ASSIGNMENT 1

Analysis and design

Team 1:

An Khoa Nguyen

Adrian Ching Cheng

Dongzheng Wu

Link our git: https://git.infotech.monash.edu/fit2099s221/CL_Lab1Team1/project

Requirement 1: Player and Estus Flask

Because one player can hold only one Broadsword so the relationship between Player and Broadsword should be an association. To be more specific, we treat the Broadsword as an attribute of the Player class that the Player must have when starting the game
 (Broadsword will be created inside the constructor of Player class ---> Association)

For Estus Flask, because each player has only one Estus Flask, we treat it as an attribute for the player.To be more specific, we treat the Estus Flask as an attribute of the Player class that the Player must have when starting the game and therefore, the multiplicity should be 1 to 1. Also, the Estus Flask should be inherited from the Abstract Item class (@see engine package). Because it might need to implement some methods that the Item Class has. Moreover, we choose to not inherit the Estus Flask from the PortableItem because we have looked through all of the requirements and it never said there is the time that the player might drop the Estus Flask. So Item class should be the best choice for the Estus Flask to be inherited.

Requirement 2: Bonfire

After looking through all of the code, we have figured out that our given code is already made by the floors and the walls for the Bonfire (@see Application.java), therefore when designing the class Bonfire, we don’t need to inherit it from Wall and Floor to build it anymore, because from now, Bonfire Class is just only drawing a letter ‘B’ in the game map. As a result, the Class Bonfire only needs to interact with the class Game map. And we treat them as an association because it is obvious that Bonfire is the child that is drawn inside the Game map. Therefore, Bonfire should be an attribute of the game map. Also, we treat the relationship between Player and Bonfire as a dependency because at the start of the game, the Player will implement the method that locate them to the Bonfire, which means the Bonfire should be treated as an instance that will give the information of their position to the Player to be located at the beginning of the game. And the Bonfire will have an association relationship with the Game map. And the Bonfire also has an action that is “Rest at Firelink Shrine bonfire”, so there is the dependency between Bonfire and Rest.

For the “RESET” features, when the player chooses to rest, the Rest Class (is a type of action so we inherit from Abstract Action class) will implement the Reset Manager that will reset all of the enemies position, health and skill and for the Undead Class, the Reset Manager will remove them from the map and refill Player’s health and the Estus Flask. Therefore, there is the dependency relationship between Reset Manager point to Player, Estus Flask, and all types of enemies class.

Requirement 3: Souls (a.k.a Money)

When looking thoroughly at the code, we recognize that we already have the Attack Action Class, therefore, we just reuse that class for reducing duplicate code. As a result, we treat it as a kill/slay enemies method. As there are three types of enemies (@see enemies class) that can be killed by the Player , there will be three dependencies from Attack Action Class point to them. Also each type of enemy will have a specific soul reward, therefore, each type of enemies will correspond to specific souls reward that the player will gain after killing them. Therefore, when the enemies are killed, it will inform the Souls Rewards Class to increase the number of souls to the Player. As a result, when the Souls Rewards Class gets the information that there are specific enemies that have just been killed, they will increase the specific number of souls to the Player. Therefore, it should be a dependency relationship from Enemies to Souls Rewards Class and from Souls Rewards Class to Player. Also, the Souls Rewards is a type of Soul so it should implement the Soul Class Interface. So there will be an Implementation/Realisation relationship between Souls Rewards Class and Soul class.

Requirement 4: Enemies

In order to prevent the enemy from entering the floor, we need to add a method in Floor class to keep the enemy out of it. We will use HOSTILE_TO_ENEMY status to distinguish the target so that enemies won't attack each other.

To follow the DRY principle, an abstract Enemies class was created which is extended from the Actor class. For different enemies, some of them have the same behaviours like attack and follow the player, walk around, active skills randomly, and these can be created by implementing from behaviour interface. The enemies should have multiple behaviours as attributes. In this way, those behaviours can be reused for different enemies by accessing from abstract enemies class.
However the UniqueBehaviour will only belong to the LordOfCinder class since it is the unique behaviour of Boss. This should be implemented based on Boss's weapon, hitpoints, and locations to achieve the required feature.
The attack behaviour should be implemented based on AttackAction. The SoulsReward class is implemented from Souls interface which will be used to gain souls after the enemy is defeated. After an enemy is defeated, it should be removed from the game map by calling a removeActor method from GameMap class.

All the particular attributes like hit points / maximum hit points, reward souls, locations, and some individual methods like revive, automatic death, and spawn from cemetery will be built in child class respectively since they are not going to share those values and functions.

To answer the question from base codes: According to our design, the LordOfCinder class cannot be an abstract class, because once it becomes an abstract class it cannot be initialized.

Requirement 5: Terrains

Valley and Cemetery are part of terrains so they are extended from Ground abstract class. Once the player walks into the valley they should be killed immediately, therefore valley class should call the hurt method in play class which represents a dependency relation with Player class. Undead will be created around the cemetery, so it can be an attribute of the cemetery.

Requirement 6: Soft reset/ Dying in the game

Playturn method should check if the player is conscious or not, once the player becomes unconscious, the soft reset will be triggered. And the soft reset should heal the hitpoints of the player to maximum, and place the player on to the bonfire. It also needs to call the reset manager to reset all the resettable stuff. Those approaches would be considered using methods so they will be presented as dependencies. Once the player is dead, the souls will be transferred to token souls which is implemented from souls interface and be placed where the player is defeated.

There is a special situation where the player dies from falling in the valley, in this case, the soft reset should check the last action of the player and use the last location of the player to determine the location step behind then put the token of souls on it. (My old design was checking the location of players each turn and once found there was a valley around, then placed a token of souls. New method could be much easier and more efficient. Instead of checking location each turn, now we only need check where the player dead, if there is valley then we one step behind and place the token of souls)
Token souls will be extended from item class so that we can use those methods like setting the capabilities, and I might use a special method to allow players to pick it instead of making it portable because it cannot be dropped except from death.

Requirement 7: Weapons

For the weapons, there are in total 4 weapons in this stage of the game which are broadsword, giant Axe, storm ruler and the Yhorm's great machete. When I was designing this part, I decided to categorize it into Axe and Sword based on the weapon type. In this case there will be 2 more abstract classes extending from the weapon item. The swords will be inherited from the sword abstract class and same situation for the Axe class. We would like to implement the burning ground in the Axe class so as to reduce redundancy of the code
About the active and passive skills, it would be stored into each weapon object as the skills are unique, therefore it will be better to store within the object.
For not dropping weapons when the player died and did something intentionally, we would be setting up methods so as to save the weapon that each actor has and when they died , we would call this method and keep the weapon when the actor respawns.

Requirement 8: Vendor

For the vendor, we first inherit from the actor as it is an actor. After linking it to the actor class, we start looking into the requirements for the vendor.

1) We need to set the vendor in a permanent location , in this case we link it to the location class and set the X,Y coordinate for it.
2) It could trade with players using souls, in this case we will be creating a trade class and this trade class will be inheriting from the souls class so as to get the methods of adding and subtracting souls from players
3) It swaps the weapon for player when the player buys a new weapon, therefore we need to link to the swap action
4) Increase Max Hp is one of the goods sold by the vendor. When we were looking through the code, we saw that there's a method in each player object which allows him to increase the maximum Hp, as the result, we will be calling this method within the player object when he buys the increased maximum Hp.
5) There is a new class called getPurchasedWeapon, this weapon is to allow the vendor to create and get the weapon which the player purchases so as to complete the trade

Storm Ruler Charge Attack
The new functions implemented are as follows. First is remove charge action, this method is to hide the charge option and open the execute action for the storm Ruler charge attack. The set Stun effect is just to have an effect of delaying the boss's movement for a round. The Close attack is to close the option of performing the charging attack and the last method is the open charge action which allows player to charge the storm ruler again