Lab 6 Team 2 Design Rationale

Requirement 1: Player and Estus Flask

There is a 1 to 1 relationship between the Player class and the Actor class as there is only 1 instance of the Player class in the game. A new class called the EstusFlask class is created. It is inherited from the Item abstract class as it is classified under Item. It has a 1 to 1 relationship as there is only 1 instance of the EstusFlask class in the game. It also uses the Abilities enums class as it has the Ability to heal. There is also an instance of the Broadsword class that is inherited from the GameWeaponItem class. It has a 1 to 1 relationship as only 1 Broadsword is instantiated. It uses the abilities class due to it having a passive. The player, EstusFlask and Broadsword class are all instantiated in the Application driver class.

Requirement 2: Bonfire

We assigned the Bonfire as a ground class and had it create the RestAction action according to requirements. Application and Bonfire have a 1 to many relationship as during future development more bonfires could be added. Each class that can be reset, enemies, player and estus flask are added into the resettables list through the resettable interface so that when the player rests they can all be reset. In the rest action sequence diagram everything in the list is removed from the map, then the list is run through again restoring all actors and the estus flask to their initial states. To stop the lord of cinder from being resurrected but still making sure he is healed on reset, a check is done to make sure he is only placed on the map if the player has not killed him.

Requirement 3: Souls

For the souls requirement no classes need to be added. Each player and enemy will be given a souls attribute which will be affected by actions just like hp would. If an enemy dies to a player its soul value will be added to the player's own and if the player dies, their soul value will be set to 0 dropping the souls they had. When interacting with the vendor the soul interface will be used to subtract from the player's soul using the value of the merchant's wares. We chose this design because as more enemies and items are added into the game previous code will not need to be changed, each new addition will just need to be assigned their soul value.

Requirement 4: Enemies

In the requirement 4 UML class diagram, I have added the new enemy skeleton and made changes to the old enemies Lord of Cinder and undead. The skeleton and lord of cinder both initialise with a weapon, with the skeletons being chosen between the greataxe and broadsword randomly each time they respawn. The enrage behavior is added to the behavior interface as future enemies could also use this behavior so we don’t want to make it specific to the lord of cinder. Yhorm will drop the Cinders of a lord upon death however it will only be created once as he can only die once hence the 1 to 1 multiplicity. Both the skeleton and undead have a unique ability, resurrection and random death respectively, which is stored in the abilities enum. We chose this solution as new enemies could share these abilities and it would save us on having to recode the same thing multiple times

Requirement 5 : Terrains (Valley and Cemetery)
The following terrains valley and cemetery both introduced new classes, and the cemetery inherits the features of ground abstract class. Which then can be used to provide a location on the map. Here we can then spawn undead close to the cemetery using the actors location abstract class. Each cemetery can spawn many undead but only one per turn. The application creates 1 or many cemeteries on the map.

Valley also inherits ground and takes on the ability to kill a player which is used as an ability when the player is on the x&y location of valley. This will create a dying spot in which to inherit the abstract class ground. Dieing spot will also implement Soul which will add the functionality of transferring souls to the object dieing spot. This will ensure that that dieing spot will have the souls the player used to have.

Requirement 6 Soft reset/Dying in the game

The dieing sequential diagram explains the dying process, The player will die() and will run the rest action which will reset instances of the player. Then it will activate a method of a new spot which will create a new instance of dying spot and transfer the souls to the instance. This will tell the user how many souls were left and that the user now has zero souls. The dieing instance will also use a check dieing spot method to check whether the dying spot is on the same x&y as a cemetery; if so it will move it to the side of the x&y.

Requirement 7: Weapon
Created a Sword and Axe abstract class that inherits GameWeaponItem class which inherits the WeaponItem abstract class. They are abstract classes as they are subclassed into GiantAxe, Yhorm'sGreatAxe. Broadsword and StormRuler classes. The newly created Axe and Sword class implements the Abilities enums class as the Abilities class has all the Item active and passive abilities. GiantAxe and Yhorm'sGreatMachete are also newly created classes that inherit the Axe class as Axe type weapons contain different abilities from Swords. StormRuler class and the Broadsword classes are also newly created classes that inherit the Sword class and the Sword class implements the Abilities enum class as all item abilities are in the Abilities enum class. The Axe and Sword classes all have a relationship of 1 to 1 with the GameWeaponItem as they are just one instance of the GameWeaponItem class. The StormRuler and Yhorm'sGreatMachete each have a 1 to 1 relationship as only one instance of class exists in the game. The GiantAxe and Broadsword have a 1 to 1..* relationship with the Axe and Sword classes as there can be one or many instances of it in the game.

Requirement 8: Vendor
Created a vendor class that inherits the abstract class Actor as this will be another actor instance that will be able to interact with the Player through the increasesHealth method. The vendor implements the soul interface as the vendor will implement these methods when selling the giant axe, broadsword and increasing target health. The vendor also has an association with the classes Broadsword and GiantAxe as these will be available with the vendor to be sold. The vendor creates a VendorAction class that inherits from the Action abstract class as the vendor needs to interact with the Player. The relationship between the

Vendor and the Broadsword and GiantAxe classes are 1 to 1..* as the Broadsword and GiantAxe are only sold by 1 Vendor but the Vendor can sell more than one instance of the classes. The vendor and the player have a 1 to 1 relationship as there is only one player instance in the game.