Requirement 5 - More Enemies

The diagram represents an object-oriented design for requirement 5 and displays the necessary classes and their relationships with each other in order to fulfil the needs of the overall system. The features requested by requirement 5 are additional to what has been defined in requirement 1, hence both diagrams will be similar and anything mentioned within the rationale of requirement 1 will also hold true here.

The addition of new enemies in this requirement, has given me reason to include two new abstract classes in the enemies package, the first of which being the GiantEnemy abstract class which is inherited by the GiantDog, GiantCrab and Giant Crayfish concrete classes, and the second, the SkeletalEnemy class which is inherited by the HeavySkeletalSwordsman and Skeletal Bandit classes. Both these classes extend the Enemy class and hence fulfil the Liskov substitution principle as their children can be substituted for any Actor object within the system. The reason I have chosen to create an additional layer of abstraction through these classes is to apply the DRY principle, HeavySkeletalSwordsman and Skeletal Bandit both use the SkeletalDeath action, this can be implemented within the SkeletalEnemy abstract class instead of being repeated in both concrete classes, the same is true of GiantDog, GiantCrab and Giant Crayfish which all use the Slam skill, repeating this code can be avoided by using the GiantEnemy abstract class which implements the Skilled interface and returns the Slam SkillAction. The open and closed principle is also observed here as the GiantEnemy and SkeletalEnemy classes are not open for modification but can be extended easily, the same was true for the Enemy class, as we have easily extended it here into two new abstract classes. The con of this approach is that we must implement a Species enum and add it to each Enemy's CapabilitySet individually to ensure Enemies of the same type/species will not attack each other. Instead of doing this, we could have, for example, had LoneWolf and GiantDog inherit from a Canine abstract class and make it so all Canines cannot attack each other. However, this results in more code repetition and a less extensible design should more giant enemies need to be added.

Another important decision to highlight is the addition of the Sword Abstract class which extends WeaponItem and implements Skilled. Since the Scimitar and Grossmesser both use the SpinningAttack skill but have different stats, it's better to create a Sword abstract class which already returns the SpinningAttack SkillAction as an implementation of the Skilled interface, and allow both concrete classes to inherit from it. This adheres to the DRY principle and LSP principle as well as the open and closed principle as the Sword abstract class is not open to modification, however can be extended in the future by more WeaponItems that use the SpinningAttack skill. A con of this approach however is that Scimitar, Grossmesser and the SpinningAttack SkillAction are now tightly coupled, in the future, if Weapons are allowed to change their skills at runtime, this could become difficult to refactor.

Lastly, the requirement for different Enemies to be spawned on either the west or east side of the map has already been solved by the inclusion of the Spawner abstract class as checking whether a Spawner is on the east or west side of the map can be implemented trivially by the Spawner class and inherited by the Graveyard, GustOfWind and PuddleOfWater classes further adhering to DRY principles.