FIT 2099 Assignment 2 Design Rationale

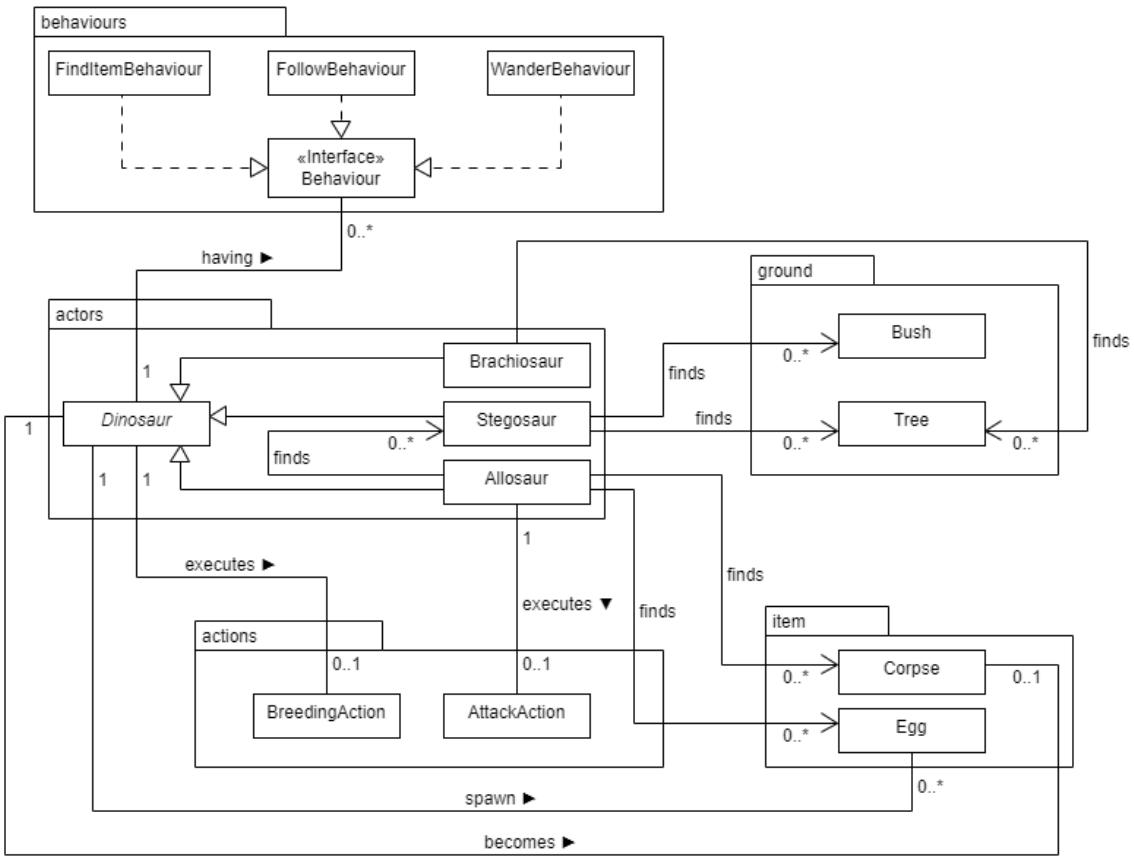
GROUP: TUTE08TEAM2

DARREN YEE JER SHIEN, NG YI JIE

FIT 2099 Assignment 2 Design Rationale

TUTE08Team2

Classes for Hungry dinosaurs, Brachiosaur, Allosaurs, Breeding, Death



Dinosaur (package game.actors)

- Responsibility: To handle all the different dinosaurs in the game. Contains methods such as **findFood** (handles looping through the map and finding the food), **findNearest** (which helps to find the nearest available dinosaur to mate) etc. The dinosaur class is an abstract class which will be extended by the other three different types of dinosaurs.
- Justification: This is done to make sure we reduce dependency for each of the different dinosaurs due to most methods being similar but with different values which goes in line with the principle of RED.
- Relationship with other classes:
  a) Association with **Egg**: 1 dinosaur lays 0 to many eggs during the whole game.
  b) Association with **Corpse:** A dinosaur will become a corpse once it is unconscious.
  c) Association with **Behaviour**: Directed association.
  d) Association with **BreedingAction** since it will be called once the dinosaur is above a certain food level.
  e) Extends **Actor**.

Stegosaur (package game.actors)

- Responsibility: Changes made so that it now extends the new **Dinosaur** class. It creates a **Stegosaur** object in a location on the map. Also modifies the specific details such as their food levels (50/100) and the different actions and behaviors within the class. using the methods and constructors of the **Dinosaur** class.
- Justification: This fulfills the DRY principle since most of the methods of the different dinosaurs are repeated. The ones that differ are overwritten within this class.
- Relationship with other classes:
  a) Extends **Dinosaur.**
  b) Directed Association with **Bush** since they will look through the bush to find fruits.
  c) Directed association with **Tree** since they will look though the tree to find fruits.

Brachiosaur (package game.actors)

- Responsibility: Creates **Brachiosaur** object in a location on the map. Also modifies the specific details such as their food levels (100/160), death etc. using the methods and constructors of the **Dinosaur** class.
- Justification: This fulfills the DRY principle since most of the methods of the different dinosaurs are repeated. The ones that differ are overwritten within this class.
- Relationship with other classes:
  a) Extends **Dinosaur**.
  b) Directed Association with **Bush** since they will look through the bush to find fruits.
  c) Directed association with **Tree** since they will look though the tree to find fruits.

Allosaur (package game.actors)

- Responsibility: Creates **Allosaur** object. Also modifies the specific details such as their food levels (50/100), death etc. using the methods and constructors of the **Dinosaur** class. Some methods are different within the Allosaur class due to their unique behaviors such as being able to find corpse, attacking other dinosaurs etc.
- Justification: This fulfills the DRY principle since most of the methods of the different dinosaurs are repeated. The ones that differ are overwritten within this class.
- Relationship with other classes:
  a) Extends **Dinosaur**.
  b) Directed association with **Corpse** as they find look for Corpse items around the map.
  c) Directed association with **Stegosaur** since they will find Stegosaurs to attack once they are hungry.

BreedingAction (package game.actions)

- Responsibility: Handles the conditions for dinosaurs to breed (i.e., how long they need to mate etc.). This is done as an action since it is not a guarantee that the player will invoke it every round unlike the behaviors.
- Justification: This reuses the Actions class and it's execute method that exists in the engine to reduce dependency which goes in line with the principle of RED. Will be called in the playTurn method in the player class.
- Relationship with other classes:
  a) Extends Actions

AttackAction (package game.actions)

- Responsibility: Handles the conditions for dinosaurs to breed (i.e., how long they need to mate etc.). This is done as an action since it is not a guarantee that the player will invoke it every round unlike the behaviors.
- Justification: This reuses the Actions class and it's execute method that exists in the engine to reduce dependency which goes in line with the principle of RED. Will be called in the playTurn method in the player class.
- Relationship with other classes:
  b) Implements Actions

FindItemBehaviour (package game.behaviours)

- Responsibility: Iterates through the entire map to find the closest instance of the specific fruit types that the dinosaur can eat. This is done as a behavior since it represents a kind of objective the action can have (in this case finding food items around the map). Will be called in the playTurn method in the dinosaur's class.
- Justification: This reuses the Behavior class that exists in the engine to reduce dependency which goes in line with the principle of RED.
- Relationship with other classes:
  a) Implements **Behaviour**.

FollowBehaviour (package game.behaviours)

- Responsibility: Makes the dinosaurs move towards dinosaurs of the same type that are available for breeding. This is done as a behavior since it represents a kind of objective the action can have (in this case following another dinosaur around the map). Will be called in the playTurn method in the dinosaur's class.
- Justification: This reuses the Behavior class that exists in the engine to reduce dependency which goes in line with the principle of RED.
- Relationship with other classes:
  b) Implements **Behaviour**.

WanderBehaviour (package game.behaviours)

- Responsibility: Makes the dinosaurs move around the map with no target. The program will decide whether to wander around or look for food randomly if the dinosaur is lower than a certain food level. This is done to ensure that the dinosaur can have enough hitPoint to breed with other dinosaurs. This is a behavior since it represents a kind of objective the action can have (in this case wandering around the map). Will be called in the playTurn method in the dinosaur's class.
- Justification: This reuses the Behavior class that exists in the engine to reduce dependency which goes in line with the principle of RED.
- Relationship with other classes:
  c) Implements **Behaviour**.

Tree (package game.ground)

- Responsibility: Contains an array list that will store the created fruit objects.
- Justification: Extends ground to use its built-in methods such as tick in order to simulate rounds passing.
- Relationship with other classes:
  a) Extends **Ground.**

Bush (package game.ground)

- Responsibility: Allows Brachiosaur to destroy bushes within a certain chance if they are standing on the location on the map.
- Justification: This reuses the Ground class that exists in the engine to reduce dependency (RED).
- Relationship with other classes:
  b) Extends **Ground.**

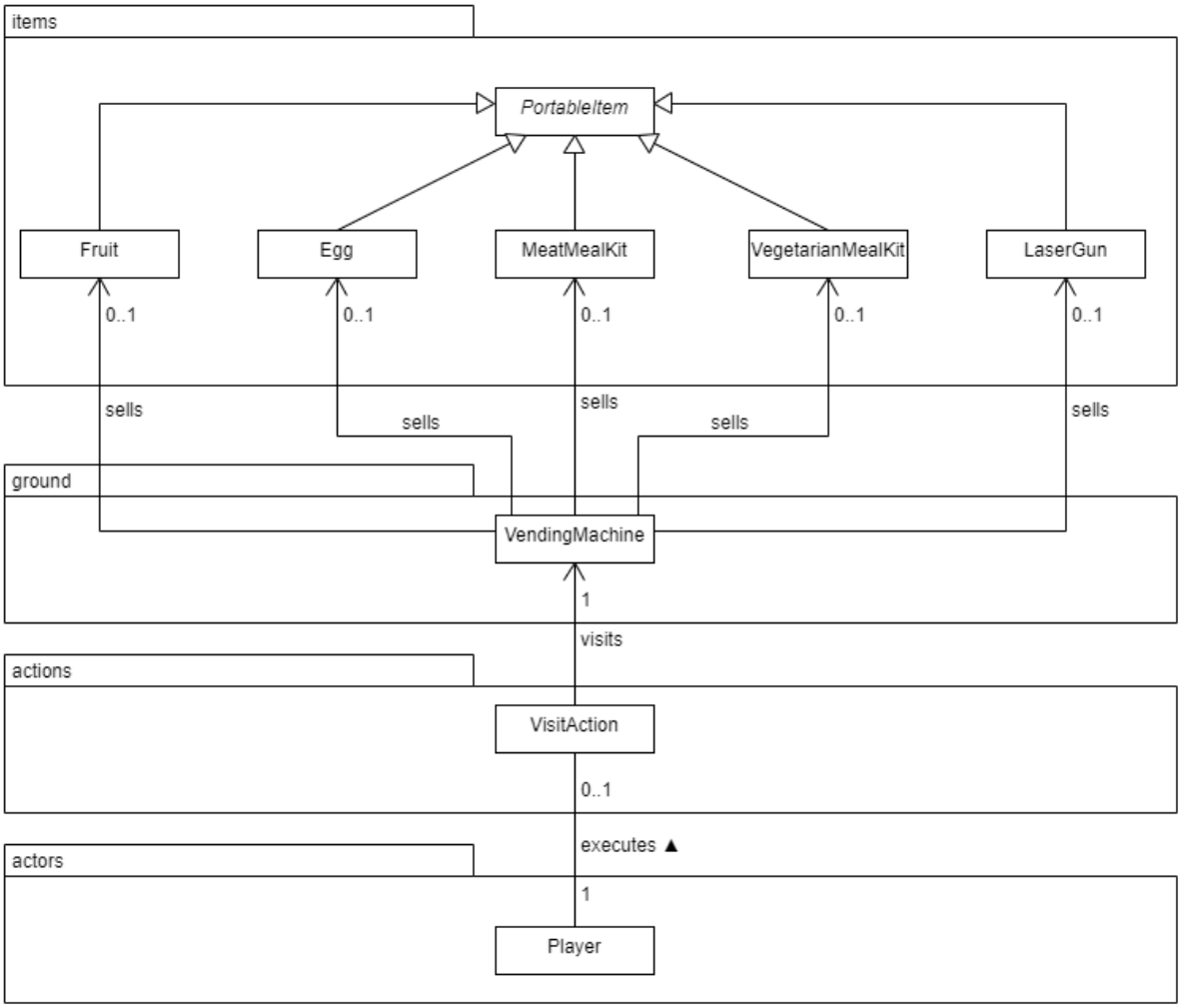
Egg (package game.items)

- Responsibility: Creates an egg object using the methods in PortableItems class. Dinosaurs use this class to create egg objects which will be placed in the map and will hatch to become adult versions of themselves. Allosaurs also find for egg objects within the map since they can eat them to gain hitPoints.
- Justification: Used universally by all three types of dinosaur to reduce dependency which goes in line with the principle of RED.
- Relationship with other classes:
  a) Association with **Dinosaur**: A dinosaur can breed 0 to many eggs within the course of the whole game.

Corpse (package game.items)

- Responsibility: Will store dead animals on the map once they are unconscious for a certain number of rounds. This will be represented by the displayChar '%' on the map. Corpse will disappear after a certain number of rounds have passed.
- Justification: This reuses the PortableItem class that exists in the engine to reduce dependency (RED).
- Relationship with other classes:
  c) Extends **PortableItem**

Classes for VendingMachine



VendingMachine (package game.ground)

- Responsibility: The "shop" for the game which allows the user to buy items as specified (such as fruits, weapons, eggs etc.). Upon user choosing to buy, an object of the item selected will be created and added into the user's inventory using the getInventory method given in the **Actor** class located in the engine . Weapon Items will be created using the existing **WeaponItem** class within the game's engine.
- Justification: The reason why we implemented it by using different classes for different items is because it will be more scalable if we were to add more items in the future.
- Relationship with other classes:
    a) Directed Association with **Fruit**: A vending machine sells 0 to many fruits during the whole game.

b) Directed Association with **Egg**: A vending machine sells 0 to many eggs during the whole game.
c) Directed Association with **MeatMealKit**: A vending machine sells 0 to many meal kits during the whole game.
d) Directed Association with **VegetarianMealKit**: A vending machine sells 0 to many meal kits during the whole game.
e) Directed Association with **LaserGun**: A vending machine sells 0 to many laser guns during the whole game.
f) Association with **VisitAction**: Handles the action of player reaching the specific location in the map that contains the VendingMachine. The action will call the VendingMachine each time the actor decides to go into the shop

Egg (package game.items)

- Responsibility: Can be bought (created) using the **VendingMachine**. Will breed a baby dinosaur once turn count is met.
- Justification: RED since it uses similar method as item class.
- Relationship with other classes:
  a) Association with **Dinosaur**: 1 egg can only breed 1 dinosaur during the whole game.
  b) Extends **PortableItem**

MeatMealKit (package game.items)

- Responsibility: Creates a **CarnivoreMeal** object that will be added into the **Array List of the user's inventory**. Can be used to feed Carnivorous Dinosaurs to restore hunger level to its maximum.
- Justification : Reuses methods from PortableItems which reduces dependency.
- Relationship with other classes:
  a) Extends **PortableItems**

VegetarianMeal (package game.items)

- Responsibility: Creates a **VegetarianMeal** object that will be added into the **Array List of the user's inventory**. Can be used to feed Herbivorous Dinosaurs to restore hunger level to its maximum.
- Justification: Reuses methods from MealKit which reduces dependency.
- Relationship with other classes:
  a) Extends **PortableItems**

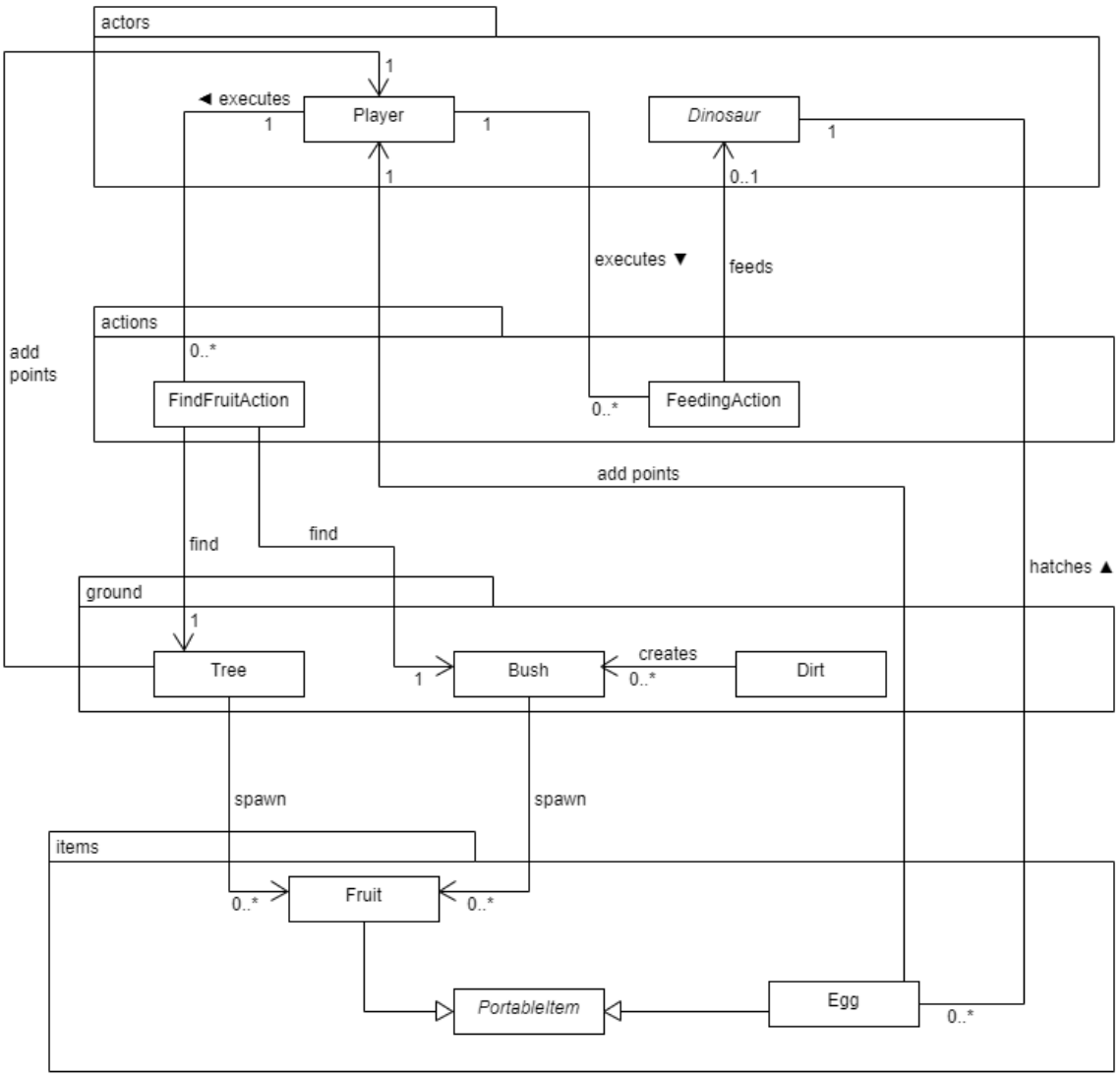
Fruit (package game.items)

- Responsibility: Creates a **Fruit** object using the methods in Item class. Used to feed Herbivorous animals to restore hunger level. Also used to handle rotten level of fruits. It can be created by **VendingMachine.**
- Justification: RED since we reuse the methods in the PortableItems class such as tick etc.
- Relationship with other classes:
  a) Extends **PortableItems**

VisitAction (package game.actions)

- Responsibility: Handles the action of player visiting the vending machine in the map. Will call the VendingMachine method to create and add the specific items into the player's inventory once bought. Will be called in the playTurn method in the player class.
- Justification: RED since we reuse the action class given by the engine and will be called in the playTurn method in the player class.

Classes for dirt, trees and bushes and ecoPoints.



Dirt (package game.ground)

- Responsibility: Changes made so that it can handle the **bush** objects that we need to create. Will have a certain probability of creating it. Will call the setGround method within the ground class to set the ground as the bush once the condition (probability) is met.
- Relationship with other classes:
  a) Directed association with **Bush**: 1 dirt can spawn no or a bush object during the whole game.

Tree (package game.ground)

- Responsibility: Creates **Trees.** Will have to change it to handle the creation of fruits (the percentage it spawns: 50%) and the percentage in which it will fall (10%). Points will be added to player once Tree is spawned. When it falls, we will add it to the inventory of the location of the Tree whereas if it is still inside the Tree, we will store it into an ArrayList.
- Relationship with other classes:
  a) Directed Association towards **Player** since we need to know who to add the ecoPoints to.
  b) Extends **Ground**

Bush (package game.ground)

- Responsibility: Creates bushes on top of **dirt**. Will have a chance of spawning **fruits**. Fruits that are spawned will be added into the ArrayList of the bush.
- Justification: Points will be added to player once fruit is spawned. We extend ground instead of having a relationship between Bush and Dirt to reduce dependency.
- Relationship with other classes:
  a) Extends **Ground**
  b) Directed association with **Fruit**. A bush will spawn zero to many fruits over the course of its operation.

Fruit (packages game.items)

- Responsibility: Creates a **Fruit** object using the methods in Item class. Used to feed Herbivorous animals to restore hunger level. Also used to handle rotten level of fruits. It can be created by **Bush or Tree.**
- Relationship with other classes:
  a. Extends **Item**

Player (package game.actors)

- Responsibility: When player walks over a location containing a tree or bush. They can choose to pick up fruit from the tree or bush, there is a chance of failing (certain percentages). It can also feed dinosaurs with MealKits purchased from the VendingMachine.
- Relationship with other classes:
  a) Extends **Actor**
  b) Association with **FeedingAction** where a player can call one to many FeedingActions over the course of the entire game (will be called every time a dinosaur is either to the North, South, East, or West of the player).
  c) Directed Association towards **FindFruitAction** which allows the player to search through the current location and harvest fruits with a chance of failing.

Dinosaur (package game.actors)

- Responsibility: Dinosaurs can create eggs through breeding and can also be fed through the FeedingAction class.
- Relationship with other classes:
  a) Extends **Actor.**
  b) Association with **Eggs** where a dinosaur can create zero to many eggs throughout the course of the game.

FeedingAction (package game.actions)

- Responsibility: Feeding action can be called upon by the player once they a dinosaur is within a certain range of them. They can choose to feed them a MealKit or fruits. Once fed, the class will invoke the removeItemFromInventory method from the actor class to remove the items fed to the dinosaurs from the player's inventory. This is done as an action since it is not a guarantee that the player will invoke it every round unlike the behaviors. Will be called in the playTurn method in the player class.
- Relationship with other classes:
  c) Implements **Action**
  d) Association with **Player**: this class will call the player to remove items from its inventory every time a feeding action is made.

FindFruitAction (package game.actions)

- Responsibility: Will be called upon by the player if they choose to pick/ find for fruit in its current location. There is a chance of failing where if the player fails, is message will be displayed and if they passed, a fruit will be added to the player's inventory. This is done as an action since it is not a guarantee that the player will invoke it every round unlike the behaviors. Will be called in the playTurn method in the player class.
- Relationship with other classes:
  e) Implements **Action**
  f) Association with **Player**: this class will call the player to add the collected fruits into the player's inventory.

Simple summary of all the main functionalities stated in the assignment specification and how they are handled in this application.

With all the changes that we have made towards the classes, it will allow us to deliver all the required functionalities of the assignment. Let us look at it in the context of all the important operations.

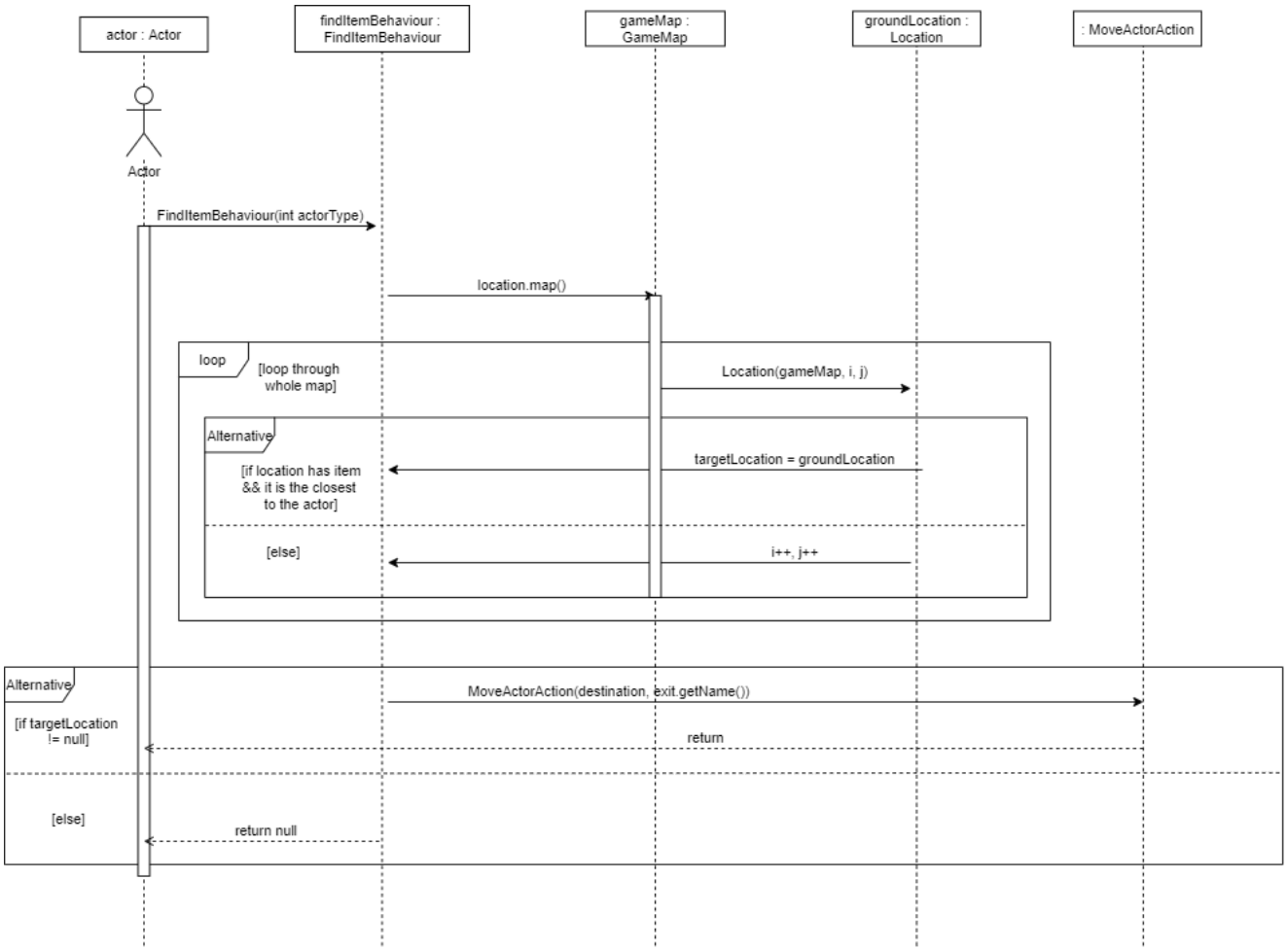
- **Dirt, trees, and bushes**:
  a. Dirt has a chance of spawning bushes using the setGround method given in the Ground class. What this does it allows us to replace the existing location into a Bush object. In addition to that, Bush and Tree can also spawn Fruit objects and will store them into their own ArrayList which can be access be the Actors. (Tree will store the ones that drop to the floor into the location's inventory list).

- **Hungry dinosaurs, Brachiosaur, Allosaurs, Breeding, Death**:
  a. The Dinosaur abstract class stores all the important methods that are used universally between the different types of dinosaurs. For each of them they will either reuse the methods (such as checkEgg which checks whether the eggs can be hatched, checkBaby which checks whether the dinosaur is an adult or baby) or overwrite them. The class that overwrites most of the methods is Allosaur since most of its operations are different from those of the other two. Once they are above a certain food level, they will loop through the map and find the nearest valid partners and will proceed to breed. Once they are unconscious (due to fighting or starving), they will become corpse and will be store in the inventory of their current location.

- **VendingMachine**
  a. A new vending machine ground type will be added onto the map using the **VendingMachine** class. In this class, we will be able to create different objects that that a player can buy (such as **eggs**, **weapons**, **fruits** and **MealKits** (**MeatMealKit** for Carnivores and **VegetarianMealKit** for Herbivores)). They will take in the option from the player's input and create / add the items into the user's inventory.

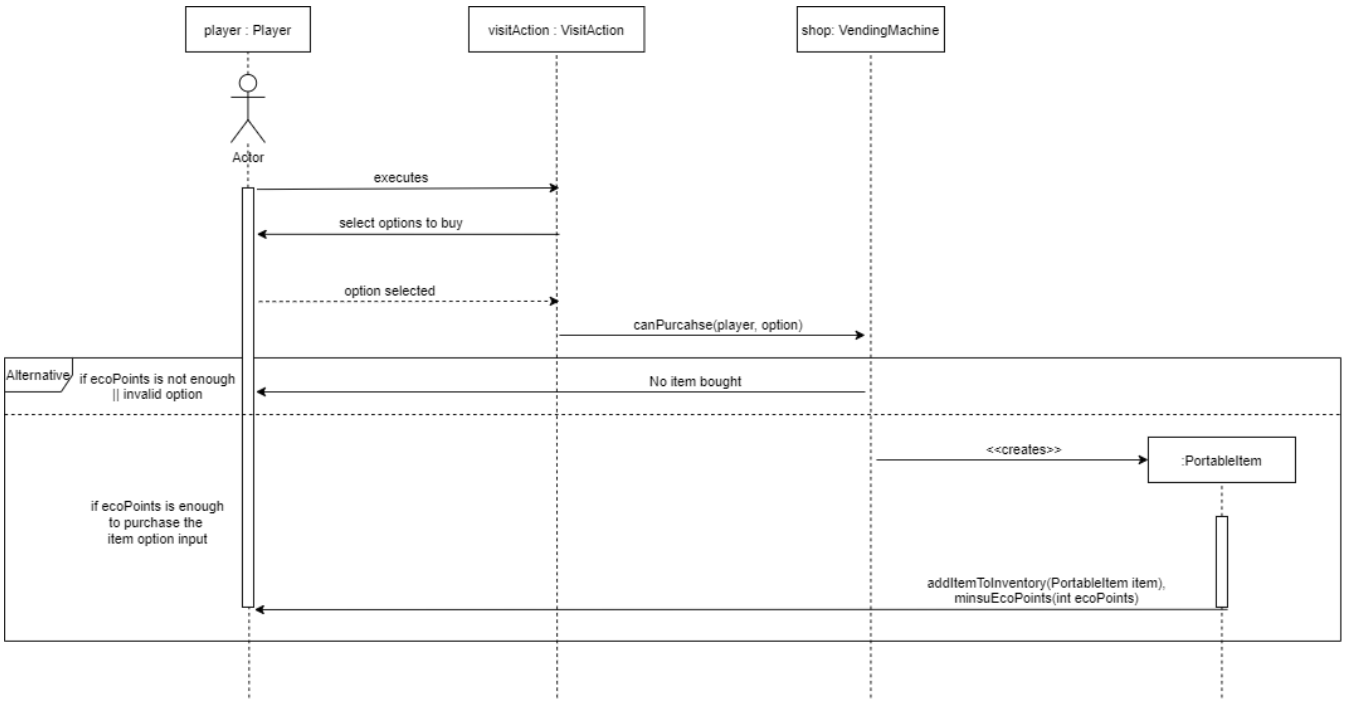
- **EcoPoints**
  a. Once a **Fruit** is created or an **Egg** is hatched, it will invoke the add points method from the **Player** class which will add certain points for when each action is made. These points can be retrieved by the **VendingMachine** and then points can be spent to purchase different products.

UML Sequence Diagram for FindItemBehaviour



- This sequence diagram describes the actor's action to find food. When an actor is hungry and there aren't any food sources around them, they will invoke this method to loop through the map to find the closest target to them to move towards them. This will continue after we finish iterating through the entire map that the actor is using the **getXRange** and **getYRange**. If they don't find any valid targets to move towards, they will exit and return null to the **MoveActorAction**. Once they find the nearest location, they will exit the find item behavior and invoke the **MoveActorAction** to move the actor towards the specific location for them to able to get the food in the area.

UML Sequence Diagram for Vending Machine



- Upon the player reaching the **VendingMachine** location on the map, the Player can choose to invoke the **visitAction** class and they will be presented with a menu containing all the available items. When the user has less **EcoPoints** needed, it will proceed to the next round and a message will be displayed on the console stating that the player did not buy anything.
- This is implemented this way reduce cases where the user keeps trying to buy items without having enough EcoPoints which goes in line of the principle of failing fast (FF).
- For each of the items selected, the vending machine will check if the player's **EcoPoints** attribute whether it is enough to purchase the specific item. If not, the request will be rejected, and the program will back to the player's **playTurn** method and display a message stating that the player did not purchase anything. If the player has enough **EcoPoints**, **VendingMachine** will invoke **creation methods** which will create the specific items and add them into the **player's inventory**. The player's **EcoPoint** will then be deducted according to the items purchased.