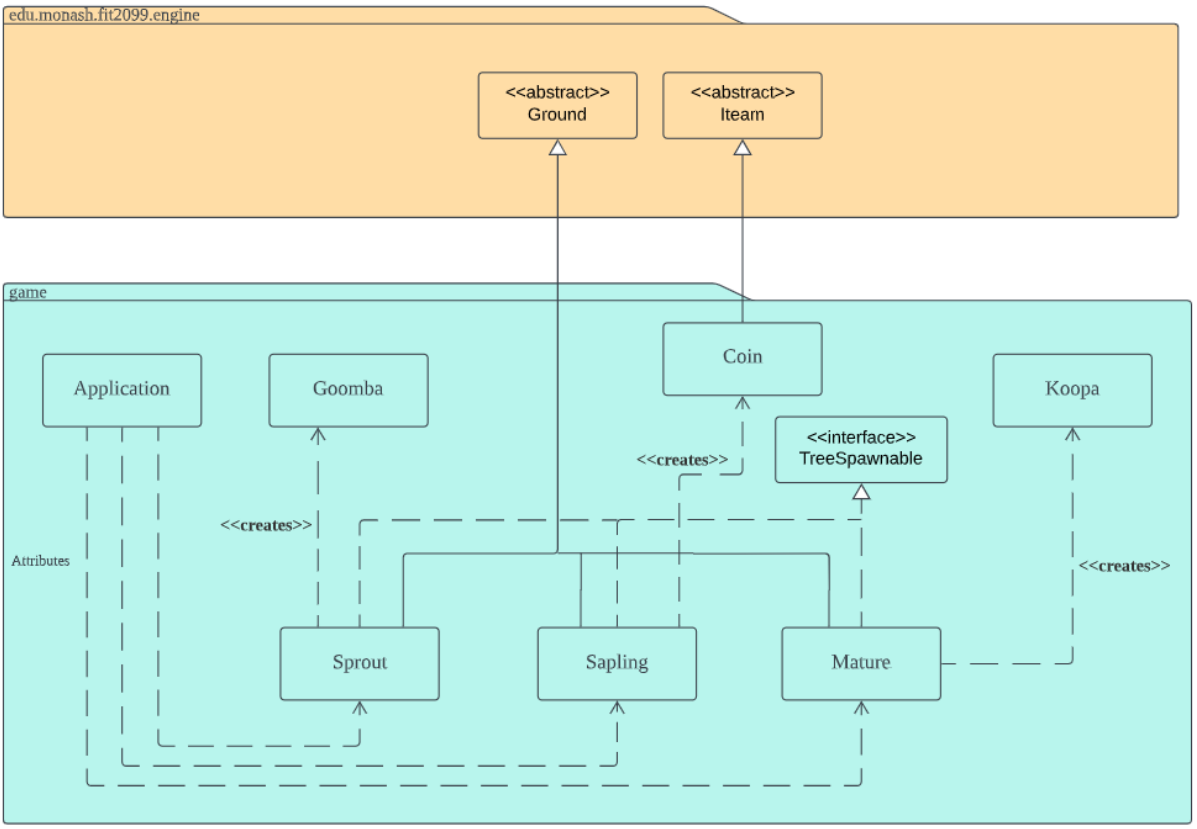
- These tasks will be done and will be reviewed by both group member.

MA_Lab11Team1

- Deadline: 10/4/2022

Tasks:
REQ5: Trading
REQ7: Reset Game

UML Diagrams
Req 1: Let It Grow



Design Rational:
The relationship in between the three stages of trees and other entities are:
1 Dirt –<<creates>> -> 1 Sprout
1 Sprout –<<creates>> -> M Goombas
1 Sprout –<<creates>> -> 1 Sapling
1 Sapling –<<creates> -> M Coins
1 Sapling –<<creates>> -> 1 Mature
1 Mature –<<creates>> -> M Koopas
1 Mature –<<creates>> -> M Sprouts
1 Mature –<<creates>> -> 1 Dirt

Each type of tree keeps track of it's own tick. It can get it's location from it's superclass (ground) each tick. Once the tree reaches it's tick limit, it would change it's displayChar. GroundFactory should detect it and update the tree appropriately.

Even though Sprout, Sapling and Mature count as Trees, making Tree class an abstract class and extending it would create multiple layer inheritance, which violates a Java OO principle.

GroundFactory and FancyGroundFactory serve as a lookup to identify each ground type (to ensure no two different grounds have the same char).

GameMap is in charge of the locations of every terrain and actor(player and enemies) on the World. GameMap uses FancyGroundFactory

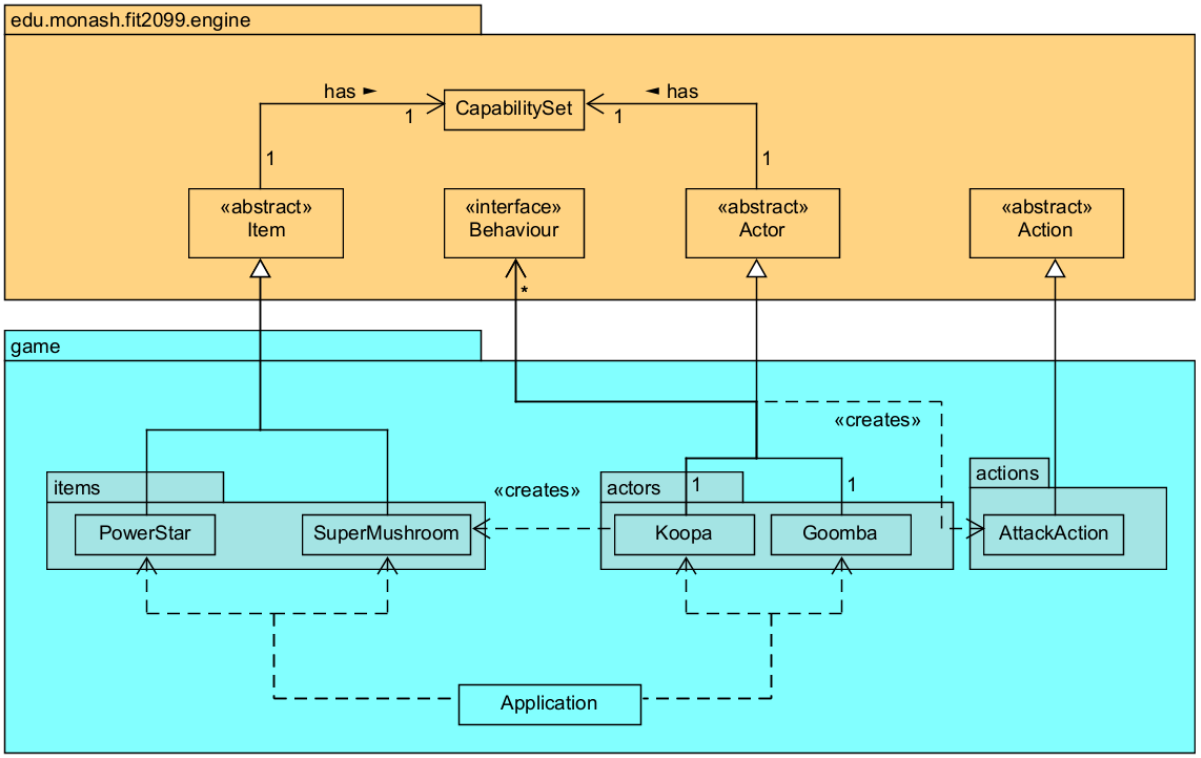
Req 2:

Notes:

- Grounds create a JumpAction for the player to choose.
- If player chooses to jump onto the pacific ground, it will call canActorEnter to check if the actor can enter it. The Jumpable interface will specify requirement behaviour of each Ground type for the player to enter, which each ground type will manage it's requirements.
- If player meets requirements, player can jump. Or if player is "M", which the interface will also allow the player to jump onto the high ground.
- Reduces dependency of having 4 grounds call jumpaction when a player is near.
- Interface allows each ground to have common, but dynamic behaviour

MA_Lab11Team1

Req 3 & 4:



MA_Lab11Team1

Design Rationale by Eugene:
***I have combined Req 3 & Req 4 ‘s UML together as the UML created is not too difficult to understand together.

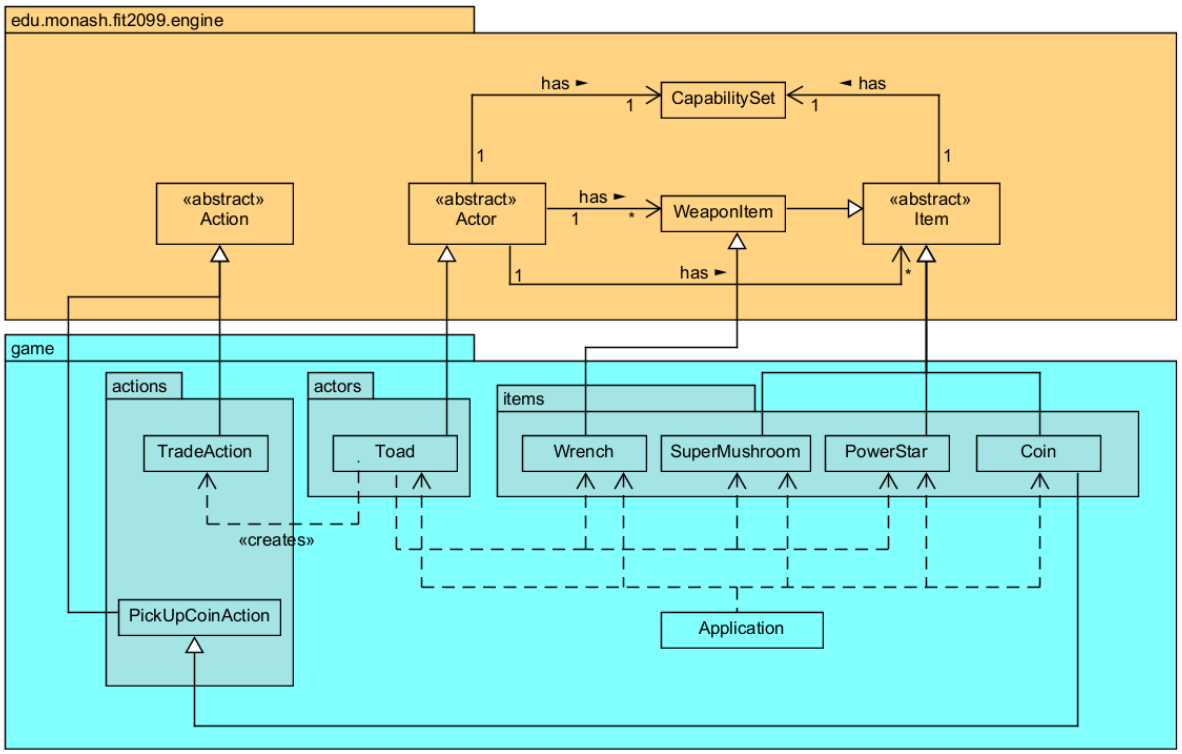
-Req 3-

1. I’ve added Koopa which also inherits the actor class (includes actor’s methods and attributes). Both the enemies(Koopa & Goomba) create an attack action which allows the other_actor to attack the enemies given that the other actor has HOSTILE_TO_ENEMY capabilities (each actor has a capability set).
2. The enemies both have the ability to attack player too as they both have HOSTILE_TO_ENEMY capabilities. To have that ability to attack, the enemies each have a map of Behaviour objects which is basically what (console is to the user). Behaviours such as WanderBehaviour, Follow Behaviour and AttackBehaviour all help shape what enemies can do.
3. Koopa will also create a SuperMushroom and put it in their backpack. Once Koopa is killed, the SuperMushroom will be dropped for the player to pick up.

-Req 4-

1. I created two new classes (magical items) called SuperMushroom and PowerStar which both inherit the Item abstract class.
2. Since both magical items are Item objects, they also have their own capability set. This means when player obtains these magical items, the player will also inherit these capabilities

Req 5:



Design Rationale:

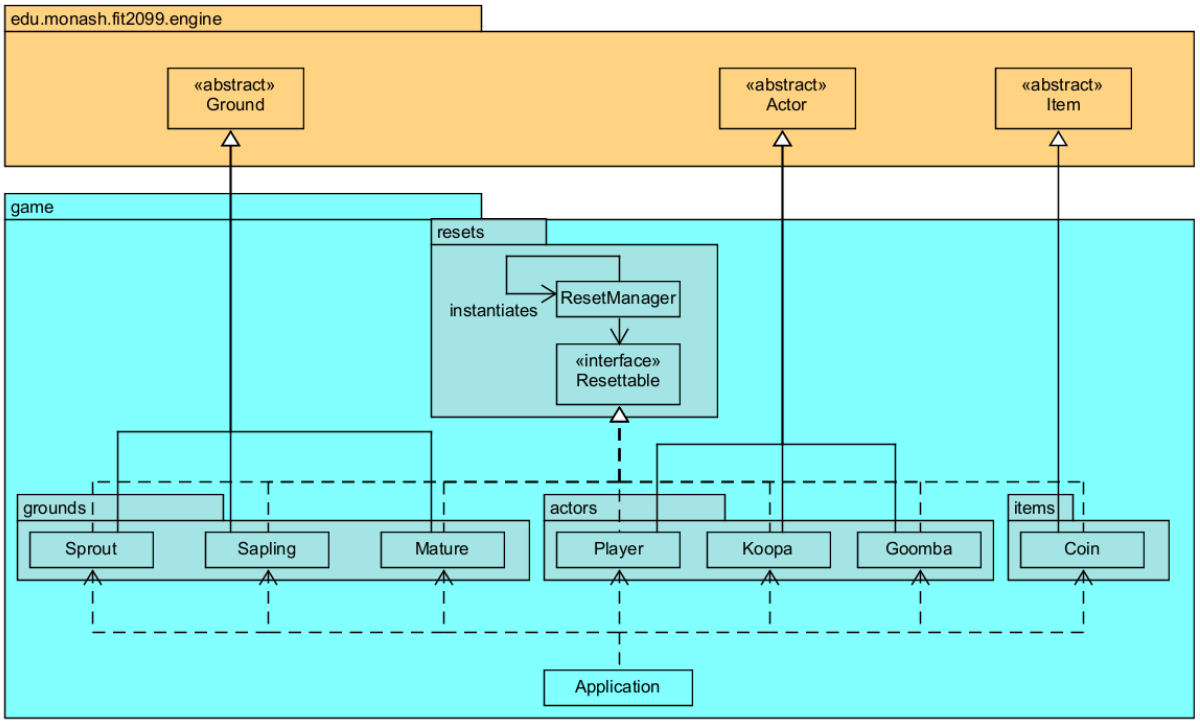
Coin extends Item, as it can exist on the terrain similar to other items such as powerstar and mushroom. Each coin will keep track of its own integer value. Coin creates a pick

1. I've created two new classes, one is Wrench class which extends WeaponItem class, and the other one is Toad class which extends abstract Actor class
2. The Wrench is a weapon item which means it can be used by the user to attack enemies. The Wrench also inherits a very important capability which is the ability to attack and kill Koopas which are in Dormant state.
3. The Toad class is a friendly actor as it cannot attack anyone, however it does have the capabilities to initiate trading of items with the player.
4. The items that are sold by the Toad class are SuperMushroom objects, PowerStar objects and Wrench objects with their respective prices.

Design Rationale by Yu Chern

1. Player has its own wallet, separate from inventory, to keep track of the balance of coins the player has picked up.
2. When a player stands on the same location as the coin, the player "picks up" the coin, and gets the integer value of the coin to add to the player's wallet.
3. The coin is destroyed.
4. This follows the one of The 3 OOP Java Principle - Classes should be responsible for their own properties.

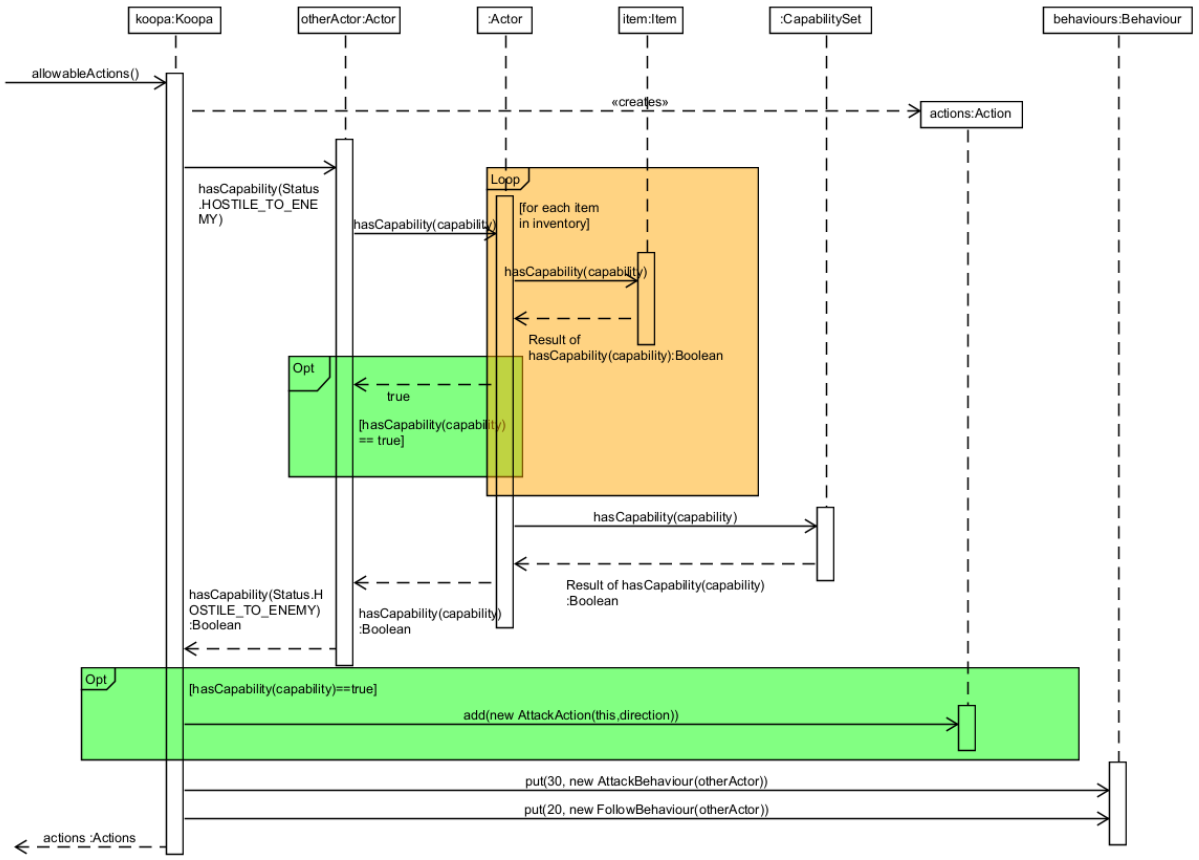
<u>Req 7:</u>



Design Rationale:
All classes of all grounds, actors, and the item coin which need to be reset implements the interface Resettable. This allows Resettable to keep

Sequence Diagrams

1. Koopa's allowableActions()



Design Rationale:

1. First, Koopa creates a list called actions which contains Action class objects.
2. Next, Koopa will check whether the otherActor has the capability of HOSTILE_TO_ENEMY. The otherActor will first check whether any items in his inventory has the capability of HOSTILE_TO_ENEMY, if so, return true. If not, otherActor will check his own capabilitySet on whether he himself has the capability of HOSTILE_TO_ENEMY.
3. If by the end of the checking phase, otherActor have that capability, Koopa will then create a new AttackAction which allows otherActor to attack Koopa.
4. As Koopa gives itself a target for otherActor to attack, Koopa will also add AttackBehaviour and FollowBehaviour into its list of Behaviours. AttackBehaviour will be added with a key of 30 as it is the highest preference. FollowBehaviour will be added with a key of 20 as it is the second highest preference. This means if Koopa has the chance to attack, it will attack otherAttack. If it doesn't attack, it will follow otherActor instead.

2. Toad's tradingAction()