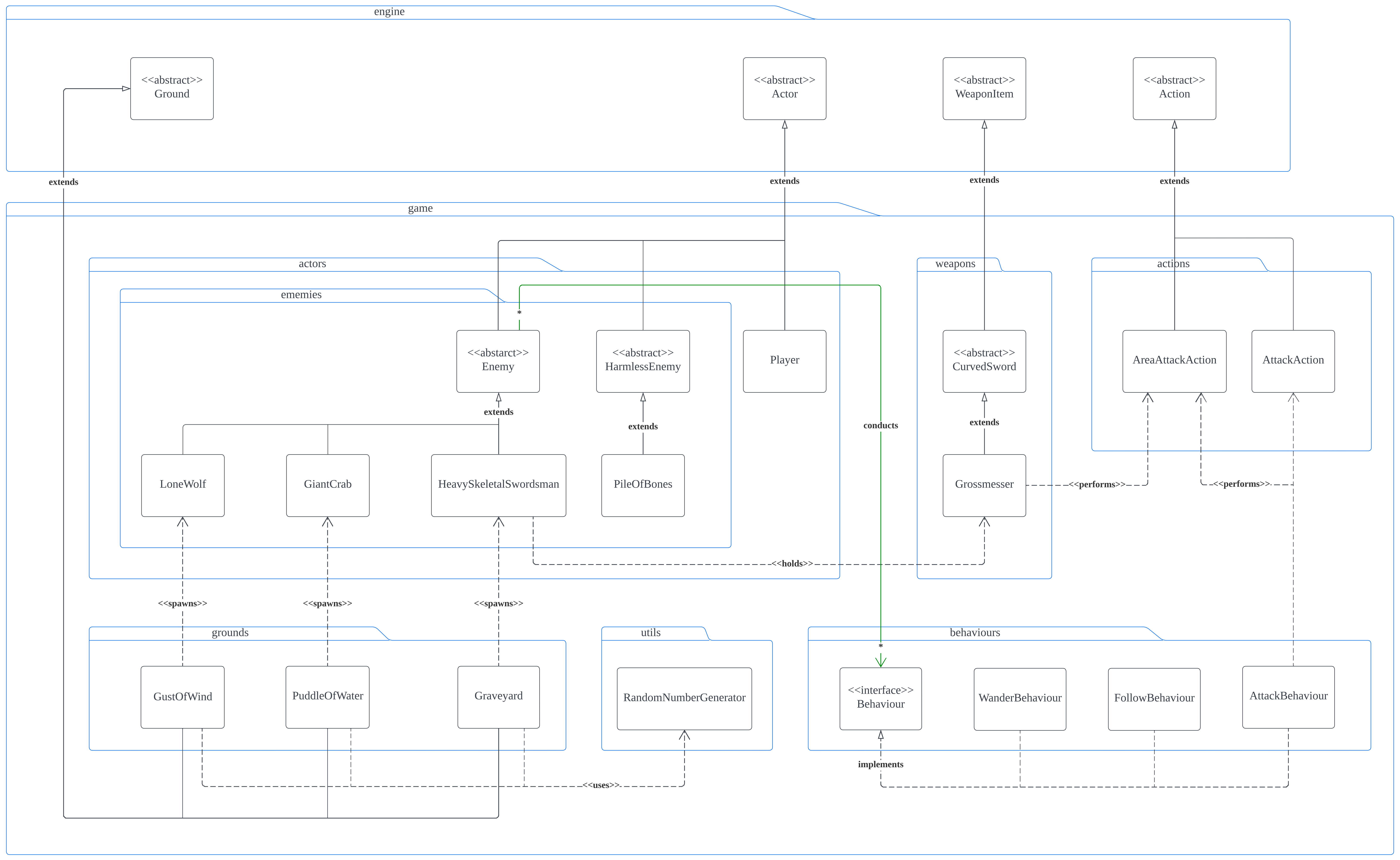
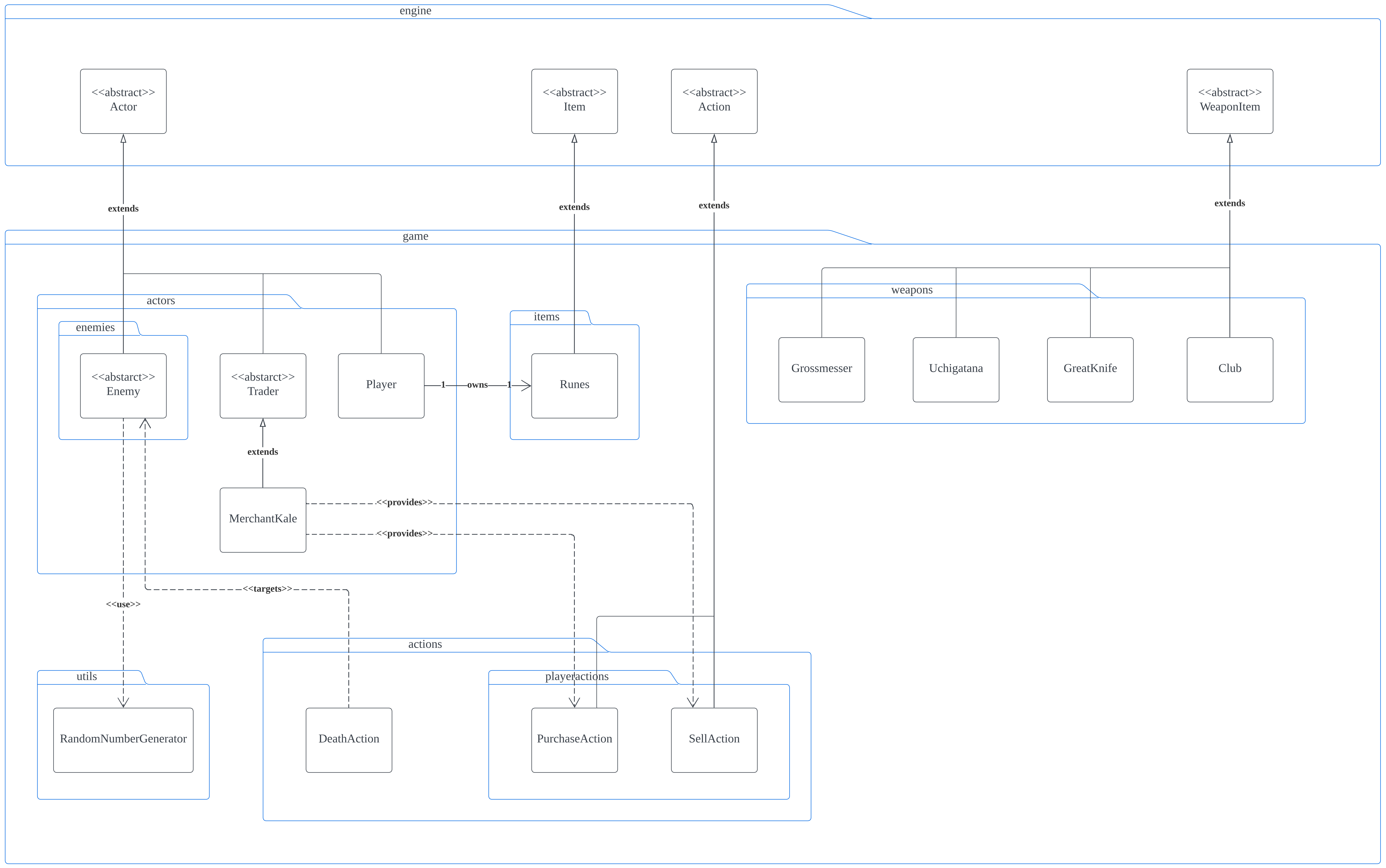
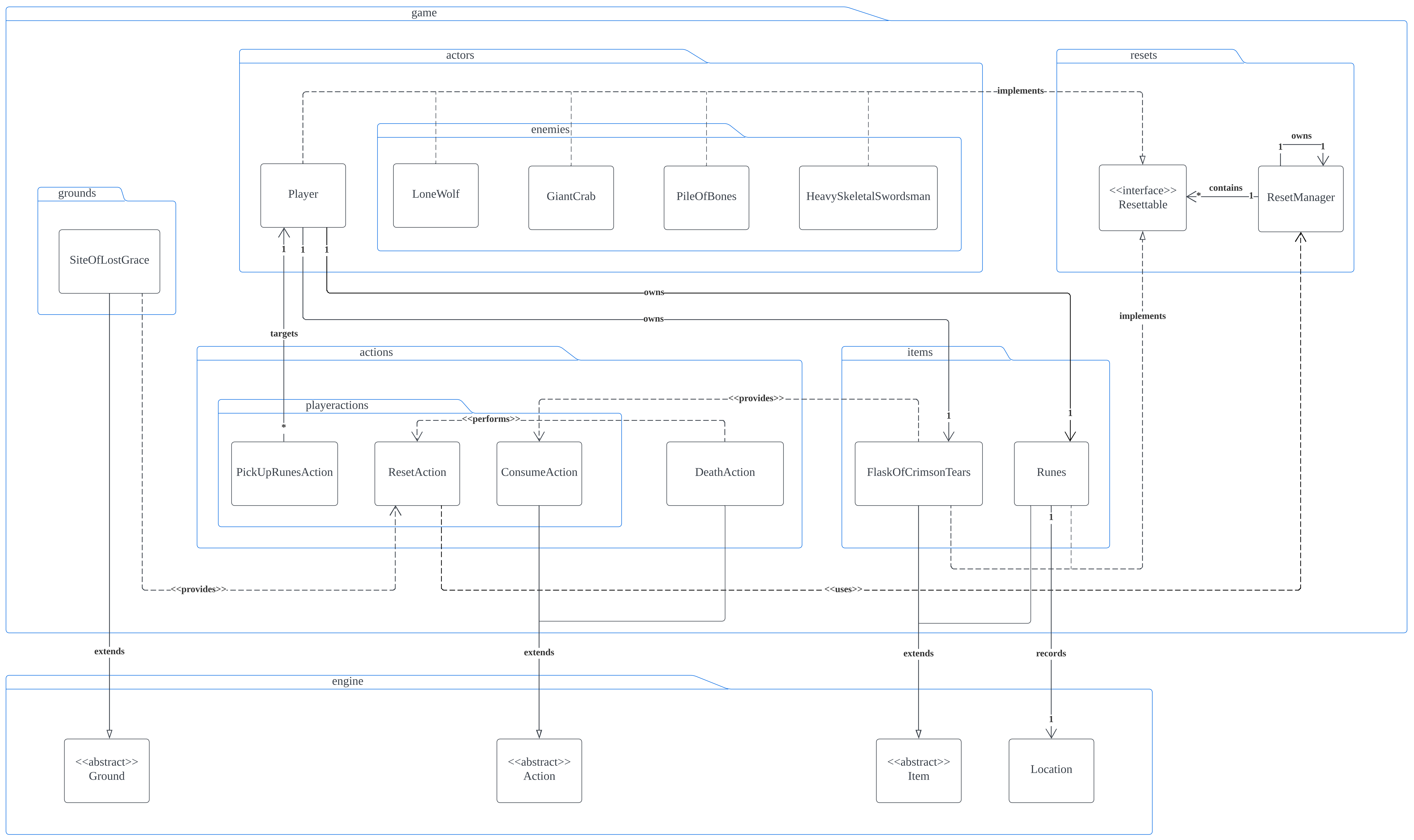
Requirement 1



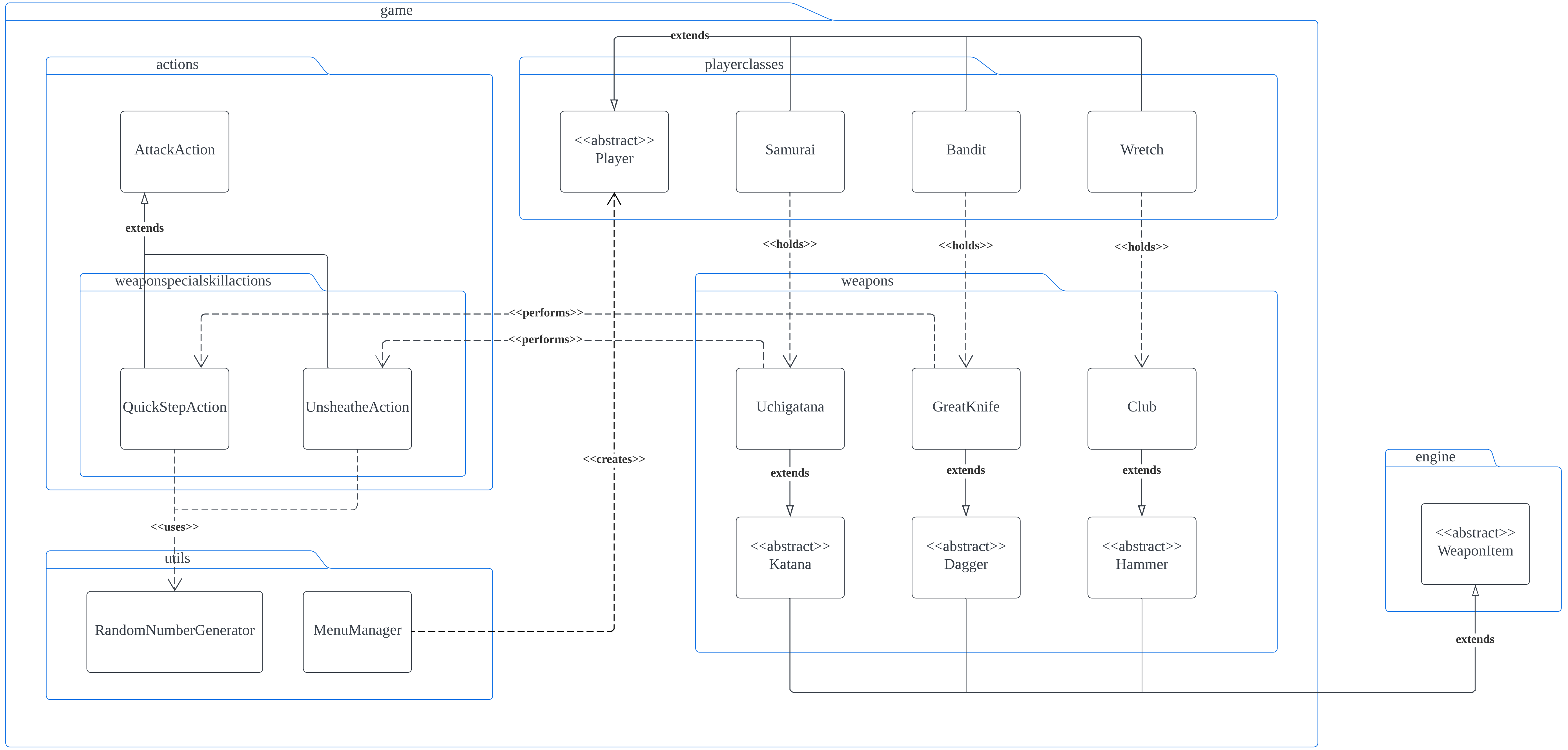
Requirement 2



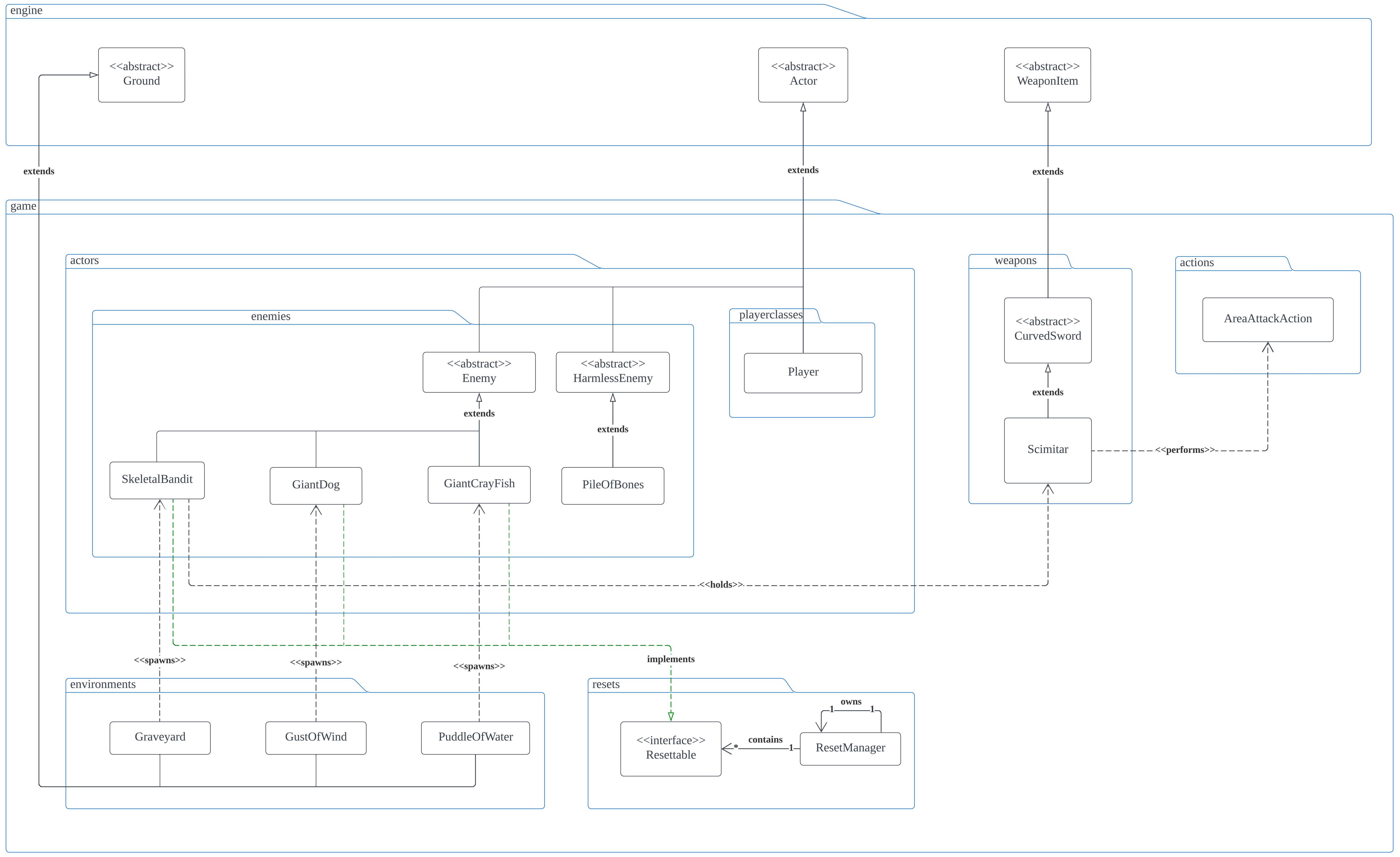
Requirement 3



Requirement 4



Requirement 5



Design Rationale

Requirement 1

The UML diagram above represents an object-oriented system for Game Requirement 1 in which there are three types of environments where each has a different chance to spawn its respective enemy at every game turn. The enemy can only attack another type of enemy unless an area attack is performed. Besides, the enemy will start to follow the player once the player comes closer to the enemy, else they will just wander around the map. In addition, there is a unique enemy known as "Pile Of Bones," which appears only when a Heavy Skeletal Swordsman is defeated and would not cause any harm to the player. If the Pile Of Bones is not defeated within three turns, the Heavy Skeletal Swordsman will be revived with full health. Besides, “Grossmesser” a weapon that is carried around by a Heavy Skeletal Swordsman can perform a single attack or spinning attack (area attack) .

To fulfil Requirement 1 part A, three new classes called “Graveyard”, “GustOfWind” and “PuddleOfWater” are created where each is the environment for their respective enemy. Each environment class has a different chance of spawning its corresponding enemy, therefore each environment class can use the “RandomNumberGenerator” to decide when to spawn their corresponding enemies on the location, based on specific spawning chance provided. To follow the DRY (Don't Repeat Yourself) principle, the classes for Graveyard, Gust of Wind, and Puddle of Water inherit the “Ground” abstract class to prevent repeating code as the “Ground” class defines all the attributes and methods that are common to all types of environments. Besides, this approach also follows the Single Responsibility Principle (SRP) as each subclass is responsible for defining its unique chance of spawning and instantiating its corresponding enemy.

In requirement 1 part B, we create three new classes for hostile creatures: “HeavySkeletalSwordsman”,”LoneWolf”, and “GiantCrab”. All three classes inherit from a parent abstract class called “Enemy”, which itself is derived from the engine code’s abstract class “Actor”. This approach follows the DRY principle which avoids repetition of code since all the enemy classes share common attributes and methods. Rather than defining these attributes and methods in each enemy class, they are defined once in the "Enemy" abstract class and inherited by all its subclasses. To elaborate, instead of coding all similar implementations in every different enemy class, we can just code a generic one in the “Enemy” parent class and then call super() in the subclasses and add some unique functionality easily. Besides, creating an “Enemy” abstract class can also reduce the dependency on the actor class (between specific enemy classes with the “Actor” abstract class”), as well as separating the functionalities of an NPC from that of players.

In our initial design, we treat Pile Of Bones as a ground, since they have no hit points, no item and weapon inventory, no attack accuracy and no despawn chance. However, there are a few limitations in this approach. Since player can perform attack action to pile of bones and they will remove and revives Heavy Skeletal Swordsman after being hit within three turns, logically they must have same precedence level as typical enemies because player can only attack an enemy and after a ground is added it cannot be removed. Besides, Ground has the lowest display level in this game, in the scenario where there is an actor or item in the same location as pile of bones, pile of bones will not be display if it is treated as a “Ground”. Also, if the pile of bones is implemented as a type of ground, it would also be difficult to keep track the number of rounds that the pile of bones has taken, revive the Heavy Skeletal Swordsman and remove itself from the map after being hit, more dependencies will be introduce for the implementation. Hence, “PileOfBones” should not be implemented as a type of “Ground”.

This led to a second alternative where we treat “PileOfBones” as an actor and is a subclass of “Enemy” abstract class. However, Pile of Bones does not have the ability to attack, follow or wander which distinguishes it from typical enemies and can be considered harmless. The limitations of this alternative is that we have to manually initialise the irrelevant instance variables to the Pile of Bones such as despawn chance and attack accuracy via calling the super() in the constructor which we used to inherit the other harmful enemies like HeavySkeletalSwordsman, LoneWolf and GiantCrab. This makes the system not robust because theoretically based on specification, “PileOfBones” is not having any of the attributes mentioned above as they will not despawn by chance and they are harmless to the others where no attack will be executed by them.

Therefore, this led to our third approach where in order to differentiate between typical enemies and harmless enemy we create an abstract class called “HarmlessEnemy” which has the same level of inheritance as “Enemy” abstract class. This approach avoids turning the "Enemy" abstract class into a "god class" that knows all aspects of harmful and harmless enemies, which would go against the SRP principle and by overriding the playTurn method from “Actor”, the number of rounds that Pile of Bones has taken can be kept track easily without introducing unnecessary dependency. Moreover, this approach is also more robust and extensible in future because if there are new types of harmless enemies introduced to the game, only a new subclass need to be created without making any changes to the “HarmlessEnemy” parent class, this follow OCP principle.

Furthermore, instead of performing a single attack, Giant Crab can decide to attack (slam) all creatures within their surroundings, so a class called “AreaAttackAction” is created that takes on the responsibility of handling any area attacks which follows the SRP principle (separating responsibility of normal attack to this unique (area) attack) where each class has a clear and distinct responsibility, and only one reason to change.

Heavy Skeletal Bandit carries around a weapon called Grossmesser, so a new class “Grossmesser” is created which extends “CurvedSword” abstract class which itself is a subclass of the “WeaponItem" abstract class in the engine's code. This inheritance structure is designed to account for other weapons that belong to the curved sword type. By having this approach, if another curved sword type weapon is added in the future, it can be easily maintained and implemented by creating a new subclass of the “CurvedSword” abstract class, making the system more extensible in future. This follows the Open-Close Principle (OCP) and Single Responsibility Principle (SRP) of the SOLID principles by allowing for easy extension of the code without modifying existing parent classes and ensuring that each subclass has a single responsibility (or their own special functionalities without letting the “CurvedSword” class to handle/manage their uniqueness). Not only that, if more weapons of type “CurvedSword” are added into the game in future, repetition of code can be avoided as calling super() will help subclasses to inherit the commonalities from parent class. Since Grossmesser can allow the user and Heavy Skeletal Swordsman to perform a single attack or spinning attack thus, it also has a dependency to “AreaAttackAction” and “AttackAction” class.

Requirement 5

The UML diagram above represents an object-oriented system for Game Requirement 5 in which it is a combination of Game Requirement 1,2,and 3 .The only main differences are that more types of enemies are added to the game and the map will be divided into two halves, with the West side spawning Heavy Skeletal Swordsman from the Graveyard, Lone Wolf from a Gust of Wind, and Giant Crab from a Puddle of Water. On the East side of the map, Skeletal Bandit will spawn from a Graveyard, Giant Dog from a Gust of Wind, and Giant Crayfish from a Puddle of Water.

Similar to the enemies in requirement 1 three enemy subclasses are created which are "SkeletalBandit", "GiantDog", and "GiantCrayfish". This follows the DRY principle which avoid repetition of code and OCP principle where if there is other enemies introduce in the future , only a new enemy subclass need to be created without modified the "enemy " abstract class.

In our design, we will make use the GameMap's width to divide the starting map into west and east half .By comparing the location of the each ground to the middle x coordinates that calculated using width, we can determine which side of the map a location belongs to. If a location's x coordinate is less than the middle x coordinates, it will belong to the west side of the map and spawn Heavy Skeletal Swordsman, Lone Wolf, and Giant Crab. On the other hand, if a location's x coordinate is greater than the middle x coordinates, it will belong to the east side of the map and spawn Skeletal Bandit, Giant Dog, and Crayfish