Design Rationale

Tong Jet Kit, Aaren Wong Cong Ming, Yew Yee Perng

Requirement 5

This UML class diagram will showcase the relationship between the enemies and players as actors in the game. The goal of the class system is to show the characteristics, abilities and behaviours of the actors in the game in modularity.

To begin with, the enemies (Heavy Skeletal Swordsman, Lone Wolf, Giant Crab, Skeletal Bandit, Giant Dog, Giant Crayfish) that play the role of foes in the game which extends the Bone, Canine and Crustaceans class. By creating these abstract Species classes, it reduces code repetition for enemies of the same kind but different territories such as Lone Wolf and Giant Dog who share the same attributes and methods. Furthermore, these abstract classes extend the Actor class thus inheriting all its properties, supporting the DRY principle for classes that share similar attributes and methods. Graveyard, Gust of Wind and Puddle of Water extends abstract class Ground as they are a type of ground. Location has an association with GameMap to get the coordinates in the game to differentiate the east and west side of the map, where GameMap has an association with ActorLocationIterator to provide the coordinates for the actors. All species also have a dependency with the abstract class Actions, that takes care of the actor's action in the game such as DeathAction. DeathAction is made abstract and has two subclasses, EnemyDeathAction and PlayerDeathAction. The creation of these two subclasses are used to differentiate the death action in the game of the enemies and the player. Hence, the Single Responsibility Principle is adhered. Both of them handle the death action of the enemies and the player, so the player and the enemies depend on them respectively. The enemies have an association with the EnemyWeapon as it handles the source of damage dealt to the players or actors in the game. However, not all have an association with EnemyWeapons as they do not equip a weapon. For those enemies without any weapons, they depend on IntrinsicWeapon to deal damage to their opponents in the game. Moreover, EnemyWeapon extends the abstract class WeaponItem as they are a weapon in the game and they also extend from the Item abstract class since weapons exist as Items in the game as they are portable hence greatly reducing code repetition. Intrinsic Weapon implements the Weapon interface since they are also a type of weapon. It does not extend WeaponItem since the Intrinsic Weapon such as fists are not portable.

Besides that, the enemies have an association with behaviour Interface as they have a collection of behaviours that appears in the game. It acts like a console menu for the enemies. For example, when an actor gets within the range of an enemy, the enemy would react by showing a behaviour resulting in an attack to the actor. This allows them to access the dependencies of Behaviours such as Wander Behaviour and Follow Behaviour. With that, they are able to wander around the map or follow the player when in range.

CapabilitySet has a dependency on the Capable interface. Actors and Weapons that have an association with the capabilitySet are used to take account of abilities that the Classes have. It stores all the information for all the weapons including intrinsic weapons that contain any special ability. Furthermore, Capable ensures that when more associations are linked with CapabilitySet, we do not have to fix CapabilitySet to accommodate for new enemies or weapons added into the game by the Open Closed Principle.