Design Rationale for Requirement 1

The diagram represents an object-oriented system for the various environments, enemies, and weapons in the game.

Each enemy type is represented in its own class, which all extend the Enemy abstract class. This is done because they share common attributes and methods which all enemies use. The Enemy abstract class itself extends the Actor abstract class, as it also shares common attributes and methods with other Actor subclasses in the game. The Giant Crab enemy's special slam attack is implemented as its own class called SlamAttack, which is a subclass of AttackAction, as it shares attributes and methods. The Heavy Skeletal Swordsman enemy's Pile of Bones ability is also implemented as its own PileOfBones class, which is a subclass of the Action abstract class. The ability of the HeavySkeletalSwordsman enemy to use the Pile of Bones ability is represented through it implementing the PileOfBonesAble interface.

Similarly, each environment is represented in its own class, which all extend the Ground abstract class. This is done because they share common attributes and methods which all grounds use. They also all implement the Spawner interface, for the enemy spawning functionality.

Finally, the Grossmesser weapon is represented in its own class, which extends the WeaponItem abstract class. This is done because it shares common attributes and methods which all weapons use. Its special attack is implemented as its own class called SpinAttack, which is a subclass of AttackAction, as it shares attributes and methods.The ability of the Grossmesser to use the special spin attack is represented through it implementing the SpinAttackAble interface.

Adding new enemies, environments, and weapons by extending these abstract classes achieves abstraction, and therefore adheres to the Open-closed Principle. In other words, extending the Enemy or Ground classes to add new enemy or environments into the game is easy and can be done without modifying existing classes.

Creating and implementing interfaces adheres to the Dependency Inversion Principle, by creating an abstract interface, and not having concrete classes depend on each other.

It also follows the DRY principle of not repeating code, by reusing common attributes and methods through inheritance.

A disadvantage of implementing and extending the system with abstract classes is that the logic and code is split among several classes rather than in one place, which can make it difficult to understand. The size of code is also larger, and takes longer to code.

Changes in Assignment 2

Each enemy now extends their respective enemy type abstract class (CanineEnemy, SkeletalEnemy, CrustaceanEnemy). This was done to easily create new enemies which shared certain capabilities, which are implemented using the Status enum class. The three environments which spawn enemies now extend SpawningGround, an abstract class which extends Ground, to reuse similar code and better adhere to the DRY principle. An advantage of this approach is that it allows new environments to be easily implemented in the future. On the other hand, one disadvantage is that the actual code and logic for spawning enemies is spread across multiple classes, making it harder to understand.

The Grossmesser weapon now extends the CurvedSword abstract class, which itself extends the WeaponItem abstract class. This was done keeping in mind that new weapons which shared the ability to perform the special area attack(formerly SpinAttack) action in Requirement 5, and reuse code via inheritance. An advantage of this approach is that it allows new weapons which can perform the special area attack to be easily implemented in the future. However, a disadvantage is that the actual code and logic for the weapons and the special area attack is spread across multiple classes, making it harder to understand.

The Pile of Bones action now instantiates a Pile of Bones skeletal enemy. Having PIle of Bones extend SkeletalEnemy prevents other skeletal enemies from attacking it. It also uses the PileOfBonesReviveAction action, which has a 1 to 1 association with the enemy it is reviving into, stored as a private variable. An advantage of this approach is that it allows new enemies which can perform the Pile of Bones action to be easily implemented in the future. However, a disadvantage is that the actual code and logic for the Pile of Bones action is spread across multiple classes, making it harder to understand.