Design Rationale

In our proposed design, we have decided to make all the weapons inherit from melee weapon class that we had found in the code given to us, since all the weapons available at this current stage are melee weapons. However it is possible to make the weapons inherit from the abstract item class as well since they are technically items that the player can equip and that the vendor can sell. We have set the estus flask to inherit from the abstract item class since it is an item in the game that is not a weapon. One other way to implement this could have been to make the estus flask an attribute in the players stats, however this would make changing the effects of the estus flask more complicated.

We created the SoftResetAction class and RestAction classes to manage these actions when they occur and to use them to call the ResetManager class. This would also mean that RestAction and SoftResetAction would be dependent on ResetManager. The Resettable interface is responsible for resetting the Estus flask, player stats and enemy stats.

For enemies, we created an abstract class called Enemies where the more specific types of enemies would inherit from. It is not necessary to have made this class, however we feel that by creating this class we make it easier for adding the same attributes and methods to all enemy types.

As for Valley and Cemetery, those classes will inherit the abstract ground class as they are also different types of terrain that can be anywhere on the map. We then made the existing Floor and Dirt classes interface with the soul interface. This is because when a player dies, a “Soul Token” is created on the ground where the player died and if the player were to fall into the valley and die then the soul token would not spawn in the valley but outside on a floor tile next to it.

Lastly, our vendor simply has an association with the abstract weapons class as the vendor is able to sell some of these weapons. Also, by making it associated with the abstract weapons class we can now easily add more weapons to the vendors shop list.

We have decided to create a class for each action of the Storm Ruler so there will be a Charge class and a WindSlash class and both classes will extend the WeaponAction class. In the Storm Ruler class, we will have an ArrayList attribute name weaponActions that contains all the action that can be performed by the Storm Ruler (by now it will consist of Charge and WindSlash). When we want to perform the Storm Ruler's action, we will loop through the weaponActions to get the action we want to perform. Since both Charge and WindSlash inherit the WeaponAction class, both of them will have an execute method that takes in an Actor and the GameMap as parameters where Actor is the target that the WeaponAction will perform to. By getting the Storm Ruler's action (Charge or WindSlash) and call its execute method, we can perform the Storm Ruler active action.