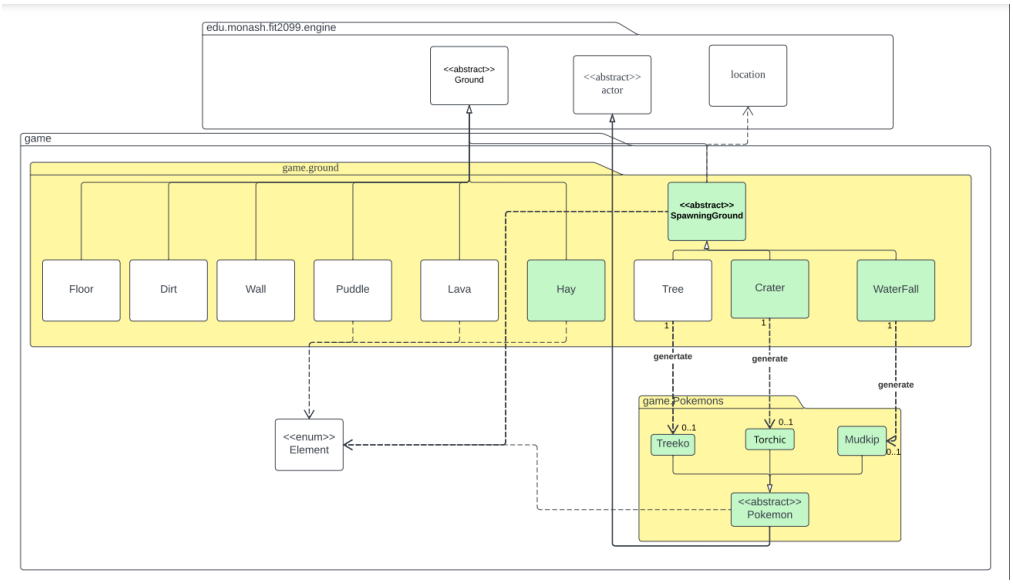
Assignment 1:Zirui Liu REQ1 & REQ2

Requirement 1: environments

Uml Diagram-



Design Rationale-

New classes to be added:

The diagram represents an object-oriented system for environment that has added 6 concrete classes and 2 abstract classes.They are:

- SpawningGround <<abstract>>
- Hay
- Crater
- WaterFall
- Pokemon <<abstract>>
- Torchic
-Treeko
-Mudkip

Reasoning:

Puddle,Lava,Hay,Wall,Floor,Dirt and the abstract class SpawningGround extended the abstract Ground class. Since they share some common attributes
and methods, it is logical(all of them are grounds) to abstract these identities to avoid repetitions (DRY).

Tree, Crater,Waterfall extended the abstract SpawningGround class. Since they share some common attributes and methods , Doing so can avoid repetitions (DRY).

Torchic,Treeko,Mudkip extended the abstract Pokemon class. Since they share some common attributes and methods, it is logical to abstract these identities to avoid repetitions (DRY).Also this design adherence to Liskov substitution principle as the behaviors of Pokemon is maintained in the three subclasses.

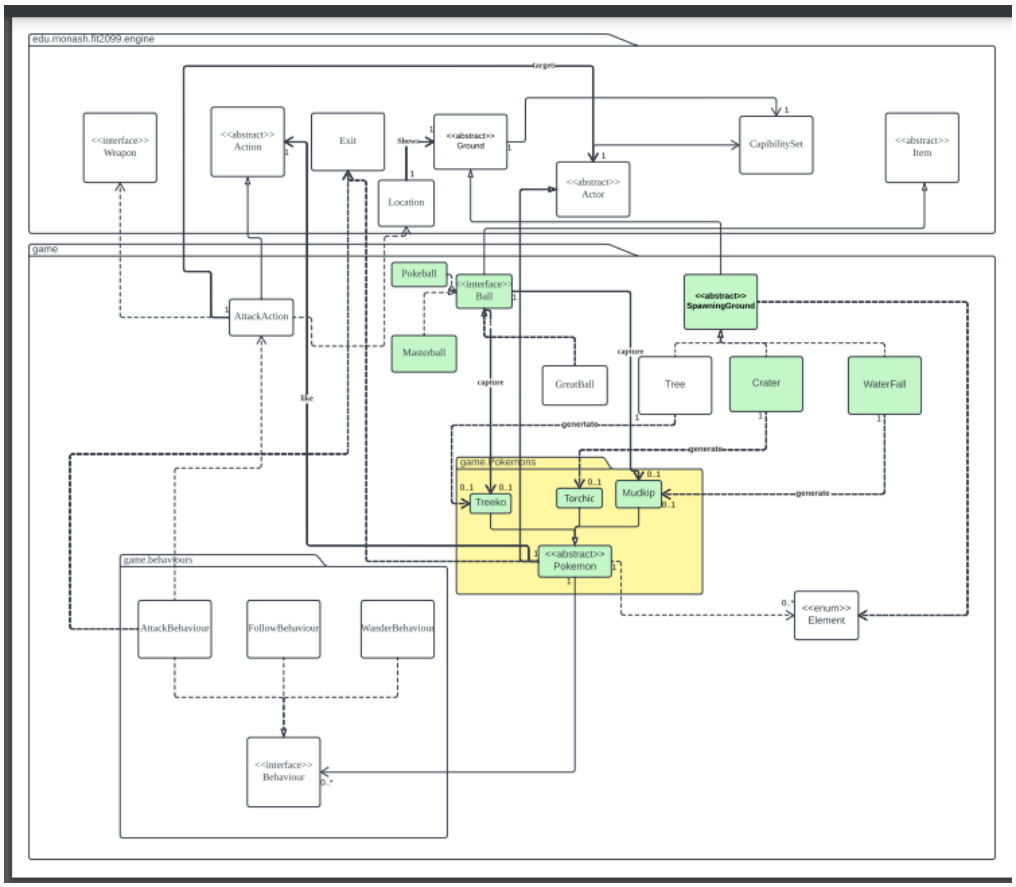
All these abstraction design adheres to Single Responsibility Principle as each of the subclasses will have their own responsibilities(for example, each spawning ground spawn different kind of pokemons specificly).

Tree has a dependency on Treeko Class, is due to the fact that each tree can spawn Treeko, so tree class knows about the Treeko and the same goes for Crater and Torchic,WaterFall and Mudkip.

Lava,Puddle,Hay,SpwaningGround and Pokemon class have dependency on Element as all of them are defined by element.The element of each ground will be stored in their CapibilitySet. All of them know about Element class but do not have element attribute as element is a enum.

Requirement 2 : Pokemon

Uml Diagram-



Design Rationale-

New classes to be added:

- MasterBall
- Pokeball
- GreatBall

Reasoning:

Automatic attack(new relationships to be added):

Abstract pokemon have an association to Behaviour abstract class because an NPC(Pokemon) have a list of attribute that stores behaviours.(according to Assignment support3)This design adheres to the Open-closed principle as when we change the subclasses of the Behaviour class,we don't need to modify code in Pokemon class.

Then the attackBehaviour class extends the Behaviour class.When we run AttackBehaviour class it will automatically run the code of check if there are some Pokemons with different elements.This check is done by iterating through every direction of destination.If there are any pokemons with different element stored on that direction destinition(data type is Location),then return attackAction. As in this process both attackAction and Exit are not stored as attributes, we apply dependency here.

Call stack: Exit->Location->gamemap.getActorAt()->Actor->Pokemon->Element

Then in the AttackAction class. It will decide which kind of weapon(or no weapon) to use based on the element of ground of the Location where the Pokemon stayThe element of the ground is

stored in the CapabilitySet of that ground already and can be accessed through functions.So AttackAction have dependency on Location and Weapon class.

Pokemons:

Each pokemon has a favourite action. Therefore I give the abstract Pokemon class an Action attribute called favouriteAction.Therefore pokemon have an association to the action.(KISS principle,keep it simple stupid).

Pokeball:

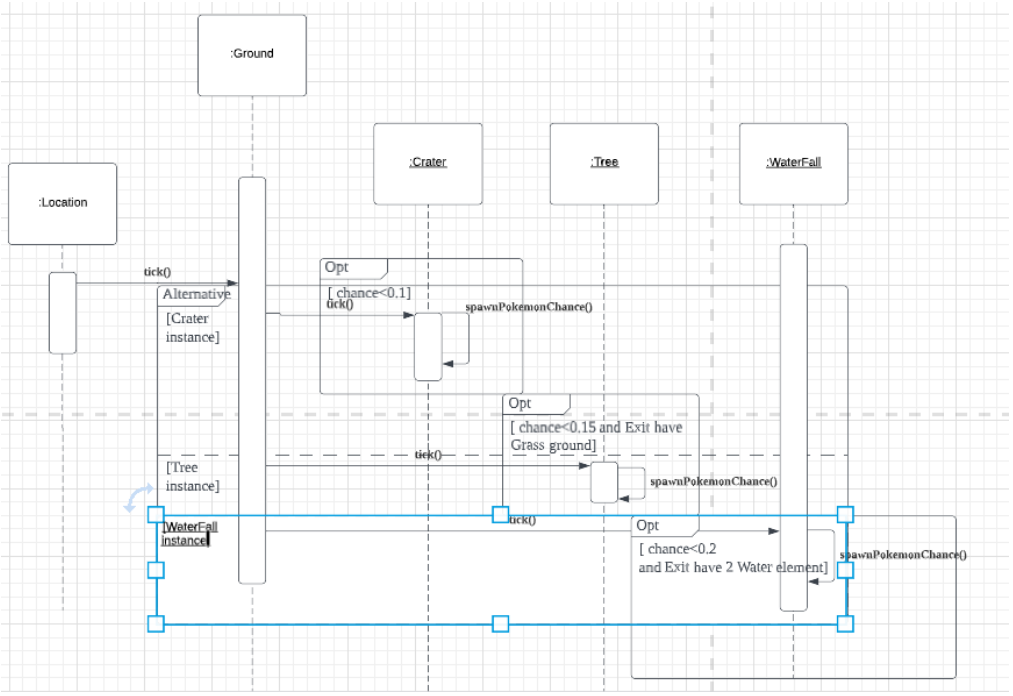
This part is planned to be done by another group member in this group but he dropped out. As it is related to req2 I will include part of it in my uml diagram.

I created an interface Pokeball which extends Item class to realise abstraction.Pokeball,Masterball,GreatBall extend from it as they share same attributes and method on capturing the Pokemon.

This design adheres to the single responsibility principle as each subclass have their own task and each of them represent one single kind of ball.

Catch means having an attribute of Pokemon instance inside the Balls.Therefore the Ball abstract class have association with Treeko and Mudkip.The association have multiplicity of 1 and 0...1 as one ball can catch 0 or 1 Pokemons.

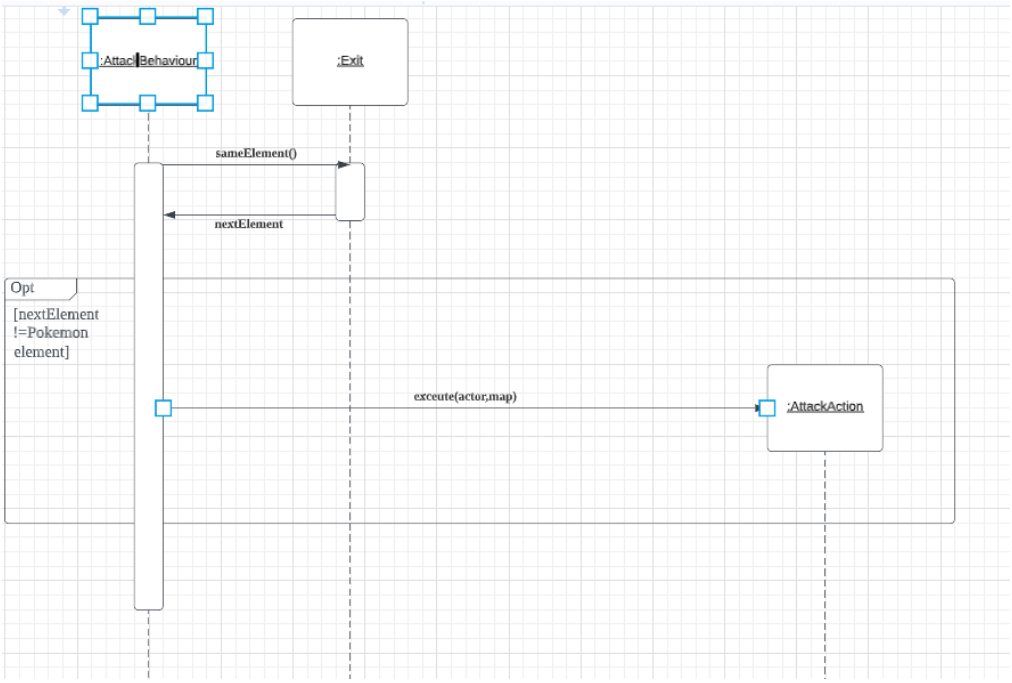
Interaction Diagram 1: Spawning



Interaction Diagram Information-

Location tick() will invoke Ground tick(). Then Ground tick() will invoke spawnPokemonChance() in instance.This design adheres to KISS principle.

Interaction Diagram 2: automatic attacking



Behaviour will be checked at the beginning of execution.If element different, the automatic attack will have.This design adhere to KISS principle as this design made implementation simple and easy.