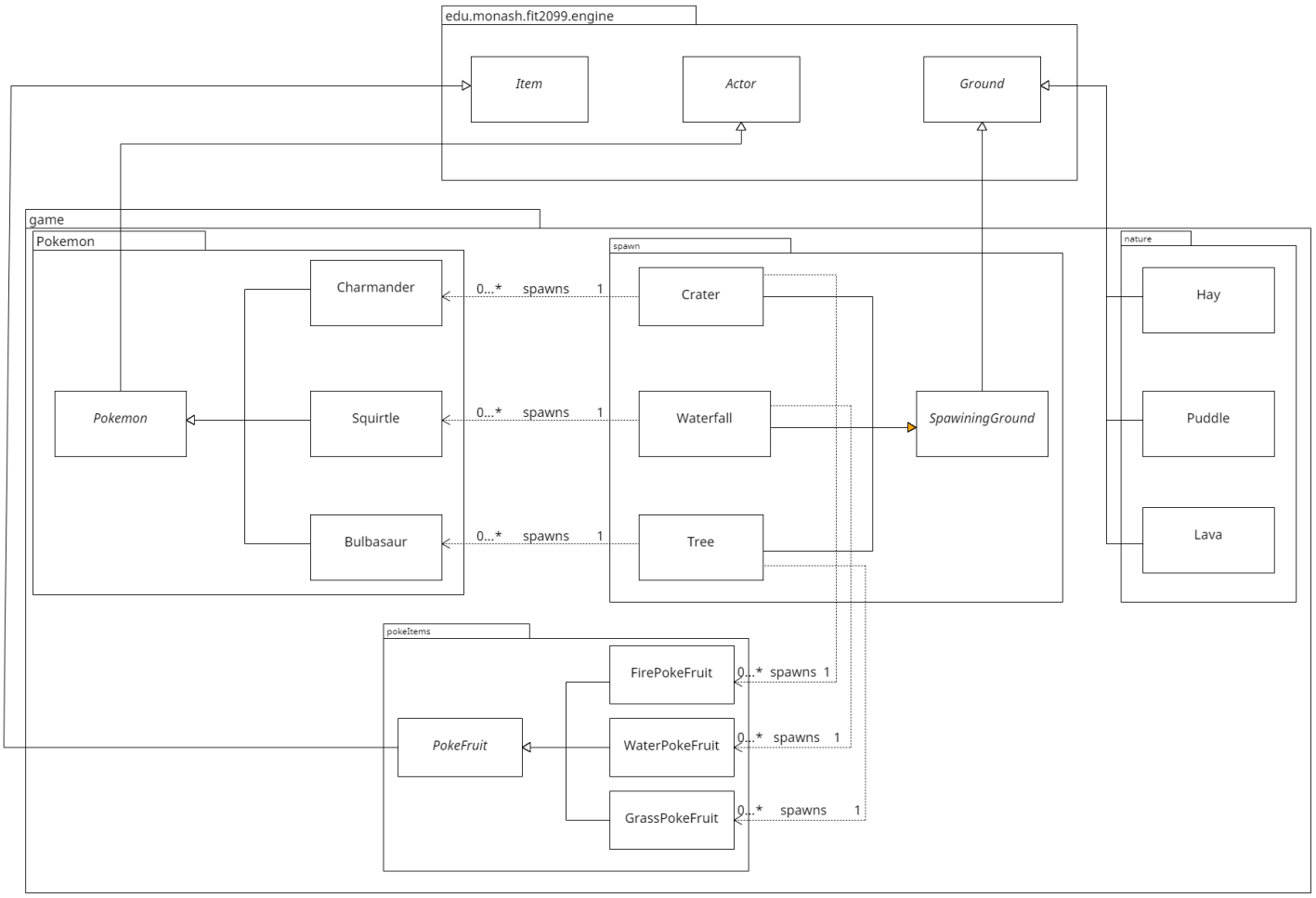
Name: Malcolm, Cecilia, Jobin
StudentID: 32578393, 32375573, 32871953
Lab 3 Group 4, Thursday 1pm - 4pm, Malaysia Campus
FIT2099 Assignment 1

REQ1 - Environment

Class Diagram



Design Rationale

In this requirement, SpawningGround extends Ground.

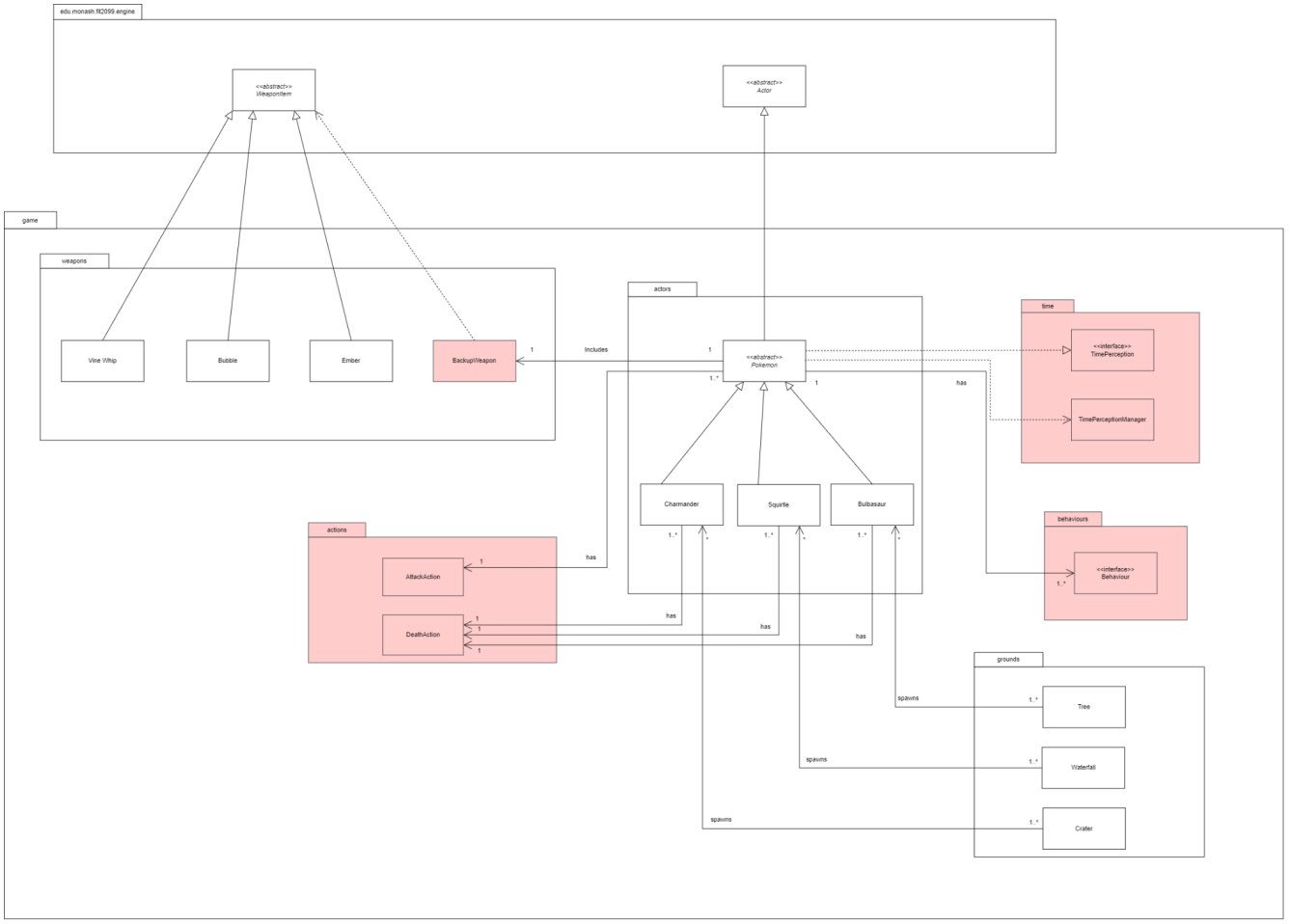
In the SpawningGround class, it has a method that helps check whether the chance of something spawning occurs or not.

Crater, Waterfall, Tree ---<<extends>>---> Spawning Ground. Crater, Waterfall and tree are the object that causes SpawningGround to randomise and checks whether the probability of a Pokemon or PokeFruit spawning.

Alternatively, we can create a method for probability inside the Ground class. However,doing this requires that all ground types can use the SpawningGround, which is not true in this requirement. This results in tight coupling as well as doesn't stick with DRY principle. Therefore the team sticked on the idea of another abstract class to be made called SpawningGround.

REQ2 - Pokemon

Class Diagram



Design Rationale

The diagram represents a part of a game that includes 6 packages which have 13 concrete classes and 3 classes inheriting an abstract class and 2 interfaces.

All three types of pokemons extended the abstract Pokemon class. As all pokemons will share some common methods and attributes, it makes sense to place those into an abstract class to avoid repetition.

Each pokemon has an association with the environment that they spawn in, as one environment can spawn one or more of the pokemons.

The abstract Pokemon class also has a dependency on DayNight class. Everytime the game switches between night and day, each pokemon will either get hurt or be healed. By adding methods from the DayNight class that hurts or heals a pokemon, we do not have to repeatedly add those methods to each pokemon class.

The abstract Pokemon class extends the Actor class. As all types of pokemon are also included as an Actor and has attributes common to an Actor class, which means it will inherit all Actor attributes.

The abstract Pokemon class also has a dependency onBackupWeapon, which stores the special attack of each pokemon into a hashmap. The abstract Pokemon class will make an attribute of BackupWeapon and store the special attack of each pokemon by calling the instance variable.

The abstract Pokemon class has an association with the interface behaviours as each pokemon would have a specific behaviour every turn. The abstract Pokemon class would call on the instance variable of behaviour as a hashmap to store the behaviours of the pokemon with integers as keys to show the priority of each behaviour.

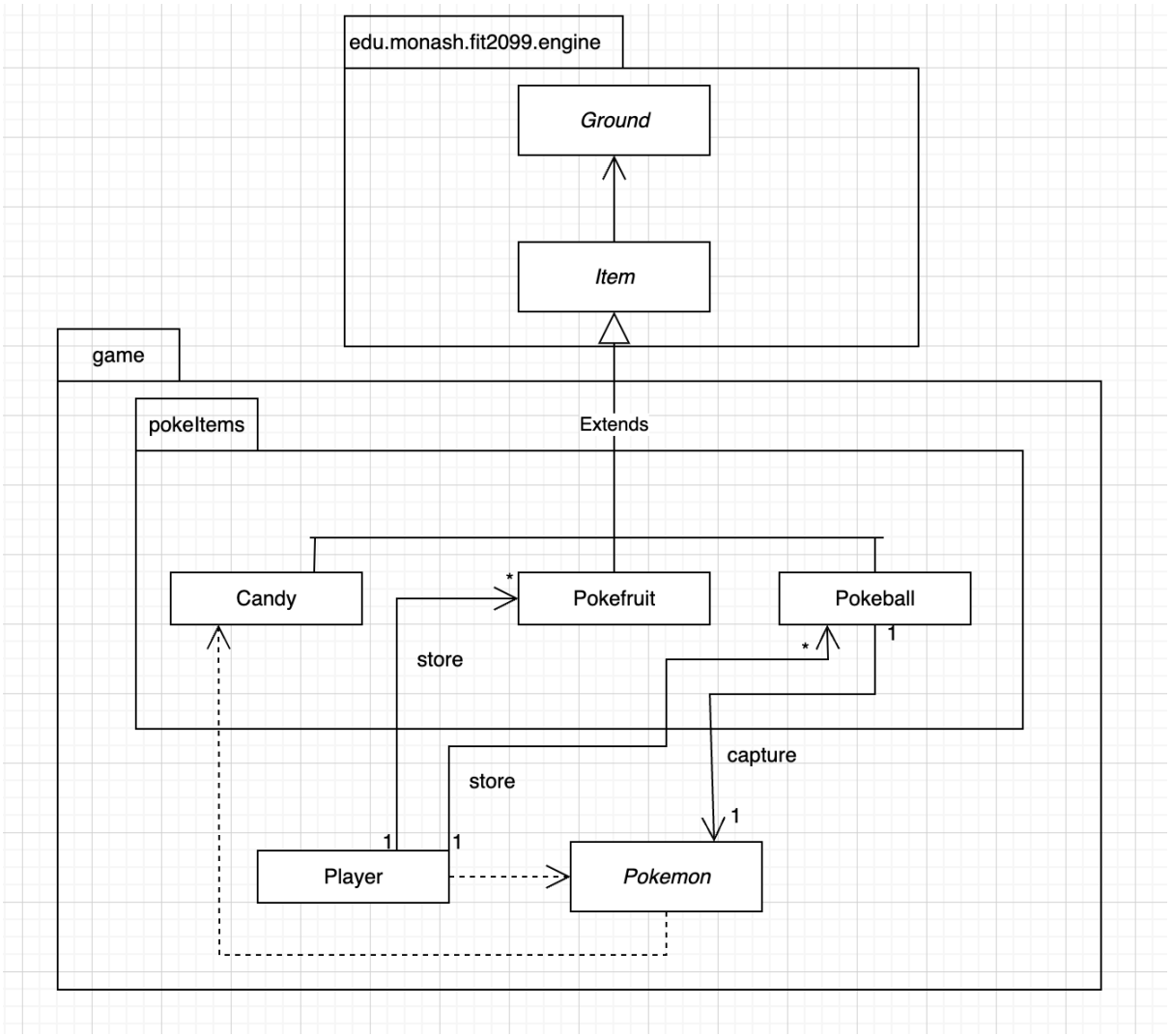
The abstract Pokemon class has an association to AttackAction as each pokemon will attack any enemy it sees. The abstract Pokemon class will initialise an attribute of AttackAction in a method that gives a new AttackAction to show what it is attacking and what direction it is attacking at.

Each pokemon class has an association to DeathAction that removes the dead pokemon from the map. In the dayEffect and nightEffect method, a certain pokemon would get hurt and may die due to the effect. If that pokemon dies, it will initialise a new DeathAction to remove the dead pokemon from the map.

*Pink coloured boxes are newly added into the uml diagram for assignment 2

REQ3 - Items

Class Diagram



Design Rationale

The diagram above shows a system for the items involved in the Pokemon game, that has 5 concrete classes and 3 of them are pokeItems that are inheriting the abstract item class.

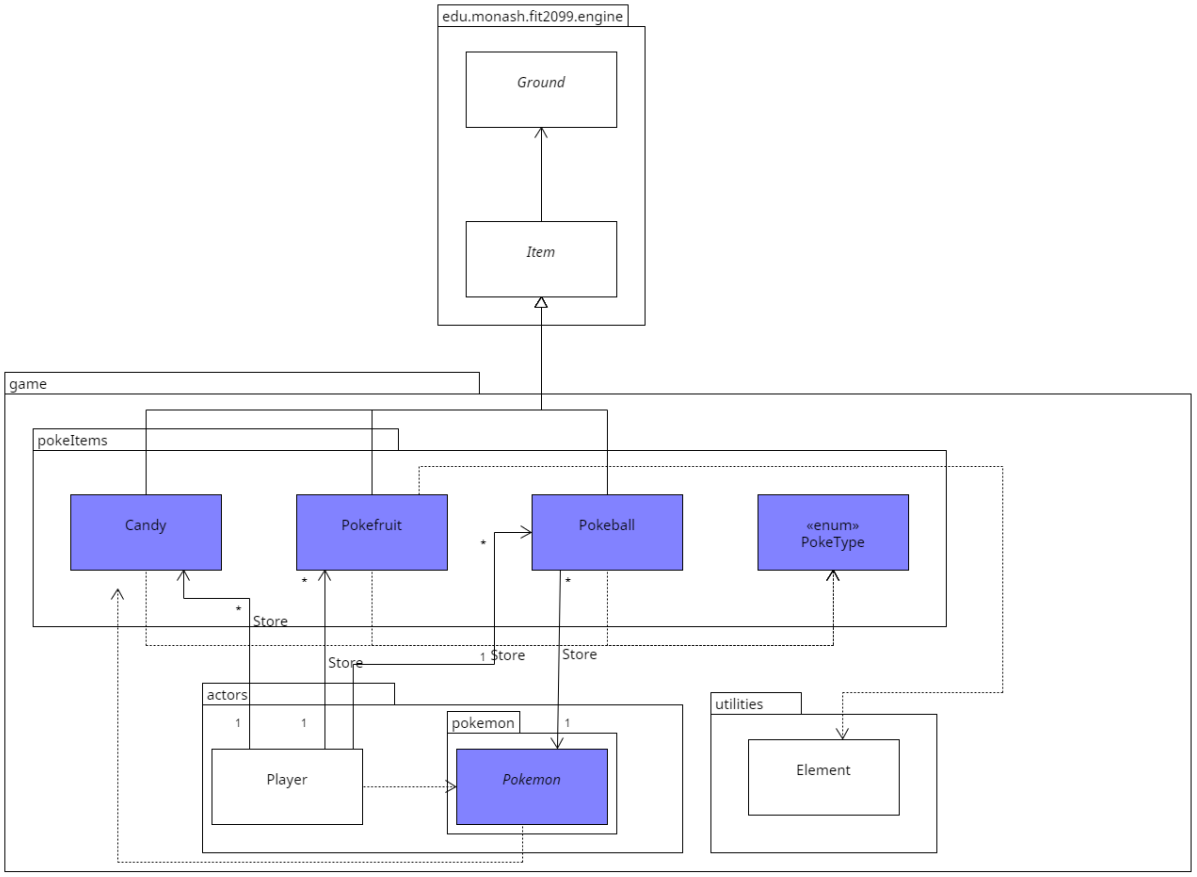
Candy, Pokefruit, and Pokeball extends *Item* . All items that inherit the abstract item class have the methods required to make the items behave properly.

All candies are given to the player by captured pokemons which is why the player has a dependency on the pokemon for the candies.

Pokefruits are picked from the ground by the player. Our design choice is the player will have an arraylist of Pokefruits in the global attribute. The arraylist will act as an inventory of pokefruits. This is why the player and the pokefruits have an association with each other.

Everytime a pokemon is captured it will be stored into a pokeball which after that the pokeball will be stored in the player class. It's the same as pokefruit where the player will have an arraylist of attributes filled with pokeballs with pokemons inside the pokeballs. This is also why the player and the pokefruits have an association with each other.

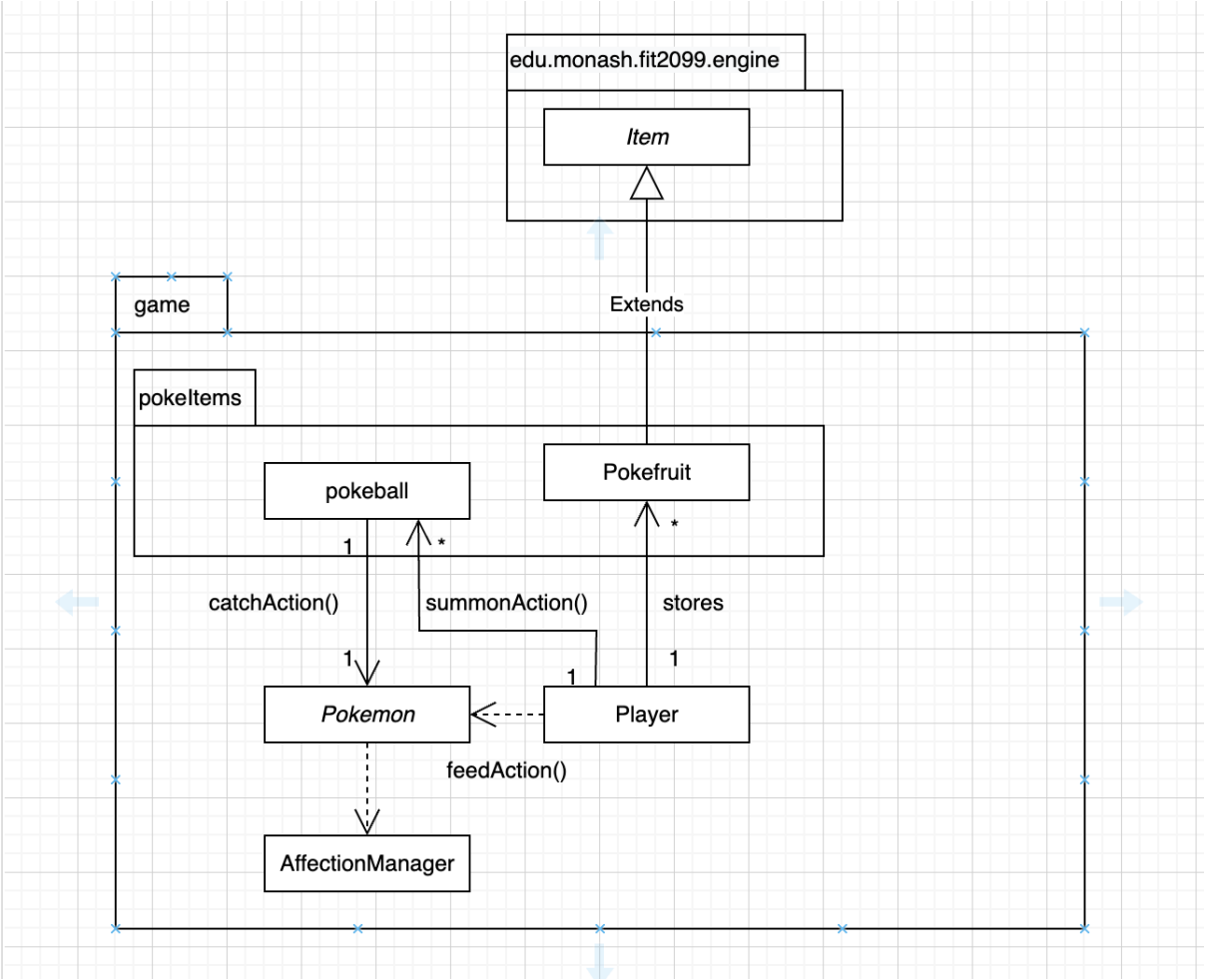
New UML Diagram



PokeFruit is not dependent on Element as Element helps differentiate between what type of pokefruit it is. We also added a new class called PokeType, where it is an enum of PokeItems, like Candy, Pokefruit and Pokeball. This will help us search for the capabilities later on with other classes. For example, we would use PokeType to help check whether candy is in the inventory.

REQ4 - Interactions

Class Diagram

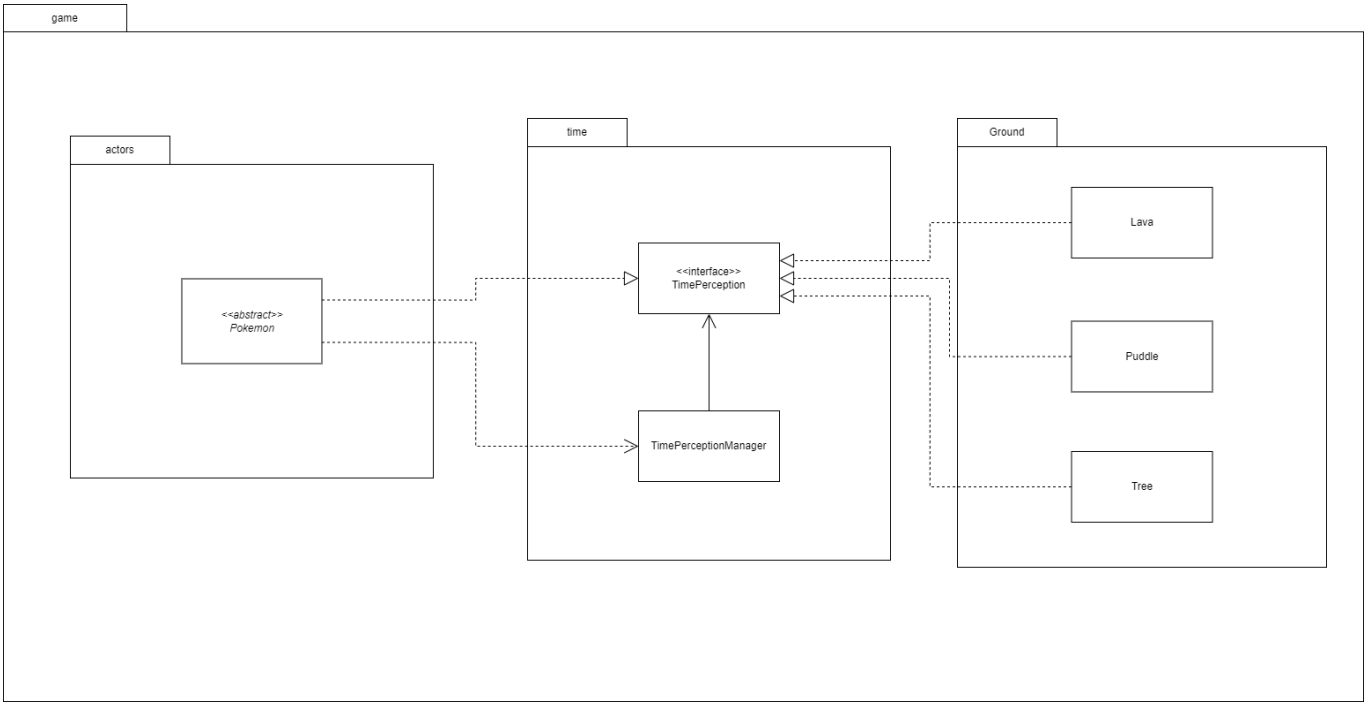


Design Rationale

The same as REQ3, pokefruits are inherited from the item class in the engine. The Player class will have dependency on the Pokemon class to feed the pokemons by using certain methods from it. In our design the Pokemon has a dependency with AffectionManager. The AffectionManager would return a pokemon's affection stats from its hashmap. When the player feeds the pokemon or does something that can affect the affection stats of a pokemon, the pokemon will use the methods given by AffectionManager to set and get the stats. When the affection stats are satisfiable for a capture, the player can catch the pokemon with a pokeball and store that same pokeball to an arraylist of pokeballs inside the Player class.

REQ5 - Day and Night

Class Diagram



Design Rationale

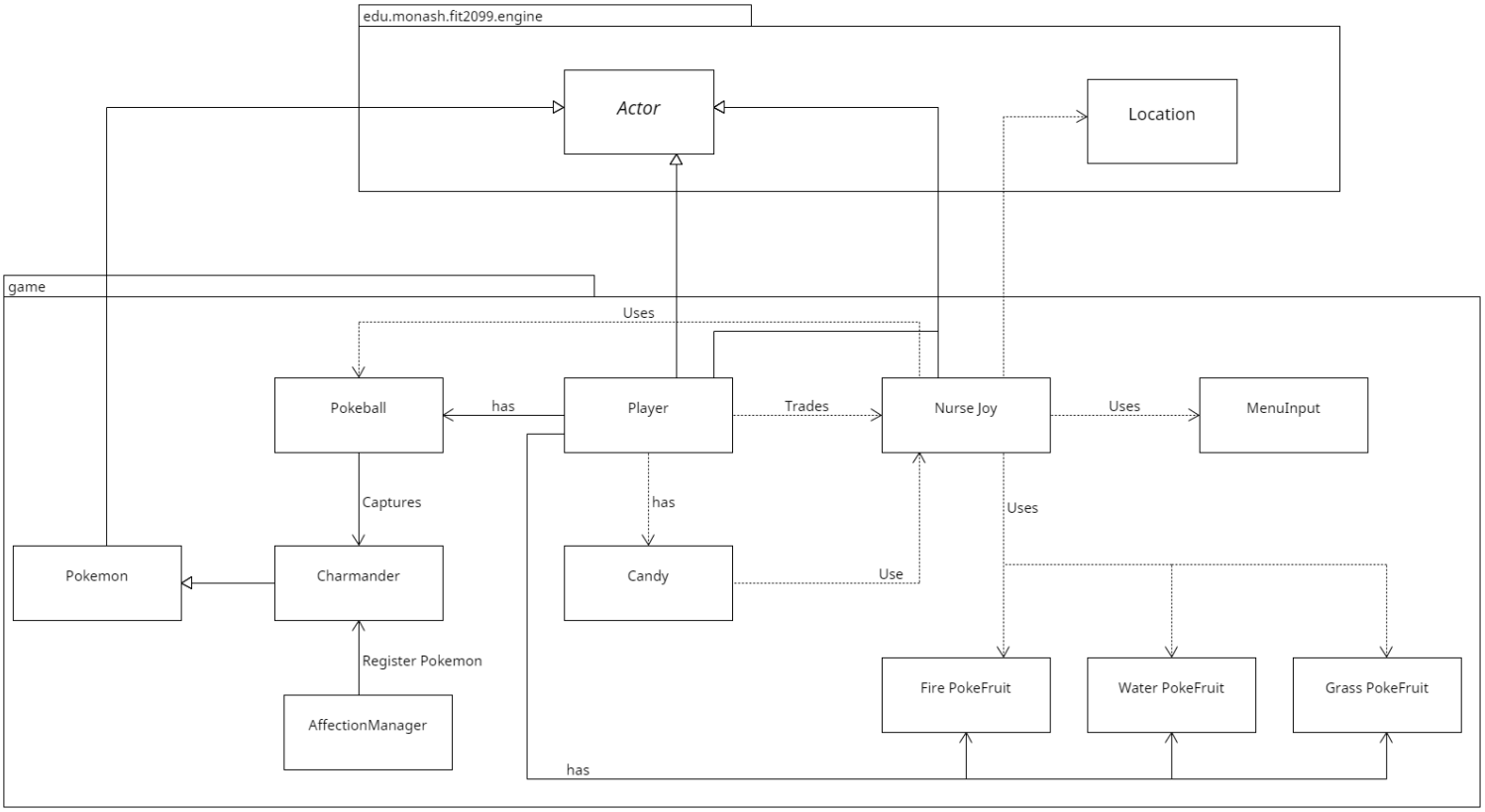
The diagram represents a part of a game that includes 3 packages which have 4 concrete classes and one abstract class and one interface.

The abstract Pokemon class implements the TimePerception interface as whenever the game switches between day and night, a pokemon will either lose health or gain health. We used the dayEffect and nightEffect method and override it in each pokemon class to implement the effects that will happen to the pokemon during day and night.

Lava, Puddle and Tree class implements the TimePerception interface. Whenever the game switches between day and night, each environment in the ground class will either expand or get destroyed. We used the dayEffect and nightEffect method and override it in the 3 grounds to implement the effects that will happen to each respective ground during day and night.

REQ 6 - Nurse Joy

Class Diagram



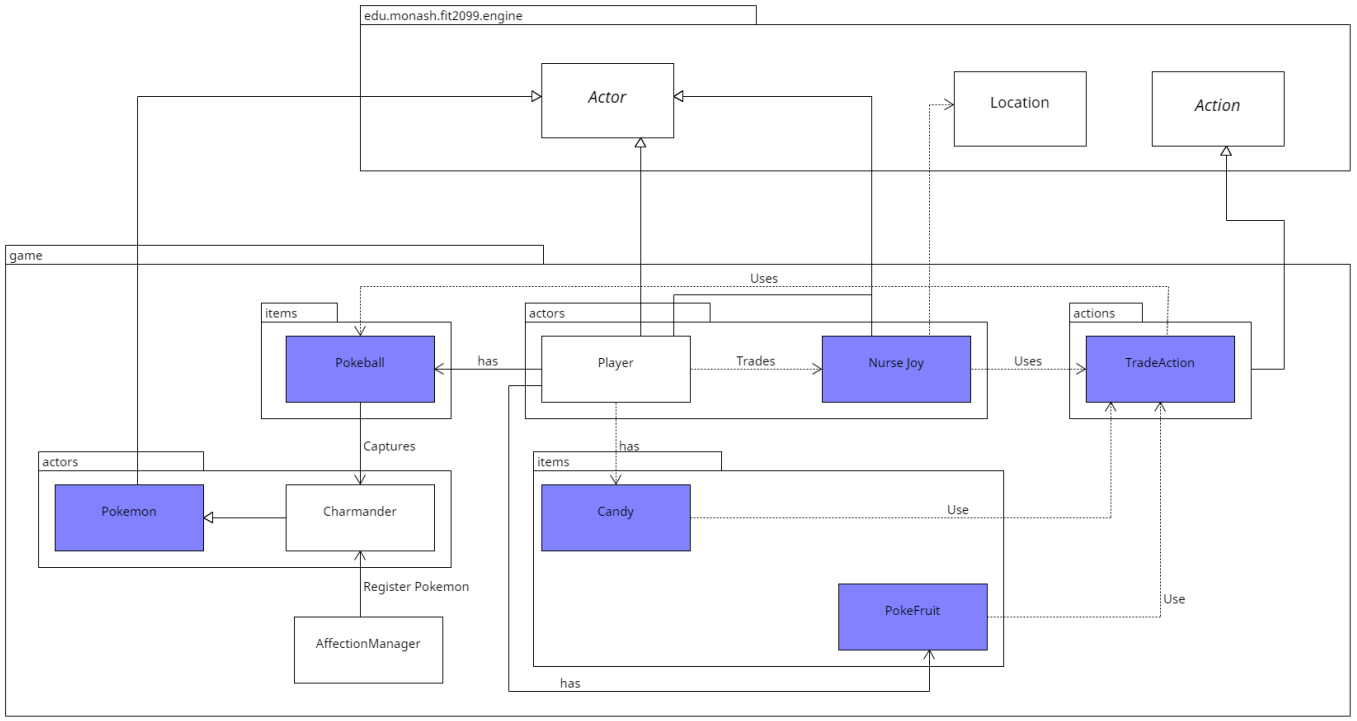
Design Rationale

Player ---<<Trades>>---> Nurse Joy. The first feature mentioned by REQ 6 is the idea of for the player to be able to trade with Nurse Joy. The Player Class can use the NurseJoy Class to help buy items from here. Therefore having menuInput Class helps the Player navigate the market easier and faster. Therefore it means that MenuInput would be a single purpose class, resulting in it to follow the Single Responsibility Principle. This means that the overall package would be more robust, resulting in high cohesion. In addition can cause the object to be an ideal object or a god object

One of the Main important features of Nurse joy is the ability to trade certain amounts of candy to get either a PokeFruit or a charmander inside a pokeball. Therefore it means that Nurse Joy can pass the item to the player's inventory. This can be done by using the functions inside Actor class. We can use addItemToInventory(Item item) where we can potentially add different element types of PokeFruit or a charmander inside a Pokeball. We also use removeItemToInventory(Item item) to remove the candy amount from the inventory for the corresponding amount of candy for the product the player is trying to spend. This is a good example of Open-Closed principle as the player class functions do not need to be changed if new types of pokefruits and items are introduced to the market by Nurse Joy.fi

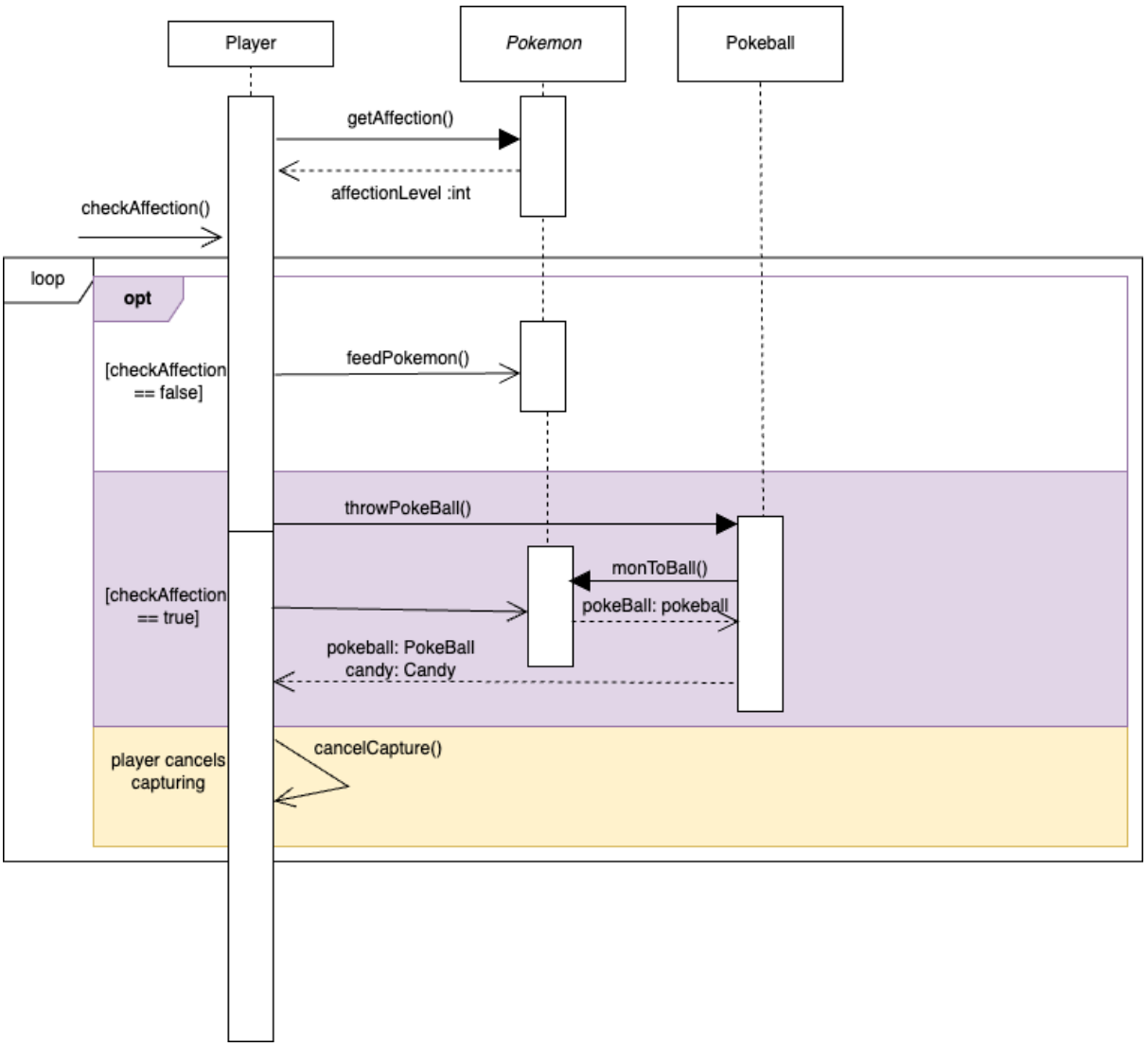
Nurse Joy ---<<gets>>---> Location. In this requirement, it is also known that Nurse Joy will spawn at the middle of the map, therefore it uses the method called containsAnActor() to check whether Nurse Joy is actually in the middle of the Game Map. Therefore It is a dependency between Nurse Joy and Location.

New UML diagrams



Nurse Joy now uses TradeAction where TradeAction would handle the Shop UI as well as take care of the transaction bit of the shop. We have done this as the Nurse Joy would be also using some of the methods in the abstract class of Action,

Interactive Diagram (Sequence Diagram)



Contribution Log.

This is the team's contribution log. Please refer to the link below.

Copy of (TEMPLATE)FIT2099 - CL_Lab3Team4 - Assignments' Co…

https://docs.google.com/spreadsheets/d/1bQrevbRdrbt7pgJxPFDnaVY3A419_dmWZ7rxKWE5RKs/edit?usp=sharing