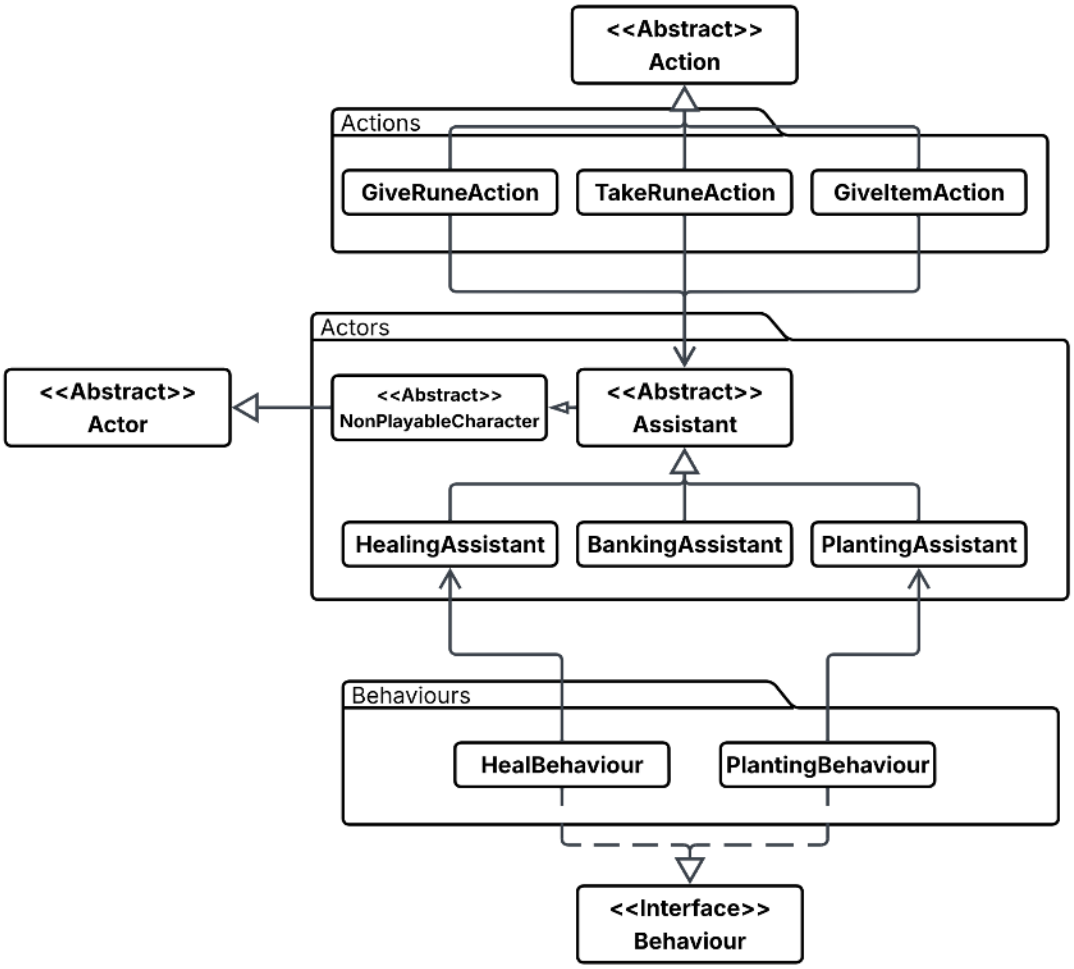
Design Diagrams

REQ 4 UML Class Diagrams:



Design Rationale

REQ 4

Some considerations that had to be made were:

A. Assistant Implementation
B. Behaviour Implementation

A. Assistant Implementation

Design 1: Creating Assistant abstract Class

Pros	Cons
- Allows a template assistant class to form the basics of all assistants, allowing for DRY - Future assistants can inherit from this abstract class and extend via their own function and methods, enhancing polymorphism - As assistant may be used in future, all classes will be able to interact with such methods, following LSP	- Introduces dependencies between abstract class and future assistant clases, may result in unwanted contractual laws in the future - More complicated to implement

Design 2: No Inheritance/Interfacing Concrete Class

Pros	Cons
- Simple implementation, no dependencies	- Repeated NPC and assistant class code - Each class has its own relationships, coding holisitic functionality to be difficult -

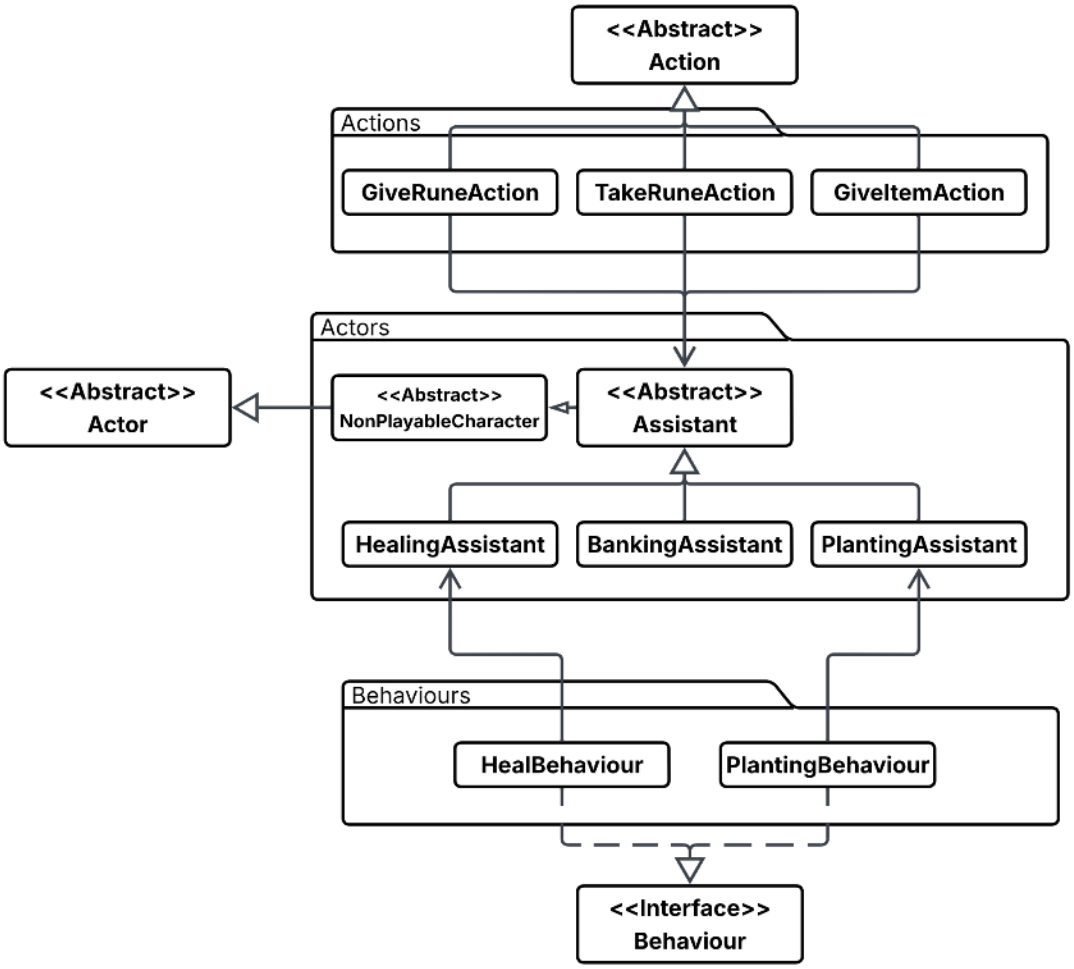
B. Behaviour Implementation

Design 1: Inherit from Behaviour interface

Pros	Cons
- Reduces repeated code between classes - Can be ordered by class ranking in instantiation	- Coupling of behaviours -

Design 2: Concrete class with no inheritance/interfacing

Pros	Cons
- Simple and easy to implement - Specific to the class	- Repeated code - Not maintainable as each class with this behaviour would need their own implementation



The diagram above shows the final design of requirement 4. This utilises design 1 from parts A and B.

By having an assistant abstract class that holds the logic required for all future assistant classes, boilerplate code is present for ease of scalability, as well as reducing repeated code through having a centralised class for all assistant related logic.

Similarly, by having both behaviours interface with the already present in engine behaviour interace, these can be properly implemented and integrated with already present behaviours that NPCs or assistants may have. Furthermore, the autonomous functionality of the assistant class is better handled through behaviours as it allows future classes to also use these behaviours, without having their own implementations.