Design rationale

Dirt, Bush, Tree

Dirt, Bush and Tree all inherit from the Ground class. This is because they need to be properly represented on any Location with a special character, and don't need any of the capabilities that the other classes like Item or Actor have, such as moving and being picked up.

The Bush class has a special interaction with Branchiosaurs, where every time its tick() method is called, it will check if a Branchiosaur is standing on its location, and has a chance to be killed (be replaced with dirt) if there is one. This is the most straightforward way of implementing this interaction that doesn't involve editing the engine, and it's also the most modular, as Branchiosaur won't know anything about this interaction whatsoever, and any changes to the interaction will only affect one class (Bush).

Dirt has a chance to grow into a Bush, which will simply replace the ground at its location. Both Bush and Tree hold an ArrayList of Fruits and have a chance to produce one each turn during the tick() method call. They also have a chance once every tick to drop a Fruit from their array onto the Location where they exist (Where it'll be stored in the Location's ArrayList). This allows the Fruit's tick() method to be called using the Location's tick() method call to determine whether it is in an Actor's inventory or not, and thus whether it should rot. This also lets actors interact with the Fruit

Dirt and Tree both implement the hasFood() interface to show that they can be eaten from, and to ensure that they have the methods necessary to check their 'inventories'.

Fruit

Fruit inherits from the PortableItem class, mainly because it needs to be able to be picked up and added to a player's inventory. It also needs to be interactable for other Actors while on the ground and sold by the vending machine (which stores a Map of Items). The Fruit also doesn't have inbuilt Actions other than the ones provided by PortableItem. The FeedActions will be provided by the eligible Dinosaurs should the player have the Fruit in their inventory to allow them to feed the dinosaurs.

PickFruitAction

PickFruitAction inherits from the Action class for obvious reasons. It allows for Actors to pick/eat Fruits in Bushes or Trees. Limitations for each Actor are implemented in PickFruitAction's code as well, such as Stegosaurs being unable to eat from trees.

VendingMachine

VendingMachine inherits from the Ground class. It shouldn't act as an Item, and doesn't need the functionality that Actor provides (like playTurn()), and therefore Ground is the most suitable. It holds a Map of Items without multiplicities. The Items themselves are used as keys, and their prices are their values. This makes it so that theItems themselves do not have to know their prices, and prices can be different across different vending machines should the need arise.

BuyAction

BuyAction inherits from the Action class, and is returned by the VendingMachine class when a player is beside it. One BuyAction is returned per affordable Item. Each BuyAction knows the Item which the player will buy should it be executed, along with its price.

Corpse

Corpse inherits from the PortableItem class. This is so that they can be picked up and dropped. It also helps HungryBehavior identify it as food for Allosaurs

Egg

Egg inherits from the PortableItem class, mainly because it needs to be able to be picked up and/or bought from the vending machine, and doesn't need to be able to move or take actions. When the egg is ready to hatch, it will create a new BabyDinosaur in its location. Eggs have an enum attribute that determines what type of egg it is, and their time to hatch is also stored as an attribute according to their type. This is so that new subclasses don't need to be created just to store two different class attributes and no new methods.

MealKit

MealKit inherits from the PortableItem class, mainly because it needs to be able to be picked up and/or bought from the vending machine, and doesn't need to be able to move or take actions. MealKits are built similarly to eggs, having an enum type that determines what kind of MealKit they are (Vegetarian or Carnivore). This is so that new subclasses don't need to be created just to store a different type of the MealKit. The MealKit also doesn't have inbuilt Actions other than the ones provided by PortableItem. The FeedActions will be provided by the Dinosaurs should the player have the MealKit in their inventory.

LaserGun

The LaserGun inherits from the WeaponItem class, mainly because it needs the damage and verb attributes. It also needs to be able to be picked up, dropped and/or purchased from the vending machine. The AttackAction for attacks should be provided by the Actors, and so won't be provided by the LaserGun

Dinosaur and BabyDinosaur as Parent Class

The various dinosaurs Stegosaur, Allosaur, Brachiosaur, babyDinosaurs etc will inherit from a generalized parent class Dinosaur, (BabyStegosaur, BabyAllosaur, BabyBrachiosaur will then inherit from BabyDinosaur) which will extend class Actor. By having them inherit from a parent Dinosaur class, we can easily implement attributes and methods shared between dinosaurs that the child classes can inherit and override as needed. Not only does this reduce code duplication, it also makes it easier to maintain and debug.

The alternative, by having Stegosaur, Allosaur, Brachiosaur extend from actor will need the same attributes (e.g. isFemale, isPregnant) and it's methods to be implemented in all Dinosaur classes. This will make it harder to make changes to dinosaurs in general as you will need to make the change in all types of dinosaur classes.

This can be further specified by having BabyStegosaur, BabyAllosaur, BabyBrachiosaur inherit from BabyDinosaur for the same reasons as mentioned above. Babies share a lot of functionality like not being able to mate and growing into an adult dinosaur after some time.

Stegosaur, Allosaur, Brachiosaur and dinosaur babies as separate classes

By having Stegosaur, Allosaur, Brachiosaur, BabyStegosaur, BabyAllosaur, BabyBrachiosaur be separate classes that share an ancestor class Actors will make it easier initialise them with different characters ('A', 'a' etc.) on the map. It also allows for us to add specific behaviours that only pertains to a specific class. For example, only BabyAllosaur and Allosaur will have an AttackBehaviour.

Otherwise, complicated logic operation would need to be done to determine the dinosaur type and its methods/behaviours/attributes which can be avoided by just having them be different classes.

Behaviours

The behaviours will implement the Behaviour Interface as that will enforce each individual Behaviour to implement a getAction() method. This will make sure that each class will have the proper method to be used as behaviours.

AttackBehaviour

BabyAllosaur and Allosaur will be associated with AttackBehaviour as they can attack Stegosaurs when they're nearby. AttackBehaviour also has a dependency on Stegosaur and Brachiosaur as the behaviour will cause them to attack the former but not the latter. This behaviour will only attack stegosaurs when they're nearby and will not follow the stegosaur to attack them. Following stegosaurs in order to attack them will be implemented in

HungryBehaviour for Allosaurs and its babies.

HungryBehaviour

HungryBehaviour will be associated with every Dinosaur as they all can go hungry. It dictates how the Dinosaur will act when it’s hungry and what food type it eats. It is also dependent on Location, Ground, Exit, MoveActorAction and Item as it requires the first four to move toward a food source and the last to determine what food it can eat.

HornyBehaviour

HornyBehaviour will be associated with every Dinosaur except baby dinosaurs. It dictates how the Dinosaur will move to find a mate to procreate. It would be dependent on Location, Exit, MoveActorAction and BreedAction as the first two will be used to determine how the dinosaur moves and BreedAction will have them procreate and make the female pregnant.