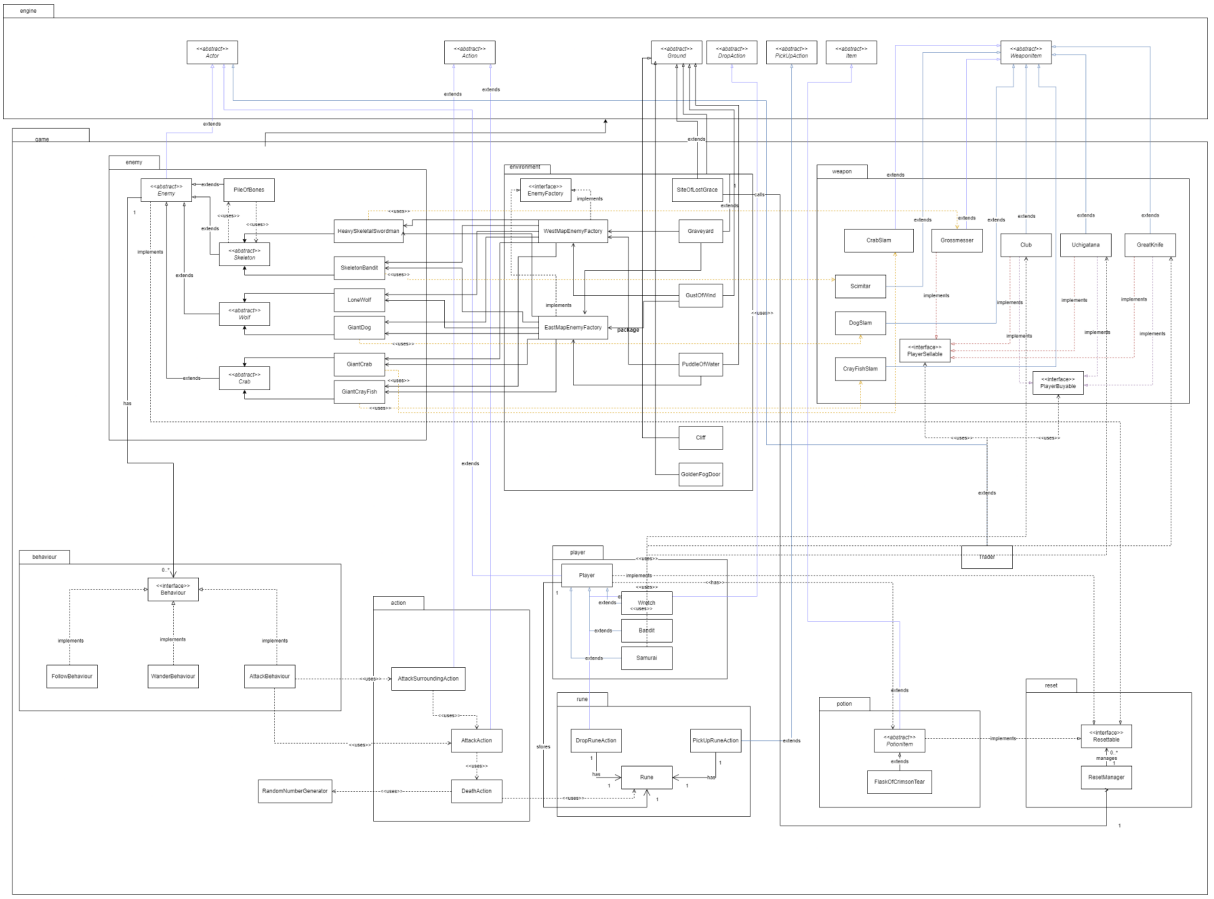
FIT 2099 Group 3
Assignment 1

Group Member:

Lee Sing Yuan (32203632)
Loo Li Shen (32619685)
Yeoh Ming Wei (32205449)

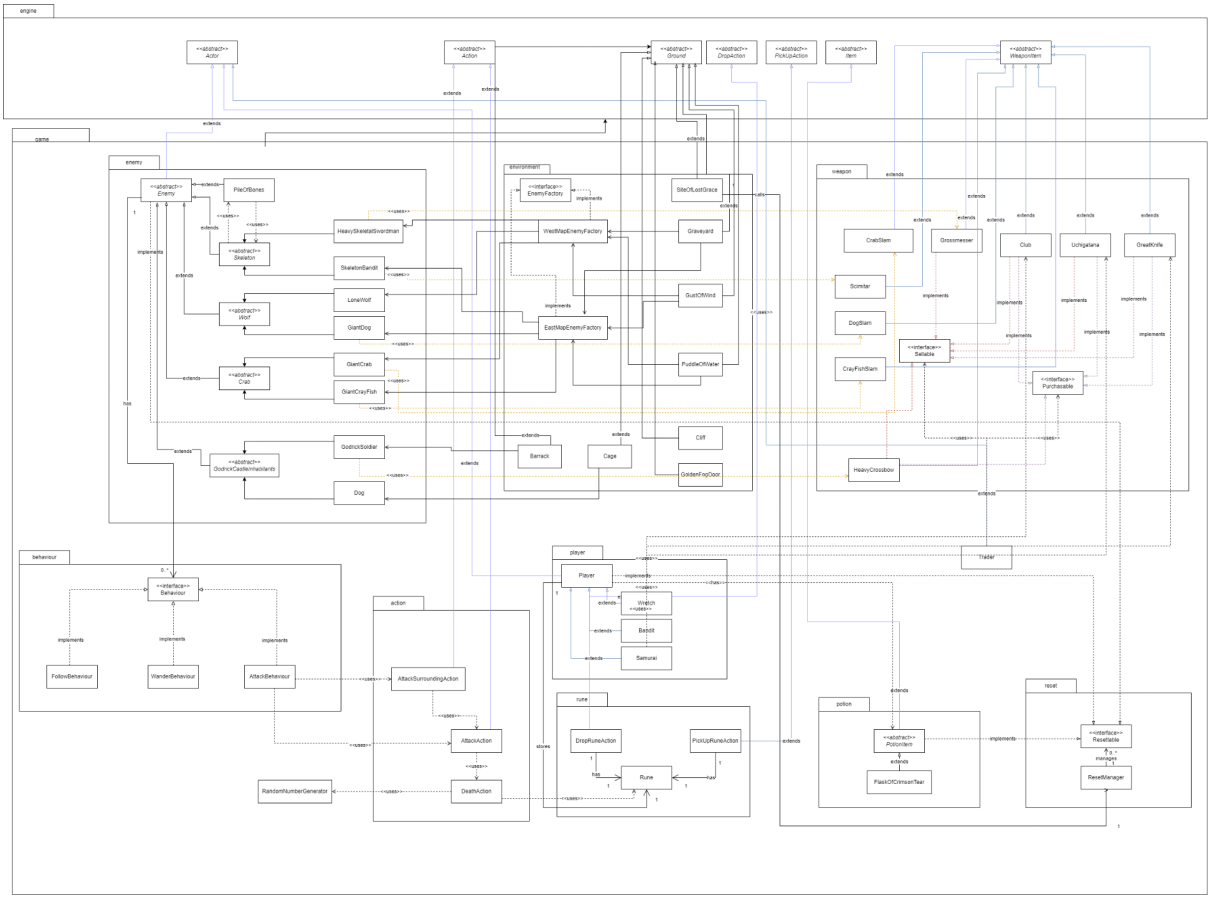
Req 1: UML

If image is too small, please see the images in GitLab



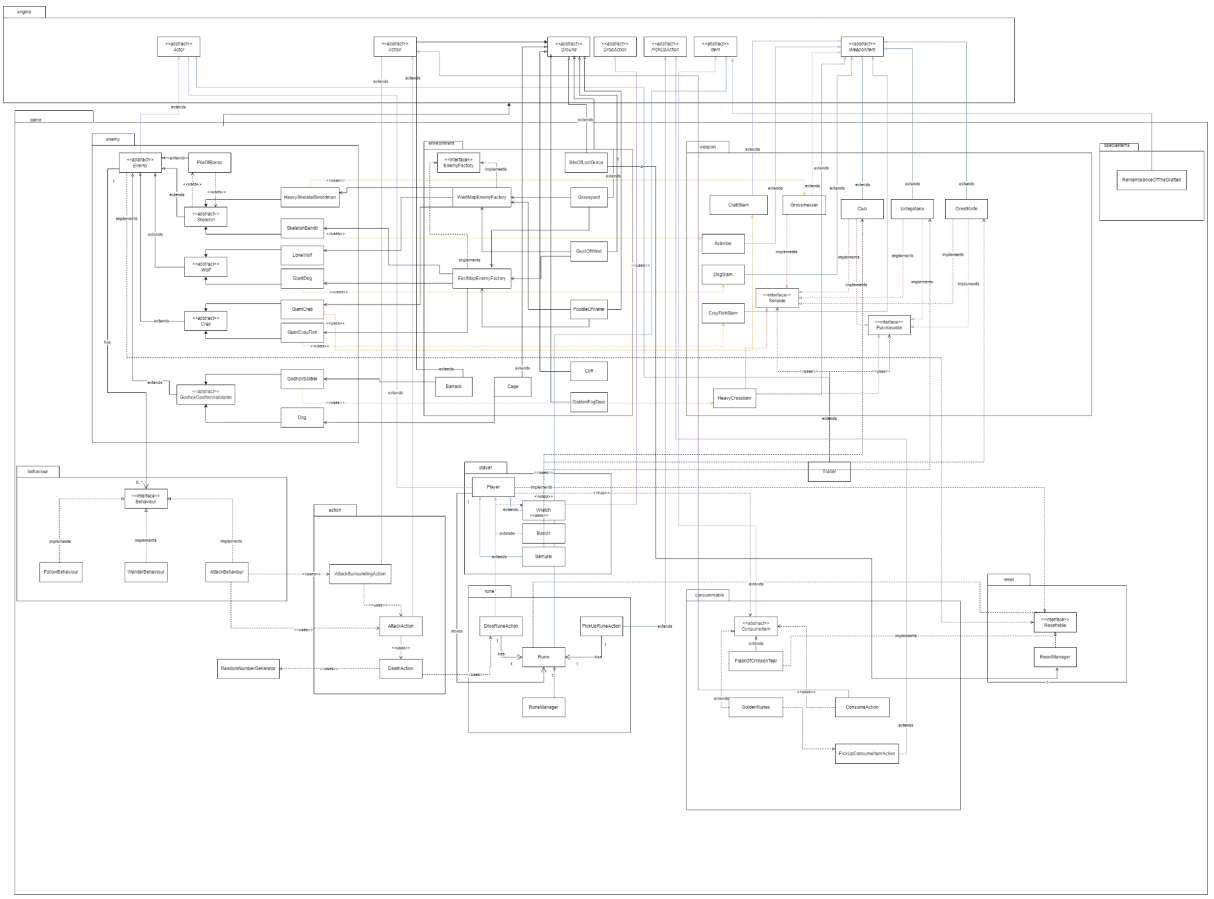
Req 2: UML

If image is too small, please see the images in GitLab



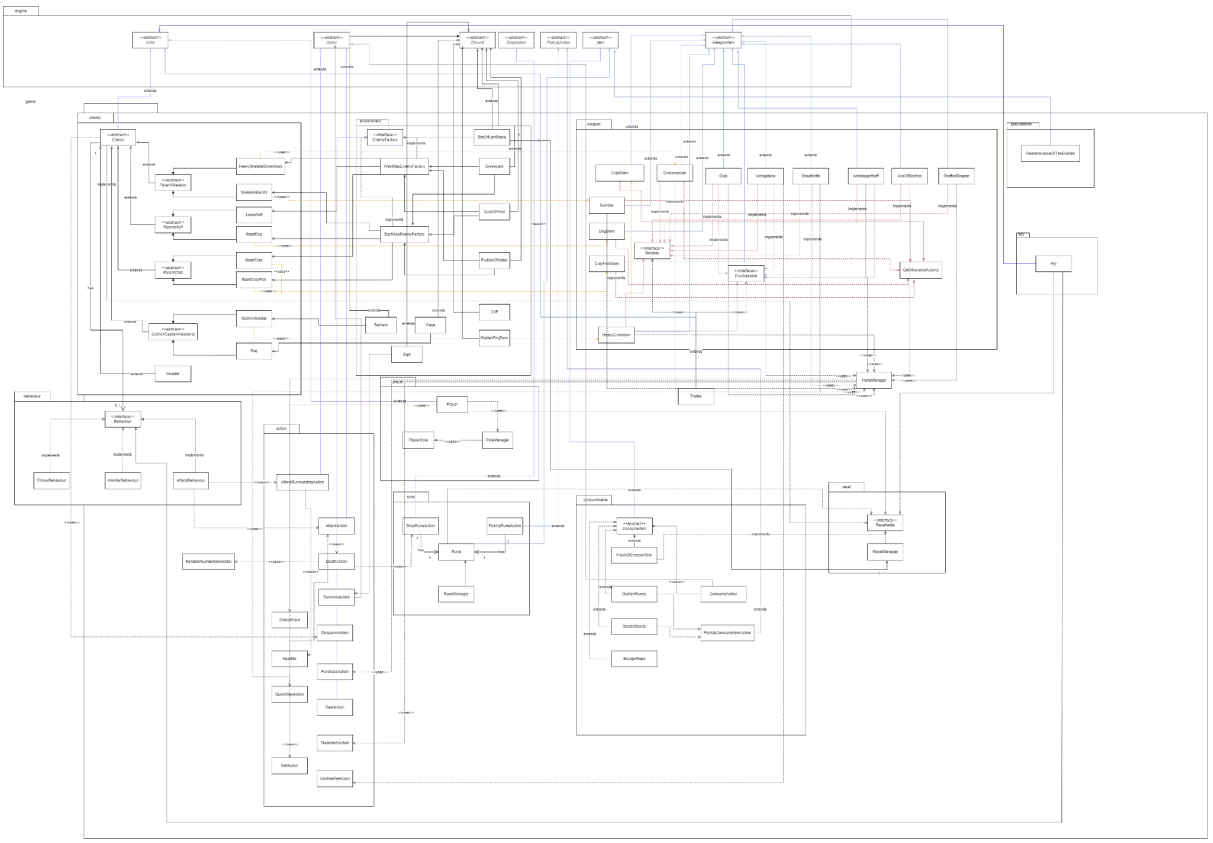
Req 3: UML

If image is too small, please see the images in GitLab



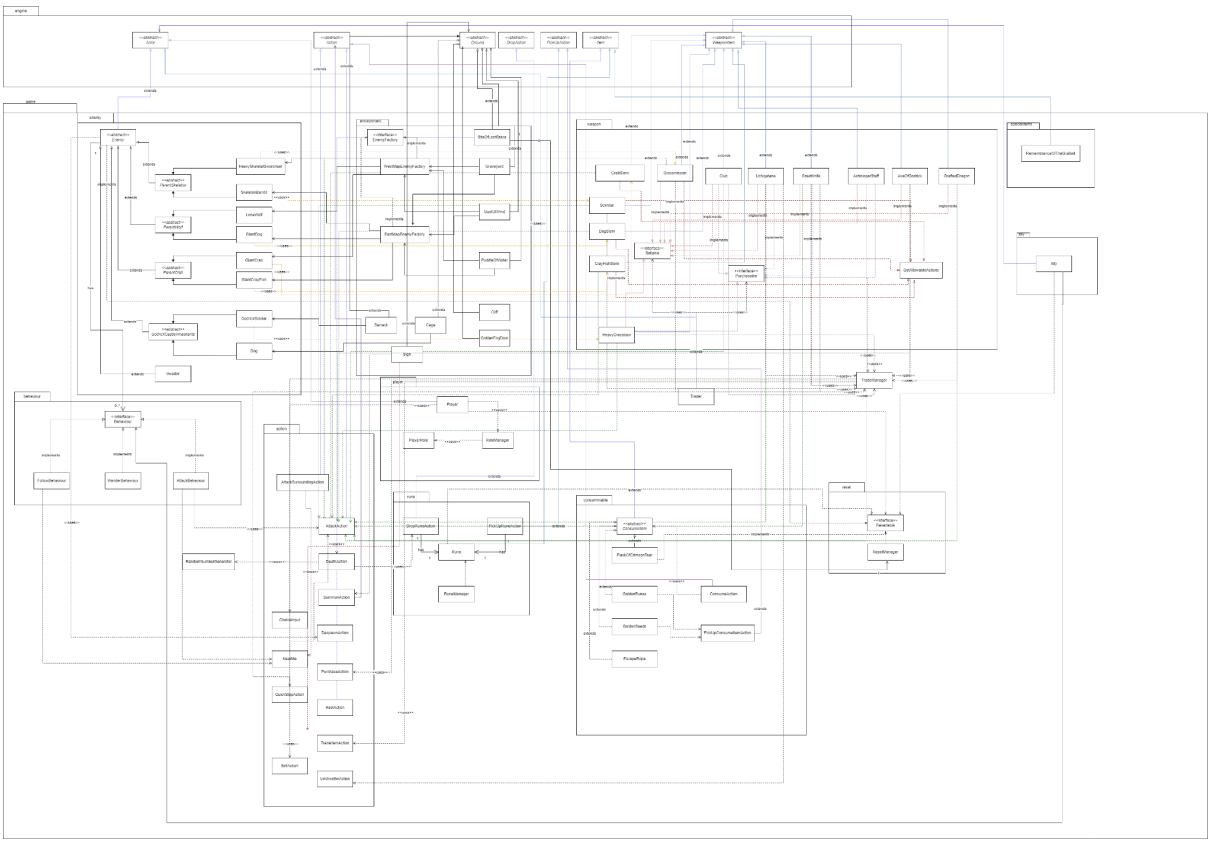
Req 4: UML

If image is too small, please see the images in GitLab



Req 5: UML

If image is too small, please see the images in GitLab



Req 1 rationale:

New Grounds

Cliff and GoldenFogDoor inherit from Ground so in future implementations, if there are further potions implementations, it will be inherited from Ground as well. This follows the Open Closed Principle.

New Maps

Made different static variables that would be assigned to the different maps This is useful because you need a global reference to the game map that can be accessed from other parts of the code.

Req 2 rationale:

Enemies

Made abstract class for all castle inhabitants and allowed Dog and Godrick Soldier to inherit. Made it this way, because they have the same type(as in cannot attack each other). Benefit, adheres to open close principle because if we were to add a new castle inhabitant, we just inherit and add more capabilities if they have. There was no alternative

Weapon

Made the weapon extend weapon item similar to all other weapons. Made all attack actions possible with this weapon in the class itself. Adheres to open close principle because if we were to add more weapons, we just have to add any actions that is possible in the class itself. The alternative to this is to make the actions in the enemy class which violates open close cause if there was a new attack which has a range of 10, we need to add the weapon and add the actions in the Enemy class

Ground

To further extend the functionality of the game, we had also added some new grounds that allowed new enemies to spawn at it. The new grounds are called Cage and Barrack that allows dogs to spawn at the castle map. Therefore, we had created new classes for each of the grounds and extended the ground abstract class. This applies the open close principle so that we only make changes at the Cage and Barrack class and do not need to touch the Ground abstract class in the engine package. As for the implementation on how the enemy spawn, I proceed to spawn the enemies using the Cage and Barrack class instead of the East and West factory map to avoid repetitive code and it is unnecessary to separate it as it violates the principle “Do Not Repeat Yourself”.

Req 3 rationale:

New trader

Made the new trader in the trader class. Applies open close principle because if there are new traders, just add them to the trader class. Alternative to this, was to make another trader abstract parent and make child class of each trader. Issues is that it will have multiple levels of inheritance and may be hard to debug.

Rune of remembrance

Made all the traders it could trade with and the things it could be traded for inside the class itself. This adheres to open close principle because it can add all the information within the class itself. It does not need to modify other classes such as trader. The alternative to this method was to add the information onto the trader which violates open close because would need to modify the trader class

Weapons

Made to provide all the “possible actions” within the class itself. This adheres to open close principle. Alternative to this was to make the enemies to check the type and return the allowable actions there

Golden Rune, ConsumeAction and ConsumeItem

As a golden rune is made to use the item, the action is actually similar to consuming a potion. So we made it so that Golden Rune can be executed in ConsumeAction.

The ConsumeAction class has a single responsibility, which is to perform an action where an actor can use or consume the item. It accepts a ConsumeItem object as a parameter, allowing different types of consumable items to be consumed. As for the golden rune, it has a functionality to use the item and add a random amount of rune to the player’s rune.

For the ConsumeItem abstract class, it also encapsulates the logic related to consuming the item and updating the uses left. The ConsumeItem abstract class is open for extension as new types of consumable items can be added without modifying the class itself. Since the golden rune is almost similar to consuming an item, the golden rune can extend the ConsumeItem class.

Req 4 rationale:

Astrologer

Stores all the information in another role manager class and will set the player's weapons,items and capabilities if user selected this role. This is done to adhere to open close principle as if we want to add a new role, just add them to the role manager. The alternative to this was to use a list which could be accessed by anyone to store instance of the roles, eg: Samurai instance. Issues with this is that too much dependency, violates open close principle and will cause referencing issue to ally and invaders.

Astrologer staff

Made the same way as all other weapons to adhere to open close principle by making the weapon itself or the class itself to get all the allowable actions that could be done with the weapon or class. Alternative to this was to make the Enemy class get all the possible actions that could be done with this weapon. Issue is that it violates open close principle as if the enemies were allowed to use the staff for example, need to change the behaviour class

Allies and Invaders

Uses the role manager similar to player. Adheres to open close principle because any new modification can be done on the class itself, does not need to change any other classes. For example, if all allies have a new behaviour, just add it to the class itself. There was no other alternative.

Summon Sign

A summon sign is a ground that allows the player to summon an ally or invader. So, we decided to extend the ground abstract class to avoid violating the open close principle. Then, we also created a summon action and used it in the Summon Sign class to have an allowable action when the player is near the Summon Sign. It used to have a Single Responsibility Principle so that the action has only a single use which is to summon an ally and invader. .

Req 5 rationale:

Extra consumables

Made two new different consumables:

Golden Seeds
The GoldenSeeds class has a single responsibility, which is to represent the consumable item "Golden Seeds" in the game. It encapsulates the logic related to using the item to increase the capacity of FlaskOfCrimsonTears and adding golden seeds to random locations on the game map. The GoldenSeeds class is open for extension as it extends the ConsumeItem class. It overrides the use method to implement the specific behaviour of using golden seeds to increase the capacity of FlaskOfCrimsonTears . It also overrides the getAllowableActions method to provide a ConsumeAction that allows players to consume golden seeds and increase the capacity of FlaskOfCrimsonTears.

EscapeRope
The EscapeRope class has a single responsibility, which is to represent the consumable item "Escape Rope" in the game. It encapsulates the logic related to teleport the player and updating the location. The EscapeRopeclass is open for extension as it extends the ConsumeItem class. It overrides the use method to implement the specific behaviour of teleporting the player using the GoldenFogDoor class. It also overrides the getAllowableActions method to provide a ConsumeAction that allows players to consume the escape rope and teleport.