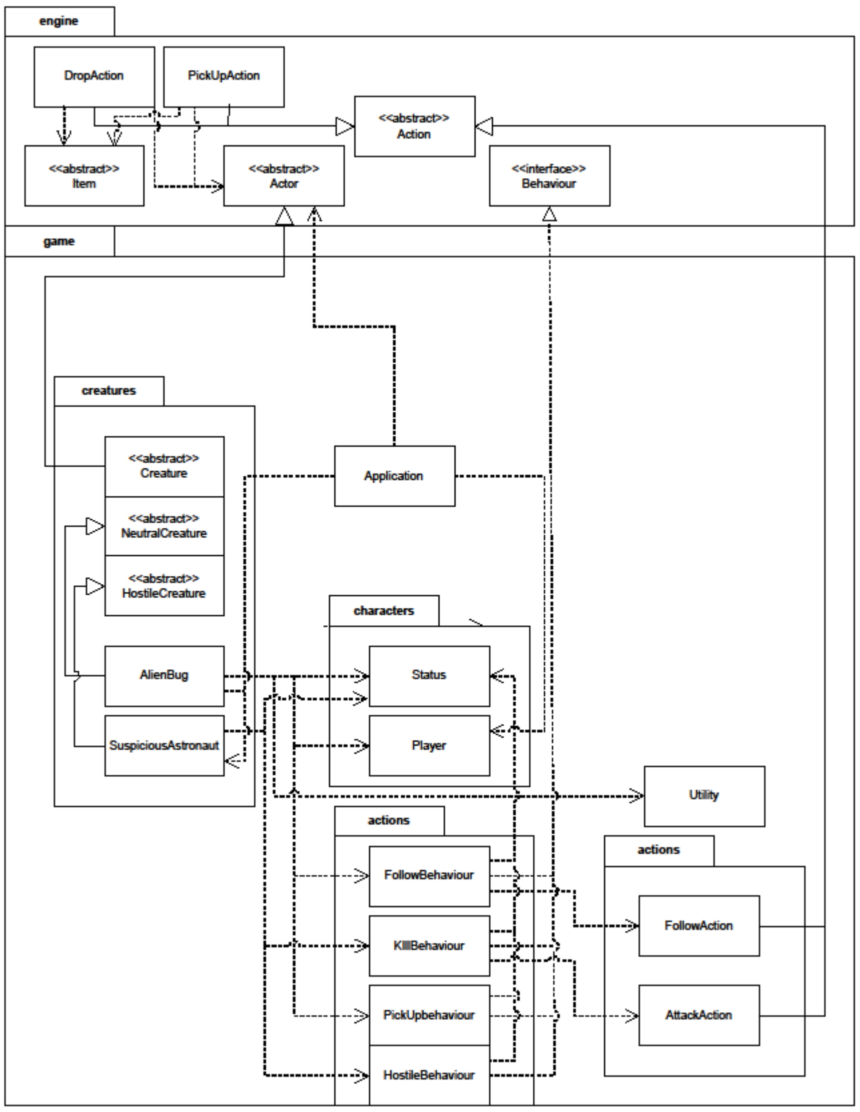
Assignment 2 Design Rationale

Requirement 2: New creature behaviours

UML Diagram:



Design Goal:

The design goal for this requirement is to implement new behaviours in AlienBug and SuspiciousAstronaut while adhering closely to relevant design principles and best practices.

Design Decision:

In implementing the new behaviours, the decision was made to create new behaviour classes extending off the Behaviour abstract class in the engine. This design choice adds flexibility in configuration of behaviours of the creatures.

FollowBehaviour’s getAction method first scans around the actor to check for a target fitting the status criteria, then it returns a MoveActorAction if possible, to get closer towards the target.

KillBehaviour’s getAction method returns an AttackAction which instantly kills the target.

PickUpBehaviours getAction method returns a PickUpAction of a random item on the ground the actor is standing on.

Alternative Design:
One alternative approach could involve adding the methods that represent the behaviours in the Creature’s playTurn method instead. However, this decreases the flexibility when it comes to configuring the behaviour and would negatively affect code scalability and maintainability. For example:

```
public class AlienBug extends NeutralCreature {
//code for wandering and picking up items
}
```

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

1. **Open Closed Principle:**
    - To modify how a specific behaviour operates or to add new behaviours, the Creature’s code would need to be modified, violating OCP.

Final Design:
In our design, we closely adhere to DRY and SOLID principles:
1. **Open Closed Principle:**
    - **Application:** All behaviours implement the Behaviour interface enabling easy extension and modification of behaviours.
    - **Why:** This is done to make the code easily extendable and maintainable
    - **Pros/Cons:** Pros are the increase in extensibility and maintainability of the code. Cons are that the complexity of the whole project will increase due to the increasing number of classes implementing Behaviour.
2. **Dependency Inversion Principle:**
    - **Application:** Having each Creature depend on the Behaviour interface instead of each individual behaviour.
    - **Why:** This is so that both the Creature and Behaviour classes are insulated from change in the code of one another.

- **Pros/Cons:** The pros are that classes are insulated from change. The cons are that complexity is increased due to the increasing number of classes implementing the interface.

Conclusion:
Overall, our chosen design provides a robust framework for implementing new behaviours. By carefully considering and adhering to the SOLID and DRY design principles, we have developed a solution that is efficient, scalable, and maintainable paving the way for future extensions.