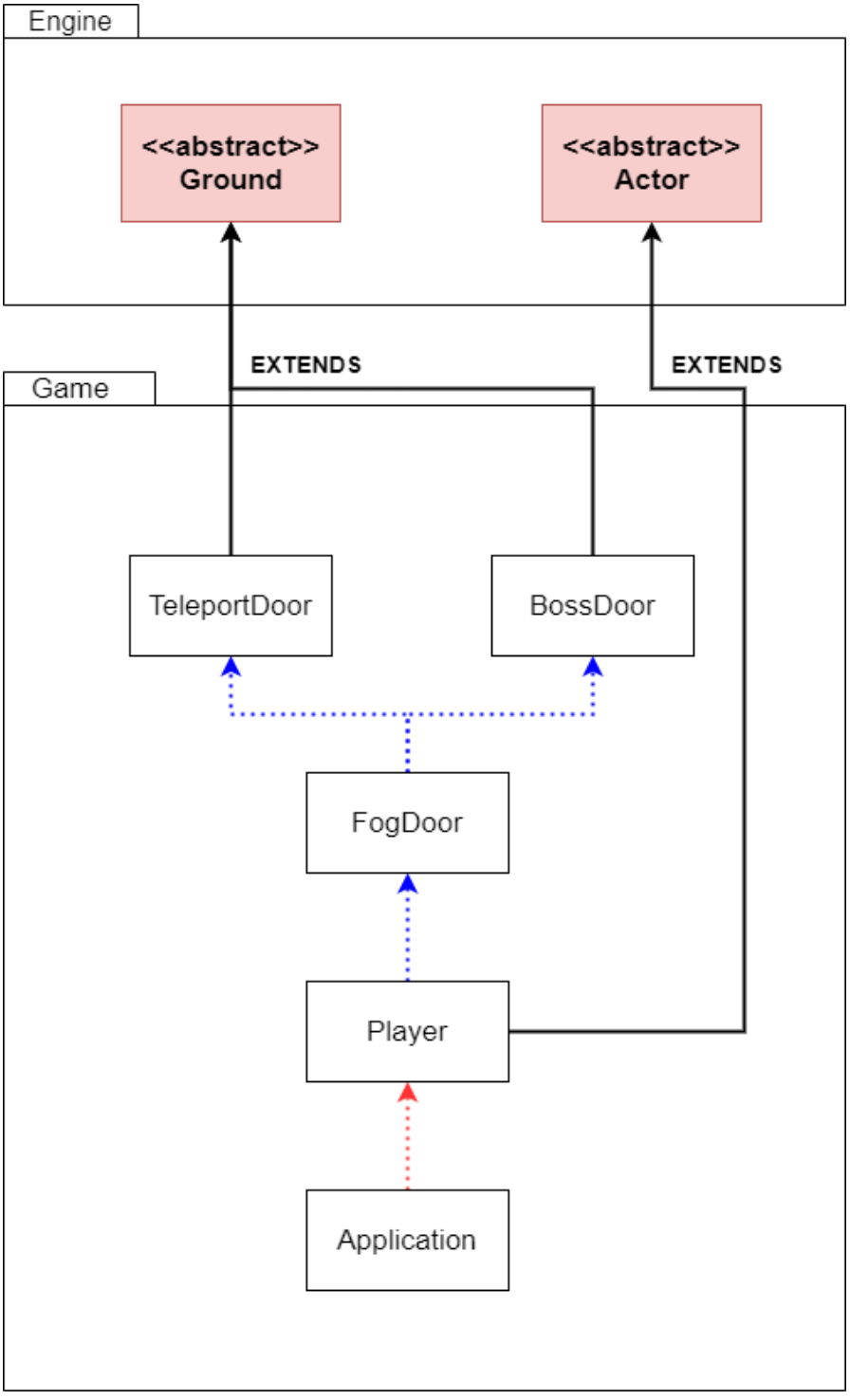
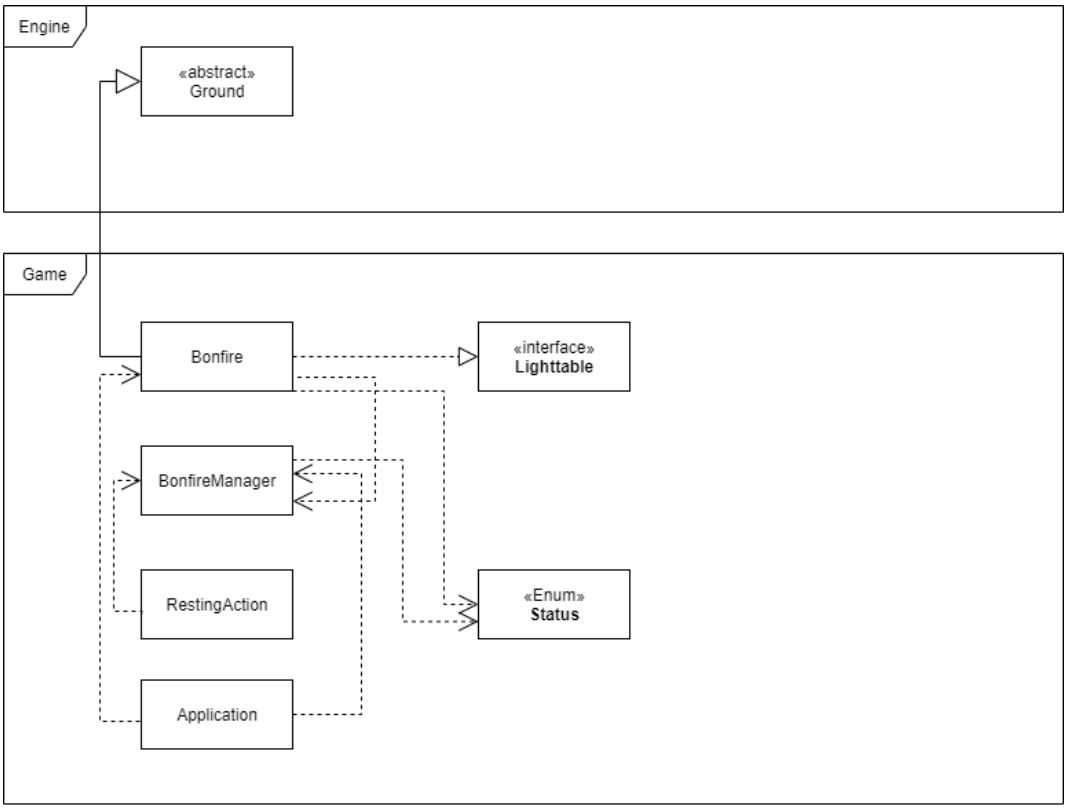
Design Document Assignment 3

UMLs

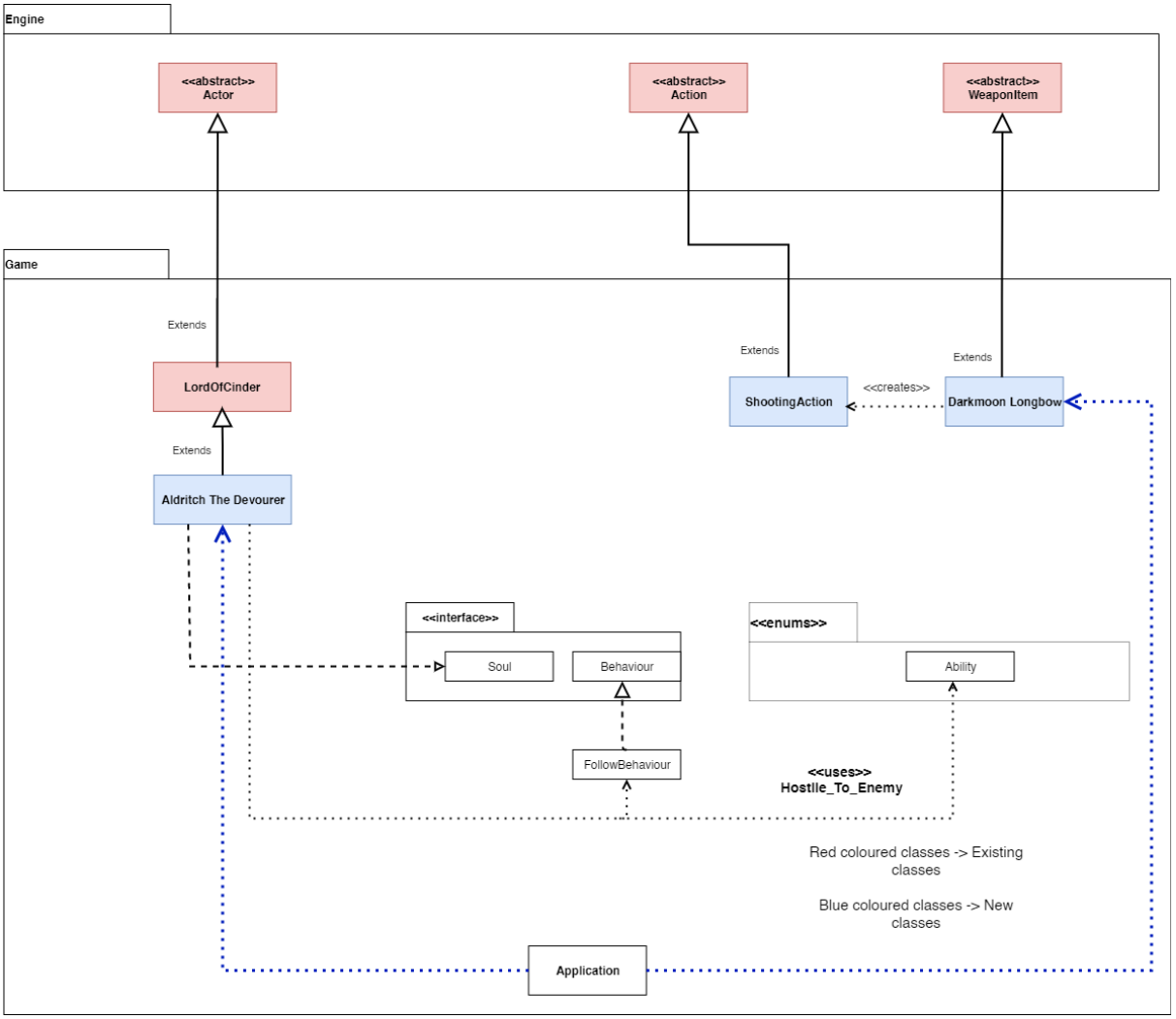
Requirement 1



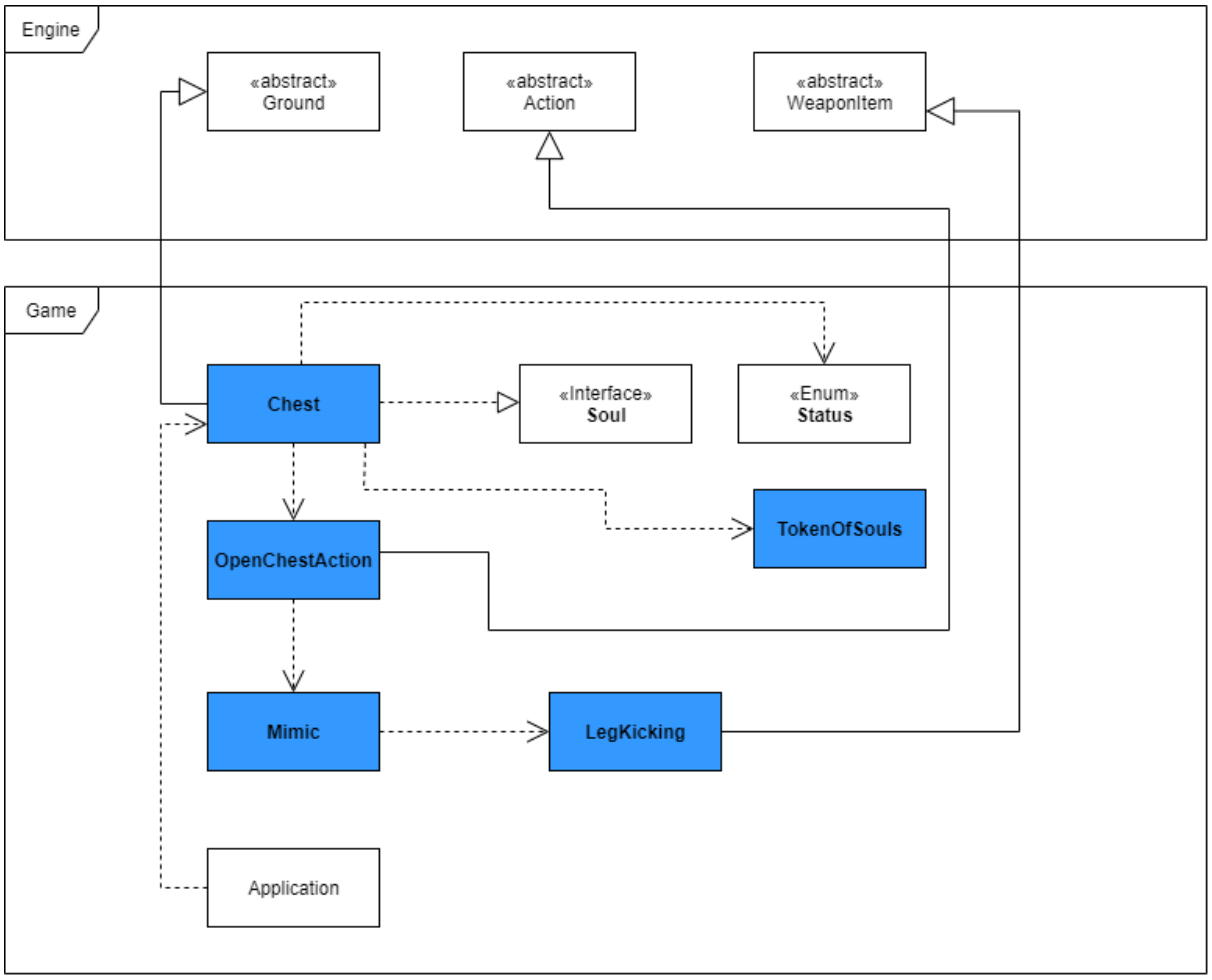
Requirement 2



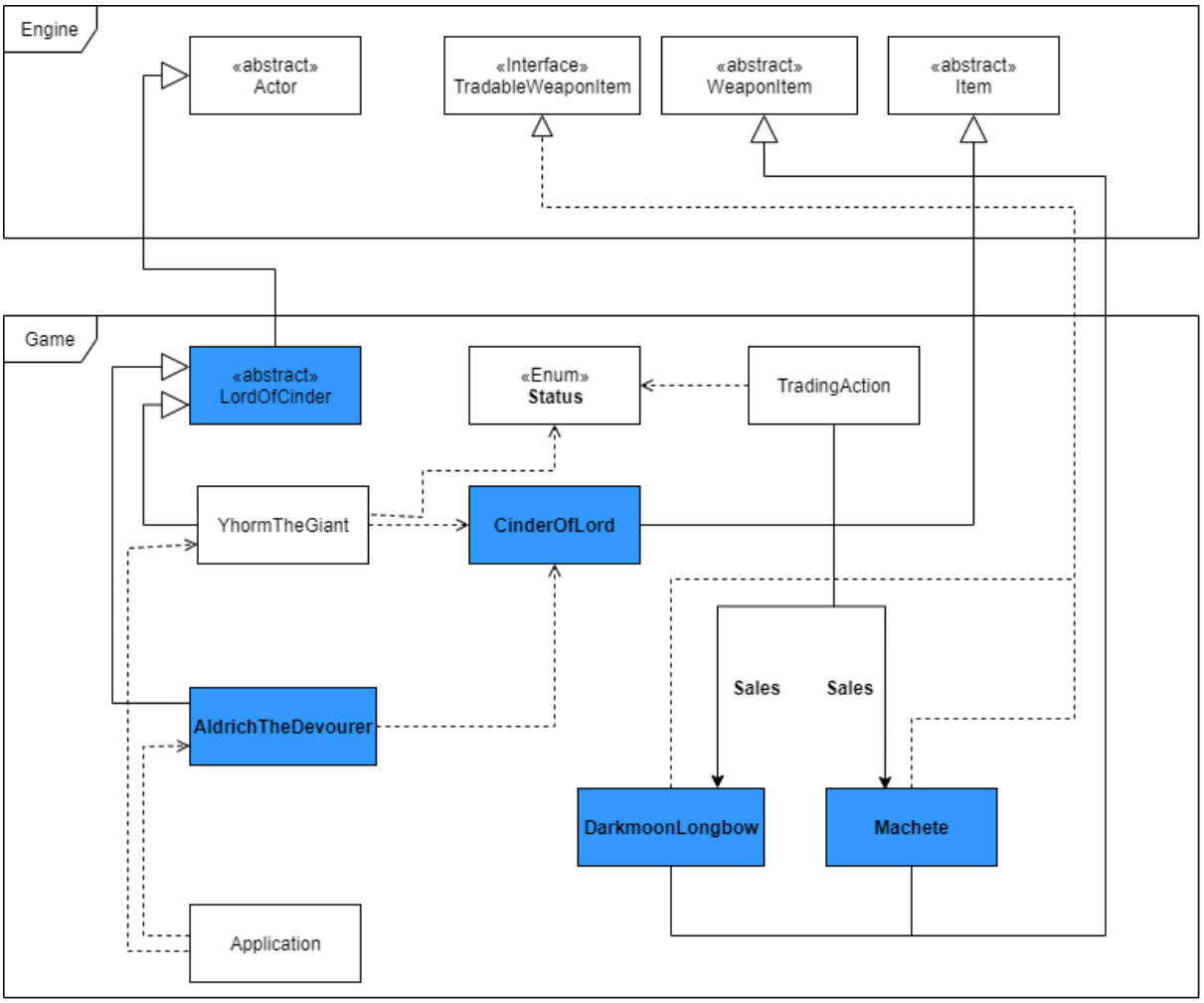
Requirement 3



Requirement 4



Requirement 5



Rationale

Requirement 1

For this requirement, A new map is added to the game alongside a new feature which is called ***Fog Door***. The ***Fog Door*** has 2 different functions, one, which teleports the player into another map and two, which closes the Boss Room so the Boss will stay inside as the third requirement said. For the ***Fog Door*** that teleport the player a new class extending the ***Fog Door*** is needed which will be called ***TeleportDoor***. While the Boss Room ***Fog Door*** will not be having any other new function except keeping the Boss inside so we can either make a new class called ***BossDoor*** extended from ***Fog Door*** but having nothing inside with only a constructor. As the ***TeleportDoor*** has different function than a ***BossDoor*** which was 'Teleporting' player to the other map on the topmost y-axis, so the constructor will be different. The constructor of the ***TeleportDoor*** will consist of a ***map.Destination*** and ***map.Origin***. Inside the ***Application*** Class, ***Fog Doors*** need to be initiated. Once the player entered the ***TeleportDoor*** the player will enter the other map and the presence of the player at the previous map is gone and a new map will be loaded in.

Since this new feature is something that is implemented in the map with the extension from the Ground class from the engine, we make sure that the location of the ***FogDoor*** has to be accessible by the player.

Requirement 2

In the extended version of '*Design O' Souls*', the Bonfire in the game has been updated with several functionalities. For starters, the Bonfire allows players to interact with now (i.e. Light the Bonfire up). Instead of adding a new method which interacts with the Bonfire, an interface namely '***Lighttable***' is created and is implemented by the Bonfire, so that a possible ***Interface Segregation Principle*** is applied for future extension's maintainability sake. Hence, an empty arrow head with a dotted line tailing is placed between the '*Bonfire'* class and the '*Lighttable*' interface class.

Furthermore, the Bonfire is also able to keep track of whether itself is the last interacted Bonfire as well as a new Bonfire is added to a new map (i.e. Anor Londo). Thus, instead of creating a new method class instance variable which keeps track of whether it is the last interacted Bonfire, an enumerate has been created, namely "***Status.LAST_INTERACTED***". This enumerate would be added to the capabilities of one of the Bonfires (i.e. addCapabilities()) when the player rests at one. Subsequently, this also explains the reason behind the dependency relationship between the '*Status*' class and '*Bonfire*' class in the UML diagram.

Moreover, the Bonfires should be implemented in such a way that whichever Bonfire is last interacted with, the player would respawn at that place. Being said, a manager class that manages the instances of Bonfire is created, namely **'*BonfireManager*'**. The BonfireManager would be implemented with an hashmap class variable which keeps account of the name of the Bonfires and their respective locations. The initialisation of these instances would be done in the '*Application*' class via a method in '*BonfireManger*' class,

namely **'*appendRestableInstance*'**, which puts a key-value pair (or Bonfire name and location pair) data into the mentioned hashmap. The '*Player*' class would also be dependent on the '*BonfireManager*' class (i.e. Dependency arrow) to move the player to a new location when died in game. To further clarify the previous statement, a method in the '*BonfireManager*' class is created, namely **'*getLastInteractedBonfire*'** to get the location of the last interacted Bonfire. The Bonfire should also be set to be the last interacted Bonfire when rested at or lit. With that being said, a method in the '*BonfireManager*' class is created, namely **'*setLastInteractedBonfire*'** to apply the "*Status.LAST_INTERACTED*" capability to the Bonfire. The method is used in both the classes '*Bonfire*' and '*RestingAction*' to set the last interacted Bonfires (which again, explains the dependency relationship among these classes with the '*BonfireManager*' class)

Requirement 3

For requirement 3,we will add a new boss to the new map we assigned to make.The difference between both are the weapons they use with the new one using a ranged weapon instead.To make things easier,the structure is changed where a **Lord Of Cinder** abstract class is make that will have Yhorm the Giant from Assignment 1 and Aldrich the new boss be extensions from the abstract class.This allows us to reuse methods from the abstract class to make even more boss classes in the future rather than making all boss classes with their own separate methods.Furthermore,the boss will be given a Longbow which would be its default weapon.In order for the Longbow to function,a shootingAction class is made where it would be able to attack enemies/actors from a given fixed distance.Not only that,having a separate action class allows us to make other ranged weapons in the future that can also use the shootingAction class which gives much more flexibility and reusability.

Most of the standard things such as soul, and having hostile to enemy ability are also implemented on the Aldrich boss with the extra inclusion of the followBehaviour.We reuse the engine code,most notably being able to detect the possible paths to detect the player where if so,will add the followBehaviour to the Aldrich allowing it to follow the player when it sees it within a 3 path radius.

Requirement 4

A class that represents the chest is created. It was decided to be an extension of Ground class so that *Actor* instances can't go through (i.e. canActorEnter method is implemented to return False).

Furthermore, an OpenChestAction class is created to be initialised into the mentioned *Chest* class (i.e. Dependency relationship in UML), consequently allowing it to have an allowable action (i.e. allowableActions method). A *Random* class is also initialised so that it can be decided in the action class itself that whether the Chest class being interacted with will transform into a Mimic or drop token of souls (i.e. Dependency relationship between TokenOfSouls and OpenChestAction). No matter the result (i.e.

Transform or drop souls), the ground would be set to its original ground (i.e. setGround method).

Finally, a Mimic class was created that would attack the player (i.e. Actors with a 'MINDFUL_ENTITY' status) with a LegKicking WeaponItem class. This class was initialised into the item of the Mimics when they were spawned.

Requirement 5

After a new enemy was included in the extension (i.e. Aldrich the Devourer), the lords of Cinder (i.e. Aldrich the Devourer and Yhorm the Giant) was added to the game with an new Item extended class initialised (i.e. CinderOfLord). This item was also added with a new capability (i.e. Abilities.TRADE_CINDER_OF_YHORM and Abilities.TRADE_CINDER_OF_ALDRICH). This also explains the dependency between the Abilities Enum class and CinderOfLord class. To differentiate the different Cinders from both entities, CinderOfLord class was initialised in their respective classes with the mentioned Abilities added (i.e. addCapabilities method).

Moreover, the TradingAction class was updated as well to go through the player's (i.e. Actors with a 'MINDFUL_ENTITY' status) inventory to check whether the player has an item with one of them Abilities added. Thus, the TradingAction would only initialise the actions for which the item necessary to be traded with is in the player's inventory. Finally, the weapons which the respective Lord of Cinder holds (i.e. Darkmoon Longbow and Machete) are initialised into the TradingAction class (i.e. Simple Association relationship between TradingAction and DarkmoonLongbow as well as Machete)

Sequence Diagram