FIT2099 Assignment 1

By: Ong Jin Bin, Mohamed Nabhaan Ali

Design rationale

Design Goal:

- To implement Tree Class, Bush Class and Fruit class
- To implement attributes fruits for ground,trees and bushes
- To implement Hunger and Breeding
- To Implement Allosaur and Brachiosaur
- To implement vending machine

Vending Machine

The *Player* can access the *VendingMachine* class to buy *Product* to be stored in the *Inventory*. Hence, the dependency of *VendingMachine* to *Player*.

The *VendingMachine* has many *Product*. All items purchasable from the vending machine implements the *Product* interface, which has attributes **priceInEcoPoints : int** and methods *buy()*. The *Product* interface is implemented to ensure all the other classes implementing it have price as well as methods to define the transaction process (purchase, sell).

VegetarianMealKit and *CarnivoreMealKit* are classes implementing the *Product* interface, which allows them to be purchased by the user as well as allowing the flexibility of different methods/attributes for these 2 classes.

Weapon is an abstract class implementing the *Product* interface, which is the parent class for *LaserGun.* The *Weapon* class cannot stand on its own, but shares default methods (such as -foodLevel of *Dinosaur* after attacking them) as well as some undefined methods which are to be implemented by the children class, depending on use cases (such as only *LaserGun* is able to deal damage to *Stegosaur)*.

Dinosaur + Egg

Dinosaur is an abstract class, which is the parent class of *Stegosaur, Brachiosaur, Allosaur*. The *Dinosaur* class cannot stand on its own, but shares attributes (such as **turnsAfterUnconscious**, **gender** etc.) and methods (such as **playTurn()** and **getNearestMateLocation()** etc.) with its children class.

The *Allosaur, Stegosaur, Brachiosaur* classes inherit the *Dinosaur* class, of which they have individual attributes and methods (such as **turnsUntilNextAttack : int** attribute and **playTurn()** method for *Allosaur*).

StegosaurEgg, BrachiosaurEgg, AllosaurEgg are classes implementing the *Product* interface, which allows them to be purchased by the user as well as allowing the flexibility of different methods/attributes for these 3 classes. *turnsTillHatched : int* counts the number of turns before the eggs hatch, and this value decrements every turn.

Fruit

Starting off with the main point of discussion which is the implementation of fruits into the game. The final decision to implement fruits as a separate class rather than as an integer attribute of ground,bushes and trees was because we needed to track where a fruit is such as whether it is on a tree,bush or ground and if it was on the ground needing to keep track of how long till it will rot. With keeping all this in mind we chose to create the Fruit Class with the integer attribute age with the method tick().After further review we decided to not use the location variable rather use the inbuilt locations in the engine to tell us where the fruit is. Another implementation change we did was rather than use a decrementation method we decided to increment age using the tick() method which will update its age and in the GameMap is the fruit is too old (age>=15) it is removed from the map

Now the ArrayList of Fruit is only in the Tree and Bush classes without being in the Ground class. This is once again done due to being able to place fruit on the ground without the need of it having be in an Array on a Ground tile.The choice to use ArrayLists rather than a normal Java Array was made solely due to the fact that the number of fruits in an instance of any of the 2 classes Tree and Bush will never be fixed. ArrayLists gives an easy way to access the Fruits methods as well.

Tree

The implementation of the Tree class was a pretty straightforward requirement of the assignment . Other than the already mentioned fact of the it having the attribute of an ArrayList of Fruit.
We scrapped the idea of using the three methods : produceFruit(), dropFruit() and removeFruit() and instead used the inbuilt method tick() to do all the features we stated. Same As before for each circumstance required in the Tree a random number is generated and depending on that value does the Tree Produce and drop Fruit. The first part will see if a fruit is added to the list or not. The second part will also generate a random number which will check if a fruit instance in the list will drop to the ground and then remove it from the ArrayList and add it to the ground. This one Method does all the features necessary from the Tree

Bush

The implementation of the Bush class was also a straightforward requirement of the assignment. It still contains an ArrayList of Fruits but like Tree we use one method rather than the previously stated two methods:ProduceFruit() and removeFruit(). This one method Tick has the same functionality of the methods described in the Tree class excluding the drop functionality.

Ground Class Implementation idea was removed completely as it was unnecessary to edit it rather the key was in the Dirt Class

Dirt

The dirt class has 2 methods , Only one is relevant to this as the other just generates a random number. The dirt class uses the already existing tick() method to generate a bush on its tile depending on whether a Tree is not nearby as well as increased chance of a Bush spawning if there are more than 2 bushes on the adjacent square to the tile.

Lake

The lake class was implemented as a derived class from ground. The method tick was overridden from the parent class adds sips to the lake depending on whether the map rained or not. It also checks against a random number and depending on that number produces fish into the lake without exceeding the max amount

The second Map

In the application a new map was created. Then new exits were added to the locations within xrange at y=0 for the initial map and new exits added for location within xrange at y=24 for the newly produced map. The addition of these exits at these point allowed easy access/travel between the maps

More Sophisticated Driver

A new class which is essentially a copy of world was created called challenge world. It is essentially a copy of the world class with a slight change in the run method. The run method in this class only runs for the input amount of turns and ends afterwards.
Application class was edited to add 3 new static methods. The first method for the menu to select which gamemode to play. The second method initiating the sandbox mode of the game which makes the game run as normal . Lastly the final method is the method which initiates the challenge mode of the application by prompting for the amount of turns and the target eco points and then initiating the game which will run max for the amount of turns prescribed.