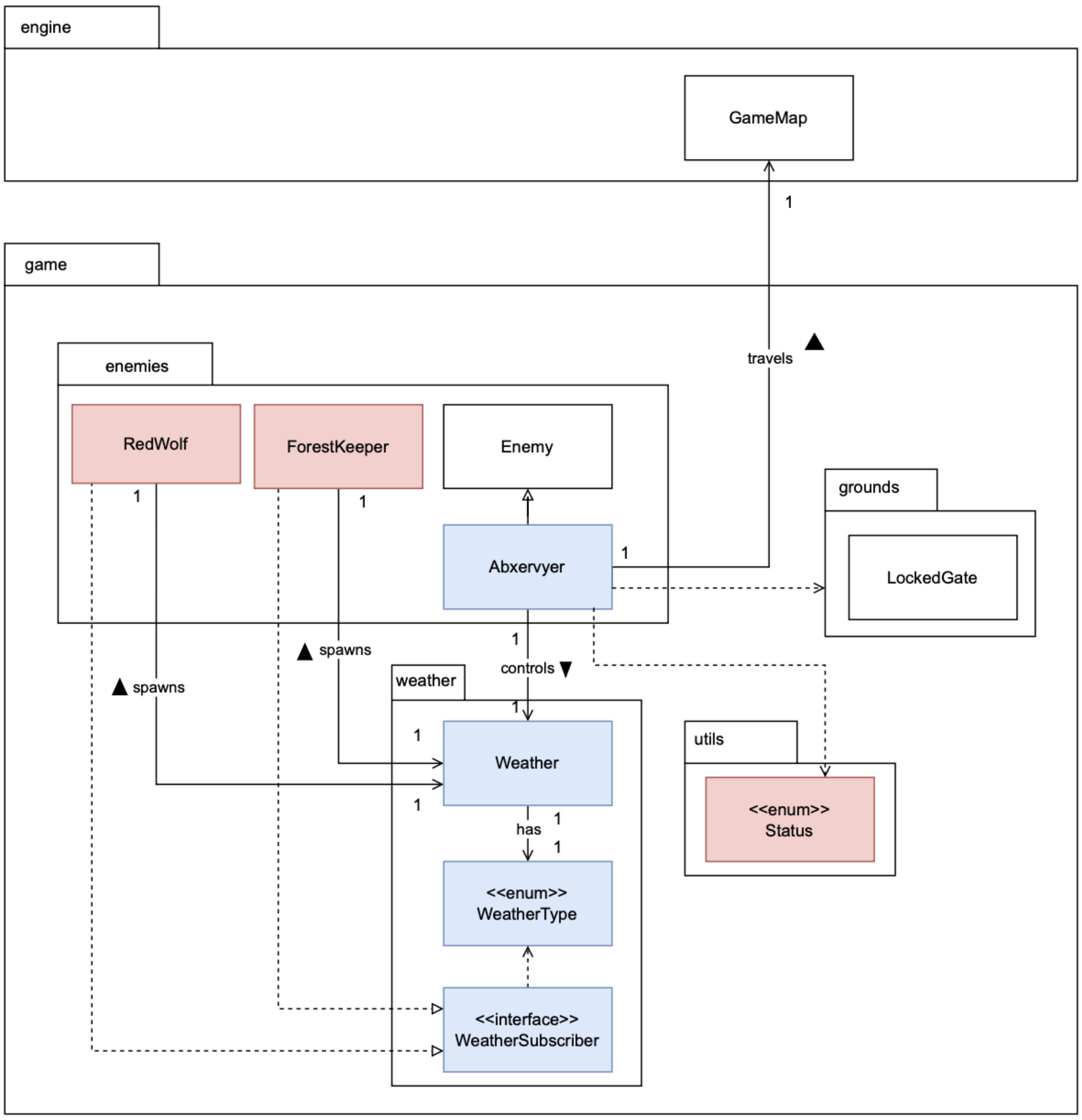
Design Rationale (REQ 5)



Changes Made from Assignment 1

This requirement largely diverges from the requirement from the previous assignment. The Abxervyer class, Weather class, and WeatherSubscriber interface are all new additions. Thus, there aren't significant modifications to the base code from Assignment 1.

Implementation

Classes Added

1. Class Abxervyer extends Enemy
2. Class Weather
3. Interface WeatherSubscriber

Codes Modified

1. Each type of enemy that is affected by the weather:
   - implements the interface WeatherSubscriber.
   - Overrides the necessary method: update.

Design Principles

Class Abxervyer

This class was created to represent the Abxervyer, The Forest Watcher.

Design Principles Followed

1. **Single Responsibility Principle (SOLID)**
   - The class Abxervyer represents a specific type of enemy in the game. It not only encapsulates behavior related to an Abxervyer such as its intrinsic weapon, the items they drop upon death, and the actions they can perform, but also its unique capabilities such as controlling weather and spawning an exit upon death.
2. **Open-Closed Principle (SOLID)**
   - This class is open for extension (can be subclassed to create variations) but closed for modification. Its core behaviors can be extended by overriding methods, but the core structure remains unchanged.
3. **Liskov Substitution Principle (SOLID)**
   - Since this class extends the Enemy class, they are expected to be substitutable for Enemy. Its overridden methods such as unconscious and playTurn are consistent with the behavior expected from the Enemy class.
4. **High Cohesion**
   - High cohesion is maintained in this class. Methods and properties in this class focus on a single role: defining the behaviour and attributes of the enemy type.

Class Weather

This class was created to represent weather conditions in the game.

Design Principles Followed

1. **Single Responsibility Principle (SOLID)**
   - The class Weather represents a specific feature of weather in the game. It holds the current weather in the world as its capability and encapsulates relevant operations on the weather such as toggling and changing the weather.
2. **High Cohesion**
   - High cohesion is maintained in this class. Methods and properties in this class focus on a single role: defining the current weather and changing them.

Interface WeatherSubscriber

This interface was created to represent objects that are affected by the weather.

1. **Interface Segregation Principle (SOLID)**
    - The WeatherSubscriber interface ensures that the classes that implement it provide specific functionality so that relevant attributes are modified according to the current weather.

Pros of the Design

1. **Abstraction-Extensibility**
    - The WeatherSubscriber interface was created to represent objects that are affected by the weather. The addition of this new layer of abstraction not only makes the system more comprehensible, but also improves extensibility of the system as any objects that its attributes are weather-sensitive can extend this interface to be implemented with necessary methods to achieve the functionality.

Cons of the Design

1. **Violation of Open-closed Principle (SOLID)**
    - Introduction of new weathers in the future requires many changes to be made within the Weather class instead of being subclassed.
    - This reduces the extensibility of the system as all methods within the Weather class will need to be modified to adapt to these changes.
2. **Future Complexity**
    - Any objects that are affected by weather stores the instance of the Weather class to keep track of the current weather condition. This adds complexity of the system as any classes that are affected by the weather will require to have an association with the Weather class.
3. **Connascence of Value (CoV)**
    - The creation of a new LockedGate upon the death of Abxervyer assumes that the destination of the TravelAction will always be Ancient Woods by hard-coding the second argument of the TravelAction constructor. This creates a connascence of value as any future changes in the destination of the spawned gate creates a need to manually change the string argument of the TravelAction constructor.