Design Rationale

1. Tree and Bush are connected to Fruit even though Location has its own item attribute

Tree and bush will store fruit that it has growing on itself and the Location class will store items that are on the ground. This enables us to distinguish between the ground types Fruits are located in.

2. Dinosaur is an interface

Each type of dinosaur has different attributes and interacts differently with the rest of the world (e.g. how it eats fruit). It is easier to having an interface to ensure each dinosaur has the required functionality expected from each dinosaur, but the exact details can be separated between each type of dinosaur.

3. Dinosaurs store an age tracker to determine whether it is a baby or not

This will minimise repeated code (DRY) as the baby dinosaur is very, very similar to the adult dinosaur.

4. Egg parent class for each type of egg

Each egg is very similar. This will minimise repeated code (DRY) and make it easier to maintain.

5. Location has an ArrayList of items at the location but Bush and Tree have their own attribute of fruit. Tree also has an attribute for dropped fruit

This is to make it easier to understand where the fruit is located as depending on where the fruit is located, different actors will interact with the fruit differently. For example, stegosaur cannot eat fruit from the tree but can eat it if it is dropped from a tree. Technically, location could store dropped fruit however, it is easier to understand the code if Tree has its own dropped fruit attribute. In order to check if a location has dropped fruit under a tree and dropped fruit was stored under Location, it would involve checking the location’s ArrayList of Items as well as checking if there is a tree at that location which is more clunky than if we were to have a dropped Fruit attribute in Tree.

6. Each item has its own location stored as an attribute

This is to make it easier to travel to an item such as to a corpse. Items do not travel as much as actors so the location can just be stored unlike for Actors.

7. Util Class incorporating many different functions.

A Util class is useful to store methods and constants that are shared by several classes since they can just be imported by the classes that use them. This follows the Do not Repeat Yourself (DRY) design philosophy, and if we wish to make changes to any of theses methods/constants, we only need to change them in one place rather than in many places if we were to not use a Util class. Many of the functions used in the util class are functions that are generally very useful for multiple classes such as NearestItem and NextToGroundType. Another benefit of putting the functions in the util class is that it means any further classes or classes that we want to incorporate later on can use these functions very easily meaning expanding the project later on is much easier.

8. Vending Machine inherits from Ground

We believe Vending Machine should be treated like a Ground rather than an Item because it acts much more like a set object on the map (like Ground) rather than an individual item that can be moved and placed in the inventory. It is easier to make it a ground than an item and remove lots of capabilities.

9. There are classes for each type of Egg that inherit from the big Egg class rather than using the one Egg class and changing each instance’s attributes to meet the requirements of each type of egg.

It is easier to create classes for each type of Egg and have predefined attributes for each type of egg. This makes it easier to understand what the properties each egg has and makes it very easy to instantiate the right type of egg in the code when dinosaurs breed.

10. Corpse is an Item rather than Ground

This makes it easier to interact with and easier for Allosaurs to eat it with having to make multiple edits to the Ground. Items sit on top of the Ground so it is easy to change with affecting other objects.

11. The argument for IncreaseEcoPoints is a String rather than an action or an Int

EcoPoints will store a hashMap to match certain actions represented by strings with an int value of how many points that action is worth. Storing it as a hashMap will make it much easier to edit the values later and also makes the function very easy to use as all it requires is a string input rather than having to pass certain objects which could make it messy.

12. Implementing MateBehaviour rather than using just MateAction and FollowBehaviour

Utilising the behaviours mean we can group together multiple different method calls that lead to Actions being returned. This makes the code easier to understand.

13. Buy item action takes in parameter the item to buy selected on the menu

Options on the menu typically show different action classes that are possible for players to take. By taking in a parameter to buy an item from a vending machine, this removes the need to create an action for returning each possible vending machine item.