Team: Tute01Team78

Team member

Abdulla Nawwaf Ali (31261949)
Yong Zi Ying (30885027)

Design Rationale [Assignment 3]

(Update Assignment 2 design rationale with new Assignment 3 requirement)

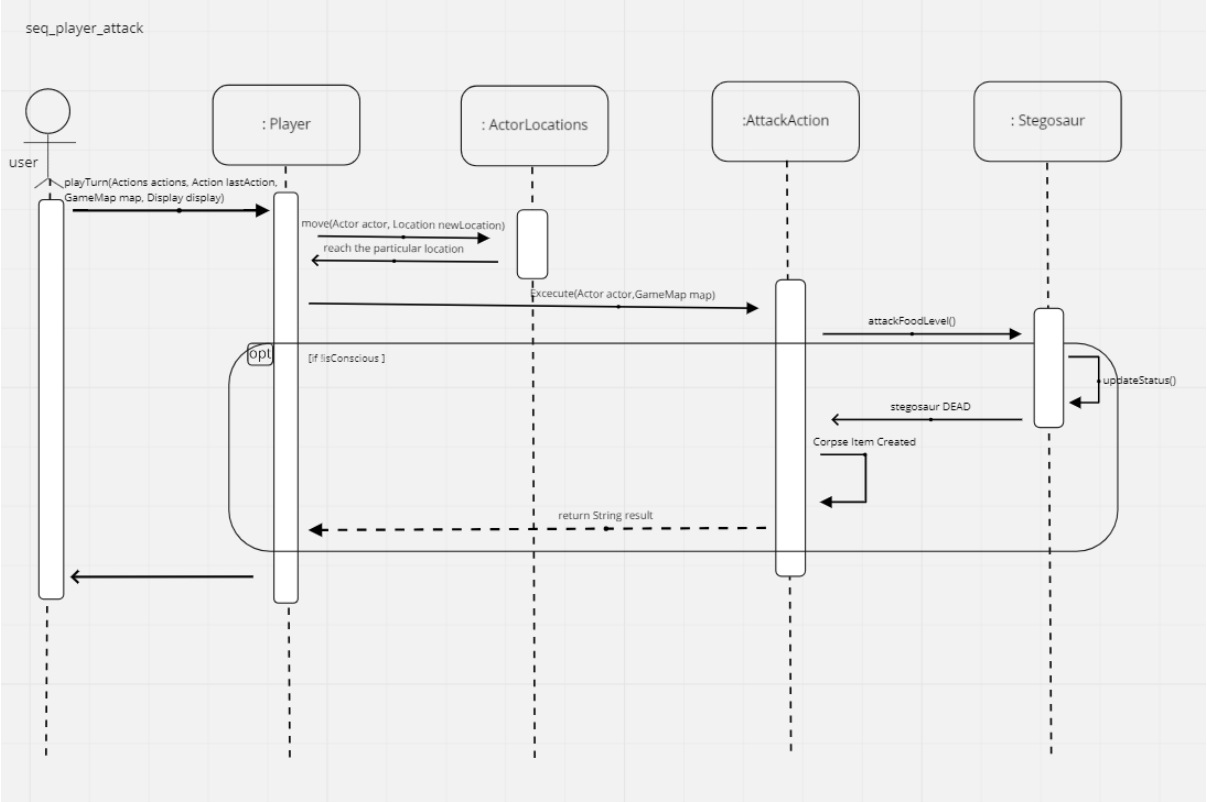
Abdulla Nawwaf Ali

Fighting

Implemented fighting using attack Action.
Edited Attack action to reduce the foodlevel of the dinosaur after the player attacks.

Team: Tute01Team78

Sequence diagram for his part [Assignment 3]



Team: Tute01Team78

Class diagram for his part [Black colour]

Team: Tute01Team78

Yong Zi Ying

Changes made will be highlighted with light yellow.
Removed design will be ~~strike through~~

Principle used in this assignment

- Multiple classes are created for each actor, ground and item. This is because each actor, ground and item have their own responsibility. (Single Responsibility principle). It is easier for me to maintain and debug the code
- The Action class in engine is used to extend for more requirements/actions for dinosaur or player instead of modifying it to fulfill all requirements/actions (Open/Closed Principle)
- Dinosaur abstract class is created in order to allow all dinosaur actors such as Stegosaur class, Brachiosaur class, Allosaur class, BabyAllo class, BabyStego class, and BabyBrachio class to extend it. All child classes can invoke all methods from its parent class (Dinosaur abstract). Meanwhile, extending the parent class may reduce the duplicated code. Reusing the code instead of copy pasting it. (Liskov Substitution Principle + DRY)
- Separate interface classes are created for EcoPoint and Probability, Both have different behaviour, when a class is implementing an interface class it should not be forced to depend upon interfaces that they do not use. (Interface Segregation Principle)
- All dinosaur actors class (child class) are depending on the Dinosaur abstract class. (Dependency Inversion Principle)
- SOLID principle is used to reduce code length smells(Bloaters) by decreasing the size of each class and length of each method. Dinosaur abstract class is used to reduce Dependency smells as all child classes such as Stegosaur class, Brachiosaur class, Allosaur class, BabyAllo class, BabyStego class, and BabyBrachio class extend it.

Dirt, trees and bushes

- In ProbabilityInterface class
    - An instance of Probability class is created in order to extend multiple classes in a class indirectly as each class can only extend one class only.

- In Probability class, it has some methods such as getChances() to generate a random number and check for the condition. If the random number is within the probability then return true, else false. getRainFallSips() method is used to calculate the amount of water that need to add into each lake by **multiplying 20** with random number in **range 0,1 to 0.6 inclusive**

The classes for plants such as dirt, tree, bush and wall will be grouped into a mapstuff package. At the very beginning of the game:

- In Application class
  - Indicate the gender for the pair of stegosaurus
  - **Create 4 new brachiosaurus on the map** and indicate their gender as well
  - Create a new *local variable* (dinoMap) for new DinoGameMap class (explained later for DinoGameMap class)
  - Loop the dinoMap, x range of dinoMap and y range of dinoMap to allow **1 % chance of the bush to grow** on any square of dirt and have **no chance for the bush to grow next to any tree**.
  - To check the neighbour tree count of the particular location, NeightbourTreeCount() method is invoke from a new DinoLocation class (explained later for DinoLocation class)
- In DinoGameMap class it *extends* GameMap
  - To add more implementation for GameMap class in engine, a new DinoGameMap class is created to avoid changes for GameMap class in engine
- In DinoLocation class it extends Location
  - To add more implementation for Location class in engine, a new DinoLocation class is created to avoid changes for Location class in engine
  - To calculate the neighbour tree count and neighbour bush count at any square of the dirt in order to have **10% changes to grow a bush** if the Neighbour**Tree**Count() method return **0** and Neighbour**Bush**Count() method return integer **more than 2 (at least 2 square of bushes)**
  - The idea is inspired from the demo game, conwayslife
- In Tree class it extends Ground and Implements EcoPointInterface and ProbabilityInterface
  - The displayChar for a tree is '+'.
  - An ArrayList with Item data type (onTreeFruit) is used to store the fruits that are produced by a tree.
  - ~~A hashmap with <Fruit, Integer> data type (groundRipeFruit) is used to keep track of the fruits that are dropped on the ground. Using a hashmap is because it is easier for me to keep track of each Fruit's age. The key of the hashmap is Fruit class and the value of the hashmap is the age of the particular Fruit.~~

- tick() method is used to calculate the **50% chance for any tree to produce 1 ripe fruit**. The displayChar of the ripe fruit on the tree is '*'. The **ecopoint will increment by 1 if a ripe fruit is produced on the tree**. (explained later for eco point)
- After ripe fruit is produced, it invokes produceRipeFruit() method to add the ripe fruit into onTreeFruit ArrayList meanwhile the **ripe fruit will have a 5 % chance to fall**. The displayChar of ripe fruit on the ground is '*'.
- If the ripe fruit falls, it invokes ripeFruitDrop() method. ~~The ripe fruit on the ground will be added into groundRipeFruit hashmap with initial value 1 for the age of the fruit.~~ **A Fruit (Item) is added to the same location of the tree.** In fruit class, tick() method will calculate the number of turns for ripe fruit to rot then remove the fruit from ground. **The fruit on the ground will rot away in 15 turns**.
- ~~fruitAgeOnGround() method is used to check the age of the ripe fruit on the ground to indicate whether the ripe fruit on the ground is rotten or not.~~
- The Actor which is player with ~~PickUpItemBehaviour and~~ SearchingFruitBehaviour can enter to the same square of the tree to **pick up the fruit on the ground** or have **40% chances to successfully pick a ripe fruit from the tree or bush** then the ripe fruit will be added into its inventory by invoking SearchingFruitAction and PickUpItemAction (reusing base code).
- ~~In PickUpItemBehaviour class, it will check if any fruit on the ground is in the same location as the player (actor) by using displayChar for fruit on ground which is 'G' to differentiate it. If there is a fruit to pick up it invokes PickUpItemAction class.~~
- In PickUpItemAction class (reusing base code), the particular item will be added into player inventory and removed from the map.
- In the SearchingFruitBehaviour class, if the player is in the same location as the bush or tree (differentiate by using displayChar). It will invoke the SearchingFruitAction class.
- In the SearchingFruitAction class, ~~a random number~~ implements ProbabilityInterface to have a 40% chance to harvest a ripe fruit from a bush or a tree. If it is harvested successfully, the ripe fruit will be added into player inventory. The displayChar for Fruit in player's inventory will be '*'. If the fruit is searched from a tree, the onTreeFruit array list needs to be updated by implementing TreeInterface. The **ecopoints (by implementing EcoPointInterface) will increase by 10 if the player successfully searches (harvest) a fruit from the tree or from a bush**.

- In Bush class it extends Ground and implements ProbabilityInterface
  - The displayChar of a bush is 'u'.

- ~~An ArrayList with Fruit datatype (bushRipeFruit) is created in order to store the ripe fruit produced from a bush.~~
- tick() method is used to calculate the **10% chance for any bush to produce 1 ripe fruit**. The displayChar of the ripe fruit on the bush is '*'.
- After ripe fruit is produced, it invokes the produceRipeFruit() method to add the ripe fruit into the location ~~bushRipeFruit ArrayList~~.

Lakes, water and rain

- In **Lakes** class it extends Ground and implements ProbabilityInterface
  - The **displayChar of a lake is '~'**.
  - The lake have food type of water means it is used to store the water source
  - Each lake will have **25 sips of water initially**
  - Each lake will have **5 fishes initially**
  - Each lake can **hold a maximum of 25 fishes** only
  - tick() method is used to calculate the number of turns for the sky to rain, **every 10 turns** with a **probability of 20% the sky will rain**. The **amount of water to be added is calculated in the Probability class**. The ProbabilityInterface is used to calculate the probability.
  - tick() method is also used to **born a new fish** in each lake with a **probability of 60 % then added into fishInLake arrayList.**
- In **Rain** class it extends Ground
  - The **displayChar of rain is '~'**.
  - probabilityToRain() method is used to make sure all **dehydrated dinosaur will alive after raining** by increasing the **water level by 10**
- In **Fish** class it extends PortableItem
  - The **displayChar for fish is '~'**.
  - Fish has the food type of carnivore

Hungry dinosaurs(Stegosaur[stego]) / Brachiosaur[brachio] / Allosaur[allo]

The main actors of this game are dinosaurs [dino] which are Stegosaur, Brachiosaur and Allosaur so a dinosaurs package is created to group all dinosaur classes.

- In Dinosaur abstract class it extends Actor
  - The main code for dino is in Dinosaur abstract class, this class will be the parent class for all dino classes. For example,

Stegosaur class, Brachiosaur class, Allosaur class, BabyStego class, BabyBrachio class and BabyAllo class.
- Stego and Brachio are herbivores while allo are carnivores.
- There's a list of behaviour that the dino can behave and those behaviours will be invoked in playTurn method.

→ dino move
- Dino can **wander around the map** by invoking WanderBehaviour class so that **dino will not feel bored.**
- In WanderBehaviour class (reusing base code), it will loop the map location and allow dino to move to the particular location, if brachio step on bush means brachio and bush is in the same location then there's 50% chances to kill the bush (maybe brachio too heavy)

→ dino eat
- Dino can **eat** as well, when **dino is hungry they will find nearby food resources to eat themself**, this will be implemented in EatingBehaviour class, EatingAction class MoveActorAction class (reuse engine code) and MoveToEatBehaviour class
- In EatingBehaviour class, it will loop the item list for the particular location and check whether the food type is suitable to eat or not. ~~I use enum to differentiate the type of food so that stego and brachio can only eat fruits and vegetarian meal kit (feedByPlayer) while allo can eat all dino corpses, dino eggs and carnivore meal kit (feedByPlayer).~~ If the dino behaves then it will invoke the EatingAction class.
- In the EatingAction class, it will check the type of food by using the item's name and increment the particular dino's food level appropriately. If brachio and a tree are in the same location, a loop is needed to loop the onTreeFruit Array in order to allow the brachio to **eat as much fruit as it can from the tree in a single turn**.
- In the MoveToEatAction class, if the dino is hungry then it will move to a nearby location to eat by checking the item's food type of the location before moving to the location then invoke MoveActorAction class to move.
- Abstract methods are removed in order to reduce duplicate code (DRY). Those different requirements for different dinosaurs will be added into the constructor of each dinosaur class.
- ~~This abstract class will have some abstract method that is needed to implement in each child class due to different requirements such as incFoodLevel(), **hungry**(), dinoUnconsious() and abilityToAttack().~~
- ~~Their **starting food level** and **maximum food level** is different, for adult dino; stego 50/100, brachio 100/160, allo 50/100~~
~~for baby dino; stego 10/50, brachio 10/60, allo 20/50~~

- ~~Their **feel hungry food level** is different,~~
  ~~for adult dino; stego below 90, brachio below 140, allo below 90~~
  ~~for baby dino; stego below 25, brachio below 30, allo below 30~~
- ~~Their **number of turns of unconsciousness** is different,~~
  ~~for adult dino; stego is after 20, brachio is after 15, allo is after 20~~
  ~~for baby dino; stego is after 10, brachio is after 15, allo is after 10~~
  ~~(zero is inclusive)~~
- ~~Their **ability to attack** is different too, dino that can attack are allo and baby allo only, the rest of the dino cannot attack.~~
- ~~From the 4 points above, I decided to make the method abstract so that I can implement it differently in each child class.~~

→ feed dino

- An actor who is a player **can feed the dino if the player is standing next to any dino**. Players have FeedingBehaviour and FeedingAction to feed dino.
- In FeedingBehaviour class, it loops the list of exits and checks if there is a target (actor) at the destination. If yes, check the player inventory is it empty or not. If it is not empty, check the food type of the target and the food type of item in the player inventory. If both are the same food type (herbivore matches herbivore) then invoke FeedingAction class.
- In FeedingAction class, if the **food to feed is fruit then it will increment the ecopoint by 10**. It will invoke the EatingAction class again in order to allow the player to feed the particular dino and the dino can eat it.
- There are some non abstract methods as well, this means that all dino behave the same for the implementation such as the decFoodLevel() method as all **dino food levels will decrease by 1 in every turn**.
- The other non abstract method will be explained in Breeding.

Thirsty dinosaurs

This new requirement will be implemented in Dinosaur abstract class.

- Stating water level, maximum water level and feel thirty water level are initialised in the constructor.
- decWaterLevel(), method is used to **decrease the water level for each dino by 1 in each turn**.
- thirsty () method is added to keep track of the water level and the status of each dino
- dinoUnconscious () method is updated to add new requirements for dino to be **unconscious (dehydrated) for 15 turns**.
- If dino is dehydrated, it will alive after raining (explained above in Rain class)

- Thirsty dino will be removed directly from the map, after been dehydrated for 15 turns means **no corpse will be created**

Breeding

Only Adult dino can breed, an Age enum is used to differentiate whether the dino is adult or baby.

- In the dino main class which is a Dinosaur abstract class, there are a few non-abstract methods implemented for the dino breeding step such as isHatching() method, setHatching() method, abilityToMate() method, layEgg() method.
- In this abstract class, **String dinoSpecies and String dinoGender** are added into the input parameter for Dinosaur constructor in order to indicate and differentiate the gender and species of each dino. Meanwhile, dinoSpecies getter and setter are created.
- isHatching() method is used to determine whether the dino is hatching an egg or not, If yes means they can't mate.
- setHatching() method is used to set the dino whether it is hatching an egg or it is not.
- abilityToMate() method is used to **determine whether the pair of dino is the same species, ~~different gender and~~ their food level is sufficient** (by using Status.BREEDABLE) **to mate and female dino is not hatching an egg**. It returns true if the pair of dino meet all the conditions else return false.
- In order to find the opposite gender with the same species. This condition will be checked in MateAction class.
- In MateBehaviour class, it loops the list of exits and checks if there is a target (actor) at the destination. If yes, the target will be the second input parameter of abilityToMate() method which is invoked in this class to check all the conditions mentioned above. If yes, it invokes MateAction class.
- In MateAction class, it will check both actor's gender. If both actors are different gender then mate successfully and setHatching to true for female dino.
- After mating, different species will have **different time(turns) to lay an egg**. For stego is after 10 turns, for brachio is 30 turns and for allo is 20 turns (zero is inclusive).
- If the time to lay an egg is reached then I will setHatching() method to false because the female dino is successfully to lay an egg. The egg will be placed on the same location as the female dino with displayChar 'E' then reset the counter to 0 for this particular female dino. It creates an instance of Egg class.
- In Egg class it extends PortableItem and implements EcoPointInterface.
    - tick() method is used to calculate the time to hatch an egg.

brachio may high chance to become extinct so i decided to make brachio egg hatch faster than the other to prevent it been extinct,
    - The **time to hatch an egg for each dino are different**, stego is after 20, brachio is after 15, allo is after 50 (zero is inclusive)
    - If the time to hatch an egg is reached, a particular baby dino is born (created) and it will add it into the current location.
    - The **ecopoints will increment after the egg hatch**, for stego egg it worth 100, brachio egg it worth 1000, allo egg it worth 1000.
    - ~~Each species of egg is differentiate by using the EggType enumeration class~~
- In each baby dino class, it has a growUp() method to **calculate the age** (turns) for each baby dino. For baby stego is after 30 turns, brachio is after 50 turns and allo is after 50 turns.
- If the age is reached for the baby dino to become an adult dino, it invokes GrowingAction class.
- In GrowingAction class, it removes baby dino from the map and adds new adult dino into the map.

EcoPoints and purchasing

In order to keep track of the ecoPoints, an EcoPointInterface class is created to allow any classes that need to implement increment, decrement or get the ecopoints.

- In EcoPointInterface
    - An instance of EcoPoints class is created in order to **extend multiple classes in a class indirectly** as each class can only extend one class only.
- In EcoPoint class, it has some method to increment (incEcoPoint() method) and get (getEcoPoint() method) the ecopoints. To decrement the ecopoint, we can just put a negative sign in the input parameter of the incEcoPoint() method followed by the ecopoint.

A **vending machine with displayChar 'V' is placed on the map**. The item that the vending machine should sell and the ecopoints (price) is coded in a VendingMachineItem enumeration class.

- In the VendingMachine class, it adds all items that the vending machine should sell into an ArrayList with Item data type (items).
- Players will have BuyingBehaviour and BuyingAction to purchase as many items as the player can if the player has sufficient ecopoints to purchase.
- **Players need to be in the same location as the vending machine in order to perform buying behaviour and buying action**.
- In the BuyingBehaviour class, I created an option menu that allows the

user to input an option to purchase any item from a vending machine.
- After an option is inputted, it invokes the BuyingAction class.
- In the BuyingAction class, the checkEcoPoint() method is used to check if the current ecopoint is sufficient to buy the particular item or not. The reason why the code for checkEcoPoint() is duplicated is because every item will have different ecopoints (prices) and I coded it in a VendingMachineItem enumeration class. In order to get the particular item values, I need to be very specific to retrieve the value from VendingMachineItem class.
- If the player has sufficient ecopoints to purchase any item then that item will be added into player inventory. Meanwhile, the ecopoint will be updated by decrementing the current item ecopoints.
- **Users need to choose option 8 to stop buying items from the Vending Machine**.

Death

Every dino will die after a particular number of turns then it will become a dino corpse. Dino corpse will remain in the game for a set period of time
- In Corpse class it extends PortableItem
    - tick() method is used to keep track of the number of turns for the dino corpse to remain on the map.
    - **Each dino corpse will have different remaining time on the map**, stego is 20 turns, brachio is 40 turns and allo is 20 turns.
    - If the number of turns is reached, the dino corpse will be removed from the map.
    - The displayChar for dino corpses is ‘x’.
    - This class will be invoked, if the number of turns for dino to be unconscious in each dino class is reached.
    - **No corpse will remain on the map for all baby dino**. Baby dino will be removed from the map, if the number of turns for dino to be unconscious is reached.

Second map

A **new map is created** in the Application class with the **same size as the existing** map. The new map is located at the North of the existing map.
- Min Y range of existing map is used to create border at the most north of the existing map
- Max Y range of the new map is used to create a border at the most south of the new map.
- Max X range of the existing map is used to create a border within this X range by adding 1. (inclusive)

Team: Tute01Team78

- The CrossingMap class extends ground.
  - The displayChar for border is '='
  - allowableActions () method is override in order to provide the actions for player to move from existing map to new old or vice versa
  - All actor except player cannot enter this border

DisplayChar List

This list is created for better visualization and understanding.

'.' → dirt	'@' → player	'_' → floor	'#' → wall
'+' → tree	'*' → tree with ripe fruit	'*' → tree with ground fruit	
'u' → bush	'*' → bush with ripe fruit	'V' → vending machine	
's' → baby stego	'S' → stego	'v' → vegetarian meal kit	
'b' → baby brachio	'B' → brachio	'C' → carnivore meal kit	
'a' → baby allo	'A' → allo	'L' → laser gun	
'E' → dino egg	'x' → dino corpse		

'*' → fruit in player inventory and fruit purchase from vending machine
'=' → border to allow player to move form one map to another map
'~' → lake == fish == water

Team: Tute01Team78

Recommendations for extensions to the game engine

Adding a new class called SearchNearbyAction class in the engine. The problem that I faced is to search for the nearby location with particular items or actors. In this assignment, searching for nearby locations is used frequently such as when a dinosaur is hungry, a dinosaur is thirsty or a dinosaur is finding a partner to mate with, requiring them to search for nearby locations that contain targetFood or targetActor then move to the location.

In the SearchNearbyAction class, searchNearbyActor(Actor actor, Location location) method and searchNearbyFood(Actor actor, Item food, Location location) method can be added. In the existing engine code, it provides getExits() method and MoveActorAction class for us. Combination of both can achieve searching nearby action but if they can be in the same class will be more useful for me. Extending the class to have the searching nearby action instead of repeating the code over and over again to achieve this action. The repeated code is a loop for the current actor return value of getExits() method and loop for the target actor return value of getExits() method then check for the condition before moving to the target location.

The advantage of creating this class is to reduce the duplication of code and reduce the dependency between classes. Another advantage is to reduce the message chain smell such as if I want to get the ground type of the destination location, it raises the message chain smell {a loop is required to loop the return list of getExist() method [a list of unmodifiable of exits] e.g.code: map.locationOf(actor).getExits().getDestination().getGround()}. In the existing engine code, PickUpItemAction class and DropItemAction class are useful. By using both classes that's not needed for me to create extra PickUpItemBehaviour class and DropItemBehaviour class to achieve this action.

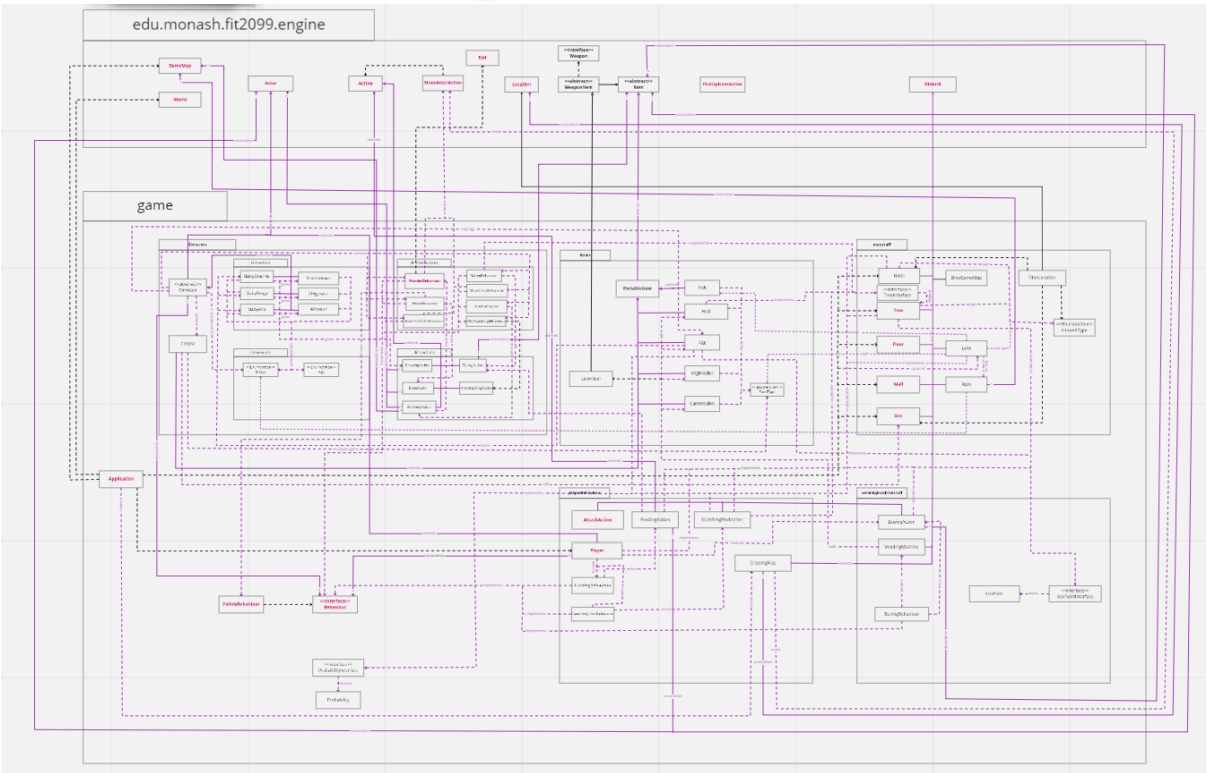
The new functionality that I would like to recommend is add more interaction between player and dinosaur. For example, if a player is near to any dinosaur, the player can pat the dinosaur then the dinosaur's food level will increase by 10 because the dinosaur is happy instead of the player can kill or feed the dinosaur only.

The paragraph above is from my point of view, the Jurassic park game is quite fun for me to implement but if there is an option for me to choose. I would like to try to implement the Harry Potter game. The Wizard world is more interesting than the dinosaur world.

Team: Tute01Team78

Class Diagram for my part [purple colour]

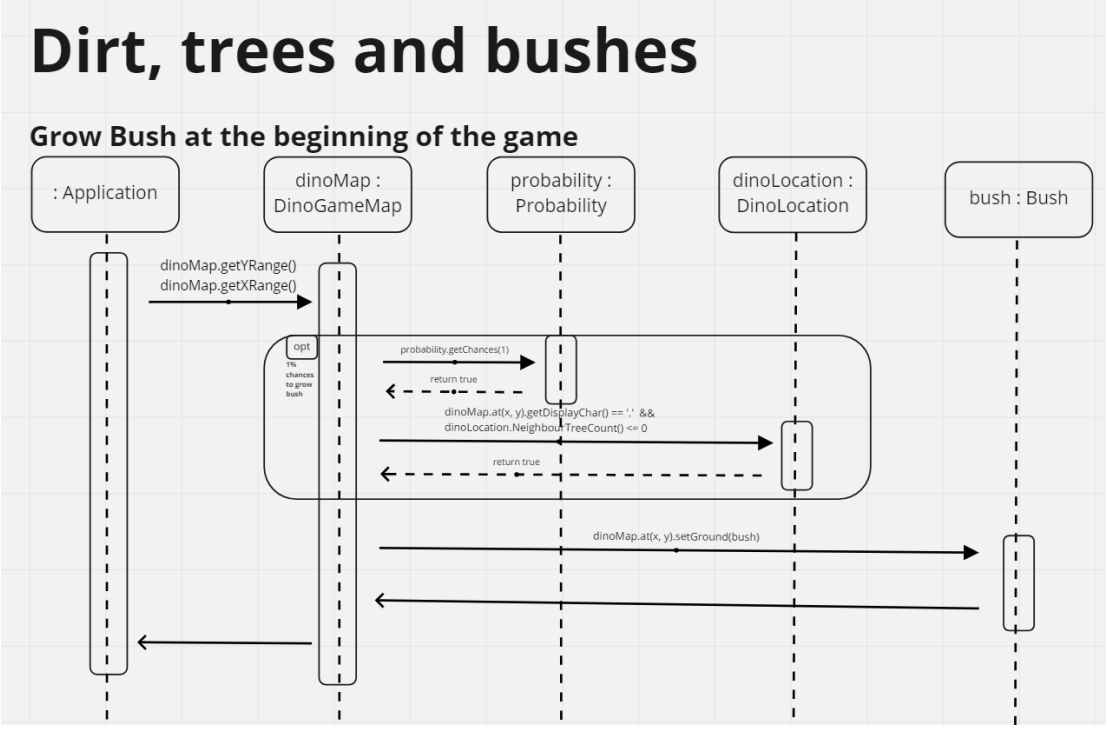
The link below is for marker to have a better view for class and sequence diagram
https://miro.com/app/board/o9J_lJVVxO8=/

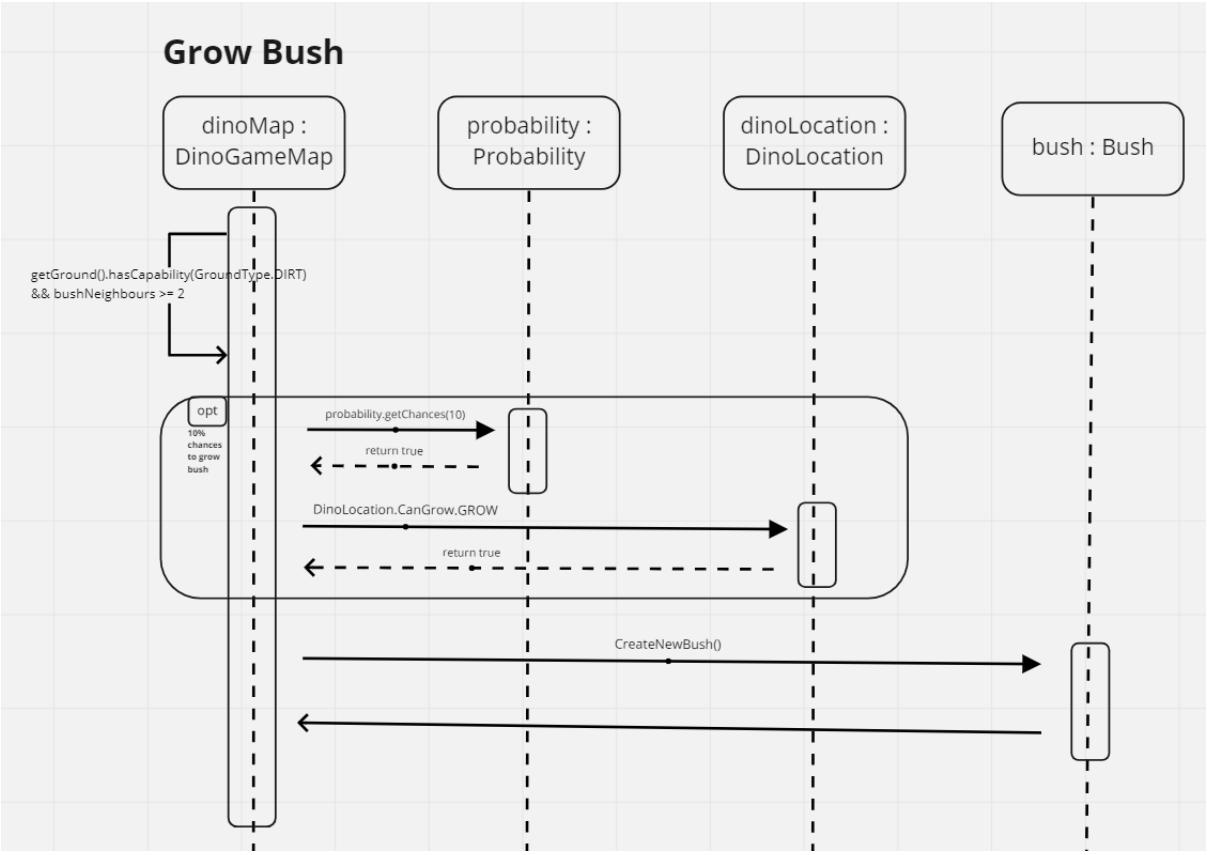


Team: Tute01Team78

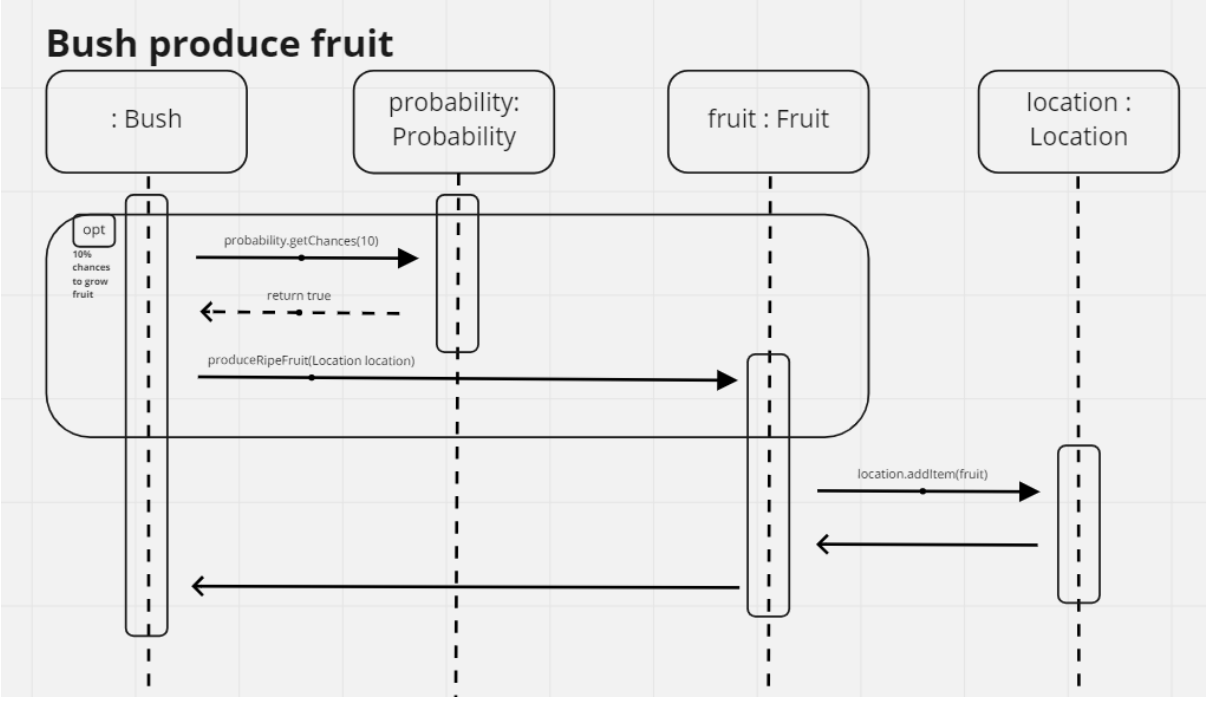
Sequence diagram for my part [under Assignment3_Sequence_Diagram]

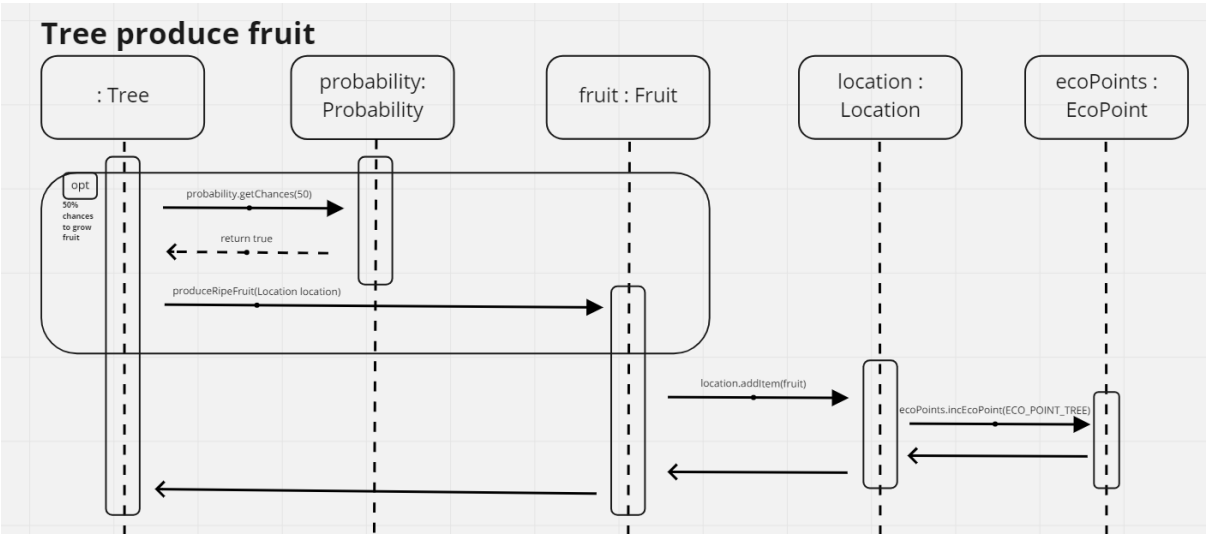
The link below is for marker to have a better view for class and sequence diagram
https://miro.com/app/board/o9J_lJVVxO8=/



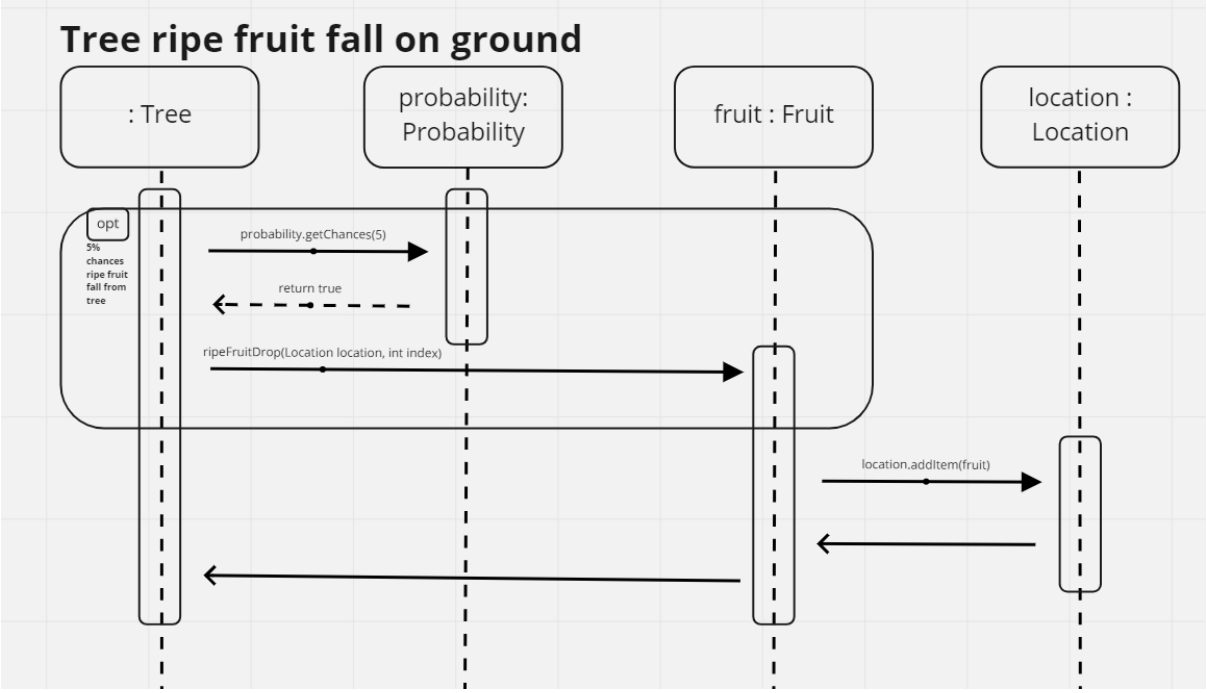


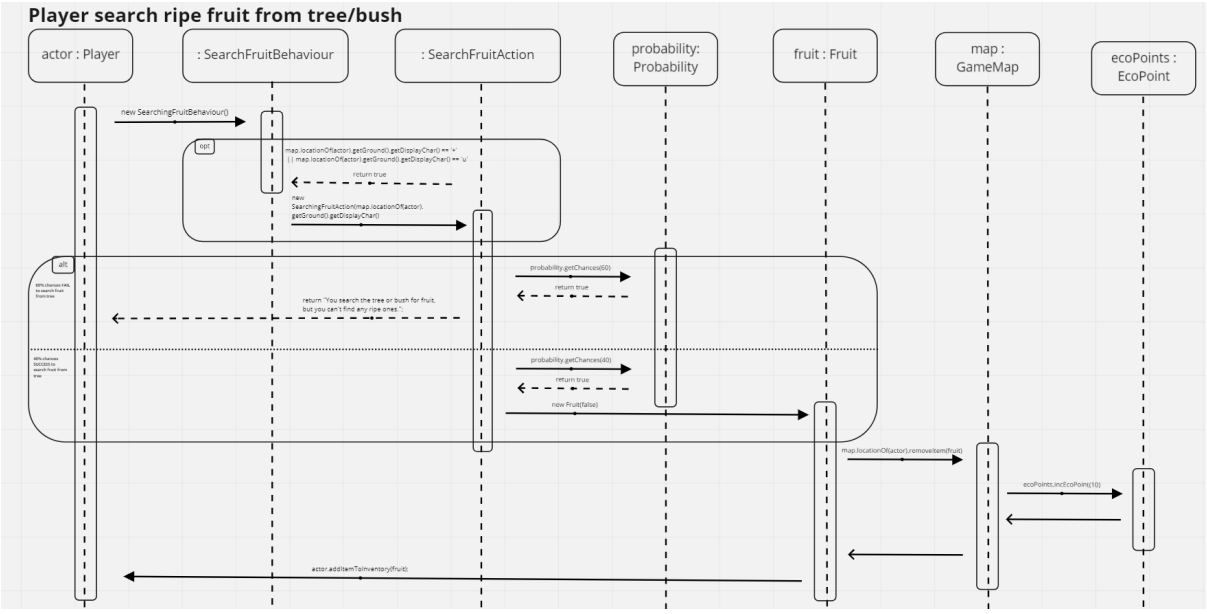
Team: Tute01Team78



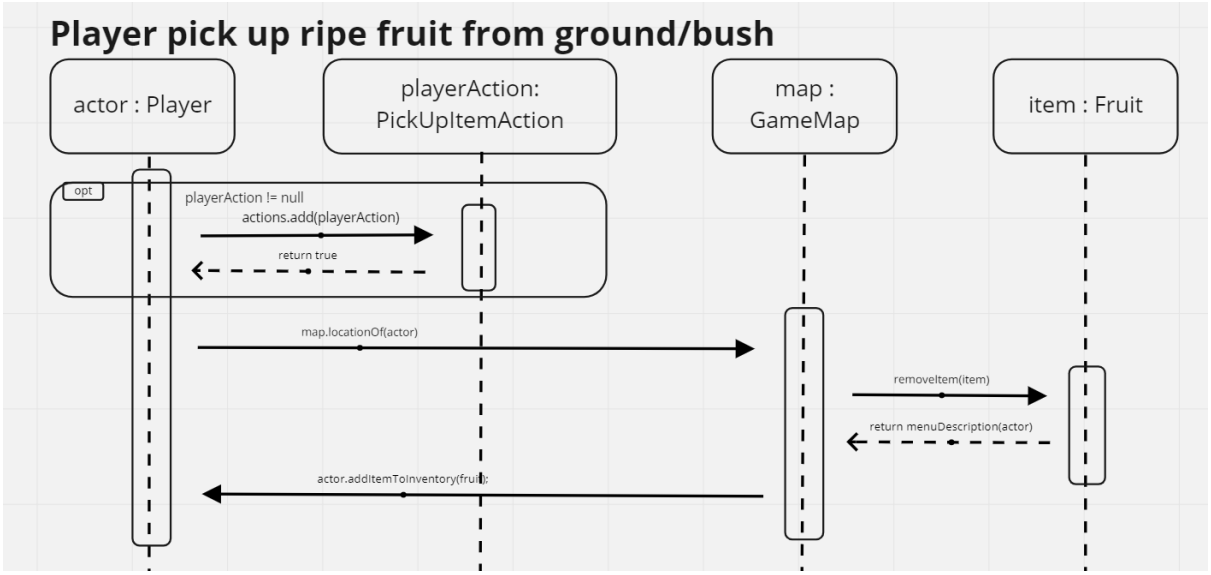


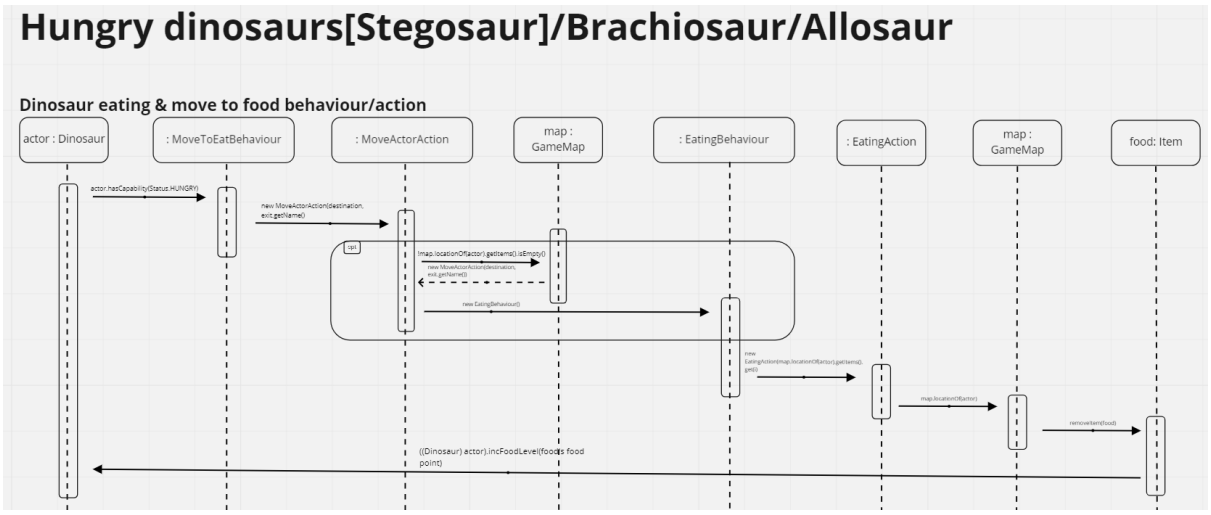
Team: Tute01Team78

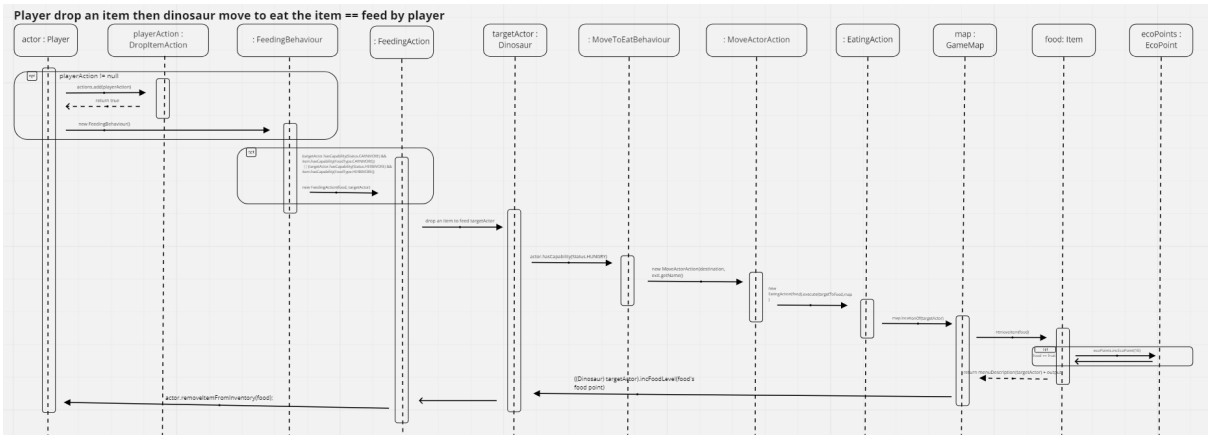




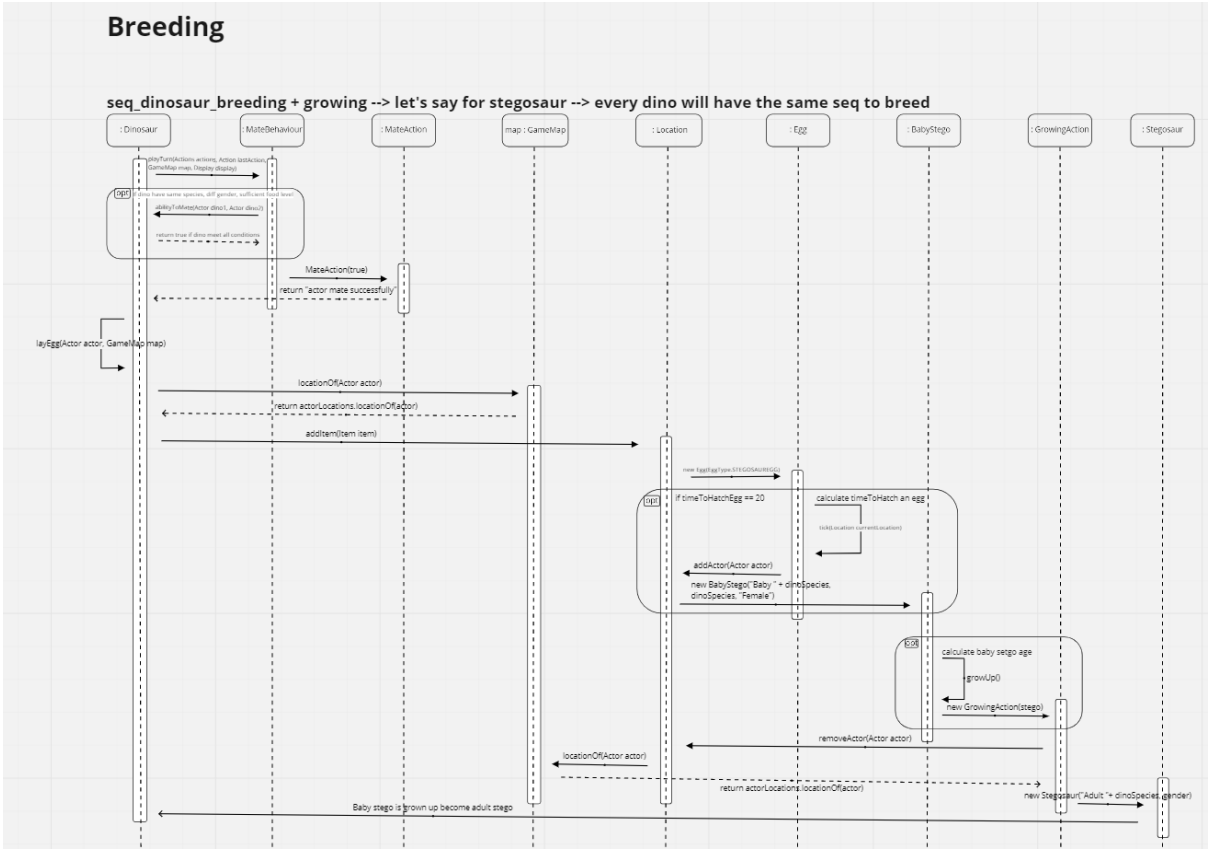
Team: Tute01Team78



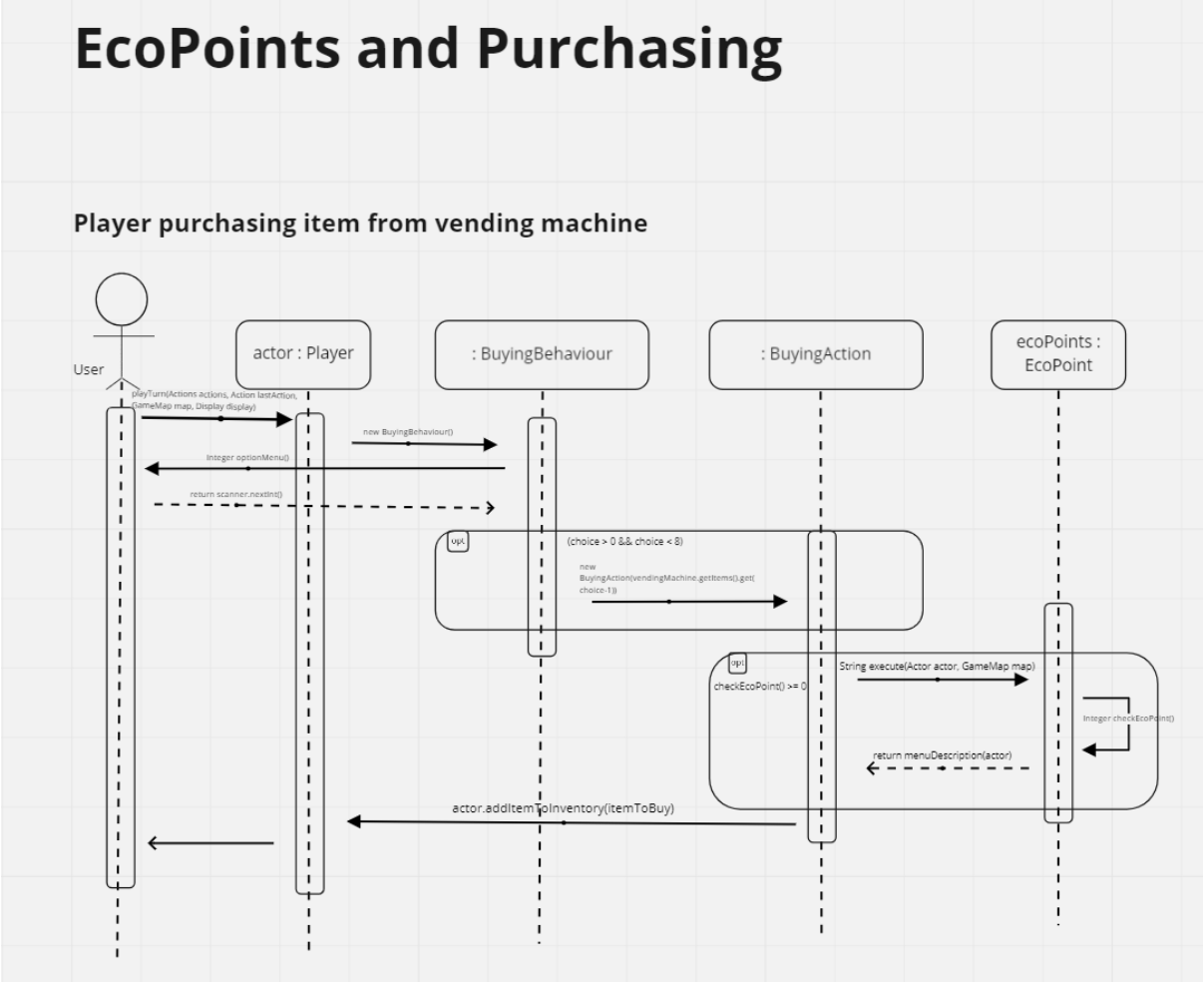




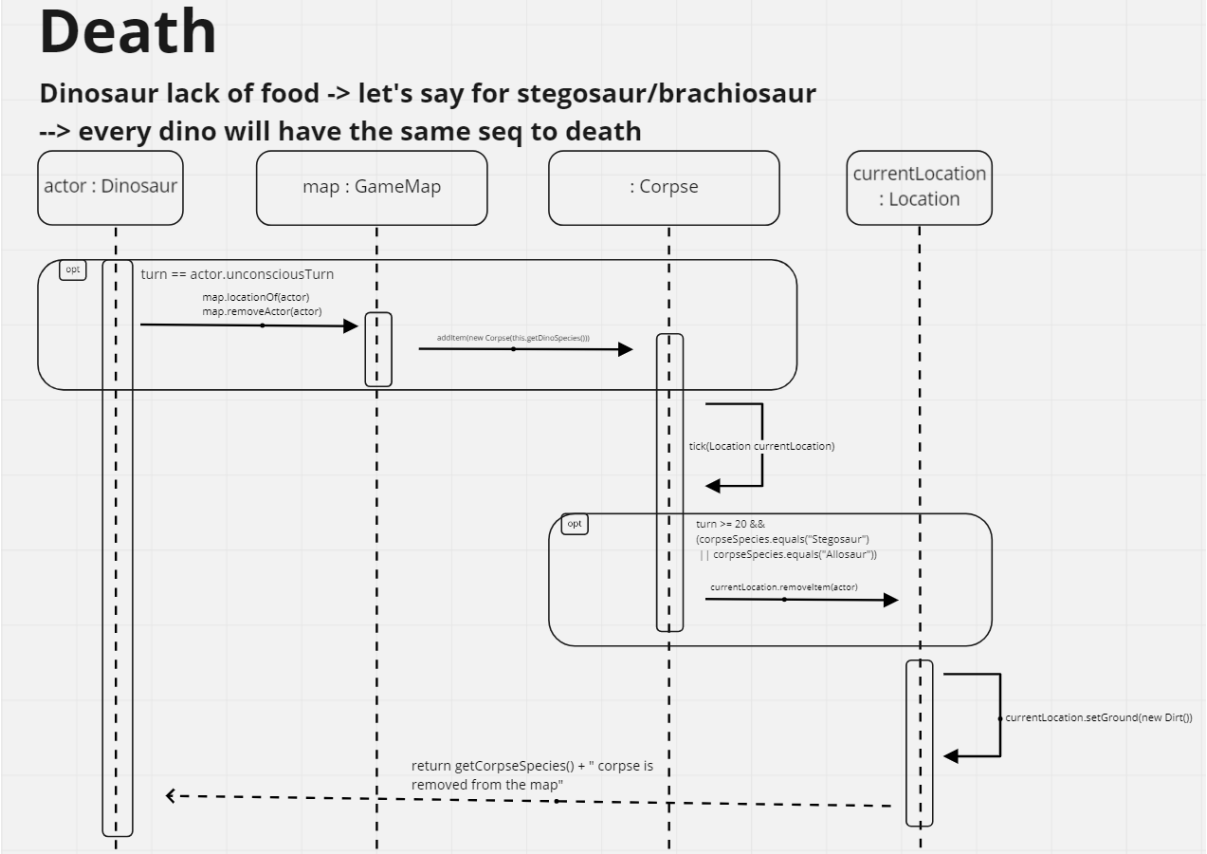
Team: Tute01Team78



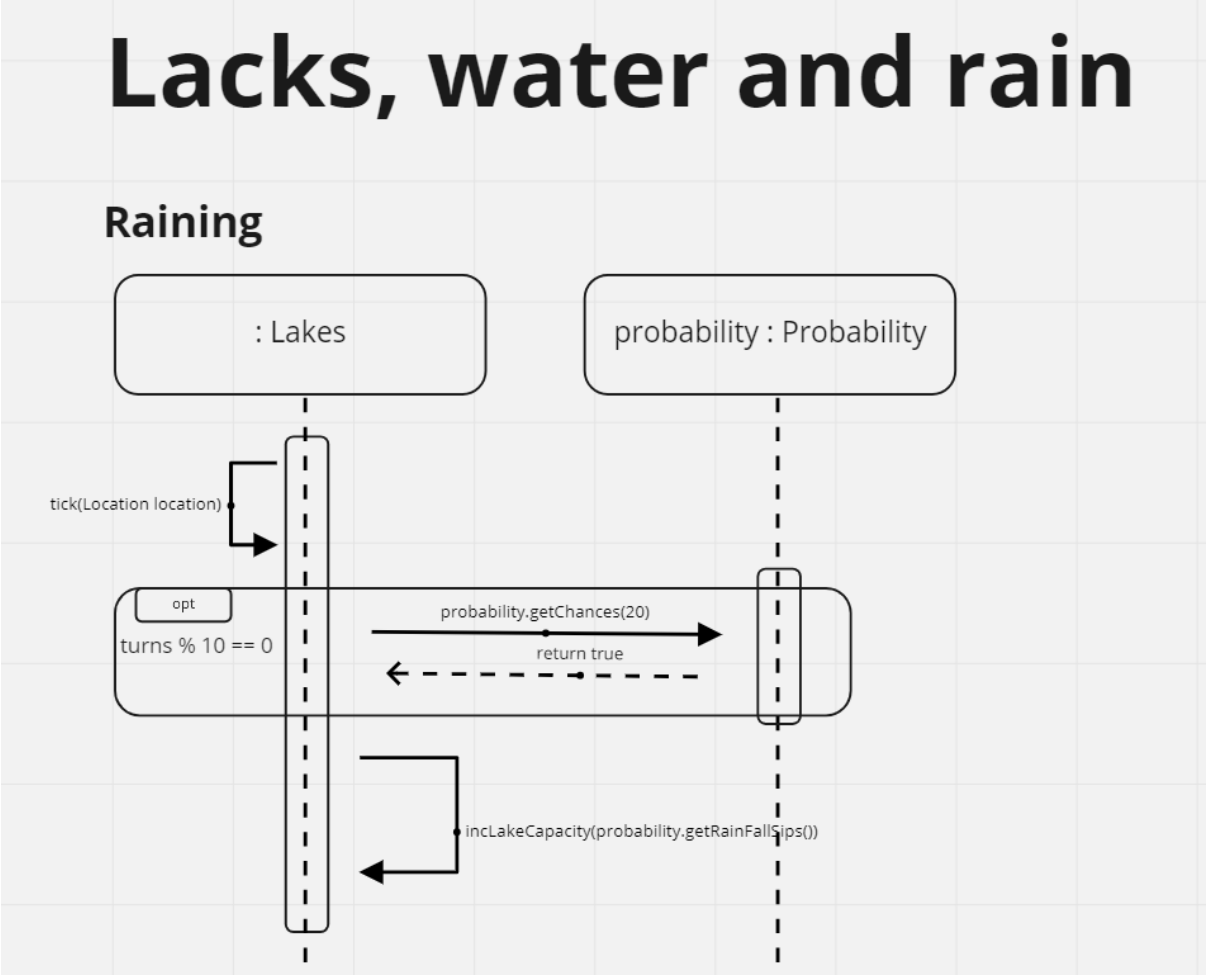
Team: Tute01Team78

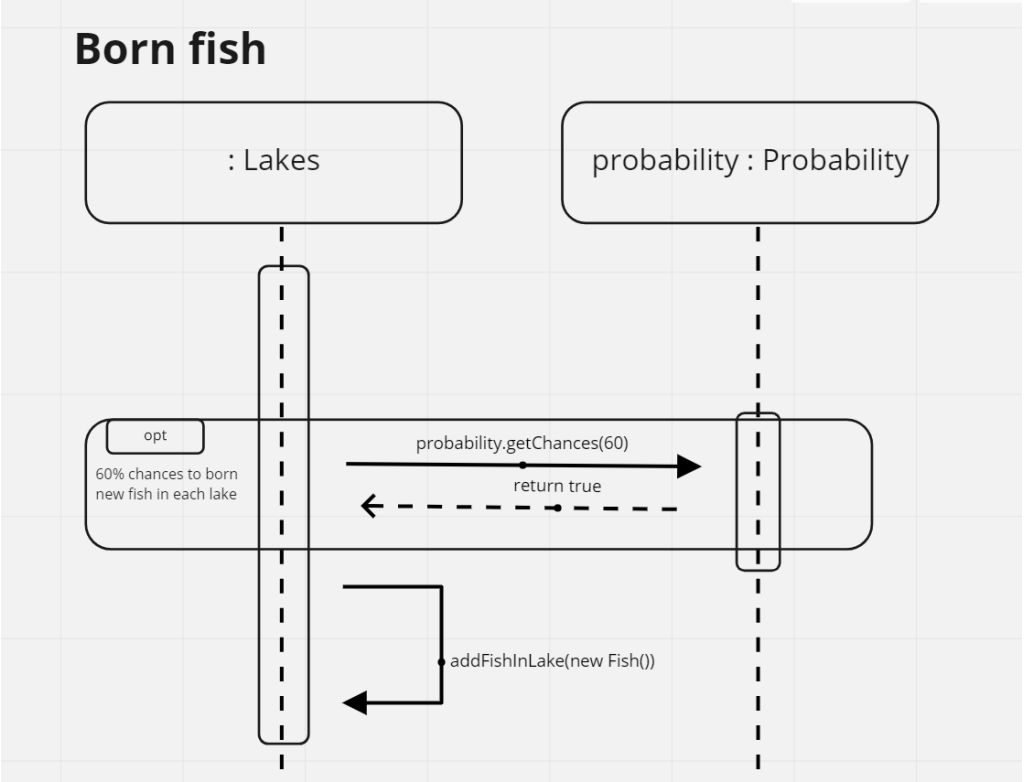


Team: Tute01Team78

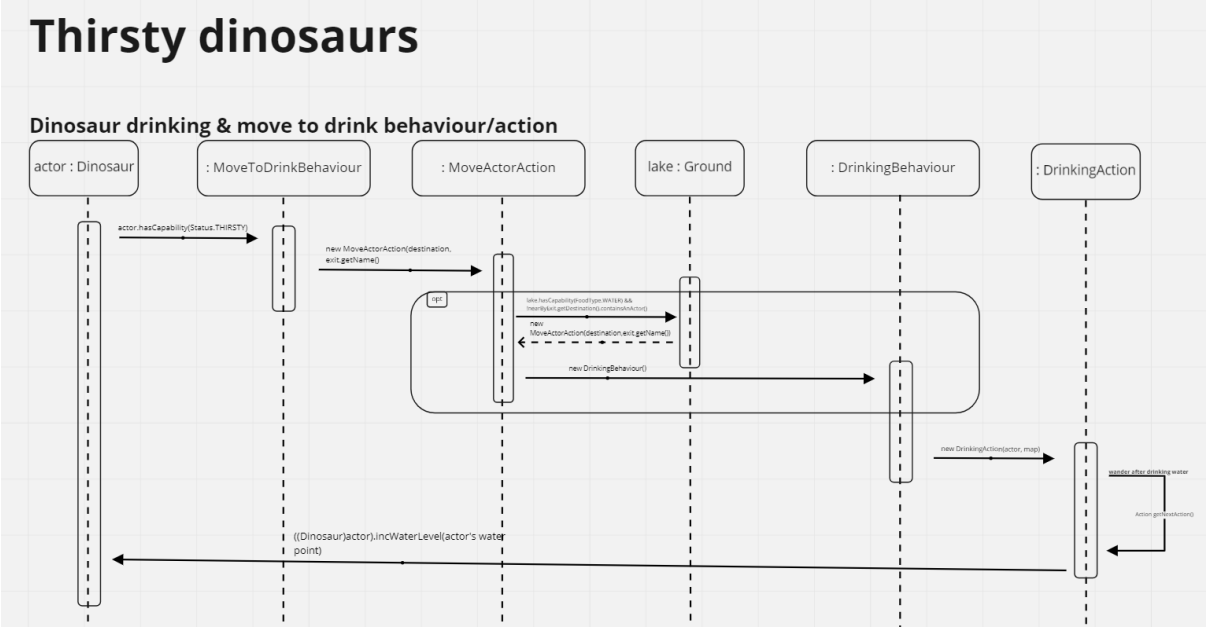


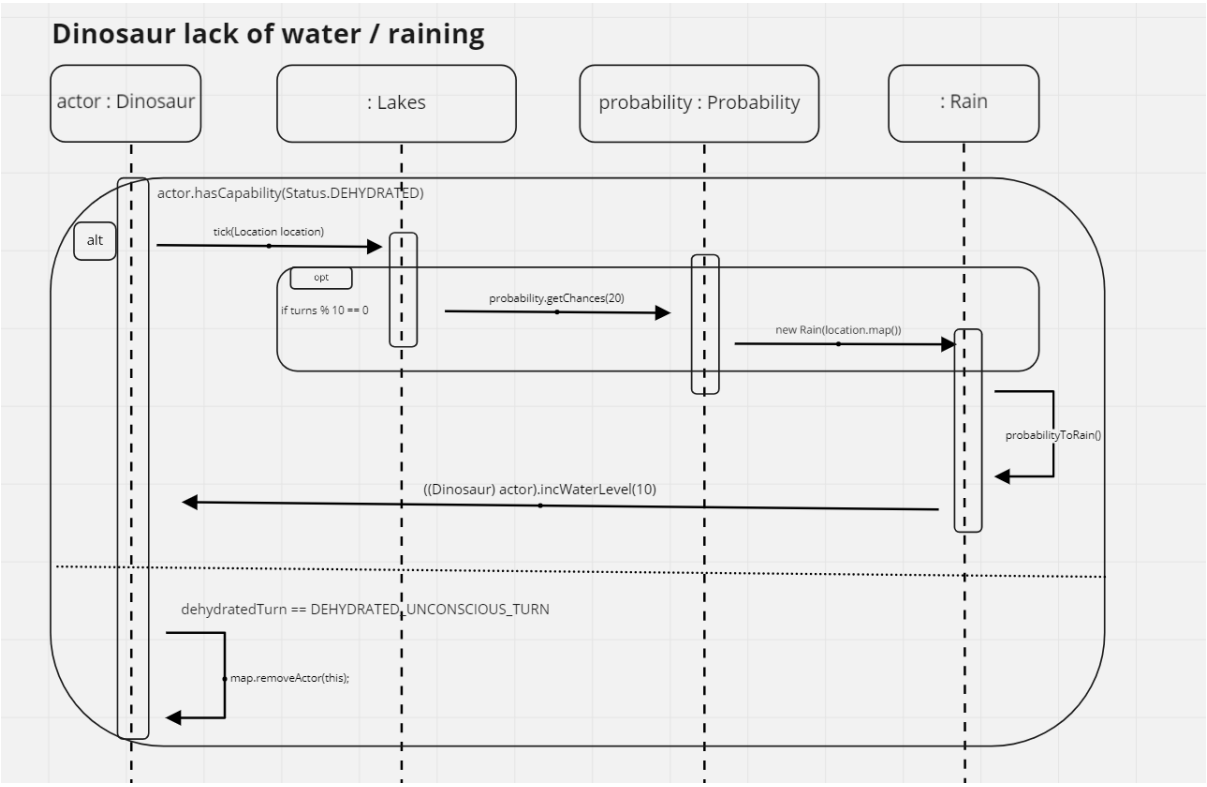
Team: Tute01Team78



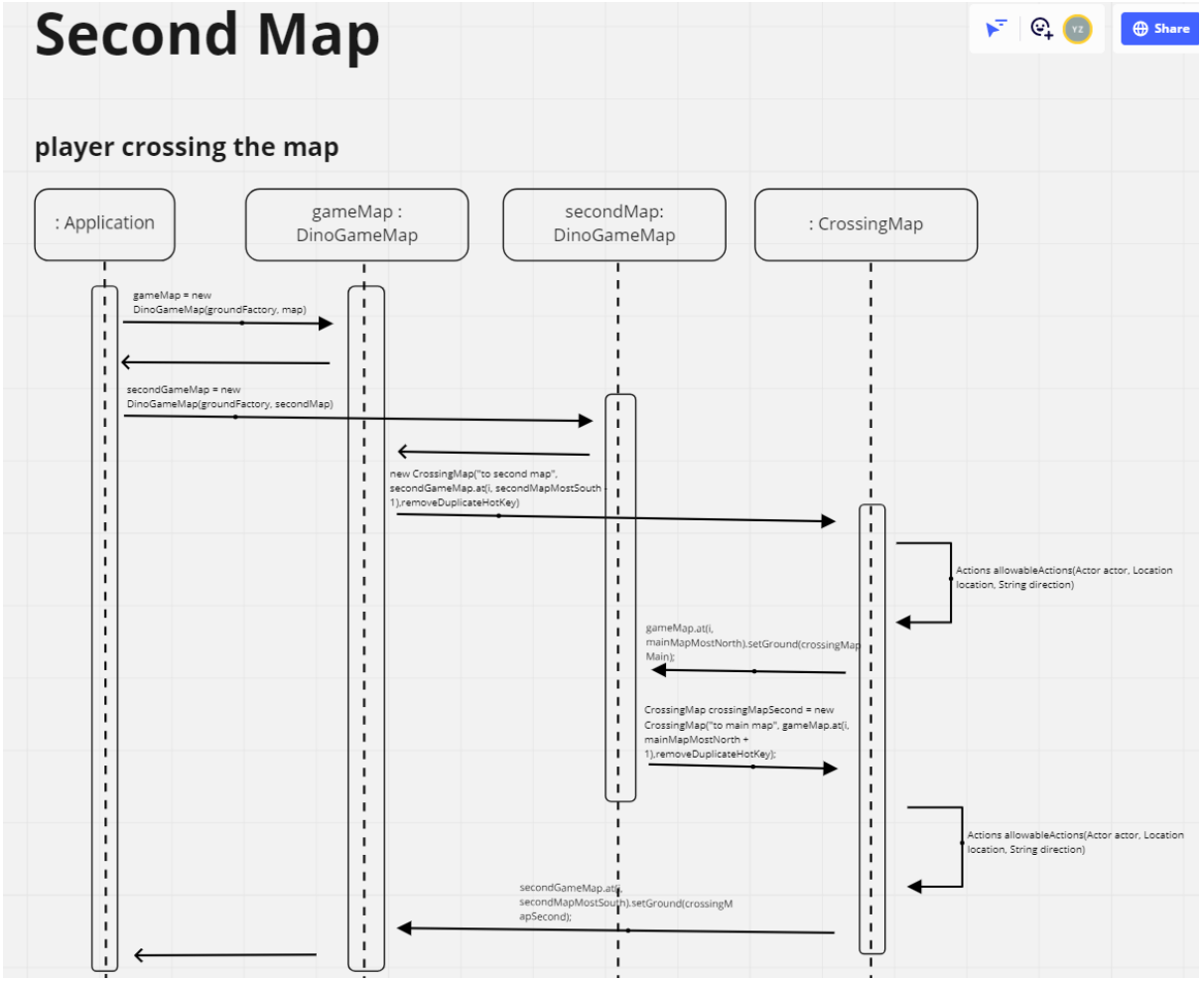


Team: Tute01Team78





Team: Tute01Team78

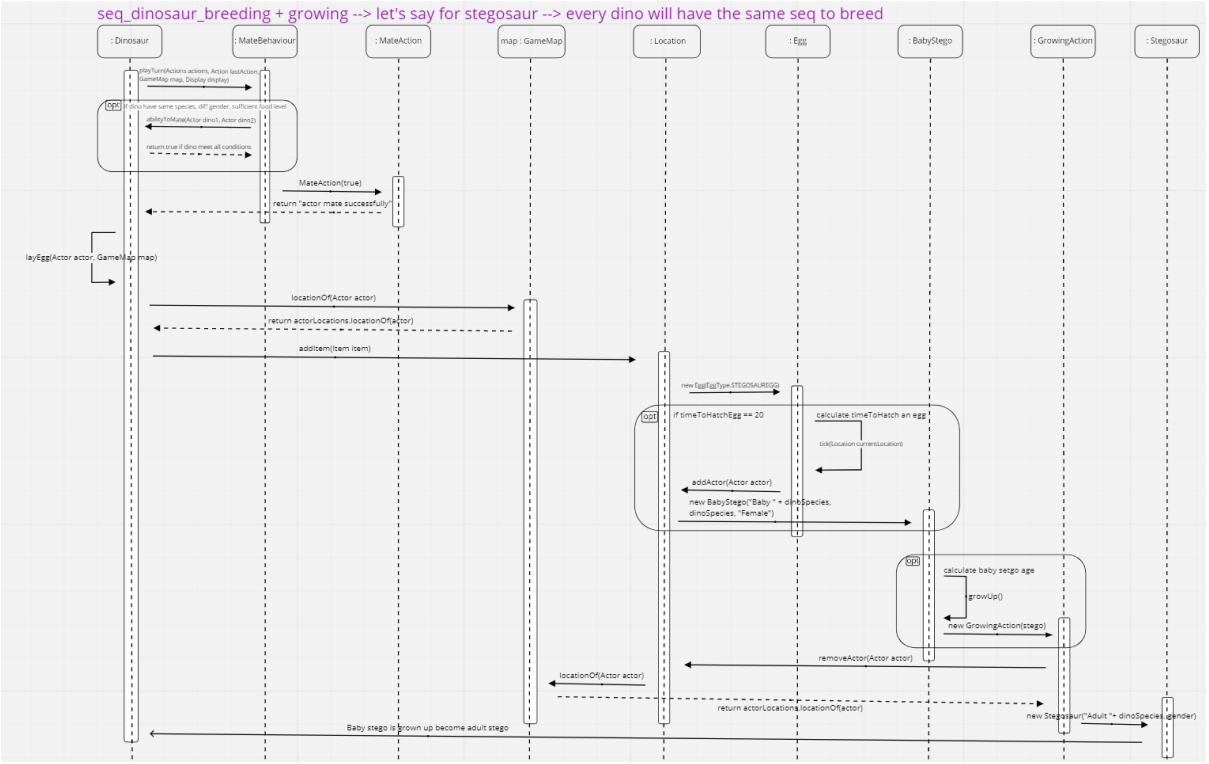


Team: Tute01Team78

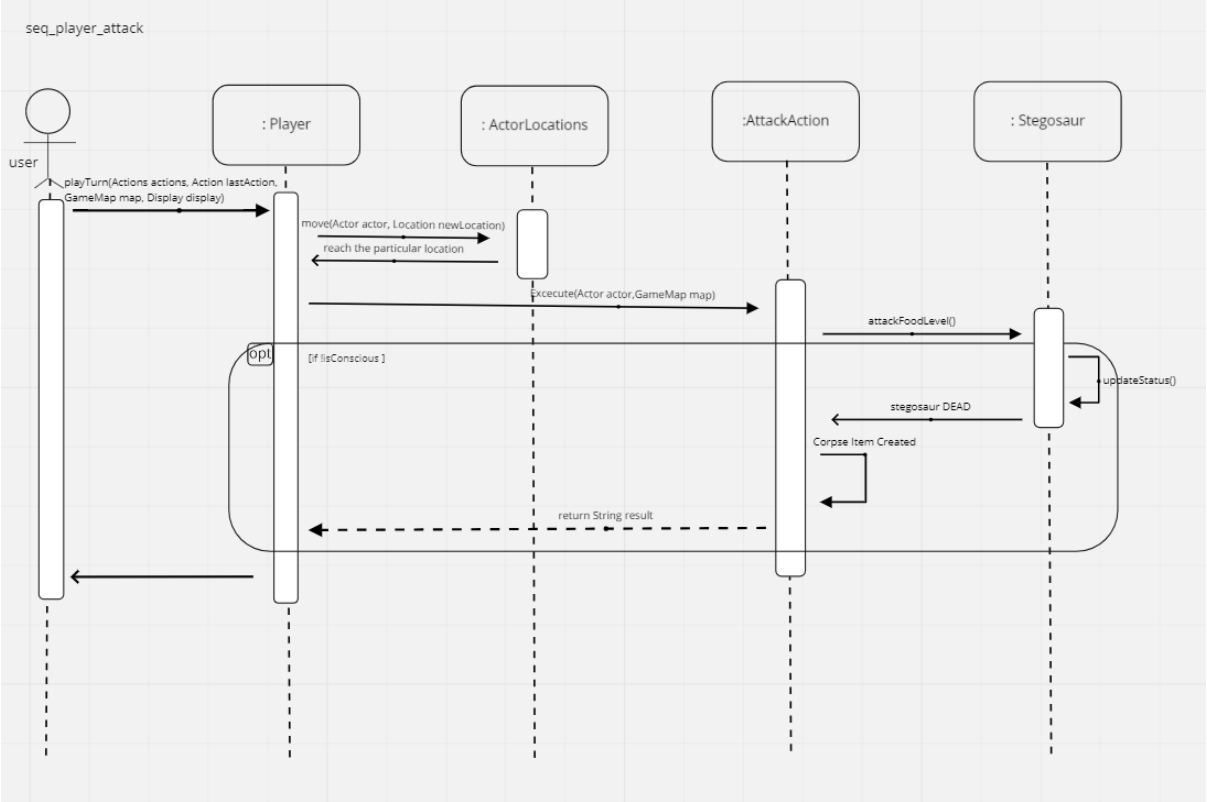
Sequence diagram [Assignment 2]

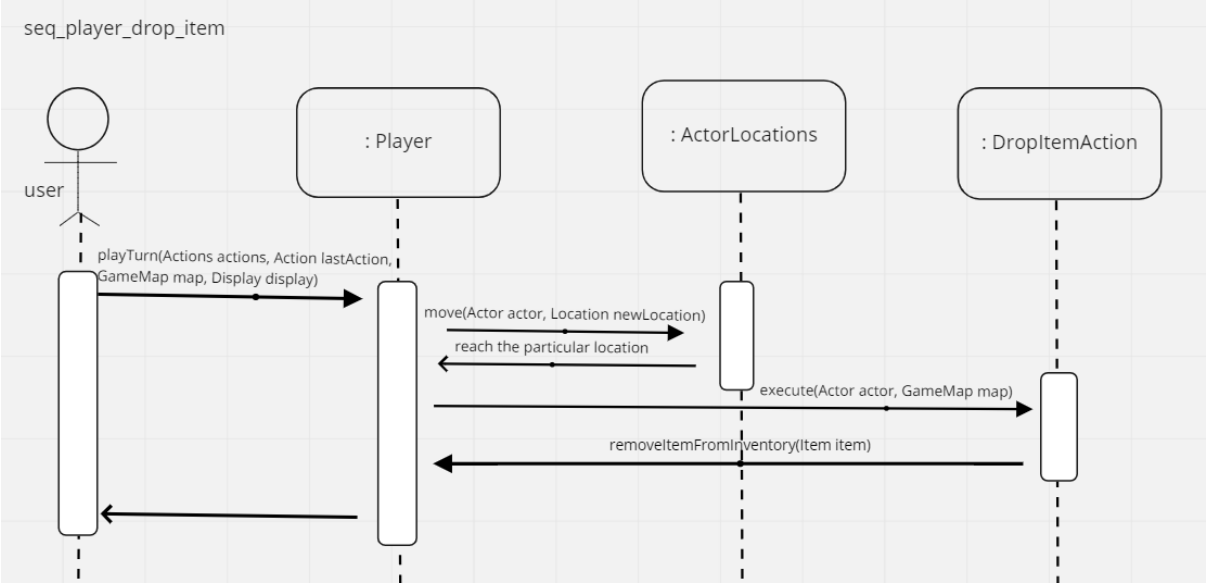
The link below is for marker to have a better view for class and sequence diagram
https://miro.com/app/board/o9J_lJVVxO8=/

→ Breeding + Growing



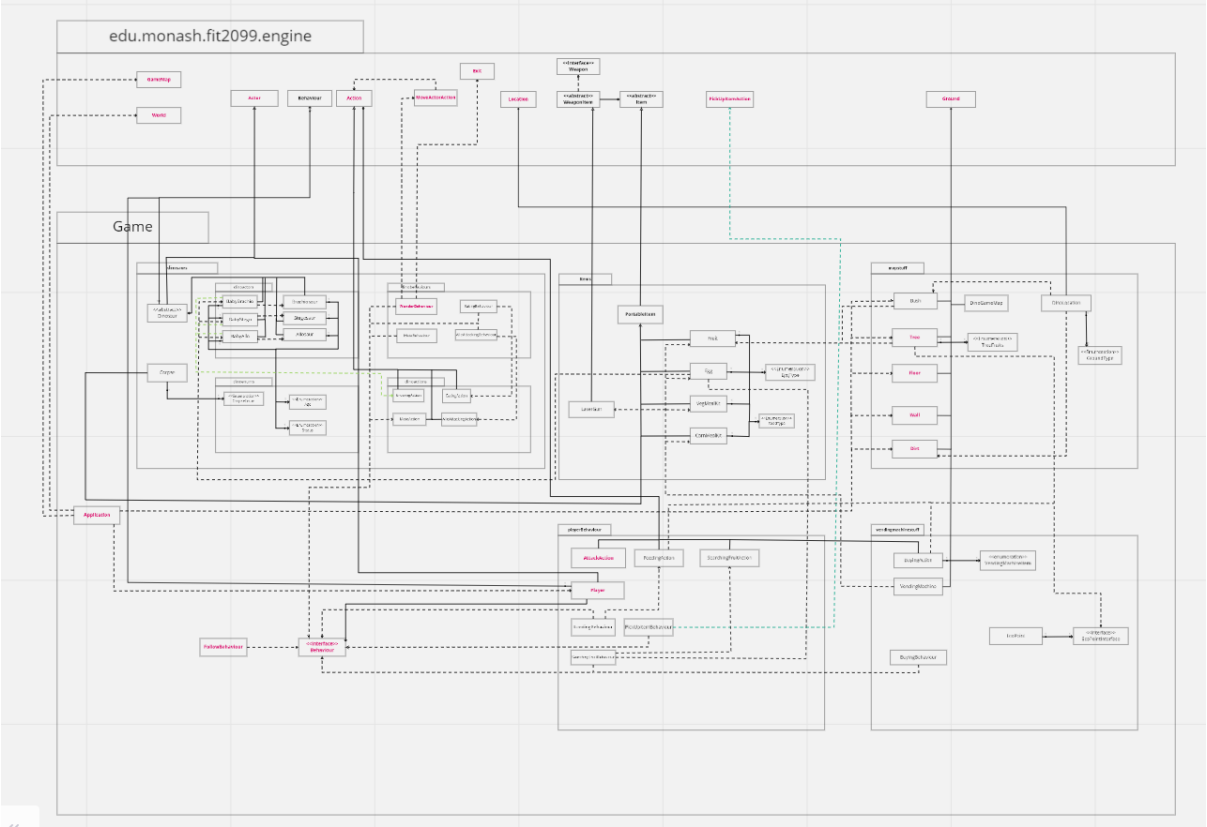
Team: Tute01Team78





Team: Tute01Team78

Class diagram [Assignment 2]



Team: Tute01Team78

A design rationale [Assignment 1]

(Reasoning for Dinosaur interface :- We created this interface because we noticed that there are some methods that a dinosaur must have when we want to implement the things needed
When we consider that we have to implement food level and breeding.)

- Dinosaur interface class
    → A decFoodLevel() method to decrement the food level
    → An attackFoodLevel () when a stegosaur is attacked by allosaurs.
    → A checkFoodLevel() method is used to check the food level of the stegosaur.
    → A findOppGender() method is used to determine the gender of the stegosaur and the food level.
    → A mate() method is used to create an dinosaur egg.
    → An updateStatus() method is used to update the latest status of each dinosaur.
    → An updateFoodLevel() method is used to update the food level for particular dinosaurs.
    → A feed() method is used to make the dinosaur feed on a food source(tree or Bush)
    → A hungry() method is used to search for the nearest food source from the current location of dinosaurs.

(Reason for changing the display char:- We gave each dinosaur a different Display Char to represent them so it's easier to differentiate what dinosaur is where on the map)

(will only feed from bushes and fruit on the ground(cannot eat fruit straight from a tree) this variance will be coded into the Feed() method which will be overridden from Dinosaur)

- Edit Stegosaur class (‘S’)
    → Stegosaur class is provided in the base code and it extends Actor class (inheritance) and has Behaviour Interface class as attribute (1-1 association) initially.
    → Now, more enumeration class is added to it such as Breedable class (determine stegosaur is breedable or unbreedable), Age class (determine stegosaur is baby or adult) and Status class (determine the stegosaur is alive or dead) as an attribute of the class (1-1 association).

→ Change the initial displayChar ‘d’ to ‘S’ to represent adult Stegosaur and it will show on the game map
→ Stegosaur class should implement Dinosaur interface class.

(Will only feed on trees that have fruits in their item list(Ripe fruit in the tree) and will not feed on fruit on the ground this variance will be coded into the feed() method which will be overridden from Dinosaur)

- Brachiosaur class (‘B’)
  → Brachiosaur class is added in the game folder. It extends Actor class (inheritance) and has Behaviour Interface class, Breedable enumeration class (determine stegosaur is breedable or unbreedable), Age enumeration class (determine stegosaur is baby or adult) and Status enumeration class (determine the stegosaur is alive or dead) as attribute (1-1 association)
  → the displayChar will be ‘B’ to represent adult Brachiosaur and it will show on the game map
  → brachiosaur class should implement Dinosaur interface class

(Will feed on eggs,Corpses and stegosaurs but will not eat fruits from bushes,trees and ground. This variance will be coded into the feed() method which will be overridden from Dinosaur)

- Allosaur class (‘A’)
  → Allosaur class is added in the game folder. It extends Actor class (inheritance) and has Behaviour Interface class, Breedable enumeration class (determine stegosaur is breedable or unbreedable), Age enumeration class (determine stegosaur is baby or adult) and Status enumeration class (determine the stegosaur is alive or dead) as attribute (1-1 association)
  → the displayChar will be ‘A’ to represent adult Allosaur and it will show on the game map
  → Allosaur class should implements Dinosaur interface class

- edit AttackAction class
  → dinosaurExecute() method is added to attack the target dinosaur and check whether the target dinosaur (Stegosaur) is alive or dead after attacking another dinosaur (allosaur).

(Also helps us to identify what dinosaurs are ready to breed and what are not when other dinosaurs are searching to breed)

Team: Tute01Team78

- Breedable class

→ Breedable class is an enumeration class. It is added in the game folder. It determines the ability of the dinosaur to breed (BREEDABLE, UNBREEDABLE).

→ (How to determine?) By checking the food level for each dinosaur to determine whether it can breed or not. The minimum food level (inclusive) for stegosaur to breed is 51, for brachiosaur is 71 and for allosaur is 51.

(Helps us to identify whether a dinosaur is an adult or baby when other dinosaurs are looking to breed)

- Ageclass

→ Age class is an enumeration class. It is added in the game folder. It determines the age of the dinosaur (BABY, ADULT)

→ (How to determine?) By checking the number of turns after the baby dinosaur is hatched from an egg to grow up to become an adult dinosaur. Baby stegosaur needs 30 turns, baby brachiosaur needs 50 turns and allosaur needs 50 turns to grow up to become adult dinosaurs.

(Helps us when generating the corpse and what dinosaur this corpse will belong to)

- Status class

→ Status class is an enumeration class. It is added in the game folder. It determines the status of the dinosaur (ALIVE, DEAD).

→ (How to determine?) By checking the food level and the number of turns for each dinosaur. If the food level is 0 and the dinosaur is not fed by the player, after 20 turns stegosaur dead, 15 turns brachiosaur dead and 10 turns allosaur dead.

(when a player does comes near this vending machine, Multiples capabilities will be given to the player(eg:- Buy a laser gun) etc for each item and they can choose from these options to buy things from the vending machine)

- VendingMachine class

→ VendingMachine class is added in the game folder. It extends ground class (inheritance) so that we can place the vending machine on the map. It also has a VendingMachineItem enumeration class as attribute (1-1 association).

→ It has Egg class, Fruit class, CarnMealKit class, VegMealKit class and LazerGun class as local variables in its methods. (dependency)
→ Those local variables mentioned above are the only items that are available in the vending machine that allow users to purchase by using eco points.
→ If player purchase an item, the item object will be created in the VendingMachine class and the item will be added to the actor's inventory (there's a method in Actor abstract class 'addItemToInventory')
→ Players can only purchase an item per turn.

(Reason:-Allows us to easily attach values to the items in the vending machine)
- VendingMachineItem class
→ VendingMachineItem class is an enumeration class. It is added in the game folder. It determines the type of items and the value of the items in the vending machine [FRUIT(30), VEGMEALKIT(100), CARNMEALKIT(500), STEGOEGG(200), BRACHIOEGG(500), ALLOEGG(1000), LASERGUN(500)]
→ The eco points (values) for each item are fixed so we decided to make it as an enumeration to store items and values.
→ An affordable(Integer ecoPoints, Item item) method with two input parameter (item and ecopints) is created in the class to check whether the eco points is sufficient to purchase the particular item

(making the variable static and also the method allows us to easily interact with the players eco points when coding for events that will increase the eco points and deducting eco points when the player buys items from the vending machine)
- Edit Player class
→ Eco points will be a public static attribute for Player class to keep track of the points in every turn. Will also include static methods to manipulate this value
→ Those points collected will be used to purchase items at a vending machine.
→ Eco points can be collected/earned if any dinosaur's egg is hatched, any fruit is produced and any dinosaur is fed by the player.

- Edit Tree class

→ Two array lists are created to keep track of the fruit that are currently in the Tree and for fruits that fall off the list.Also edited so that trees have a 50% chance of producing a fruit.
→ Tree class has Item class as attribute (1-1 association) and Fruit class as local variable (dependency).
→ To keep track of the ripe fruit that is produced by a tree and the ripe fruit that fall on the ground.
→ the displayChar will change from either ‘t’ or ‘T’ into ‘*’ when there’s a ripe fruit fall on the ground
→ a method getRipeFruit(Fruit fruit) will be added to keep track of the ripe fruit that has fallen from the particular tree.

- Bush class
→ Bush class is added in the game folder. It extends ground class (inheritance).Will have a 10% chance of producing fruit.
→ Bush class has Item class as attribute (1-1 association) and Fruit class as local variable (dependency).
→ ripeFruit() method is created to keep track of the ripe fruit that is produced by a bush.
→ the displayChar will be “u”

(If an egg is placed on the Dirt,the map will display the eggs DisplayChar instead of the dirt)
- Edit Dirt class
→ Dirt class has Item class as attribute (1-1 association).
→ After a player purchased a dinosaur egg from a vending machine (addCapability()), it is allowed to place the egg on dirt (hasCapability() to DropItemAction()).
→ for dinosaur’s egg we will use the same displayChar

(We can give items a displayChar so we can display the egg on the Map)
- Egg class
→ Egg class is added in the game folder. It extends Item class, Actor class (inheritance) and has EggType enumeration class as attribute. (1-1 association)
→ (How to determine?) By checking the number of turns to determine whether the female dinosaur laid an egg or not. Female stegosaurs needs 10 turns, female brachiosaur needs 30 turns and female allosaur needs 20 turns to lay an egg after mating.

→ There's a method in the class to determine how many turns is needed for a dinosaur egg to hatch. Stegosaurs's egg needs 15 turns, brachiosaur's egg needs 10 and allosaur's egg needs 5 turns to hatch.
→ The displayChar on the game map for stegosaur's egg will be 's'
→ The displayChar on the game map for brachiosaur's egg will be 'b'
→ The displayChar on the game map for allosaur's egg will be 'a'
→ timeToLayEgg() method is used to keep track of the time (number of turns) needed for a female dinosaur to lay an egg.
→ timeToHatch() method is used to keep track of the time (number of turns) needed for an egg to hatch.

(Helps us to identify what kind of egg it is)
- EggType class

→ EggType class is an enumeration class. It is added in the game folder. It determines the type of the dinosaur's egg (STEGOSAUREGG, BRACHIOSAUREGG, ALLOSAUREGG)
→ (How to determine?) By checking the number of turns after the dinosaur mate.

(Fruit is an item so it can be stored in the inventory and other item lists on which belong to map object)
- Fruit class

→ Fruit class is added in the game folder. It extends Item class (inheritance) to allow the capability for users to use it after the player purchased it at a vending machine.
→ Capability being to feed it by actor to either Stegosaurs or Brachiosaur to increase their food level by 20.
→ It is the food source for stegosaurs and brachiosaur as both of them are herbivores.
→ Players can pick the fruit from the ground or purchase it from a vending machine (30 eco points). Those fruits will be added into the actor's inventory. (there's a method in Actor abstract class 'addItemToInventory')
→ A addTreeFruit() method is created to add the fruit that is either pick up by player
→ A fruitDescription() method is created to return String message "Fruits that fallen from the particular tree is added"

Team: Tute01Team78

- CarnMealKit class

  → CarnMealKit class is added in the game folder. It extends Item class (inheritance) to allow the capability for users to use it after the player purchased it at a vending machine.

  → Capability being to feed it to an Allosaur to get their food level to the maximum.

  → It is the food source for allosaur as it is carnivore.

  → Players can purchase it from a vending machine (500 eco points) and the meal kit will be added into the actor's inventory. (there's a method in Actor abstract class 'addItemToInventory').

- VegMealKit class

  → VegMealKit class is added in the game folder. It extends Item class (inheritance) to allow the capability for users to use it after the player purchased it at a vending machine.

  → Capability being to feed it to either Stegosaurs and Brachiosaur to get their food level to the maximum.

  → It is the food source for stegosaurs and brachiosaur as both of them are herbivores.

  → Players can purchase it from a vending machine (100 eco points) and the meal kit will be added into the actor's inventory. (there's a method in Actor abstract class 'addItemToInventory').

- LaserGun class

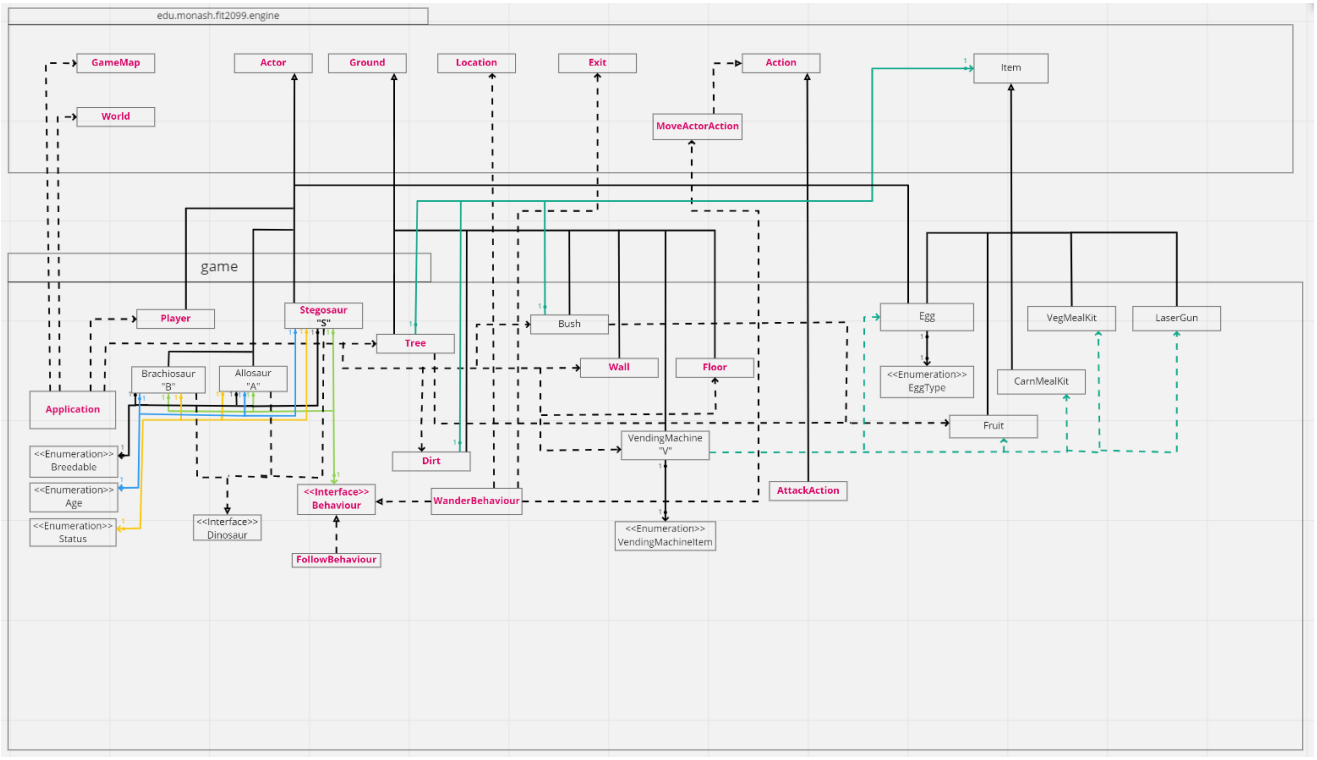
  → LaserGun class is added in the game folder. It extends Item class (inheritance) to allow the capability for users to use it after the player purchased it at a vending machine.

  → Capability to kill the Stegosaur in 1 or 2 hits if it is run out of food sources.

  → Players can purchase it from a vending machine (500 eco points) and the laser gun will be added into the actor's inventory. (there's a method in Actor abstract class 'addItemToInventory').

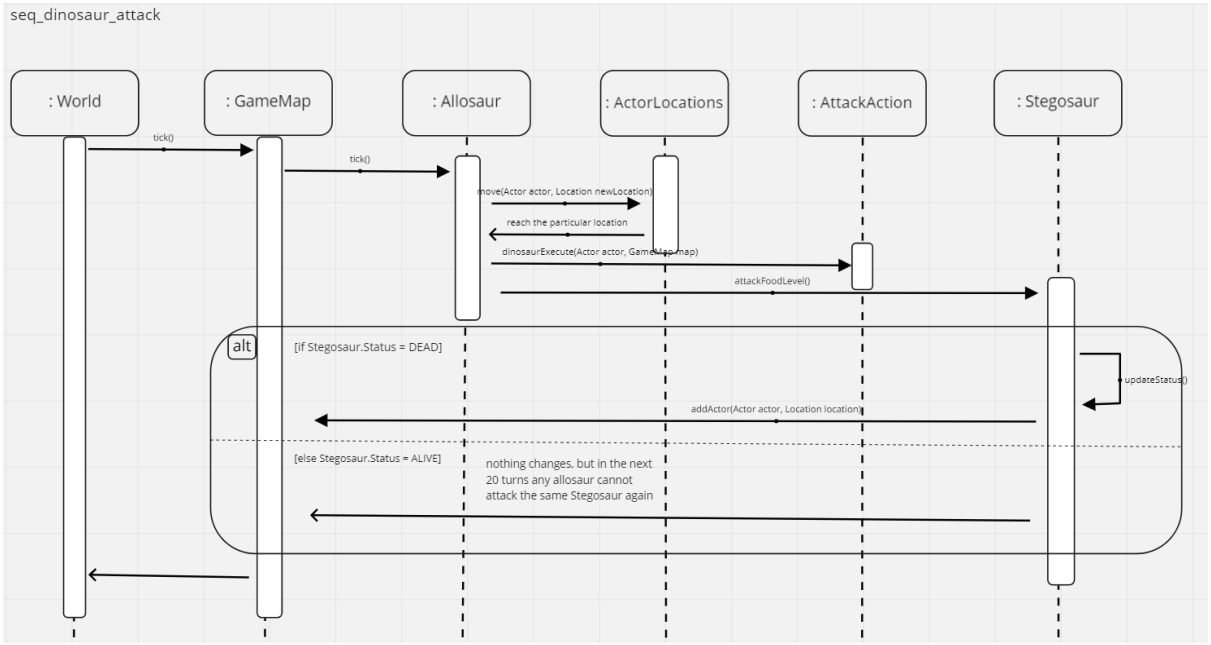
Team: Tute01Team78

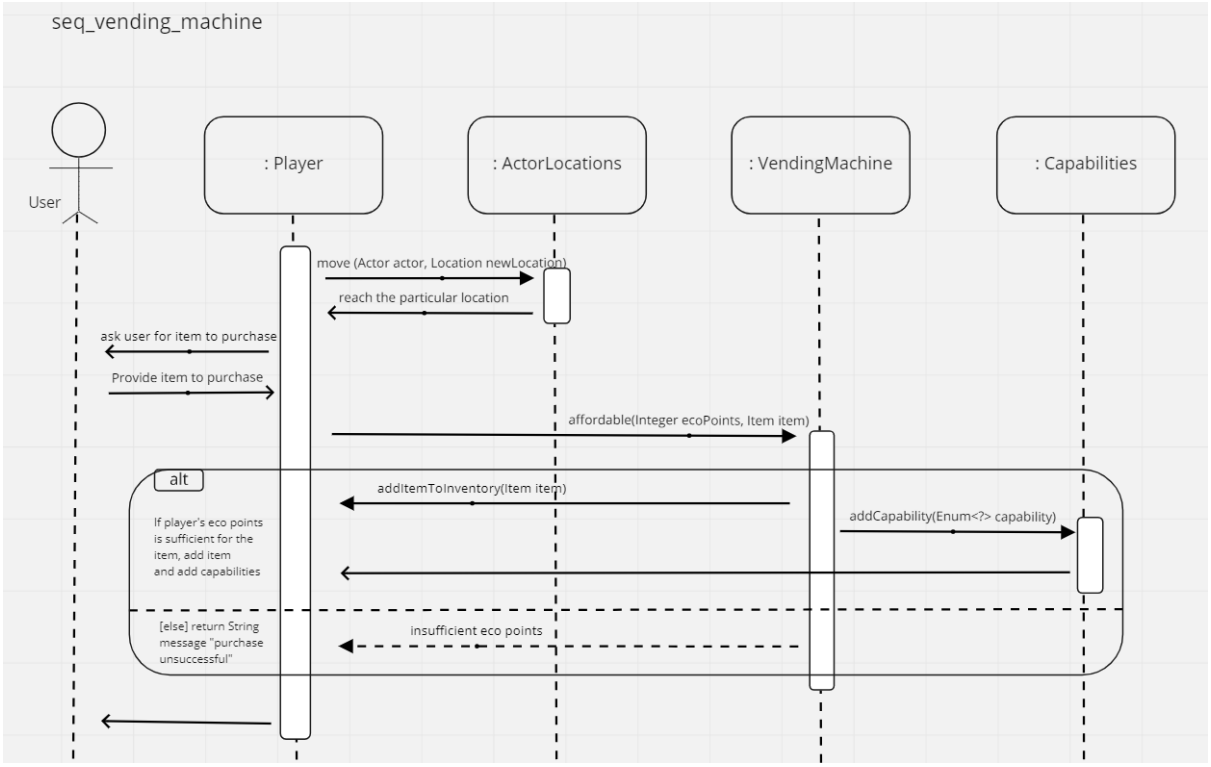
Class Diagram



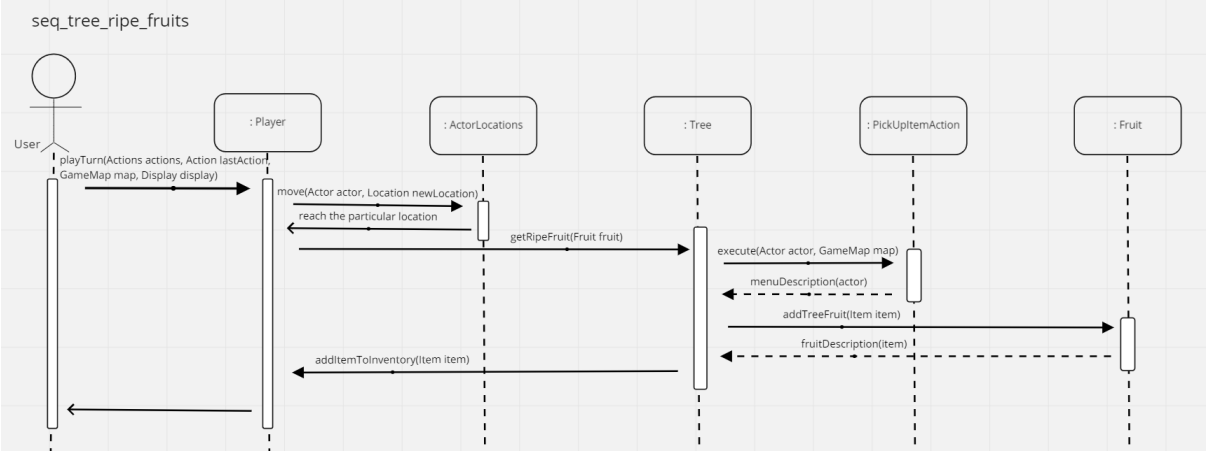
Team: Tute01Team78

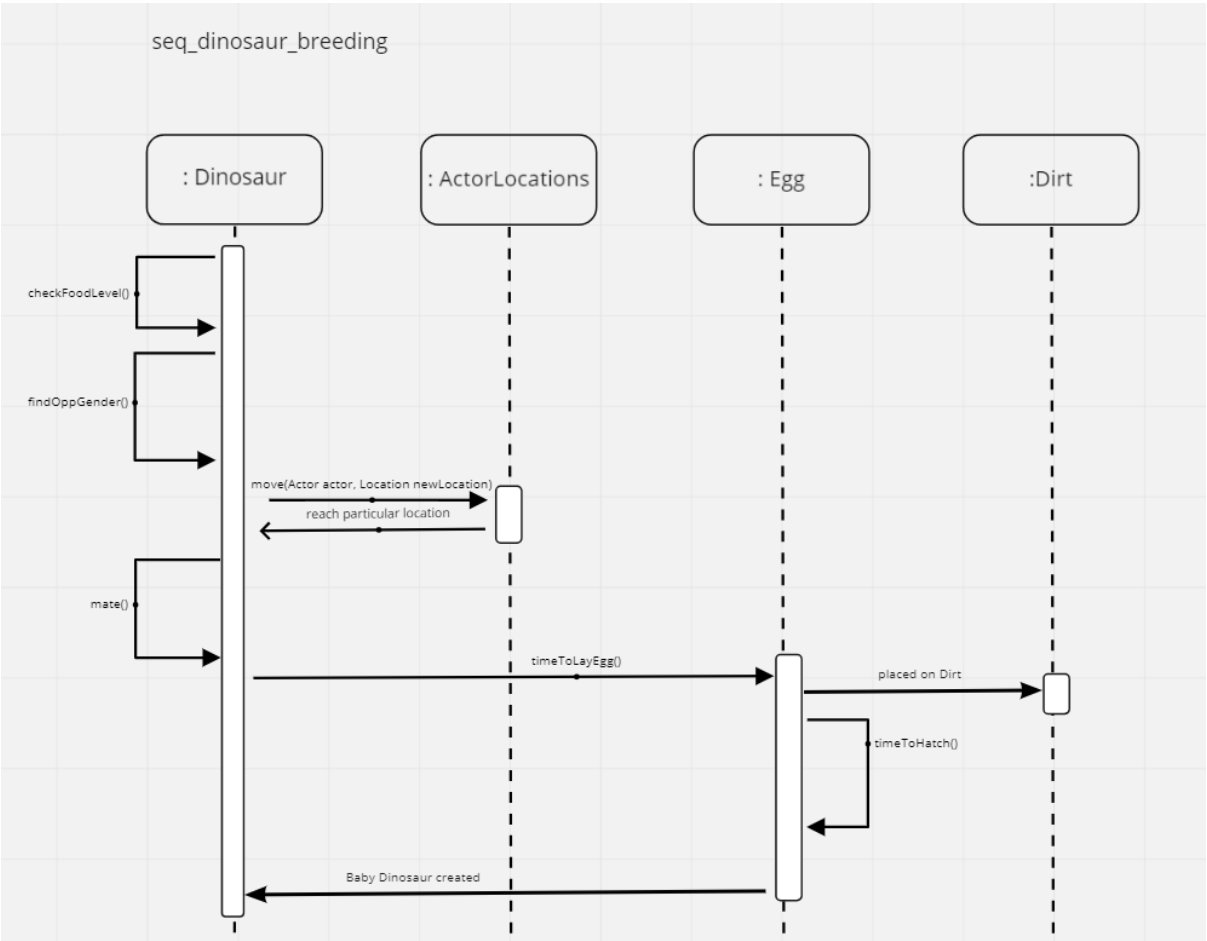
Sequence Diagram



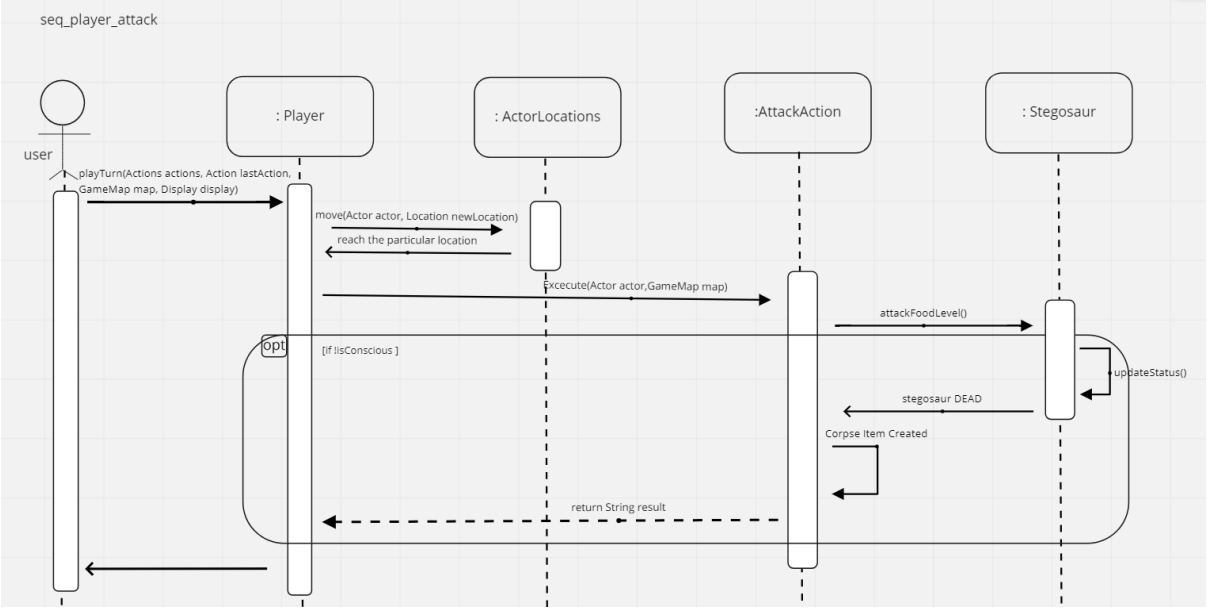


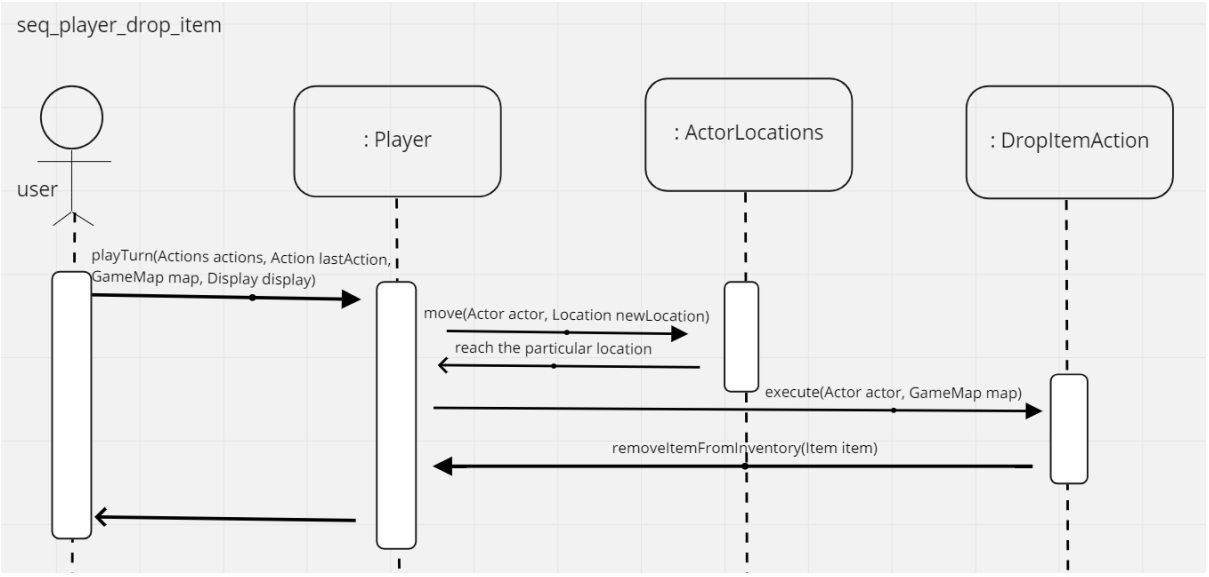
Team: Tute01Team78





Team: Tute01Team78





Team: Tute01Team78