Design Rationale:

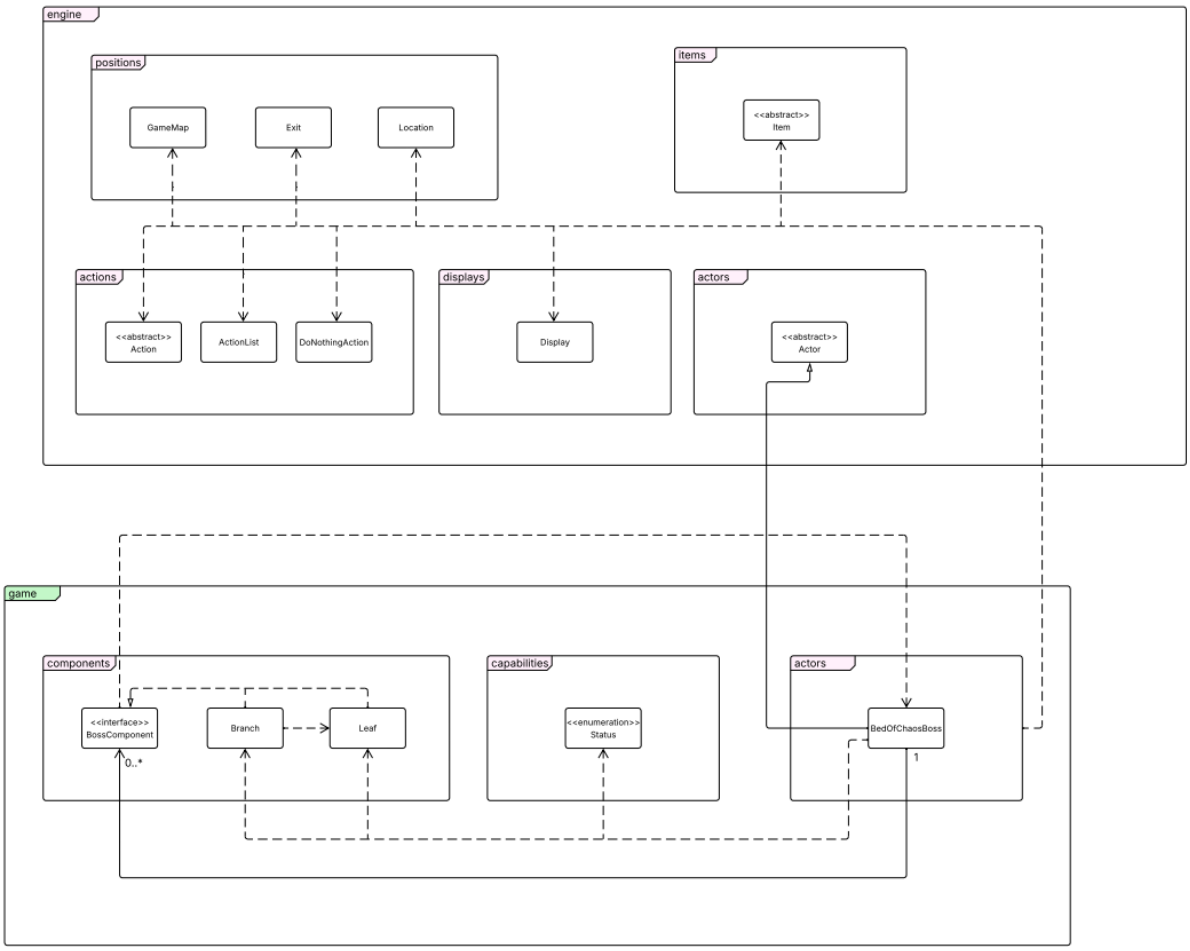
REQ 2:

DESIGN 1: BedOfChaos handles the implementation of its own branches and leaves via their own lists, where each growth calculation method handles conditionals

Pros	Cons
Allows for simplicity as no abstraction layers are required which reduces the initial code complexity	Poor scalability as adding new components would require the modification of multiple methods
Allows for the direct modification of components related to the BedOfChaos boss in one class	Tight coupling as the boss depends on concrete component classes which reduces flexibility

DESIGN 2: Implement the BossComponent interface that allows for the interaction with both the Branch and Leaf objects, using polymorphism

Pros	Cons
Allows for scalability as new components can be added without the need of modifying the boss class	More complex to implement due to the interface abstraction
Each component encapsulates its own logic and therefore is easier to modify	More classes mean the code may be harder to maintain
Decoupled as the boss class only relies on an interface	



The diagram above shows the implementation of requirement 2 using design 2, in which abstraction and polymorphism is used to create a flexible and maintainable system. Through the use of the interface BossComponent, the branch and leaf classes are able to define their own unique behaviours for growth, damage contribution, and healing without the being tightly coupled with the boss itself, closely aligning with the Single Responsibility Principle. The design was chosen such that it adheres to the Open/Closed Principle where new components can be added without modifying the existing BedOfChaosBoss class. This was achieved through the use of polymorphism where new components automatically integrate the damage calculation and growth logic. This design also allows for the Liskov Substitution Principle as the implementations of BossComponent can be used where a component may be used. In conclusion, design 2 was chosen over design 1 due to the clean and scalable structure that reduces the overall dependencies and allows the system to be easily extended.