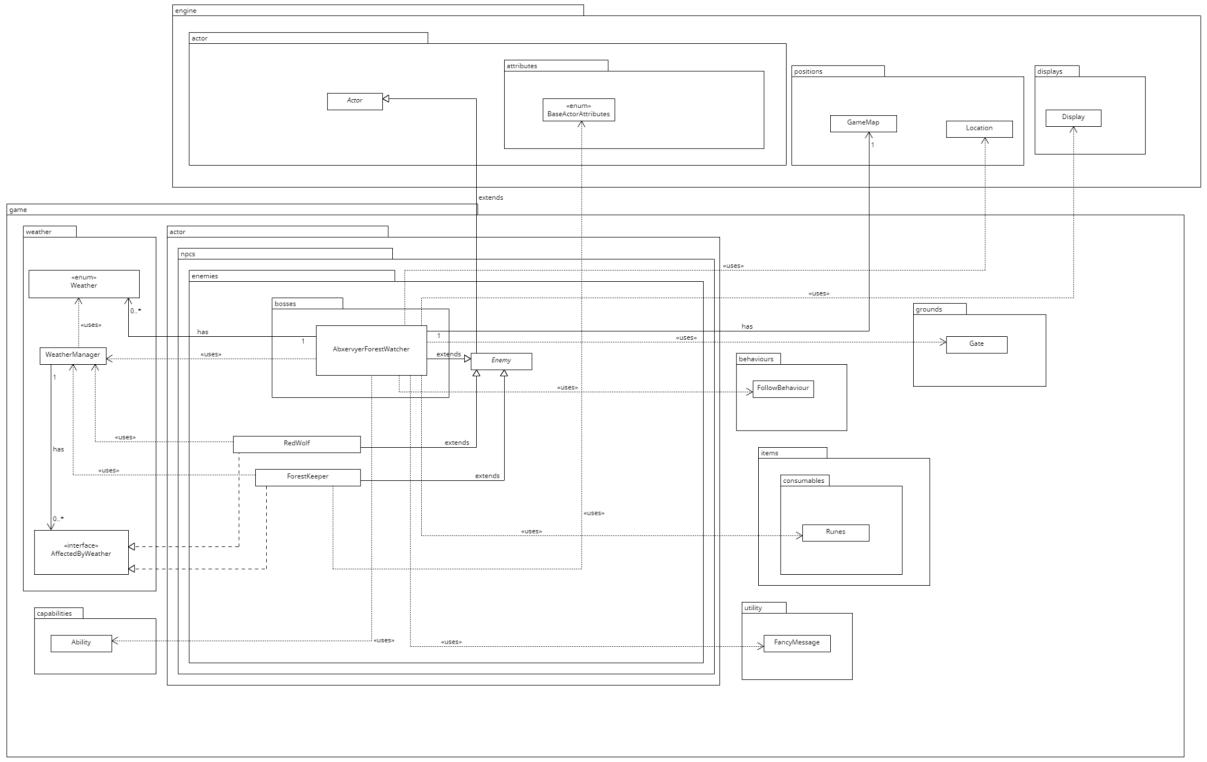

The main objective for REQ5 is to implement the boss “Abxervyer, The Forest Watcher”, and his ability to control the weather, which affects the attributes of the enemies in Ancient Woods as well as in the boss map.

Our original approach was different from this, however after some deliberation, we decided to change it. In the end, we decided to have an “AffectedByWeather” interface that is only implemented by enemies that are affected by the change of weather, in this Assignment, only RedWolf and ForestKeeper are affected. Inside the “AffectedByWeather” interface, we have “sunnyWeatherModifications” and “rainyWeatherModifications”. This adheres to the Interface Segregation Principle (ISP) since we do not force other enemies that are not affected by weather to implement these methods.

We also have a ‘WeatherManager’ class, this class has the attributes of a list of ‘AffectedByWeather’ entities and the current weather. This class also has static methods like ‘addAffectedByWeather’, ‘executeWeatherModifications’ and ‘setWeather’. We call ‘addAffectedByWeather’ inside the constructor of RedWolf and ForestKeeper to add them to the list of ‘AffectedByWeather’ entities. Inside the ‘executeWeatherModifications’ method, we loop through the list of ‘AffectedByWeather’ entities and call their “sunnyWeatherModifications” or “rainyWeatherModifications”, depending on the current weather. The ‘setWeather’ accepts a ‘Weather’ Enum, and sets that as the current weather. We call ‘executeWeatherModifications’ and ‘setWeather’ inside the playTurn of Abxervyer, since the weather only changes as long as Abxervyer is alive.

By doing this, we adhere to Single Responsibility Principle (SRP), as ‘WeatherManager’ is only responsible for managing the weather of the game.

Below is our original implementation (text coloured blue):
Original approach:
We split this requirement into 3 main problems:
1. Modifying the attributes of the enemies
2. Informing Maps about the weather changes made by Abxervyer
3. Informing the entities inside the map about the weather changes of the map

To solve the first problem, we created 2 interfaces 'AffectedByRainyWeather' and 'AffectedBySunnyWeather', which have 'sunnyWeatherModifications' and 'rainyWeatherModifications', these methods are the methods that will modify the attributes of the enemies. We have the enemies that are affected by weather implement these interfaces, and call these methods inside the PlayTurn of the enemies in order for the modifications to take place. This implementation adheres to the Interface Segregation Principle (ISP) as only enemies that are affected by the weather need to implement these interfaces, other enemies do not need to be burdened by methods they don't need.
Solving the second problem was much harder. We quickly realised the main hurdle of the requirement is due to GameMap not having 'CapabilitySet' like other GameEntities', this makes it difficult to keep track of the current weather of the map. As such we decided to go for the next best thing, adding the weather Enum Status to Ground instead. Since Grounds make up the map, this effectively makes the entire map have the weather status without needing to modify the engine code.

We decided to have an abstract 'WeatherMap' class that both 'AncientWoodsMap' and 'AbxervyerMap' extend. Abstract 'WeatherMap' class has a method 'setWeather' which accepts an Enum Status, this is the weather Status we want to set the map to, the method loops through all the Grounds inside the map and removes any current weather Status, and replaces it with the weather Status we passed through as an argument.

Abxervyer's constructor takes in the argument of a list of 'WeatherMaps', this represents the maps he has control over. Abxervyer also has a list of Enum Status, which represents the list of weathers he can toggle between. Inside the PlayTurn of Abxeryer, we have a loop that only runs every 3 turns, this loop goes through the list of 'WeatherMaps' and calls the 'setWeather' method, passing in the next weather in the Weather list.

Inside the constructor of the abstract 'WeatherMap' class, we call the 'setWeather' method and pass Enum Status 'SUNNY' as an argument, this serves as our default weather.

The third problem is effectively solved as well, all we have to do is check if the ground the enemy is standing on has a particular weather Status, and if so, call the corresponding weather modification method that we defined above.

Advantages of our current implementation vs the original implementation
Better extensibility
In our original implementation, we had to loop through every map that contained enemies that are affected by the weather. This means, if we decided to add enemies that were affected by weather in other maps in the future, we would have to modify the existing code. Creating a new class for each map that has entities that are affected by the weather, which would result in class proliferation in the future.

Better performance

Our new implementation has better performance as we do not need to loop through all the grounds for all the maps that have entities affected by weather changes. In the future, when we add more maps that are affected by the weather, the performance may suffer greatly since we would have to loop through each ground of each map to set its weather.

Disadvantages

Potential violation of ISP

Our current implementation assumes that every entity that is affected by sunny weather, will also be affected by rainy weather and vice versa. But perhaps this isn't the case for future enemies, if we wanted to add a new entity that is affected by rainy weather but not sunny weather, the entity would still have to implement the 'sunnyWeatherModifications' method.