Group CL_Lab02Team03
Assignment 2

Design Rationale

Design Goals

The project is a “rogue-like” game that has a Pokemon theme. In this game, the user can control the player character to catch and battle pokemons. The design goals for the this project are:

- Be able to extend the number of Pokemons
- Have Single Responsibility that delegates the responsibilities of the game to the respective class
- Have the base game function be open for extension but closed for modification

The design rationale for the project game will be split into 5 parts, covering all requirements for the game.

Req1: Environment

The following diagram represents an object-oriented implementation for the environment of the game that has 2 different types of grounds: Ground Class which represents the regular ground with no additional features, Pokemon Ground Class which represents grounds with element types and spawn features, and Spawnable which are grounds that can spawn instances of pokemons.

All possible grounds extend the abstract Ground Class. The idea to create another abstract class that implements the grounds for this game was considered depending on possible considerations for future extension, however after considering the base engine it was decided that the game should be more focused on additive features implemented in the abstract Pokemon Ground class instead.

The Pokemon Ground class that extends the Ground class implements the additional features of ground that have element types and can spawn. This made it possible to assign the ground a type with the Element enum making it more extendable rather than creating an interface for each type. The abstraction of Pokemon Grounds avoids the repetition of features.

The Spawnable interface was implemented in order to have additional features for classes that can spawn instances of pokemons. This was implemented in an interface because it allows us to be able to reuse the functionality in the possibility where we want to spawn a pokemon in grounds that are not element typed. In addition, this Spawnable interface is responsible for the features for spawning Pokemon instances and all pokemons belong to this Pokemon class. Crater, Waterfall, Tree spawn a pokemon therefore the Pokemon class is an abstract class and each pokemon extends it. Having the interface reduces the overall need for dependencies when spawning pokemons.

Req2: Pokemons

The following diagram represents the features for the game's Pokemon characters. The Pokemon will have its respective element types and it will spawn from its respective spawner. In addition it will have a favorite action, and both an intrinsic attack and a special attack.

All pokemons extend the abstract Pokemon class which extends the Actor class from the game engine as they are Pokemon Actors that have different grouped features compared to the Player class. This makes it possible to reduce the repetition of coding each pokemon individually and simply the features can be extended from the Pokemon base class.

The Attack Action class will extend the Action class from the engine and this will be extended to implement the intrinsic attacks: Scratch and Tackle. The BackupWeapon Class will implement features for the Special Attacks that can be equipped when certain conditions are met.

Req3: Items

The following diagram represents the items that are used in the game which there are two types of items at the moment, items that are Balls for and items that are not balls.

The abstract class PokeBall is implemented as the base class for two types of Greatball and Msterball which can have varying additive functionalities and characteristics. This limits coding each ball distinctly and also allows straightforward extension when creating new types of pokemon balls such as QuickBall.

The candy class also extends items and will be able to be picked up when the conditions are met thus it interacts with the player. The item abstract class can be easily extended when new items are introduced and when new types of introduced abstract classes to group the items can be created.

Req4:Interactions

The following diagram represents the interactions the player can have with Pokemons. The 4 main parts to the pokemon interactions are, 1. Changing Pokemon Affection levels, 2. Player Interactions with the Pokemon, 3. Capturing and Summoning Pokemons, 4. Npcs action(buy and pick up)

The abstract PokemonAction class extends the abstract Action class to implement the Capture, Summon and Interaction actions. The Player will perform actions through the PokemonAction class and the AffectionManager uses the class to track the resulting affection levels of the Pokemon. In each iteration of the game the engine will track the available actions by checking the ActionList thus showing the available action to the user. This implementation allows the affection tracking and action to build off of the engine action class. As all actors use the Action class this will implement the Single Responsibility principle and delegate the responsibility to the respective class.

Req5: Day and Night

The main purpose of the UML diagram shown below is to place the time package with other related classes. There is an enumeration class that contains DAY and NIGHT, the two types of time shift. TimePerceptionManager manages the interface class Time Perception. The Time Perception interface class allows defining the consistent features that can occur in the requirements, in which some Ground classes (Lava, Puddle, Tree) are affected by the time shift resulting in distinct features in that time period, all Pokemon classes also have similar day and night features.

So our first step is to draw the relationship between the time package. Connect all the classes that exist in the time package. Then make connections between specific classes and TimePerceptionManager. And another important thing is to make connections between the pokemon abstract class and specific Ground classes to implement the interface class TimePerception so that we can implement the method dayeffect and nighteffect in those classes. Since some special Ground classes have the chance to turn into other Ground classes (i.g Lava -> Dirt), we also made the connection between the abstract class Ground and Location. So that we can implement the method in the location to print the new Ground class in the game map.

In this part, I used the Single Responsibility Principle. There's a loop in TimePeriodmanager that will create a CommonTimePerception after 5 turns and then put it in TimePeriodmanager. The method of calculating the turns is in TimePeriodManager class. The CommonTimePerception must implement the method declared in the TimePerception interface. I think the TimePeriodManager is to control whether it is day or night. If it is the day, go to dayeffect method in CommonTime. Timeperiod specifies that there are several types of time(we can easily expand more functions in this way), by this way we can easy to add a new type of time

in the future. The interface class is the methods that you want to implement like what affects you during the day or night.

Req6: NPCs

The main function of Req6 is to set up two actors: ProfessorOak and Shopkeeper. These two npcs need to inherit from the actor class. Because npcs can not be attacked , we need to add a status(*IMMUNE*) to make them not be attackable. Professor and shopkeeper class also need to inherit their respective actions from the action class. We will add BuyAction and PickUpAction in action class which will help the professor and shopkeeper to play the role for their functions. By this way, we can extension more npcs and functions in the same way.