A3 Design rationale (A2 updated)

1. Travelling Between Maps

A. New grounds

Add 2 new classes `Cliff` and `GoldenFogDoor` that extend Ground are added into the game to make the map more interactive.The `Cliff` is a special ground type that causes the player to instantly die and lose runes on the last location. The mechanism belongs to reset, in order to reduce the repetition of code and improve the maintainability, the death determination could be done with the `ResetManager` class. The rune dropped by death could also be done with the previous death setting, which helps to reduce repetition (DRY).

The `GoldenFogDoor` allows the player to travel around different maps. The door connects Roundtable Hold (safe place, 1 door) – Limgrave (2 doors) – Stromveil Castle (2 doors) – Boss room (no door). They both implement the abstract methods within the `factory` class to allow the player to travel to other parts of Limgrave. Before two parts of map class with ` EastEnemyFactory` and `WestEnemyFactory` will belong to new class `Limgrave`. It extends to interface `EnemyFactory`. By just re-adjusted the existing map, it allows for code reuse and specialization, also can create a new class that inherits from a base map class, easily add or modify functionality as needed.

2. Inhabitant of the Stormveil Castle

A. Grounds

Two new grounds are added in this requirement which are Cage '<', and Barrack 'B'. As these are part of the Stormveil Castle map, a class `StormveilFactory` is created, extending the `EnemyFactory` interface. Similar to 'EastEnemyFactory` and `WestEnemyFactory` made for Req 5 of Assignment 2, it implements the abstract methods within the factory class and sets spawns for the newly added enemies.

This requirement can be achieved without the new factory class, as it can be done with the old factories but this new `StormveilFactory` is created for better readability and extensibility. It violates OCP as the spawns are set as methods but as for a trade off, new enemies for the Stormveil Castle and different sides such as northeast, northwest, etc., can be added to the map with ease.

B. Enemies

Two new enemies, `Dog` and `GodrickSoldier` are also added in this requirement. Similar to Assignment 2, the two new classes extend `Enemy` abstract base class, following DRY principle. As the behaviours of these enemies are the same as the previous enemies, no change needs to be made in the base class. And to stop `Dog`

and `GodrickSoldier` from attacking each other, an additional capability `EnemyType.CASTLE` is added. Just as before, the `AttackBehaviour` checks if they are of the same enemy type before initiating attack.

As for the weapon for `GodrickSoldier`, a new weapon created for Req5 is used.

3. Godrick the Grafted

Golden Runes as an new item, that cannot be purchased. It can be found scattered across the maps. And it will implement by `location` class to picked up or dropped off by the player.

As the the Axe of Godrick and Grafted Dragon are new weapon item, two new child classes of `WeaponItem` is created – `Axe of Godrick` and `Grafted Dragon`.

And these two items can not buy using Runes, and the `Rune` objects implementation of the interface's deduct function will not call `RuneManager` to try to deduct the money. But these tow weapons still can sellable to the new trader `FingerReaderEnia` , and still create SellTradeAction. So in A2, we set use DIP allows trading to be more flexible in terms of what can be traded. ISP is followed so that an
item only needs to implement whether it is buyable or sellable, and not both, if they cannot be exchanged both ways. Using an interface also enables these items to be given or taken from the actor differently depending on the item.

Also, we also would like to allow the player to exchange Axe of Godrick and Grafted Dragon to select a trade action. At first, I set a new action `ExchangeTradeAction` created that they take an item to be exchanged, then calls the required methods to execute the trade. Having the `changeable` items implement the `changeable` interface. But following, I choose to take an instance of `RemembranceOfTheGrafted` as a parameter, then the chosen weapon is added to the trader's inventory as a sellable item by calling the addSellableItem method, which suggests that the trader has a sellable item. It clearly states that the trade involves exchanging the specified item for a weapon. If there are future changes or additions to the trade system, such as introducing new items for exchange or modifying existing ones, the method can handle these changes by accepting the appropriate item as a parameter. This removes the repetition of already existing code and thus follows DRY principle.

5. Creative Requirement

For the creative requirement, a new weapon `RiversOfBlood` is added with a special skill `LifeSteal`.

A. RiversOfBlood

Class `RiversOfBlood` extends base class `WeaponItem` as it has similar characteristics with other weapons, therefore following LSP and implements existing `Sellable` interface so that this weapon can be sold to Merchant Kale. This eliminates the need to repeat code, and thus follows DRY principle.

B. LifeSteal

This new weapon added has a special skill which heals the weapon holder a fraction of the damage dealt. For this, a new class `LifeStealAttackAction` extending `Action` base class is created. And to increase the weapon damage when held by the player, `Status.HOSTILE_TO_ENEMY` is used to check the holder and the damage is increased making use of the `ModifiedWeaponItem` created in Assignment 2. This removes the repetition of already existing code and thus follows DRY principle.