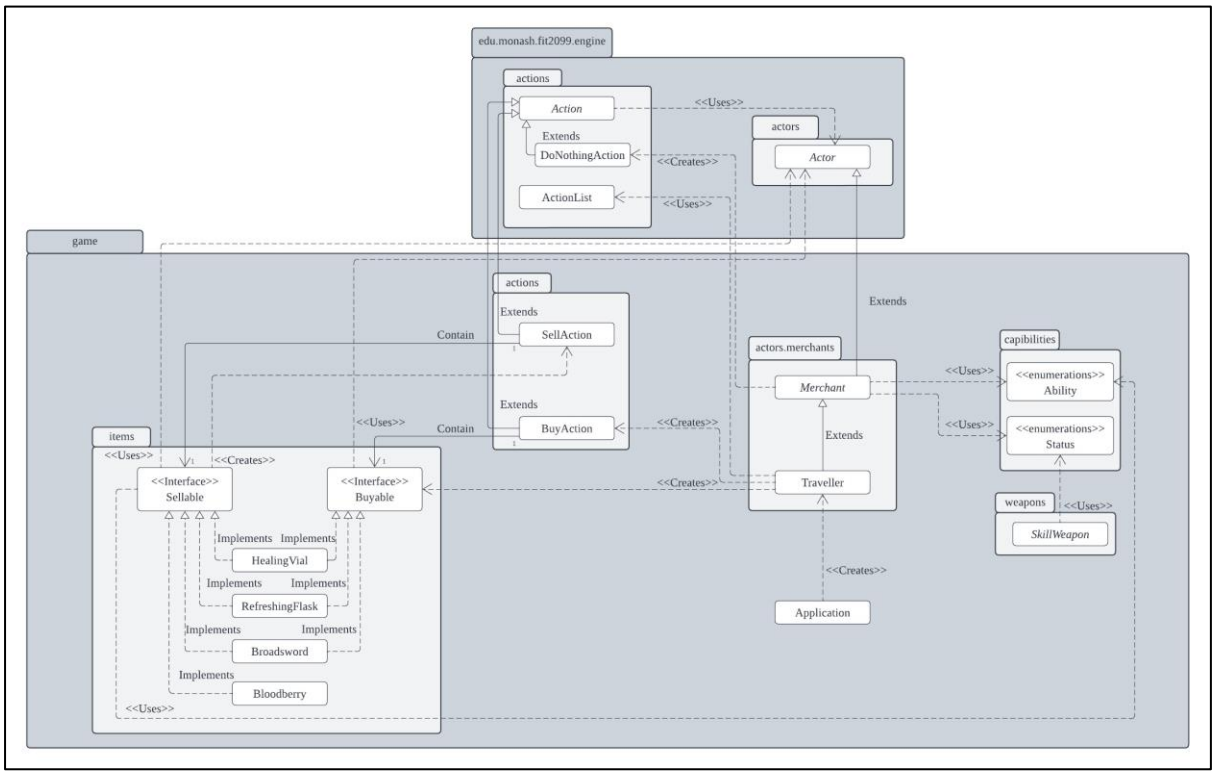
FIT2099 Assignment 2 Design Rationale

Requirement 3:



- Traveler class is extended from a Merchant abstract class which is extended from Actor class. Merchant abstract class acts as a parent class for all the actors that can buy or sell items to the player. Having a Merchant class allows us to prevent code repetition (DRY) since all the common characteristics of the merchants can be added into the Merchant abstract class so the subclass that inherits from the Merchant class will also have this characteristic.

- All the merchants will have a neutral status as they cannot be attacked by any of the actor in the game. The merchant will also have a trading ability so that if a player encounters an actor with this ability, they can sell items to them.

- Traveler can be added as a subclass of Ground with the following reasons:
  - Traveler does not need playturn method as it cannot move around. They also cannot attack or being attacked by other players, which fulfill the characteristic of a ground type.
  - Traveler should be able to sell items to the player. This can be simply done by returning a buyAction when the allowableAction method in the ground class is called.
  - Travelers does not need a HP. They don't necessarily need a wallet and item inventory since they can sell an item infinite number of times to the player and they have infinite amount of cash. Besides, if they bought an item from the player and the item is not sellable, it will be useless to add the item in the inventory.

  However, in our design, traveler is added as a child class of the Actor because it is possible that a wandering traveler is introduced to the game in the future. In this case, if the Traveler is implemented as a subclass of Ground, we might need to modify the whole class since a Ground

is not allowed to move. This will violate OCP principle and will cause us a lot of time to modify the code. Hence, the Traveller class is implemented as a child class of the Actor.

- Buyable and sellable interfaces are introduced. Having two separate interfaces for this feature follows ISP principle as some items may be sellable but they are not buyable. If the two interfaces are combined into one, then we might need to provide a dummy implementation if we encounter the case we mentioned above where some items may only support one function but not two.

- Sellable item will have a base selling price as an attribute in its class. This is because the price of every item sold by the player to the merchant is fixed. However, the price of the items sold by the merchant may be different. Hence, the constructor of the buyAction will accept an extra argument, price, which allows the merchant to declare their own price for the items they sold.

- BuyAction class is referenced to a Buyable interface and SellAction class is referenced to a Sellable interface. This allows the BuyAction and SellAction class does not tightly coupled to a particular item. This follows OCP and DIP principle as if there is a new item which is buyable added into the game in the future, we simply just need to implement Buyable interface to the item and it can then be bought from the player without needing to modify any of our existing class. Having a BuyAction and SellAction class separately also follows the SRP principle as each of the class manage only one responsibility of their own.

Pros:

- This design follows OCP principle as if there is a new item or actor added, we do not need to modify the existing code but simply just extends from it.

Cons:

- Traveler class implements as a subclass of Actor has a few redundant features, for example, the health and stamina (they cannot attack or being attacked), the playturn method (basically just returning a new DoNothingAction), the wallet and iteminventory (they have infinite supply of item and cash)
- Since every traveler has different items to sold and the item is sold in different price, the allowableAction for every Traveler has to be manually coded.

Future extensions support:

- If a merchant type actor is added to the game and they can move around, simply override the playturn method and add the wander behaviour to the concrete subclass of the merchant class. In our design, we are assuming most, if not all, of the merchant type cannot move around.
- If a new buyable or sellable item is added into the game, simply implements the respective interface to the item.
- For a new merchant type actor, specified the item that is sold by the merchant and the respective price in the allowableAction method.