S1 2022

FIT2099 Object-Oriented Design and Implementation

Assignment 3: Further Design and Implementation

Lab_14Team6:

30041880 Junhao Li
31950124 Ashton Sequeira
29693500 Kenda Wan

Table of Contents

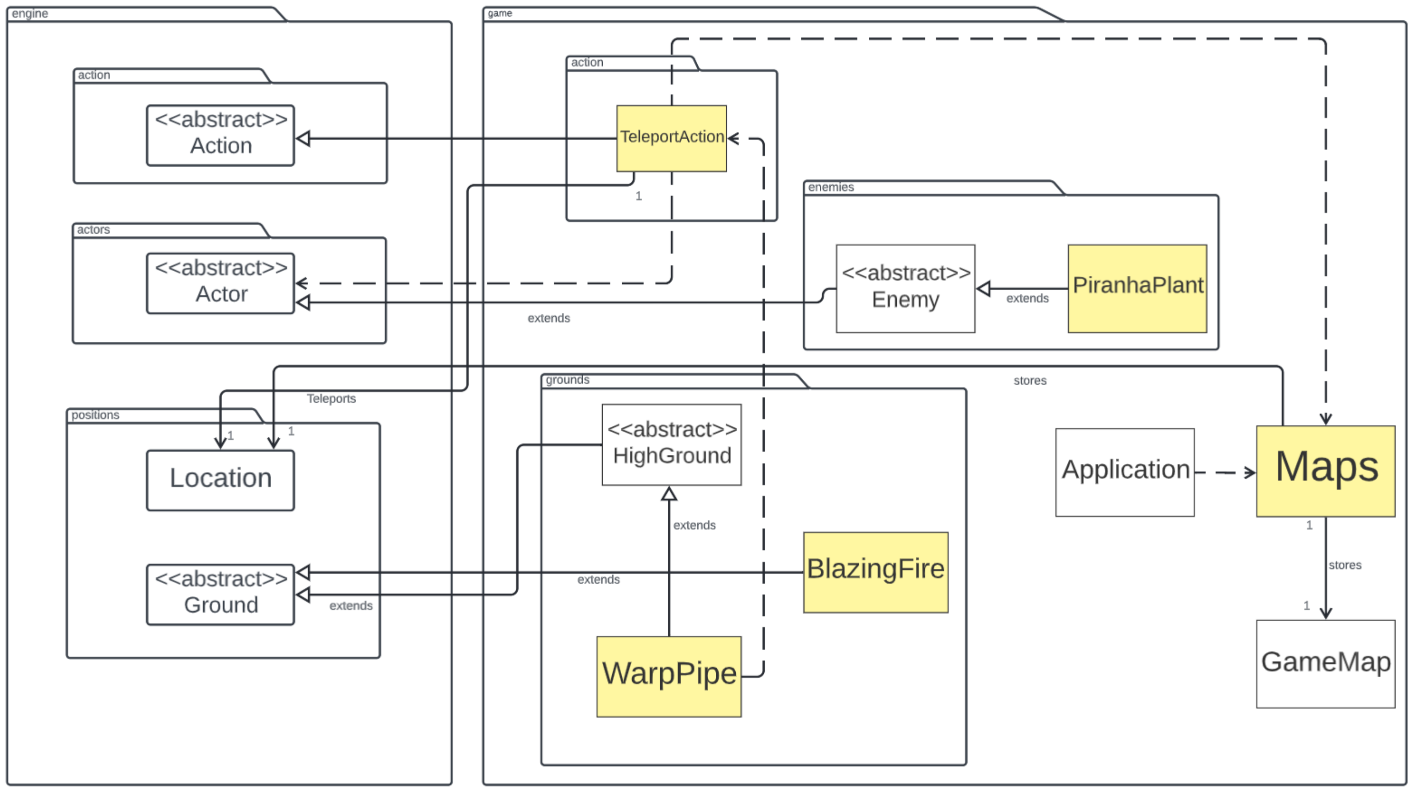
Introduction

For this assignment, we are required to further implement the FIT2022 Rogue-Like game on Mario from the previous implementation and design.

Our group decided to enter a **Structured Mode** where implementations will follow set features. The new Requirements for this assignment will allow the player; Mario, to successfully complete the walk through of the game by survival, and achieving his final goal; to save Princess Peach. But there will be many challenges and steps to get to Princess Peach. He would have to go through dangerous levels with many of his enemies along the way, then beat Bowser to obtain the golden key to unlock Princess Peach.

This document will have **UML designs** to update the designs from the previous assignment, as well as the **rationales** to clarify its respective principles of Object-Oriented Design SOLID principles.

REQ 1: Lava zone 🔥



A **new Maps class created.Maps has an association with GameMap and Location.Application and TeleportAction has a dependence with Maps:** The Maps class acts as manager class and methods of the Maps class are made static so that required classes like Teleport Action have access to the different types of Game Maps.

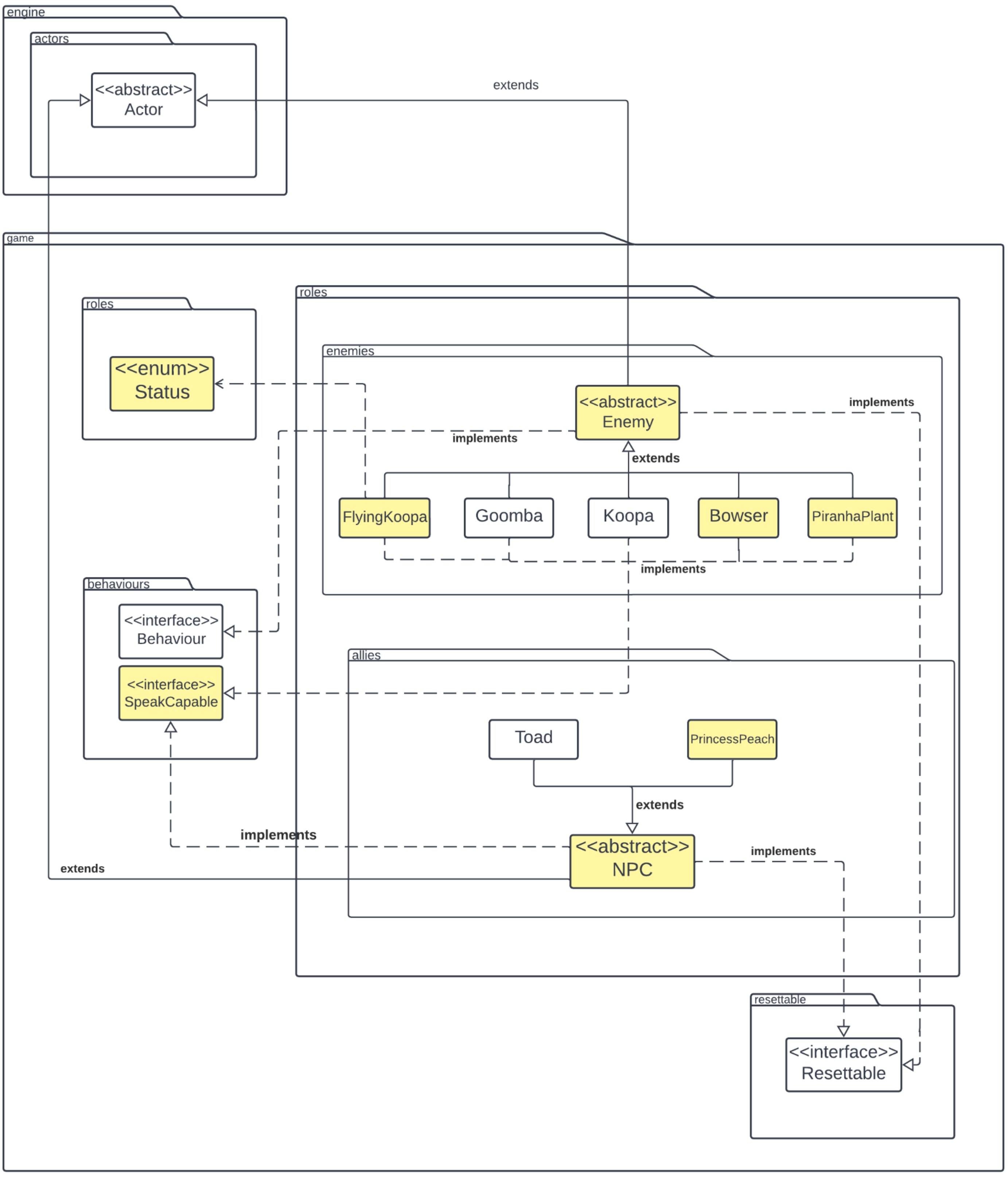
New Teleport Action extends Action class, where the player will be allowed to use when near a **New WarpPipe (c)** Grounds class which extends HighGrounds. The two new implementations follow the Single Responsible method where both classes have independent usage with their own objects, as well as the use of the general loose coupling of its respective Abstract parent classes.

For WarpPipes that exist on the main Map, if Mario jumps into LavaZone and back into the main Map , those WarpPipes will be blocked (on top of the WarpPipe) by the **new PiranhaPlant** class which extends the abstract enemy class (more about this new class in REQ 2 below). In general, players will have to retrieve **AttackAction** from Action class when near the PiranhaPlant to cause damage and hurt to kill the PiranhaPlant. Once killed, it will be removed, and then the **TeleportAction** of Action class will be added to the player's actionlist.

BlazingFire class extends Ground class.

FIT2099 S1 2022 A3 Further Design and Implementation

REQ2: More allies and enemies! 🦸 ☠️



Lab14_Team6: 30041880 Junhao Li, 31950124 Ashton Sequeira, 29693500 Kenda Wan

Updates:

- The classes ‘Toad’ and ‘PrincessPeach’ extend a new abstract class ‘NPC’ in the ‘game’ package, which extends the abstract class ‘Actor’ of the ‘engine’ package.

- The classes ‘Bowser’, ‘PiranhaPlant’ and ‘FlyingKoopa’ extend the abstract Enemy class in the ‘game’ package, which extends the abstract class ‘Actor’ of the ‘engine’ package.

- The ‘NPC’ class implements the ‘SpeakCapable’ interface.

- The child classes of Enemy implements the ‘SpeakCapable’ interface.

The classes ‘Bowser’, ‘PiranhaPlant’ and ‘FlyingKoopa’ are inside the enemies package to create cohesion. By loose coupling these classes to extend the abstract Enemy class, we are able to have mere use of the methods implementations of the Enemy class, and be able to implement those objects to suit the individual child classes. For example, the child classes all implement the grand-parent’s super constructor method `public Enemy(String name, char displayChar, int hitPoints,` and so on. Additionally there are other methods that we can `@Override` methods such as the individual class’ playTurn and allowableActions towards the classes’ needs. Since the child classes are responsible for their own terms and abilities, while they extend an abstract parent class, we are using the **Single Responsibility Principle** where there is reason to change the individual classes’ methods and fields.

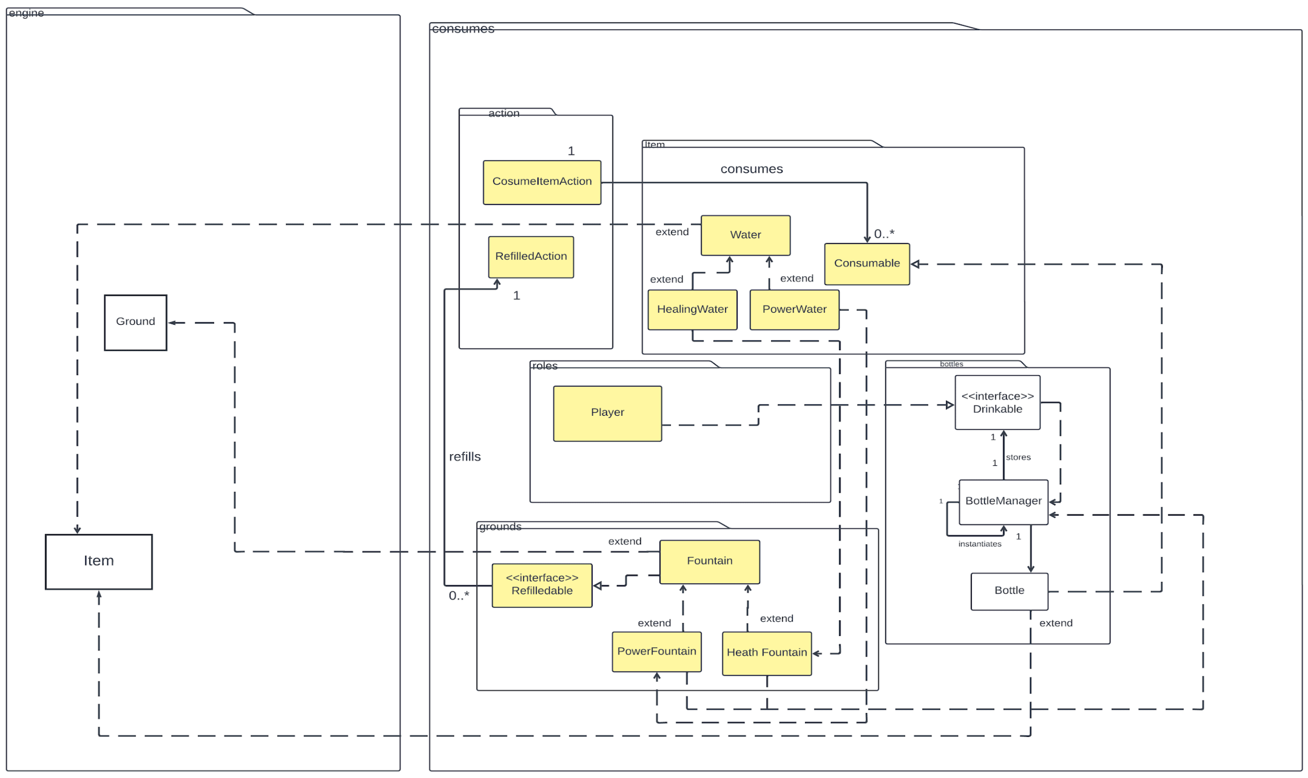
We create new capable interfaces: ‘SpeakCapable’ this is to allow the individual classes to implement whenever they need these robustness toward their own class needs. The implementation of these methods for the classes is to ensure we stay within conditional logic and not inheriting from a class or method that is not necessary or not required for these classes. Here, we are following the **Liskov Substitution Principle (LSP)**, where abstract classes are extended but the individual child classes implement specific capabilities only if they require.

For ‘FlyingKoopa’ to extend Enemy class instead of the Koopa class is to ensure that the Single Responsibility principle and the LSP is followed. Ensuring that Koopa and Flying Koopa are different, we implemented the playTurn and Resettable caller in Flying Koopa. Alongside the methods, the constructor of Flying Koopa is different from Koopa too. By following the LSP rule, we avoid violating that single class ability implementation as well as better security in the design flow for all the child classes of Enemy class.

Both **NPC and Enemy classes implement Resettable** from the Reset package in the game package.

Changes made to previous assignment: Enemy class has general public getters and constructor, while the child classes have their Lists of behaviours and allowable actions.

REQ3: Magical fountain ⛲

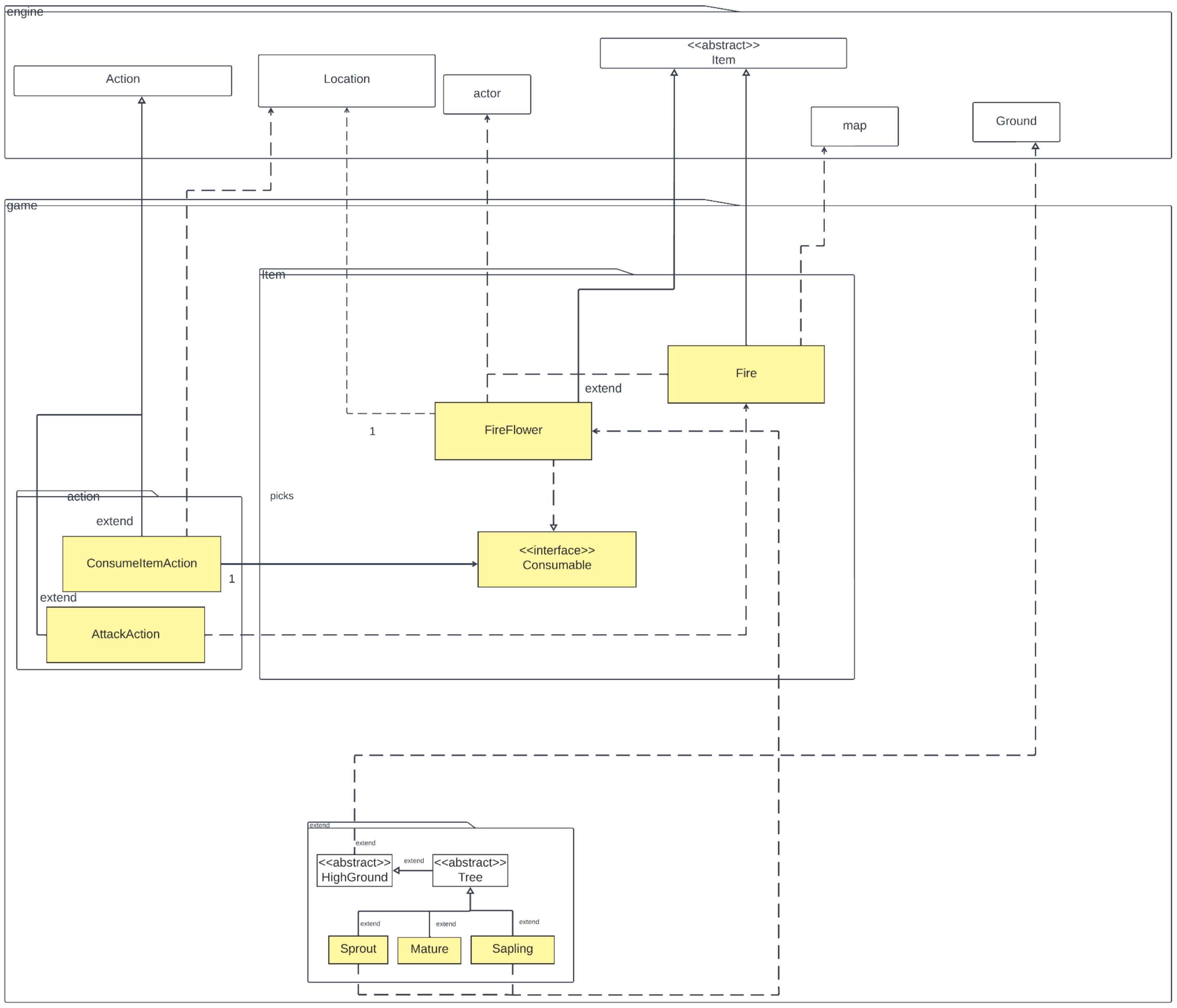


Creating BottleManager to store the player who implements the Drinkable interface and bottle. BottleManager is a static factory method that stores the drinkable and bottle. Therefore, it is open for extension through implementing the drinkable interface to the classes. It is closed for modification by not changing the way of the origin code. Hence, it meets the **Open closed principle(OCP)**.(New updated)

PowerFountain, HeathFountain extends Fountain, Fountain extends Ground and implements RefilledAble. According to the **Liskov substitution principle(LSP)**. PowerFountain and HealthFountain inherit the functionalities as a Fountain. These child classes can implement more methods in their own classes if required. Since Fountain extends Ground, all the fountains have ground functionalities. Based on interface segregation principle(ISP). Fountain implements RefilledAble interface, since it is not allowed to implement this method to the ground. Otherwise, all types of grounds can be Refilledable. But now, only PowerFountain, HealthFountain, and Fountain are refillable.If there are more fountains to be extended, the system supports extending more classes without modifying the original code **Open closed principle(OCP).** (New updated)

PowerWater, HealingWater extends Water. Water extends Item. By doing this, PowerWater and HealingWater inherit Water that has the functionalities of the item. That satisfies the **Liskov substitution principle(LSP).** (New updated)

REQ4: Flowers 🌻 (Structured Mode)



FireFlower and Fire extend Item. It meets the **Liskov substitution principle(LSP)** through showing FireFlower and Fire as subclasses that have an instance of the base class. Therefore, they have the same functionalities as item class. By doing this, it can reduce the repetitive methods (New updated)

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.

REQ5: Speaking 🗣(Structured Mode)

[the UML for this REQ5 is included in REQ2 of FIT2099_CL_Lab14Team6_A3 where allies and enemies have the SpeakCapable Interfaces, so please refer to REQ2 above for the UML and Design Rationale]

WBA

WORK BREAKDOWN AGREEMENT

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

```
REQ 1-5:
   JavaDoc:
       JunHao Li, Kenda Wan

   Implementation:
       Q1: Ashton Sequeira          (New Map,WarpPipe,BlazingFire,Maps,TeleportAction)
       Q2: Kenda Wan, JunHao Li     (NPC: Princess Peach, Enemy: Bowser, FlyingKoopa,
                                      PiranhaPlant)
       Q3: JunHao Li                (Bottle, Fountains)
       Q4: JunHao Li                (FireFlower, FireAttack)
       Q5: Ashton Sequeira          (Speakable, Statements)

   UML and Rationale:
       Q1: Ashton Sequeira
       Q2: Kenda Wan
       Q3: JunHao Li
       Q4: JunHao Li
       Q5: Kenda Wan
```

Diagram developer:
Designs diagrams 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 form to update the progress of the team.

Diagram reviewer:
Reviews and checks the diagram that is designed for quality assurance. The
A diagram reviewer needs to review diagrams designed by other diagram developers. 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

22/05/2022