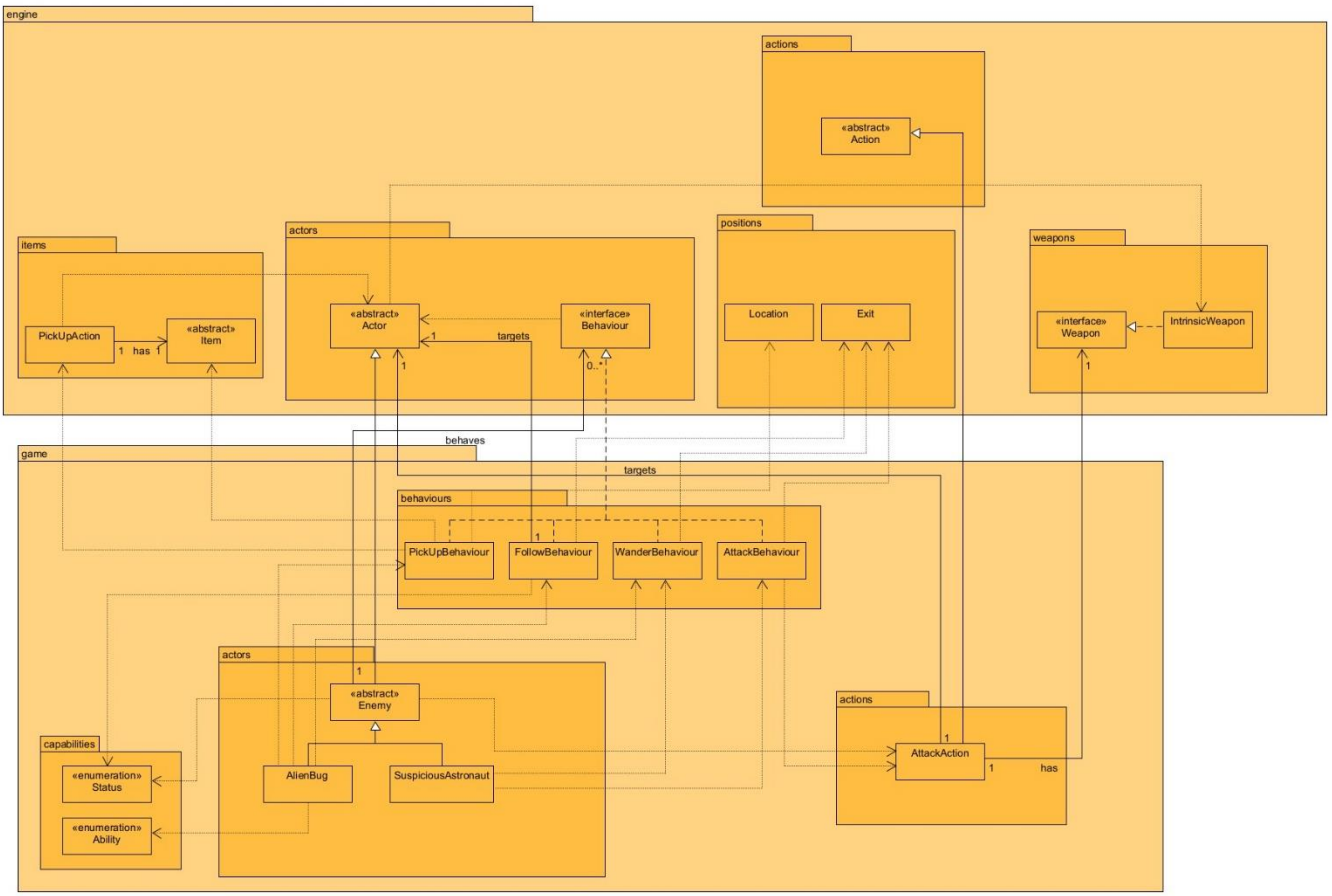
Assignment 2 Design Rationale

Requirement 2: The imposter among us

UML Diagram:



The design goal for this assignment is to introduce the unique logic and behaviours of the new actors AlienBug and SuspiciousAstronaut that were first added in REQ1 while adhering closely to relevant design principles and best practices.

The Alien Bug has 2 hit points, cannot attack, and can wander around the map and pick up scraps. If the Intern is within the surroundings of the Alien Bug, the Alien Bug begins to follow the player, stopping to pick up any scraps along the way, before continuing to follow the player again. The Alien Bug follows the player until it is defeated. Upon defeat, the Alien Bug drops the scraps that it picked up. The Alien Bug can enter the Intern's spaceship.

The Suspicious Astronaut wanders around, and if the Intern is within its surroundings, it attacks the Intern, instantly killing them with 100% precision. The Suspicious Astronaut has 99 hit points and cannot attack any other creatures hostile to the Intern. The Suspicious Astronaut cannot enter the Intern's spaceship.

For the Alien Bug to be able to enter the spaceship, while still preventing other actors such as HuntsmanSpider and SuspiciousAstronaut from doing so, the Floor class's canActorEnter method was overridden to only return true if the Actor has the ENTER_SPACESHIP capability. This capability is added to Player and AlienBug, in their classes' constructors, allowing them to enter the spaceship.

In order for the Alien Bug to follow the player, a new FollowBehaviour class was added. The FollowBehaviour class in our design largely follows the same implementation as that given in the mars package. The FollowBehaviour class allows an actor such as the AlienBug to start following a target actor once the target is within their surroundings. The main challenge of implementing this part was that the constructor of FollowBehaviour from the mars package required an Actor as an argument. In the mars package, the bug was given the FollowBehaviour in the Application class. This was not feasible for our implementation as each AlienBug spawned of a crater needed to have the FollowBehaviour, with the player as the target. We decided to modify the constructor of FollowBehaviour, to initialise target as null. If target is null, FollowBehaviour then checks each of the exits of the actor to look for a target. Once a target has been found, the original implementation can the be used and the actor with FollowBehaviour follows its target.

In order for the Alien Bug to pick up scraps, a new PickUpBehaviour class was added. PickUpBehaviour gets a list of items at the location of an actor, randomly picks an item and returns a PickUpAction to pick up that item. If there are no items, it returns null.

The AlienBug was made to drop all its items upon defeat by overriding its unconscious method.

Both FollowBehaviour and PickUpBehaviour implement the Behaviour interface. By implementing the Behaviour interface, FollowBehaviour and PickUpBehaviour can be used wherever Behaviour is expected, without breaking the program. This adheres to the Liskov Substitution Principle. The Behaviour interface contains only a single method, getAction, which is used by all behaviours. No behaviour is forced to have a method that it does not use, this adheres to the Interface Segregation Principle.

An alternative design would be to have a Behaviour "God Class", that would contain the logic of all behaviours. This would eliminate the need to have individual behaviour classes. However, a class such as this would absolutely violate the Single Responsibility Principle, as the class would be responsible for all kinds of behaviours. This would make the code hard to maintain. A Behaviours God Class would also violate the Open-Close Principle, as the Behaviours God Class would need to be modified whenever a new type of behaviour were to be added. A God Class would also increase the level of connascence in the design, making it harder to extend or maintain.

The SuspiciousAstronaut and AlienBug are able to wander around using the WanderBehaviour. Similar to the HuntsmanSpider, the SuspiciousAstronaut has AttackBehaviour, which allows the SuspiciousAstronaut to attack actors with the status HOSTILE_TO_ENEMY. SuspiciousAstronaut deals enough damage to instantly kill the player, this is done by overriding its getIntrinsicWeapon method to return an intrinsic weapon with damage equal to Integer.MAX_VALUE. This allows the SuspiciousAstronaut to instantly kill the player, no matter how much the player increases their health.

Overall, our chosen design provides a robust framework for adding the logic and behaviours of 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.