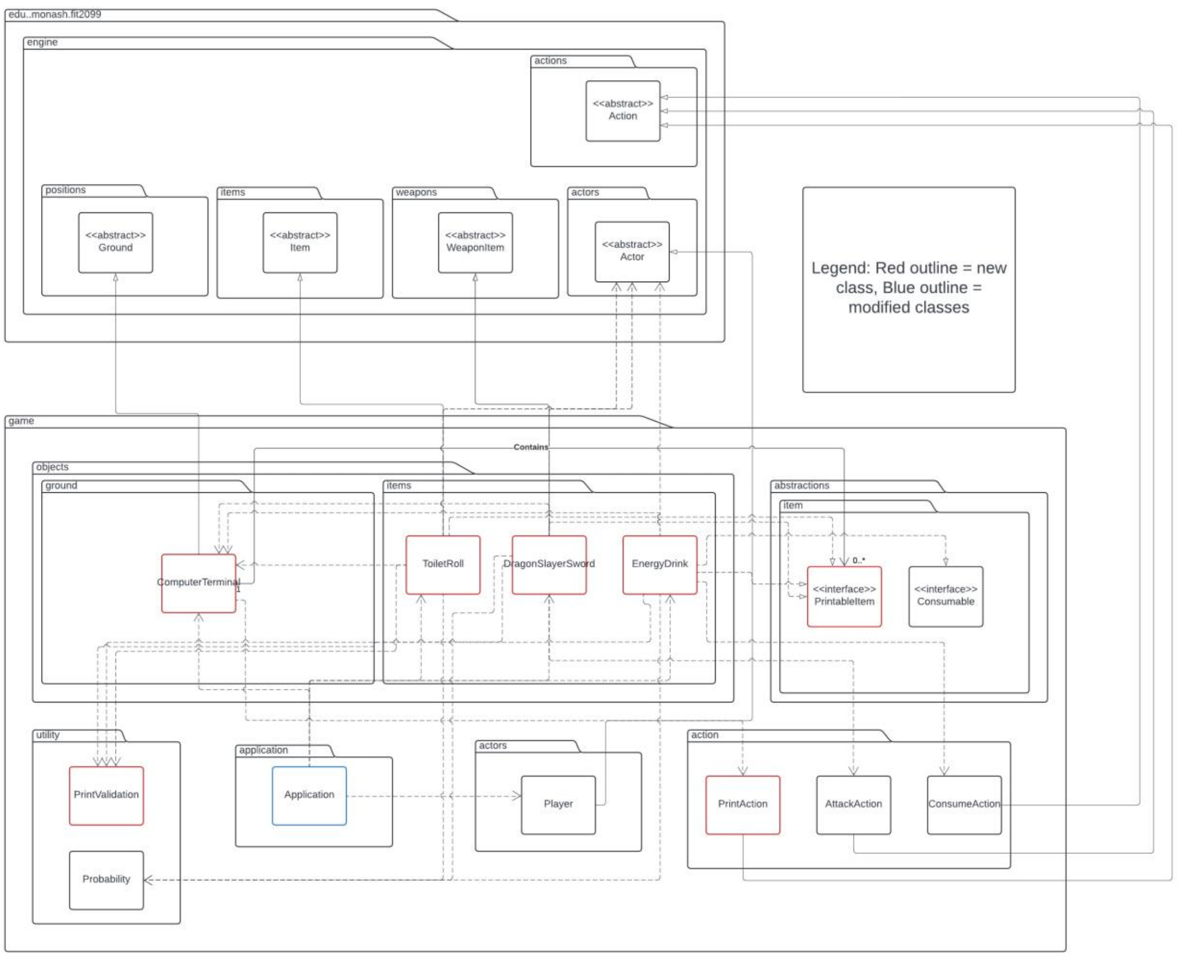
Assignment 2 – Requirement 4 Design Rationale – Group 4

UML Diagram



Design Goals

In this particular requirement the goal implement the computer terminal functionality in addition the aim is to implement various items with different abilities in energy drinks, dragon slayer swords and toilet paper rolls.

In implementing these features, it is an aim of our group to ensure extensibility, appropriate abstraction, decrease the number of dependencies whilst ensuring that each class has their own single responsibility.

Design & What OOP Design Principles Were Applied

In terms of the implementation of the computer terminal class, it was built to inherit from the Ground class as we assume it is a stationary object. This class, when created gets passed in a list of the items it is able to print. When the player is within close proximity to it will be given the option to print each item from the terminal, this was implemented through the creation of the PrintAction class which is responsible for the transferring of the item into the player's inventory. In addition, we made the assumption that computer terminal would be the only ground type to be able to print items in this game, therefore we did not make an abstract class for all grounds that can print to inherit.

A PrintableItem interface was created in which all items that have the ability to be printed inherit. The method in this interface is where the respective item can implement its unique printing process. Therefore, the classes EnergyDrink, DragonSlayerSword and ToiletRoll all implement PrintableItem and override this method with the implementation for their respective printing executions.

The EnergyDrink implements the Consumable interface class as it can be consumed by an actor, it therefore returns the ConsumeAction in the allowableActions method so that the player has the option to drink the energy drink for health points. The DragonSlayerSword inherited from the WeaponItem class as it can be used to attack other actors, it therefore returns the AttackAction in its allowableActions method. The ToiletRoll class inherits from the Item class as it has no other functionality or abilities than simply being picked up and dropped.

In the design the following principles were abided by: DRY, open-closed, abstraction and dependency inversion.

How They Were Applied & Why in this Way

- Do not repeat yourself (DRY) principle was implemented in our design by grouping the similar code of checking if player has sufficient funds, and then transferring to their inventory all into a static method in the PrintValidation method, therefore this made us not have to repeat this same code in each PrintableItem's class.
- Open-Closed Principle was used in this design through the use of abstract and interface classes. For example, the creation of the PrintableItem class ensures that in the PrintAction and ComputerTerminal classes we can simply denote the type of the item to be printed as a PrintableItem, therefore if there are more items that can be printed in the future current implementation doesn't need to be customized. The use of Item, and WeaponItem to be inherited by ToiletRoll, DragonSlayerSword and EnergyDrink, shows the open-closed principle as when these classes were added we did not need to customize any of the existing code.
- The Dependency Inversion principle was abided by in this requirement as the creation of the PrintableItem we can simply in the PrintAction and ComputerTerminal classes define the type of their attributes at PrintableItem's beside having to create separate attributes for each item that can be printed, therefore there is only a dependency on the PrintableItem class, therefore reducing dependency's.

Design Pros and Cons

Pros

- Extensible – easy to add future items that can be printed in the future
- Each class has a single responsibility
- Deduction of dependencies where possible
- Use of abstraction
- Lack of repeated code

Cons

- No abstraction for computer terminal which may be problematic if other ground types in the future can also print items
- Execution of printing Is technically done in each printable item as each printing execution is unique

Alternative Designs

An alternative design that our team thought of weas to combine the spawning implementation from Assignment 1 with printing. Our thought process with this idea was to use the existing code we had in order to reduce complexity and make use of abstraction. However, we chose not to follow through with this because we realized printing and spawning are inherently different. For example, in spawning the implementation is based off the idea that spawning is done randomly and autonomously, whereas printing is prompted by the player, therefore making this approach unfeasible.