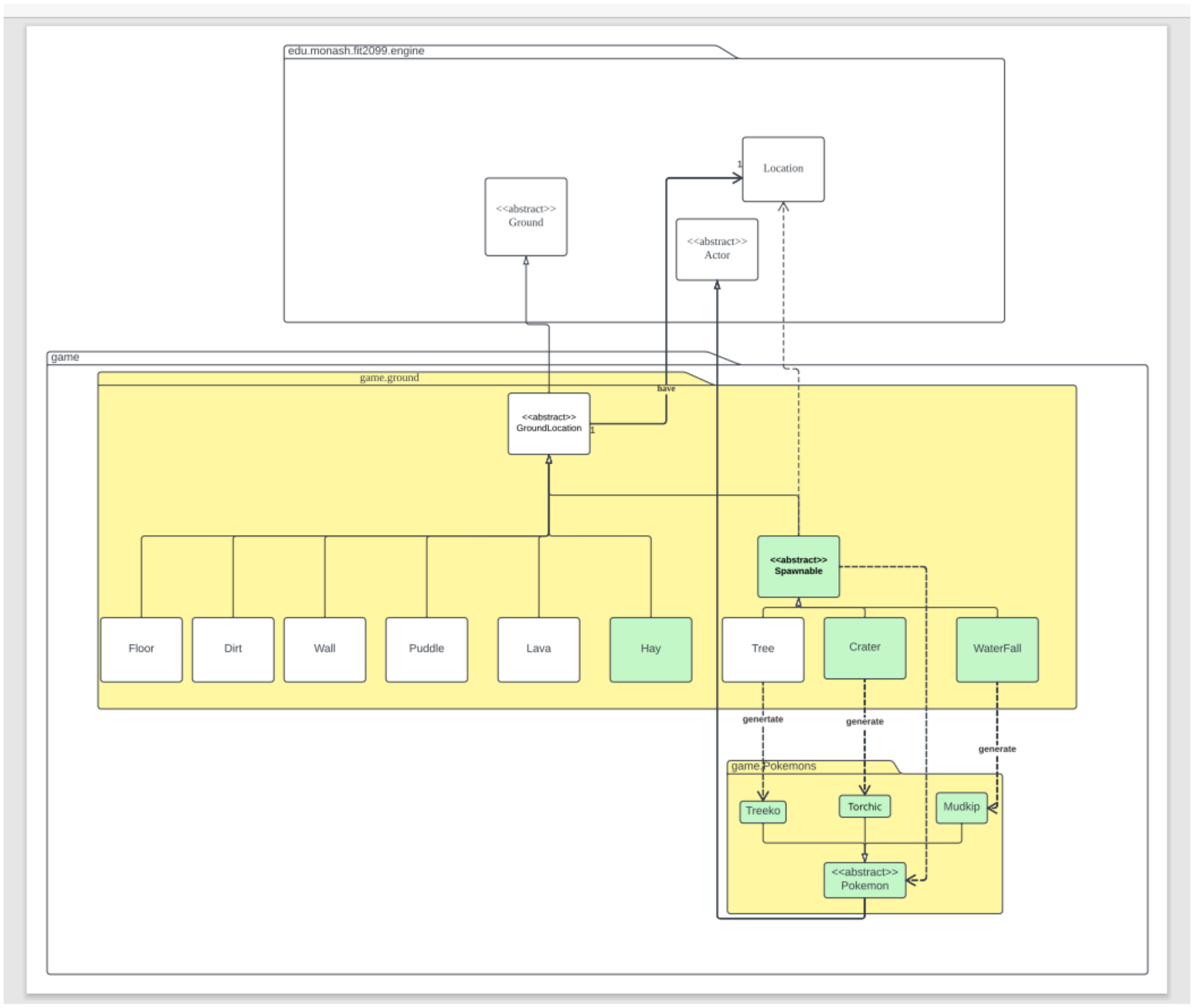
Assignment 1:Zirui Liu REQ1 & REQ2

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 3 abstract classes.They are:

-GroundLocation<<abstract>>

- Spawnable <<abstract>>
  - Hay
  - Crater
  - WaterFall
  - Pokemon <<abstract>>

- Torchic
-Treeko
-Mudkip

Reasoning:

1.

Puddle,Lava,Hay,Wall,Floor,Dirt and the abstract class Spawnable extend the abstract GroundLocation 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).

PROS:

We implement the spawnable mechanism inside the Spawnable class but not in each individual class.This approach can help us achieve abstraction and can avoid concrete spawning classes like Tree depending on Location directly.

CONS:

No CONS here.

2.

GroundLocation Class extends Ground and associates to Location class.This is for req4 and will be discussed later in req5.

3.

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 kinds of pokemons specificly).

Tree has a dependency on Treeko Class, 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.

PROS:

Easy for implementation. Easy to know which spawning ground generate which kind of Pokemon.

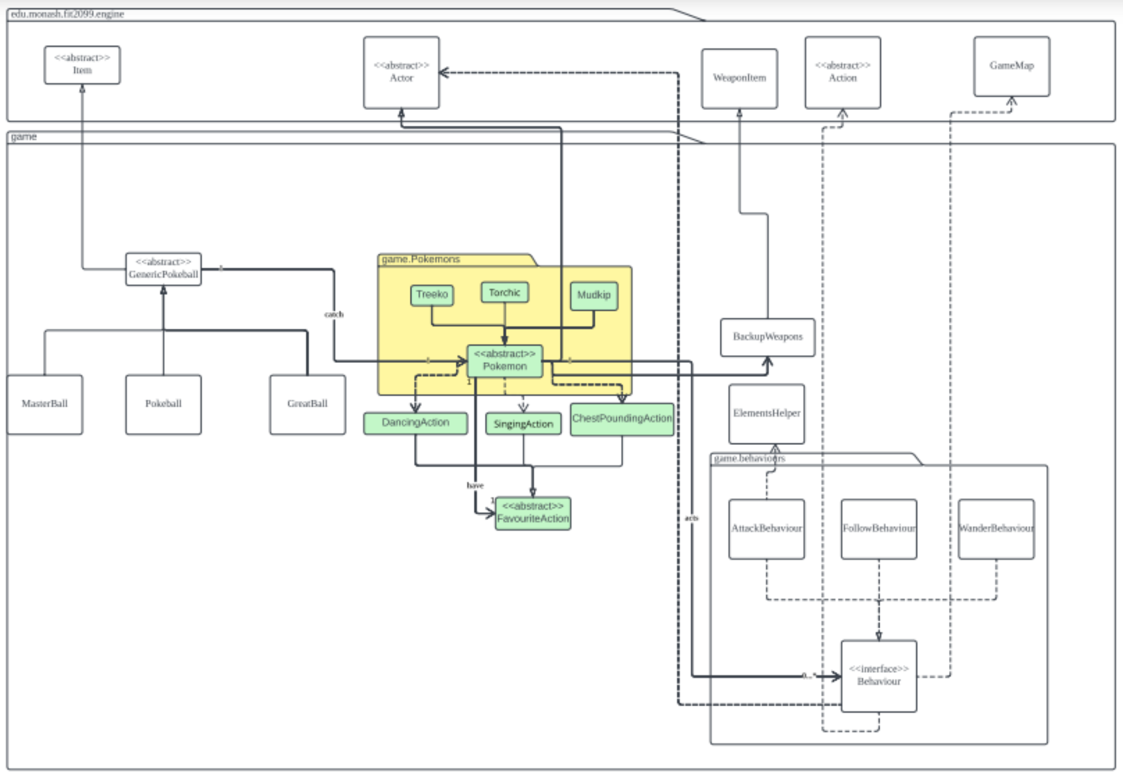
CONS:

Concrete class directly depend on other concrete classes, which may violate the Dependency inversion principle.But if we use the abstract class to decide which kind of pokemon to spawn, the Spawnble class will directly depend on concrete Pokemon classes which violates DIP and even worse.

3.
The element of each ground/Pokemon will be stored in their CapibilitySet.
PRO:
Instead of signing each pokemon an attribute of Element.We use capability set to manage the elements.This can make better use of elements and reduce dependency from concrete Ground and Pokemon classes to Element class.
CONS:
Using capabilitySet too much may lead to the program running slower as there are many loops inside.

Requirement 2 : Pokemon

Uml Diagram-



Design Rationale-

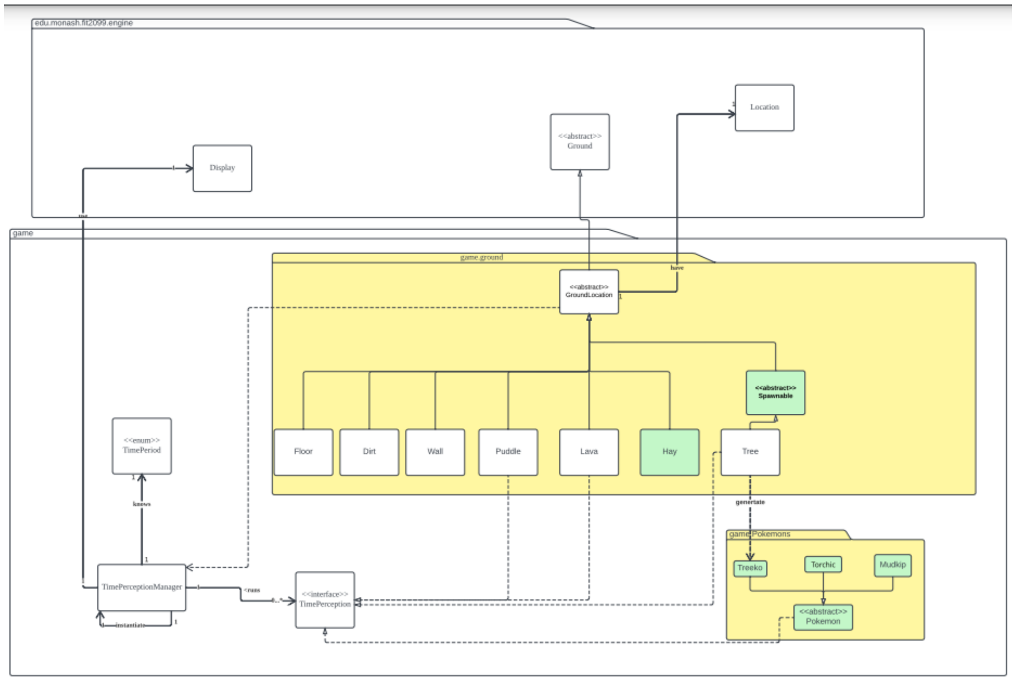
Reasoning:
1.Behaviours:
Abstract pokemon have an association to Behaviour interface because an NPC(Pokemon) have a list of attributes that stores behaviours.(according to Assignment support3).

Then the attackBehaviour,FollowBehaviour,WanderBehaviour class implements the Behaviour class.Instead of the three concrete classes depending on the action and GameMap directly,this design has Behaviour class depending on GameMap,Actor and Action.
PROS:
This practice adheres to the open closed principle as it opens for implementation but is closed for modification.This design can also achieve abstraction by depending the interface rather than concrete classes instead of the concrete ones,which can avoid lots of unused dependencies.

Pokemons:
Each pokemon has a favourite action. I associated the pokemon abstraction class to the three concrete classes of favourite actions.
CONS:
This may be a bad practice as this may violate the Dependency inversion principle.But since we are asked to list all these three actions in the menu.Therefore We have to do this instead of just having an association between Pokemon and favouriteAction class.
PROs:
Nothing is connected to concrete Pokemons which is good.This adheres to the open closed Principle as it is open for inheritance but closed to modification
Pokeball:
I created a GenericPokeball class to achieve abstraction.Pokeball,Masterball,GreatBall extend from it as they share the same attributes and method of capturing the Pokemon.Compared to using only on Pokeball class and define them by enum, 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 Pokeballs.Therefore the abstract class has association with Pokemon.Compare to depend the GenericPokeball on Pokemons , this practice has better maintenance as it adheres to DIP.(The abstraction depends on abstraction rather than concrete class).
PROs :better maintenance , abstraction achieved.

Requirement 5: Day and Night

Uml Diagram-



Design Rationale-

New classes to be added:

- None

Reasoning:

1. TimeperceptionManager have association on Display class instead of just using system methods to print.
   PRO:

All codes are in the Display class but not separated in the system, which makes further modifications(like added features) easier and display nicer.Otherwise if we use the System method to print,we would need to go through all the files to find the System.out.println usages and change it to use the methods provided by the log4j library when trying to add more features in the future.

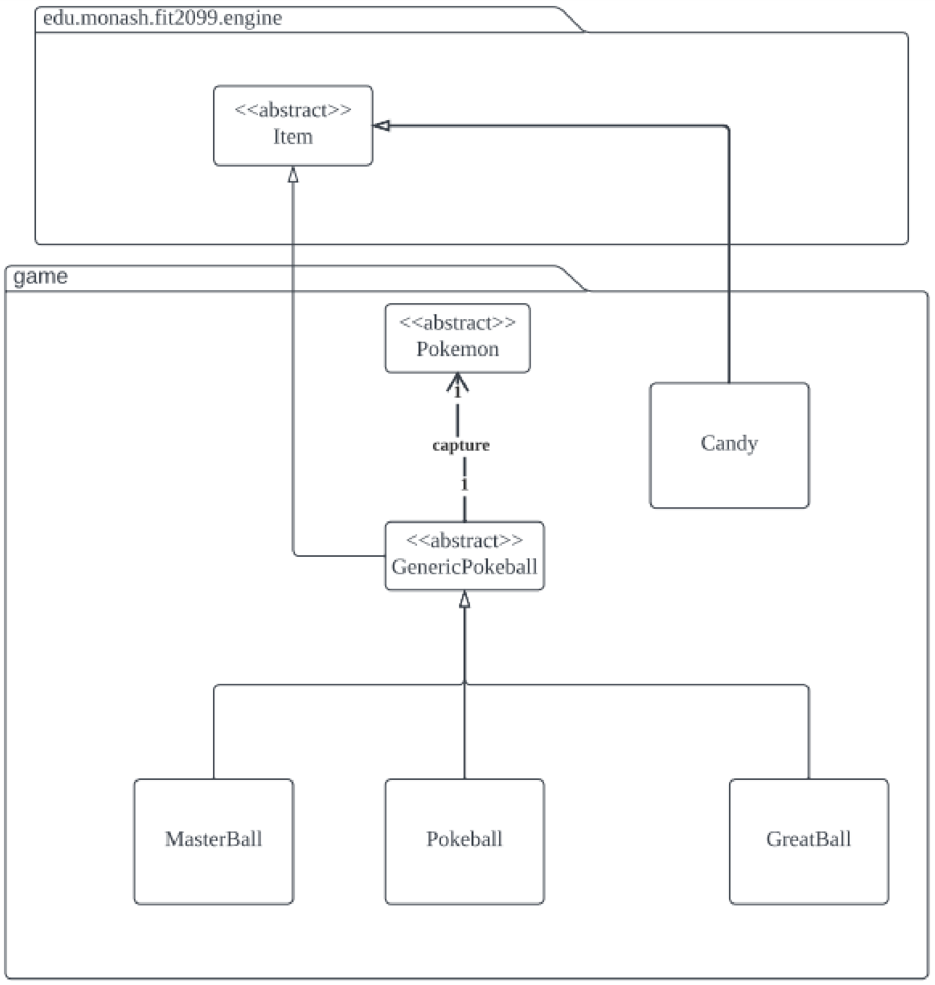
Cons:one more dependency added, which added more complexity.

2. Added GroundLocation abstraction class which associates to Location.

   Compare to the design of association concrete ground classes to Location, this design have pros and cons:

   PROs:Better maintenance;achieve abstraction;adhere to open closed principle as it is open to inheritance and close to modification.

   CONs:One more level of abstraction increases risk of violate Liskov substitution principle in the future.As too much abstraction may result in some inheritance issues.(subclasses may not behave in the same way in the future).


edu.monash.fit2099.engine
<<abstract>>
Item
game
<<abstract>>
Pokemon
1
capture
1
Candy
<<abstract>>
GenericPokeball
MasterBall
Pokeball
GreatBall