Design Rationale

By: MA_AppliedSession1_Group3

REQ 3: The Ancient Woods

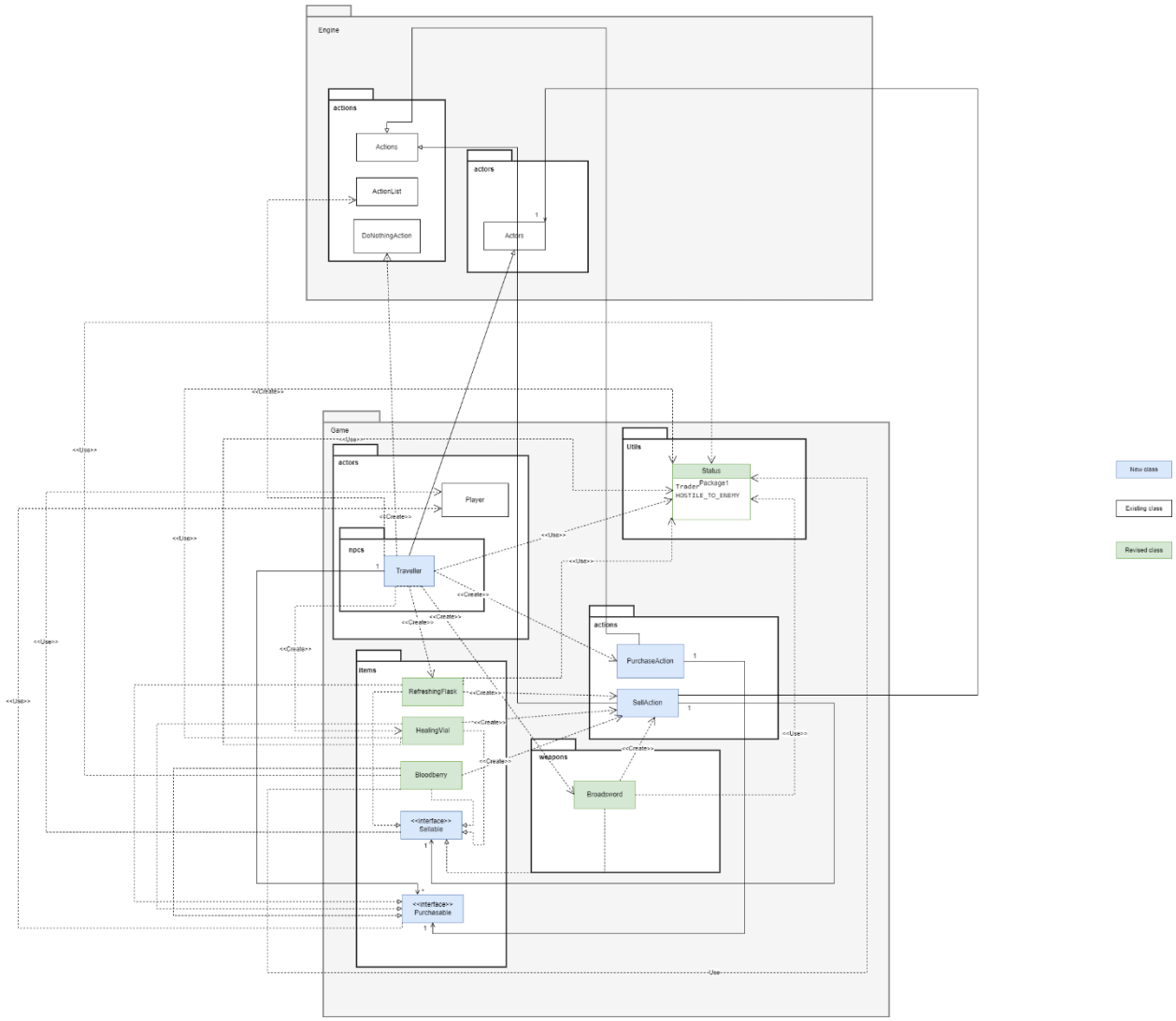

Figure 3: UML class diagram of REQ3 - The Isolated Traveller

The provided diagram shows the entire system implementation for Requirement 3, The Isolated Traveller. The primary goal of this design is to create an actor, which is known as Isolated Traveller, allows the player to purchase items from the traveller's inventory. The player can also sell items from their inventory to the traveller.

Purchasable part: Thus, we have implemented interface, "Purchasable" for Healing vials, Refreshing flasks, Broadswords which will be purchased by player. The class which implements getPurchasingPrice() and purchasedBy(Actor actor). Interface Segregation Principle (ISP): The Purchasable interface has a clear and specific purpose: it represents items that can be purchased in a game. This adheres to the ISP, which suggests that interfaces should be client-specific, with only the methods relevant to the client (in this case, items that can be purchased). This aligns with the Open/Closed Principle as these classes must utilise the "purchasedBy(Actor actor)" method in the "Purchasable" interface, while the extension for the "purchasedBy(Actor actor)" method is allowable. This means that these classes might have different implementations with the "purchasedBy(Actor actor)" and "getPurchasingPrice()" method. Moreover, the "Purchasable" interface only contains two methods for the implementation of these classes, it does not violate the Interface Segregation Principle.

PurchaseAction, extends Action class and have a parameter of Purchasable items(the item will be purchased), Encapsulation: The class encapsulates the item to be purchased (Purchasable item) as a private field, ensuring that this data is kept within the class and not directly accessible from outside. The execution of this action relies on the purchasedBy(actor) method of the Purchasable interface to handle the purchase logic. This promotes information hiding and encapsulation.

Traveller, extends Actor, will have a attribute: An ArrayList (purchasables) is used to store the purchasable items. This is an appropriate data structure for holding a collection of items. The configure() method in the Traveller class is responsible for adding purchasable items to the Traveller's inventory, making them available for purchase by other actors. This method is a form of menu configuration, as it sets up the list of items that can be interacted with in the game's trading system. In allowableActions, it will check whether the actor is Player or not, it will add the PurchaseAction to iterate all purchasable item in the purchasables list.

Sellable part: Thus, we have implemented interface, "Sellable" for Healing vials, Refreshing flasks, Broadsword, Bloodberry which will be sold by player. The concept is similar to the purchasable part, Interface Segregation Principle (ISP) and Open/Closed Principle, Interface Segregation Principle are used. The class which implements getSellingPrice() and soldBy(Actor actor). SellAction is similar to the PurchaseAction.

The allowableActions method in the Broadsword class is responsible for determining the actions that can be performed involving the Broadsword when interacting with another actor at a specific location. Thus, if the actor is Trader, the broadsword will create (new SellAction(this)) to the list actions to be implemented by the system (using game engine, like consume).

Code Reusability:

The design promotes code reusability through the use of interfaces and encapsulation. For example, multiple items can implement the Purchasable and Sellable interfaces, allowing for the reuse of common purchase and selling logic.

Modification done on classes created in Assignment 1 for REQ 1

Broadsword: Broadsword extended “SkillWeaponItem” class from A1 because it is unnecessary to create a new class for skill weapon item, we can simply use Ability.SKILL to distinguish the item which have skill or not. We make the class lesser but still works normally.