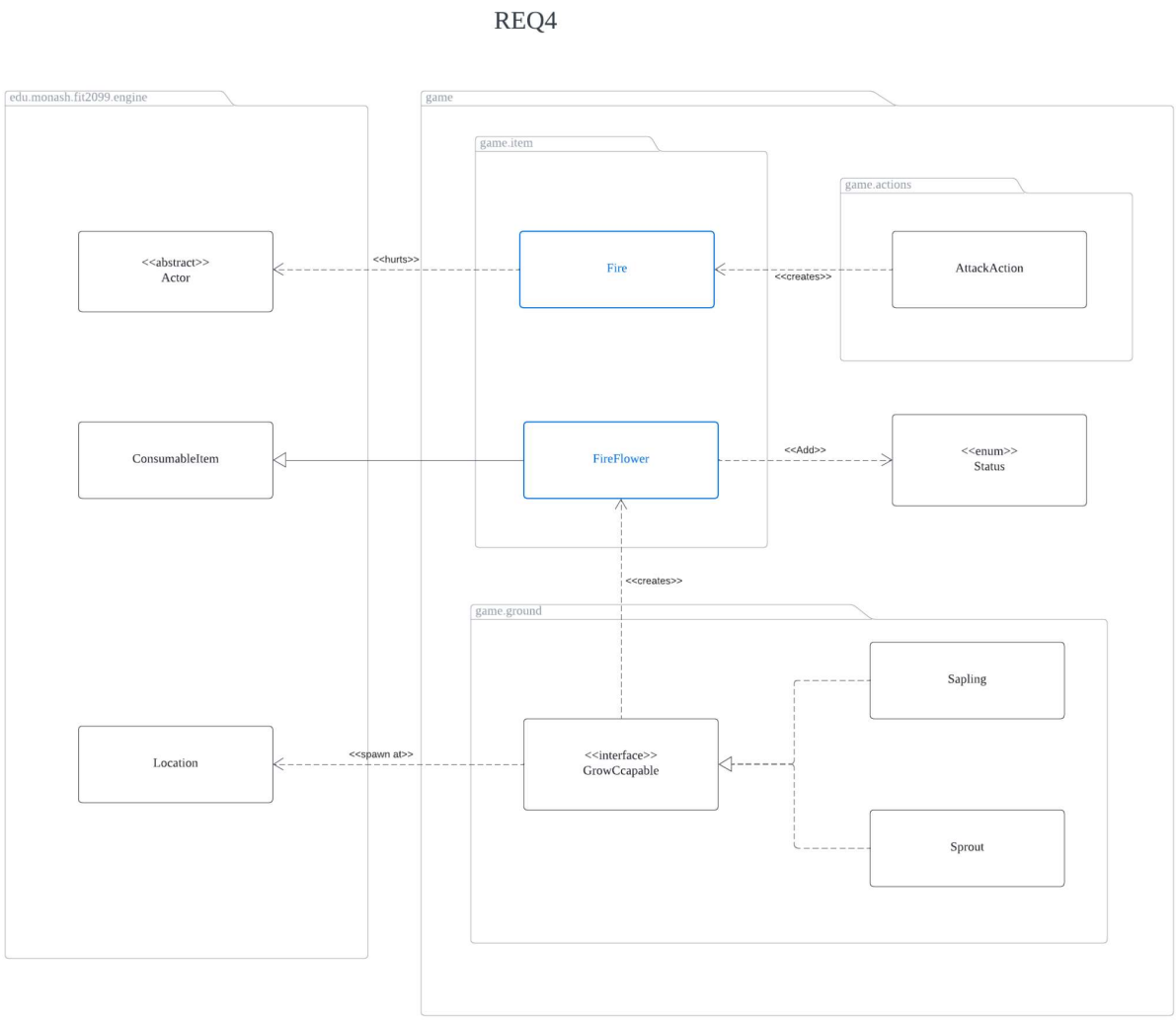
REQ4

Class Diagram:



Requirement:

- 50% chance to spawn/grow Fire Flower during the growing stage of the tree (Sprout -> Sapling and Sapling -> Mature)
- Mario can consume the fire flower and gain Fire Attack
- Multiple Fire flowers in one location
- Actor can consume the Fire Flower and use fire attack on the enemy
- After consuming Fire flowers, attacking will drop a fire at the target's ground
- "Fire attack" effect will last for 20 turns
- Fire will stay on the ground for 3 turns

- Fire deals 20 damage per turn

Design Rationale:

In order to fulfil part 1 of REQ4, we have created a new class called **FireFlower.** FireFlower class is a subclass of **ConsumableItem.** Due to the earlier implementation of **ConsumableItem** (making it as an abstract class), now we can easily implement the FireFlower class as a new item that can be consumed. Firstly, we override the consumeItem() method in **FireFlower** class to implement what should be done when the actor consumes the fire flower. Below is the code implementation and explanation,

1. Firstly, we will get the current location of the actor by using `Location currentLocation = map.locationOf(actor)`
2. Similar to the implementation of power star and super mushroom, we will remove the item after we consume the item by using `currentLocation.removeItem(this)`
3. Similar to the implementation of power star, we will add the FIRE_ATTACK capability to the **FireFlower** object by using, store it into the inventory of the player and remove the consumeAction of the **FireFlower** object so that it would not provide the consume option in the menu by using,
   ```
   this.addCapability(Status.FIRE_ATTACK)
   actor.addItemToInventory(this)
   this.removeAction(consumeAction)
   ```

We have added a new attribute called turnsLeft in the **FireFlower** class. This attribute is responsible for keeping track of the number of turns of “fire attack” effect left in the player after he/she consumed it. To do that, we overridden the tick() method of the **FireFlower** so that it will decrement the turnsLeft for every turn until it is equal to 0, and we will remove the **FireFlower** object from the player’s inventory (the effect has over).

The design above follows the **Single Responsibility Principle (SRP)** such that the responsibility of keeping track of the number of turns of fire attack effect will be given to the FireFlower object itself instead of the actor who consumed it. The FireFlower class will just need to focus on its single role, provide the actor that has consumed it the fire attack effect and remvoe it when the effect has over.

The spawning of fire flower during the growing stage of the tree is done in the **GrowCapable** interface, **Sprout** and **Sapling** class. Firstly, to fulfil the **DRY** principle, we have introduced the new default method called `spawnFireFlower()` method in the **GrowCapable** interface. This is very useful since it is the same code that will be used in both of the growing stage of the tree. In this method, we will check the random probability of the fire flower spawn, if it meets the probability, it will create a new **FireFlower** object and add it to the currentLocation of the tree. This method will be called in the overriden `grow()` method in the **GrowCapable** objects, which are **Sapling** and **Sprout** class for now. All the necessary checking and spawning will be done in the `spawnFireFlower()` method itself and the **Sprout** and **Sapling** objects just need to call it.

In order to implement **Fire Attack**, those actors that are able to perform Fire attack would be given a `Status.FIRE_ATTACK` capability. **Bowser** would be given this capability upon instantiation and other actors that have consumed **Fire Flower** would also be given this capability. In **AttackAction**, if the player has `Status.FIRE_ATTACK` capability, the `menuDescription` would be added a “with fire” message to indicate that the player would be performing a **Fire Attack.** During the execution of **AttackAction**, if the actor does not miss its target and its has the `Status.FIRE_ATTACK` capability, a **Fire** would be spawned at the location of the target. This **Fire** object extends **Item** and it is set to be not portable. Fire also has a counter attribute which is initialised to 3. This counter attribute would be useful for us to track if **Fire** should be removed from the map as **Fire** should only last for 3 turns. This also follows the **SRP** as **Fire** is responsible for tracking its own turns before it fades.

In **Fire’s** `tick` method, any actor that is on this **Fire’s** location would be hurt with 20 damage. We can do this by just getting the actor at **Fire’s** location, and if the result is not None, means we have an actor standing on the Fire, who would be damaged by 20 HP. At the end of the `tick` method, the counter attribute would also be decremented by 1. When the counter attribute reaches 0, the **Fire** would be removed from the map. If the **Fire** burns an actor to death, all of the droppable items in the actor’s would be dropped and the actor would be removed from the map. If a **Koopa** that has not entered dormant state is burnt to death, the **Koopa** would be put into dormant state. A **Koopa** that has entered dormant state would also get burned however it would still not destroyed until a **Player** destroys it with a **Wrench.**