REQ4 – Design Rationale – Assignment 1 – Dean Mascitti- 33110360

Design Goals

The aim of this requirement is to implement a metal pipe functionality that the player can pick up put in their inventory and then use to attack hostile creatures, in addition the intern should be able to punch enemies as well. The other part is to implement functionality for the intern to be able to consume the fruits from req2 and regain health through consumption.

The design goals of this requirement include ensuring that future extensibility is though through and accounted for, as well as implementing classes in which are easily extensible and adhere to good OOP practice, such as not repeating code, making code easily added on without refactoring and use of abstraction where appropriate.

Design & What OOP Principles Were Applied

The metal pipe class was added and inherited the WeaponItem abstract class from the engine code provided, the WeaponItem class has much expected functionality of items used to attack other actors. Also added a SpecialScrap class which gives special scraps the ability to return the list of their special abilities and MetalPipe inherits this class.

In terms of creating heal options for intern with fruits a HealAction class as created which allows the player to have the option of consuming the fruit and healing themselves, and then executing this action. This action is added to the allowableActions method for the Consumable class as we are under the assumption all consumables can be consumed and potentially heal the player if in their inventory. Also added a SpecialScrap class which gives special scraps the ability to return the list of their special abilities and Consumable inherits this class.

The ability of the player to also attack hostile creatures with their bare hands was added as the player's intrinsic weapon, as it is assumed intrinsic weapon refers to the actors fundamental/given weapon/way of attack.

The design OOP principles used in this requirement include: open-closed, DRY, extensibility, abstraction and single responsibility.

How They Were Applied & Why in This Way

- The reason that the Metal pipe class inherited the WeaponItem abstract class is because this class from the engine already had a lot of attributes and functionalities that the metal pipe required, therefore using ready made classes rather than making my own very similar class was the better option under the DRY principles. This also adheres to the abstraction and open-closed principles as through the use of WeaponItem abstraction and the adding of the MetalPipe class functionality without changing any existing code.
- The reason why a HealAction was made is because it is a relatively generic action in that it does not refer to a certain type of healing or doesn't restrict any classes from using it, therefore promoting extensibility for other actors to be healed by a consumable, therefore if this is the case where other actors can in the future consume something and be healed this

action is ready made and would not need any other implementation or repetition of code, therefore abiding by DRY and open-closed principles. Also taking the responsibility away from actors and consumables in doing the heal action adhering to the single responsibility principle.

Design Pros and Cons

Pros

- Largely extensible for more healing applications and special items that can be used as a weapon
- Polymorphism capabilities
- Adding new features requires little refactoring original code
- Effective use of provided engine code

Cons

- Special scraps are somewhat not related functionality wise (they can have different functionalities)
- Double abstraction – can possibly lead to overriding method issues and lack data leaks

Alternative Designs

An obvious alternative would have been to make a functionality connection between all future and current special scraps; however, I found many issues with the prospect of this. Firstly, weapon scraps already have an abstract class in the engine code being WeaponItem, so for other special scraps like ones who can heal it does not make much sense to create an abstraction between the two because the only difference they hold is there allowable actions, and this method is already abstracted in the Item class. Therefore, instead I chose to make an interface for all special scraps named SpecialScrap, so that they can be grouped together and used in upcoming assignments to be traded and sold.