Design Rational Requirement 4

REQ4: Magical Items

Overview

REQ4 is about adding Magical Items into the game. The items allows the player to consume them and gain extra power and effects so that the player may have an easier time to play the game. There are two Magical Items in the game, they are Super Mushroom and Power Star. Each of them has their own unique abilities that enhances the player experience of playing the game.

Super Mushroom (^)

When a player consumes Super Mushroom:

- The player max HP is increased by 50 and its current HP is healed to the max.
- The player's display character turns to uppercase (m → M).
- The player can jump freely with no fall damage and 100% jump success rate.

The effect lasts indefinitely till the player receives any damage (from the enemy). Once the effect wears off, the player's display character turns back to lowercase (M → m).

Power Star (*)

When a player consumes Power Star:

- The player does not need to "jump" to higher ground. Instead, the higher ground will be destroyed and turn into dirt.
- Destroyed ground drops a Coin (5$).
- The player receives immunity and does not take any damage from enemies.
- Under the effect of Power Star, a successful attack from the player will always kill the enemy.

Power Star lasts for 10 turns. If within the 10 turns it does not get picked up by a player, it will disappear from the game. If Power Star is picked up by the player, it will last for 10 turns in the player's inventory, before disappearing from the player's inventory. When the player consumes Power Star, the effects written above last for 10 turns only.

Implementation

Super Mushroom

In order to implement Super Mushroom as a Magical Item into the game, a class called SuperMushroom is created, and it is a subclass of the Item abstract class, as shown below.

```
public class SuperMushroom extends Item {
  /* creates a super constructor since its a subclass
  * the implementation continues below
  */
}
```

Since the SuperMushroom effect lasts for an indefinite time until the player receives damage from enemies, we can utilize the given enum class Status and use it to set the status of the player.

```
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".
  EFFECT_SUPER_MUSHROOM // example, use this status when the player consumes super mushroom
}
```

We can create a ConsumeItem abstract class, and it is a subclass of Action abstract superclass. Then we can create a subclass of ConsumeItem named ConsumeSuperMushroom class.

```
public abstract class ConsumeItem extends Action {
  //abstract class for other items to be consumed
}
```

```
public class ConsumeSuperMushroom extends ConsumeItem {
  /* a specific class to run the consume super mushroom action
  * when run, increase player HP by 50
  * set status EFFECT_SUPER_MUSHROOM
  * set player displaye character uppercase
  */
}
```

The effect last until the player receives damage from the enemies, therefore, we can implement it inside the given AttackAction class.

```
public class AttackAction extends Action {

  /*
  * ... given code in the class
  */

  @Override
  public String execute(Actor actor, GameMap map) {

    /*
    * given method code ...
    */

    /* check if target has EFFECT_SUPER_MUSHROOM
    * if yes, remove the target's status,
    * reset the target's display character to lowerCase
    * deal damage to the target
```

```
    */

    /*
    * ...  code continues below
    */
  }

  /*
  * ... given code in the class
  */
}
```

Not only that, if the effect hasn't worn off, the player can jump to any place with 100% success rate and no fall damage will be taken. We can also implement this in the JumpAction class.

```
public class JumpAction extends Action {
  /*
  * some code...
  */

  /* a method to check if the actor has EFFECT_SUPER_MUSHROOM
  * if yes, the actor jumps with 100% success rate
  * and does not take any fall damage
  */
}
```

Power Star

In order to implement Power Star class, we would do exactly the same as to how we implemented Super Mushroom.

```
public class PowerStar extends Item {
}
```

Likewise, when a Power Star is consumed, a status will be granted to the player. Unlike the status for Super Mushroom, the status for Power Mushroom must be displayed in the console.

```
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".
  EFFECT_SUPER_MUSHROOM, // example, use this status when the player consumes super mushroom
  IMMUNITY //example, use this status when the player consumes power star
}
```

A similar consume class will also be implemented.

```
public class ConsumePowerStar extends ConsumeItem {
}
```

We can utilize the special status effect to grants the subsequent special effects to the player. For example, under the special effect, each successful attack made by the player is an instant kill to the enemies.

```
public class AttackAction extends Action {

  /*
  * ... given code in the class
  */

  @Override
  public String execute(Actor actor, GameMap map) {

    /*
    * given method code ...
    */

    /* check if actor has IMMUNITY status, if yes
    * actor damage to target = a very large number,
    * deal damage to the target
    * else, do normal damage target
    */

    /*
    * checks if target has IMMUNITY status
    * if yes (that means the target is the actor), actor takes 0 damage
    * else, continue Super Mushroom code above
    */

    /*
    * ...  code continues below
    */
  }

  /*
  * ... given code in the class
  */
}
```

Furthermore, we can use this status effect to make the jump effect as well. For example:

```
public class JumpAction extends Action {
  /*
  * some code..
  */

  /* special check for IMMUNITY status
  * if yes, actor does not need to jump, move normally to the tile
  * sets the ground to dirt, then drop the coin on the ground
  */
}
```

While high ground will be destroyed and convert to dirt, and drops a coin.

```
public class Coin extends Item {
  /*
  * some code...
  */

  /* special method for IMMUNITY status
  * drops the coin on the ground
  */
}
```

Justification of Implementations

Super Mushroom — Advantages

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

OCP is used when the SuperMushroom class is created and it is a subclass of the abstract class Item. This way, SuperMushroom will have its dedicated class, thus its own changes will not affect other classes.

Not only that, ConsumeSuperMushroom is a subclass of the ConsumeAction. This also follows the idea of OCP while implementing.

SRP is used when the player attacks any enemies after consuming a Super Mushroom. Since AttackAction class is given, it shall only be used as one single action only. We create an if-statement to check if the player has the effect status, if yes, the player's attack action is modified.

This if-statement is not and shouldn't be placed in the SuperMushroom class because it's an AttackAction, therefore it should be grouped and placed within the AttackAction class.

The special effect when consuming a Super Mushroom is placed inside the given enum special class. This prevents excessive use of literals, which will make the code extra messy.

Super Mushroom — Disadvantages

By putting an if-statement inside the AttackAction class, this means that every time when a non-actor (Goomba/Koompa) attacks the player, the if-statement will always run and check if the player is under Super Mushroom effects. This is inefficient and resources are wasted to run this check.

Power Star — Advantages

Similar to the advantages of the implementation method of Super Mushroom, OCP and SRP design principles are also used when implementing Power Star item. OCP is used when a dedicated PowerStar class is created, and also ConsumePowerStar subclass is created, and SRP is also used when the attack action is modified and placed inside AttackAction class, with similar reasoning written above. The special effect of Power Star is also created with the same reason written on the Justification of implementation of Super Mushroom.

Power Star — Disadvantages

Similar to Super Mushroom's disadvantage, two if-statements are placed within the execute() method, one is for dealing damage, another one is within the SuperMushroom check . This is very inefficient as every time an attack action runs, it will spend time checking these conditions.

Furthermore, if the player has *IMMUNITY* status, whenever the player performs a jump action, the jump action will be overwritten to normal move action, and then it will call the special method in Location class that sets the ground to dirt, and then Location special method will call Coin class special method to drop a coin on the ground. This is similar to a "spaghetti code", and it is an inefficient way to code out the effects. If there is a change in the method, it will affect the subsequent methods and cause a chain of errors.