Assignment 1 Design Rationale: CL_AppliedSession10_Group1

Requirement 4 – Terminal, purchase logic and new items.

This design document considers implementation of a new terminal. This terminal allows users to purchase items that are classified as 'purchasable'. The terminal system has a list of purchasable items, appending the purchase action of each item into the players allowable actions list. The terminal's list of purchasable items uses individual purchasable factory methods implementing 'PurchasableFactory'. In doing so, each item is constructed before the terminal calls the set ground method (prior to terminal initialisation) in Application.

An essential feature to code robustness and reusability is ensuring that the codebase mitigates unnecessary replication by reusing identical methods. One way of ensuring that this is counteracted is the use of abstraction. Each purchasable class abstracts item logic from the original game engine. This allows us to consider the DRY principle. Moreover, the creation of a factory pattern for initialisation of purchasable items allows us to further abstract away from concrete dependencies. Lastly the use of a purchasable interface allows us to abstract purchase logic into each item since the goal is that every item can have some unique functionality upon its purchase, throwing errors, changing the amount charged; whatever it may be. Bundles of code are typically abstracted when they change for the same reason, however when code changes for a unique reason it leads to use of the singular responsibility principle.

Each purchasable class extends Item or Weapon Item, depending on what form of functionality is desired. However, the goal of this requirement is that items are unique in their individual purchase logic. Therefore, creation of singular classes necessarily provides the codebase to work upon any form of item to be bought. Therefore this approach considers the singular responsibility principle. Another important feature of codebase is ensuring that classes facilitate extension without modification. This feature is known as the open closed principle (OCP). One area of our code that particularly considers this is in the dependency injection of the terminal class. Our initial design hard-coded the purchasable items as associations; however, this is inconsiderate and assumes all terminals sell the same items and must add statements every time a new item is created. This approach is of low code robustness and poor reusability.

An advantage of our implementation is through utilisation of factory design patterns when implementing purchasable items. That is, it loosens the coupling code by separating terminals purchasable item construction code from each items unique purchase logic. This design implementation is further advantageous through its use of dependency injection. As previously highlighted, utilisation of a list of purchasable(s) separates the initialisation of the terminal from what it Is selling. This way the nature of its purchasable items is no longer an association. In this way it is less concretely implemented and flexible, adaptive to future consideration or changes of code.

An alternative design may consider a more abstract implementation for the execution of the purchase action. This is advantageous as our current implementation assumes that we only purchase an item if the current wallet credits will not exceed the items cost demand. This doesn't consider purchasable items that are of negative cost or other extraneous purchase designs. Another inconsistency of our design may be highlighted for the fact that only one purchasable can be bought per turn.