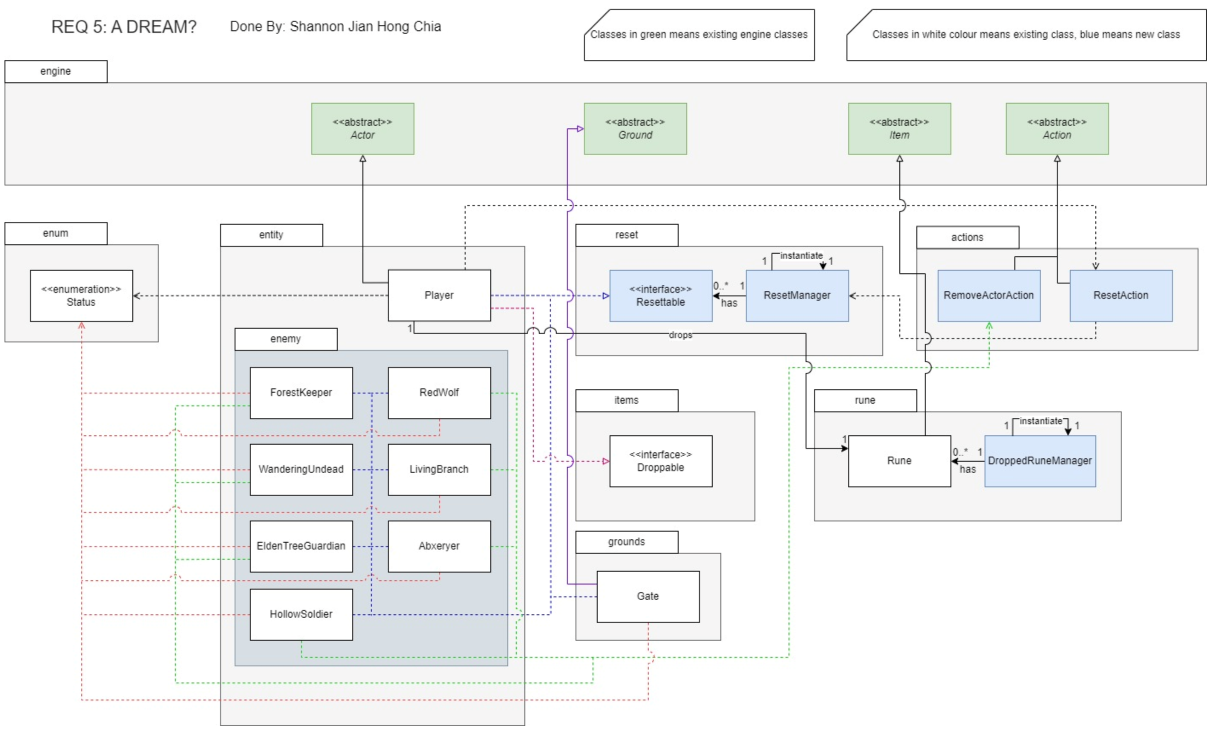
Design Rationale: Req 5 (Done by Shannon Jian Hong Chia)



Brief

The UML Diagram represents Requirement 5 for assignment 3, where 5 new classes were introduced, where 2 of it are classes that follows the Registry and Singleton pattern, 2 new action classes to perform specific action, and 1 new interface.

This requirement focuses on resetting the game to the initial state after the player dies, such as despawning entities, restoring bosses' health to its max HP, and dropping all rune possession on the place where the player died.

Reset Manager

The Reset Manager uses the Registry and Singleton pattern to handle all Reset functions for registered Resettable classes. This approach provides centralized control (OCP) among all Resettable classes.

The registry and singleton pattern are essential, allowing us to link all resettable entities to the reset manager, enabling easy global resets through one centralise interface. Despite having multiple setback, like close and tight relationship between Resettable entities with the Reset Manager, this approach has greater potential for future expansion, accommodating a growing number of resettable entities in the game.

Reset Action

The ResetManager class invokes the ResetAction, triggering the reset() methods for all registered Resettable classes through abstraction (DIP) to invert dependencies. This focuses solely on one clear aspect of the game's behaviour (SRP).

Resettable

The Resettable interface indicates entities/items that are eligible for resetting (ISP). It introduces a reset() method, callable by ResetAction() for global resets (DIP, LSP). This offers flexibility in implementing classes with custom reset behaviors, such as despawning enemies on player death or restoring the boss's health to maximum if players fails to defeat it.

RemoveActorAction

The RemoveActorAction manages actions to remove actors from the map and unregister them from the ResetManager class. It strictly adheres to SRP by concentrating solely on actor removal and unregistration. This design promotes better future implementations and enhances potential extensibility (OCP).

While it's possible to use the location.removeActor() method directly, introducing a new class allows us to group relevant methods together for consistent and easily understandable code. This approach also supports future extensibility (OCP) by enabling the addition of extra instructions to be executed if necessary.

DroppedRuneManager

Similar to the ResetManager, this class is designed to exclusively manage the dropping of runes from any entity (SRP). It registers and tracks each dropped rune from entities, storing them in the registry. Additionally, the DroppedRuneManager facilitates the modification of dropped runes, enabling their removal from the map through operations like location.removeItem().

While it's feasible for the Rune class to implement the Resettable interface (ISP), it's not preferred because it lacks control over the despawn action of the rune. By utilizing a singleton/registry class, we gain better control over individual rune actions. Perhaps for future improvement, we can even introduce features like "Despawn protection" to prevent certain runes from being despawned (OCP).