Design Rationale REQ5

Implementation & Why:

In this requirement, we first created the ‘Abxervyer’ class which is the boss. The class extends ‘EnemyActor’. ‘EnemyActor’ is an abstract class that controls most of the specifications of the entity like name, displayChar, hitPoints and more.The "Abxervyer" class adherence to the Single Responsibility Principle is maintained as the "Abxervyer" class focuses solely on defining the boss character's attributes and behaviours. The class encapsulates properties like name, displayChar, hitPoints, and more, which aligns with the single responsibility principle by keeping these aspects of the character centralised.

We also override the method ‘allowableActions’ to have implement the ‘FollowBehaviour’.In ‘allowableActions’ we check if the ‘otherActor’ has the capability type of ‘EntityTypes.PLAYABLE’, which indicates that it is a player that is within the ‘Abxervyer’ exits, and also checks to make sure that ‘Abxervyer’ don’t already have an existing ‘FollowBehaviour’. If it meets all conditions, ‘Abxervyer’ will start following the ‘Player’.

For the boss to walk around freely in void, in the ‘Void’ class under the ‘tick’ method, we had ‘if’ statements to check if the actor that is currently on the void is the boss or not by checking the actor's capabilities. The ‘Void’ class will only kill if the actor standing on it does not have the capability of ‘EntityTypes.BOSS’. This follows the open closed principle, as instead of modifying the existing behaviour directly, we use the ‘tick’ method. The ‘Void’ class remains open for extension without altering its existing code.

We also created an Enum class called ‘Weather’ to store all different types of weather, in this case only SUNNY and RAINY. This allows for easier implementation in the future if more weathers are to be added. The “Weather” enum class adheres to the open closed principle as it provides a clear and extensible representation of different weather states.

New weather conditions can be added easily in the future without modifying existing code. We also created a ‘WeatherControl’ class that updates the weather every three turns. In the constructor, the initial weather is set to SUNNY, and we have a variable called ‘turnsSinceLastChange’ to act as a tracker to track how many turns have passed. We have a final variable called ‘TURNS_PER_WEATHER_CHANGE’ set as 3, so that in the method ‘updateWeather’, ‘turnsSinceLastChange’ will increment by 1 every round and if it is bigger or equal to ‘TURNS_PER_WEATHER_CHANGE’, it will call the method ‘toggleWeather’ to change the weather and set ‘turnsSinceLastChange’ back to 0. The method ‘toggleWeather’ is used to change the weather every three turns. We made our fields static and getCurrentWeather() static because there will always only be one weather so there is no need to make new instantiation at all. It is also because getCurrentWeather() is being used outside of the class so it can be accessed without storing an instance in external classes.

The "WeatherControl" class follows the Single Responsibility Principle by managing weather changes exclusively. It encapsulates the logic related to weather control.
Adherence to the Open-Closed Principle is maintained as new weather control strategies or features can be added without altering the "WeatherControl" class.

The class also implements the logic for toggling weather conditions, which further separates concerns and enhances maintainability.

Inside the ‘Abxervyer’ class, we override ‘playTurn’ method and call the method ‘updateWeather’ so the weather will change as long as the boss is alive.

Inside the ‘ForestKeeper’ class, we created a final ‘DEFAULT_RATE’ and set it to 0.15 as the default spawning rate for forest keeper, and have a ‘rate’ that equals ‘DEFAULT_RATE’. We override the ‘playTurn’method and use ‘if’ statements to check for current weather conditions. If the weather is SUNNY, we will double the spawning ‘rate’. If the weather is RAINY, we will have the rate back to ‘DEFAULT_RATE’ and increase its health by 10HP.

Inside the ‘RedWolf’ class, we created a final ‘DEFAULT_RATE’ and set it to 0.30 as the default spawning rate for red wolf, and have a ‘rate’ that equals ‘DEFAULT_RATE’. We also have a final ‘DEFAULT_DAMAGE’ and set it to 15, and a ‘damage’ that equals ‘DEFAULT_DAMAGE’. We override the ‘playTurn’method and use ‘if’ statements to check for current weather conditions. If the weather is SUNNY, we will triple the ‘damage’ and have the ‘rate’ be ‘DEFAULT_RATE’. If the weather is RAINY, the ‘rate’ will be multiplied by 1.5 and ‘damage’ will be ‘DEFAULT_DAMAGE’.

Pros

- The "Abxervyer" class focuses solely on defining the boss character's attributes and behaviors, adhering to the SRP.
- The "WeatherControl" class exclusively manages weather changes, following the SRP.
- The "Void" class handles void behavior without modification through the "tick" method, adhering to the OCP.
- The "WeatherControl" class can easily accommodate new weather control strategies or features without altering its existing code, following the OCP.
- The "Weather" enum provides a clear and extensible representation of different weather states, enhancing code readability and maintainability.
- Properties and behaviors of entities like "Abxervyer" are encapsulated within their respective classes, promoting code organization and reducing complexity.

Cons:

- The code, especially within "Abxervyer," "ForestKeeper," and "RedWolf" classes, may become complex as more weather conditions or entity-specific behaviors are added, potentially making it harder to maintain.

Extendability:

Adding New Bosses:

- The codebase is extendable for adding new boss characters by creating classes that extend "EnemyActor" and defining their unique attributes and behaviors.

  Adding New Weather Conditions:
- Extending the "Weather" enum with new weather conditions is straightforward, allowing easy implementation of additional weather types.

  Modifying Entity Behavior:
- Existing entity behaviors can be modified or extended by updating the respective class implementations (e.g., "Abxervyer," "ForestKeeper," and "RedWolf") based on weather conditions or other criteria.

  Weather Control Strategies:
- The "WeatherControl" class can easily accommodate new weather control strategies or variations by extending its functionality without affecting existing code.

  Adding New Entity Types:
- The codebase can be extended with new types of entities (e.g., new enemy types or playable characters) by creating classes that adhere to the established design patterns and principles.