Design Rationale

Name: Koe Rui En

ID: 32839677

<u>REQ 3: The Wandering Undead</u>

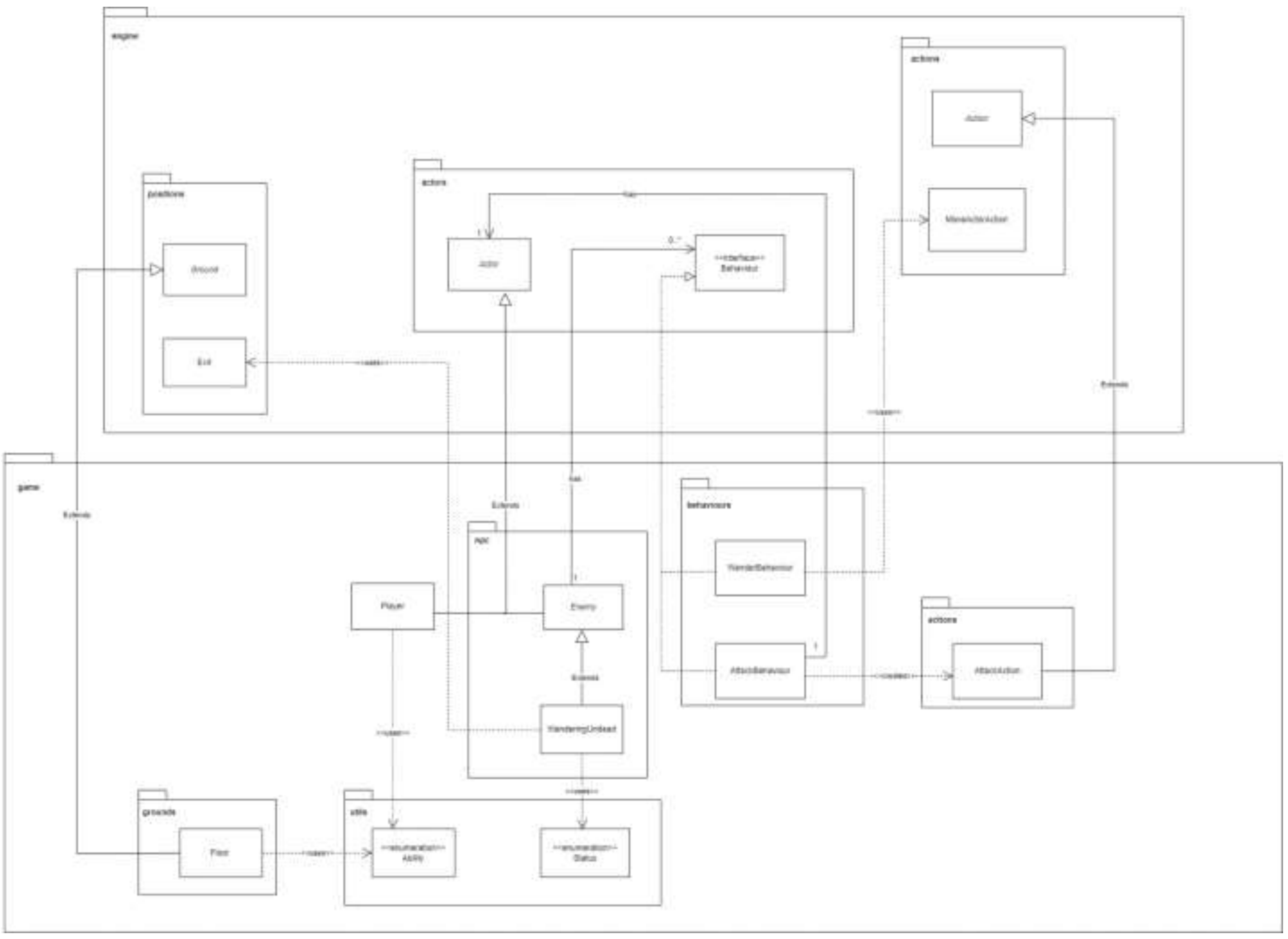

Figure 1: UML class diagram of REQ3 – The Wandering Undead

The provided diagram shows the entire system implementation for Requirement 3, The Wandering Undead. The primary goal of this design is to implement the first enemy that the player encounters which is Wandering Undead who has various behaviours, such as attack and wander and to check the actor so they can enter the floor. There are new classes are proposed in this system, WanderingUndead class, AttackBehaviour. The existing classes in the game package, Player class, Status class, AttackAction class, WanderBehaviour class, Ability class and Floor class, will interact with new classes that mentioned above and also with the existing classes.

The Floor class extends from the abstract Ground class from the engine as well as the Player class extends from the abstract Actor class from the engine, as this was mentioned in the previous design rationale of REQ1: The Player. In this requirement, we decided to have an Ability enum class and Player class will be assigned an ability to do something, thus resulting in a dependency relationship towards this Ability Enum class. Furthermore, the Floor class also has dependency relationship with this Ability enum class to check the current actor who step on it whether they are qualified to pass this terrain. The main reason to have this Ability enum class is to assign different abilities to the player perform various tasks based on their abilities. Alternatively, if the Ability enum class does not be used, we will need to use if-else statement and instanceof operator (checking class type) in Floor class method to check the current Actor actual subclass which involves extra dependencies. Thus, our design adheres to the Open/Closes Principle (OCP) because assigning a flexible ability to the actor so that the existing code in the Floor class does not need to be modified even if more actors will be introduced in the later implementation of the game.

Furthermore, the WanderingUndead class extends from the abstract Enemy class also stated in the previous design rationale of REQ2: The Abandoned Village's Surroundings. In this requirement, we decided to have the WanderingUndead class have a dependency relationship with the Status enum class to check every actor who it encounters before performing any actions to them. Therefore, the Wandering Undead can prevent to attack their own kind and only attack other actors including the player. Alternatively, if the Status enum class does not be used, we will need to implement if-else statement and instanceof operator (checking class type) in AttackBehaviour class to ensure that enemies of the same type will not attack each other which involves extra dependencies and violates the Open/Closed Principle (OCP) because when new enemies are added, the existing code of the AttackBehaviour class have to be modified to ensure the condition of enemies not attacking their own type holds true at all times. Besides, the WanderingUndead class has dependency with Exit class to determine its surrounding contains any actors. This can help this enemy to decide its behaviour to be performed towards that actor.

Since every enemy (inhabitant) regardless Wandering Undead share same behaviour, we also decided to have abstract Enemy class have association relationship with the Behaviour interface class which will be implemented by the WanderBehaviour class and the AttackBehaviour class. The AttackBehaviour class will handle any attack actions that are executed by the enemies and not initiated by the player, so this AttackBehaviour class will create AttackAction class to perform the attack action, indicating there is a dependency relationship between AttackBehaviour class and AttackAction class. In addition, the AttackBehaviour is also associated with the Actor abstract class so that every enemy can perform attack actions on that actor. Besides, the WanderBehaviour class will handle the movement of all enemies, so this WanderBehaviour class has a dependency relationship with the MoveActorAction from the engine. Our design is not only can reduce multiple dependencies between enemies and their behaviours, but it also aligns with the Dependency Inversion Principle (DIP) where all of behaviours depend on abstractions instead of concrete implementations. In such way, we can prevent to perform modifications to our existing code in all our Enemy classes if any new Behaviour class is introduced can be easily extended in later implementation.