DESIGN RATIONALE

The overall design choices are made keeping in mind the extensibility of the game. The details of the design choices for each requirement are as follows-

Requirement 1: Player and Estus Flask

Design Choices:

1. **Display Health/Hit Points**
   To display the health/ hit points of the Player, we can implement code in the pre-existing playTurn method of the Player class so there is no need for a new class. The attribute of hit points is already found in the Actor class so a new health system doesn't need to be implemented. We only need to display it. This can be done by calling the methods of the Display class.

2. **Hold Estus Flask:**
   The player needs to hold an Estus Flask, for this feature, we can create a new EstusFlask class that extends the Item class with private attribute quantity. We need to extend the Item class so that we can use it as an item and print out its attribute(number of charges) in the console.

3. **Cannot Drop Estus Flask:**
   The player cannot drop the Estus Flask so we can override the pre-existing DropItemAction method to return null. We need to override the DropItemAction so that the player is not allowed to drop the Estus Flask. This will be a good design choice as we can make use of the pre-existing system to fulfill this requirement.

4. **Drink Estus Flask**
   The player should be able to drink the Estus Flask, so we can create a new class called DrinkEstusFlaskAction that we will put as an allowable action for the EstusFlask class. We need to create this class so that we can use the execute method of the Action class to access the player's instance and add the health of the player.

5. **Display Charges of Estus Flask**
   To display the number of charges of the Estus Flask in the console, we can implement code in the pre-existing MenuDescription method of the Action class. This will be a good design choice as the pre-existing system is sufficient to fulfill this requirement.

Requirement 2: Bonfire

Design Choices:

1. **Create Bonfire Class**
   The player starts the game at the Firelink Shrine(Bonfire), for this, we can create a new Bonfire class that extends the Ground. We need to extend the Ground class so that we can access the methods and private attributes that are already present in the Ground class.
   We do not extend the Actor class as it does not have most of the attributes an Actor should have, like isConcious, hurt, etc. However, since BonFire is a Ground object, we cannot add it into the map using methods in GameMap. Hence, we need to modify the initially given map so that there's a Bonfire (displayed as B) on the map.

2. **Create ResetAction class**
   To perform the Rest at Firelink Shrine action, we can create a class named ResetAction that extends Action and has a private attribute of the class ResetManager. The purpose of creating this new class is so that we can get the Map as a parameter which will allow us to access all the actors in the map and check if they are resettable. Then, it will only reset those actors who are resettable to their original position, while others will be removed from the map. Another advantage of extending the Actor class is we can use methods in the Action class too like menuDescription if we want to.

Requirement 3: Souls

Design Choices:

1. **Create new method rewardSystem at AttackAction class**
   When the player slays/kills enemies, the player gains a certain number of souls from them.
   We have implemented this feature inside the pre-existing AttackAction class along with a helper function named rewardSystem. When the Player kills a target, the helper function is then called to get the appropriate rewards for the particular slain enemy, which is then handled in the AttackAction execute function. This function will then use the methods in Player class to increase the souls of players.

2. **Override methods of Soul class in Player class**
   Methods like addSouls and subtractSouls need to be overridden so that player's souls can increase or decrease after implementing the methods.