Design Rationale

1. 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.

1. 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. It is more data to store and update through all the functions however.

1. 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.

1. 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. The problem is that this ground is very different to other Ground types like dirt such as the actor should not be able to pass through it and items should not be dropped at that location. The class will need to be updated to ensure this functionality is maintained.

1. 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.

1. 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 without affecting other objects.

1. 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. It however does make it more difficult to follow how the classes interact and will have to be represented in a sequence diagram most likely.

1. PurchaseAction takes in user input to buy selected items from the menu

This will be the most clear method for the user to choose what to buy. This will be better than printing out all the options of goods to buy in the initial menu. Now it will work by the user first selecting they would like to use the vending machine and then another menu will ask what they want to purchase. This does mean that there are more steps involved in purchasing goods and the code is very long even though it is not very complicated.

1. EcoPoints has all static methods and static variables

EcoPoints refer to the points that are gained for different actions taking place in the world and are used by the player to purchase from the vending machine. It initially seems to make more sense to the player store an instance of EcoPoints. However, some of the actions that increase eco points do not involve the player at all (e.g. Fruit being produced by a tree) and if we were to have EcoPoints stored in player, the player would have to be passed through to tree. Having EcoPoints have static methods and variables means that it can be referenced by any class without having to go through player which overall reduces dependencies (RED). A negative of this is that this class is not as well protected due to it being accessible by everything.

1. Flora Class added

Bush and Tree have many functionalities that are very similar such as how it can grow fruits. To avoid repeating code (DRY), a flora class was created so that methods that are the same for both classes can just be inherited from flora. This can make the code harder to understand with more objects being used.

1. Made new GameStart class (removed World)

The old World class is no longer called by Application. The reason for this is that it did not have the functionality required for a more sophisticated game driver such as being able to count the number of turns and end the run function when a certain number of turns was reached. It was decided that a whole new class would be better as many of methods would have to be rewritten to account for the new functionality. Some of the code was copied but any methods that were not needed such as endGameMessage were removed. This means that the World class is now taking up space for no reason anymore however this did not outweigh making the added functionality much easier to code up and much easier to understand.