Assignment 1 Design Rationale

Lab 7 Team 3

Jingyun Geng, Tawana Mbaya, Yee Lim

Our design implementation can be split into the following 9 categories and subcategories. Each category will contain details on how and why we have chosen to implement concepts in a particular way.

1. Player and Estus Flask

To display the hitpoints of the character, a new method will be added to the player class that, when called, returns a string of the players name and health in the format (Name 100/100), this method will then be called within the playTurn method and its value will be printed to the console. It is done this way so that the current hit points of the character are updated each turn in case they change.

In order to implement the estus flask. A new class called DrinkEstusFlask needs to be added which extends the action class. This class will have the number of drinks remaining as an attribute as well as methods which will set it's menu description, the starting number of flasks and the character that the player inputs in order to perform the action. In addition to this the class will also contain methods which check if the player is out of estus flasks and notify the user in the case that they attempt to perform the action with 0 flasks left. There will also be a method used to reset the number of flasks to the original amount. The final method in the class will be it's execute() function which when called will consume an estus flask and raise the player hit points by 40% of the maximum hit points. By implementing the estus flask as an action it makes it simple to add to the menu without the need to make many new modifications.

2. Bonfire

The Bonfire is displayed in the game as the letter "B" and is where the player first starts the game (Firelink Shrine). The Bonfire only has one action "Rest at Firelink Shrine Bonfire". Once the player is within the radius of the Bonfire, It will print the options in the console and if the player chooses to rest, it will reset the player's health/hit points to the maximum, refill estus flask to maximum charge and lastly resets the enemies' position, health and skills.

This can be implemented by having a method resetHealth() in the Player class or ResetManager where it will reset the player's health/hit points to its initial health/hit points. Similarly we can create a method resetEstusFlasks() to reset the DrinkEstusFlask count. resetUndead(). resetSkeleton(), resetYhorm() can be implemented in the classes containing the enemies (Undead, skeleton,Yhorm) and their original positions and skills. It will wipe out all undeads from the map. Hence once the "Rest at Firelink Shrine Bonfire" is selected, it will call ResetManager which will contain a method that calls all these methods and it will reset all its features. By implementing in this way, making the bonfire class have an association relationship with the classes of enemies, DrinkEstusFlasks, and Player class. We can avoid having to repeat ourselves in the bonfire class. It also means that each of the classes are responsible for implementing changes to their own attributes which can avoid having multiple classes changing attributes and reduce messiness in the overall implementation.

Classes used: DrinkEstusFlask, ResetManager,Player, Application, Menu, Undead, LordOfCinder

Classes created: Bonfire

3. Souls

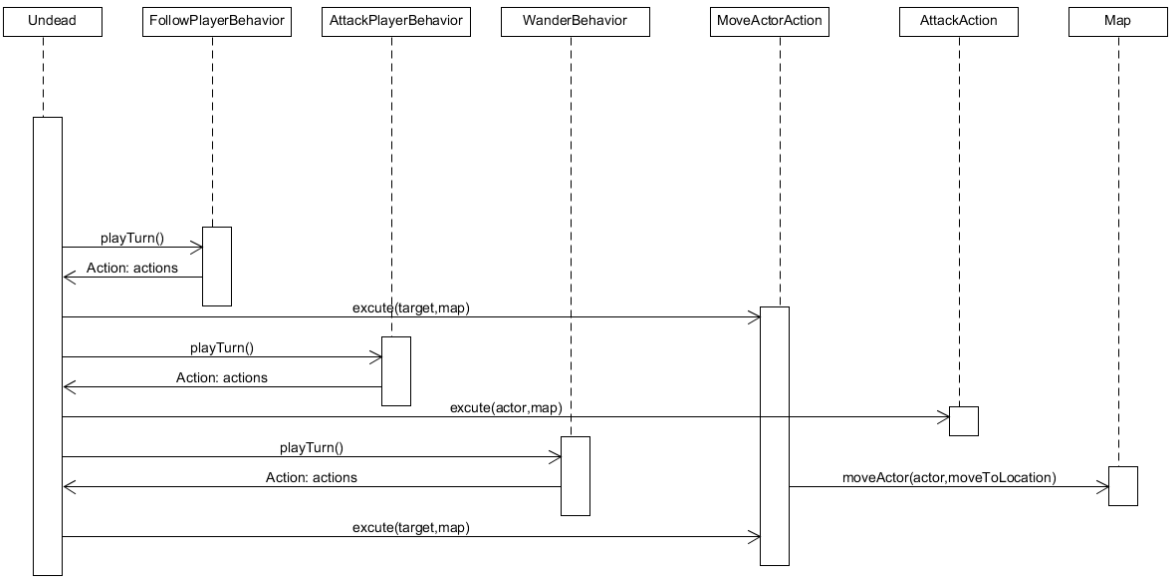
We can make a static final variable updatedSouls (initialised to 0) which contains the number of souls the player has at any given time. We can keep track of this by having a getSouls() method in the class that will return the current number of souls of the player. We will increase the updatedSouls only when the player has slain enemies, hence we can make a method increaseSouls(enemy), when the parameter is which enemy is slain. Then it will increment updatedSouls depending on the enemies. Lastly we will have resetSouls() for when either the player has died or starting over. By making the variables static final, we are avoiding excessive use of literals and not repeating the same integers multiple times.
Interface used: souls
Class used: Player
Class created: Souls

4. Enemies

- Undead

In order to implement the Undead, there already has a class for Undead which extends the Actor class, it already setted up some attributes of the Undead, hit point(50), display character('U'), we will added another attributes 'soul' to this class which is the souls player can get after kill this enemies. The arraylist behavior includes all the behavior the Undead has, that means we will create some behaviors class which implement the Behavior class for the Undead behavior which are DieInstantlyBehavior (10% chance to die instantly every turn), FollowPlayerBehavior (follow the player when it detects a player) and AttackPlayerBehavior. And also will use some current behavior, WanderBehavior. For current action, Undead only can be attacked by the player, we will make attacking the player an allowable action as well. Then we will add a method 'die' for in Undead class, it will give the souls to the player and remove this Undead from the map, but there will be a validation in this method, if the Undead is dead because 10% chance die instantly, it won't give the soul to the player, just remove this Undead from the map.
Class used: Player, Actor, WanderBehavior, AttackAction, Undead
Class created: DieInstantlyBehavior, FollowPlayerBehavior, AttackPlayerBehavior

Below is a sequence diagram shows some behaviors of Undead



- **Skeleton**

In order to implement the Skeleton, we will create a class called 'Skeleton' which extends the Actor class. In this class the attributes will be hit point(50), display character('S'), an arraylist of behavior, soul, weapon, reviveTimes and initial position. Skeleton can randomly get a weapon Broadsword - or Giant Axe, the initial position is for every time when the player reset the game, Skeleton will be reborn at the initial position. For the behaviors, it has some of the same behaviors as Undead, WanderBehavior, AttackPlayerBehavior and FollowPlayerBehavior. This class will also have a method 'Die' but not the same as the method in Undead class, because it will have a 50% chance to heal to maximum hit points, the reviveTimes initial is 0, after revive it will be 1, when reviveTimes ==1 it won't be revive again, then Skeleton truthly died gives 250 soul to player.

Class used: Player, Actor, WanderBehavior, AttackAction, FollowPlayerBehavior, AttackPlayerBehavior

Class created: Skeleton

5. **Yhorm the Giant (Lord of Cinder)**

In order to implement the LordOfCinder, there is already a class called 'LordOfCinder' which extends Actor class. LordOfCinder will have these attributes, hit point, display character('Y'), weapon and behavior, it will have the same behavior of FollowPlayerBehavior (only in the room) and AttackPlayerBehavior. We will create a method to detect the hit point it has, when the hit point is below 50%, it will use the weapon action which will be the skill of weapon Yhorm's Great Machete, at the same time showing a message to the player the giant has used this skill. There also will be a 'Die' method for this class, but different from the other two before, this method will give the player souls and drop a weapon on the map.

Class used: Player, Actor, WanderBehavior, AttackAction, FollowPlayerBehavior, AttackPlayerBehavior, LordOfCinder

6. **Terrains (Valley and Cemetery)**

In order to implement the valley class, first the method canActorEnter() in the class valley must be modified to allow players to enter. Next the playTurn() method within the player class would need to be modified to include a check if the player moved this turn and if so, check the class of Ground the player stepped on. If the class is a valley, then a soft reset is performed. By doing it like this we ensure that code regularly checks the player location and functions properly without the need to modify much of the existing code.

To create a cemetery a new class needs to be created named "Cemetery" which extends the ground class. Within the constructor whe character by which cemeteries can be identified will be determined. The method will also contain a method spawnUndead() which generates a random number from 1-4 and if the number is 2, an undead will be spawned in one of the locations next to the cemetery thereby making it a 25% chance that one is spawned.

7. **Soft Reset/Dying in the game**
   - **Token of souls (extra feature)**

In order to implement a soft reset the player class must be modified to implement the "Resettable" interface, The method resetInstance() then needs to be added which stores the original location of the player with the variable "startLocation". The method will check if the

current player hit points are <=0 using the isConsious() method. If they are, the method will read the current location of the player,reset player health , fill the estus flask charges, and update the players current position to the position of the firelink shrine bonfire.

A new class called TokenOfSouls will also be created which extends PortableItem. The class will have an attribute “souls” which is used to store the number of souls the token holds. Upon death, if the location of the player is not a valley, an instance of the class will be created and the token will be placed in the location that the player died. If the player was in a valley then the location to the right is checked, continuously until a non-valley location is found where the token can then be dropped.

Implementing the soft reset and token of souls in this way is beneficial as it maximises the use of existing code through the use of interfaces and inheritance, thereby reducing the need to repeat ourselves.

8. Weapons

According to the game design engine, weapon is just an interface, which is implemented by the abstract class WeaponItem. The WeaponItem class is associated with the abstract class WeaponAction. In the game, there are 4 kinds of weapons: Broadsword, Giant Axe, Storm Ruler, Yhorm’s Great Machete, these weapons can be used by player and enemies except the Yhorm’s Great Machete and Storm Ruler, they are only used by the Yhorm, and the play respectively .the Effectiveness of weapon is decided by actor, positions of actors in game map,and weapon's abilities.

With provision for the above-mentioned, we design 4 weapon classes: BroadSword, GiantAxe, StormRuler and GreatMachete, they extend the WeaponItem class(abstract). BroadSword class describe BroadSword, which has properties of name(string),Damage(int),SuccessHitRate(long), Price(int) and CriticalStrike(bool), CriticalStrike presents strike mode, When Critical Strike mode is enabled, Broad Sword has a 20% success rate to deal double damage with a normal attack.

GiantAxe class describe Giant Axe, which has properties of name(string),Damage(int), SuccessHitRate(long), Price(int) and SpinAttack(bool), SpinAttack presents Spin attack mode ,Spin attack gives a holder an action to swing the weapon.When it is swung, any enemies that stand in adjacent squares of the holder will receive 50% of this weapon damage (i.e.,25 damage).

StormRuler class describe Storm Ruler,which has properties of name(string),Damage(int),SuccessHitRate(long), Price(int), ChargeTimes(int) and PassiveAttack({Critical Strike,Dullness}), Critical Strike has a 20% success rate to deal double damage with a normal attack. Dullness attacks enemies that are not weak to Storm Ruler will decrease its effectiveness.The damage is reduced by half,but the hit rate remains 60%. StormRuler class poses two actions(operations): Charge() and WindSlash().Charge action enables the holder charge the weapon for three turns to unleash a wind slash; When Wind Slash is executed, it deals 2x damage ‘wind slash’ to Yhorm with a 100% hit rate and stuns it.

GreatMachete class describes Yhorm’s Great Machete, which has properties of Name(string),Damage(int), SuccessHitRate(long) and EmberFormActived(bool). Ember FormActived: The holder becomes aggressive, the holder’s success hit rate is increased by 30% , he burns the surrounding/adjacent squares

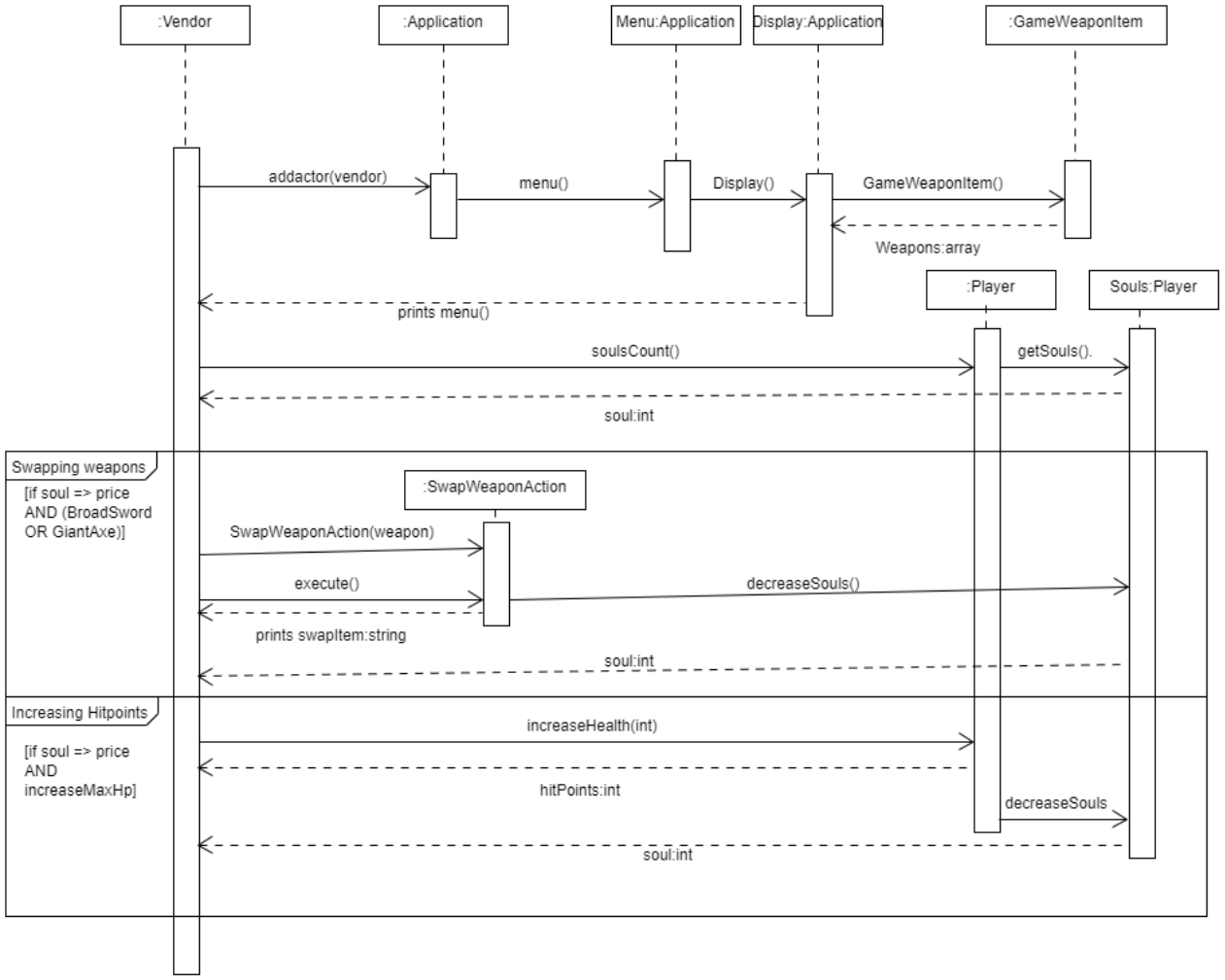
9. Vendor

Souls are to be traded to the vendor to purchase weapons and upgrade the player's attributes. The vendor, called Fire Keeper, will be situated at one location for the entirety of the game, and once the player gets within its radius it will display a menu containing all the options available to be purchased. Using the console, when the player chooses an option, vendor class will call the souls class and if getsouls() > price of the chosen option. The vendor class will replace the weapon of the player by calling SwapWeaponAction class and changing to the chosen weapon with its new abilities and attributes. If the player were to choose the increase Maximum HP option, it will call the player class where it will have an increaseHealth(int number) method. We can create a static final variable increaseHP = 25, and call this method, the player class will increment the players hit points to 25. By making all the integers (prices) static final variables, we can avoid excessive use of literals. Also by calling the player class and weapon class it makes each of the classes responsible for changing their own attributes. Vendor will have an association relationship with SwapWeaponAction, Player, GameWeaponItem and Application. As The Vendor class will create attributes of the classes and require the methods in them.

Class used: SwapWeaponAction, GameWeaponItem, Application (to input location of vendor), display, Menu

New class: Vendor

Below is the Uml sequence diagram for Vendor()



Below is the UML Sequence Diagram for the Soft reset.