Design Rational Requirement 3

REQ3: Enemies

Overview

REQ3 is all about the enemies in this game. There are two enemies in the game, Goomba and Koompa. Each of them has their individual game mechanics that will be implemented in this requirement.

Goomba (g)

- Starts with 20HP.
- Deal 10HP upon every successful attack, has 50% chance to register a successful attack.
- Has 10% chance to despawn (removed) from the map in every turn.

Koompa (K)

- Starts with 100HP.
- Deals 30HP on every successful hit, has 50% chance to land a successful hit.
- When defeated, Koompa will turn into Dormant state (D), and it stays on the ground and does nothing.
- Wrench must be used to destroy the shell (when Koompa is in Dormant state). Wrench deals 50 damage and 80% hit rate.
- Destroying the shell drops Super Mushroom.

Once engaged in combat mode (either player attacks the enemy or the other way around), the enemy will follow the player until it is defeated.

Implementation

Goomba

The Goomba class is given already, thus we don’t need to create another class just for the character. So, at first, Goomba’s basic stats will be added to the class as its global attributes

```
public class Goomba extends Actor {
  /*
  * CONSTANT 10% chance to be removed from the map
  * sets HP attributes
  * sets HP per damage, and hit percentage
  */
}
```

For every turn, Goomba has 10% chance to be removed from the map. Thus, we can implement a method to remove Goomba if Goomba hits the 10% chance.

```
public void removeGoomba (){
  /* if goomba hits the CONSTANT 10% chance created
  * goomba is removed from the map.
  */
}
```

Koompa

Since there is no given Koompa class before, we shall create its own dedicated class. Similar to Goomba, Koompa's basic stats will be declared as a global attribute.

```
public class Koompa extends Actor {
  /*
  * sets HP attributes
  * sets HP per damage, and hit percentage
  */
}
```

There is another similar check that is needed to perform if the player manages to defeat the Koompa. If the player defeats Koompa, Koompa will turn into Dormant state. We can put Dormant state in the Status class, and we implement the Status into the AttackAction as well.

```
public enum Status {
  HOSTILE_TO_ENEMY, // use this status to be considered hostile towards enemy (e.g., to be attacked by enemy)
  TALL, // use this status to tell that current instance has "grown".
  DORMANT // use this status to tell Koompa is in Dormant state
}
```

Since Koompa has a special state when it is defeated, a special class dedicated to deal with destroying the shell of Koompa.

```
public class DestroyShellAction extends Action {
  /* code goes here
  */
}
```

Justification of Implementations

Goomba — Advantages

The implementation method written above uses several design principles, mainly Open-closed Principle (OCP) and Single Responsibility Principle (SRP).

Goomba has its own dedicated class, and it is a subclass of Actor superclass. This is essentially OCP as whatever changes Goomba class has, it will not affect other classes, especially Actor superclass.

SRP is used when Goomba's attack action is placed inside AttackAction class. This is because AttackAction has one and only one purpose/responsibility only: to create and register the attack action made by one actor to a target.

Goomba — Disadvantages

NA

Koompa — Advantages

Similar to Goomba's justification written above, Koompa also uses OCP and SRP while implementing its methods.

Since Koompa has its own special state when defeated (but not killed), we put the status inside the Status enum class so that we can avoid using excessive literals.

Koompa — *Disadvantages*

NA