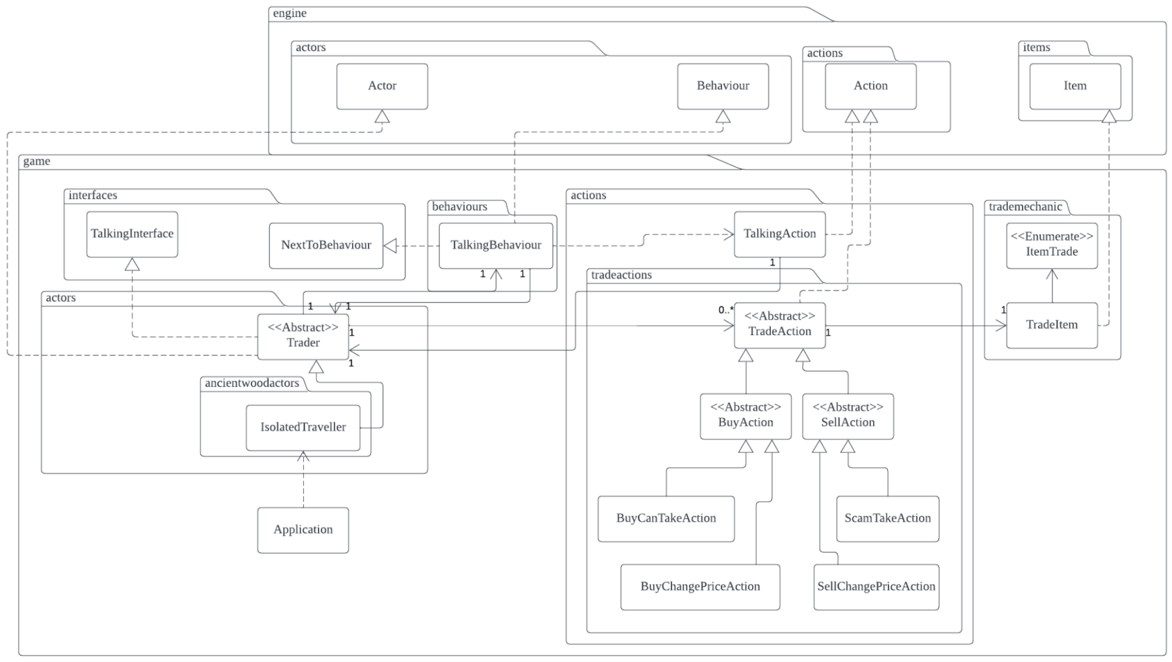
Assignment 2 question 3



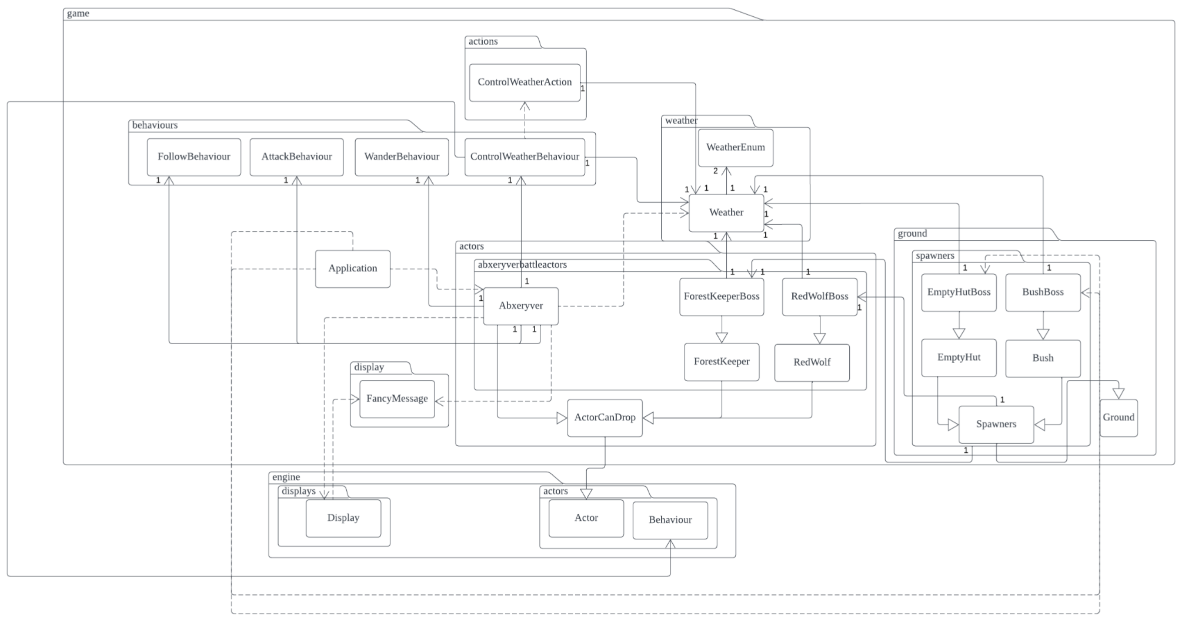
The trader also checks if the player is closeby to start talking to them. The trader has a TalkingBehaviour to help check if the player is too far away from the actor. Doing such an implementation will allow any actor to check if the actor is too far away from the player. The TalkingInterface is used for actors to check if the actor is talking or not.

Trader is an abstract class so that it will be easy to implement another trader simply by extending from trader. As a result, to implement the trader, someone can add the trades and implement any behaviour.

Trader also maintains dependency inversion because trade action depends on this abstract class and the lower modules extend from this abstract class. This helps due to the fact that trades may have different actions, if statements are avoided by storing the trade actions instead of the item.

To maintain single responsibility, each possible trade action is split into extending from a buy action or a sell action which extends from trade action. A trade action contains the item TradeItem which contains the original item. TradeItem is used because the original item class does not have a way to obtain the copy of the item, to check if the player has the right item to sell and to add a capability that the item can be traded.

Assignment 2 question 5



Since ForestKeeper and RedWolf are explained in assignment 2 question 1 design rationale, to make it less clustered, the arrow to the behaviour is not shown.

A new class called weather is created and stores a hashmap of WeatherEnum as a key and a boolean as the returned value. This will make it easier to check which weather it is currently.

The boss, Abxeryver, controls the weather through the ControlWeatherBehaviour. Every 3 turns, the behaviour will return a ControlWeatherAction which will randomly choose a different weather from the current weather. The boss extends from ActorCanDrop to enable the boss to drop items if it is unconscious.

The classes that would get affected by the weather are called [class]Boss instead. The reason being is to follow O and L in SOLID principles because it is extending from the original classes as well as substituting a super class with the sub class. Therefore I can change the sub classes without modifying the super classes.

How these boss classes work is that since they all store the same weather, they can check from the hashmap what the current weather is and change behaviour if needed.