DESIGN RATIONALE AND DOCUMENTATION

Here is an updated list of all the classes we have added to the codebase thus far:

a. Ambient
   i. game.ambient.Bush extends engine.Ground
   ii. game.ambient.Lake extends engine.Ground

b. Dinosaurs
   i. game.dinosaur.Dinosaur extends engine.Actor
   ii. game.dinosaur.Brachiosaur extends game.dinosaur.Dinosaur
   iii. game.dinosaur.Allosaur extends game.dinosaur.Dinosaur
   iv. game.dinosaur.Pterodactyl extends game.dinosaur.Dinosaur

c. Items
   i. game.item.Corpse extends game.item.PortableItem
   ii. game.item.Egg extends game.item.PortableItem
   iii. game.item.Fruit extends game.item.PortableItem
   iv. game.item.MealKit extends game.item.PortableItem
   v. game.item.LaserGun extends engine.WeaponItem
   vi. game.item.VendingMachine extends engine.Item
   vii. game.item.Fish extends engine.Item
   viii. game.item.Teleporter extends engine.Item

d. Actions and Behaviours
   i. game.action.BreedAction extends engine.Action
   ii. game.action.DrinkAction extends engine.Action
   iii. game.action.EatActorAction extends engine.Action
   iv. game.action.EatItemsAction extends engine.Action
   v. game.action.EndGameAction extends engine.Action
   vi. game.action.FeedActorAction extends engine.Action
   vii. game.action.VendingAction extends engine.Action
   viii. game.action.DropFruitAction extends engine.DropItemAction
   ix. game.action.PickFruitAction extends engine.PickUpItemAction
   x. game.behaviour.SeekBehaviour implements game.behaviour.Behaviour

The content from page 2 to 7 was written for Assignment 2. You may want to jump to page 8 for explanation of new functionalities in Assignment 3.

Introduction

The aim of this project is to create a dinosaur themed rogue-like game based on an existing system. There are multiple classes that are readily available in the system and together they form a base of a sophisticated game engine to be built from. The main objective is to add several new features and functionality that increase the overall playability of the game.

Ambient

To allow dirt squares spawn bushes by chance, a new class game.ambient.Bush that extends engine.Ground has been created. And game.ambient.Dirt.tick has been implemented to convert locations of game.ambient.Dirt to game.ambient.Bush accordingly. Instead of extending game.ambient.Dirt, game.ambient.Bush is made to extend engine.Ground directly because the bare dirt is represented by "." and we would like to assign a different displayChar, so ":" is used to represent a bush. To better improve realism, bushes should die and revert to bare dirt with all bush fruits destroyed when they get stepped on by big dinosaurs, in this case is the Brachiosaur, as we do not want the game to end up with a map full of bushes and bush fruits which in turn leads to a situation that the Stegosaur will always be self-sufficient and hence nearly immortal. So, game.ambient.Bush.tick has been implemented to meet this requirement.

Dinosaurs

Several new classes of dinosaurs will also be created to increase their overall variety, such as game.dinosaur.Brachiosaur and game.dinosaur.Allosaur that all extend game.dinosaur.Dinosaur. Previously we have suggested to create new baby classes. However, during code implementation, we realized that majority part of the baby and adult counterparts is performing the same tasks, except that their starting food levels are different and baby dinosaur should grow by age and do not possess the ability to mate. And this can be accommodated by differentiating them with their declared names and use this attribute to perform a check wherever required. However, there were still repetition of getters and setters of attributes in the three dinosaur classes, as well as repetition of codes to search for the nearest food source or potential mate involved in the playTurn methods. To solve this and avoid the violation of "Don't repeat yourself" principle, a new abstract class called Dinosaur has been created to contain the common codes and all three dinosaur classes are made to extend from it. Also, for better clarity in the display, different displayChar are used to represent adult and baby dinosaurs. During each turn, all dinosaurs should lose one hit point and since dinosaurs are Actor objects and the Actor class does not provide a tick method, the decrement of their food levels is done in the playTurn methods.

Portable Items

On the other hand, three new classes of items which can be picked up by a player and/or dropped onto the ground are created. To allow trees and bushes spawn fruits by chance, a new class game.item.Fruit that extends game.item.PortableItem has been created with implementation in game.ambient.Tree.tick and game.ambient.Bush.tick. All these fruits are differentiated by names to meet the requirement that different dinosaurs feed on different types of fruits and collected in the items list (ArrayList<Item>) of the respective location. Besides, game.item.Fruit.tick has been implemented to convert tree fruits to dropped fruits and remove any rotten dropped fruits from the items list accordingly. However, during code implementation, we realized that the engine code prioritizes the displayChar of item in a location when printing the map, which leads to messy display when the map is filled with fruits. To solve this, all Fruit objects are made to use the same displayChar as the Ground it currently sits on.

As dinosaurs will lay eggs after they mate and leave a corpse as they die, two new classes game.item.Egg and game.item.Corpse that extend game.item.PortableItem have been created. When a female dinosaur becomes pregnant after mating, a game.item.Egg object will be collected in its inventory list (ArrayList<Item>). This egg will be carried along until the time for delivery, such that it will be removed from the inventory list, dropped onto the ground, and then collected in the items list of the respective location. And the hatching of eggs into baby dinosaurs have been implemented in game.item.Egg.tick accordingly.

Similarly, a dinosaur will be removed from the game map upon its death and a game.item.Corpse object will be added to the items list of the respective location. Again, to better improve realism, different types of corpses should vanish after several game turns. So, game.item.Corpse.tick has been implemented to remove any corpses from the items list accordingly. Hence, all the corpses are differentiated by names, which also helps to meet the requirement that different types of corpses provide different food level to the Allosaur.

Eco Points, Vending Machine and Purchasable Items

Another new feature to be added is the "eco points", a currency-like collectible that enables the player to trade them in at the vending machine for various items. It is represented as an integer attribute in game.Player. Whereas, previously we have suggested to implement in game.Player.playTurn to keep track of and update the value of eco points based on different conditions during each game turn by making use of the method parameters. However, during code implementation, we realized that this is impossible and so we have resorted to increment the eco points in the part of code that conducts the relevant situation, by looping through the whole map to search for the player.

Apart from that, it is also required to place on the map a vending machine that sells various items for eco points. To implement this, a new class game.item.VendingMachine that extends engine.Item has been created. And a game.item.VendingMachine object is instantiated in game.Application and located randomly on the map. While for the purchasable items, two new classes game.item.MealKit that extends game.item.PortableItem and game.item.LaserGun that extends engine.WeaponItem have been created.

Regarding the selling prices, previously we have suggested to represent them as integer attributes in the involved classes of purchasable items. However, during code implementation, we realized that this is redundant as in the game.action.VendingAction to be invoked by the vending machine, a menu of purchasable items is displayed to the player, which should have included the selling price. Besides, as there are two different types of meal kits, they are differentiated by names to meet the requirement that they are sold at different prices in the vending machine, and likewise for the eggs. Likewise, the damage of the laser gun has been set randomly as an integer within the range of 50-100 to ensure it kills a Stegosaur in one or two hits.

Actions and Behaviours

When a dinosaur gets hungry as its food level drops below a certain value, it should move towards a food source and eat it. To implement this, previously we have suggested to create two new classes game.SeekBehaviour that implements engine.Behaviour and game.FeedItemAction that extends engine.Action. For this, game.behaviour.SeekBehaviour has been created and its getAction method should move a hungry dinosaur closer to a suitable food source. But for game.FeedItemAction, we have changed our idea. At first, game.FeedItemAction.execute was meant to perform the action of eating an item as well as feeding an actor. However, during code implementation, we realized that these two actions have different mechanism, as eating only involves an item and a dinosaur but feeding involves an additional third party which is the player. So, we have resorted to create two new classes game.action.EatItemAction and game.action.FeedActorAction to cope with eating and feeding respectively. Then, they should be added as one of the allowable actions of an object that invokes it. For game.action.EatItemAction, the object that invokes it should be the edible items, which include game.item.Fruit, game.item.Egg, game.item.Corpse and game.item.MealKit. Whereas for game.action.FeedActorAction, the object that invokes it should be a hungry dinosaur standing next to the player.

Now, back to where we began, to fulfil a hungry dinosaur, the playTurn methods have been implemented to return a respective game.action.EatItemAction when there is an edible item on the ground it is standing, or game.behaviour.SeekBehaviour.getAction to move it closer to the nearest suitable food source. However, during code implementation, we realized that Brachiosaur can actually eat as many fruits as it finds in a tree in a single turn. So, we have then created another new class game.action.EatItemsAction that extends engine.Action, which accepts as input a list of items instead of a single item and Brachiosaur's playTurn method has been implemented to return this action in place of game.action.EatItemAction.

In addition to the above, a hungry Allosaur, being a carnivore, should also move towards a living Stegosaur and attack it. To implement this, previously we have suggested to use the existing classes game.FollowBehaviour that implements engine.Behaviour along with a new nested class in Allosaur class called AllosaurAttackAction that extends game.AttackAction. However, during code implementation, we realized that there is no need to do so as the provided AttackAction can be reused by just adding a few lines to check if the attacker is an Allosaur then it deserves some hit points from a successful attack. This way, we have again avoided the violation of "Don't repeat yourself" principle.

Whereas, for the mating feature of a dinosaur, previously we have suggested to just return game.FollowBehaviour.getAction from game.Stegosaur.playTurn, game.Brachiosaur.playTurn as well as game.Allosaur.playTurn to move the respective dinosaur closer to a target of the same species and opposite gender once the food level condition is met. While this is still necessary, but we were missing the part when the dinosaurs are close enough to each other they should breed and produce offsprings. So, to implement this, a new class game.action.BreedAction that extends engine.Action has been created to allow dinosaurs to breed. To further ensure female dinosaurs do not mate with any other male dinosaurs while being pregnant, mateCoolDownPeriod & mateCoolDownCounter have been added as two new integer attributes in the Dinosaur class. mateCoolDownPeriod is meant to be the number of turns for which a mated dinosaur should wait before mating again. Whereas mateCoolDownCounter is a just a counter initialised as 1, decremented if the dinosaur is an adult, and set as the value of mateCoolDownPeriod after the mating process. While mating is only allowed when mateCoolDownCounter is 0, we have then prevented the baby dinosaurs from breeding.

In order for a player to feed a dinosaur with an edible item from his/her inventory, a mechanism implemented earlier which adds game.action.FeedActorAction as one of the allowable actions of the dinosaurs has already enabled this functionality implicitly, as this action will be added to the actions list passed from engine.World.processActorTurn to game.Player.playTurn, which will then display it to the user and have them select an action.

To allow a player interact with the vending machine, a new class game.action.VendingAction that extends engine.Action has been created. And game.action.VendingAction.execute has been implemented to display a menu of purchasable items to the user, deduct the eco points of the player accordingly and add respective item to his/her inventory. Likewise, it has been added as the allowable action of game.VendingMachine so that this action will be added to the actions list passed from engine.World.processActorTurn to game.Player.playTurn, which will then display it to the user and have them select an action.

Lastly, to implement the feature of player picking up a fruit by chance, our original thought was to reuse the existing classes engine.PickUpItemAction and override the fruit's getPickUpItemAction method to introduce the probability. However, during code implementation, we realized that this logic is not quite reasonable as this will mean that the player is made to try pick up a fruit even before the player selects an action. Thinking that it should be implemented in the action's execute method but since we are not permitted to modify the engine classes, it then becomes necessary for us to create a new class game.action.PickFruitAction that extends engine.PickUpItemAction in order to override the execute method.

Overall Responsibilities of New Classes

- game.ambient.Bush extends engine.Ground
    - To represent a bush with displayChar = “:”

- game.dinosaur.Dinosaur extends engine.Actor
    - An abstract class for creating various dinosaurs

- game.dinosaur.Brachiosaur extends game.dinosaur.Dinosaur (stats for baby counterpart)
    - To represent an adult Brachiosaur with displayChar = “B(b)”, starting hitPoints = 100(10) and maxHitPoints = 160

- game.dinosaur.Allosaur extends game.dinosaur.Dinosaur (stats for baby counterpart)
    - To represent an adult Allosaur with displayChar = “A(b)”, starting hitPoints = 50(20) and maxHitPoints = 100

- game.item.Fruit extends game.item.PortableItem
    - To represent a fruit with displayChar = (displayChar of Ground at its location) and will be instantiated with different names such as “tree fruit” and “bush fruit”

- game.item.Egg extends game.item.PortableItem
    - To represent a dinosaur egg with displayChar = “o” and will be instantiated with different names such as “stegosaur egg”, “brachiosaur egg” and “allosaur egg”

- game.item.Corpse extends game.item.PortableItem
    - To represent a dinosaur corpse with displayChar = “%” and will be instantiated with different names such as “stegosaur corpse”, “brachiosaur corpse” and “allosaur corpse”

- game.item.VendingMachine extends engine.Item
    - To represent a vending machine with displayChar = “v” and portable = “false”

- game.item.MealKit extends game.item.PortableItem
    - To represent a meal kit with displayChar = “m” and will be instantiated with different names such as “vegetarian meal” and “carnivore meal”

- game.item.LaserGun extends engine.WeaponItem
    - To represent a laser gun with displayChar = “/”, damage = a random integer within the range 50-100 and verb = “shoots”

- game.action.FeedActorAction extends engine.Action

    - To allow a player feeds a dinosaur with an edible item from his/her inventory, The edible item can be a fruit, an egg, a corpse or a meal kit.

- game.action.EatItemAction extends engine.Action
    - To allow a dinosaur eats an edible item from the ground. The edible item can be a fruit, an egg, a corpse or a meal kit.

- game.action.EatItemsAction extends engine.Action
    - To allow a Brachiosaur eats as many fruits as it finds in a tree in one turn.

- game.action.VendingAction extends engine.Action
    - To allow a player interacts with a vending machine.

- game.action.BreedAction extends engine.Action
    - To allow a dinosaur to breed.

- game.action.PickFruitAction extends engine.PickUpItemAction
    - To allow a player picks a fruit from a tree or a bush by chance.

- game.action.DropFruitAction extends engine.DropItemAction
    - To allow a player drops a fruit from his/her inventory.

- game.SeekBehaviour implements game.Behaviour
    - To allow a dinosaur seeks and moves closer to a food source.

Additional functionalities in Assignment 3

To further implement more functionalities and make the game even more interesting, the water system is introduced so that there will be natural forming lakes, precipitation and the dinosaurs will require water to survive. On the other hand, a new type of dinosaur is introduced, followed up by a new map and a more sophisticated game driver.

To create natural forming lakes, Lake class is added. Since a lake is a type of terrain, it is reasonable to extend it from engine.Ground to gain access to its attributes and methods. A Lake will contain 25 sips of water and 5 Fish when first created. Fish is a new class that extends engine.Item and will act as a food source of the Pterodactyl. Every turn there will be a chance to spawn a Fish in the lake and this will be implemented in the tick method of the Lake.

Dinosaurs will be able to drink water from the lakes to prevent themselves from dying of thirst. DrinkAction that extends engine.Action will be used in conjunction with logics in playTurn to allow dinosaurs to find a nearest lake and drink from it when it is thirsty. There will also be a chance to rain every 10 turn that refills the lakes by a certain amount and provides water to dinosaurs that are unconscious due to thirst. The functionality will be implemented in the Player class to find lakes and unconscious dinosaurs around the map to increase their water level. The Player class is one of the most suitable class to implement the functionality in as its playTurn method is unique that it will only be executed once every turn since there will only be one Player. This will remove the repetition of code and redundancy by removing the need to check in every lake and dinosaur every time it rains, as this keeps in line with the "Don't repeat Yourself" principle.

As for the new type of dinosaur, Pterodactyl that extends abstract class Dinosaur is created. To simulate it as a flying dinosaur, a string attribute is used to represent its condition, whether "flying" or "landed". Since it can possibly eat more than 1 fish caught from the lake when flying over, a similar implementation to enable Brachiosaur eats as many fruits as it finds in a tree is reused, such that an EatItemsAction will be returned from the dinosaur's playTurn method. Here again reusing the code is in line with the "Don't repeat Yourself" principle.

Pterodactyls can only breed and lay egg on the tree. For breeding, this condition is implemented by simply including it in the check within getAllowableAction method of the Pterodactyl before returning a BreedAction. Whereas to ensure the female Pterodactyl only lays egg on the tree, we have chosen to make it stay on the tree throughout her pregnancy. A DoNothingAction will be returned for those turns. To prevent her from getting unconscious out of hunger or thirst, we have specified an additional condition for breeding – other than being well-fed, the female Pterodactyl must be hydrated enough as well with water level above 40.

The longest distance a Pterodactyl can fly for is 30 squares. For this, an integer attribute called "flyingFuel" is created in the class, which decrements in every game turn and is used to determine the condition of respective dinosaur. While logically a flying Pterodactyl can land on a tree whenever it is over one, we have intentionally made it to not stop for a recharge unless the flying fuel is below 10. This is to increase its chance of being on the land, hence the risk of being exposed to an Allosaur and brings out a more challenging and interesting game.

Meanwhile, to accommodate with the need for Allosaur to devour the landed Pterodactyl, EatActorAction class is created instead of reusing AttackAction. For this, our justification is based on the several differences that they have, such that the eating is instantaneous, it will not leave a corpse, and since Pterodactyl is a relatively small sized dinosaur, Allosaur should be able to pounce on it accurately and eat it right away, unlike an attack which has a 25% chance to miss. So, it seems more appropriate to separate them, rather than overwhelming the execute method of AttackAction with many if-else statements.

To implement a second map and allow the Player to travel back and forth, a Teleporter that extends engine.Item will be created at the North edge and South edge of the first and second map respectively. Each of the teleporters will have a MoveActorAction that move the Player to the other map. The addition of the Teleporters will be done during initialisation of the game world in Application as it is counted as a part of the game world.

Finally, to create a more sophisticated game driver, another game menu loop will be added in Application such that when a game ends the menu will be displayed instead of terminating the program. The player will be able to select the desired game mode and if challenge mode is selected, the player will be required to enter a number to set the various goals to a factor of that value since only one character can be read. A new EndGameAction that extends engine.Action will be added to the list of actions of the Player to allow the player to end the game and return to the game menu anytime.

Apart from all the above, we have likewise updated a few changes to the previous implementation from Assignment 2. They are as follows.

- First, we have removed the FollowBehaviour class, which was intended to figure out a MoveActorAction that will take the actor one step closer to the targeted actor, particularly used in tracing a living Stegosaur for a hungry Allosaur to conduct an attack, as well as in finding potential mates for the well-fed dinosaurs. However, we have just realized that the internal logics for both FollowBehaviour and SeekBehaviour are the same, so we have decided to remove the former and replaced all involved implementations with the latter.
- Second, we have removed EatItemAction and replaced its usages with EatItemsAction for the same reason above, as a single item can be simply transformed into an array list and passed into the latter as an argument.
- Third, we have combined all major implementations within the playTurn methods of four dinosaur classes, as one version in the abstract class Dinosaur. To do this, we have created additional attributes to hold the dinosaur-specific values i.e. magic numbers, such as the age when a baby dinosaur should grow into its adult counterpart, the hit points when a dinosaur starts getting hungry, as well as the maximum turns allowed for being unconscious before dying. Then, everything can be generalized.

With the three changes above, we have successfully removed a lot of unnecessary repetitions in our codes, once again achieving the "Don't repeat Yourself" principle.