MONASH University

FIT2099 G2 Revised Design Document

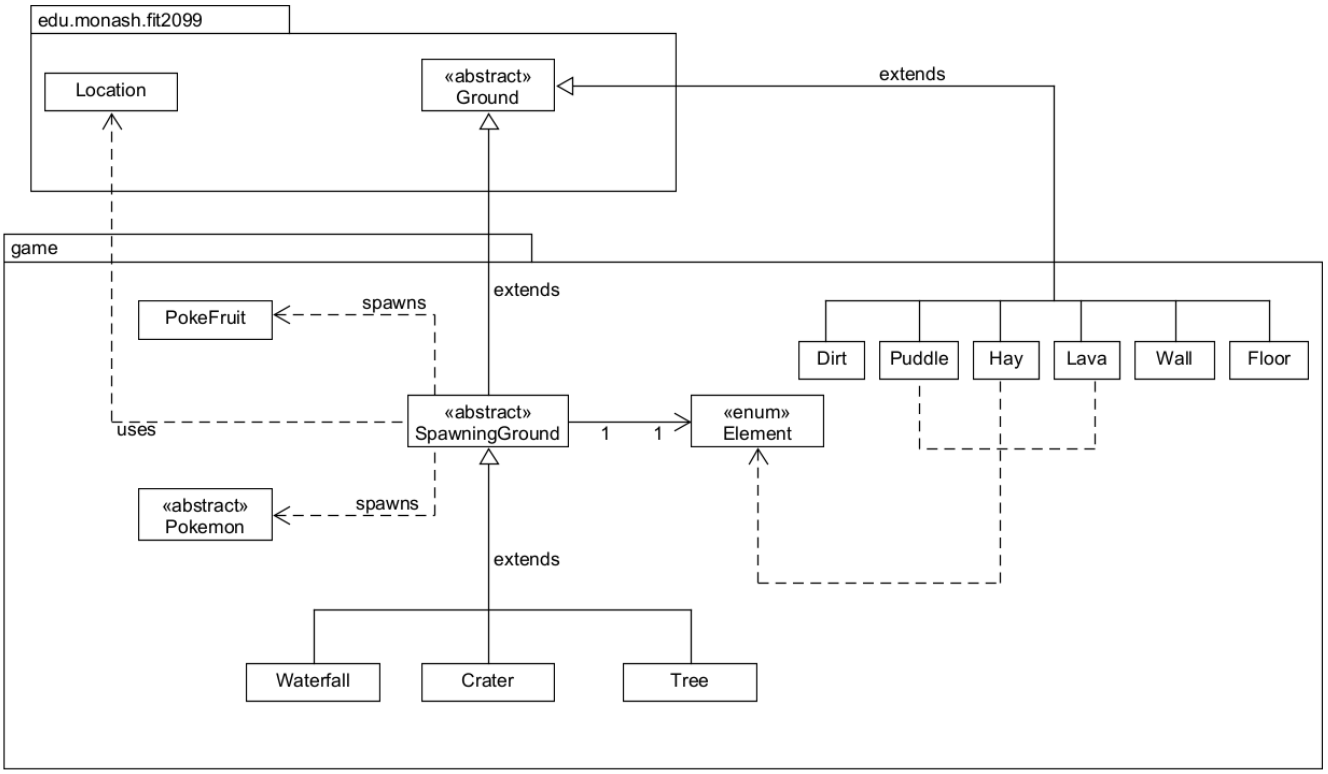
Team Members:

Shyam Kamalesh Borkar (32801459)
Eng Lim Ooi (30720680)
Arrtish Suthan (32896786)

CONTENTS

REQ1: Environment

UML Class Diagram



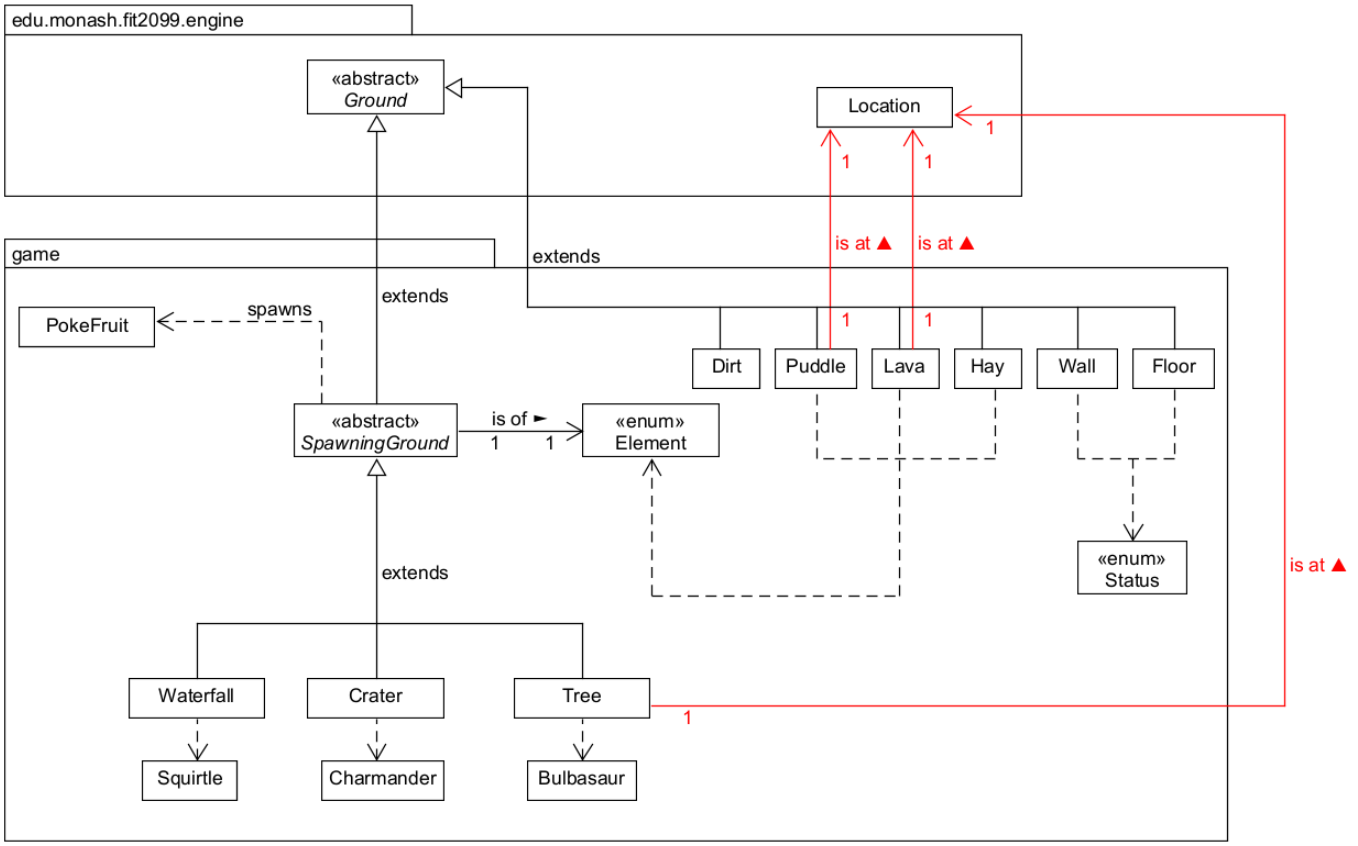
Design Rationale

In this section, the design of the environment system in the game will be explained and the reasoning for the designs too. The three concrete classes Waterfall, Crater and Tree all extend from the abstract class called SpawningGround. This abstract class in turn inherits from the abstract ground class from the provided game engine. Creating this new SpawningGround abstracts the common features of spawning the pokemons and pokefruits according to their element type into one class that the child classes like Waterfall, Crater and Tree can inherit to perform the spawning features without repetition. Hence, the dependency of SpawningGround on pokemons and pokefruits. This means that this design conforms with the DRY (Do Not Repeat Yourself) principle as it avoids repetition of common functionality. The SpawningGround class is dependent on location as it uses it to carry out its spawning features. Since the spawning is to be done at every turn the code for this feature will be embedded into the tick method that allows the common game objects to experience time.

The SpawningGround class has an attribute of a single enumeration Element so that each of its child classes have a single element that it supports as mentioned in the requirements. Using this element type attribute of the spawning grounds, a check can be made to investigate the surrounding grounds to check if they are of the same type as the SpawningGround and as a result the decision of whether a spawn is done or not is made.

The rest of the grounds (Dirt, Puddle, Hay, Lava, Wall and Floor) directly inherit from the abstract Ground class. However, the concrete classes Hay, Lava and Puddle are dependent on the enumeration Element as they will add this capability to associate their objects with the correct element type.

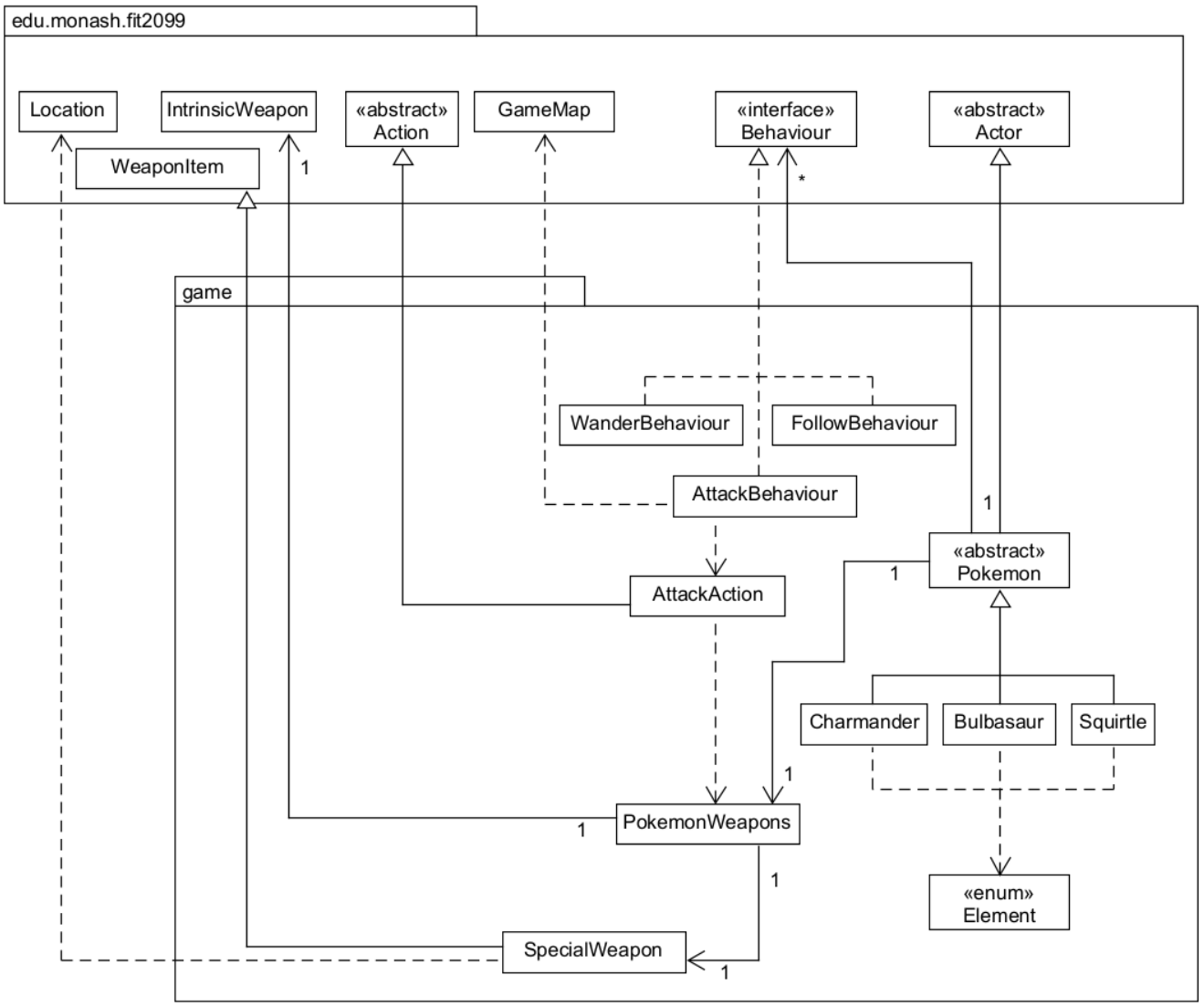
REQ 1 Revised Version



Changes: Each ground that inherits from the spawning ground is dependent on the respective pokemons that they are going to spawn.Hence, the dependency on the particular pokemons. Wall and floor now depend on Status enumeration to identify that it cannot be expanded by getting another ground setting over it. Tree, Puddle and Lava have an attribute of Location type. This attribute stores the location at which the ground is situated. This is necessary as the day and night effect methods of these grounds required checking the location they are at.

REQ2: Pokemon

UML Class Diagram



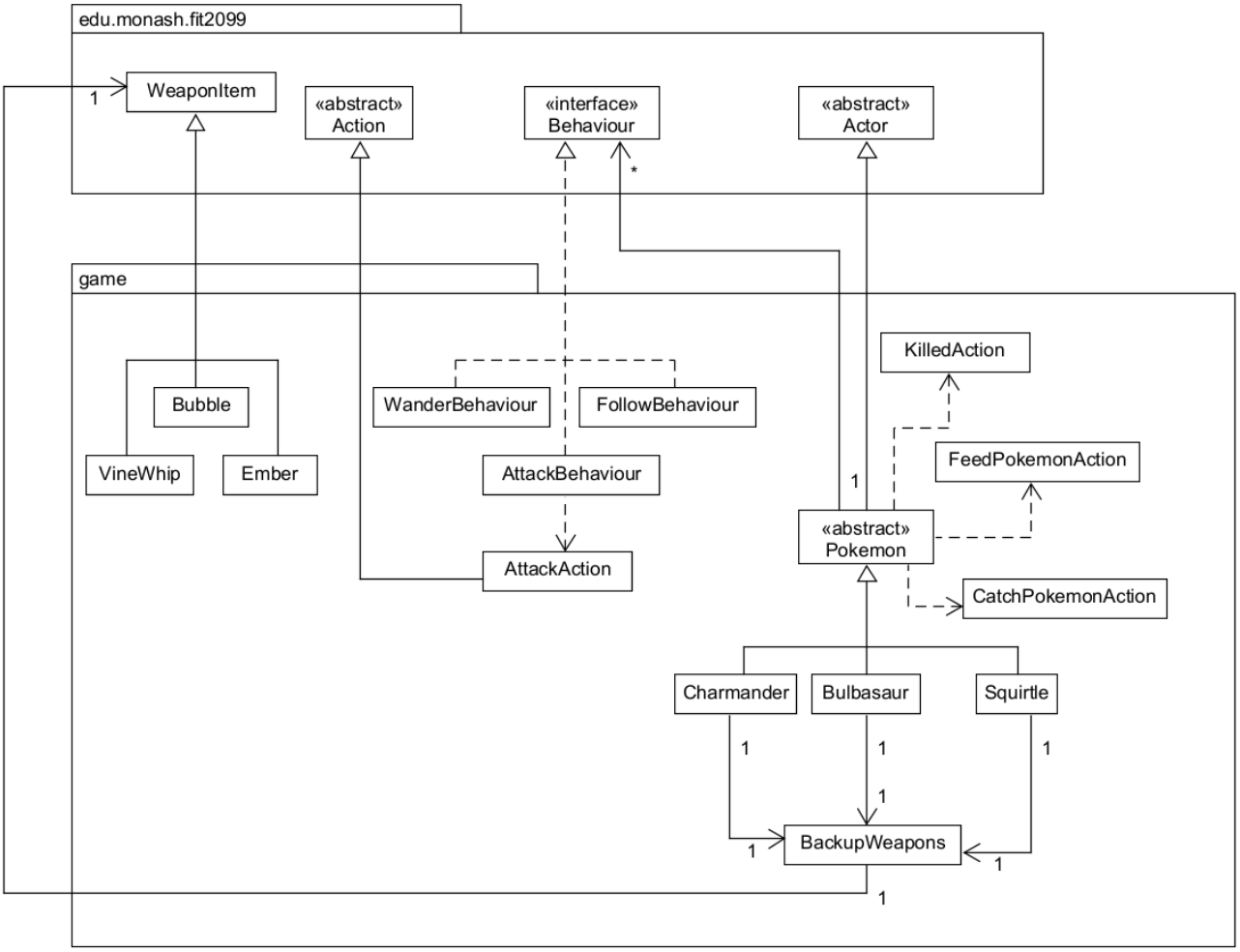
Design Rationale

This requirement section involves the pokemon and all of the information and actions that they possess. Primarily, using the DRY principle, an abstract Pokemon class is created in the game files in order for us to create pokemon involving the Pokemon's

attributes and methods in other specific Pokemon classes. This allows code without having to duplicate the code in three different classes. Bulbasaur, Squirtle and Charmander are classes that inherit the attributes and method of the main Pokemon class directly, whilst also utilising the element enumeration that denotes the Pokemon element type. The Pokemon class also has a catchable attribute which denotes whether a Pokemon can be caught.

The Pokemon class also has a list of behaviours of the behaviour interface, which possesses all of the methods involving the pokemon's behaviours (wandering, following and attacking). The WanderBehaviour and FollowBehaviour possess the methods from the behaviour interface and allow the pokemon to use those methods. The AttackBehaviour depends upon the AttackAction class, which will check through the PokemonWeapon class. The PokemonWeapon class has two attributes one of which is SpecialWeapon and IntrinsicWeapon. This will allow all the pokemons to carry out their respective intrinsic and special attacks. Selection of a special weapon object occurs when the requirements for the special attack are met, or otherwise an IntrinsicWeapon will be returned.

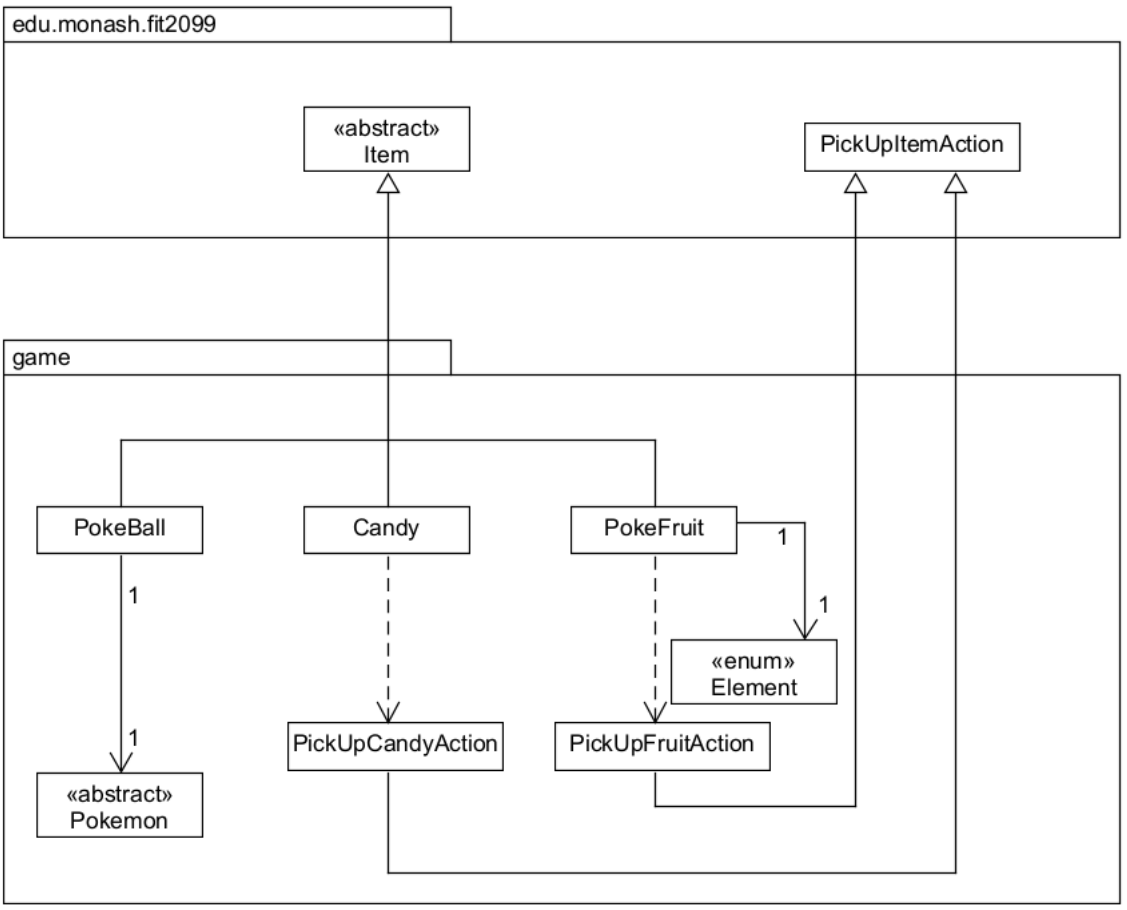
REQ 2 Revised Version



Changes: The special weapon of each of the three pokemon now has a dedicated class to it that inherits from the weaponItem class in the game engine. Each pokemon has an attribute of BackupWeapons class. This class will store the special weapon of each of the pokemon which will be removed and added to the pokemons inventory using the toggle weapon method. Each pokemon is dependent on the FeedPokemonAction and CatchPokemonAction as it returns it as allowable actions to the player if conditions are met. The KilledAction has been added so that the pokemons can return this action once it is removed from the map as a result of being killed so that a NullPointerException is not raised while the game is being played. SpecialWeapon and PokemonWeapons from the previous UML did not seem feasible as the team intended to use the BackupWeapons class that was provided with the starting files.

REQ3: Items

UML Class Diagram



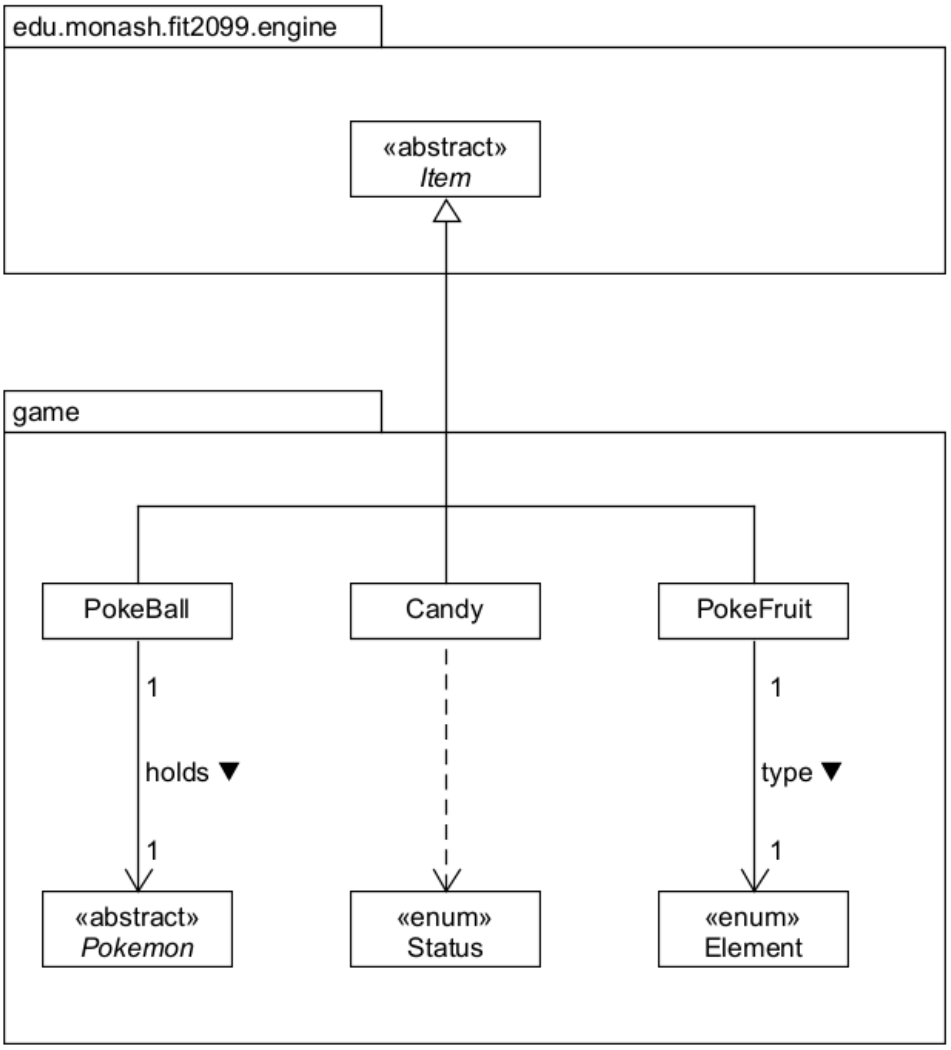
Design Rationale

In this game there are three items in total which are Pokeball, Candy and Pokefruit. Since these are items that the player can hold in its inventory these concrete classes extend the functionality of the Item abstract class that is available in the game engine. By doing this this design is following the DRY (Do Not Repeat Yourself) principle since the items in the pokemon game now have the common attributes and functionality of an item in the game. The item pokeball has its own unique attribute which is of a single pokemon type. This is a one-to-one relationship as it is known that the pokeball can only have one pokemon captured inside it.

The pokefruit class has an attribute of type enumeration Element. This will allow the pokefruit to be identified as a particular element type when implemented in the game such as Fire pokefruit. This element can be set using an argument to the constructor of this class whenever the pokefruit is created by its spawning ground and created as part of a trade under Nurse Joy. The candy is a simple class that has its own

pickup candy action like the pokefruit class does and these action classes inherit the PickUpItemAction class from the game engine. Hence, again complying with the DRY principle. The remaining actions that the player can do that are related to these items are discussed in the next section/requirement.

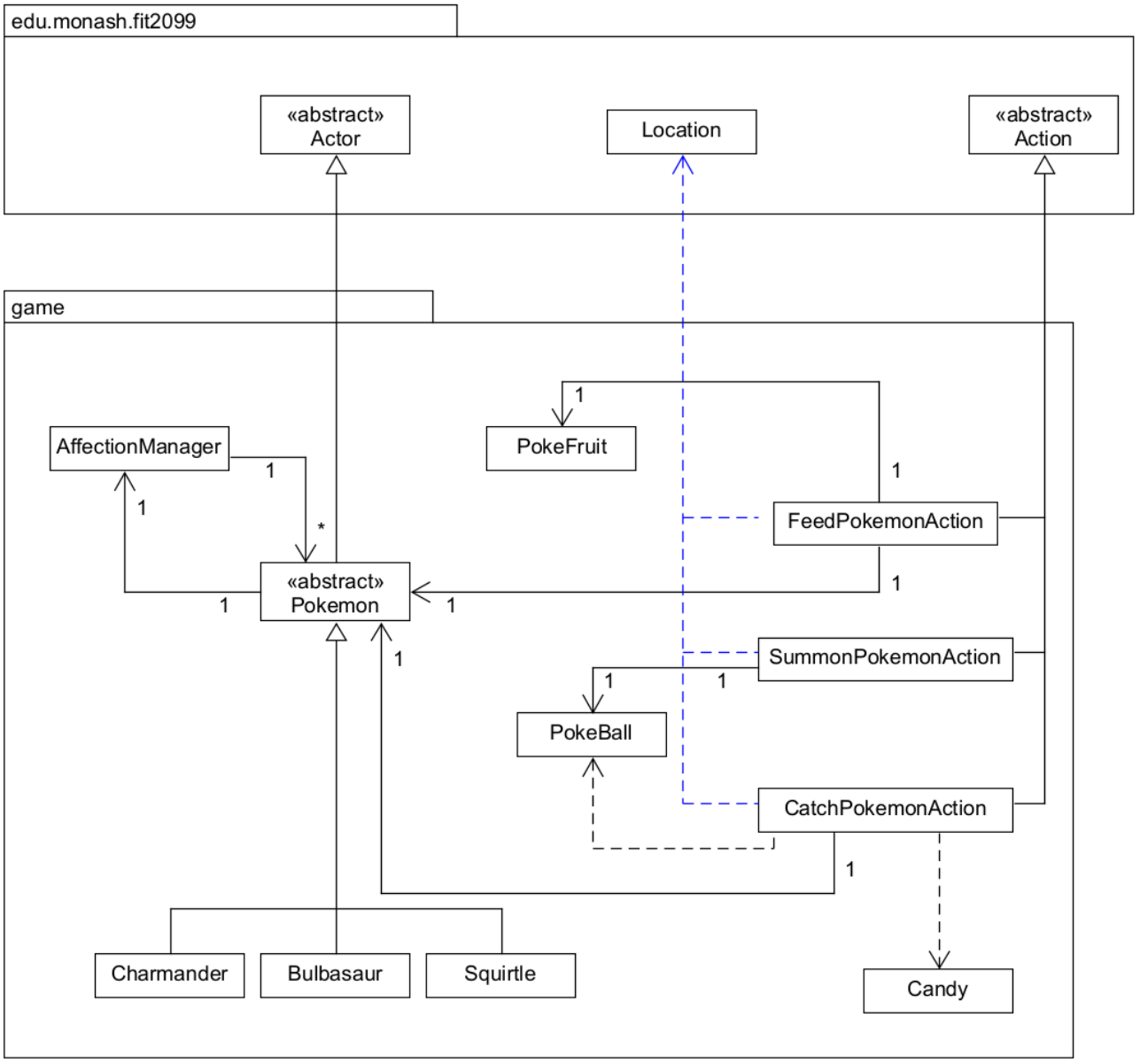
REQ 3 Revised Version



Changes: There is no need for the PickUpFruitAction and the PickUpCandyAction and hence as a result the classes were not implemented and were removed from the UML. These pickup actions are taken care of by the game engine if the item is portable. This applies to the drop item actions as well. The candy is dependent on the Status enumeration as this enum is added to the Candies capability to identify if an item in the player's inventory is actually a candy or not.

REQ4: Interactions

UML Class Diagram



Design Rationale

This requirement includes the Pokemon interactions possible for the player, including feeding, summoning and catching Pokemon. Since the AffectionManager class can only create one object of itself (singleton class), this object must be passed through the respective constructors so that it is available for the pokemon to use. Hence, the pokemon class has an association with the AffectionManger class and all the

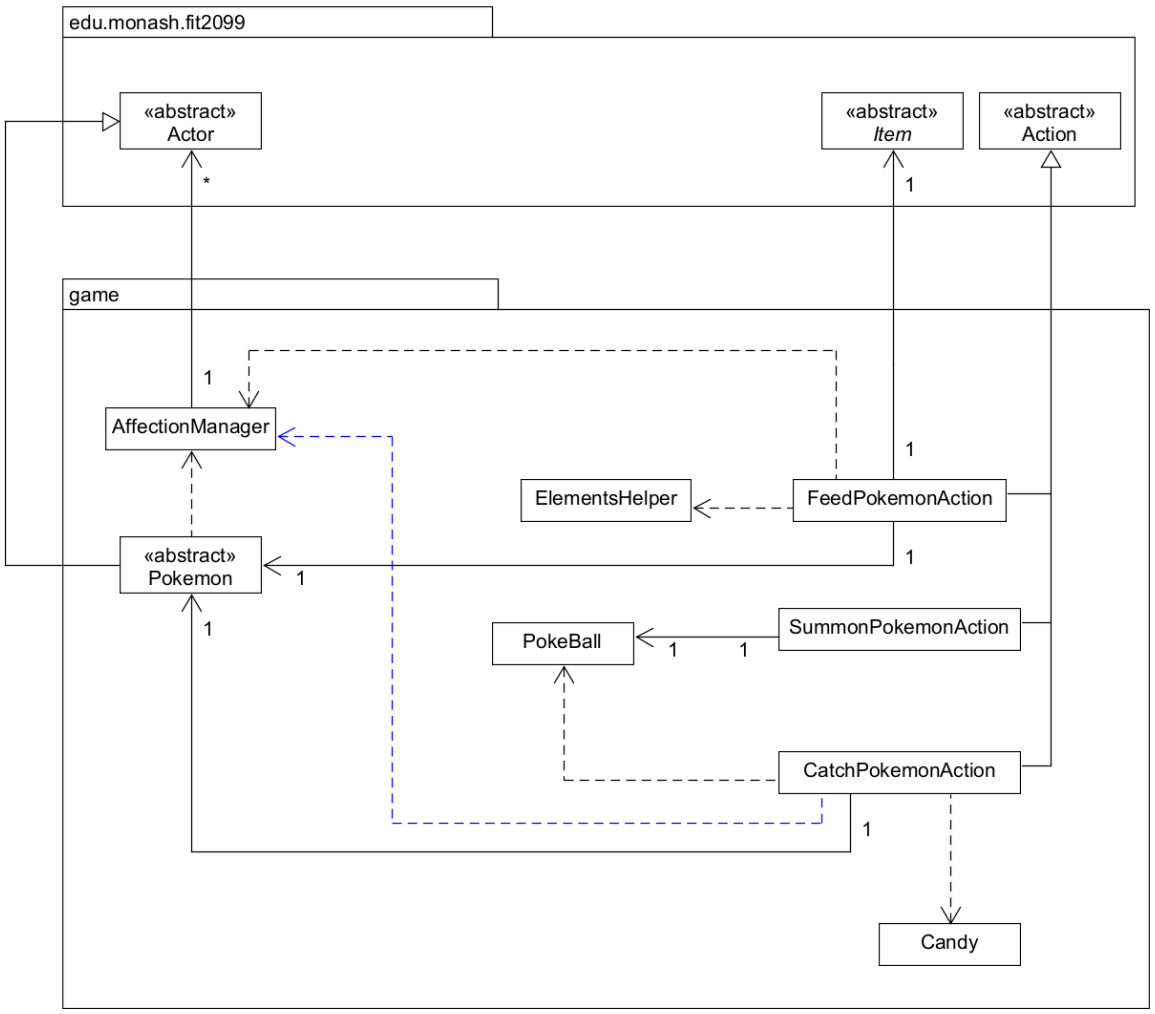
pokemons will have access to the same AffectionManager object keeping all the data relating to the affection points of the pokemon intact. The affection manager will have all the methods that handle the affection points of the pokemon.

The FeedPokemon action, SummonPokemonAction and the CatchPokemonAction all inherit from the abstract Action class as it has all the relevant properties to carry out the intended functionality for the actions the player needs to perform. The FeedPokemonAction will have an attribute of pokefruit that is obtained from the player's inventory and the element of this pokefruit is tested with the element of the pokemon the actor is trying to feed. Depending on the result of this test the affection points of the pokemon will either increase or decrease.

The SummonPokemonAction has an attribute of the Pokeball from the player’s inventory that will hold the pokemon that needs to be summoned onto the game map.

The CatchPokemonAction will be dependent on the pokeball class and the candy class. This is because an object of the pokeball is created only when the affection points of the pokemon is satisfactory for it to be caught and if it is able to be caught a candy item object will be created and placed at the location at which the caught pokemon was on.

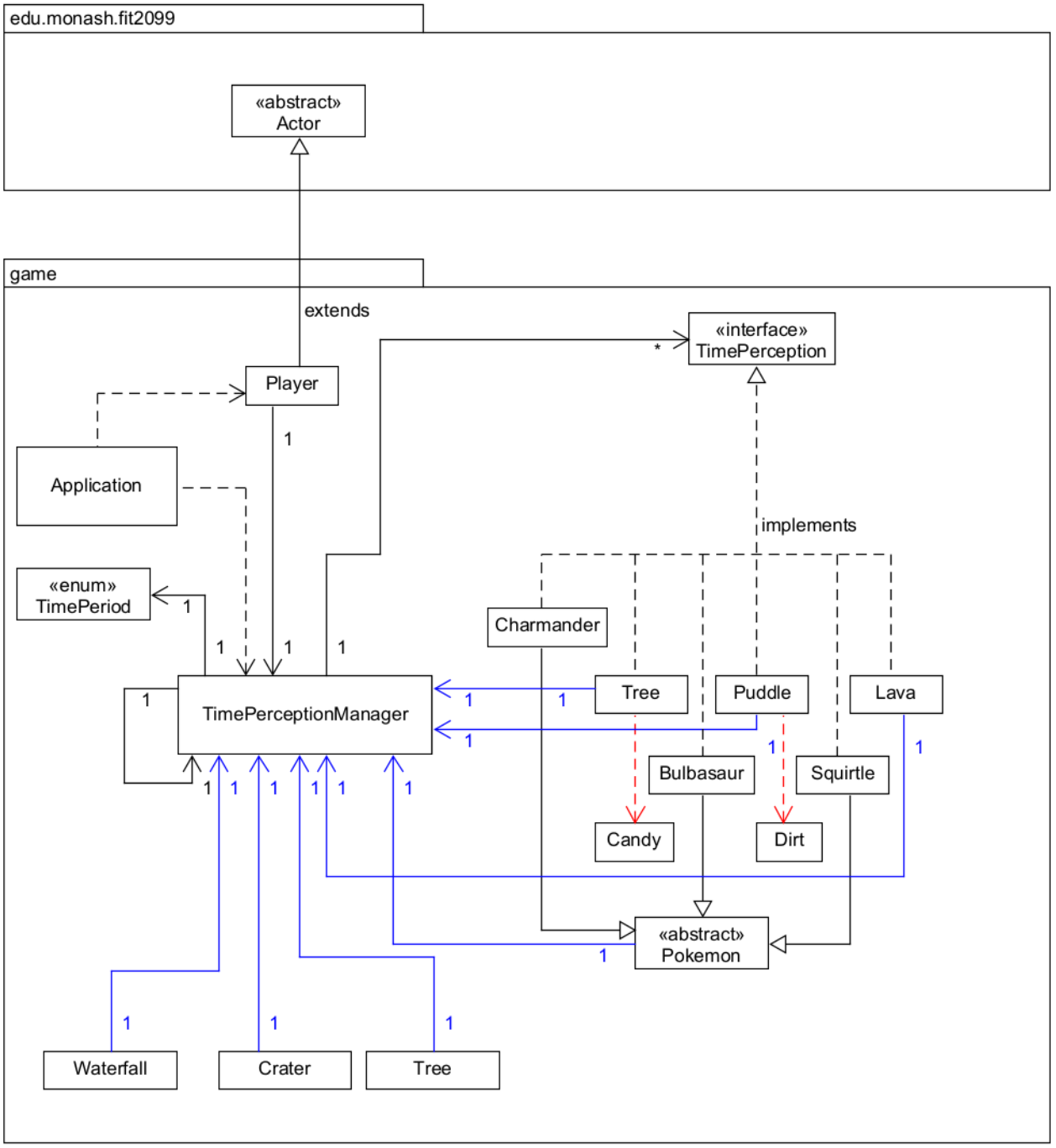
REQ 4 Revised Version



Changes: Each pokemon is dependent on the affection manager class as they are required to register themselves as an instance. The FeedPokemonAction is also dependent on the affection manager as it will use it to increase and decrease the pokemon's affection points respectively. The same applies to CatchPokemonAction. It uses the affection manager to change affection points when a catch is not successful. FeedPokemonActoin has an item as an attribute which is essentially the pokefruit item. It also has a pokemon attribute which is the pokemon that will be fed the pokefruit. It then uses the ElementsHelper class to compare the elements of the Pokemon and the Pokefruit. Hence, the dependency on the class.

REQ5: Day/Night

UML Class Diagram

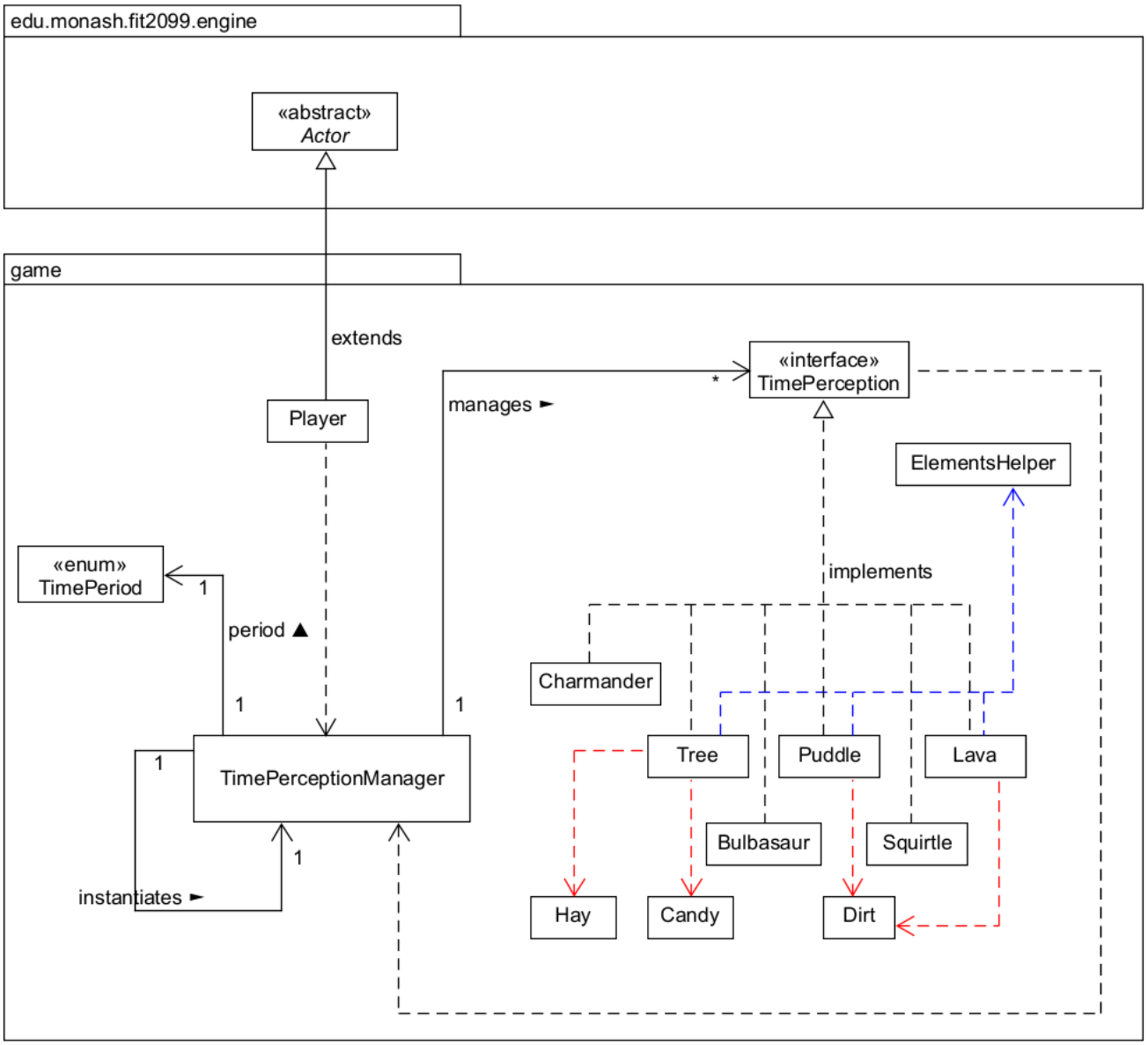


Design Rationale

Our day and night design requirement uses all the files given in the time perception package of the game files. Since only one object of the TimePerceptionManager can be created at a time (singleton class) this single object must be available to all the classes that have a relationship and some functionality with perception of time. In the Application class itself an object of the TimePerceptionManager will be created using the getInstance method and this object will be passed into the constructor of the relevant classes as it can be seen from the multiple one-to-one association relationships in the UML class diagram. The concrete classes that actually have time perception functionality implement the TimePerception interface to have all the relative abstract functions for the day and night effects that each of these classes can implement in their unique way according to the requirements. In each of these classes that implement the TimePerception interface, their instance of Time Perception Manager that points to the same single object will be used to add themselves (append method) into the list of objects that implement Time Perception. This design adheres to the Liskov Substitution Principle (LSP) as we are able to get each class that uses the Time Perception interface to make use of the correct methods (day and night effect)and its implementation.

From the diagram it can be seen that the player class also has an attribute of type Time Perception Manager. In the playturn method of the player the methods of the TimePerceptionManager will be called to make the relevant changes such as increment the turn and shift from day to night and vice versa (run method of TimePerceptionManger). Additionally the message indicating the turn and time period will be printed using the single TimePerceptionManager object in the playturn method. Also a method that iterates all the objects that implement the interface and call the time effect methods accordingly will be called in the playturn method. This avoids having to include this functionality in all the relevant time perception classes and hence does not violate the DRY principle.

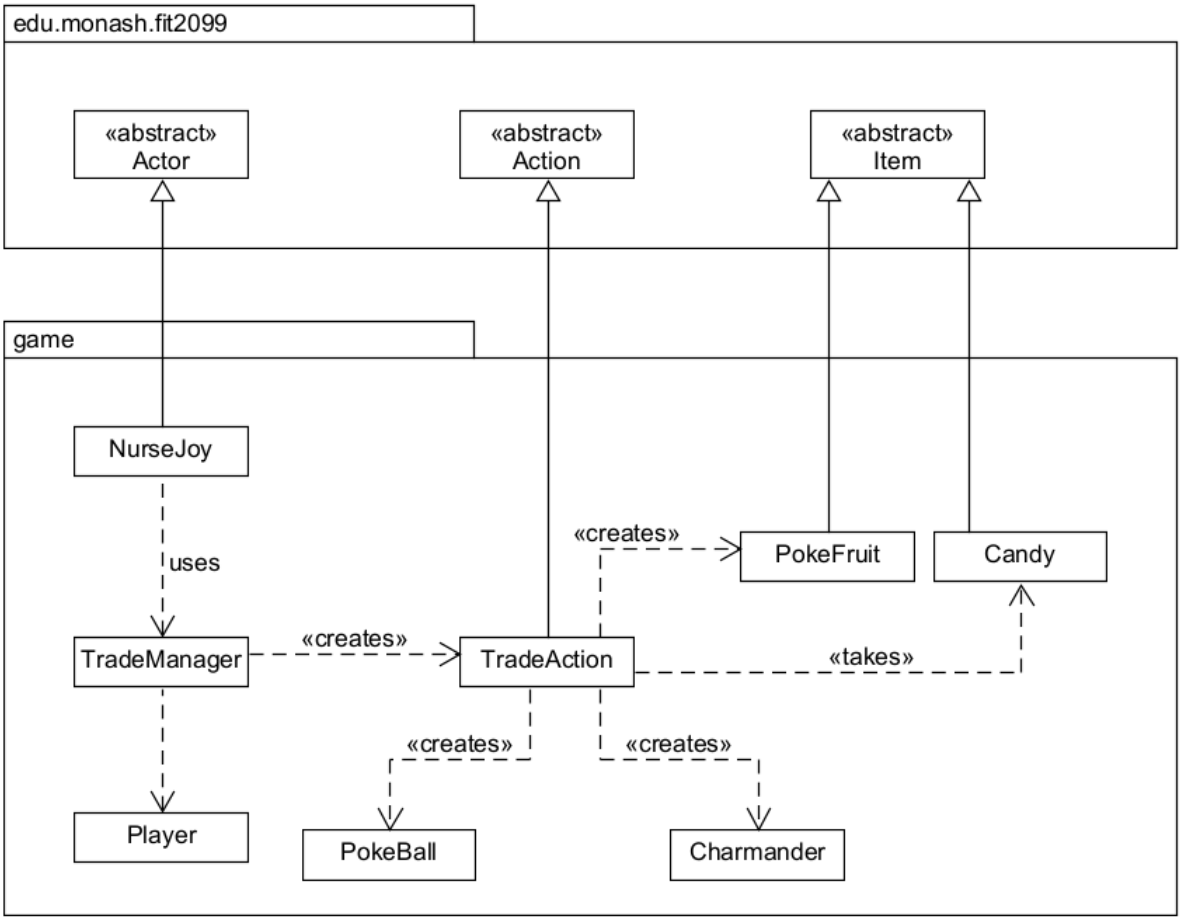
REQ 5 Revised Version



Changes: Thought process was wrong in the previous UML when using the singleton TimePerceptionManger class. Player is dependent on the TimePerceptionManger as it uses it to run all the TimePerception objects. Tree, Puddle and Lava are dependent on the ElementsHelper class to compare elements with surrounding grounds to carry out day and night effects appropriately. The Tree class is dependent on the Hay class as it might use it to carry out expanding features from the time perception.

REQ6: Nurse Joy

UML Class Diagram

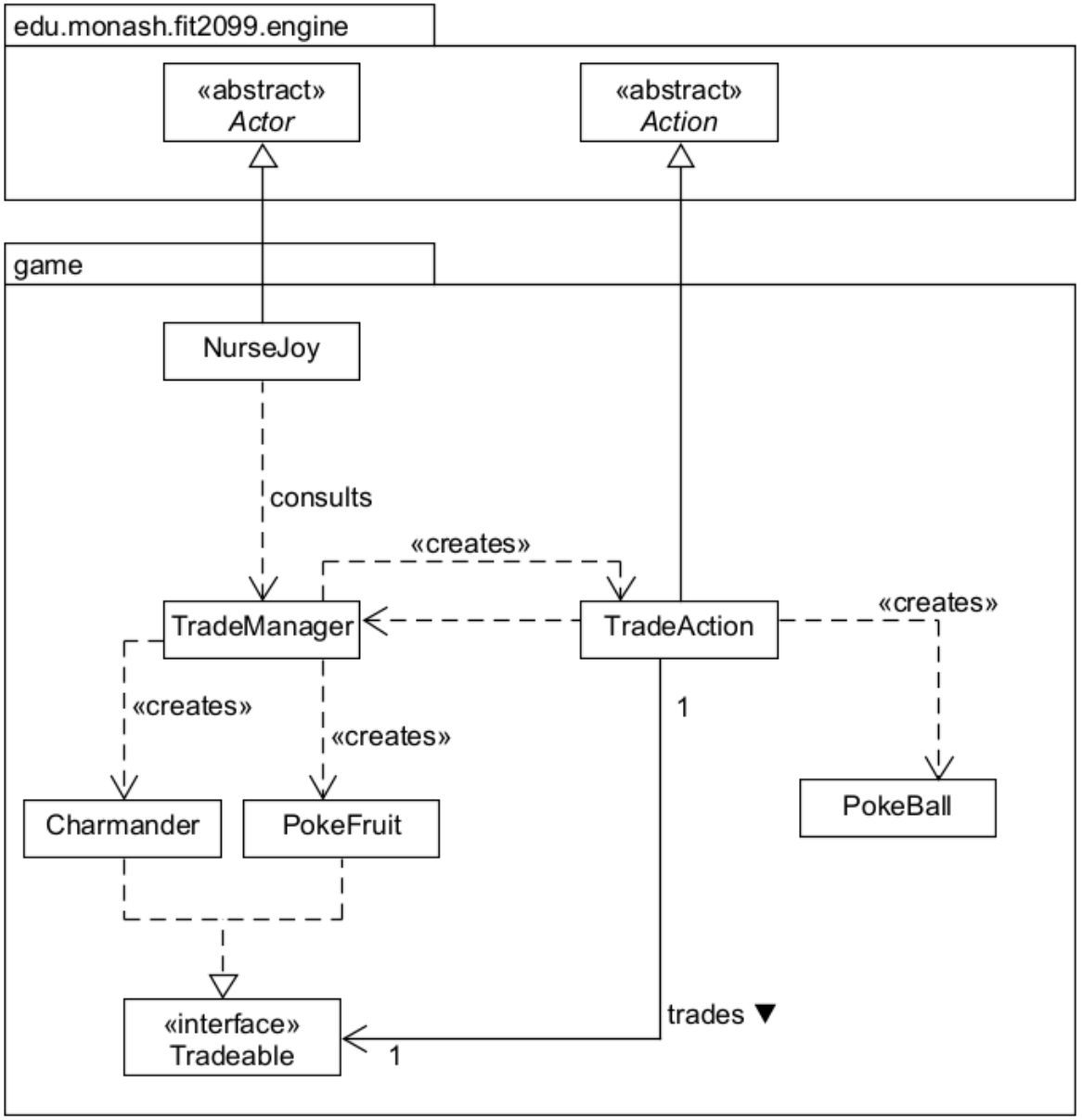


Design Rationale

This requirement specifies the existence of a nurse joy in the game. Since nurse joy is another character like the player or more accurately an non playable character (NPC), it will extend the abstract actor class and share its methods and attributes. This allows nurse joy to interact with the player of the game and offer the trading functionality to the player in the form of a list of actions using the available method called allowableActions. The playTurn method of the nurse joy class will override the same method in the parent class and return a DoNothingAction as the nurse does not have to do anything other than provide possible trades for the player. A trade action concrete class has been created to provide a trade action that the player can use on nurse joy to get the relevant items like the charmander and the poke fruits as a result of the trade.

The trade action class inherits from the action class as it is able to provide a menu description for the player to select the trade to perform and an execute function that produces a message when the player has carried out a trade action. For the design to resonate with the single responsibility principle a new concrete class called TradeManager is added. This class completely takes over the task of deciding the trade actions that the player is capable of doing by checking the amount of candies that the player possesses in its inventory. After this the TradeManager returns to the nurse joy a list of all the possible trades that nurse joy returns in its allowableActions method in the form of a list of trade actions. Creating this additional class prevents the NurseJoy class from developing into a GOD class with a lot of responsibilities. It might also be so that in the future Nurse Joy will have to provide more functionality to the player and hence, keeping the single responsibility principle in mind a new concrete class can be created to handle the new responsibilities.

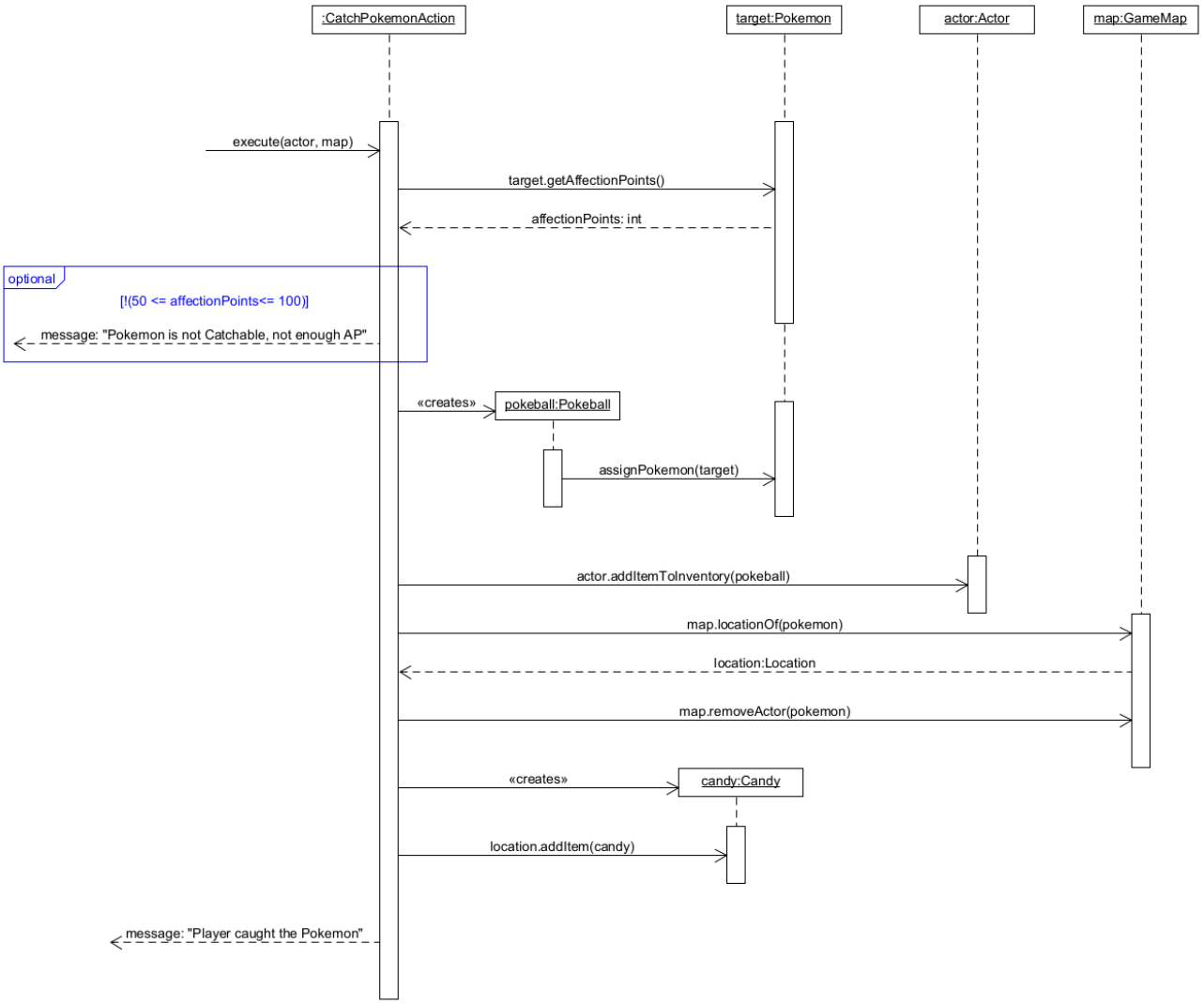
REQ 6 Revised Version



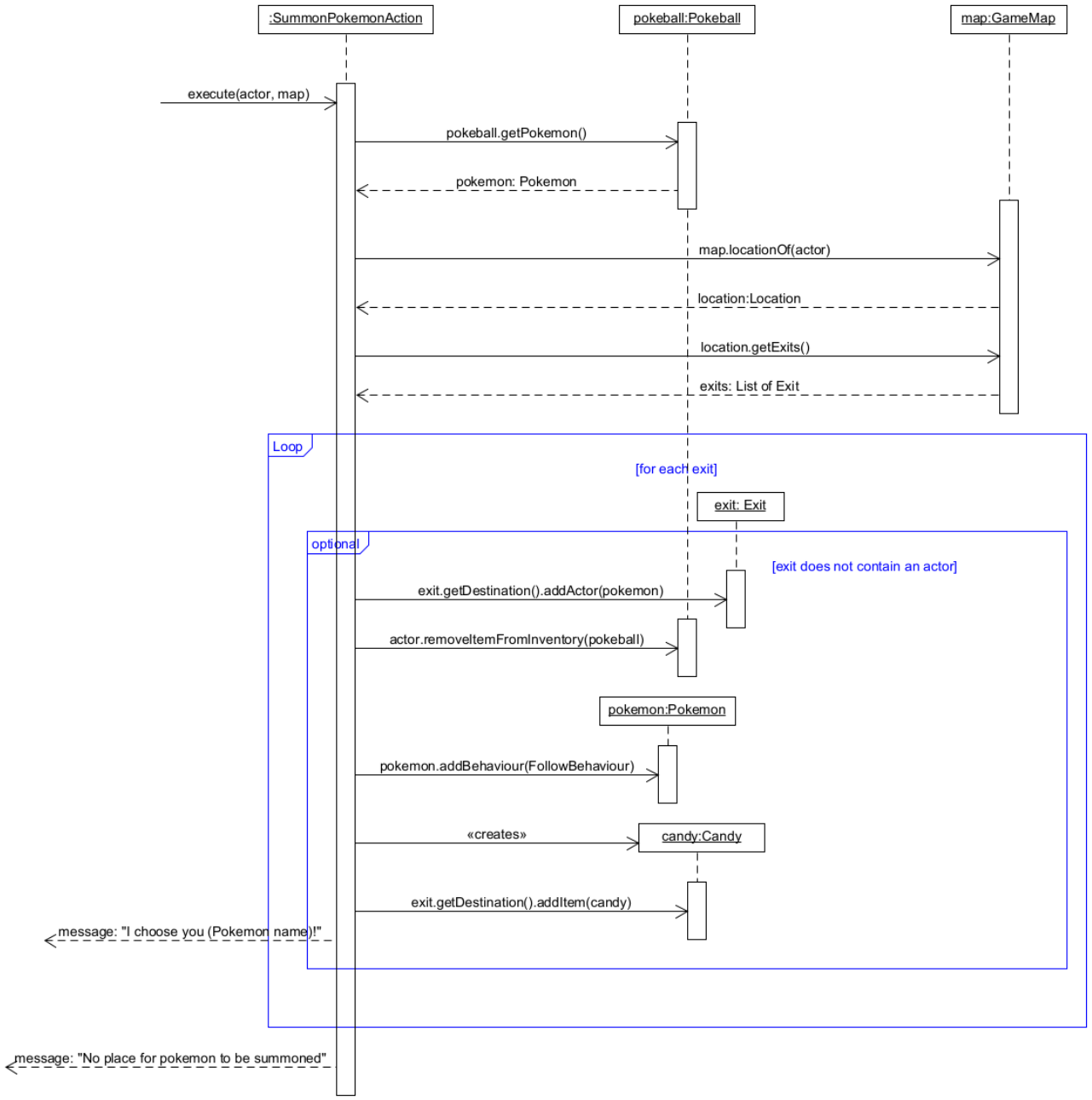
Changes: Tradeable interface is introduced. Classes that are tradeable objects like charmander and pokefruit will implement this interface. TradeAction has one Tradeable attribute. The TradeManger class is dependent on the Charmander and Pokefruit class as it creates it when returning all the TradeActions the player can make to the NurseJoy.

Sequence Diagrams

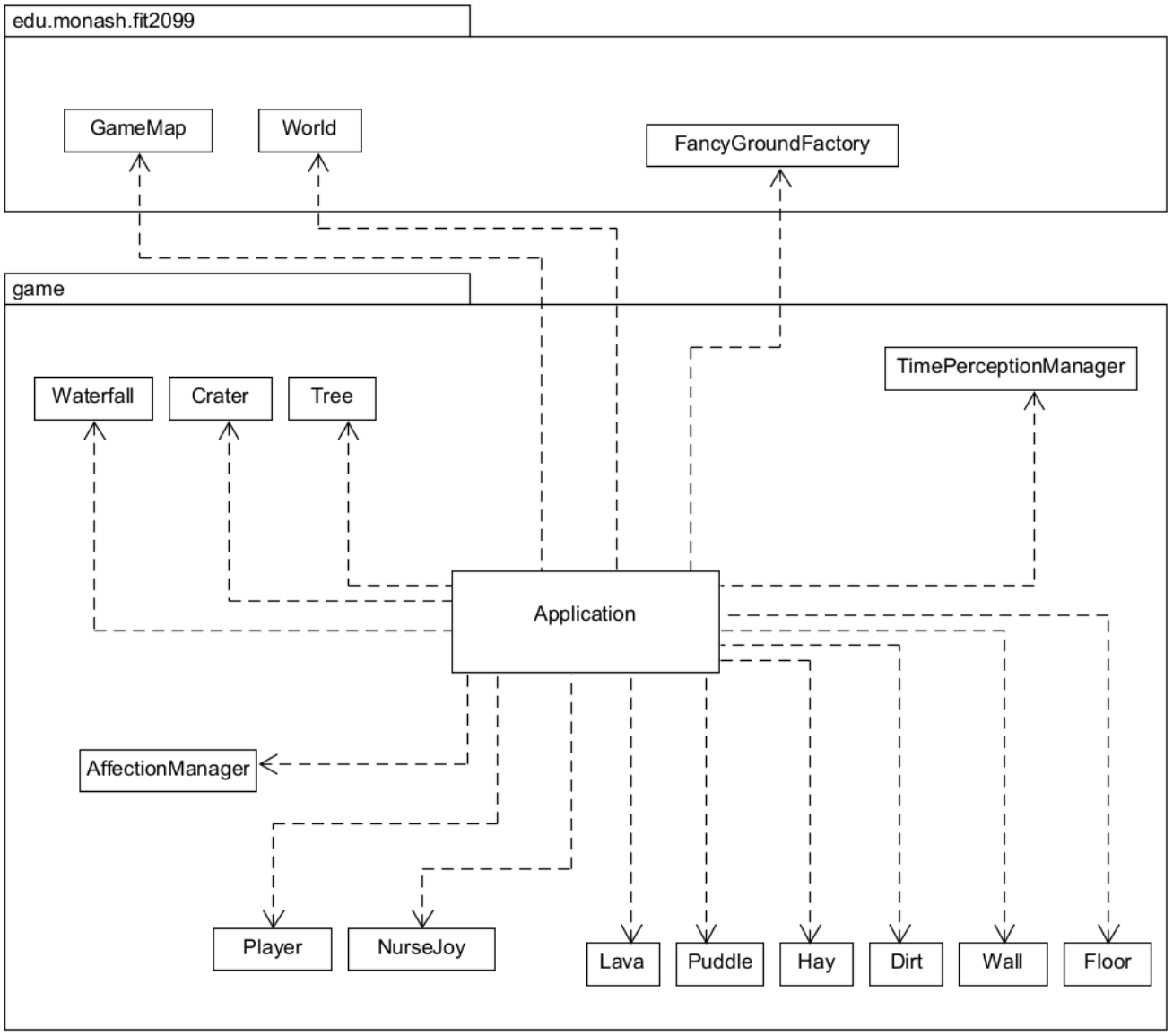
CatchPokemonAction



SummonPokemonAction



The Overall Application of the Pokemon Game



Revised Overall Application