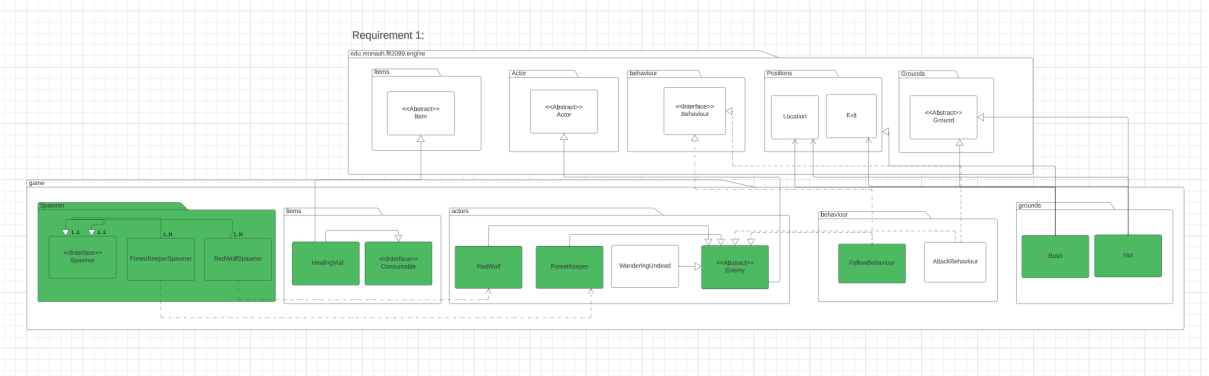
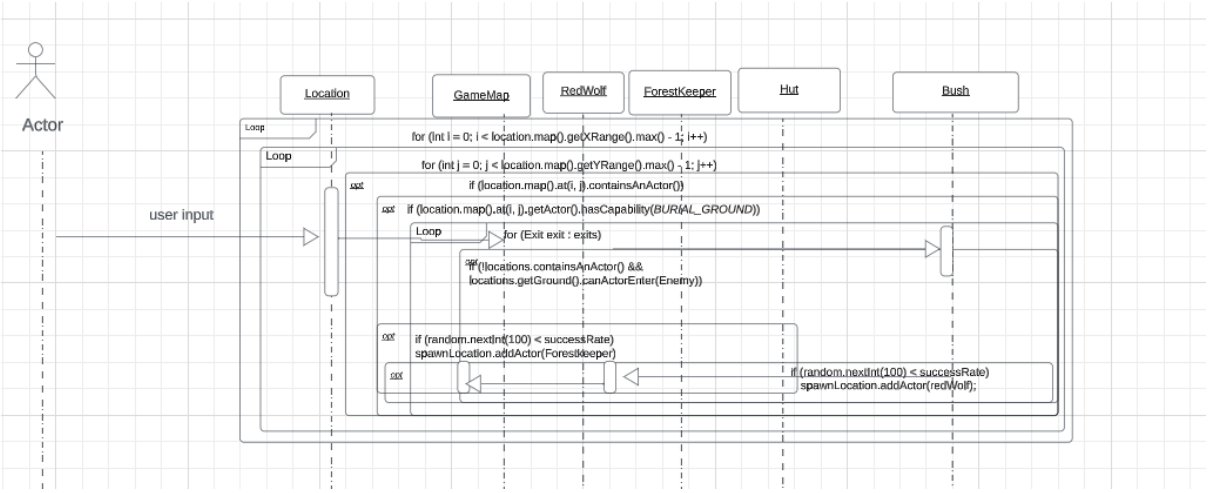
FIT2099 Assignment 2 Design Rationale - Andrew Lee 30864941

Requirement 1 (UML diagram)



Requirement 1 (Sequence Diagram)



Design Rationale:

Pros:
Spawner:
I have created a spawner interface to handle spawning action, that is implemented with each of the enemy spawning behaviour, extracted the common functions and put it in my interface so whenever I create a new enemy, It would follow the same structure thus making it easy to implement. Furthermore, this abides by the single responsibility principle and open close as it is open for extension and close for modification as much as possible.

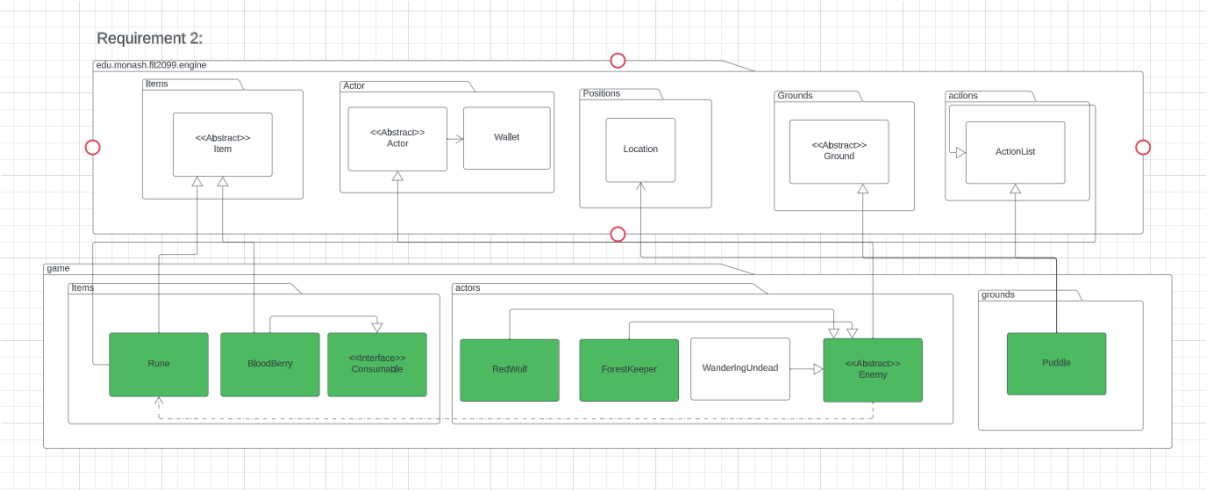
FollowBehaviour:
This is abstracted and implemented via a behaviour interface and this follows the single responsibility principle. The follow behaviour encapsulates that and it is also put in an abstract enemy class where all enemies share the same characteristics, such as Wandering, Attack and Follow. This makes it easy to create new enemies that follow this specific set of behaviours and makes it very extensible by nature, without modifying other existing classes.

Cons:
The implementation that I have created does not fully encapsulate the idea of Open Close, as I have only tried to deviate responsibilities to other classes, which is single responsibility

but It is a design flaw, as the implementation is very similar to the previous but only deviated responsibilities and created more classes in OOP design.

FIT2099 Assignment 2 Design Rationale - Andrew Lee 30864941

Requirement 2 (UML diagram)



Requirement 2 (Sequence diagram)

I've introduced a Consumable interface to unify all items that restore the player's hit points or stamina. Each item under this interface can be consumed and provides a unique action, accessed via the getActionSpecific method.

Pros:
Single responsibility principle:
The consumable separates the concern from the others, as it is only handling the actual healing and replenishing of the Baseplayer's attribute without affecting the correctness of the program.

Open/Closed:

The game's design is open to extension and closed for modification, everytime we add a new class that falls under consumable, it will have all the methods for healing the player, the class simply needs to override to apply individual characteristics of its consumable, the actions are also returned by itself which we don't need to handle or use additional enumerations for.

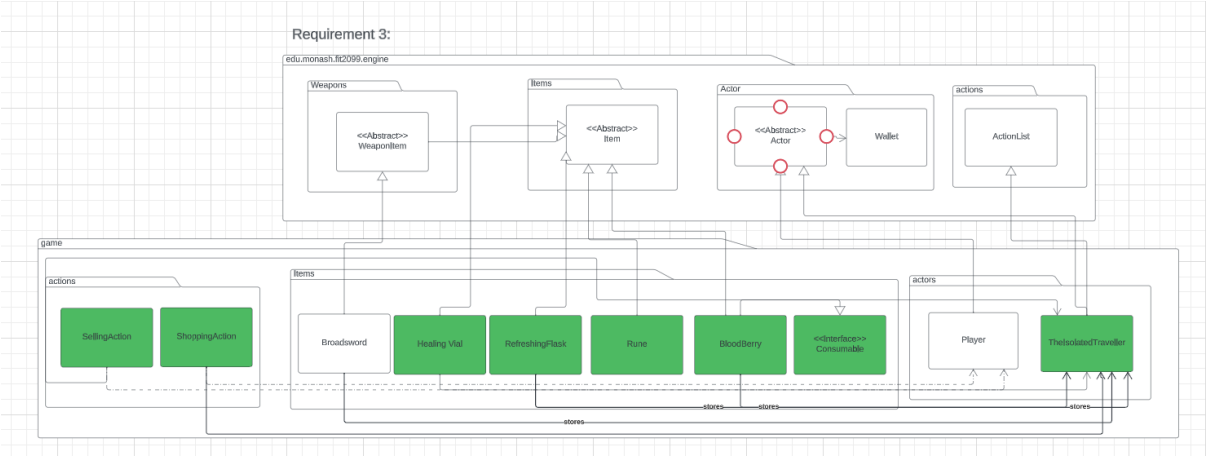
Dependency Inversion:
We are relying on abstractions rather than actual concrete classes, this will ensure our high level logic is not dependent thus reducing coupling in our program.

Cons:

Downcasting:
For handling consumables, our program still requires downcasting in our main method for calling. However, the pros outweigh the cons in this case.

FIT2099 Assignment 2 Design Rationale - Andrew Lee 30864941

Requirement 3 (UML diagram)



Requirement 3 (Sequence diagram)

Pros:

Single responsibility Principle:
ShoppingAction handles buying items for the player whereas SellingAction handles selling the actual items.
This indicates that there is only one responsibility for one action, which provides the layer of separation of concern. Furthermore, this promotes information hiding as the external actors do not know what the actions are in the main method call without executing the program for output.

Open/close:

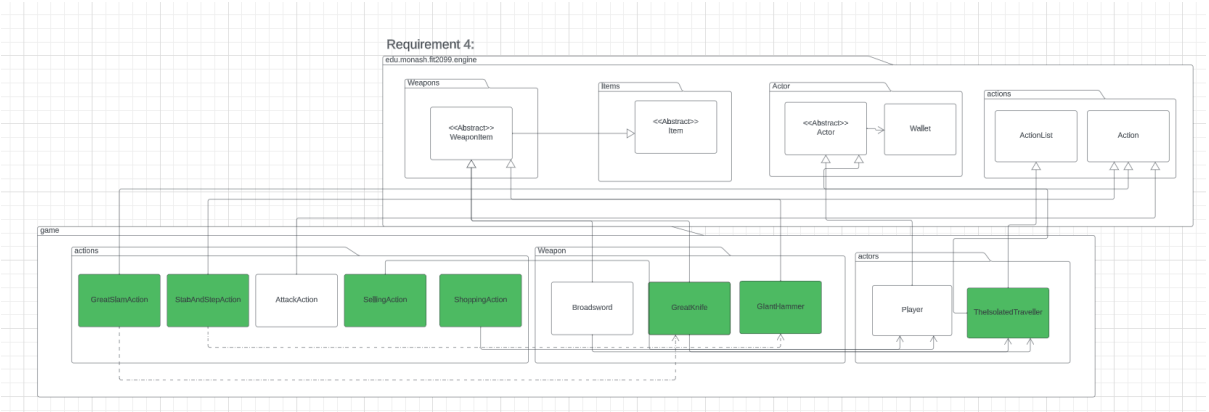
Although this can be improvised, it sort of abides by the Open close as I have implemented a hashmap, so there is no need for major changes to the existing code. Although, the main class of method calls still needs to be modified, such as IsolatedTraveller can add more items in the inventory and put buying and selling prices. In the future, we would like to have an interface to handle this if it is possible, to implement an int getprice() so that it can return the price of the items.

Cons:

We are relying on TheIsolatedTraveller to provide us the option to sell, which means we are relying on that concrete class instead of an abstraction, if TheIsolatedTraveller suddenly gets deleted.

FIT2099 Assignment 2 Design Rationale - Andrew Lee 30864941

Requirement 4 (UML diagram)



Requirement 4 (Sequence diagram)

Changes:
I have implemented a skillable interface that activates and deactivates a weapon while returning an action specific under the enum ability can attack. So a skillable interface will be implemented to all the weapons that have a special skill, furthermore, I make it return a specific status (not a good approach) but a way to bruteforce attack action.

Pros:
Single responsibility for each of the actions as mentioned above follows the same separation of concern.

Dependency inversion:
We have a skillable interface that we rely on instead of the actual concrete classes. This means

Cons:
I made it return a status in order for the attack action to recognise this is a weapon with skills. However, this is a bad approach simply because I did not account for the case where an intrinsic weapon does not fall under skillable. In order to bypass that, I had to make an instance of a skillable weapon for the implementation to work. This would indicate that it is not extensible, as you would have to modify the classes again and again in order to implement new weapons and behaviours.