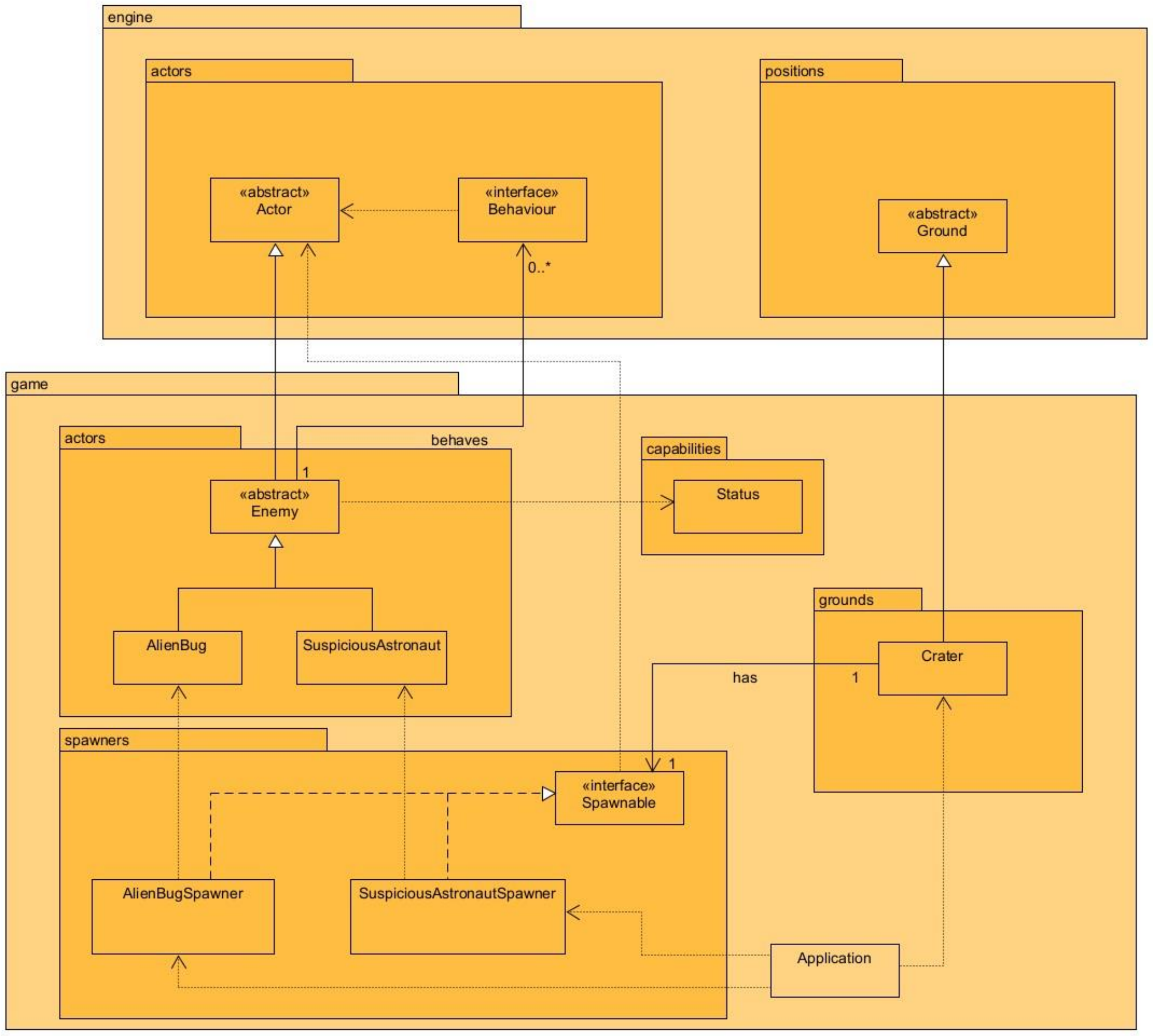
Assignment 2 Design Rationale

Requirement 1: The moon’s (hostile) fauna II: The moon strikes back

UML Diagram:



The design goal for this assignment is to introduce two new Actors to the game, Alien Bug and Suspicious Astronaut while adhering closely to relevant design principles and best practices. A crater can spawn an Alien Bug with a 10% chance, and a Suspicious Astronaut with a 5% chance at each turn of the game, with each crater instance only spawning one type of creature.

In implementing the two new classes AlienBug and SuspiciousAstronaut, it quickly became obvious that the two new classes along with HuntsmanSpider shared a lot of identical code. The implementation of the playTurn and allowableActions methods of the three classes were exactly the same, and the three classes all had a TreeMap behaviours attribute. This repetition of the same code was a violation of the DRY principle. Therefore, the decision was made to create an Enemy abstract class that the three classes would extend.

An alternative design considered was to continue our design from Assignment 1 and not introduce a new Enemy abstract class. HuntsmanSpider, AlienBug and SuspiciousAstronaut would then continue to extend the Actor class from engine. This was considered as it was unclear whether future enemy NPCs would necessarily continue to share the common code that the three classes share. However, this would lead to a violation of DRY.

In our design, the Enemy abstract class allows us to closely adhere to the DRY principle. Common enemy behaviour has been placed in the Enemy abstract class, preventing code repetition in HuntsmanSpider, AlienBug and SuspiciousAstronaut and any future enemy NPCs with similar behaviour that may be added when the game is extended. Our design also adheres to the Liskov Substitution Principle. This is because all child classes of Enemy can be used wherever Enemy is expected. With our design it is now easier to add new enemy NPCs to the game, as common behaviour does not need to be repeated while unique behaviour can be implemented in the new enemy's class. These new enemy NPCs can be added without the need to modify the Enemy abstract class. This shows that our design adheres to the Open-closed principle, as the Enemy class is open for extension, but closed for modification.

A drawback to this design is that not all future enemy NPCs will necessarily also share the behaviour in the enemy abstract class. For example, an enemy that the player cannot attack could be added.

To have the AlienBug and SuspiciousAstronaut spawn from a crater, we decided to continue using our design from Assignment 1, which is the creation of spawner classes such as AlienBugSpawner and SuspiciousAstronautSpawner that implement the Spawnable interface, which the Crater class has as an attribute. An advantage of this design is that it is easily to extend the actors that Crater can spawn. In order for Crater to spawn a new Actor, you would simply need to create a Spawner class which implements Spawnable for that actor. Crater would then be able to spawn this actor, without the need to modify the Crater class. This adheres to the Open-close principle.

One alternative approach was to stray from our design choice in Assignment 1, and get rid of the spawner classes. The crater class would instead take in an instance of Enemy instead of Spawnable in its constructor. This would remove the need for any spawner classes or the Spawnable interface. However, with this design, classes that extend Enemy would have to handle the logic that Crater was originally responsible for.

This alternative design is not ideal because it violates various design and SOLID principles. If the crater class took in instances of Enemy instead of their associated Spawner classes, the classes that extend Enemy would not only have to be responsible for their own behaviour, but also their spawning behaviour. This is a violation of the Single Responsibility Principle.

Therefore, another advantage of our chosen design is that the spawner classes help us adhere to the Single Responsibility Principle. The Crater class is responsible for spawning creatures, the spawner classes are responsible for creating instances of their associated actor and determining whether the said actor can spawn or not, while the actor classes are responsible for their own behaviour.

Overall, our chosen design provides a robust framework for adding the new AlienBug and SuspiciousAstronaut actors. By carefully considering design principles along with the requirements of the assignment, we have developed a solution that is efficient, paving the way for future extensions.