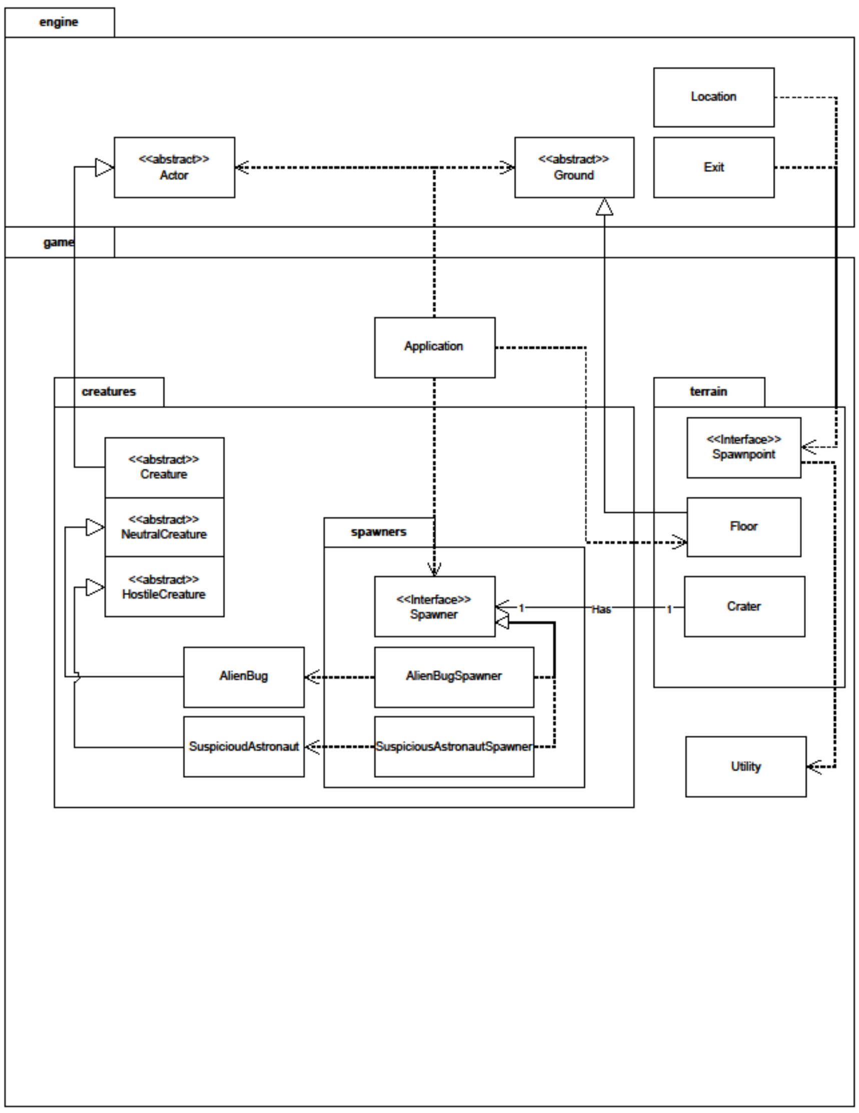
Assignment 2 Design Rationale

Requirement 1: Adding AlienBug and SuspiciousAstronaut Creature types

UML Diagram:



Design Goal:

The design goal for this requirement is to implement the addition of 2 new types of Creatures, AlienBug and SuspiciousAstronaut and make them spawn from Craters while adhering closely to relevant design principles and best practices.

Design Decision:

In implementing the addition of the 2 new Creatures, the decision was made to implement the classes and their respective Spawner classes to be used as an attribute in Crater which implements the Spawnpoint interface that indicates Crater is capable of spawning new entities. The reasoning behind this is to keep enabling Crater to spawn new instances of a specific creature or any other spawn-able object with no consideration to the type of entity it spawns. This flexibility helps with scalability as more spawn-able entities added in the future can be spawned from Crater without modifying the code for Crater. Maintaining the class will also be easier due to its single responsibility in spawning an entity.

Alternative Design:
One alternative approach could involve adding all the types of entities into Crater with their spawn methods:

```
public class Crater{
        public void spawnAlienbug{
//code to spawn AlienBug
}
        Public void spawnSuspiciousAstronaut{
//cod to spawn SuspiciousAstronaut
}

}
```

Analysis of Alternative Design:
The alternative design is not ideal because it violates various design and SOLID principles:

1. **Single Responsibility Principle:**
    - Approach above results in a God Class which violates the SRP. This makes the maintainability worse as the class now has many characteristics to consider.

Final Design:
In our design, we closely adhere to the DRY and SOLID design principles.

1. **Single Responsibility Principle:**
    - **Application:** Each Creature Spawner generates a new instance of a set specific type of creature.
    - **Why:** It provides the ability to generate completely new instances of Creatures without side effects.
    - **Pros/Cons:** The pros are that the code is maintainable as the class only has a single purpose. The cons are that this increases the level of abstraction in the system which leads to the increase in the complexity of the system.
2. **Liskov's Substitution Principle:**
    - **Application:** Made all Creature spawners implement the Spawner class.
    - **Why:** This is to make spawning of entities simpler as all Spawners can be treated similarly even if they have different behaviours (polymorphism).

- **Pros/Cons:** The pros are that any class implementing the Spawnpoint interface can utilize any type of Spawner. Both sides are also isolated from change in the code of the other side.

Conclusion:
Overall, our chosen design provides a robust framework to add the new Creature types. By carefully considering the DRY and SOLID design principles, we have developed a solution that is scalable and maintainable, paving the way for further extensions in the future.