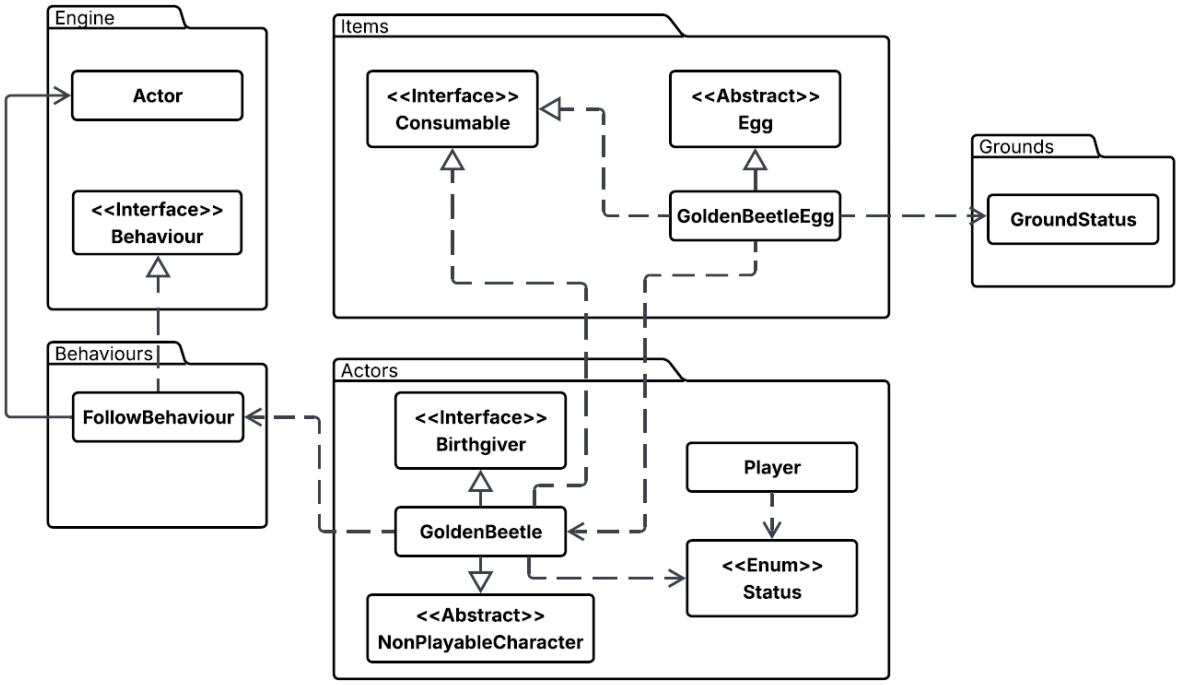
Req 2 UML Diagram



Req 2 Design Rationale

Some considerations that had to be made were:

A. GoldenBeetle and GoldenBeetleEgg Implementation

B. Consumable Implementation

C. Follow Behaviour Implementation

A. GoldenBeetle and GoldenBeetleEgg Implementation

Design 1: Inheriting from NPC and Egg Class

Pros	Cons
• Reduces Code repetitiveness as it uses pre-defined code already in the system	• Introduces dependency between the abstract classes and the children classes, leading to more complex design

• More easily maintainable as each class has its own responsibility	

Design 2: Creating All New Code for Concrete Class

Pros	Cons
• Easy to implement • Little dependencies	• Lots of repeated code • Not maintainable as classes have a lot of code • Class may have multiple responsibilities

B. Consumable Implementation

Design 1: Inherit from Consumable Interface

Pro	Cons
• Uses defined code, reduces repeated code • Abstracts consumption logic and dependencies away from the concrete class, allowing single responsibility	• Introduces dependencies which may be unwanted

Design 2: Creating Individual consumption interaction methods with actors

Pros	Cons
• Easy to implement • Few dependencies	• Increases repeated code • Hard to maintain as each interaction with other actors needs to be defined and maintained

	• Not modular as scaling up would be resource intensive

C. Follow Behaviour Implementation

Design 1: Implement Behaviour Interface

Pros	Cons
• Reduce repeated code • Allows each creature to depend on this, rather than defining their own follow behaviour, increasing scalability • Easier to maintain	• Increased complexity

Design 2: Create follow behaviour logic for individual concrete classes

Pros	Cons
• Simple to implement • No dependencies	• Difficult to maintain and scale • Repeated code between actors

The REQ 2 UML Diagram shows the final design for requirement 2, which utilises design 1 from parts A, B, and C to implement GoldenBeetle and GoldenBeetleEgg classes, as well as the FollowBehaviour implementation.

Having a NPC and Egg abstract class that both Golden Beetle and GoldenBeetle Egg inherit from allows common behaviour to be implemented, as well as reducing the need to repeat code that was already within the system (ReD). Furthermore, by abstracting away the NPC logic, the concrete classes can focus on their own individual logic where needed (SRP).

By implementing the consumable interface for both of these concrete classes, once again common behaviour, as well as predefined relationships and dependencies can be abstracted away into being a consumable.

Finally, to create the following methods, the behaviour interface was implemented. This allows a direct connection between the game engines defined methods to be abstracted away, allowing the FollowBehaviour to focus on its own logic.