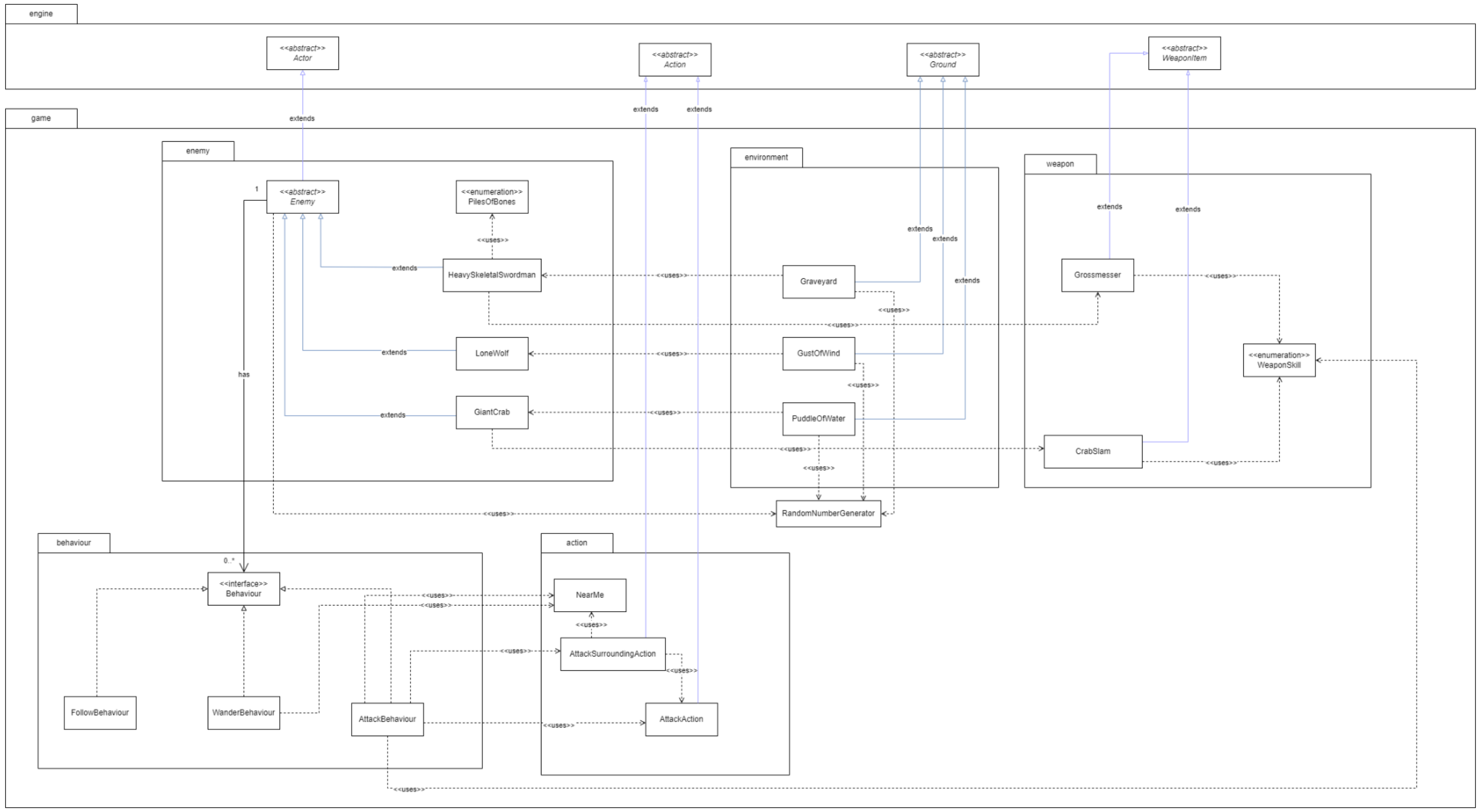
FIT 2099 Group 3
Assignment 1

Group Member:

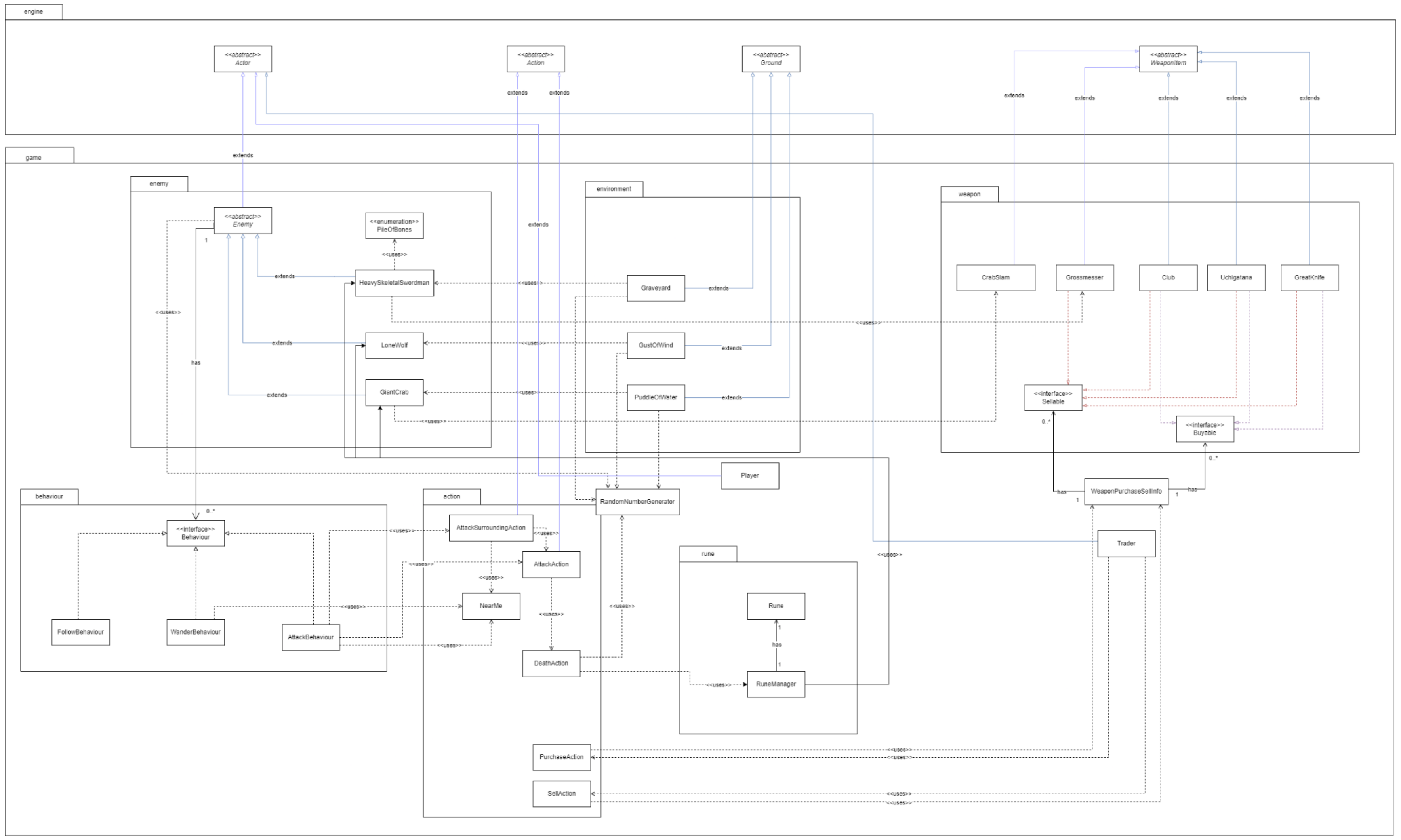
Lee Sing Yuan (32203632)
Loo Li Shen (32619685)
Yeoh Ming Wei (32205449)

REQ 1:UML



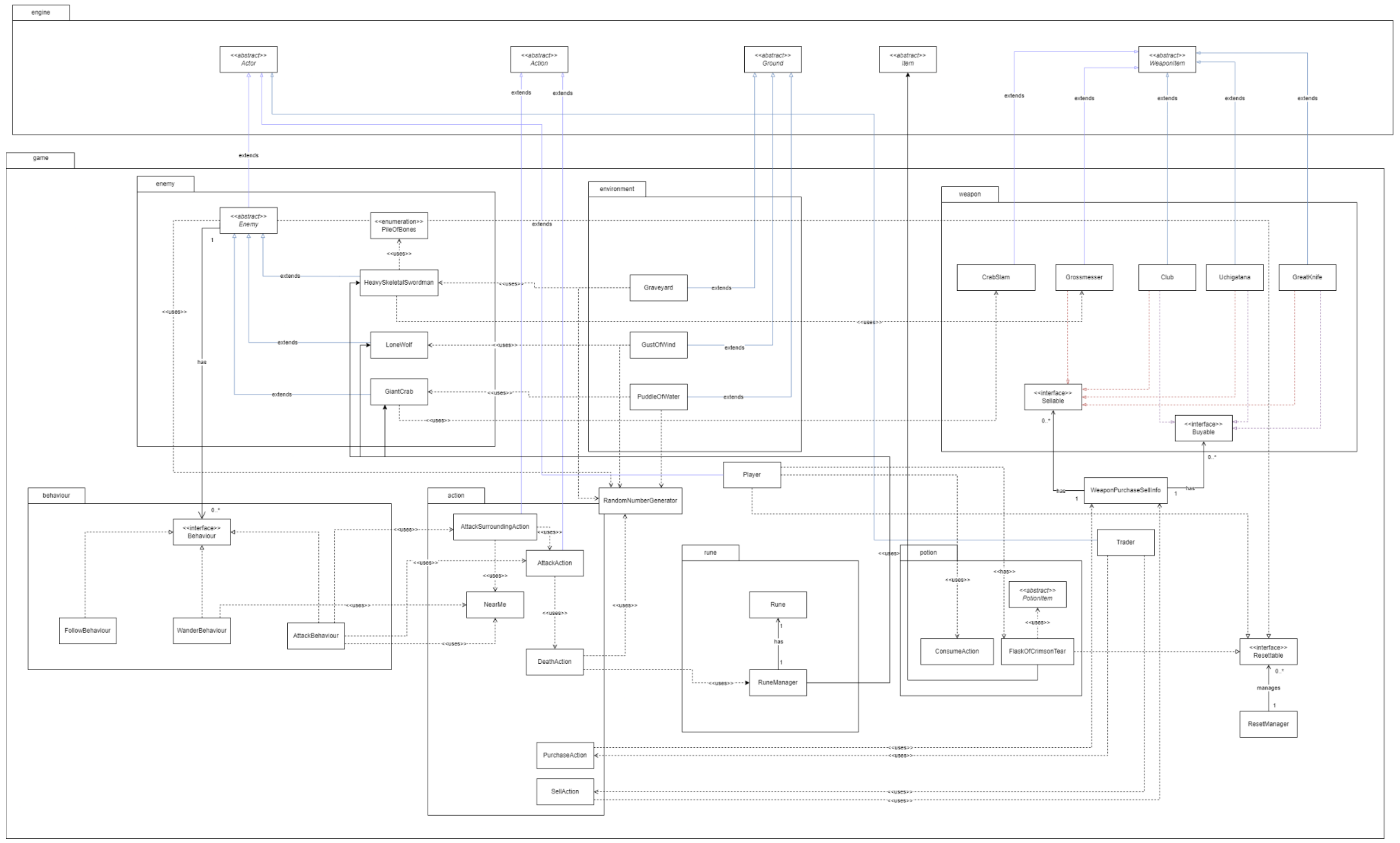
** Please refer to our asgn2 folder if the image is too blurry.

REQ 2: UML



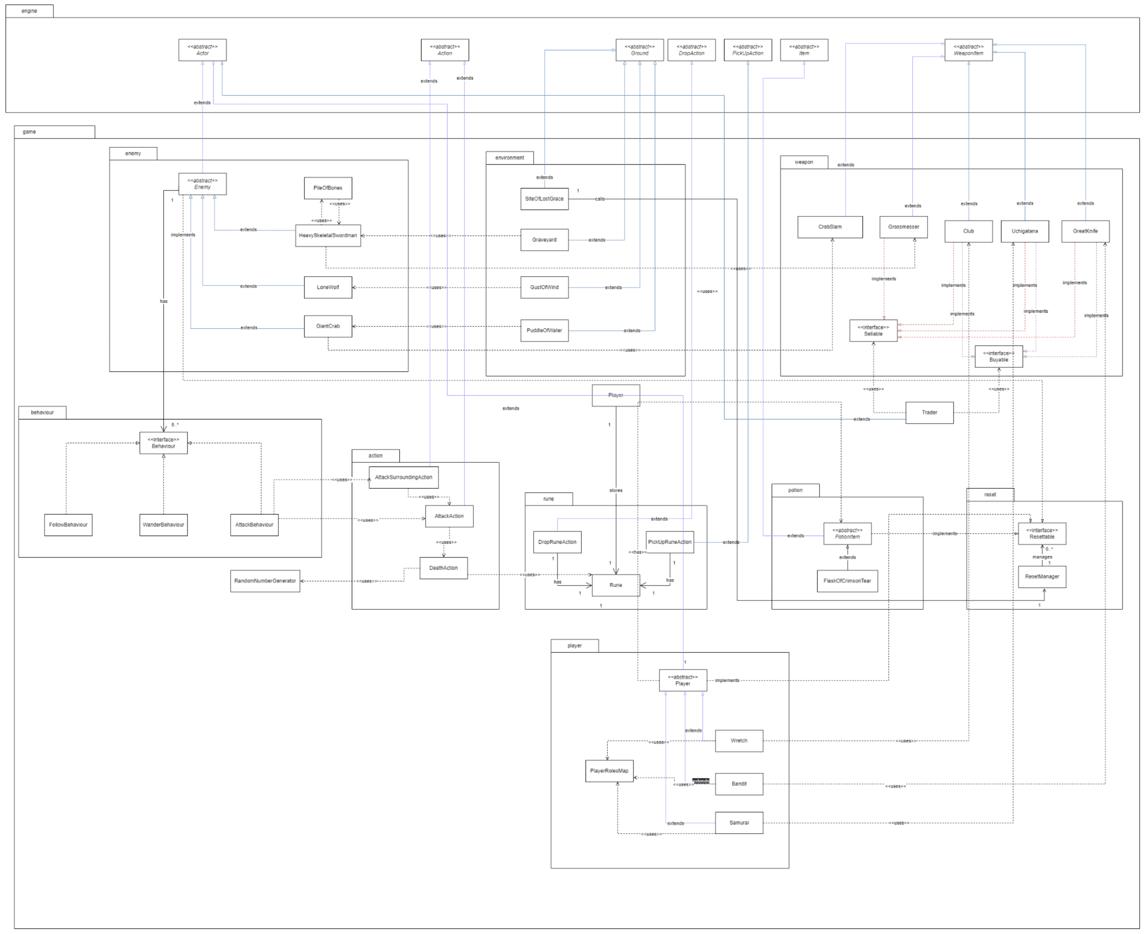
** Please refer to our asgn2 folder if the image is too blurry.

REQ 3: UML



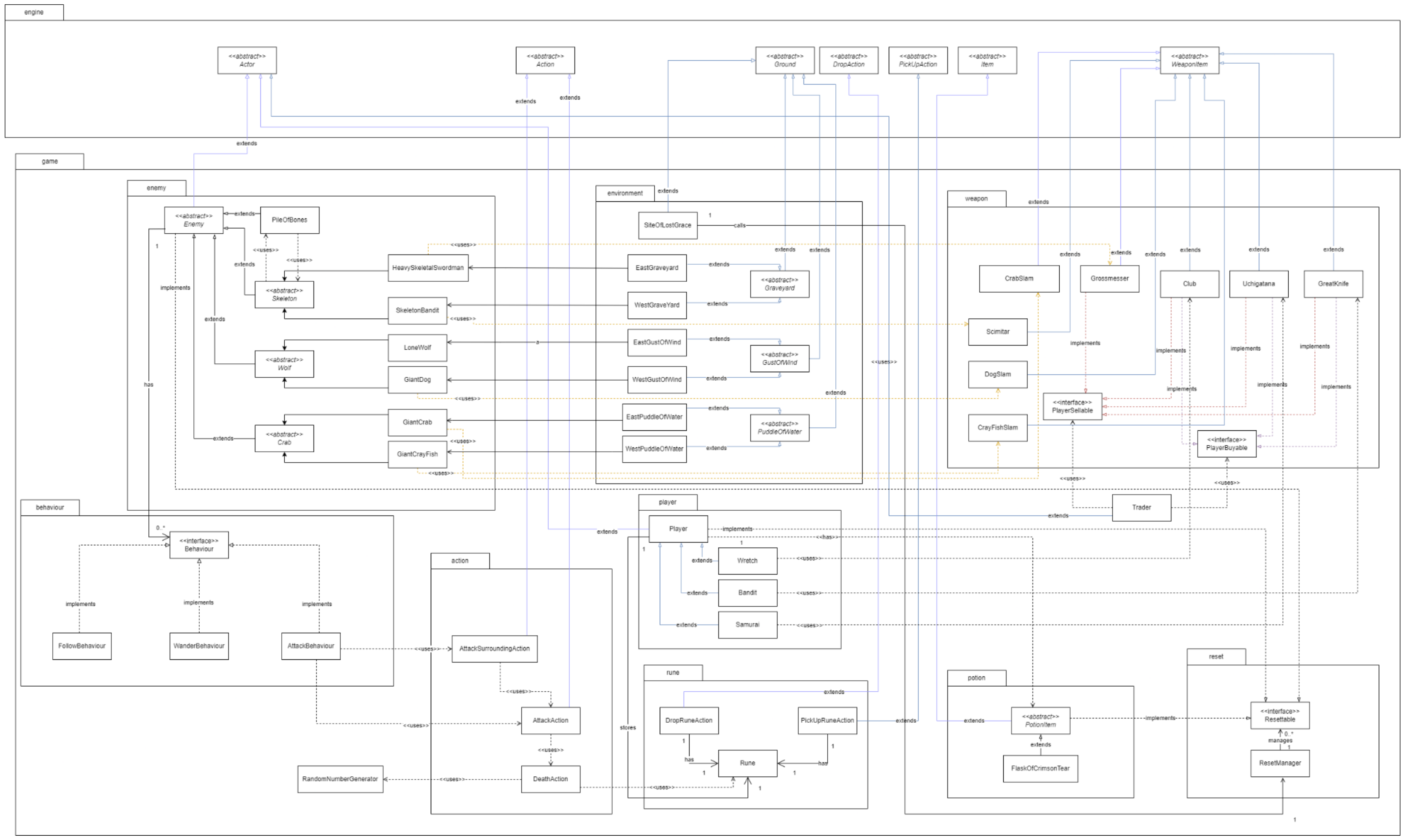
** Please refer to our asgn2 folder if the image is too blurry.

REQ 4: UML



** Please refer to our asgn2 folder if the image is too blurry.

REQ 5: UML



** Please refer to our asgn2 folder if the image is too blurry.

Rationale

REQ1:

Behaviours

The behaviours and the actions obtainable from the behaviours are added at the Enemy abstract class because all enemies will have these behaviours, this will reduce the amount of code repetition(DRY). The pros of this method is that all enemies will have the same behaviour and if there is a new type of behaviour, just add it to the Enemy abstract class. The con is that, all the enemies must have the behaviours. Means that there cannot be a new enemy that will not have Wander Behaviour

Actions

The goal of the Action classes is to have single responsibility. For example: AttackAction is to change the state of the target from unharmed to harmed. The pros of having actions handle a single responsibility is that they act as a building block. For example, AttackSurroundingAction which loops through all the targets and creates new instances of AttackAction. The cons is that because the actions are built on other actions, it may be hard to debug if the logic is incorrect.

Environment

The goal of the environment is to decide whether to spawn an enemy or not. It fulfils the single responsibility principle. The pros of doing this is that in the future, we can build more complicated environments using the already existing environments. The con is that, may need to add conditions if there is new types of enemies spawning for the same environment.

Enemies

Made to inherit an abstract parent class to apply DRY. The benefit of doing this is that all enemies can have the same foundation.

Weapon

The weapons with surrounding attack skills or representing surrounding attack skills will add capability. This is to make it adhere to the Open Close Principle. Example: a new weapon with surrounding attack is made, it just needs to add capability. The pro is that new weapons with the surrounding attack skill will not need further modification to the existing code.

REQ2

Enemy runes

The design goal was to avoid typecasting enemies to find their max and min range for runes that can be transferred to player. The pro of this method was that it also fulfils open close principle. The con is that the hashmap maybe changed anywhere within the code which will make it hard to debug. The alternative to this was to type cast. The benefits of the static hashmap outweigh the type cast because if the addition of runes to the hashmap is controlled,

Buying and selling actions for weapons

The design goal of this was to avoid typecasting of weapon Items to their interfaces by using static hashmaps. The pro is that this method is also open close because if a new weapon is added, only need to create the weapon class and add it to the static hashmaps if they are implementing the interfaces. Cons is that, the weapons need to be instantiated once in order to be inside the hashmap so this may violate open close principle. The alternative to this was to typecasting which violates Liskov's principles.

REQ3

Flask Of Crimson Tears

Inherits from Item so in future implementations, if there are further potions implementations, it will be inherited from Item as well. This follows the **Open Closed Principle**. Another thing added in Flask Of Crimson Tears was adding a static hashmap, this is to check whether the item being consumed is of the FlaskOfCrimsonTears class or not.
This check is necessary because the ConsumeAction class is designed to be used with other potion items as well. If it is used with a different potion item, the if block will not be executed, and the code inside it will not affect the player's health or the number of uses left for the item.
The alternative was to use instanceof() to check however it is a downcasting.

Site of Lost Grace

Site of Lost Grace inherits from Ground abstract class. The Site of Lost Grace has only one responsibility which is to manage the player’s progress and determine whether the player has been in the site before or not **SRP**. Site of Lost Grace will then call upon ResetManager which implements the resettable interface.

REQ4

Player archetypes or roles

Uses an abstract player class to adhere to DRY. The pro of this method is that there will be lesser code duplication. The con of this method is that it may be hard to debug cause of many levels of inheritance.

Weapons skills

Made like action classes, adheres to the single responsibility principle. The pro of this method is that the skills can be debugged easier. The con is that if new weapon skills were to be added, need to add conditional checks. The alternative to this method was to use an interface for skills, but this would need type casting to check cause of how everything is managed as weapon item

REQ5

New Enemy types

Made using a new abstract parent to reduce code repetition (DRY). The pro of this method is that enemies that use Special skill like PileOfBones requires them to inherit the skeleton parent.

Environment split

Coded inside the environment to reduce dependency on other classes. Pro reduce dependency, Con need to add conditions if got new type of enemy or different chances of spawning.