Assignment 1: Design Rationale by Group “A”
Gabriela Bayabos (31256120)
Teh Jian Xiang (31192084)

Allosaur:

Female Allosaurs will be the only types of Dinosaur with a “children” attribute. The children attribute will be an ArrayList of Egg’s. This was implemented as since the Allosaur are carnivorous, we want to prevent a mother Allosaur from eating its own Egg as soon as it has been laid. We also allowed the father of an Allosaur Egg to eat its own offspring. This did not present an issue as we’ve assumed that once breeding, a male Allosaur will travel elsewhere in the World. This consideration did not need to be applied to Stegosaurs as they are herbivores.

Dinosaur:

The Dinosaur class will be an abstract class which Allosaur and Stegosaur inherit from. Dinosaur’s will implement basic attributes such as type, stage (baby, adult) and gender. We chose to make this an abstract parent class to Allosaur and Stegosaur as there will be many similarities in methods and attributes between the two child classes, so this will help our code to be concise and abide by D.R.Y.

We will also add the attribute “can_breed” for all Dinosaurs. The can_breed will be of boolean type. This will be set to false for baby Dinosaurs as well as Female Dinosaurs who are pregnant with an Egg. Once a Female Dinosaur has laid an egg, or once a baby Dinosaur has reached adulthood, the “can_breed” attribute will be set to true.

Ground:

An ArrayList of type Object called items will be used to hold all the objects currently standing on a Ground at coordinates (15,3) for example. We chose to make it an ArrayList and not an Array since ArrayLists are mutable, and we can easily add or subtract objects from an ArrayList. Ground instances will have another 2 attributes: x and y coordinates. If a Map is created with dimensions 30 x 30, 900 Ground instances will automatically be instantiated starting with the coordinates (1,1), (1,2), (1,3) … (30, 30). These coordinates **must** be unique to avoid having duplicate locations on a Map.

EcoPoint:

EcoPoint will be the currency used throughout the game. We did not find it necessary to create an EcoPoint class because a single EcoPoint instance could simply act as $1.00, and would not need to have a value attribute unlike paper money in the real world. As such, we let EcoPoints be an attribute for the Player class that can simply be added to or subtracted from when a Player makes a purchase.

Egg:

Eggs will be laid 10 turns after a female Dinosaur has bred. We plan to add an attribute called type to indicate whether an Egg is of type Stegosaur or Allosaur. When an Egg hatches, we will create a baby Dinosaur instance based on the Egg type, and then delete the Egg instance. The baby Dinosaur will be created on the same Ground that the Egg was deleted from.

Assignment 1: Design Rationale by Group “A”
Gabriela Bayabos (31256120)
Teh Jian Xiang (31192084)

Design reasoning:
Initially we thought that Egg should be a part of the stage attribute for a Dinosaur instance such as egg, baby, adult. However, we decided against this because it did not seem fit to be able to purchase a Dinosaur instance with stage “egg” from a VendingMachine.
We also faced the issue that once a female Allosaur lays an Egg, it might simply eat it because they are carnivorous. To avoid this we decided to add the “parent” attribute to Eggs

The parent attribute can be “null” since not all Eggs will have a parent such as those purchased from a VendingMachine.

Food:

The following classes will all inherit from Food: Egg, Fruit, Grass, Hay, MealKit.
Design reasoning: We chose to create this inheritance because since all of the above can be purchased and eaten by Dinosaurs, they will all have the same attributes like cost and the amount they will increase a Dinosaur’s food level by when consumed.
The only reason we chose not to add Stegosaur to this list is because even though it can be eaten by an Allosaur, Adult Stegosaurs cannot be purchased from a vending machine.

Fruit:

The attributes Fruits will have (cost and increase_food_level_by) will be inherited from Food. Fruit instances will be deleted completely after 20 turns from lying on the Ground. This will be to mimic food rotting. This would help prevent Ground instances items from being over populated by rotten fruit and will help us save memory.

Grass:

Grass will be contained within the items list of Ground.
Grass that is harvested will be removed from the a Grounds items.
Grass instances will be deleted when any of the following happens:

- Grass is harvested by Player
- Grass is grazed on by Stegosaur

Design reasoning: We chose to delete the Grass instance when it is harvest by a player so that there would be no confusion on whether Grass should still remain on the Ground. Upon harvesting, a new Hay instance will be created and added to the Players inventory.

Similarly, we chose to delete a Grass instance when grazed on by a Stegosaur to mimic it being eaten by the Stegosaur and so that the Grounds resumes to being simply Dirt.

Hay:

Hay is created and stored within the JavaArray of Inventory whenever a grass instance is harvested by a player. It is also stored in the HashMap of the VendingMachine with its value

being 20 eco points. A Hay instance is deleted and removed from the players Inventory when being fed to a Stegosaur.

Design reasoning: A Hay instance is deleted completely during feeding to mimic food being eaten and digested. This increases the Stegosaurus’ food level by 20.
We initially thought of making Hay inherit from the Grass class but realised that this wouldn’t be necessary as we did not need to add methods to the Hay class that were similar to the Grass class.

Inventory:

Inventory contains items collected by the player. Each item consumes 1 space of the inventory. When grass is harvested by the player, hay is created in the player’s inventory.

Design reasoning: JavaArray is used because the size of inventory needs to be fixed to maintain the fairness of the game. Players can’t collect items when the inventory is full.

LaserGun:

Even though a LaserGun has very similar attributes and capabilities as the other food items in a VendingMachine, since a LaserGun is inedible, it will not inherit from Food. The LaserGun instances will have a constant cost attribute of 500. As it is not mentioned in the requirements whether a LaserGun can attack an Allosaur, for now we will assume that this is not possible.

MealKit:

The MealKit is a simple class that will inherit from Food. An additional attribute we will add to it is “type” to determine whether it is a vegetarian or carnivore MealKit. Based on this, we can determine whether a specific Dinosaur can eat a specific MealKit instance.

A MealKit instance will be deleted once it has been fed to a Dinosaur. If a Dinosaur already has a maximum food level, a MealKit will not be fed to a Dinosaur. This is to prevent Players from accidently wasting this item and their EcoPoints.

Player:

To mimic a roguelike game, players will be able to choose their own names and be equipped with EcoPoints and an empty Inventory when starting the game. The Player class will likely be the most extensive in the game alongside the Dinosaur type classes.

Stegosaur:

The Stegosaur class will inherit from Dinosaur. This is because we expect any type of Dinosaur to have the same capabilities such as eating, breeding, getting hungry, attacking etc.

Assignment 1: Design Rationale by Group “A”
Gabriela Bayabos (31256120)
Teh Jian Xiang (31192084)

Since Allosaurs have capabilities and methods that Stegosaurs do not, we decided to create 2 separate inheriting classes rather than simply adding a “type” attribute to the Dinosaur class.
Once a Stegosaur dies we do not intend to delete it right away, this allows time for Allosaurs to potentially feed on its carcass. Only after 15 turns and provided that an Allosaur is not currently eating a carcass, will we delete the Stegosaur instance. This is to prevent our map from being littered with the carcasses of dead Stegosaurs, mimicking a carcass turning to bare bones and disappearing into the ground.

Tree:

Tree’s can also be contained in the items of a Ground instance. This is to help us decide the likelihood of adjacent locations growing Grass. We initially thought of adding an ArrayList attribute to Tree to contain all of its Fruits. We decided not to implement it this way because there was no need to. Instead we simply decided to instantiate a Fruit instance (with a 40% chance of success) and add it into a Players inventory every time they try to pick from a Tree.
Tree’s will also contain an unlimited number of Fruit. We chose to make this number unlimited as it helps us standardize the chances of the Tree dropping Fruit and a Player picking Fruit.

Design reasoning: If we had set a number of Fruit contained in a Tree, the chances of a Player picking fruit would be dependent on the number of fruit available, and we wanted to prevent this number from constantly changing. The same reasoning was applied with the dropping of fruit.

TurnCount:

Turn count will be a way for the system to monitor how many turns have been made. We chose not to make a TurnCount belong to a Player in the case later on we choose to have multiple players on a single Map. This would cause problems for example: When a baby dinosaur grows into an adult after 30 turns, it would not be logical for these 30 turns to belong to the player who bought and hatched the Egg but rather 30 turns in the world (Map) the Egg is in.

As such, TurnCount will be updated anytime **any** player on the map makes a move.

VendingMachine:

Vending machine contains a number of items which are stored in a hashmap. Items in vending machine are unlimited as there is only one vending machine in the whole map.

Design reasoning: Hashmap is used because hashmap has a faster access to the price corresponding to the item.

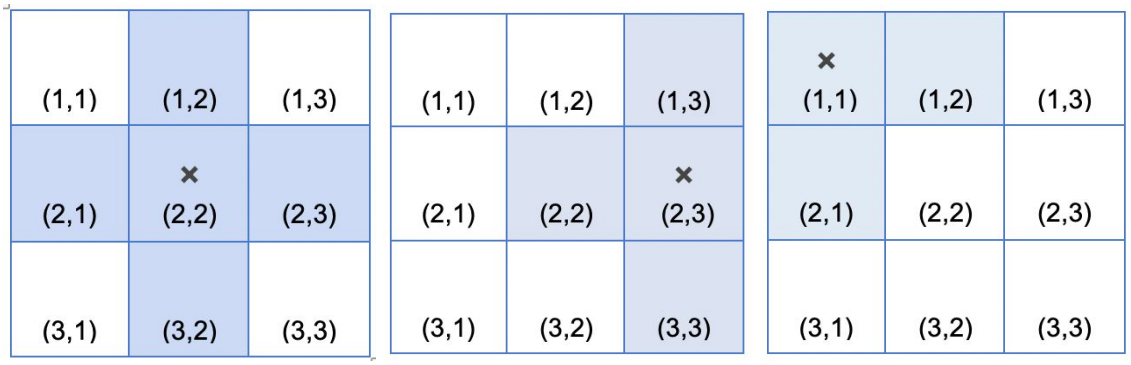
Assignment 1: Design Rationale by Group “A”
Gabriela Bayabos (31256120)
Teh Jian Xiang (31192084)

World:

When a player first starts playing the game, a World and GameMap will be created based on a set of dimensions eg. 30 x 30. For now we will let players choose the dimensions of the world. Once created, 900 Ground instances will instantly be created, we will loop through these and based on an algorithm will add Grass to them. To do so, we will need to ascertain whether a plot of Grass is next to another location with Grass.

The Grid:

If a player created a 3 x 3 World.



For each and every object with a game that needs awareness of its adjacent location, we will base it off of the Grid above. If “x” is at coordinates (2,2) it can only move among the highlighted blue squares. Likewise, to ascertain the probability of grass growing at “x”, adjacent locations will be evaluated to see if they too contain Grass or a Tree.