Design Rationale

By: MA_AppliedSession1_Group3

REQ 4: The room at the end of the forest

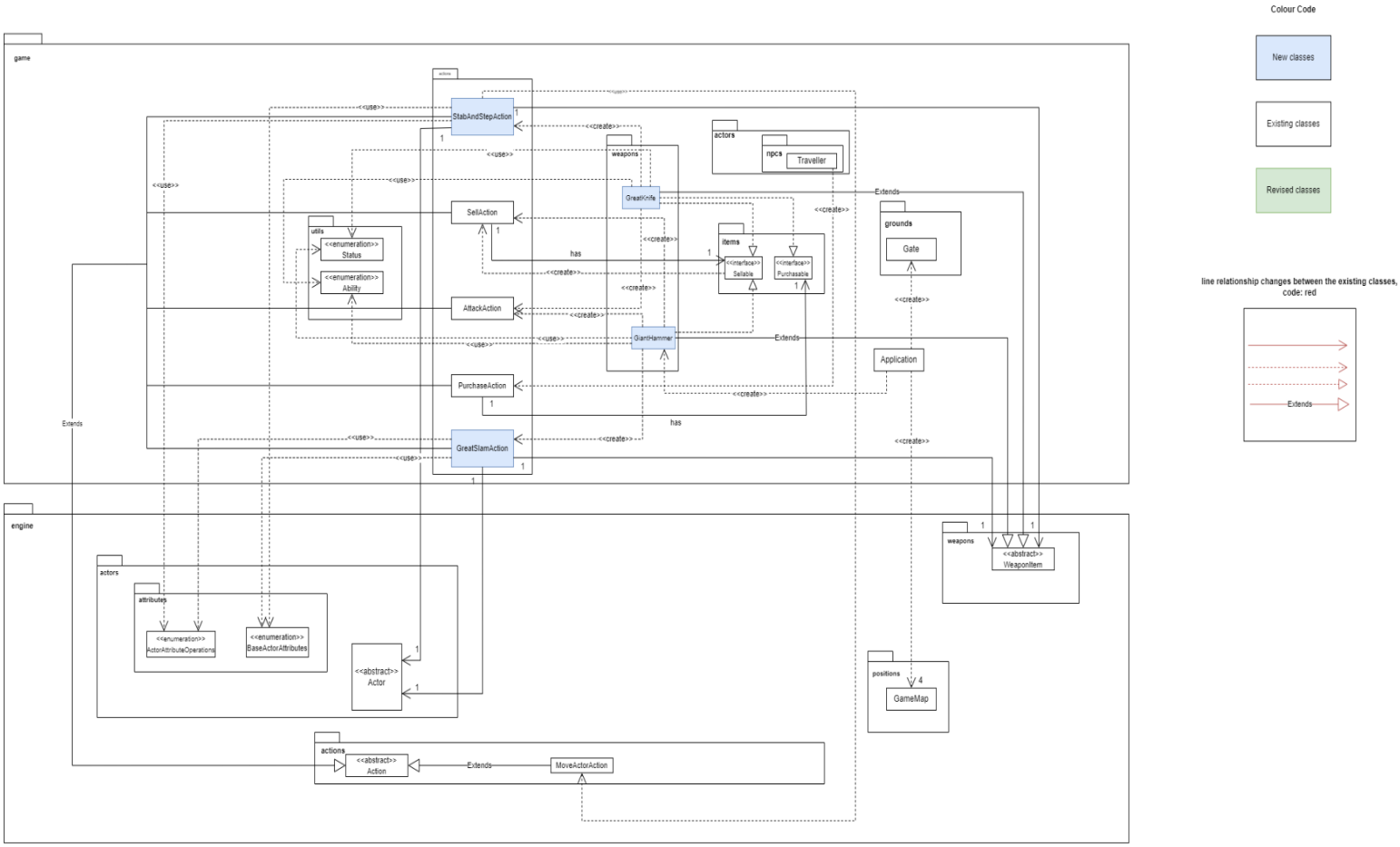

Figure 1: UML class diagram of REQ 4: The room at the end of the forest

The provided diagram shows the entire system implementation for Requirement 4, The Room at the end of the forest. The primary goal of this design is to create a new GameMap, which is Abxveryer Battle Map, and a gate is also created to allow the player to travel to the new game map. A Great Knife, a new weapon offered by the Isolated Traveller with its weapon skill, Stab and Step action, to allow the player to attack enemies and step away to safety within the same turn. A Giant Hammer with its weapon skill, Great Slam action, is created in this requirement and is located in the Abxveryer Battle Map. Both Great Knife and Giant Hammer can be sold, but only Great Knife can be purchased from the traveller. Thus, new classes: GreatKnife class, GiantHammer class, GreatSlamAction class and StabAndStepAction class. These classes will interact with the existing classes in the extended system.

GreatKnife and GiantHammer aren't like Broadsword in that they don't need to take a turn to activate their special skills, but instead they can just apply their specials to the enemy. Players can use these two weapons to attack enemies either normally or with special skills, so we need not only Attack Actions but also Special Skill Actions in our allowableActions(other…) method. For StabAndStepAction and GreatSlamAction, they need to know the target, stamina used and the weapon. These three pieces of information are the attributes of these two action subclasses. It's worth noting that for GreatSlamAction, target refers to the enemy that will be attacked by 100% of the weapon's damage.

Since I made the appropriate adjustments to everyone's code and design prior to this before doing REQ4, it also made my task easy (only implementation not design). Based on the design of REQ3, I just need to have both classes implement the Sellable interface and override the corresponding methods. At the same time, add the SellAction in their respective allowableActions(otherActor...) when the otherActor is TRADER. GreatKnife can be purchased from Trader, so it needs to implement the Purchasable interface and override the corresponding method, and then add this one item to the Traveller's ArrayList.

The core idea inside the execute method of StabAndStepAction is to execute AttackAction followed by MoveActorAction, which is to select the first unoccupied position found. And, for the GreatSlamAction, changing the damage dealt by a weapon really only requires changing the weapon's damage multiplier. First perform an AttackAction to attack the target, then set the damage multiplier to 0.5f, this ensures that the damage of the attack from now on is only 50% of the original damage. To get the “secondary” enemies, we need the main target’s location so that we can get its surroundings. This means that it’s better to store the main target’s location before attacking it. Otherwise, if the main enemy dies after the attack, trying to get the location at that point will report an error because the enemy is no longer in the game. Finally, check the surroundings and perform an AttackAction on every enemy in the surroundings. At this stage, we only care if there's an actor in a position, we don't care if the actor is an enemy or not, because according to the rules, the player himself also takes damage.

We encapsulate the logic required for that specific action in the specific sub action class. This is a good practice of Single Responsibility Principle (SRP). Also, new action can be introduced without affecting others. Changing the implementation of one action class does not affect other action classes. This approach makes the system more robust and easier to extend, thus adhering to OCP.