Design Rationale REQ_3

Design Rationale

Implementation & Why:

Monologue: Interface

- Interface Segregation Principle is adhered to as the monologue interface class is only implemented by actors that has monologue.
- Contains a monologue() void method that stores the initial monologue not affected by conditions
- Contains setMonologueList() that ensures that any class that extends Monologue interface must return an array list of Strings that represent the conversations.
- MonologueConditions() takes care of all the monologue that has conditions, where the conditions are defined and checked if the monologue should be added.

monologueOptions: Arraylist<String>

- An arraylist in each of the actors that stores all the monologues in an arraylist.

hasItem: Class

- A utilities class that takes an actor and the item to find in the parameter.
- A method actorHasItem() that iterates through the inventory of the actor and returns a true value if the instance of item searched is found. False otherwise.
- Open closed principle is adhered to as in the event where we want to find multiple items, instead of having to loop through the inventory and have multiple if statements checking for the items, just call the class.
- DRY principle (Do not repeat yourself) as the actorHasItem() logic is not repeated in all the actors that is required to find the item.

monologueAction(): Action class

- An action class that takes in the actor, arraylist of monologueActions.
- When the action is executed, it randomly selects a random index within the length of the arraylist of monologueOptions using the built in get() function. The selected monologue is returned.

Blacksmith:

- Blacksmith takes in the abxervyer object during instantiation to be used in the condition checking.
- Implements the Monologue interface (Interface Segregation Principle)
- AllowableActions calls the monologue() method that adds the initial monologue not affected by conditions to the monologueOptions arraylist.
- The rest of the monologue is added after the conditions are checked.

Instead of creating a class to store the monologues of the actor, we chose the approach to do the monologue as an interface to reduce coupling. It is justified as instead of creating a class for each of the actors that has a monologue which might result in many classes in the future where other actors with monologues are added, an interface allows the actor class themselves to handle the conditions and the state of the list.

Pros:

HasItem utility class follows the single responsibility principle as the class exhaustively takes care of checking if an actor has the item in their inventory. The class method actorHasItem can be used by all actors in the game to find any Item by utilising the getClass() method that gets the root class of the item.

Our implementation of adding and removing from the monologueOptions arraylist makes sure that all the monologue String(s) in the arrayList always fulfils the conditions and can be printed out.

Having the monologueOptions ensures that the arrayList to store all the monologues is initiated in the actors as an arrayList.

Cons:

The checking for the monologue conditions as well as adding/removing from the list may seem repetitive at times.

Extendability:

The use of the Monologue interface allows future actors that would have the option to have monologue interactions to just implement the interface.

MonologueConditions handles all the conditions and adding/removing of the condition based monologues that allows for high level customization.