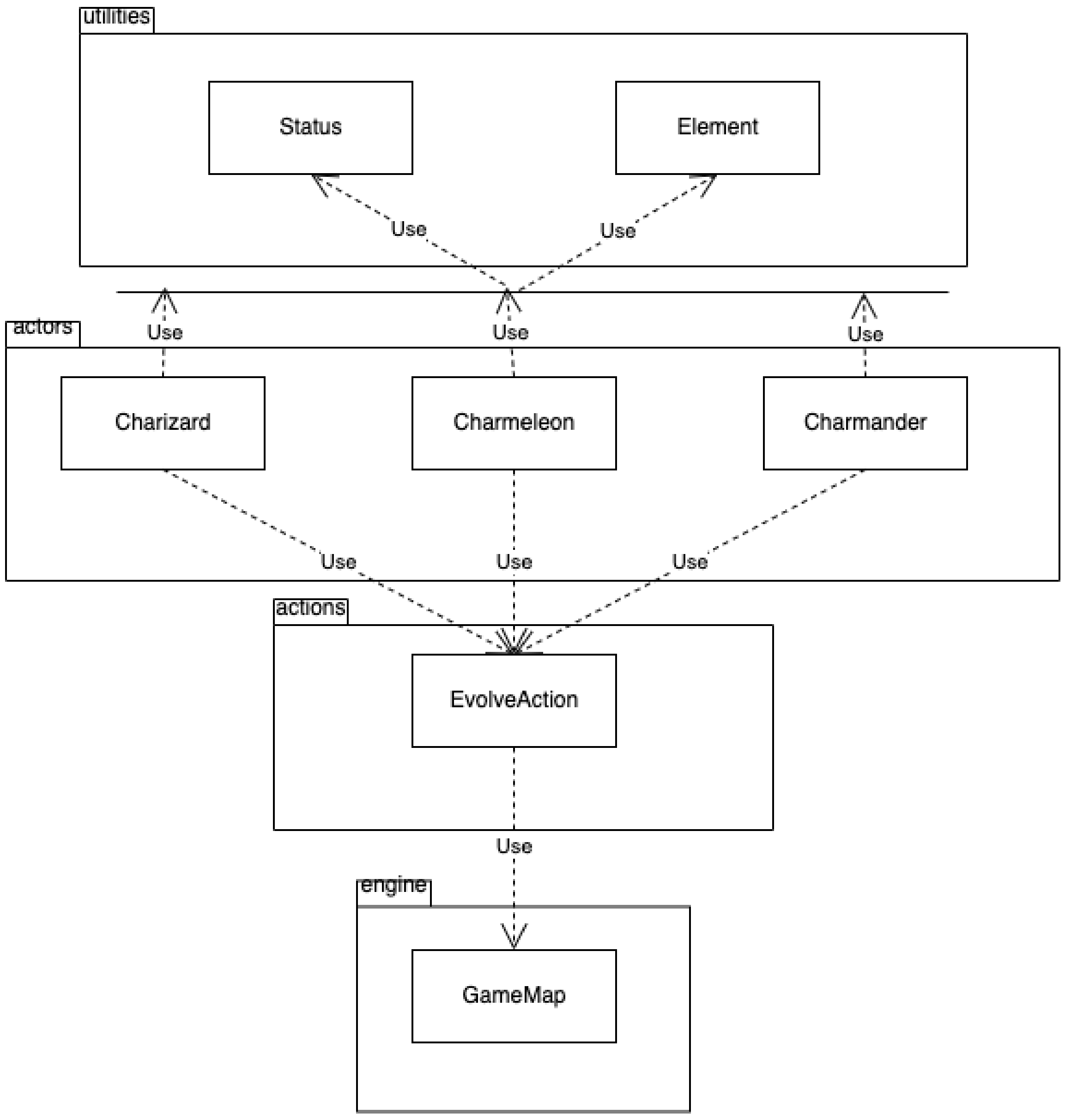
Requirement 1 - Evolutions

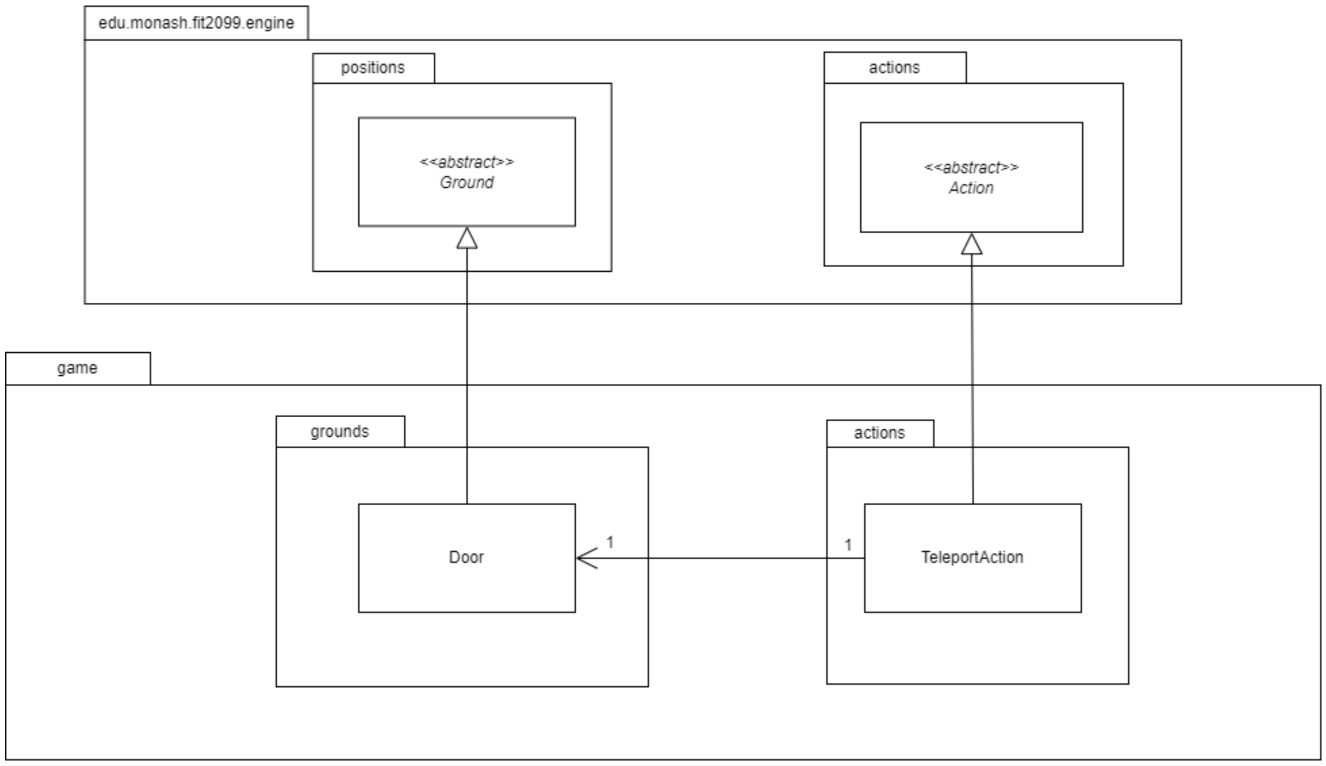


The diagram represents how evolutions work on this program. Firstly, we see that the Pokemons are adding capabilities from the utilities package to determine their element and status.

Now what the EvolveAction does is when a pokemon meets the requirements for an evolution it will call the EvolveAction method where the method will get the current map its playing in and swapping the current pokemon with the new evolved pokemon in the map with the methods provided from GameMap.

Requirement 2 - Pokemon Centre (new map)

Class Diagram



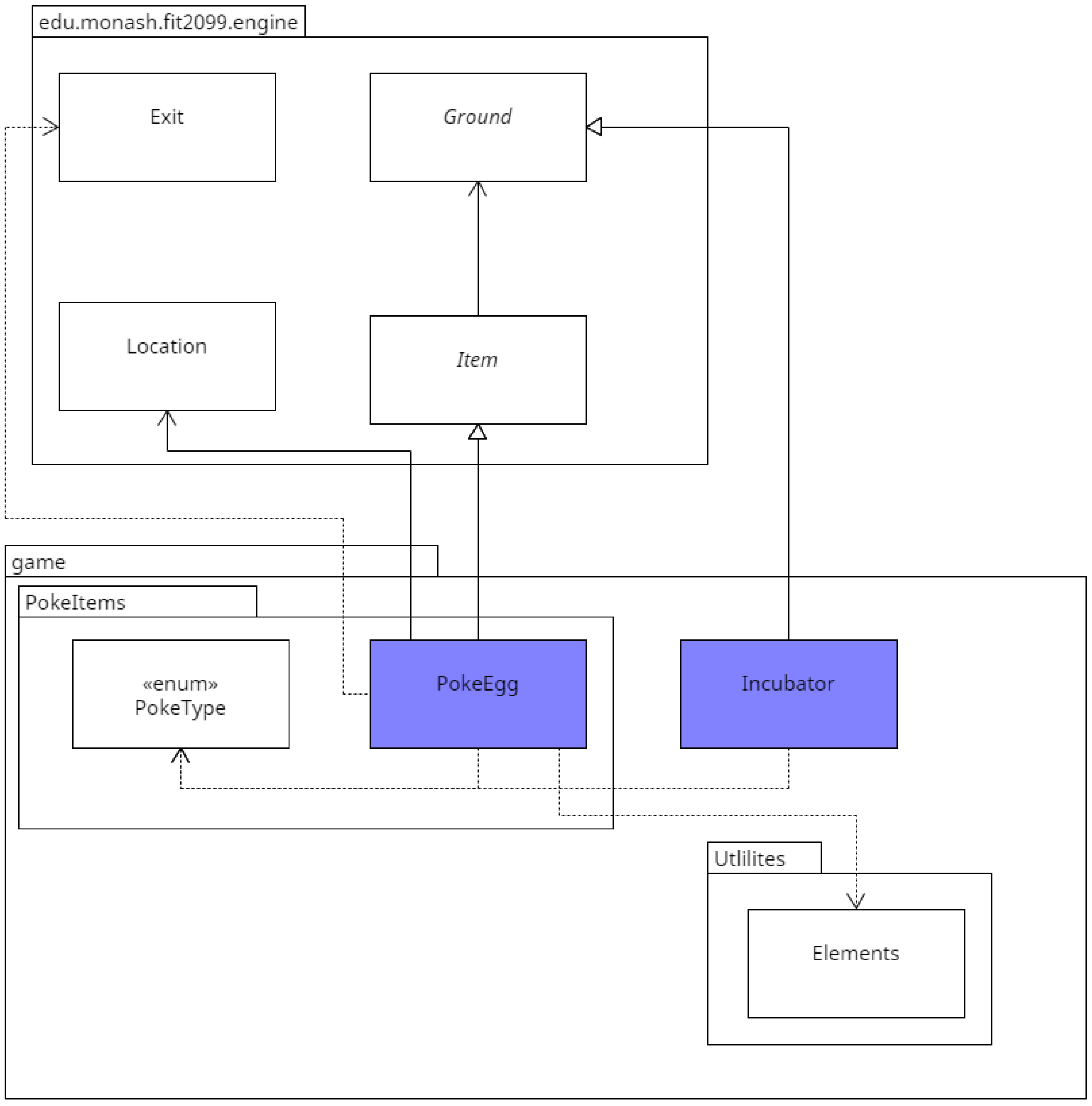
The diagram represents a part of the game where a player can teleport from one map to another through a door. It has 2 concrete classes.

The Door class extends the abstract class Ground as it has all properties of a Ground, hence it will prevent any repetition of methods from happening. By using the allowableActions method from the Ground class, we can implement the door to act as a teleporter when it acts as one of the exits for the player.

The TeleportAction class has an association on the Door class as it has a Door attribute that represents the target door it will teleport to. The constructor of the TeleportAction has a parameter of type Door which represents the target door it will teleport the player to.

The TeleportAction class extends the abstract class Action as it is a type of action, and hence will have all properties a general action will have, preventing any repetition of methods from happening. By using the execute method in the abstract Action class and overriding it, we can implement a way of teleporting from one map to another.

Requirement 3 - Pokemon Egg and Incubator

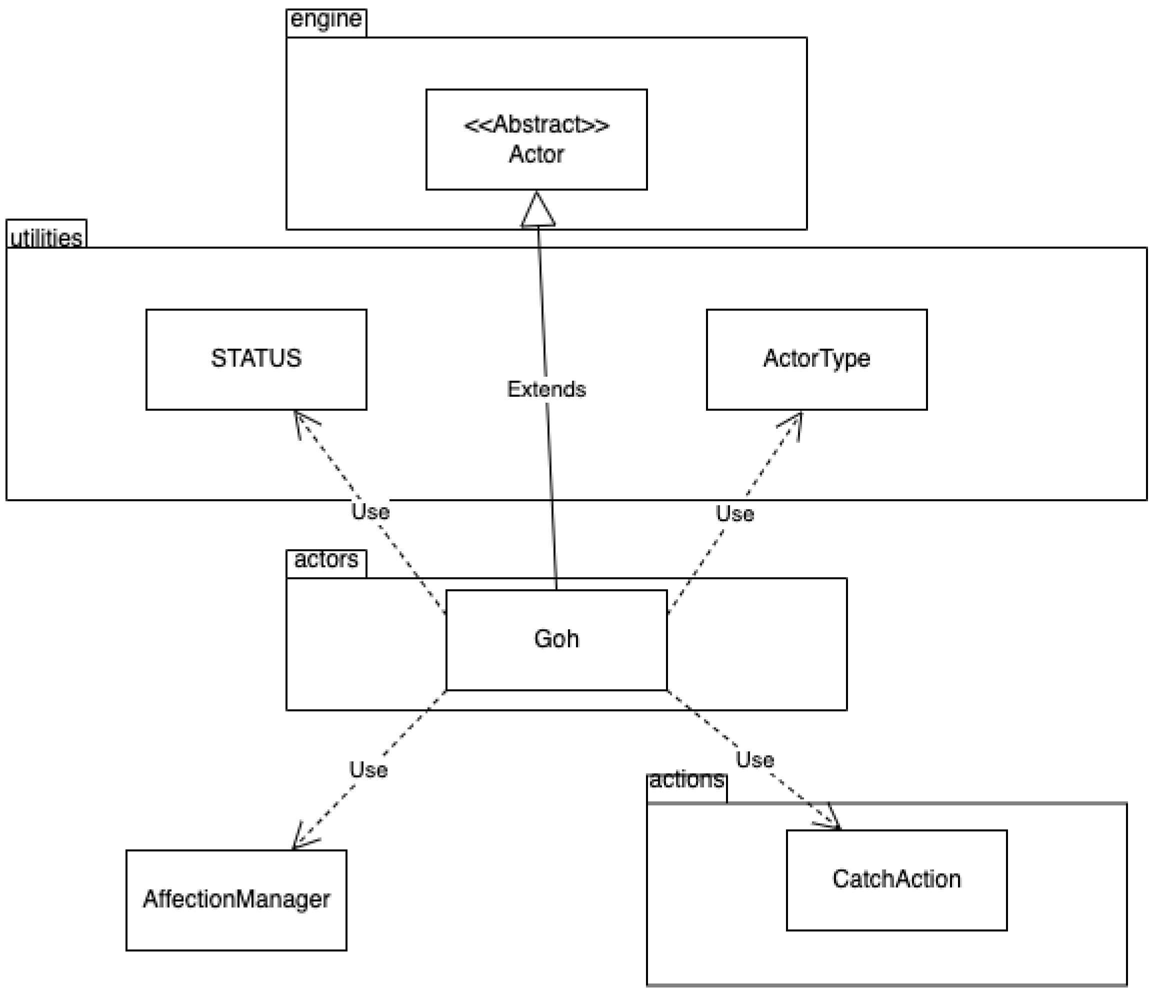


The diagram represents a part of the game where the player would buy Pokemon Eggs and put it into the incubator for the pokemon to hatch.

The Incubator class would be inherited from the ground abstract as it has all the properties of the ground abstract, therefore there was no need to repeat the code, resulting in not failing the DRY principle. It also has the capability called Incubator in PokeType, so that the team can search it later on for ground Type in PokeEgg.

The PokeEgg class would be inheriting the item class as it has all the properties of item class. So the DRY Principle is still intact. The PokeEgg Class has the capability of the element and the POKEEGG in the PokeType. We do this so that we can differentiate the element of the pokegg and the incubator when dropped at the incubator. The team then also is dependent on the location as we would need its tick and currentLocation for the egg to hatch.

Requirement 4 - Goh (New AI Trainer)

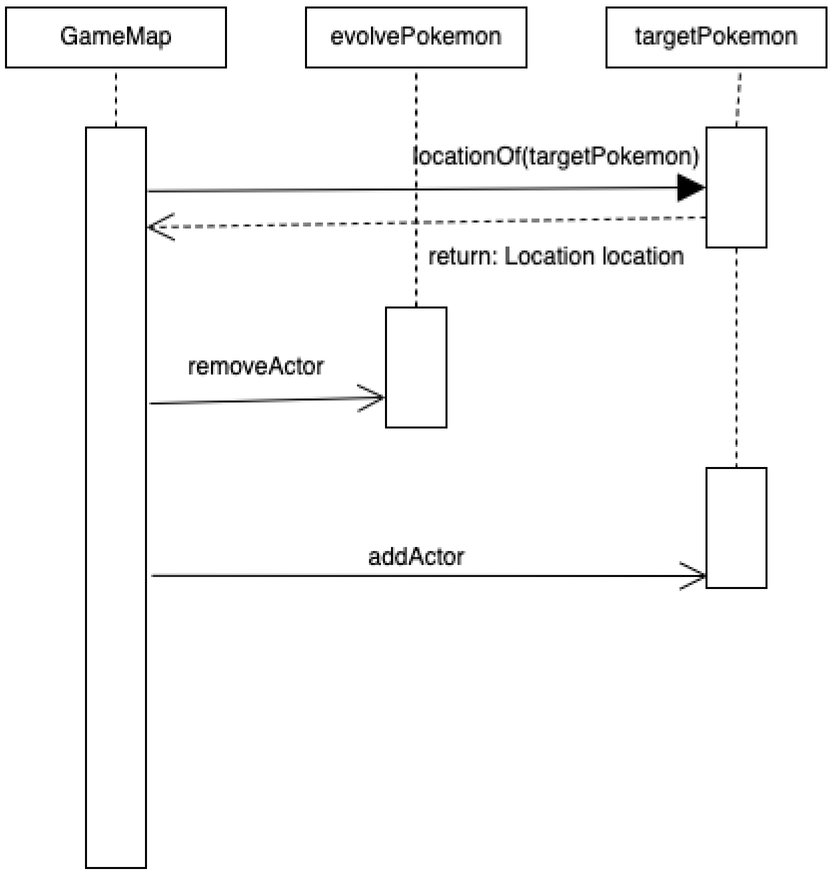


The diagram represents the new AI trainer, Goh. Firstly, we can see that Goh is using the status and actorType from the utilities package to determine the type of actor Goh is and to determine the status of Goh.

When Goh encounters a pokemon he will try to feed it, but instead of using the existing FeedingAction, Goh directly manipulates the AffectionManager because FeedingAction has behaviours that are not aligned with the requirements such as "Whenever Goh stands next to any Pokemon, he will give a Pokefruit to the corresponding Pokemon (disregarding their elements).", this won't work with the current FeedingAction which has consequences if the pokefruit elements are not the same.

Whenever Goh has a sufficient amount of AP towards a pokemon he will catch it. And he does this through the CatchAction. Since Goh is not a player, we will put everything that Goh can do every turn in a list and execute all of them in that turn.

Sequence Diagram - Evolution


GameMap
evolvePokemon
targetPokemon
locationOf(targetPokemon)
return: Location location
removeActor
addActor