Requirement 1: Environments and Enemies

The diagram represents an object-oriented design for requirement 1 and displays the necessary classes and their relationships with each other in order to fulfil the needs of the overall system. The diagram is organised by package; in the top half, we have classes from the provided game engine, while the bottom half displays the classes and packages within the game package that have been modified and extended to implement the required features.

After receiving feedback from our initial Assignment 1 designs, we have made significant changes to the system design, which have helped us reduce the number of unnecessary classes and decrease the overall coupling and dependancy between classes. These changes have also improved the overall maintainability and extensibility of the system. We have thoroughly tested the updated design and are confident in its ability to meet the requirements of the project.

The first of the significant changes is the removal of the mortal abstract class. This class inherited from actor and provided an abstract method to obtain a death action that was specific to the actor as well as an overriden hurt method to perform conscious checks outside of attack actions. Previously, in our original A1 document, we had both players and enemies inherit from this class, as they had different effects on the world on death. However, this approach involved the use of downcasting actors provided by the engine actor iterator into our custom mortal child class in many locations, including the attack action and behaviour classes. This method of downcasting is generally considered bad practise as it breaks the open and closed principle in a few cases. In the future, we may want to implement more actors that can be attacked and interact with other actors; however, we will be forced to make them inheritors of mortal due to the majority of the combat system being written to accept mortal as a parameter; hence, the system becomes more difficult to extend smoothly without modifying legacy code. Additionally, this approach also breaks the interface segregation principle, as new actors who do not necessarily need to inherit from mortals will need to do so in order to participate in combat (attacking, skills). Hence, to remedy this, we have removed both mortal and multiple death actions from the game. allowing us to handle all our combat logic as actors; in its place, we use capability checking within the attack action to ascertain the actor's role in the system in order to trigger the appropriate response within a single universal death action.

The second significant change to the system is a revision to how skills are handled and presented to both the player and actor. Previously, we had an interface called skilled that denoted an actor capable of performing skills as well as a separate abstract class of actions called skill actions. After some careful deliberation, we realise that both of these classes are unnecessary and have swapped to a different approach all together. Now that skills are like any other action and inherit directly from the action superclass, enemies are also no longer responsible for providing a skill; instead, we obtain skills by overriding the getSkill() methods for weapons, which are provided in the engine class. This allows us to reduce the overall coupling between skills and enemies, as now the relationship has been simplified to:

weapon creates skill → enemy receives skill action from weapon → enemy executes skill. This adheres more closely to the single responsibility principle. Furthermore, certain enemies who are capable of using an AoE skill now implement an AoECapable interface; this achieves the dependancy inversion principle as our abstractions do not rely on concretions but rather this interface.

Finally, we have added a despawn action into the game, allowing enemies to despawn naturally through the use of the game engine. This adheres to the single responsibility principle once again, as the responsibility of enemy despawns has been abstracted into a separate class rather than being handled directly by the enemy, thus avoiding unnecessary coupling.