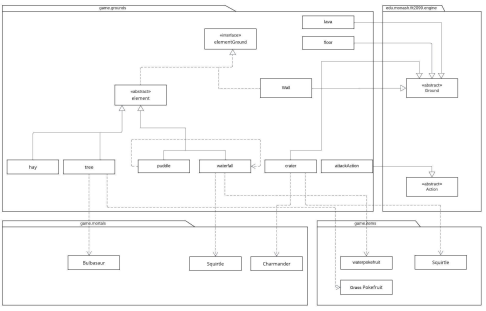
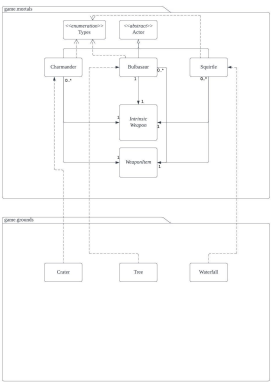
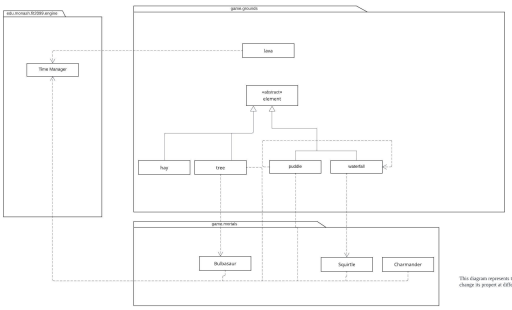


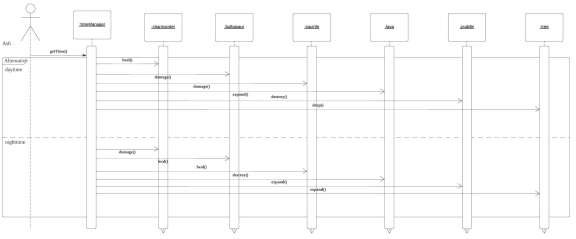
Lava will extend the **Ground** Class since it possesses characteristics of a Ground. This includes the ability for actors to move onto it. We can also make use and override the tick() method of the Ground class to implement the damage inflicted to the actor on the Lava.
– Follows DRY principle by extending from Ground. Makes use of pre-existing code within Ground which allows extension without modification which follows open-closed principles
we have a element abstract that is parent for all terrain with elements and extend to the interface elementGround.

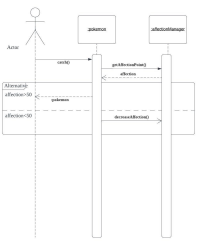


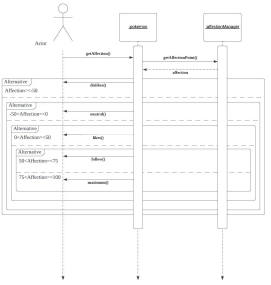
This diagram represents an object oriented system which represents a roughlike game. It makes sense that the classes Charmander, Squirtle and Bulbasaur inherit the abstract class actor as they share similar attributes and so should be inherited. Furthermore, they will use enumeration to represent their individual typings. To simulate their intrinsic and special attacks, the class Intrinsic Weapon will simulate their intrinsic attacks while their special attacks may be simulated by the class WeaponItem.

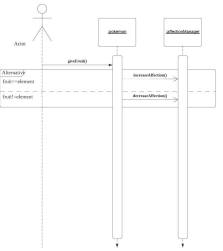


This diagram represents the system of time, the dependencies between time and items and terrain change its proper at different times.

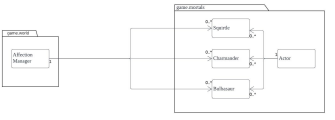


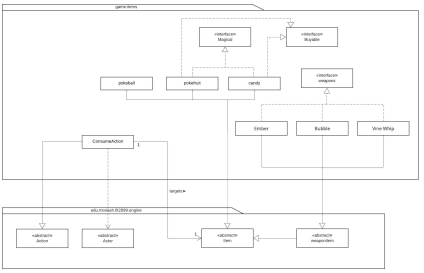






The UML diagram represents the affection system of the game. Affection is represented by Objects in Affection Manager. Bulbasaur will have an affection attribute while the game will have only 1 instance of the Affection Manager which is a class that is used to manage affection interactions such as decreasing or increasing affection.





Main role:
The main purpose of the **ConsumeAction** class is to provide the player with the ability to consume certain items. When an item is consumed, it will call the 'addCapabilities()' method and the 'addAffection()' method implemented from the **Magical** interface (which is further explained below). This allows any new capabilities and buffs to be applied appropriately to the player based on the item they consume. At the end of consumption, the item is removed from inventory.

Design Rationale:
The ConsumeAction requires several attributes that have already been implemented via the Action class which determined the Action class to be a suitable parent class when designing the ConsumeAction class. Additionally, since the ConsumeAction class is an action and generally maintains the "running" of the parent class's behaviour (being a preferentable action), the extension from the Action class adheres to good design principles, in particular with LSP (Liskov Substitution Principle).

Main role:
The main purpose of the Magical interface implemented by candy and pokehut which both extend the Item class, is to distinguish items that can be consumed and provide affection from regular items that do not. The interface provides methods such as addAffection() which apply effects from the item once consumed.

The implementation of this interface follows the open-closed principle of the SOLID principles where the implementation of new consumable items that possess effects require no modification of the Item class or interface. This is due to the fact that the addAffection() method can be overridden when implementing the new Class to comply with the desired affects.