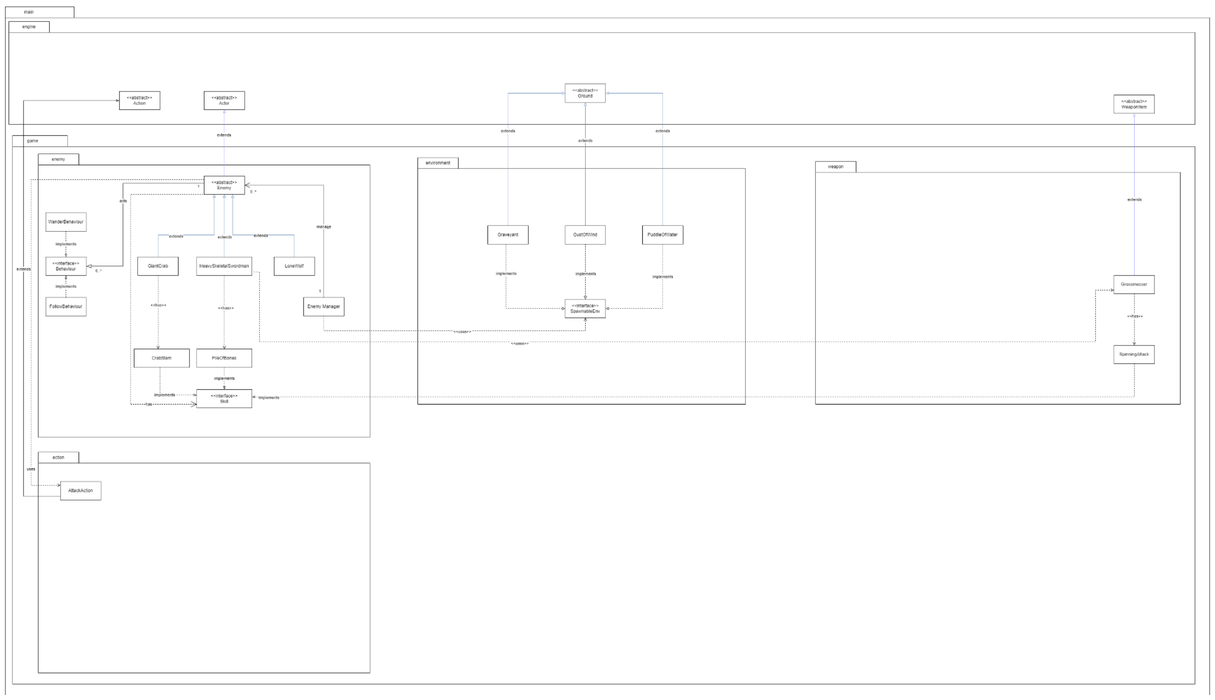
FIT 2099 group 3 assignment 1

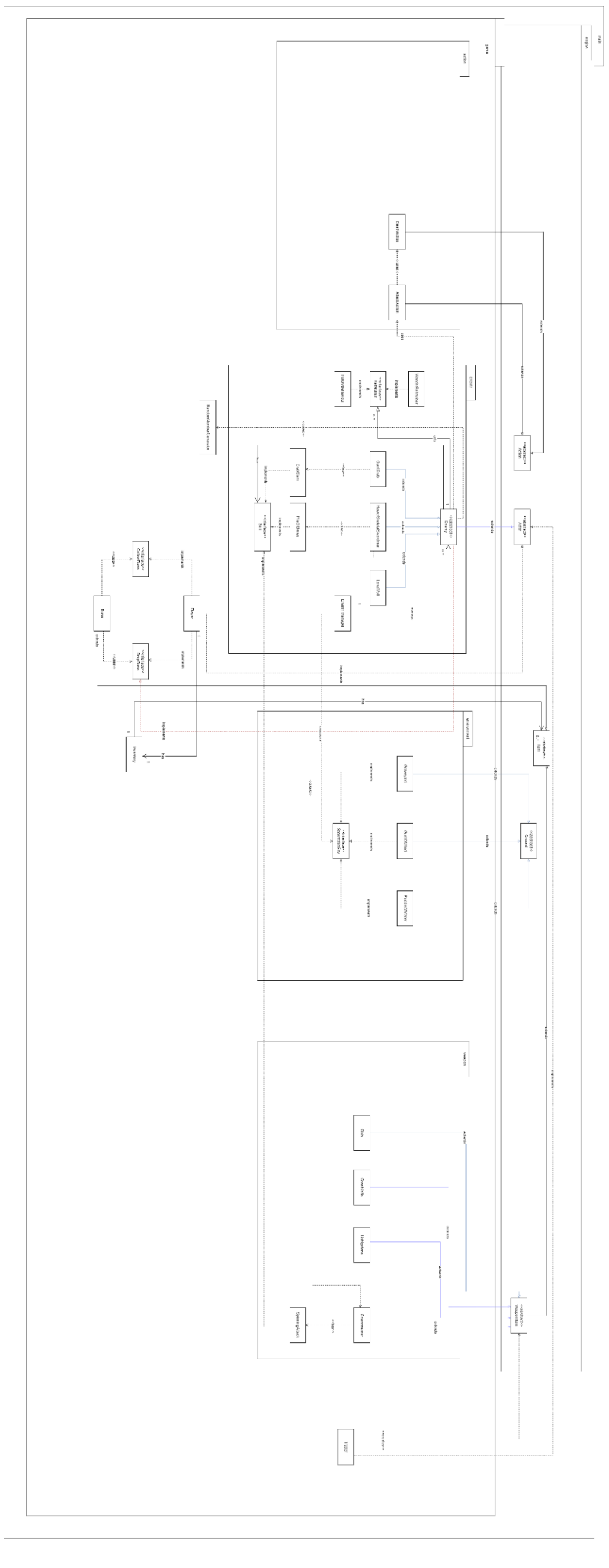
Lee Sing Yuan
Loo Li Shen
Yeoh Ming Wei

REQ 1:UML


main
engine
<<abstract>> Action
<<abstract>> Actor
<<abstract>> Ground
<<abstract>> WeaponItem
extends
game
enemy
<<abstract>> Enemy
1
0..*
acts
manage
WanderBehaviour
implements
<<interface>> Behaviour
0..*
implements
FollowBehaviour
extends
GiantCrab
HeavySkeletalSwordman
LoneWolf
1
Enemy Manager
<<has>>
CrabSlam
PileOfBones
implements
<<interface>> Skill
has
environment
Graveyard
GustOfWind
PuddleOfWater
<<interface>> SpawnableEnv
<<uses>>
weapon
Grossmesser
<<has>>
SpinningAttack
action
uses
AttackAction

Rationale

REQ 2: UML



REQ 2: Rationale

2.A)

rune class , drop rune interface and collect rune interface
The **goal** of the rune class is to be able to drop and collect runes. It is **implemented** by considering runes as items without capabilities. WE will ensure that runes are never able to add or remove capabilities by overriding the methods.

For enemies,
Enemies will implement drop runes and get a random amount of rune upon instantiation of the RUNE class

For player
Player will implement both drop and collect rune.

The alternative to this method was to have runes be represented by an integer. The issue with this method is how would we display the runes when dropped.

2.B)

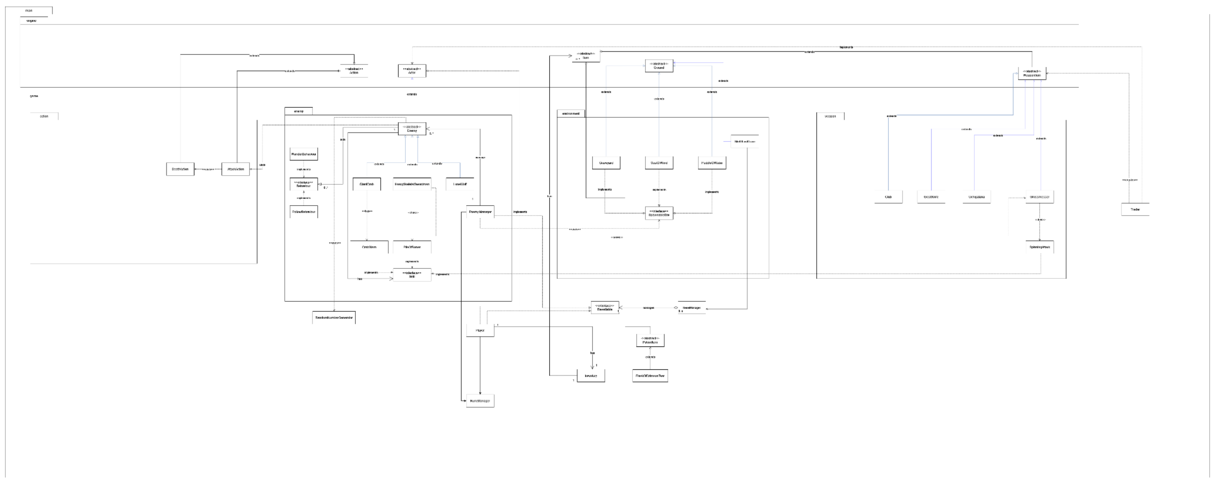
Trader
The **purpose** of the trader class is to handle selling and buying of weapons. Since, the trader can be considered an actor, it inherits the actor class.

Trading
Selling or buying weapons, will be implemented using an if condition. For example: if this weapon is sellable, then sell else don't sell. This method follows the concept of **open close principle**, as in if a new type of weapon exist and its sellable but not buyable, we don't have to modify the code in the trader because we just have to set the attributes of the weapons to be accordingly.

2.C)

Creating weapons
All weapons will inherit from weaponItem **to apply the DRY principle.**

Storing weapons
The player will be associated with a class called Inventory which acts as an itemManager. The **goal** is to separate the player from managing the inventory(SRP) .The inventory will be **implemented** to have an ArrayList<Item> and to add and remove items from the inventory.



REQ 3: Rationale

In this design rationale, the "PotionItem" class is created as a parent class for all future potions. This is a good approach since it allows for defining common properties and methods that all potions will have, such as a name, description, and addCapabilities like healing. This adheres to the Liskov substitution principle where objects of a superclass should be able to be replaced with objects of its subclasses without affecting the correctness of the program.

Furthermore, the "FlaskOfCrimsonTears" class can inherit from the "PotionItem" class and add the "heal" capability. This approach makes it easy to add new potions in the future by simply extending the "PotionItem" class and adding the specific capabilities for each potion.

Additionally, an abstract "PotionItem" class is used for all potions because eventually the potions would be used up, so resetting would be made easier. The "Site Of Lost Grace" class is created to inherit from the abstract ground class, and it has an association with the "ResetManager" class, which is responsible for resetting the state of the game. The "ResetManager" class manages the "resettable" interface, which is responsible for resetting the "enemyManager" and "Player". The use of the "enemyManager" class is good because it handles all of the enemies rather than going through the enemies individually.