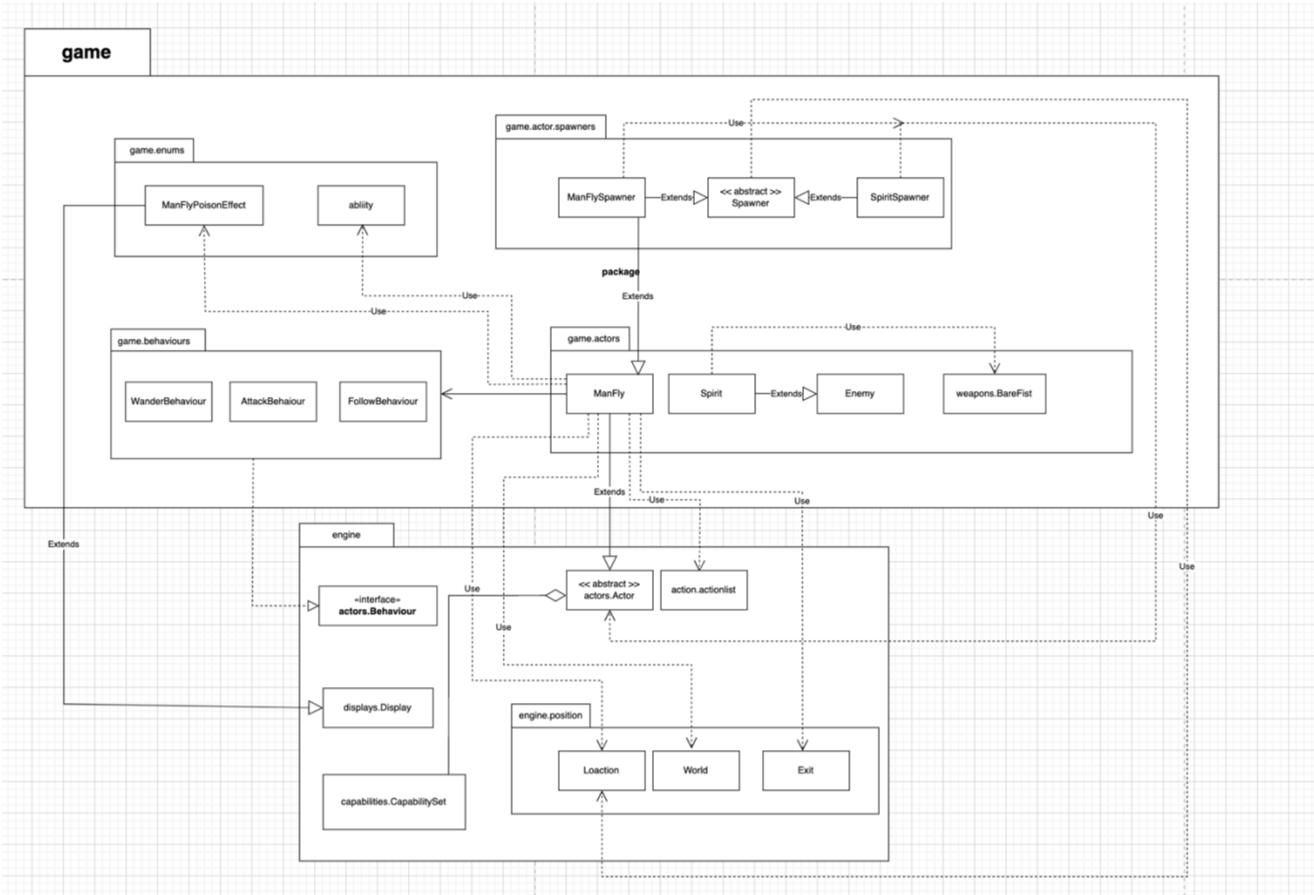
FIT2099 Assignment 2 Design Rationale For REQ3 ----ZhaoZhao34433961

UML diagram :



Design rationale:

Implementation 1:

Man_Fly Enemy Detection and Attack Logic Implementation I implemented two main methods for detecting nearby enemies and determining follow-through: direct range-based checking and behaviour-driven following.

Classes Modified / Created	Roles and Responsibilities	Rationale
Manfly	Represents ManFly enemy actors, with abilities such as wandering, following and using poison to attack nearby actors Relationships: 1, Use FollowBehaviour to track hostile actors when they are nearby. 2, Use WanderBehaviour to define wandering behaviour. 3, Inherits from Actor to handle actor-related functions, including movement and behaviour execution	**Design 1**: Traverse all nearby locations and manually check actors **Pros**: this design directly uses the calculated Manhattan distance to cycle through all neighbouring positions to search for enemies within a specific radius. It clearly represents the spatial relationship between the player and the enemy **Cons:**Duplicate code: the logic of manual checks can lead to duplicate code, especially if more specific conditions need to be checked multiple times in the follow behaviour **Design 2:** Using the game engine's existing actor detection methods **Pros**: Maintainability: Since there are fewer custom checks, there is less risk of introducing errors when changing participant detection logic. **Cons**: As the engine handles the detection, it may be more difficult

ManFlyPoisonEffect	Represents a poison effect that ManFly applies to its target. It deals damage over time (10 points of damage per turn, lasts for up to 2 rounds, and can stack) and is automatically removed at the end of its duration.	to tailor the behaviour to specific gameplay requirements, such as detecting only certain enemy types or applying conditions to detect how it occurs **Final Choice:**The final design chose Design 2 , which relies on the engine's detection method to avoid the complexity of manually checking each neighbouring position. This reduced overall complexity and improved code maintainability. The system is more optimised and code duplication is minimised. This also follows the **Open-Closed Principle（OCP）**, which allows the detection logic to be extended in the future without modifying the existing code. The design follows the **Single Responsibility Principle (SRP)** by separating concerns between ManFlyAttack, WanderBehaviour, FollowBehaviour and ManFlyPoisonEffect.

Implementation 2:

The Spirit class was implemented to represent the enemies in the game. Spirit comes with the default inherent weapon BareFist . The design should ensure that Spirit is reusable, integrates with the game's combat mechanics, and interacts well with the core engine framework.

Classes Modified / Created	Roles and Responsibilities	Rationale
Spirit	The Spirit class extends the Enemy class and represents a specific type of enemy in the game Relationships: 1,Inherits from the Enemy class and makes use of the behaviour and logic associated with enemy participants. 2,Interacts with the BareFist weapon class to define its attack behaviour.	**Design 1**: **Pros**: By inheriting from the Enemy class, the Spirit class reuses common enemy logic, such as life management, movement behaviour, attack modes, and so on. This follows the **DRY principle**, as the common logic is shared, thus reducing duplication. **Cons:** Lack of Customisation: The Spirit class does not offer any unique combat mechanics, such as special abilities that distinguish it from other enemies. As a result, gameplay interactions can become repetitive during combat. **Design 2:** The design allows Spirit to have different behaviours (e.g. WanderBehaviour or FollowBehaviour) to make fighting them more dynamic. **Pros**: Using the same intrinsic weapon (BareFist) ensures that weapon behaviour is centralised and maintainable. If the weapon logic needs to be updated, it can be done in one place following the **DRY principle.** **Cons**: Adding more complexity requires additional weapon mechanisms that may violate the **SRP** if not handled properly .
Spirit Spawner	The SpiritSpawner class extends the Spawner class and is responsible for generating Spirit Actors in the game. Relationship 1,Inherits from Spawner and uses generative logic to create instances of the Spirit class.	

		Final Choice: Design 1 was chosen because of its simplicity and adherence to **OCP**. the class can be easily extended to add new behaviours without violating key principles such as **SRP** or **DRY.**