MONASH University

FIT2099 G2 Assignment 3 Design Document

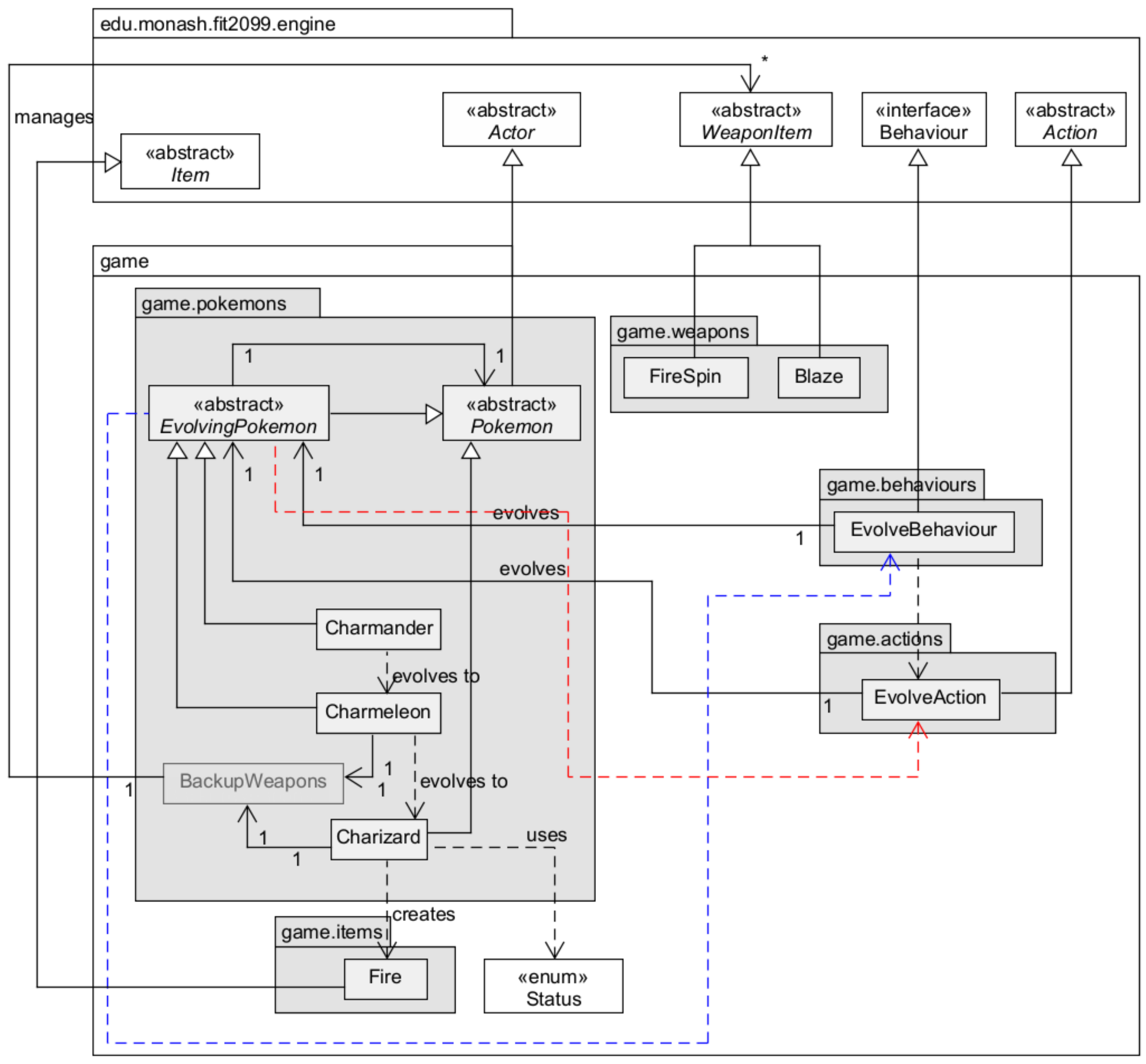
Team Members:

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

Contents

REQ1: Evolution

UML Class Diagram



Design Rationale

To obey the DRY (Do not repeat yourself) principle a new abstract class is introduced which is the EvolvingPokemon class. This class abstracts the features of evolution that a pokemon that can evolve should have, like a method to check if the pokemon should evolve or a method that returns the pokemon that it should evolve to. This class in turn inherits from the pokemon class so the main features of a pokemon can also be used at the same time. The EvolvingPokemon abstract class also has a one to one relationship with the Pokemon abstract class as it has an attribute that stores the instance of the pokemon that the EvolvingPokemon is supposed to evolve to. So now, pokemon that are capable of evolving should inherit from the EvolvingPokemon abstract class and those that cannot evolve should directly inherit from the Pokemon abstract class. The new concrete pokemon classes

Charmeleon and Charizard are introduced. The Charmander and Charmeleon are capable of evolving and as a result they inherit from the EvolvingPokemon abstract class, while the Charizard class inherits directly from the Pokemon abstract class. There are two new weapon items, the Blaze and the Fire Spin. Hence, the new Blaze and FIreSpin classes that inherit from the WeaponItem class are added to the game. Charizard depends on the new Fire item as it will create fire in its surroundings when it equips the FireSpin weapon. It checks if the weapon it has equipped is a FireSpin weapon using the Status enumeration.

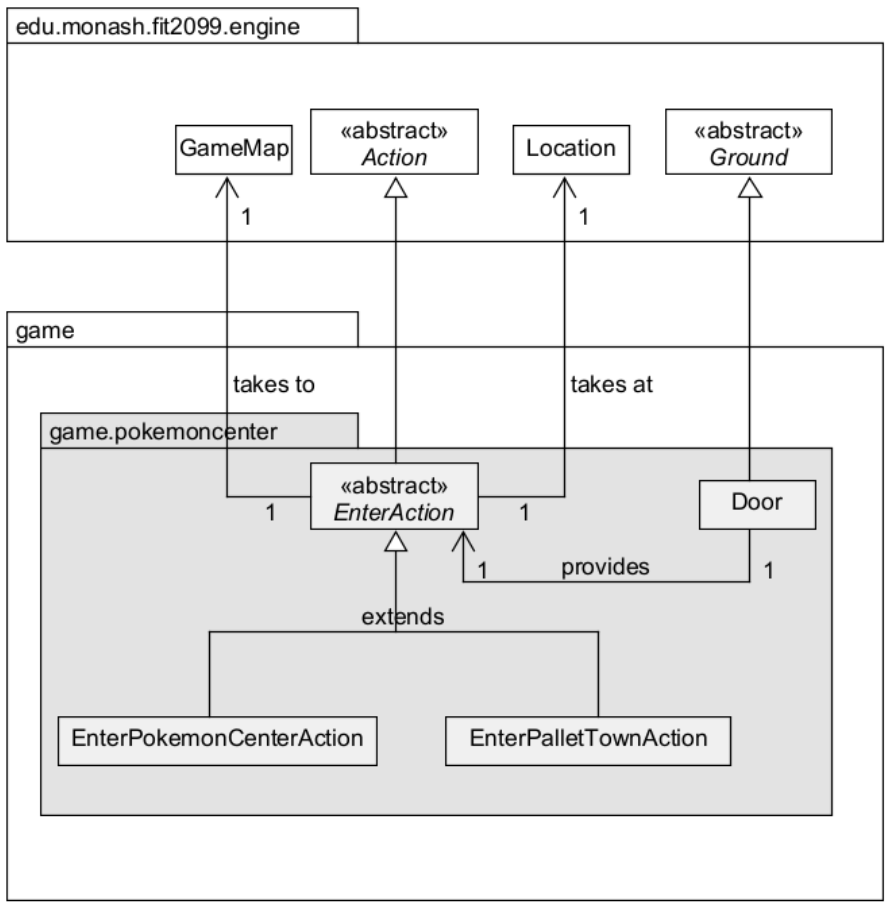
Charizard and Charmeleon classes both have a one to one relationship with the BackupWeapons class as it is the manager for their weapons system. The BackupWeapons system was refactored due to the fact that now the Charizard and Charmeleon pokemons can have more than one special weapon. For this reason the BackupWeapons class now has a one to many relationship with the WeaponItem class instead of a one to one relationship like earlier.

A new EvolveBehaviour is introduced to allow for the EvolvingPokemon to evolve if the Pokemon has survived long enough and there are no actors in the surrounding. The EvolveBehaviour has a one to one relationship with the EvolvingPokemon so that it can use its methods to decide if the evolution should happen or not. The EvolveAction class inherits from the Action abstract class and like the EvolveBehaviour it has a one to one relationship with an EvolvingPokemon class. This is so that the EvolveAction can carry out the Evolution successfully. The EvolveAction also allows the player to manually trigger the evolution of the pokemon if it has 100 affection points. The EvolveBehaviour is dependent on the EvolveAction as it will only return an EvolveAction if the pokemon meets the criteria to evolve.

All the classes in this requirement meet the Single Responsibility Principle (SRP).

REQ2: Pokemon Center (New Map)

UML Class Diagram

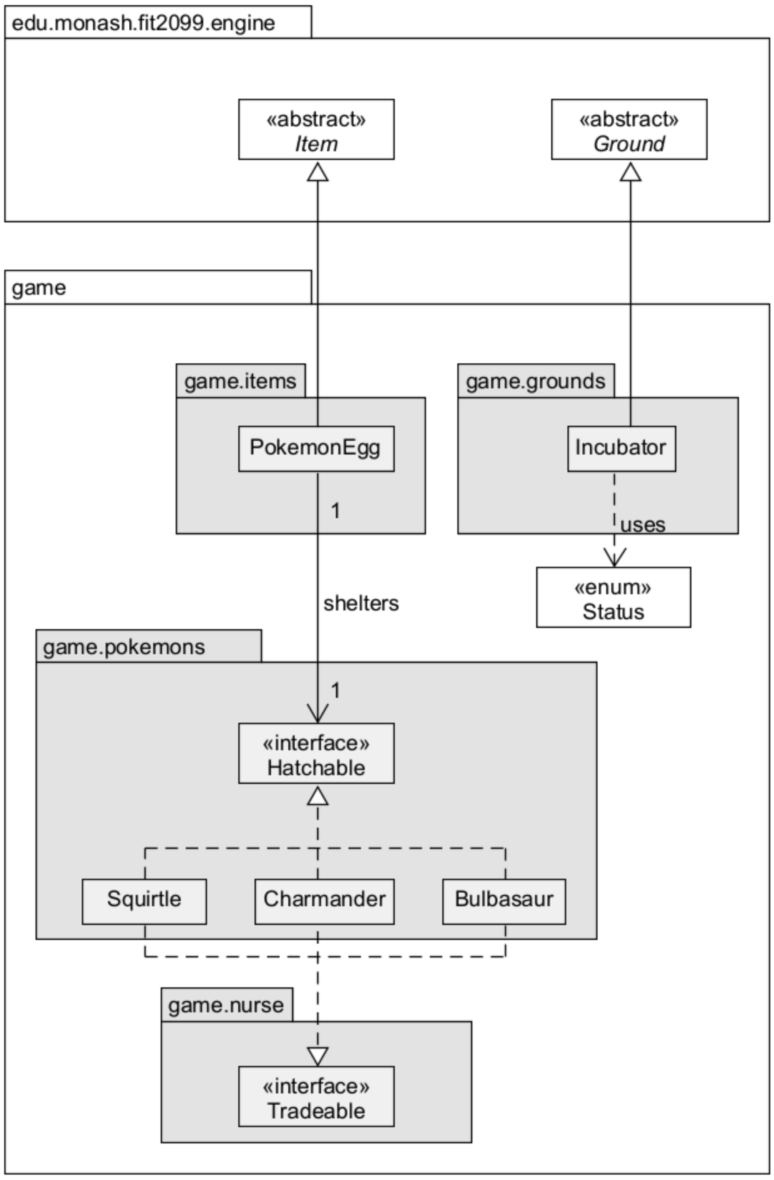


Design Rationale

For this requirement there needs to be an enter action for the player to choose when it is on a door. Hence, we have created a new EnterAction that inherits from the Action class present in the game engine. The door inherits from the Ground class as it is logical for the door to be a ground. It can prevent pokemons from entering it and it will also allow for the user to execute the enter action when the player stands on it. Each door has a specific location and map that it takes the player to. For this reason a door class has an attribute of an enter action (one to one relationship). So, this way each door has an enter action that it can provide to the player of the game. The EnterAction has been made an abstract class because it contains the main logic for removing an actor from one location in a map and adding the same actor to another location in another map. This follows the DRY (do not repeat yourself) principle. The two concrete classes that inherit from this abstract EnterAction class are the EnterPokemonCenterAction and EnterPalletTownAction. These two classes are present to override the menu description method of the action class to provide a specific menu description to the user. Like, “Enter Pallet Town” or “Enter Pokemon Center”. The door and the enter action follow the single responsibility principle as the door is only responsible for providing the respective enter action to the player and the enter action is only responsible for moving the player from one map to another.

REQ3: Pokemon Egg and Incubator

UML Class Diagram



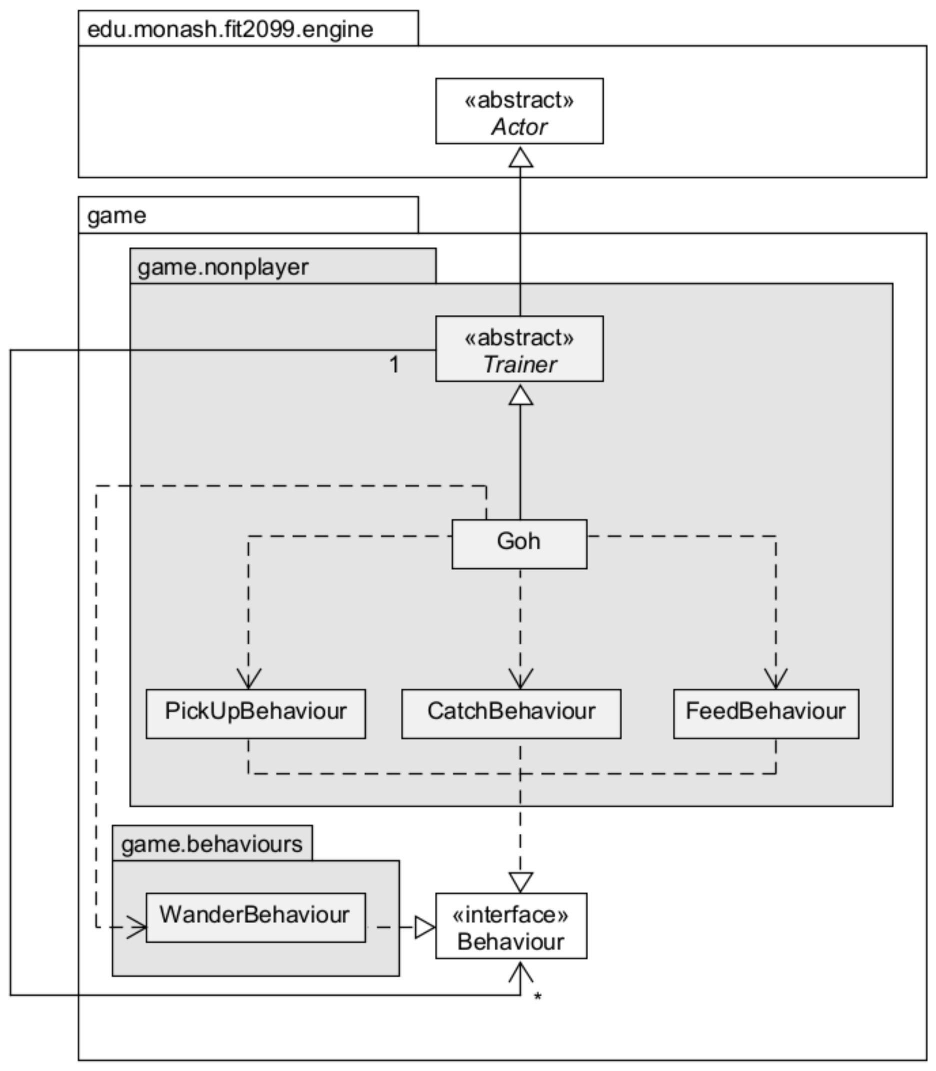
Design Rationale

A new Incubator class is introduced. This class inherits from the Ground class in the game engine to adhere to the DRY principle and so that it can inherit features that a ground possesses. The incubator has the ability of Status Incubate. This is so that when the egg is on a ground it can be understood whether that ground is an incubator or not. The PokemonEgg inherits from Item class as it is an item that can be stored in the inventory, picked up and dropped on the ground. The PokemonEgg concrete class has an attribute of Hatchable (one to one relationship). Hatchable is an interface that those pokemon that can be born through eggs will implement. This follows the Interface Segregation Principle (ISP). This is because the Hatchable interface is not forcefully placed in a single interface where other functionality exists. So pokemons only have to implement Hatchable if it has the capability of being born in an egg. The design has been updated to now have all the base pokemons (Squirtle, Charmander and Bulbasaur) to be Tradeable (earlier only Charmander was Tradeable) so that their eggs can be bought from Nurse Joy. The tick method of the PokemonEgg class checks if the ground it is on is an incubator. Using this it will increase the hatch time and check if it is the required time to hatch the pokemon. If it is, the pokemon will be spawned on top of the incubator and the egg will be removed. The PokemonEgg follows the single responsibility principle (SRP) as it only contains the logic to hatch the pokemon from the egg appropriately.

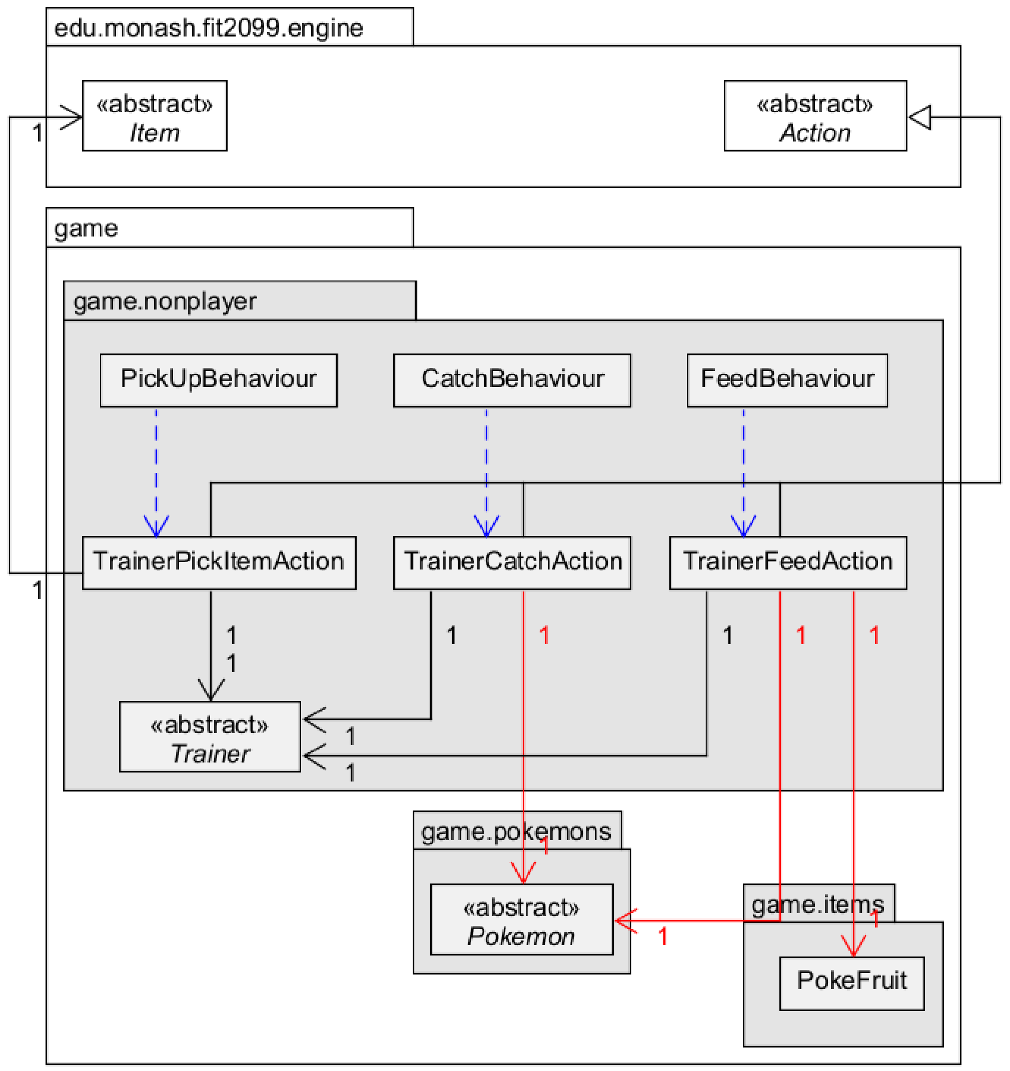
REQ4: New Trainer

UML Class Diagram(s)

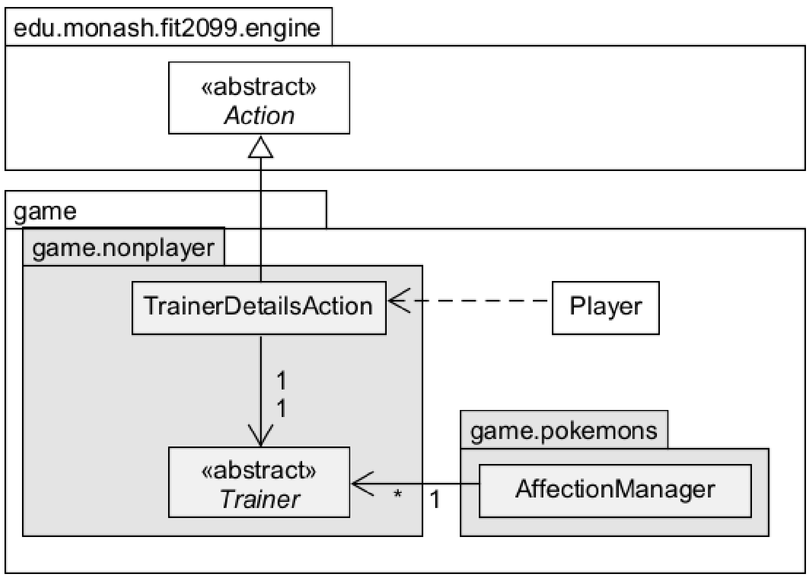
Overall trainer design



Behaviours and actions design for trainers



Details action for trainers



Design Rationale

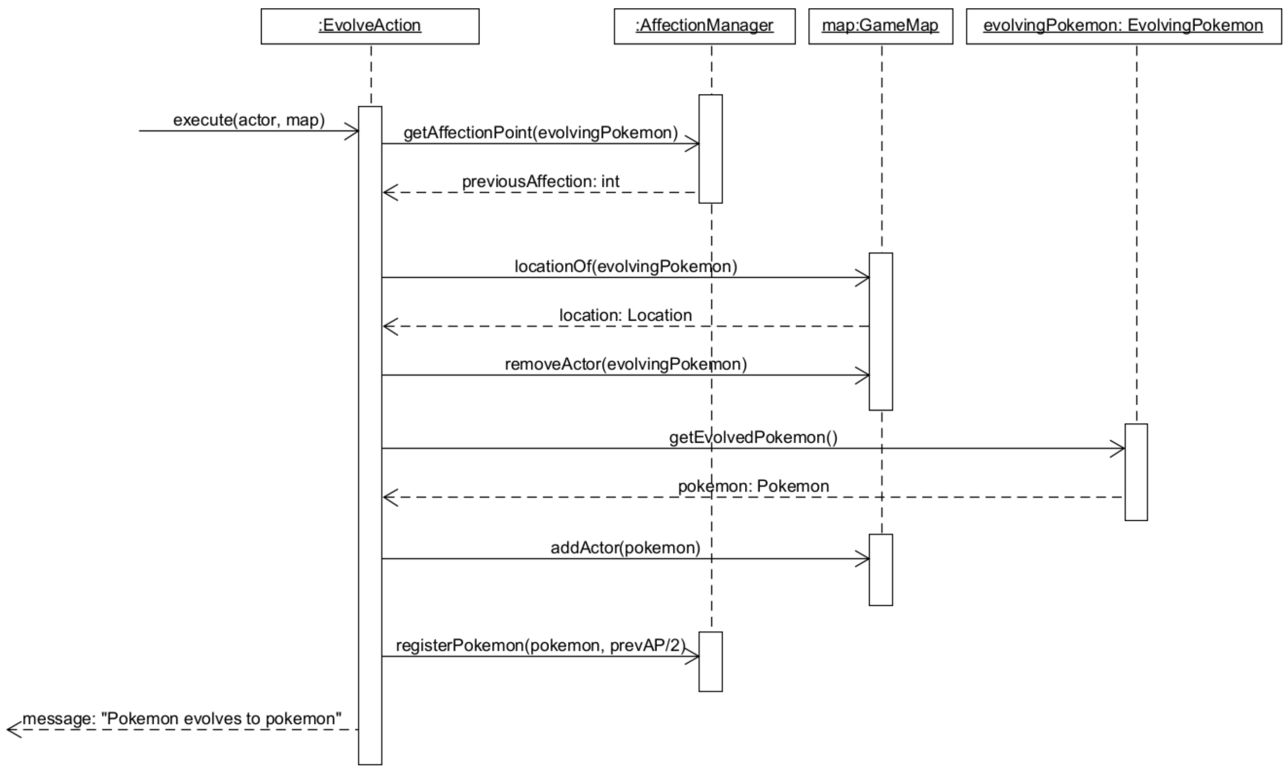
To obey the DRY (do not repeat yourself) principle a new abstract Trainer class. Any new trainers that will be introduced to the game can inherit from this class. Trainer class has an attribute of a list of behaviours. Hence, the one to many relationship with the Behaviour interface. The new Goh class is added to represent the Goh trainer and it inherits from the abstract Trainer class. There are three new behaviour classes that have been introduced - PickUpBehaviour, CatchBehaviour and FeedBehaviour. The Goh class will depend on these behaviour classes (add it to its behaviours) and the already existing WanderBehaviour that is also used by pokemon classes. Each of these behaviours follow the SRP (single responsibility principle)

Three new actions have been added to the application to cater for the previously mentioned behaviours. These are the TrainerPickItemAction, TrainerCatchAction and TrainerFeedAction. Each of the behaviours mentioned above will depend on these respective actions. The TrainerPickItemAction has a Trainer attribute and an Item attribute. The TrainerCatchAction has a Trainer attribute and a Pokemon attribute, hence the one to one relationship with them. The TrainerFeedAction on the other hand has three attributes all one to one relationships. They are the Trainer, Pokemon and a PokeFruit. Each of these actions follow the SRP principle.

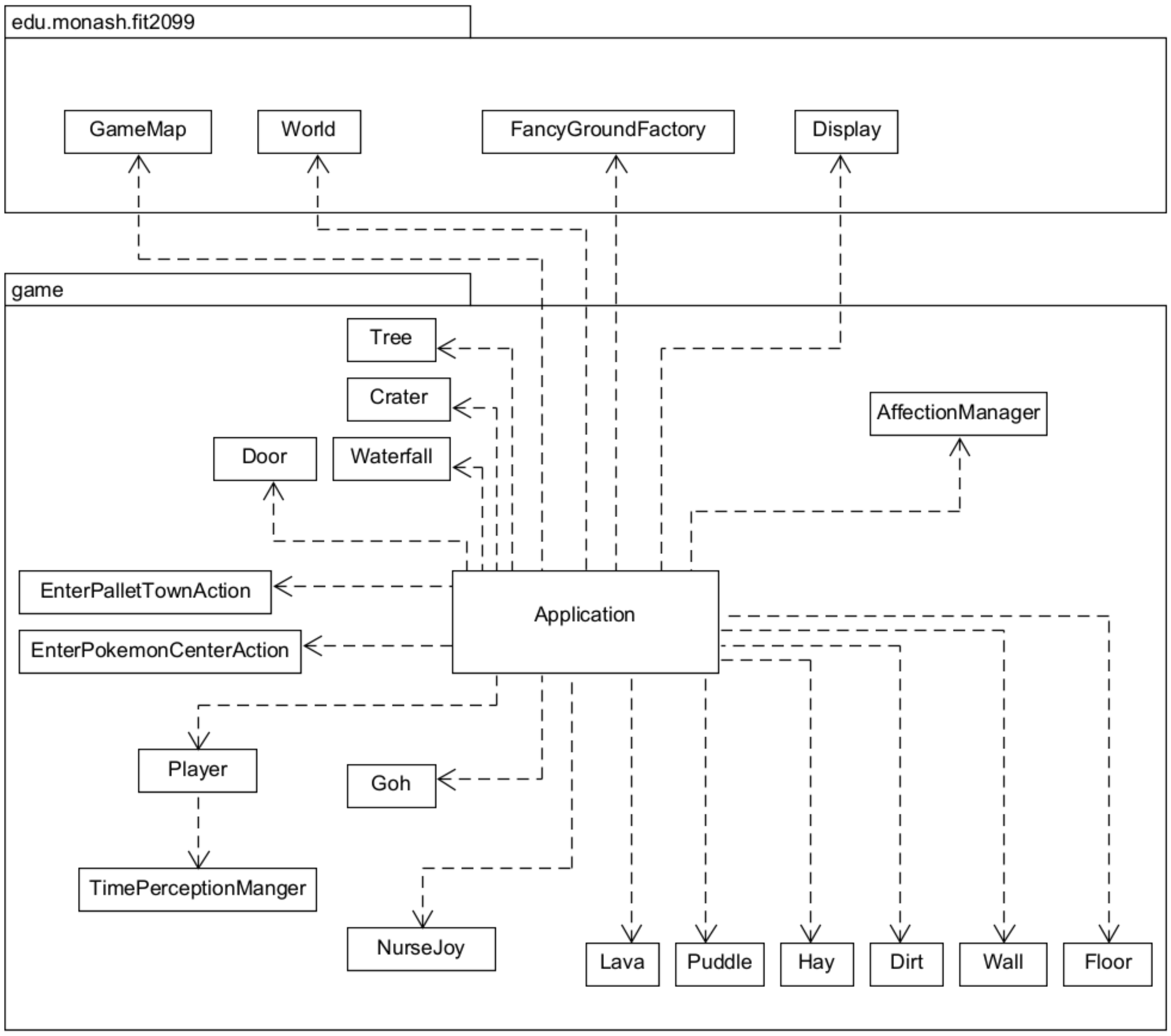
A new TrainerDetailsAction is introduced into the game so that the main player can choose an action to see details about all the trainers in the game. To cater this the AffectionManager class has been refactored. It now has a one to many relationship with a Trainer (List of Trainers). To display the details of a trainer the TrainerDetailsAction has a Trainer attribute (one to one relationship). The Player class (main player of the game - ash) has a dependency on the TrainerDetailsAction as it will have to create an action for each of the trainers on the map every play turn.

Sequence Diagram(s)

Evolve Action Sequence Diagram


:EvolveAction
:AffectionManager
map:GameMap
evolvingPokemon: EvolvingPokemon
execute(actor, map)
getAffectionPoint(evolvingPokemon)
previousAffection: int
locationOf(evolvingPokemon)
location: Location
removeActor(evolvingPokemon)
getEvolvedPokemon()
pokemon: Pokemon
addActor(pokemon)
registerPokemon(pokemon, prevAP/2)
message: "Pokemon evolves to pokemon"

The Overall Application of the Pokemon Game


edu.monash.fit2099
GameMap
World
FancyGroundFactory
Display
game
Tree
Crater
Door
Waterfall
AffectionManager
EnterPalletTownAction
EnterPokemonCenterAction
Application
Player
Goh
TimePerceptionManger
NurseJoy
Lava
Puddle
Hay
Dirt
Wall
Floor