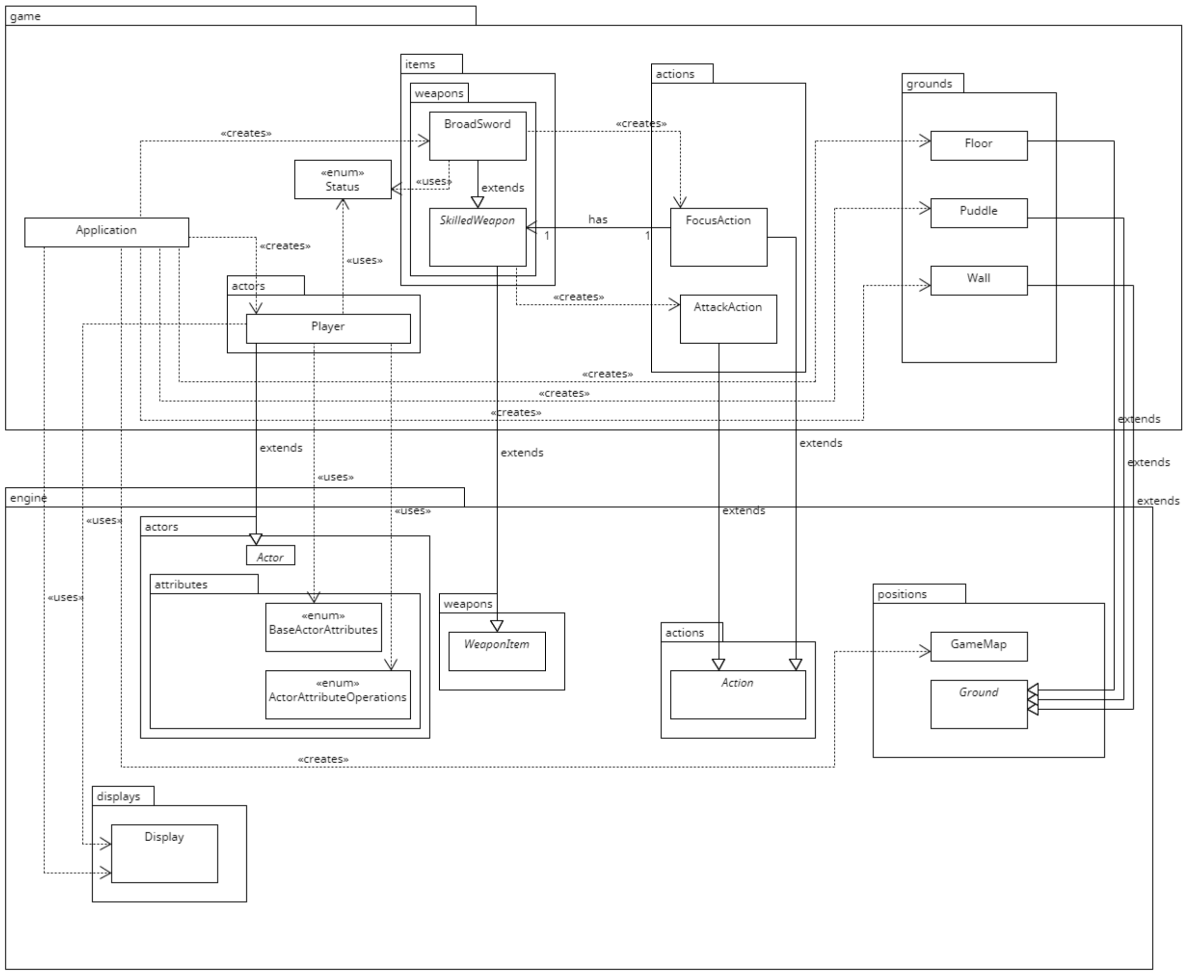
REQ1 UML Diagram.



The primary objectives of REQ1 are to define the Player actor, the Broadsword weapon, and its special skill 'Focus'.

To do this, I made the Player class rely on the Display class to show game details during each turn. It also uses BaseActorAttributes enum class and ActorAttributeOperations enum class to update the Player's attributes like stamina. Having Player use the Status enum class enables us to assign flexible statuses to Player, we can add a new status to the Player without modifying the existing code, which adheres to the Open/Closed Principle (OCP).

For the Broadsword and skill implementation. I made an abstract SkilledWeapon class that extends the WeaponItem class from the engine. This class allows us to keep track of the remaining turns of the skill and whether it's been toggled on. Since weapons share some common attributes and methods, we abstract these identities to avoid repetitions (DRY).

The Broadsword class, which possesses the Focus special skill, extends the SkillWeapon class and depends on the Status enum class. This dependency helps identify weapons with different statuses in future game implementations.

To implement the Focus skill, FocusAction class is created. FocusAction's constructor accepts a SkilledWeapon as an argument and has an association with SkilledWeapon. FocusActions uses methods defined in abstract class SkilledWeapon in order to activate and keep track of the Focus skill's duration.

This implementation adheres to the Dependency Inversion Principle (DIP) as FocusAction relies on the abstraction of a SkilledWeapon instead of concrete details of its implementation to activate the skill. As new weapons with skills are added, we can just have them inherent from SkilledWeapon, we do not need to modify the code in order to accommodate new skilled weapons in the future, this follows (OCP).

Alternative Implementation

Use Enums to keep track of the status of the 'Focus' skill. This implementation can be more simple and straightforward. It also involves less abstractions, which reduces complexity. It may be advantageous to use an enum implementation in a small-scale game. However, it is less extensible in the future, as we add more weapons with different special skills and varying durations.

Advantages:

Extensibility

The advantage of this implementation over an Enum implementation is: We do not have to create a new Enum class for every new skill we add in the future. Thus, this implementation is more extensible.

Flexibility

This implementation accommodates weapons with diverse behaviours and attributes. If different weapons require additional methods or attributes, we can implement them within the respective subclasses without affecting other parts of the code.

Disadvantages:

Complexity

The use of abstract classes and inheritance can introduce some complexity to the codebase. As our game grows and we add more weapons, it can become complex and harder to manage. This may make it difficult for new collaborators to understand and use the codebase, which can lead to maintenance challenges.

Tight Coupling

Inheritance creates a tight coupling between parent and child classes. Changes to the abstract class can impact all its subclasses, potentially introducing unintended side effects.