Requirement 1 Design Rationale

Teleportation Circle and Livmeld

The teleportation circle implementation in the Elden Thing game, alongside the development of the Limveld world map, demonstrates strong adherence to the SOLID and DRY principles. Following the Single Responsibility Principle, the TeleportationCircle class focuses exclusively on teleportation behavior, while the construction of the Limveld map and its integration into the world is handled within Application.main(), keeping responsibilities clearly separated. The design supports the Open/Closed Principle by enabling new areas like Limveld to be added through configuration—by instantiating new TeleportationCircle objects and attaching MoveActorAction instances targeting Limveld locations—without modifying existing class code. It also complies with the Liskov Substitution Principle, as TeleportationCircle extends Item and behaves consistently within the game engine's expectations for interactable items. Although the Interface Segregation Principle isn't directly enforced, the current design avoids bloated interfaces and could be enhanced further through more granular abstractions. The Dependency Inversion Principle is upheld by injecting teleportation actions into the circles at runtime, decoupling the teleportation logic from the target area implementation. Adhering to the DRY Principle, both the teleportation mechanics and the Limveld map setup reuse generic structures—such as using the same TeleportationCircle class for both directions of travel and defining map transitions using shared patterns. Together, the integration of Limveld as a new game map and the modular teleportation system illustrate a well-structured, extensible design that promotes maintainability and scalability in line with object-oriented best practices.

Disadvantage

While the teleportation circle and Limveld map implementation is effective and modular, it presents some disadvantages that warrant consideration. One key issue is the reliance on manual configuration within Application.main(), which introduces tight coupling between the world setup and teleportation logic—making the system less dynamic and harder to scale if many maps or teleport points are added.

Behaviour Selection Strategy

The implementation of the behaviour selection mechanism in this system aligns well with both the SOLID and DRY principles. By introducing the BehaviourSelectionStrategy interface, the design adheres to the Open/Closed Principle, allowing new strategies (like PriorityBasedStrategy and RandomSelectionStrategy) to be added without modifying existing code. The Single Responsibility Principle is also upheld, as each class has a clear, focused purpose: the NonPlayerCharacter handles character logic, the strategy classes encapsulate behaviour selection logic, and Behaviour defines actions. The Liskov Substitution Principle is maintained since any strategy can be used interchangeably without breaking the NPC's behaviour selection logic. The Interface Segregation Principle is reflected in the simple and focused BehaviourSelectionStrategy interface. Furthermore, Dependency Inversion is achieved as NPCs depend on the abstraction (BehaviourSelectionStrategy) rather than concrete implementations. From a DRY perspective, the code avoids duplication by centralizing behaviour selection logic into strategy classes, preventing repetition across different creature types. This also improves maintainability, allowing adjustments to behaviour logic in one place rather than scattered checks in each creature class. Overall, the design is modular, extensible, and promotes clean separation of concerns.

Disadvantage

As all the Behaviour strategies rely on the BehaviourSorter class, any change in the functionality of BehaviourSorter may result in the requiring of every strategy being updated, which is Connascence of Type. This connascence is however allowed because BehaviourSorter is unlikely to change and is a suitable system that easily allows adding new Behaviour priorities to an Enum class that makes the code easier to extend, so it is seen as an acceptable tradeoff.