Assignment 3 REQ4: Flowers

Overview:

To achieve this feature, there will be three new classes (i.e., Fire, FireFlower, and FireAttackAction) created in the extended system, and five existing classes (i.e., Player, Enemy, Status, Sprout, and Sapling) will be modified. The design rationale for each new or modified class is shown on the following pages.

1) Fire class

Why I chose to do it that way:

Fire is a new class that extends the Item class. According to the assignment specification, it says that the fire will stay on the ground for three turns. Hence, I created a pointer and initialized it as 3. For each tick method being called, the pointer will be minus 1, and the fire will be removed once the pointer is equal to 0. Besides that, it will deal 20 damage per turn to the actor currently standing in the same location with the fire. After the actor gets the damage, if the actor is not conscious, then we will remove it. However, we could not directly remove those unconscious actors that can hide in the shell or drop a key, so that the next play turn method of those actors can do the operation (i.e., hide in the shell or drop a key).

With this design, we are following the Single Responsibility Principle as the method within the Fire class only shows the properties of a Fire. Additionally, for each of the properties of a Fire instance (e.g., damage per turn, display character, and so on), I store them as private constant class attributes. Consequently, we are following the “avoid excessive use of literals” principle as it makes our code more readable.

2) FireFlower class

Why I chose to do it that way:

According to the assignment specification, after the actor consumes the fire flower, it can perform a fire attack action for 20 turns. However, I realized that this magical item is very similar to the power star, as it also gives the actor a particular buff for a certain number of turns. Hence, I will do the same thing I did to PowerStar class, which is to let FireFlower class extend MagicalItem and override the currentStatus() and updateStatus() to do the corresponding operation. By doing so, there is no other code needed to be modified to keep track of the effect of the fire flower (e.g., remaining turn, when will it be expired, and so on) as the MagicalItem class and BuffManager class created in Assignment 2 already does the job. In other words, I just need to override those two methods based on the description and effect of fire flowers to make the implementation work. Besides that, we will add the FIRE_ATTACK constant to the actor that consumed it. Please refer to the Status class section for more details regarding this constant.

With the above design, we are following the DRY principle as I did not create a class that has repeated code that is the same as those magical items (e.g., super mushroom and power star) since I just extended the MagicalItem class and use the common code. Besides that, we are following the Single Responsibility Principle as the method within FireFlower class only focuses on the buff that can be given to the consumer (i.e., player). Moreover, for each of the properties of a FireFlower instance (e.g., name, display character, and so on), I

store them as private constant class attributes. Consequently, we are following the “avoid excessive use of literals” principle as it makes our code more readable.

3) Status enum

What changed in the design between Assignment 2 and Assignment 3 and Why:

In Assignment 3, I created a new constant called FIRE_ATTACK. This constant will be added to the actor once it consumes the fire flower. As a result, fire attack action will be available to the actor.

Why I chose to do it that way:

With this design, we will not use any more literal to specify whether a Player/Actor has a particular capability in another class as we can make use of these constants. Hence, we are adhering to the “avoid excessive use of literals” principle. This kind of implementation also provides us more flexibility for future development as we can simply add additional capability when needed.

4) FireAttackAction class

Why I chose to do it that way:

FireAttackAction class allows the actor to perform a fire attack action (i.e., dropping fire on the target’s location). It is a new class that extends Action. It will be available to the player once it consumes the fire flower. Once the player executes the fire attack action, a fire will be dropped at the target’s location. However, if the player is currently invincible, it will directly remove the target. Besides that, if the target can drop a key, then a key will spawn at that location. Eventually, a string with a suitable description will be printed out to notify the user that a fire attack action has been performed.

With the above design, we are adhering to the Single Responsibility Principle as FireAttackAction class only focusing on what should happen when a fire attack action is performed.

5) Player class

What changed in the design between Assignment 2 and Assignment 3 and Why:

The only change that I made between Assignment 2 and Assignment 3 is that I added a new if statement in playTurn such that it will display “Mario can use FIRE ATTACK!” when the player has the capability of FIRE_ATTACK. The rest of the part remains the same as previous assignments. Thus, please refer to the design rationale for this class in previous assignments for more details.

6) Enemy class

What changed in the design between Assignment 2 and Assignment 3 and Why:

The only change that I made between Assignment 2 and Assignment 3 is that I added a new if statement in allowableAction such that it will add a FireAttackAction instance to the actions list if the when the player has the capability of FIRE_ATTACK (i.e., allow the player to

perform fire attack action on the enemy). The rest of the part remains the same as previous assignments. Thus, please refer to the design rationale for this class in previous assignments for more details.

7) Sprout class

What changed in the design between Assignment 2 and Assignment 3 and Why:

The only change that I made between Assignment 2 and Assignment 3 is when the Sprout instance' age reaches 10, before we change the type of ground at the location to be Sapling, we will have a 50% chance to spawn a fire flower on that location. This could be done by using an if statement. However, the rest of the part remains the same as previous assignments. Thus, please refer to the design rationale for this class in previous assignments for more details.

8) Sapling class

What changed in the design between Assignment 2 and Assignment 3 and Why:

The only change that I made between Assignment 2 and Assignment 3 is when the Sapling instance's age reaches 10, before we change the type of ground at the location to be Mature, we will have a 50% chance to spawn a fire flower on that location. This could be done by using an if statement. However, the rest of the part remains the same as previous assignments. Thus, please refer to the design rationale for this class in previous assignments for more details.