FIT2099 Assignment 1

MA_Lab4Team1
Lionie Annabella Wijaya (31316115)
Yeoh Wai Liang (32241674)
Ching Shen Ian (29273617)

Design Rationale

Player and Estus Flask

Application creates a player object named Unkindled represented by "@" with 100 hit points at the start of the game. The Player extends Actor class to inherit all the necessary attributes and methods as a character in game. Now that the Player inherits Actor, he can store things in his inventory. To display the player's HP in the console, in every turn when the showMenu method is called, the player's description method is called to be printed out. We will add two items to the inventory, a Broadsword and an Estus Flask. Both items can be created and added to inventory when Player is constructed or alternatively we can create the items in application and add it after the Player is constructed. Both designs are okay but we decided to go with the first option as it is simpler to implement and will not be confused with creation of other items or components in Application class. The Broadsword is stored inside inventory as the default code has provided a getWeapon method which traverses through inventory to find a possible weapon.

An Estus Flask class is created which extends the abstract class item as Estus Flask is a unique potion that cannot be dropped and it is independent from the player class. Estus Flask has a charge attribute of three, which is always displayed prior to call and will not enable drinking action if charge hits zero. To be able to use the estus flask, Player calls a drinking method on the Estus Flask instance and provides his current HP, then the instance calls the heal method of the Player if charge is not zero. Alternatively, we can create a drinking method action but we ignored this to prevent dependency in case there is another drinkable item in future implementation. Estus Flask also implements a recharge method to fill the charge attribute back to three. To display the option to drink Estus Flask in the console, in every turn when the showMenu method is called, the Estus Flask description is printed out and if selected will call the drinking method.

Bonfire

Application creates a Bonfire object positioned in the middle of map which extends abstract class ground to enable display in game map and provide interaction with player through rest action. When rest action is taken, reset manager is run to reset all instances that implement Resettable interface, allowing reset on attributes, abilities, and location depending on the actor. Reset manager here resets Player by calling the heal method to refill health to maximum on player and calls recharge method on Estus Flask in the player's inventory. It resets Skeleton and LordOfCinder instances position, health, skill, and following behaviour. It however does not reset Undead instances like other enemies, instead all Undead instances will be iterated and removed from the game map. The reset manager run method is designed this way to enable reuse later on for scenarios when the player dies,

such that we can do necessary adjustments according to the dying scenario before resetting and spawning player back to Bonfire. This way we reduce redundancy of creating duplicate methods.

Souls

A player can earn souls by killing enemies, exchange souls with a vendor for some upgrade or purchase, and lose souls which are transferred into an item called token of souls if he dies. Therefore, these classes Player, Enemies, and TokenOfSouls should have some mechanisms to enable exchanging, subtracting, or adding souls which is achieved by implementing the provided Soul interface. The methods in each class are overridden, the player class should be able to transfer, add, and subtract souls but the other classes should only be able to transfer souls as the player is the only one collecting and using the souls. In addition to this, these classes each have attribute souls of type integer to indicate number of souls it holds, having this attribute makes it easier later to do exchanging of souls as when a soul object is passed. Player by default has zero souls and enemies have souls according to the specifications of the number of souls they hold when they die. TokenOfSouls souls depends on the situation later on when the player dies which is explained in the soft reset section.

Enemies

Enemies class is added in which extends the actor class to provide interactions with actors and capabilities. Undead, LordOfCinder, and Skeleton classes extend this Enemies class. Alternatively, we can ignore creating Enemies class and directly extend Undead, LordOfCinder, and Skeleton classes to actor class. We choose to extend it to Enemies class first to provide polymorphism to implement possible requirements applying only to enemies, including adding the status of hostile to the player to avoid enemies attacking each other, adding limitation to locations to prevent enemies entering, and Following behaviour to follow player. For Skeleton and LordOfCinder, we add a method to enable random active skills from weapon equipped. By default, Skeletons and LordOfCinders are created by application at the start of the game and Undeads are spawned by cemeteries placed in the maps (more information about spawning in Terrains). To enable reset later, we need to note the first location where Skeletons and LordCinders instances are placed after creation by adding the initialLocation attribute to these classes which is initialized during construction of instances. To display the enemies' information in the console, in every turn when the showMenu method is called, all alive enemies' description methods are called to be printed out.

Undeads cannot hold any weapon, so unlike Skeleton and LordOfCinder which have WeaponItem in their inventory, Undead class has IntrinsicWeapon instead. To enable undeads walking around aimlessly, Wandering behaviour is implemented. In each turn of game, all Undead instances are checked and if it's Following behaviour is not active then it calls a method to determine a 10% chance of dying, if it dies this way we make sure no exchange of souls takes place and it is immediately removed from the game map. Skeleton carries a random weapon which we can immediately construct when a skeleton is created, we can implement a behaviour to select a random weapon from a list of weapons permitted to be held by skeletons. This will also provide reusability in the future if a child from the actor class needs to select a random weapon. Skeleton class must implement an additional check

whether it has died before or not by a boolean attribute and a method to resurrect itself. If the skeleton dies and it never died before, set the boolean attribute to true then call the method to give a 50% chance of healing to max points. If after calling the method skeleton does not heal or skeleton dies again after dying once checked through the boolean attribute, then soul transfer is enabled and skeleton is removed from the map. LordOfCinder can only equip YhormMachete, so we can immediately construct a YhormMachete instance when LordOfCinder is constructed and add it to it's inventory. When LordOfCinder's HP is less than half, we can implement an ember form capability which will activate the weapon's skill and burn surrounding dirt. The burning ground extends from dirt class and is created for each turn and removed after three turns, the burning ground should also implement the capability to only hurt the Player. When LordOfCinder dies, soul transfer happens and a Lord of Cinder item is created and dropped at the ground where it dies.

Terrains (Valley and Cemetery)

Both valleys and cemeteries are areas in the map and they should be added in as classes which extend from the Ground class to provide display, capabilities, and interactions with actors. By default, both these areas are present in the map from the start of the game thus Application should create several of them. Valley class overrides methods to guarantee only a player can step on them and we can add a method that reduces player's HP to zero if the player did step on the valley. Alternatively, we can instead add a status that player's fallen when valley is stepped then perform a capability check in each turn of the game. We picked the first option to reduce redundancy as in each game turn we still need to check if a player died from being attacked anyway, thus using the first option we can eliminate extra checks and reuse the check available.

While the game is running, in each process of the player's turn, cemeteries may randomly spawn an undead. A method can be created inside the Cemetery class that creates and returns an Undead object or null by the given probability, and this method is invoked for each cemetery in each turn inside the World class. If successful and return is not null, the World class will add the Undead inside the game map. Alternatively, we can calculate probability beforehand in the World class for each cemetery to determine if an Undead is created and construct an Undead by calling a method in Cemetery that always returns an Undead instance if the boolean value is true. We decided to go with first option as it is more clear and we can separate the dependency of spawning Undeads with the World class as it can be confusing if in the future refactoring is needed such as if the criteria of spawning Undead changes, we will need to refactor the World class which is not efficient therefore mechanism of spawning Undead should be kept only in the Cemetery class.

Soft reset/Dying in the game

When the player dies then everything in the game will be reset and the player will respawn at the latest bonfire. We can make use of the implementation of the reset manager described earlier in Bonfire for the reset part. After reset, we need to take care of creating Token of Souls and respawning the player to Bonfire. In order to create TokenOfSouls, players must die and the location of death must be known. Therefore, at each turn before any action is taken, the player's location must be recorded in advance to provide location to create the token. This also solves the problem if the player dies in a valley as the last

location where the player is standing has been recorded. When a player dies, we check if Token of Souls exists in the game map, if it exists then the player has died before and the token is removed as only one token should be available at a game map. Afterwards, the Player creates an instance of TokenOfSouls and transfers his number of souls to the token before adding the token at their last dying location which stays in the map even when the world class gets reset. If the player interacts with the token, the token adds it's number of souls to the player and gets removed from the map. Only after resetting and creating a token of souls, then we move the player to the bonfire.

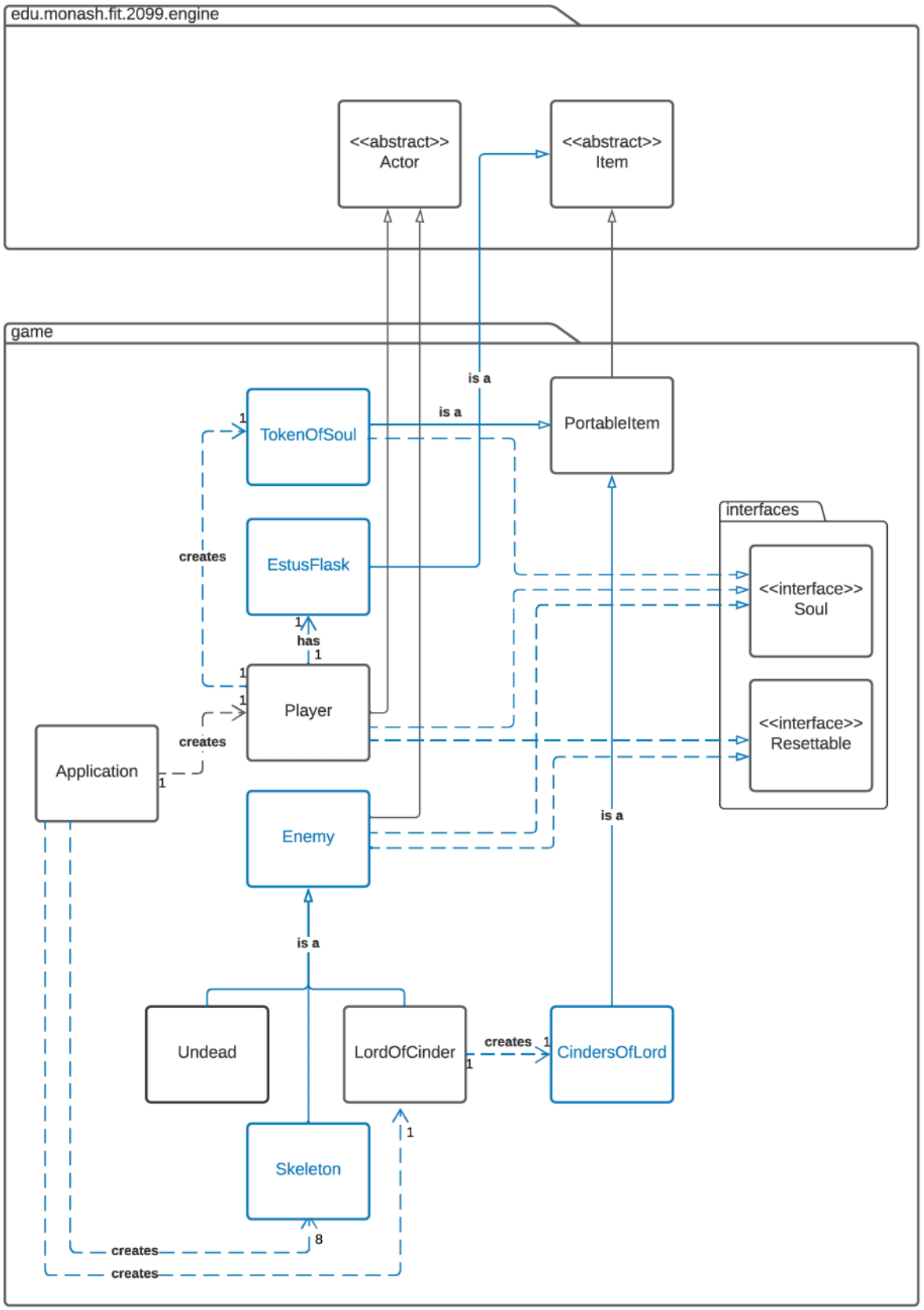
Weapon

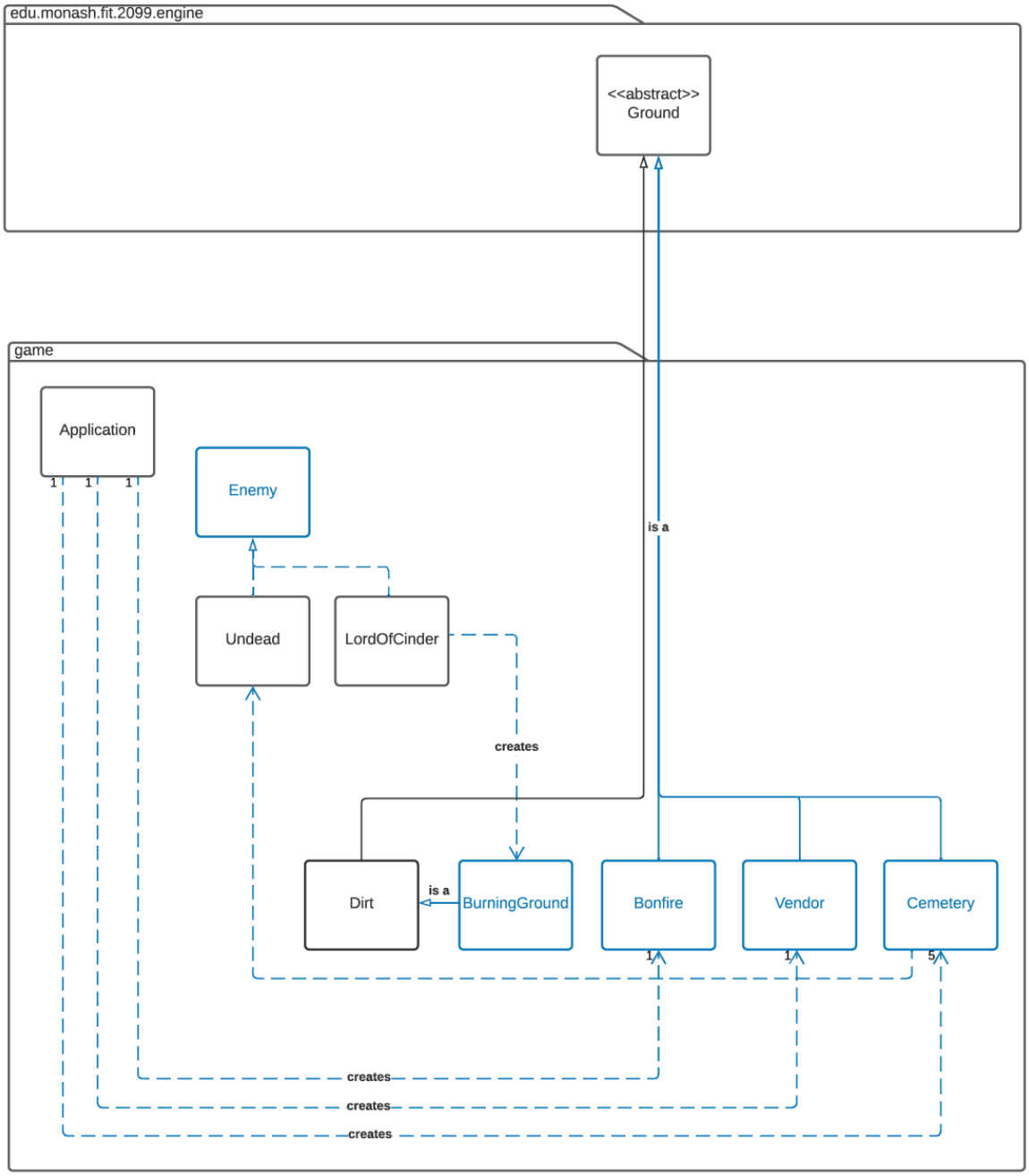
Only a player, skeletons, and lord of cinder can carry a weapon. Each weapon will have its own class for instance, Broadsword class, GiantAxe class, StormRuler class, and YhormMachete class so a total of 4 classes is created. All the weapons extend the GameWeaponItem class. These classes are created as each weapon will have its own unique active and passive skills and some of the weapons can only be purchased from the Vendor. Therefore, some methods will be created in each class to implement their unique passive and active skills. Besides, each weapon will have different price and success hit rate and damage that also extends the WeaponItem class. There are only 2 weapons that can be purchased from the vendor which are Broadsword and GiantAxe, thus the Vendor can create these classes. StormRuler is the only weapon that can be equipped by the player that is not available in Vendor and it is placed next to Yhorm the Giant, thus the application creates an instance of StormRuler that is placed inside the game map.

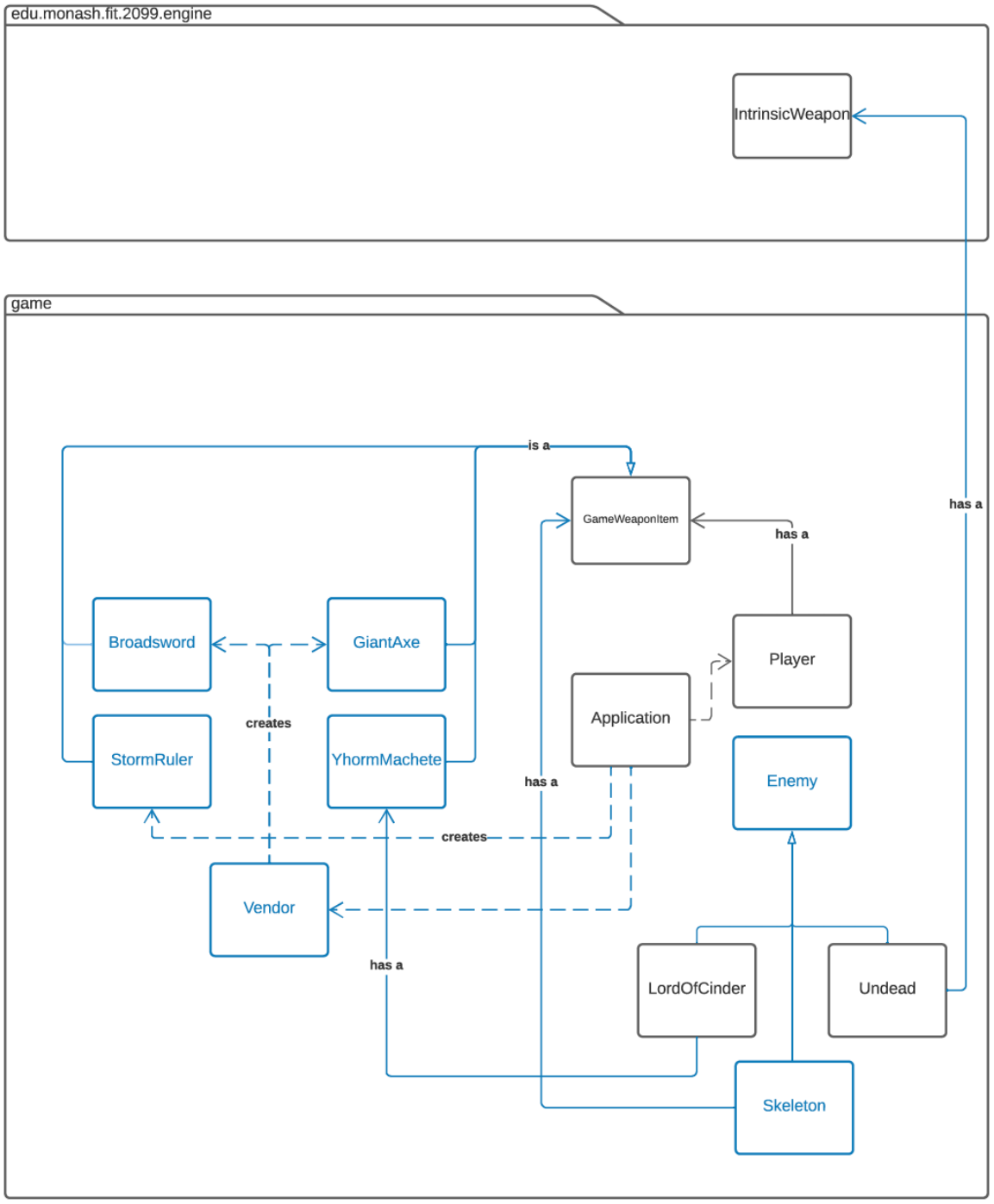
Vendor

Applications creates a Vendor instance named Fire Keeper at the start of the game positioned in the middle of map which extends abstract class ground to enable display in game map and provide interaction with player through trading souls. The Vendor has several methods to provide purchase options (buy Broadsword, buy Giant Axe, and increase maximum HP). For each option, the player's number of souls is checked to determine if the transaction is successful and if souls are enough, the purchase option continues either by creating the weapon object and swapping it with current weapon in the player's inventory or invoking player's increaseMaxHp method. Buying weapons forces the Vendor to create an object instead of having it available beforehand as it is possible for the Player to buy the same weapon more than one time depending on conditions. Successful transaction will subtract the number of souls owned by the player and return the weapon back to the player in which player invokes SwapWeaponAction to swap the current weapon. At each transaction, transaction status is printed out including the purchase option.

Class Diagram

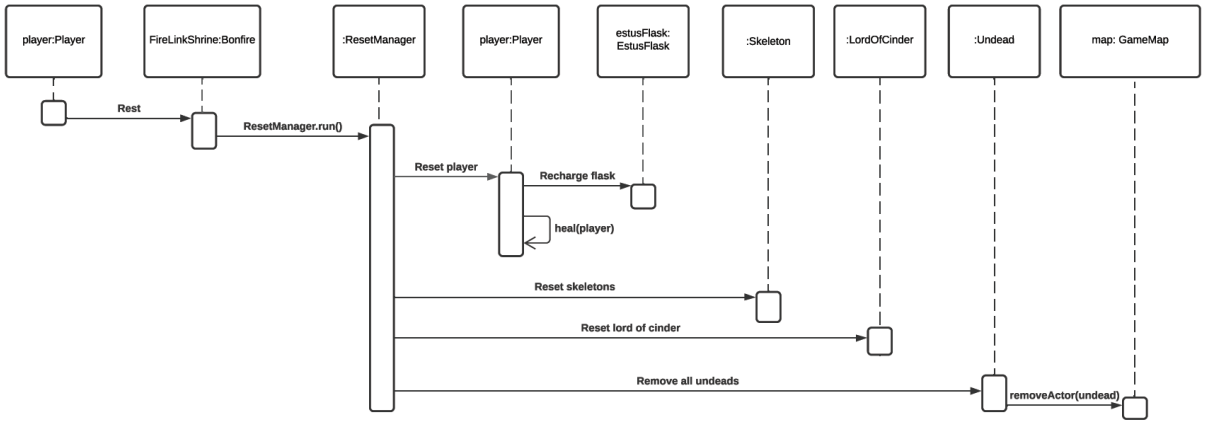

edu.monash.fit.2099.engine
<<abstract>>
Actor
<<abstract>>
Item
game
is a
is a
TokenOfSoul
PortableItem
interfaces
creates
EstusFlask
<<interface>>
Soul
1
has
1
1
1
Player
<<interface>>
Resettable
creates
Application
1
Enemy
is a
is a
Undead
LordOfCinder
creates
1
1
CindersOfLord
1
Skeleton
8
creates
creates


edu.monash.fit.2099.engine
<<abstract>>
Ground
game
Application
Enemy
Undead
LordOfCinder
Dirt
BurningGround
Bonfire
Vendor
Cemetery
is a
is a
creates
creates
creates
creates
1
1
1
1
1
5

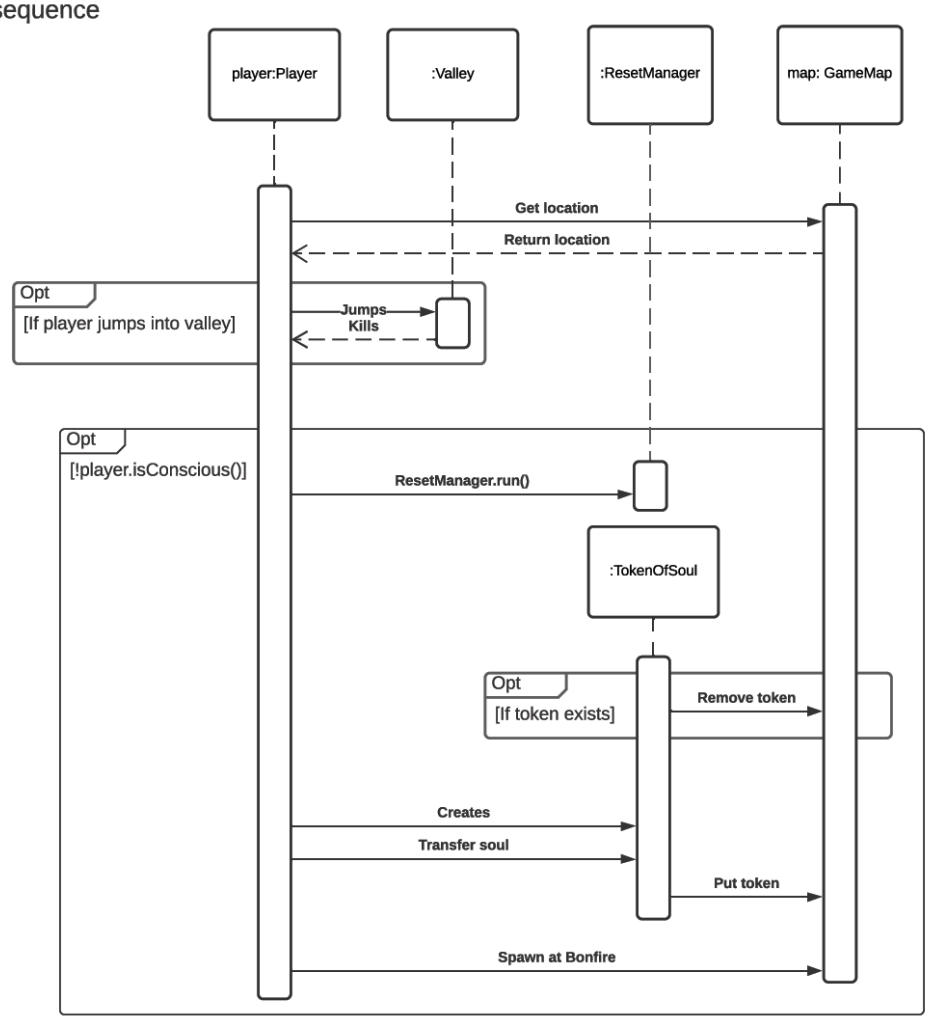

edu.monash.fit.2099.engine
IntrinsicWeapon
game
is a
GameWeaponItem
has a
has a
Broadsword
GiantAxe
Player
creates
Application
StormRuler
YhormMachete
Enemy
has a
creates
Vendor
has a
LordOfCinder
Undead
Skeleton

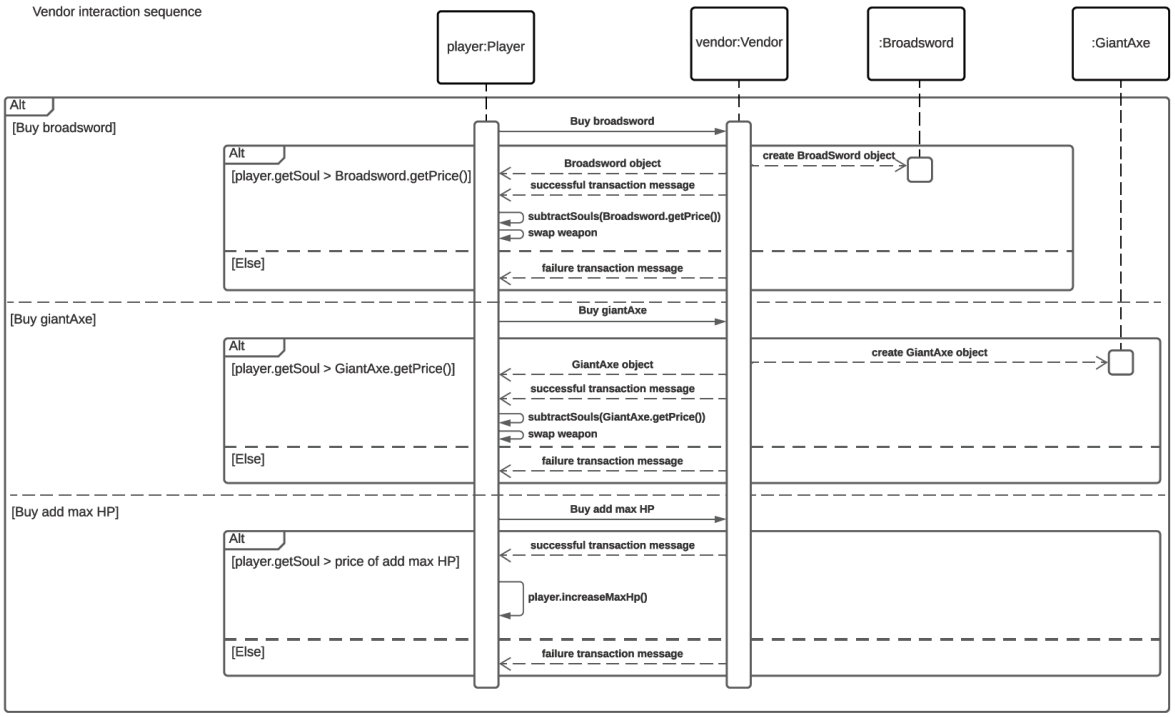
Interaction Diagram

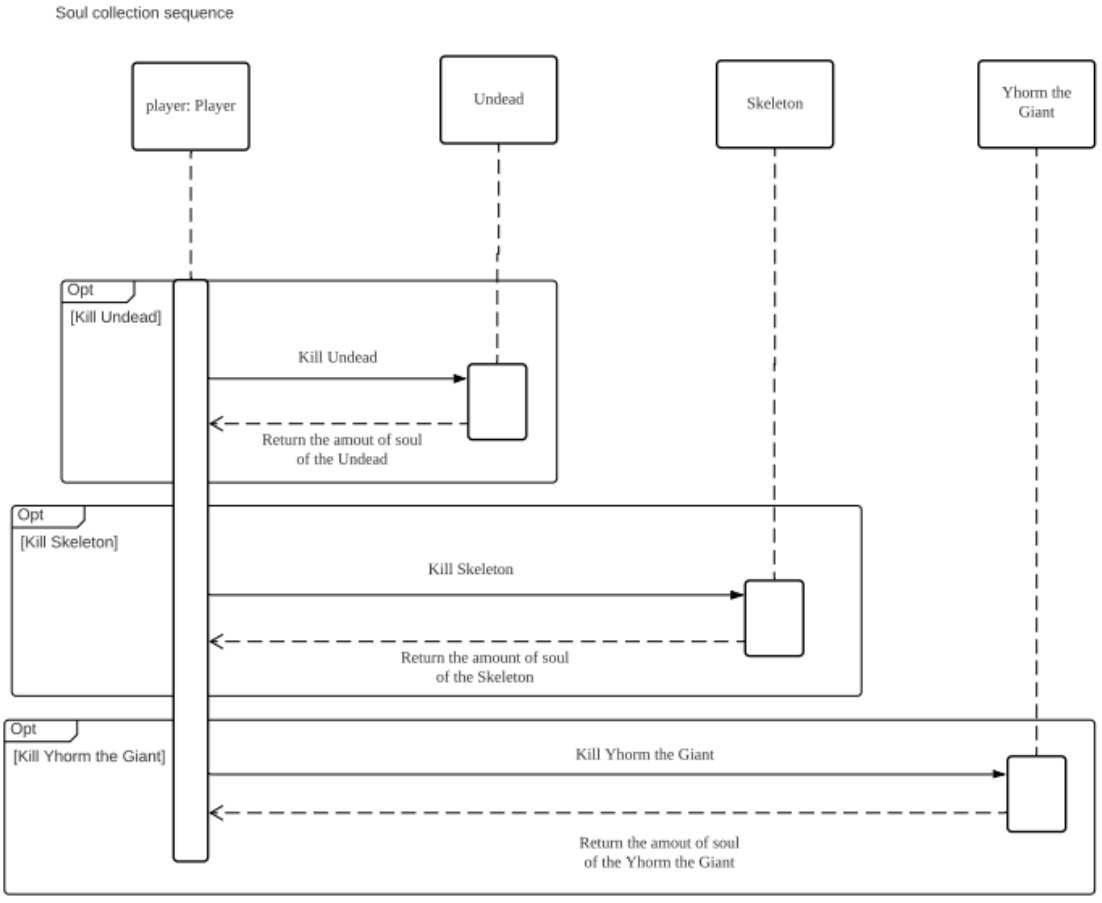
Player resting at bonfire sequence



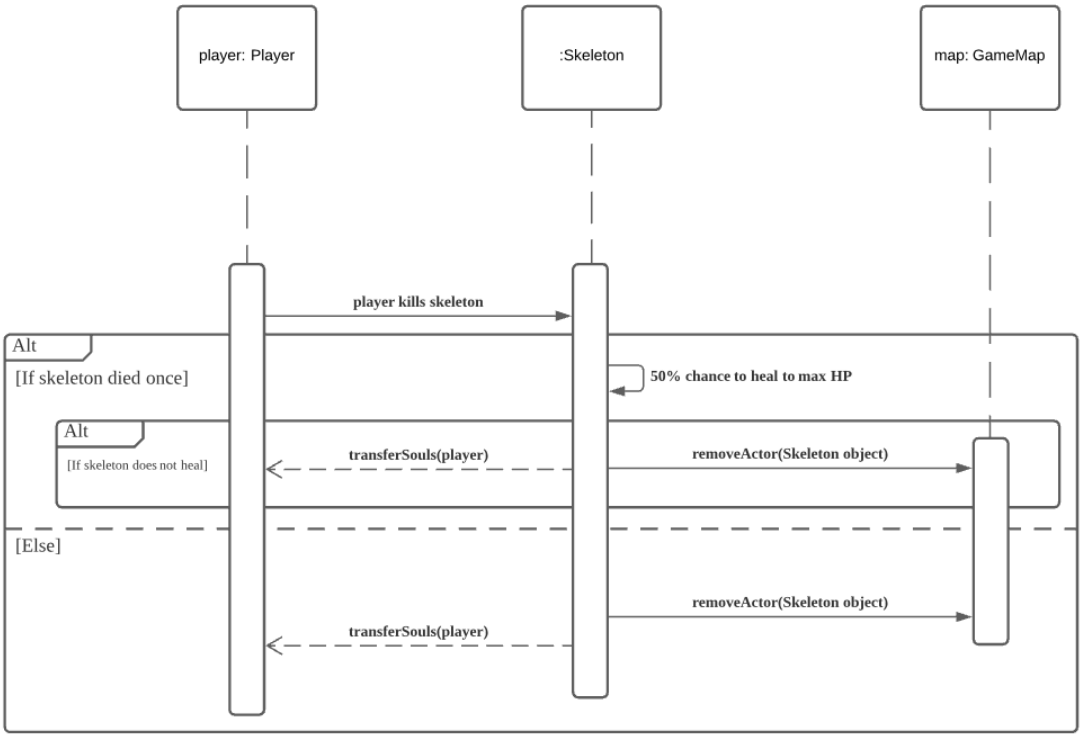
Player die / soft reset sequence




Vendor interaction sequence
player:Player
vendor:Vendor
:Broadsword
:GiantAxe
Alt
[Buy broadsword]
Buy broadsword
Alt
[player.getSoul > Broadsword.getPrice()]
Broadsword object
create BroadSword object
successful transaction message
subtractSouls(Broadsword.getPrice())
swap weapon
[Else]
failure transaction message
[Buy giantAxe]
Buy giantAxe
Alt
[player.getSoul > GiantAxe.getPrice()]
GiantAxe object
create GiantAxe object
successful transaction message
subtractSouls(GiantAxe.getPrice())
swap weapon
[Else]
failure transaction message
[Buy add max HP]
Buy add max HP
Alt
[player.getSoul > price of add max HP]
successful transaction message
player.increaseMaxHp()
[Else]
failure transaction message


Soul collection sequence
player: Player
Undead
Skeleton
Yhorm the Giant
Opt
[Kill Undead]
Kill Undead
Return the amout of soul of the Undead
Opt
[Kill Skeleton]
Kill Skeleton
Return the amount of soul of the Skeleton
Opt
[Kill Yhorm the Giant]
Kill Yhorm the Giant
Return the amout of soul of the Yhorm the Giant

Skeleton resurrects



Spawning undeads in cemetry