Design Rationale: Updated for Assignment 3

Improvement from Assignment 2

Estus Flask

The Estus Flask instance was removed as an attribute from the Player class and is instead added as an instance to the inventory of the Player. This removes unnecessary dependency between the Player class and the Estus Flask class. Instead, the inventory of the Player is searched for an instance of the Estus Flask class to get the Player's Estus Flask.

Death Actions

Player, Enemies and Lord of Cinder class now have their own death action classes, KillPlayerAction, EnemyDeathAction, and LordOfCinderDeathAction instead of death's being handled with the AttackAction, thus reducing coupling. This ensures that each action class follows the Single Responsibility Principle and removes the multiple responsibilities of attacking and killing enemies the AttackAction had before.

Stunnable Interface

The waitTurn attribute was removed as a variable from classes that needed to be stunned, like YhormtheGiant. Instead, it is implemented as an interface and implemented by all the classes that need to be stunned. This now follows the Interface Segregation Principle and ensures that clients are not exposed to unnecessary methods they never use.

Enemy Class

Previously, the Enemy class was only used to check if an instance of an actor is an enemy and did not implement the methods shared by all enemy child classes, making high-level modules dependent on low-level modules. This violated the Dependency Inversion Principle as well as the DRY Principle. Now all the shared methods that a default enemy would have such as playTurn() and getAllowableActions() have been implemented into the abstract enemy class so that the methods can be reused across classes instead of being implemented repeatedly.

Storm Ruler Charging Actions

Previously, when we executed ChargeAction, three turns would take place automatically, but now, it works as it's supposed to and the Player has to call ChargeAction manually, and can't attack enemies while doing so. We did this by implementing a Chargeable interface that any chargeable weapon will use. This makes the code more dynamic and eliminates any possibility of violating the DRY Principle.

Random Die Behavior

Previously, in Undead WanderBehavior was added first, then RandomDieBehavior, which prevented it from randomly dying in the game since WanderBehaviour was always executed and so it never did RandomDieBehaviour. This has been changed by adding RandomDieBehavior first, then WanderBehavior, which now allows for random deaths and follows an appropriate behaviour order, improving the functionality of the game overall.

Burn Ground Action

Previously, Fire was made only when the EmberFormAction class was executed, which meant that it could only be performed by Yhorm The Giant. A new class BurnGroundAction was created to do the Burn Ground action that creates the fire, and so allows any actor holding Yhorm's Great Machete to perform this action. It even makes it so that the actor holding the weapon does not get damaged by the Fire it creates. This makes the implementation more dynamic, prevents violation of the DRY Principle, and reduces coupling in the EmberFormAction class.

Kill Self Action

The KillSelfAction class has been removed and its methods were placed into the RandomDieBehavior class so that it calls itself instead of KillSelfAction. This improves the quality of implementation overall because there are fewer unnecessary dependencies across classes, higher cohesion, and the existing ones are properly and thoroughly utilized.

Design Rationale for Assignment 3

Lord of Cinder

Lord of Cinder is now an abstract class, which its child classes AldrichTheDevourer and YhormTheGiant extend from. This ensures that we don’t violate the Dependency Inversion Principle as well as the DRY Principle by repeating shared methods across the child classes. It also follows the Open/Closed Principle since LordOfCinder will be open for extension but closed for modification, since the role of Lord of Cinder is well defined.

Requirement 1

In order to move from one map to another, a new action class, UseFogDoorAction, was made. This follows the Single Responsibility Principle because a new class is created to perform the Fog Door action.

Requirement 2

We re-used the Bonfire class by making a constructor that takes in its own location so that we can also store the location in the Bonfire class, and so we also create Bonfire objects in a different way than using FancyGroundFactory. This allows us to follow the Single Responsibility principle as we didn't have to create other Bonfire classes to represent several bonfires in the game.

Requirement 3

Aldrich the Devourer is a child of Lord of Cinder, whose only difference between Yhorm the Giant is its playTurn method which has been modified to allow for ranged attacks. Otherwise, it is exactly the same as YhormTheGiant where it follows and attacks the player, dropping the Cinder of Lord when it dies. This follows the Open/Closed principle since we made Aldrich inherit the Lord of Cinder class.

Requirement 4

We used the Dependency Inversion Principle to make the Chest and Mimic classes since they are coded children of Enemy class and that Mimic inherits Enemy class's playTurn method instead of overriding its own special one, since it's a basic enemy that just follows and attacks the Player when the Player goes near, just like Undead enemies.

<u>Requirement 5</u>

We followed the DRY principle for the Cinder of Lord by making sure that all Lord of Cinder classes drop a Cinder of Lord via the LordOfCinderDeathAction.