FIT2099 Assignment 2: Assignment Design Changes made

Group members:

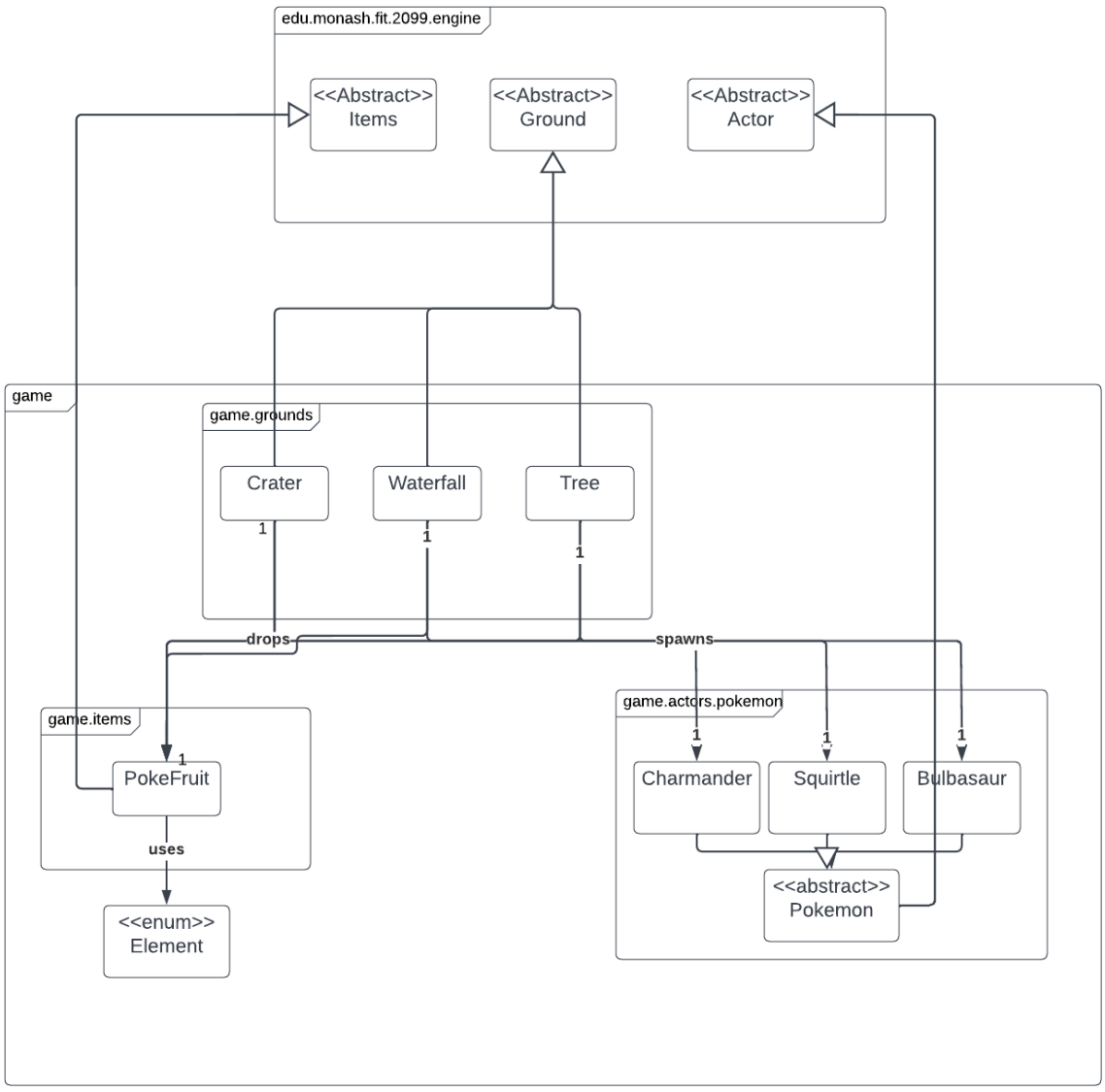
Raunak Koirala

Rohan Pahwa

Aman Jain

Class Diagrams & Sequence Diagrams - Requirements:

Requirement 1 (Updated):



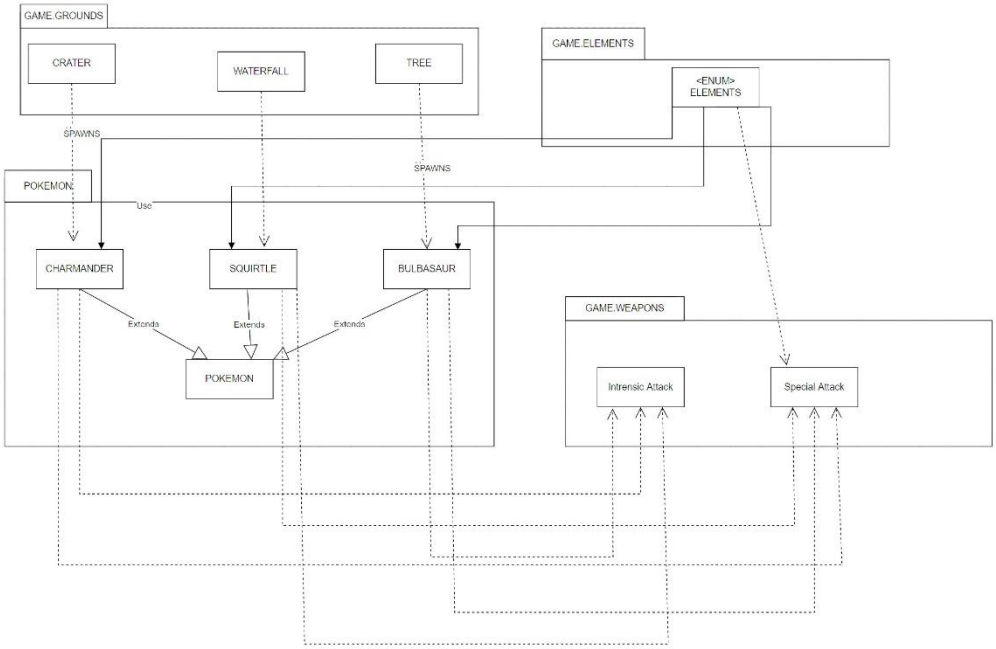
UML Requirement 1 Rationale:

For requirement 1, I decided to make the Crater, Waterfall, and Tree classes all abstract and extend them to Lava, Puddle and Hay classes respectfully in order to make use of the inheritance concept. I also decided to add a class Pokémon in order to have a parent class for all Pokémon as they will all have similar functions. Through using these abstract classes, instead of writing similar code for each class, we can use the code within the abstract class and utilise those methods in the others. This ensures that code isn't repeated and is easier to maintain. The crater, waterfall and tree classes are similarly extended from the ground abstract class as they all have similar functionalities and although they share the same parent class, they still maintain different characteristics thus adhering to the Single Responsibility Principle (SRP) and the use of extending from abstract classes also ensures that the Open-Closed principle isn't violated by improving maintainability and extensivity.

Updates in Assignment 2:

When implementing, we realised that Lava, puddle, and hay didn't necessarily have to inherit the other ground classes and thus instead inherited the ground abstract class like the others and some implemented TimePerception when task 5 was done. There was also a design flaw as we didn't need 3 classes for the 3 elements of pokefruit and instead simply add that element to the constructor using the Enum. Pokémon remains the same and is still used as an abstract class for all Pokémon as keeping code organised and not repeating it (in line with DRY principle).

Requirement 2 (Updated):



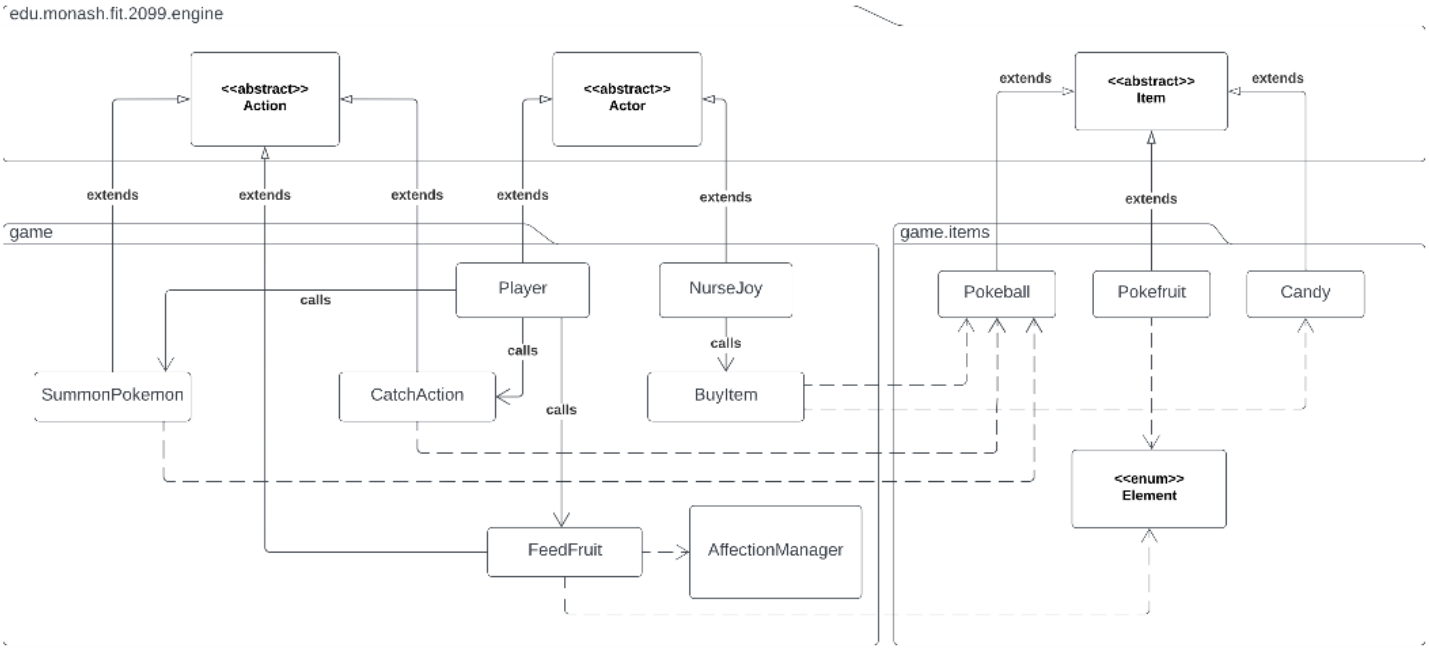
UML Requirement 2 Rationale:

For this requirement, the classes: Charmander, Squirtle, and Bulbasaur inherits all the properties of the Pokémon class. According to the requirement, brief Charmander spawns from Crater ground hence there is a dependency relationship between them, same goes with the waterfall and Squirtle classes and also the Tree and Bulbasaur classes.

The class fire is associated with Charmander because he is fire-type Pokémon and so is Water and Squirtle and Grass and Bulbasaur.

The weapons are dependent on the Pokémon types and the ground they are standing on, as described in the assignment brief. For example, the weapon Ember can only be equipped by Charmander given the condition he is standing on its type of ground i.e., Fire. The same is the case with Bubble and Vine Whip.

Requirement 3 (Updated):



UML Requirement 3 Rationale:

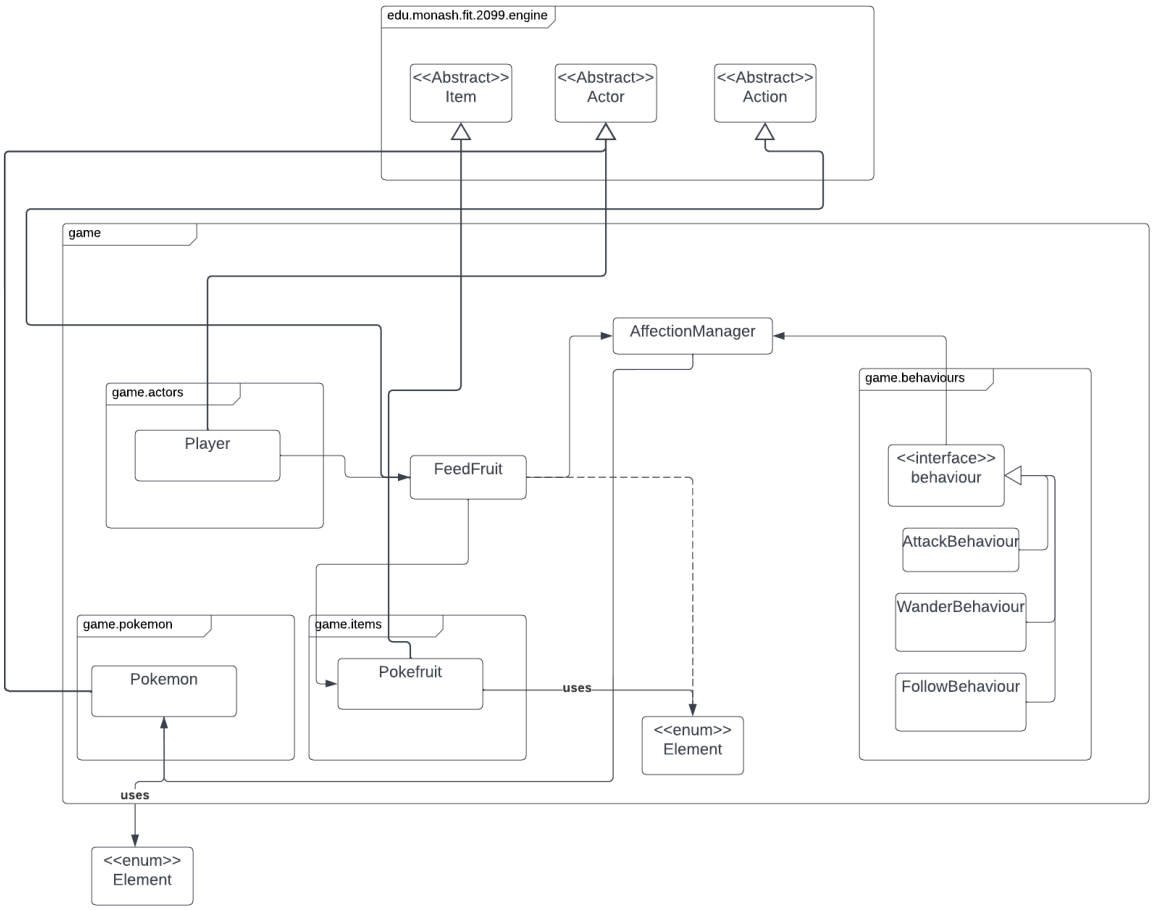
The Requirement 3 UML features most of the existing classes from both the engine and game package. The new classes created are inside the package “game.items” and also the NurseJoy class. The role of the new items classes is to hold the unique and similar properties of each item, and allow the items to be initialised into the game world. The NurseJoy class is responsible for holding the transaction of “candies” in exchange for other items. The new classes in game.items extend the abstract “Item” class, as they share some common properties and the use of repeated code is kept to a minimum. The items also interact with the Player class which extends the abstract “Actor” class. The relationships that exist between the Actor class and game.items classes are there to show that the items have an effect on the Pokémon, who are also Actors. A separate class for Pokémon could be created to make this clearer and help obey the Single Responsibility Principle as currently the Actor class is responsible for many different actors. However, a separate class for Pokémon was not included as this would make the diagram more complex and harder to read. The actions

for dropping and picking up the items are also shown, and the abstract Ground class is included as this will hold the items that are dropped.

Updates to Assignment 2:

The design for requirement 3 remains mostly unchanged, but some new classes have been created which help adhere to SRP. Previously, the Player class was responsible for summoning and catching Pokémon directly. However now the SummonPokemon, CatchAction and FeedFruit classes have been created which the Trainer class can call to perform their respective actions. The PickupItemAction, DropItemAction and the abstract Ground class have all been removed as they served no purpose in our implementation. The NurseJoy has also now delegated some of its responsibilities to the new BuyItem class, where the trading of items and Pokemon (Pokeballs) takes place.

Requirement 4 (Updated):


edu.monash.fit.2099.engine
<<Abstract>>
Item
<<Abstract>>
Actor
<<Abstract>>
Action
game
AffectionManager
game.actors
Player
FeedFruit
game.behaviours
<<interface>>
behaviour
AttackBehaviour
WanderBehaviour
FollowBehaviour
game.pokemon
Pokemon
game.items
Pokefruit
uses
<<enum>>
Element
uses
<<enum>>
Element

UML Requirement 4 Rationale:

This requirement is about using the Pokémon's affection for the player in order to determine if the Pokémon can/cannot be captured and if it will follow the player around (behaviours of the actor). The requirement also determines if the Pokémon's affection increases, or decreases based on fruits given to them and their element. This affection point system is managed by the Affection Manager class which would store the Pokémon with their respective affection points. Furthermore, the status Enum is also utilised in this design as this makes it easy to know which Pokémon are catchable and which are not and can be changed based off affection points. This Enum ensures that code doesn't use an excessive number of literals and follows the open-closed principle by keeping the code much more concise and making it, so we won't have to code a various range of conditional for checks.

We also create a ConsumePokeFruit action class that acts as a subclass for Action and is used to program what is done if the poke fruit is consumed by Pokémon. Instead of consuming the poke fruit directly in the Items subclass, we regard the intended usage of the engine and follow the single responsibility principle in this scenario by reflecting that most of the functionalities are executed after the poke fruit is given to the Pokémon.

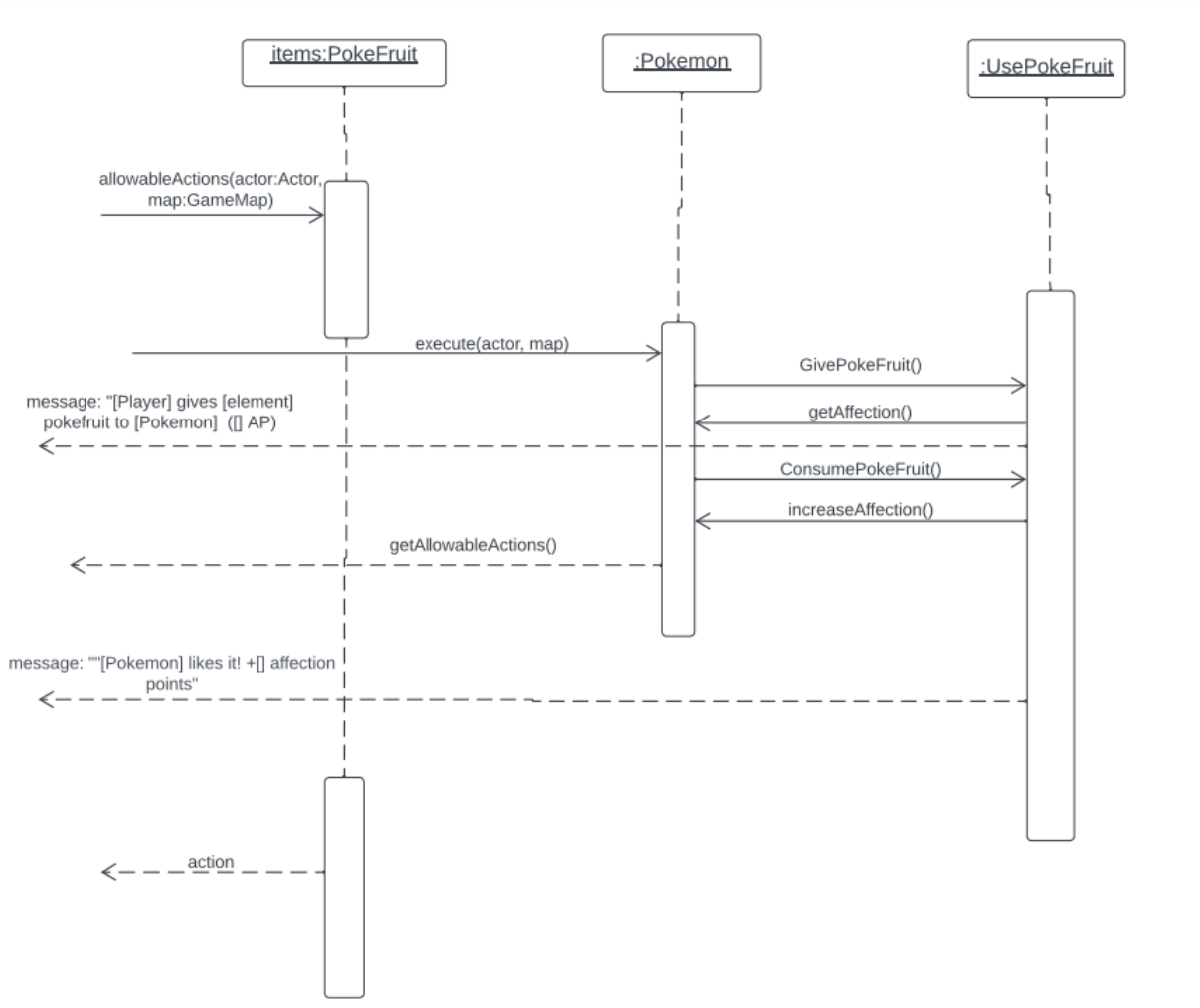
Updates in Assignment 2:

Removed some unnecessary links from the diagram and renamed some packages. Also renamed action to feed Pokémon as FeedFruit instead of ConsumePokeFruit where the player feeds the Pokémon instead of Pokémon consuming fruit, as used within the code.

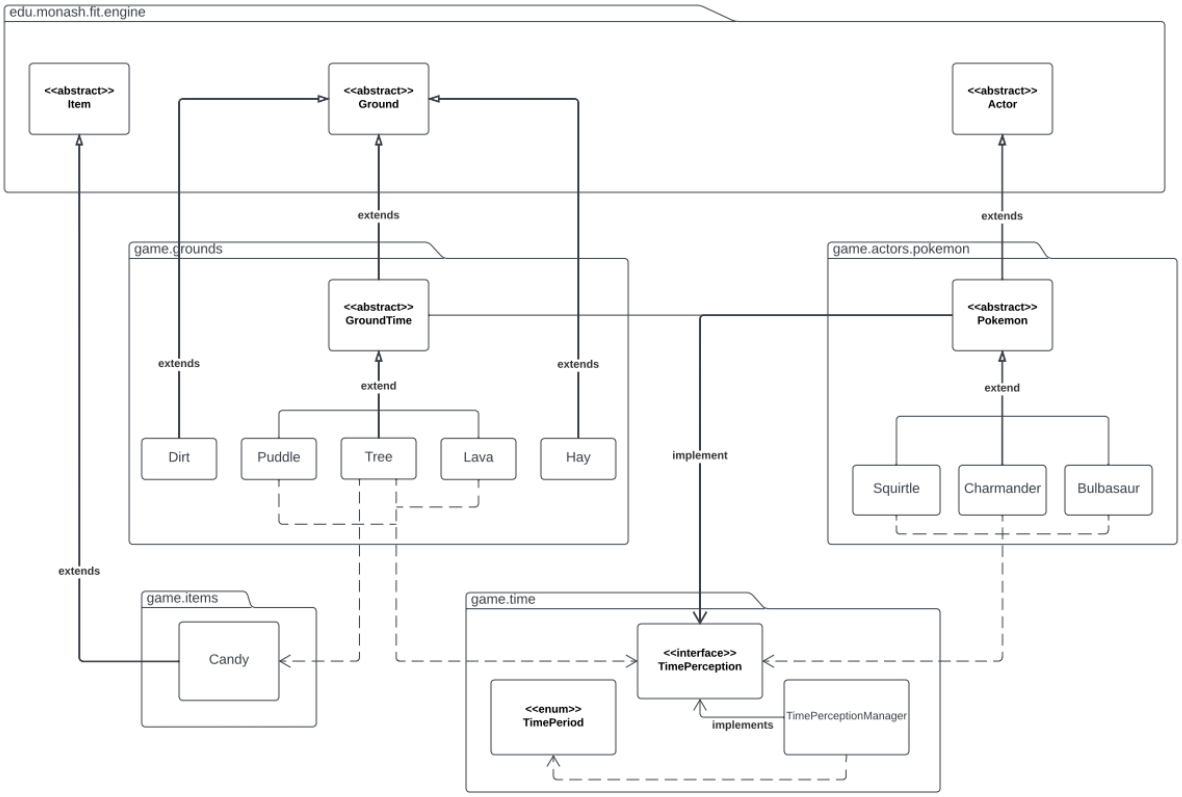
The sequence diagram below follows the same process utilised however it checks if player has pokefruit in the inventory before attempting to feed the pokemon.

The given Sequence Diagram outlines the process of a ConsumePokeFruit by a Pokémon to increase affection:

Interactive Diagram:



Requirement 5 (Updated):



UML Requirement 5 Rationale:

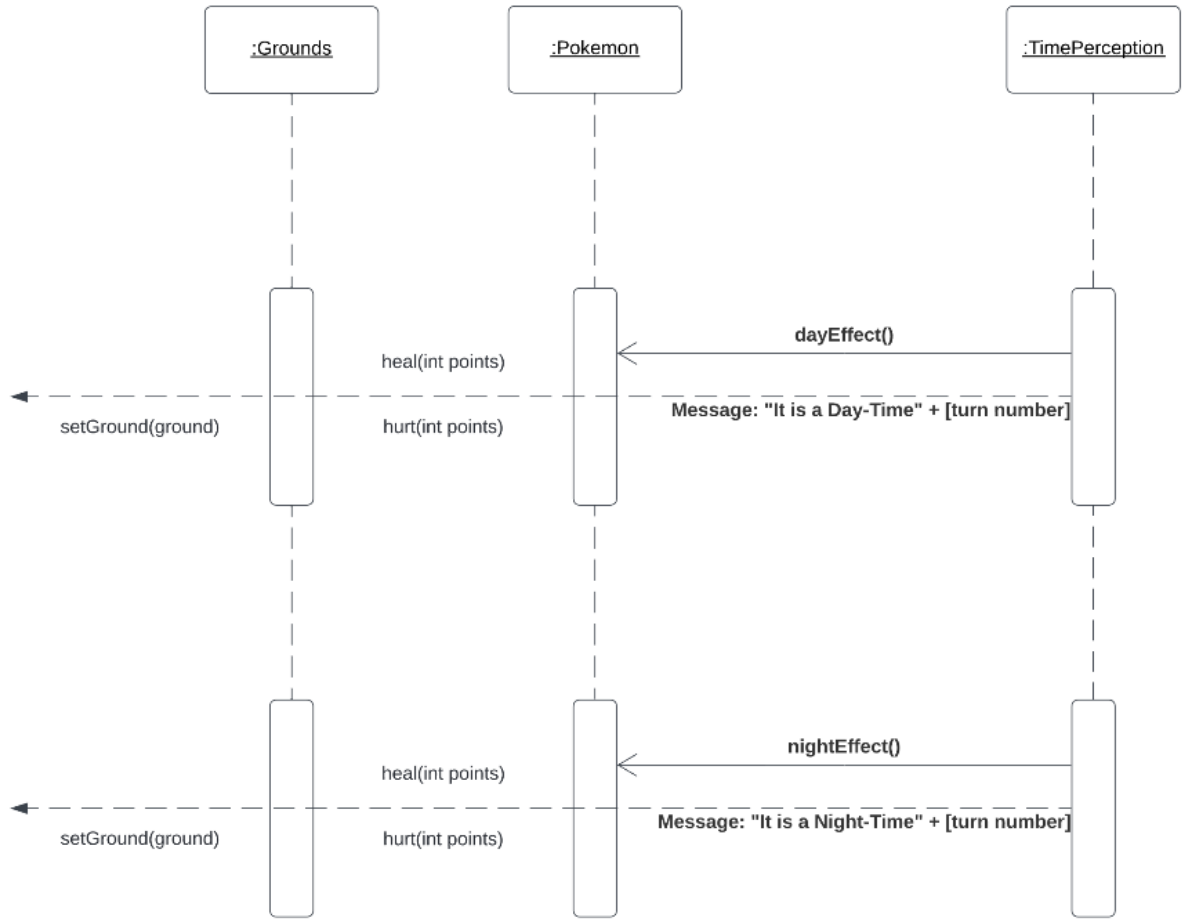
This diagram represents how the classes inside the game.grounds package and the new game.actors.pokemon classes perceive time and as a result experience changes. The classes in the game.grounds package based on what they perceive from the TimePerception interface, (which determines the time of day using the other two game.time classes) convert themselves into their respective ground type using the FancyGroundFactory class, which creates the new Ground objects using the abstract Ground class. The Ground class is made abstract to avoid the repetition of code (DRY). The Tree class also has a chance of dropping a candy, which it does so using the DropItemAction class. The Pokemon which extend the Actor class also perceive time from the TimePerception interface, which would then apply the statuses of either restoring their hit points or taking damage. It was decided the classes

that apply these statuses where not included in this diagram as these behaviours and actions are complex enough to require their own UML diagram.

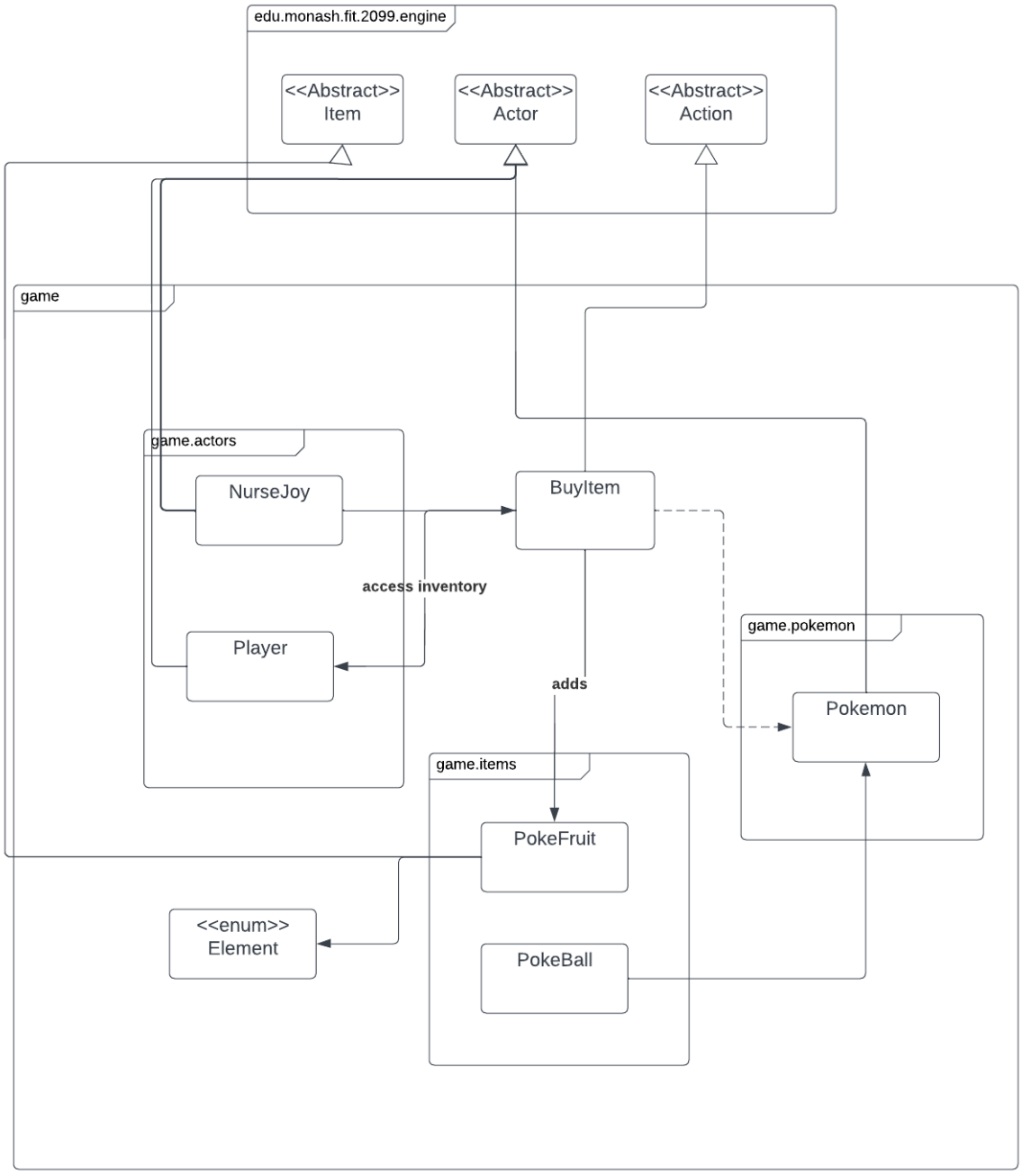
Updates to Assignment 2:

The updated design for assignment 2 remains mostly unchanged, however a few tweaks have been made to eliminate instances of DRY and make the code generally cleaner. The abstract Pokemon class has been created so that not all Pokemon have to directly implement TimePerception, but can implement and perceive time through the Pokemon class which they extend. Another new abstract class created is GroundTime, which serves the same purpose as the Pokemon class but obviously the GroundTime class allows the different grounds to perceive and implement TimePerception. The interface GroundFactory and class FancyGroundFactory have been removed from the updated design, as they served no purpose in the creation of new grounds. The DropItemAction has also been removed, as it was unnecessary in dropping the candy.

Interactive Diagram (Updated):



Requirement 6 (Updated):



UML Requirement 6 Rationale:

In this requirement, the class NurseJoy depends on the buyer i.e., the player that whether they want to swap items for candies. The nurseJoy inherits candyTrade that depends on inventory for the number of candies and also associates classes Pokefruit and Charmander for a trade. The Enum class Pokefruit consists of water pokefruit, fire Pokefruit, and grass pokefruit, for an eligible exchange.

Items is an abstract class and class Charmander and PokeFruit inherits this class.

Charmander is a Pokémon, so it inherits all its properties from the Pokémon class.

Updates in Assignment 2:

Instead of using an Enum for pokefruit, we simply used the element Enum in the constructor for the pokefruit and used this when buying different pokefruit types. Furthermore, our design now utilises the PokeBall which takes in a pokemon within its constructor in order to store them within the item. This is used when trading with NurseJoy and buying a Charmander within a PokeBall. The BuyItem action is created in our code in order to provide an action for NurseJoy in order for the Player to interact and buy from them. This new action also helps us display the menu description and provide player with the option to trade.

The new interactive diagram goes through the BuyItem action implemented. Action invoked through NurseJoy, it uses the amount of candy stored in the players inventory to determine whether or not the player can afford the poke fruit or the Pokémon. If they can afford it, It ensures that the item bought is added to their inventory and that the cost of the item, and therefore the number of candies in the inventory, is removed accordingly. If the player cannot afford the candy, it lets them know with a message as well.

Interactive Diagram(Updated):