Design rationale

Environment

SpawningGround extends Ground. Abstract SpawningGround class that inherits abstract Ground class has the necessary methods to generate terrain. SpawningGround class is similar to Ground class, the difference is that it is a dynamic ground. And, the reason to make SpawningGround to be an abstract class is that abstract class is useful to create new types of terrain class, like Crater class, Tree class and Waterfall class.

Crater extends SpawningGround, **Tree extends SpawningGround**, **Waterfall extends SpawningGround.** They are all dynamic ground, hence they inherit from SpawningGround.

Crater---<<spawn>>--->Charmander, **Tree---<<spawn>>--->Bulbasaur** and **Waterfall---<<spawn>>--->Squirtle**. These 3 types of terrain will spawn 3 different types of pokemons. And, the terrains don't need to change any information, like behaviours and capabilities of pokemons. Thus, using the dependency relationship is enough for them. It is also an alternative to have an association relationship between the three terrains and the three pokemons. In other words, pokemons will be the attributes of the terrains, and the capabilities of the pokemons can be modified in terrain classes. However, it is better to reduce the dependency between classes to prevent high coupling.

Charmander extends Pokemon, Bulbasaur extends Pokemon, Squirtle extends Pokemon. These 3 types of pokemon inherit abstract Pokemon class, because it contains the behaviour of a pokemon.

SpawningGround---<<create>>--->Pokemon and **SpawningGround---<<drop>>--->Pokefruit.** SpawningGround abstract class has Pokemon abstract class and Pokefruit concrete class inside. This is because SpawningGround always comes with these two objects. It is a place to create a pokemon and drop fruit at every turn. The reason to make Pokemon an abstract class, and a Pokefruit concrete class is that there are many pokemon classes, like Charmander, Bulbasaur, Squirtle, but Pokefruit only needs one class with a single enum element difference. It is also possible to make Pokefruit as an abstract class, but it needs to generate three more concrete classes to inherit the Pokefruit abstract class. Hence, it is better to set Pokefruit as a concrete class. Both Pokemon abstract class and Pokefruit concrete class are aligned with the Open-closed Principle.

SpawningGround---<<drop>>--->Pokefruit and **Pokefruit---<<has>>--->Element.** SpawningGround (Crater, Tree, Waterfall) drops pokefruit according to its location. And, each pokefruit has a different element (Fire, Water, Grass). From the game map, replace the SpawningGround with Pokefruit. Both of the relationships are dependency, as it only requires reading the information from the classes.

SpawningGround---<<has>>--->Element. The SpawningGround abstract class itself also has an element (Fire, Water, Grass). And, it needs to generate different element types of SpawningGround, thus it needs to have a stronger relationship, which is the association relationship.

Puddle extends Ground, Lava extends Ground, Hay extends Ground. Puddle, Lava and Hay concrete classes inherit the abstract Ground class which has the elementary methods to form a type of terrain with its respective capabilities. This has aligned with the Dependency Inversion Principle, as Ground abstract class does not depend on Puddle, Lava and Hay concrete classes.

Hay---<<has>>--->Element, Lava---<<has>>--->Element, Puddle---<<has>>--->Element. Hay, Lava and Puddle have three different types of element. And, they need to have a stronger relationship with Element class to add the capability of the element to each terrain. Hence, they use an association relationship with the Element class.

Day and Night

Charmander extends Pokemon, Squirtle extends Pokemon, Bulbasaur extends Pokemon and **Pokemon extends Actor.** Pokemon abstract class inherits Actor abstract class that has the necessary methods to generate an actor. Charmander, Squirtle and Bulbasaur concrete classes inherit a Pokemon abstract class that has the necessary method to generate a pokemon. This is aligned with the open-closed principle because the abstract classes don't allow modification, but are open for extension.

Lava extends Ground, Puddle extends Ground, Dirt extends Ground, Hay extends Ground. Lava, Puddle, Dirt and Hay concrete classes inherit the Ground abstract class that has the necessary methods to generate a type of terrain.

Puddle---<<convert>>--->Dirt. From the game map, replace Puddle with Dirt. And, it is not needed for a strong relationship, a dependency relationship is enough. However, it is also an alternative way to create an association relationship between Puddle and Dirt concrete classes. Puddle will need to use the Dirt class as an attribute. After checking whether the actor is on the location through if-statement, use a random number generator to keep the 10% chance and change the Puddle to Dirt inside the statement. In order to prevent high coupling, it is better to use a dependency relationship between Puddle and Dirt concrete classes.

Tree extends SpawningGround. Tree is a dynamic ground. And, it inherits SpawningGround that has the necessary methods to form a dynamic ground.

Tree---<<convert>>--->Hay and **Tree---<<drop>>--->Candy.** From the game map, replace Tree with Hay or replace Tree with Candy. Both of them are not required for a strong relationship, a dependency relationship is enough.

Charmander---<<implement>>--->TimePerception,
Squirtle---<<implement>>--->TimePerception,
Bulbasaur---<<implement>>--->TimePerception,
Lava---<<implement>>--->TimePerception,
Puddle---<<implement>>--->TimePerception,
Tree---<<implement>>--->TimePerception. These concrete classes have different behaviours during day and night, and they perform them through TimePerception interface.

TimePerceptionManager---<<use>>--->TimePerception and **TimePerceptionManager---<<use>>--->TimePeriod.** TimePerceptionManager will add an object instance (time) to an ArrayList. And, use TimePeriod to shift between day and night. TimePerceptionManager has a weaker relationship with TimePeriod, which is the dependency relationship, because it only reads TimePeriod and it will not modify anything inside the enumeration TimePeriod. However, TimePerceptionManager needs to have a stronger relationship with TimePerception interface, which is the association relationship, because TimePerceptionManager needs to manage the day and night effects of the TimePerception interface.