# Design Rationale

Name of the project: Assignment 1

Team name: PJs

Team members: Joleen Tang, Pasan Jayanetty.

Purpose of this project is to append and amend certain functionality to a text-based “rogue-like” game.

By the end of this project, the team expects all the required functionality to be operational.

The team has decided that the following **amendments** are necessary.

## **Grass**

* Grass to be an extension of Ground class. Grass will be created if:
  + At the start of the game (check for round one from tick()) each Dirt location will have ~2% chance to grow Grass. The resulting percentage will be determined by a random number generator. If the random number generated is greater than the probability (GRASS\_PROB) the location will set Ground to be Grass instead of Dirt.
  + Further, during each turn, locations that has Dirt as Ground type will check for getSurrounding(). This method returns a list of adjacent locations. Then it will check what the ground types are for these adjacent locations. Depending on the ground types, the probability of growing grass differs. If the probability is greater than GrassProbs, the location will change its Ground type to Grass instead of dirt.
  + Creating Grass will also add a Grass Item to the location’s Items list.
  + Grass.tick(location) will check if location has a grass item in its inventory; if it does, nothing will happen but if it doesn’t, Grass.tick() will set the location’s ground to be a new Dirt()
* Harvesting Grass:
  + If the location the actor is currently at has Grass as its Ground, then the actor will be able to perform HarvestAction. HarvestAction (which will be included in GroundInterface) will add Hay (which extends PortableItem) to the Player’s inventory. Further, the Grass Item in the Location’s Items list will be removed.

## **Fruit**

* Dropping Fruit:
  + Fruit extends PortableItem.
  + Fruit will have integer attribute rotCounter which is initialized as 0. After every turn incrementRotCount() will increment rotCounter. If rotCounter reaches 20, rotten() will return True.
  + In Tree.tick(location), a random number generator will generate a number. If this number is greater than the probability of dropping fruit(Tree.DROP\_CHANCE), a Fruit will be added to the location’s Items List.
* Picking Fruit:
  + A player has the ability to pickup fruit that is part of the location.
  + PickUpItemAction extends Action.
  + When the Player calls PickUpItemAction(Fruit), it will remove the Fruit from the player’s current location’s Item list and add it to the Player’s inventory using addItemtoInventory(Fruit)
* Harvesting Fruit/Tree:
  + Tree.HarvestAction (implemented from GroundInterface) will be called
  + Tree.HarvestAction will have a random number generator. If the number generated by this generator is greater than the chance of harvesting (Tree.HARVEST\_CHANCE) it will return Null back to the Player.
  + Else, it will add a Fruit to the Player’s inventory using Player.addItemtoInventory(Fruit).

## **Dinosaur**

* Dinosaur attributes:
  + Dinosaur will be implemented as an abstract class that both stegosaurs and allosaurs will inherit from
  + Each dinosaur will have the following attributes:
    - String gender – a String that contains the gender of the dinosaur.
      * Useful for breeding.
      * If the dinosaur gender is female, they will also have a LayEggCounter. To determine when an Egg will be hatched.
    - Int age – an integer variable that contains the age of a dinosaur.
      * A dinosaur is a baby until it reaches 30 years of age.
      * Every turn (determined by tick()) increases the age by 1.
    - Int foodLevel – an integer variable that contains the numerical value of food reserves.
      * Necessary to determine if a dinosaur is hungry or not.
      * Will increase/decrease depending on the Food the dinosaur eats and other Actions.
    - ArrayList edibleFood – an ArrayList that contains a list of all the Items (food) the dinosaur can eat.
      * Useful to check if the dinosaur can eat a certain item or if it can be fed a certain item.
* Breeding:
  + BreedBehaviour extends Behaviour.
  + BreedAction extends Action.
  + Egg will extend PortableItem.
  + Dinosaurs will have a layEggCounter. However, it will only be incremented if the dinosaur is female.
  + BreedBehaviour will be checked during each turn. It will check for suitable mates (stegosaurs that have more than 50 foodLevel, are of the opposite gender, and have an age over 30) on the map.
  + If there is a suitable mate in an adjacent square, BreedBehaviour will call BreedAction.
  + BreedAction will increment the female’s layEggCounter by 1.
  + Once the layEggCounter is greater than 0, it will increment by one during each Stegosaur.playTurn(). When it reaches 10, it will add an Egg to the mother’s current location’s item list.
  + Once layEggCounter reaches 10, the layEggCounter will be reset back to 0.
  + If there is no suitable mating partners adjacent to the dinosaur’s current location, BreedBehaviour will check for the nearest suitable dinosaur to breed using closestMate(). Then BreedBehaviour will call FollowAction and make the dinosaur move toward the nearest suitable mate.
  + Egg Item will be added to the Female Actor’s inventory.
  + Egg class will have integer attribute hatchCounter. If hatchCounter is equal to HATCH\_NUM then there will be a check to see if there is an adjacent location with no Actors in it. If there are no Actors, a new baby Dionosaur is created at one of the free locations. However, if there are no free adjacent locations, hatchCounter will be decremented so the check for free locations can try again next turn.
  + Then the Egg will be removed from the inventory of the Female Actor.
  + HATCH\_NUM for Stegosaur was decided to be 3 due to their short hatching time of around 24 hours.
* Eating dinosaur:
  + EatBehaviour extends behaviour.
  + EatAction extends Action.
  + EatBehaviour is called during each turn for each individual dinosaur (by Stegosaur.playTurn()). EatBehaviour checks for the foodLevel attribute of the dinosaur. If the dinosaur is hungry (if foodLevel < HUNGRY), it will check if the dinosaur has food in its inventory or in the current location, and print a message informing the Player that they are hungry as well as their current location (will tweak the frequency of this message later). If there are no items that are edible in either the location or in the inventory, then EatBehaviour will look for a location with edible food in the map by using MovetoItemBehaviour (extends Behaviour). MovetoItemBehaviour will return a location with an edible food item. Subsequently, EatBehaviour will move the dinosaur to the Location returned by MovetoItemBehaviour using MoveActorAction.
  + If the location or the Actor’s inventory includes edible food (Checked using the ArrayList edibleFood) EatBehaviour will call EatAction.
  + EatAction will remove the food item from the location or the Actor’s inventory. Further, EatAction will increment foodLevel depending on the type of food.
* Feeding Dinosaur:
  + If the Player is in an adjacent location to a dinosaur, the Player can call FeedAction which extends Action. FeedAction will let the Player choose an item from its inventory. Remove the said item from the inventory using removeItemInventory() and increase the dinosaur’s foodLevel depending on what was fed.
* Dying dinosaur:
  + During each turn, there will be a check performed to see if the foodLevel of the dinosaur is 0, if the foodLevel is 0, then tooHungryCounter will be incremented by 1.
  + If tooHungryCounter is 20, then dies(Stegosaur, GameMap) method will be called. This method will add a Corpse item (extends PortableItem) to the location’s (which is returned from GameMap.locationOf(Stegosaur)) item list, and then remove the dinosaur from the world using GameMap.removeActor(dinosaur).
  + If the foodLevel increases before the tooHungryCounter is 20, then the tooHungryCounter will be reset back to 0 (implemented as part of FeedAction())
  + Dinosaur.dies() may also be called in dinosaur.hurt() if its hit points drop below 0

### **Stegosaur**

* Upon research, the team has decided to make the HUNGRY constant to be 30. This was limit was chosen after realising that Stegosaurs used to eat “Hundreds of pounds” of plant material on a daily basis.
* Currently, the edible item list for stegosaurs includes fruit, grass, and vegetarian meal kit

### **Allosaur**

* Currently, the edible item list for allosaurs includes corpse, egg, and carnivore meal kit
* Allosaur attacking:
  + AttackBehaviour class extends Behaviour class.
  + AttackBehaviour will check if there are Stegosaur’s in adjacent locations to the Allosaurs. If there are Stegosaurs in adjacent locations, then AttackBehaviour will call AttackAction.
  + AttackAction will call Stegosaur.hurt().
  + This aggressive behaviour will be called before EatBehaviour
* Allosaur eating:
  + Allosaur eating and feeding will use the same methods as Stegosaur eating and feeding.
  + However, if there is no food available on the map, the Allosaur will hunt Stegosaurs.
* Allosaur Hunting:
  + If there are no food items in the Allosaur’s inventory and on the map, then HuntBehaviour (extends Behaviour class) will be called.
  + HuntBehaviour will look for Stegosaurs in the map and, if there are no adjacent stegosaurs move the Allosaur towards the closest Stegosaur using FollowBehaviour and MoveActorAction.
  + If the Allosaur is adjacent to a Stegosaur, the Allosaur will attack the Stegosaur using AttackAction.
  + If the Stegosaur dies from being attacked, a Corpse item will be dropped in the location the Stegosaur was in, which the Allosaur can then eat next turn.

## **Vending Machine**

* Eco points:
  + Player will have an integer attribute ecoPoints. This will be useful to check if the Player has enough credit to purchase different items from the Vending machine.
  + Certain actions will increment ecoPoints of the Player throughout the game. Each Action will have addEcoPoints() method which increases the Player’s ecoPoints respectively.
* Purchasing:
  + VendingMachine extends Ground. VendingMachine will be placed at the start of the game at a given location.
  + Items that can be purchased from the VendingMachine will have a static integer price.
  + BuyAction extends Action class.
  + If the Player is in the location where the ground type is VendingMachine, the Player is able to call BuyAction.
  + BuyAction will ask the Player to choose an item from a display of items.
  + There will be an Interface called VMItem. All the items purchasable from the VendingMAchine will be implementing VMItem. This is done so VendingMachine purchasable items can have a price attribute useful to check if the Player can afford the respective Item.
  + Once the item has been obtained from the Player, BuyAction will check for the price of the item by accessing the VMItem.price(). Then the price of the Item returned from VMItem.price() will be compared to the amount of eco points the Player has using ecoPoints. If the Player has more ecoPoints than the price of the item, then the item will be added to the Player’s inventory using addItemtoInventory(Item). Further, it will reduce the respective amount of eco points from the Player by decrementing ecoPoints attribute.
  + However, if the Player does not have sufficient eco points to purchase an item, printBrokeMessage() method will be called. This method will display a message indicating that the Player cannot buy the Item due to insufficient funds.
* Laser Gun:
  + LaserGun extends WeaponItem.
  + LaserGun will have a damage level of half the maximum hitPoints of a Stegosaur.

Our team will implement the aforementioned amendments over the next few weeks.

UML class diagram link:

<https://app.lucidchart.com/documents/edit/56bdc45f-2b87-4e27-8ef1-0c36aa98714a/o8dxuE6i5s.Q?shared=true#?folder_id=home&browser=icon>

UML sequence diagram link:

<https://app.lucidchart.com/documents/edit/5c2cb2b5-957d-4fa7-a288-df11f3244dec/.b.x1VccSAc7?shared=true#?folder_id=home&browser=icon>