I created an abstract class called ‘EnemySpawner’ that extends the Ground abstract class, this abstract class handles the spawning logic. All spawners like Graveyard, Hut and Bush extend abstract class EnemySpawner. Since all these spawners share some common attributes and methods, we abstract these identities to avoid repetitions (DRY).

Since enemies share common behaviours, I made ForestKeeper and RedWolf class extend the abstract Enemy class. Since they share some common attributes and methods, abstracting these identities avoids repetition, adhering to the (DRY). Abstract Enemy class also has a dependency on the Exit class to assess the presence of other actors in its vicinity, this determines whether it attacks or just wanders. It depends on the Status enum class to ensure it only attacks Players, not other Enemies.

FollowBehaviour extends the Behaviour interface, this ensures that ForestKeeper and RedWolf follow the player around.

Advantages:
This implementation of having an abstract ‘EnemySpawner’ is better than the alternative of having an interface implementation. Since the spawning logic across all enemies remains the same, and the only differences are the enemy type that is spawn and the spawn rate, we can just write the spawning logic once in the abstract ‘EnemySpawner’ class and have all spawner subclasses extend it. An interface implementation would require a lot of repeated code across all the subclasses.

Disadvantages:
It may be less flexible than an interface implementation, as the spawning logic is already defined in the parent class. This means that any classes that have special spawning logic will be slightly limited.