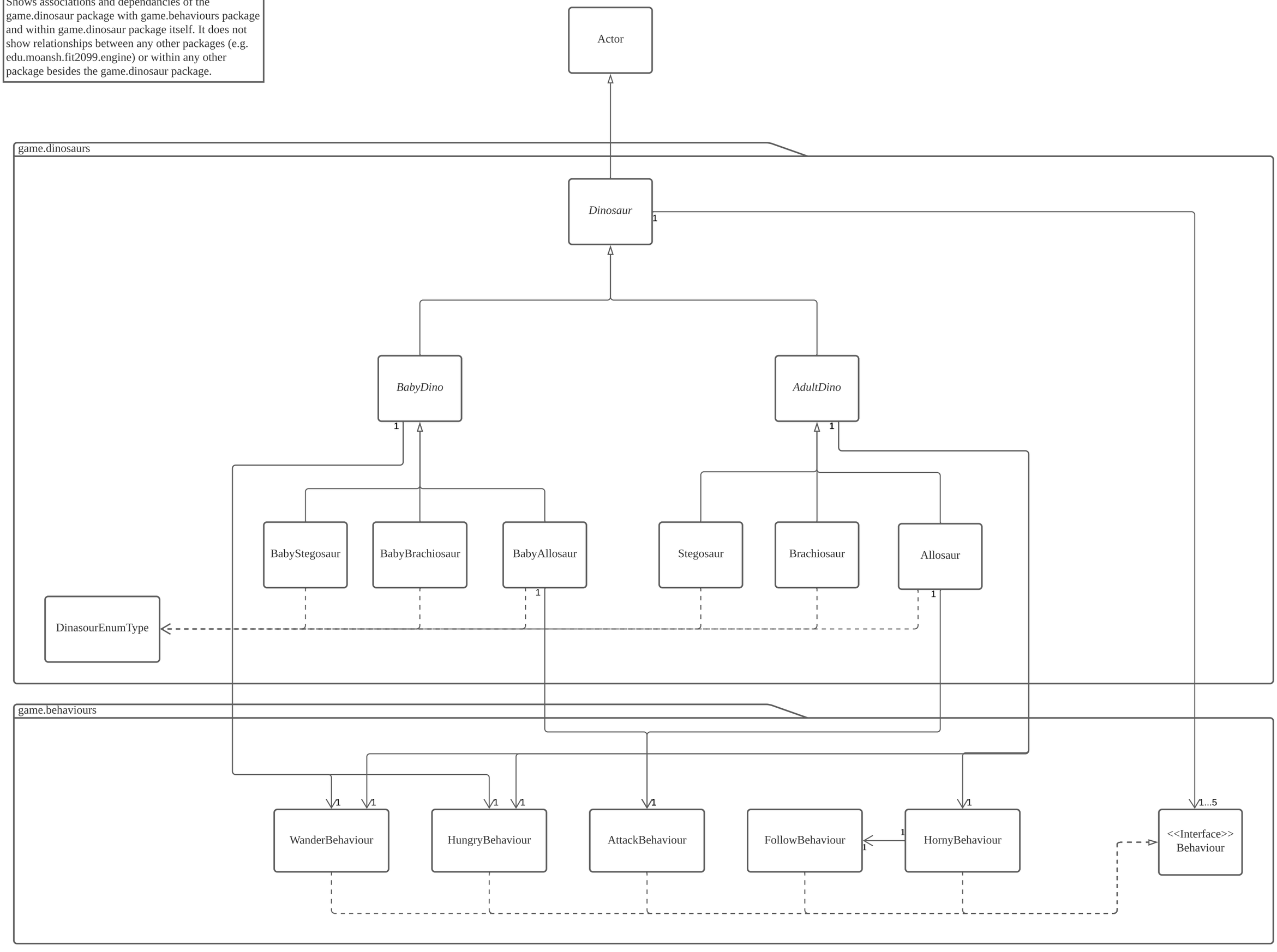
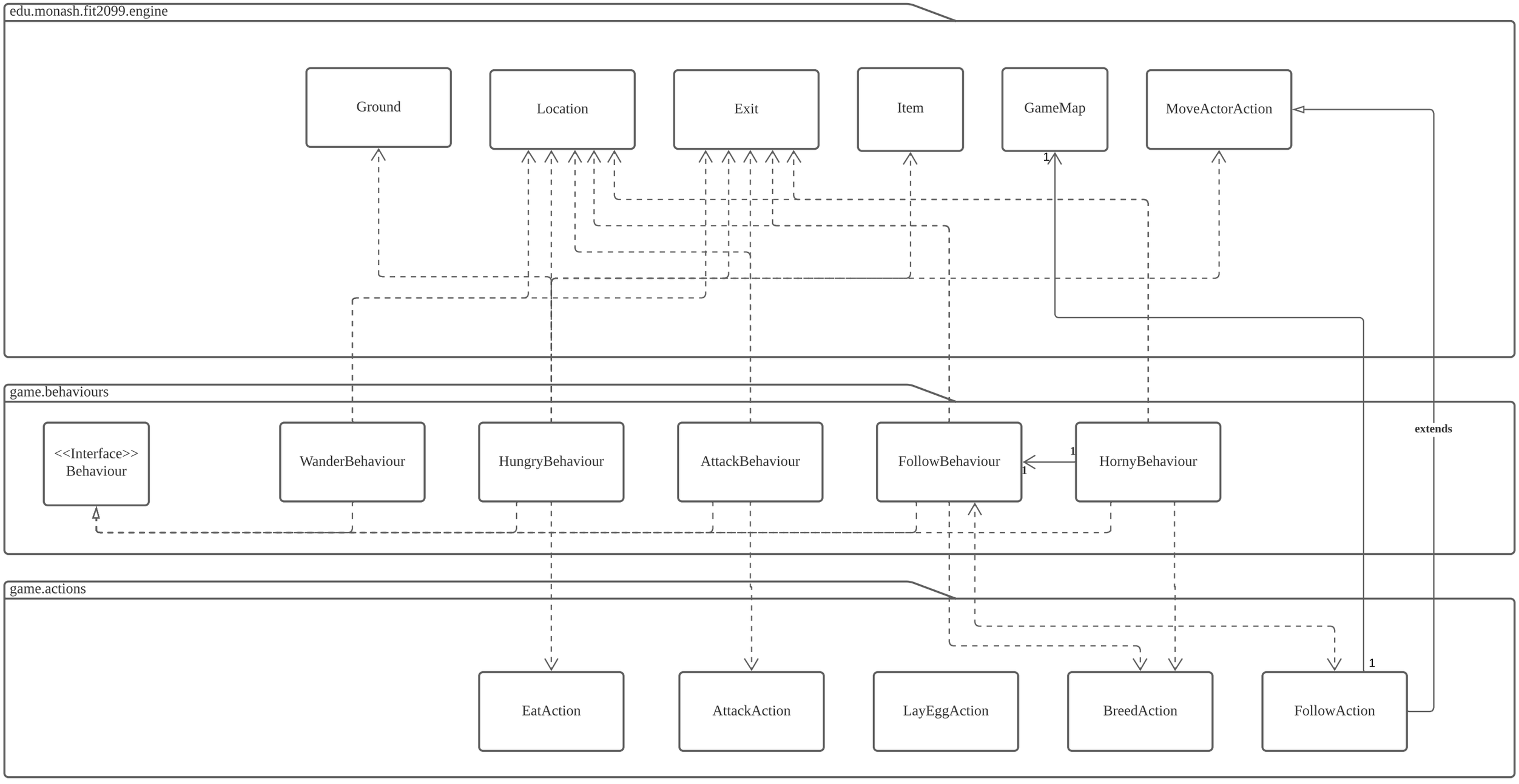
Class Diagram - Dinosaur and Behaviour

Shows associations and dependancies of the game.dinosaur package with game.behaviours package and within game.dinosaur package itself. It does not show relationships between any other packages (e.g. edu.moansh.fit2099.engine) or within any other package besides the game.dinosaur package.



Class Diagram - Behaviours

Shows associations and dependancies of the game.behaviour package excluding relationships with game.dinosaur (see Class Diagram - Dinosaur and Behaviours). It does not show relationships between it and game.dinosaur or relationships within any other package besides game.behaviour.



Class Diagram - Actions

Shows associations and dependancies of the game.actions package. It does not show relationships within other packages.

game.actions

Action

QuitGameAction

PickFruitAction

FeedAction

DieAction

BuyAction

GrowUpAction

EatAction

AttackAction

LayEggAction

BreedAction

MoveActorAction

FollowAction

edu.monash.fit2099.engine

Display

PortableItem

Item

Location

FollowBehaviour

GameMap

Actor

Game

game.items

Fruit

Corpse

Player

game.dinosaurs

BabyDino

Dinosaur

Allosaur

BabyAllosaur

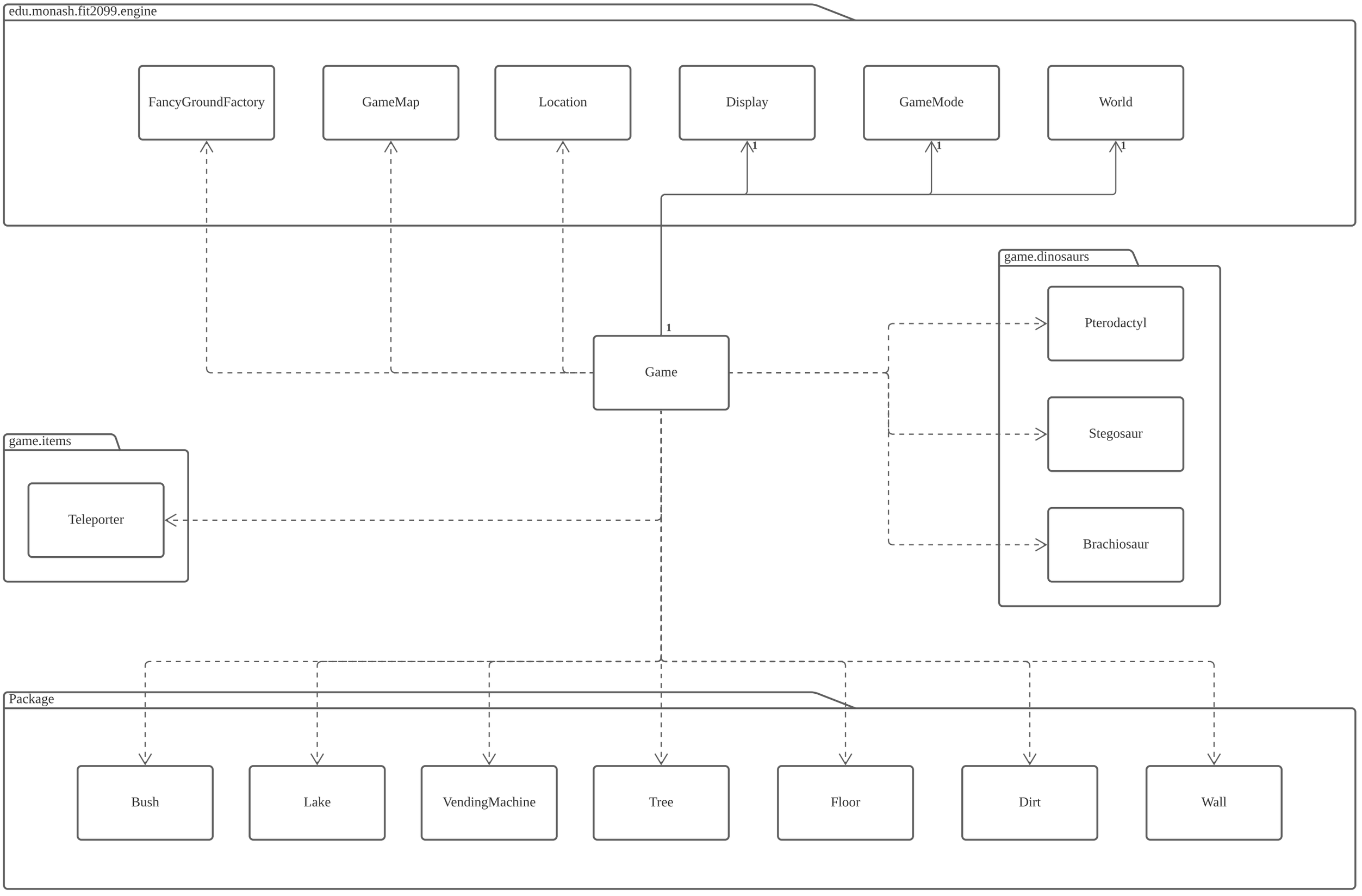
AdultDino

interfaces

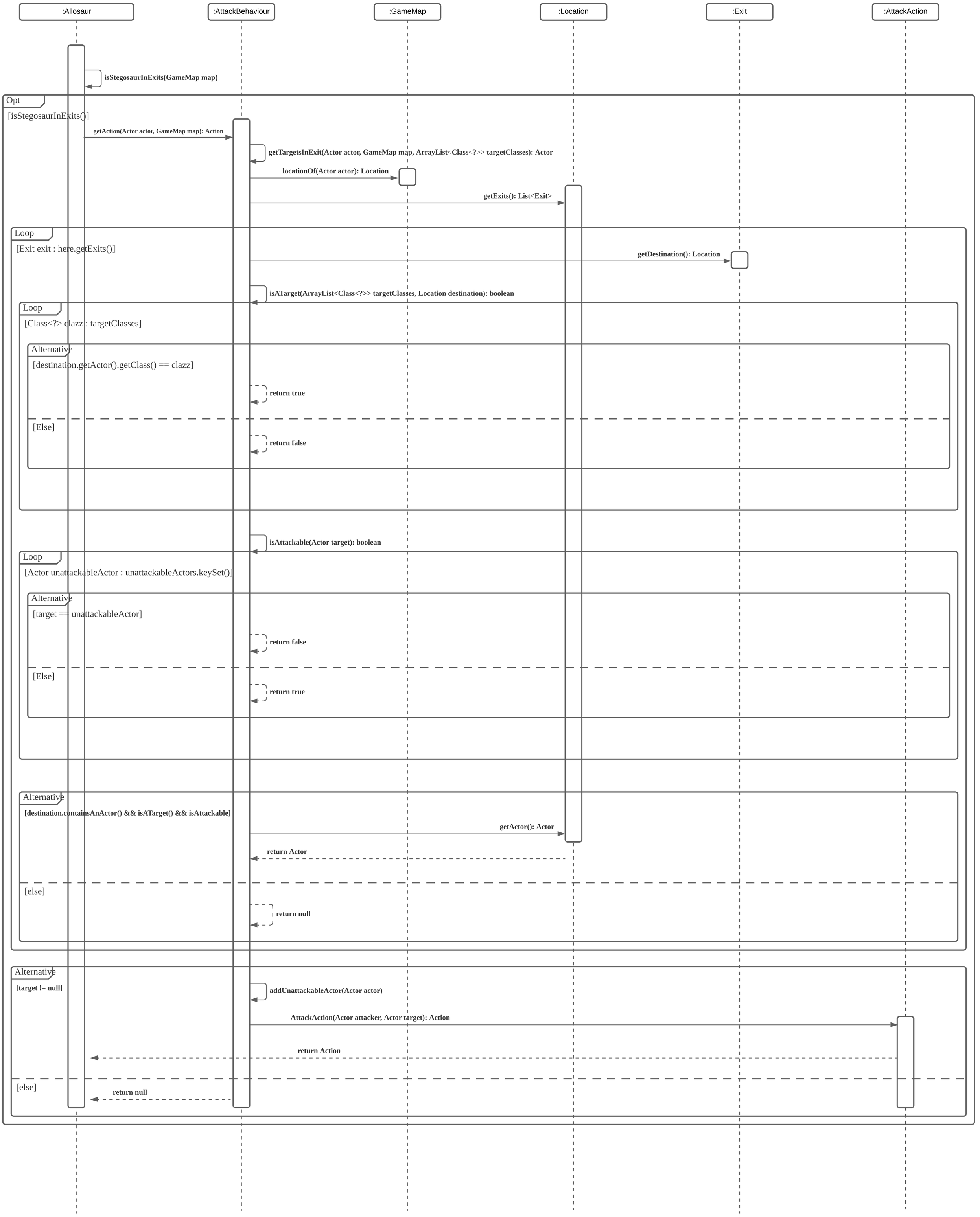
<<Interface>>
hadFood

Class Diagram - Game

Shows associations and dependancies of the Game. It does not show relationships between other classes.



Details Allosaur's AttackBehaviour scenario



Details Stegosaur HornyBehaviour scenario

stegosaur :Stegosaur
:HornyBehaviour
:GameMap
:Util
:Location
:Exit
:BreedAction
:FollowBehaviour

getAction(Actor actor, GameMap map): Action

Alternative
[actor.isFemale() && actor.isPregnant()]

getPossibleTargets(AdultDino dino, GameMap map): List

Loop
for (int x : xNumberRange)

Loop
for (int y : yNumberRange)

at(int x, int y): Location
getActorAt(Location location) : Actor

Opt
[(possibleTarget != null) &&
(possibleTarget != dino) &&
(possibleTarget.getClass() == dino.getClass()) &&
(dino.isFemale() ^ ((AdultDino) possibleTarget).isFemale()) &&
((dino.isFemale() && !(dino.isPregnant())) || (((AdultDino) possibleTarget).isFemale() && !((AdultDino) possibleTarget).isPregnant()))]

locationsWithTargets.add(map.locationOf(possibleTarget));

Alternative
[locationsWithTargets != null]

getNearestTarget(Actor actor, GameMap map, List<Location> locationsWithTargets)
locationOf(Actor actor) : Location

Loop
[Location location : locationsWithTargets]

distance(Location a, Location b): int

Opt
[distance > newDistance]

closestActorLocation = location

getActorAt(Location location) : Actor

Alternative
[target != null]

isTargetInExit(Actor target, Actor actor, GameMap map): boolean
locationOf(Actor actor): Location
getExits(): List<Exit>

Loop
[Exit exit : here.getExits()]

getDestination(): Location

Alternative
[destination.getActor() == target]

return true
BreedAction(AdultDino target)
return BreedAction

[Else]

return false
FollowBehaviour(Actor subject)
getAction(Actor actor, GameMap map): Action
return MoveActorAction

[Else]
return null

[Else]
return null

[Else]
return null

Details an instance of HungryBehavior being called by an actor

actor: Actor
map: GameMap
: HungryBehavior
exit: Exit
here: Location
aLocation: Location
closestFoodLocation: Location
aGround:Ground
bGround:Ground
food: ArrayList
item: Item
: Util

getAction(actor, map)

Alternative
[actor is an instance of Dinosaur]

getFood(): foodClasses
getFromTheseEatsThese(): groundToFoodMap
locationOf(actor): here
getFoodLocation(here, map, foodClasses, groundToFoodMap
new
locationsWithFood: ArrayList
getXRange(): xRange
getYRange(): yRange

Loop
[x in xRange]
Loop
[y in yRange]
Alternative
[groundToFoodMap cointains Ground class]
at(x , y): aLocation
getItems(): items
Alternative
[items isn't empty]
Loop
[for item in items]
getClass(): itemClass
Alternative
[foodClasses contains itemClass]
at(x, y): aLocation
add(aLocation)
at(x, y): aLocation
getGround(): aGround
isEmpty()
getClass(): groundClass
Alternative
[aGround implements hasFood and is not empty and groundToFoodMap contains groundClass
at(x, y): aLocation
add(aLocation)

Loop
[location in locationsWithFood]
distance(location, here)

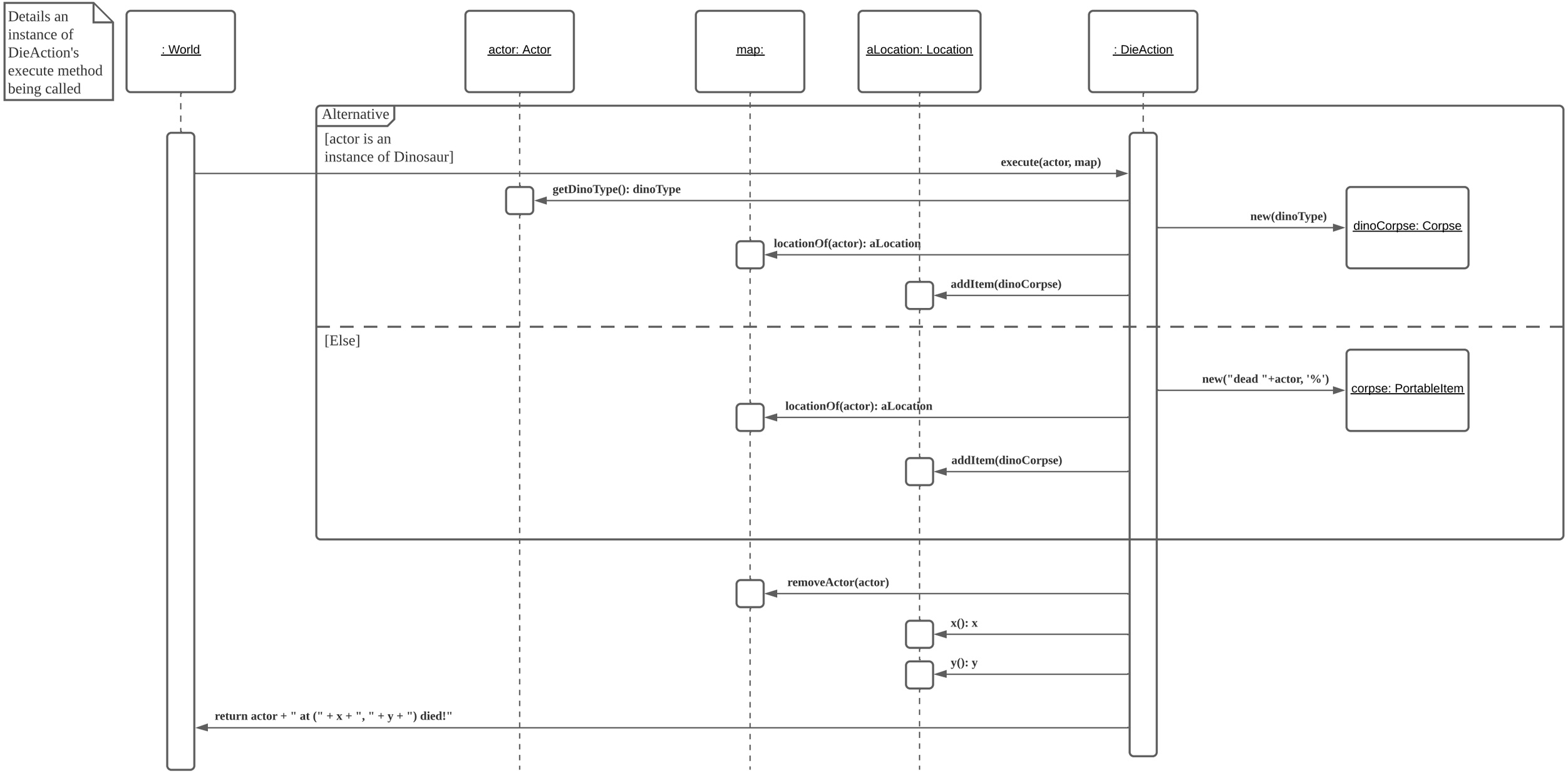
return closestFoodLocation

Alternative
[closestFoodLocation is null]
return null

Alternative
[closestFoodLocation == here]
getGround(): bGround
isEmpty()
getClass(): groundClass
Alternative
[bGround implements hasFood and is not empty and contains groundClass]
getClass(): actorClass
getClass(): groundClass
Alternative
[actorClass is Brachiosaur or BabyBrachiosaur and groundClass is Tree]
getFoodInstance(): foodInstance
getFood(): food
size(): quantity
new(foodInstance, bGround, ,quantity)
eatActionA: EatAction
return eatActionA
new(foodInstance, bGround, 1)
eatActionB: EatAction
return eatActionB
[Else]
getItems(): items
Loop
[item in items]
getClass(): itemClass
Alternative
[foodClasses contains itemClass]
new(item, closestFoodLocation, 1)
eatActionC: EatAction
return eatActionC

distance(here, closestFoodLocation): currentDistance
getExits(): exits

Loop
[exit in exits]
getDestination: aLocation
canActorEnter(actor)
Alternative
[actor can enter]
distance(aLocation, closestFoodLocation): newDistance
Alternative
[newDistance<currentDistance]
getName(): eName
new(aLocation, eName)
MAAction: MoveActorAction
return MAAction
[Else]
return null


Details an instance of DieAction's execute method being called
: World
actor: Actor
map:
aLocation: Location
: DieAction
Alternative
[actor is an instance of Dinosaur]
execute(actor, map)
getDinoType(): dinoType
new(dinoType)
dinoCorpse: Corpse
locationOf(actor): aLocation
addItem(dinoCorpse)
[Else]
new("dead "+actor, '%')
corpse: PortableItem
locationOf(actor): aLocation
addItem(dinoCorpse)
removeActor(actor)
x(): x
y(): y
return actor + " at (" + x + ", " + y + ") died!"

edu.monash.fit2099

engine

WeaponItem

Item

1

Action

Ground

game.actions

FollowAction

BreedAction

FollowAction

LayEggAction

1

1

BuyAction

PickFruitAction

DieAction

EatAction

FeedAction

game.grounds

game.items

VendingMachine

LaserGun

PortableItem

<<interface>>
hasFood

Fruit

Tree

Bush

Dirt

Corpse

Egg

VegetarianMealKit

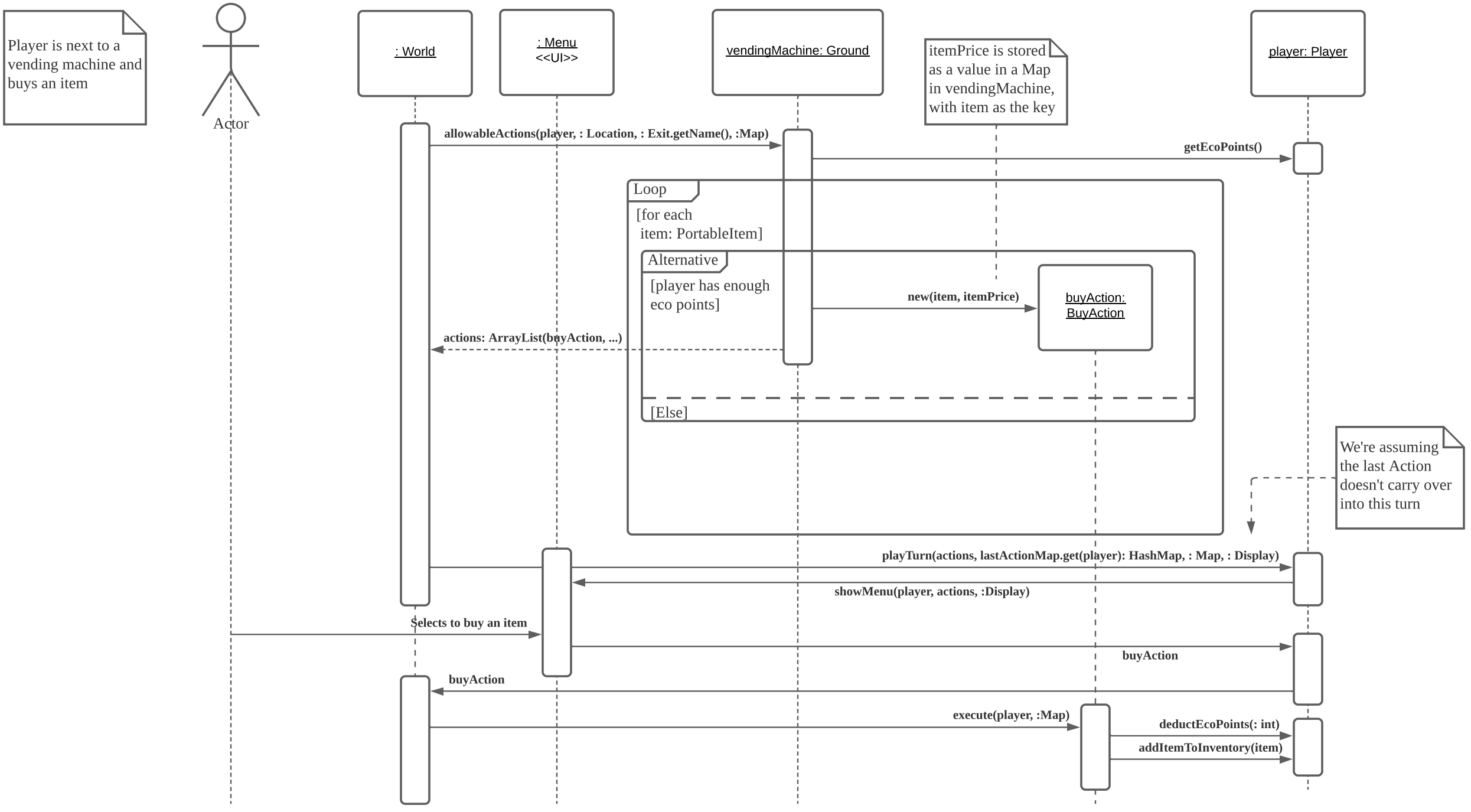
CarnivoreMealKit

game.dinosaurs

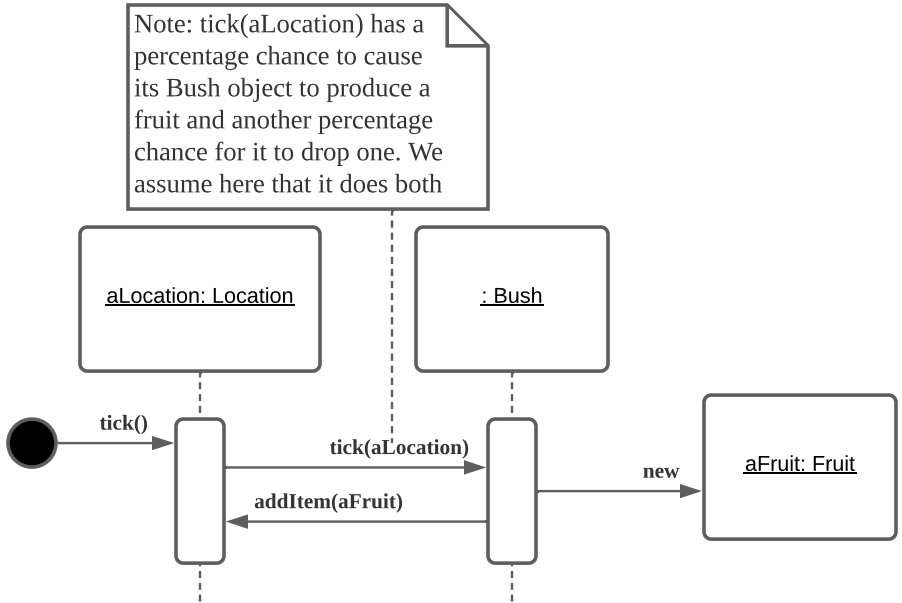
Dinosaur

Brachiosaur

Dinosaur

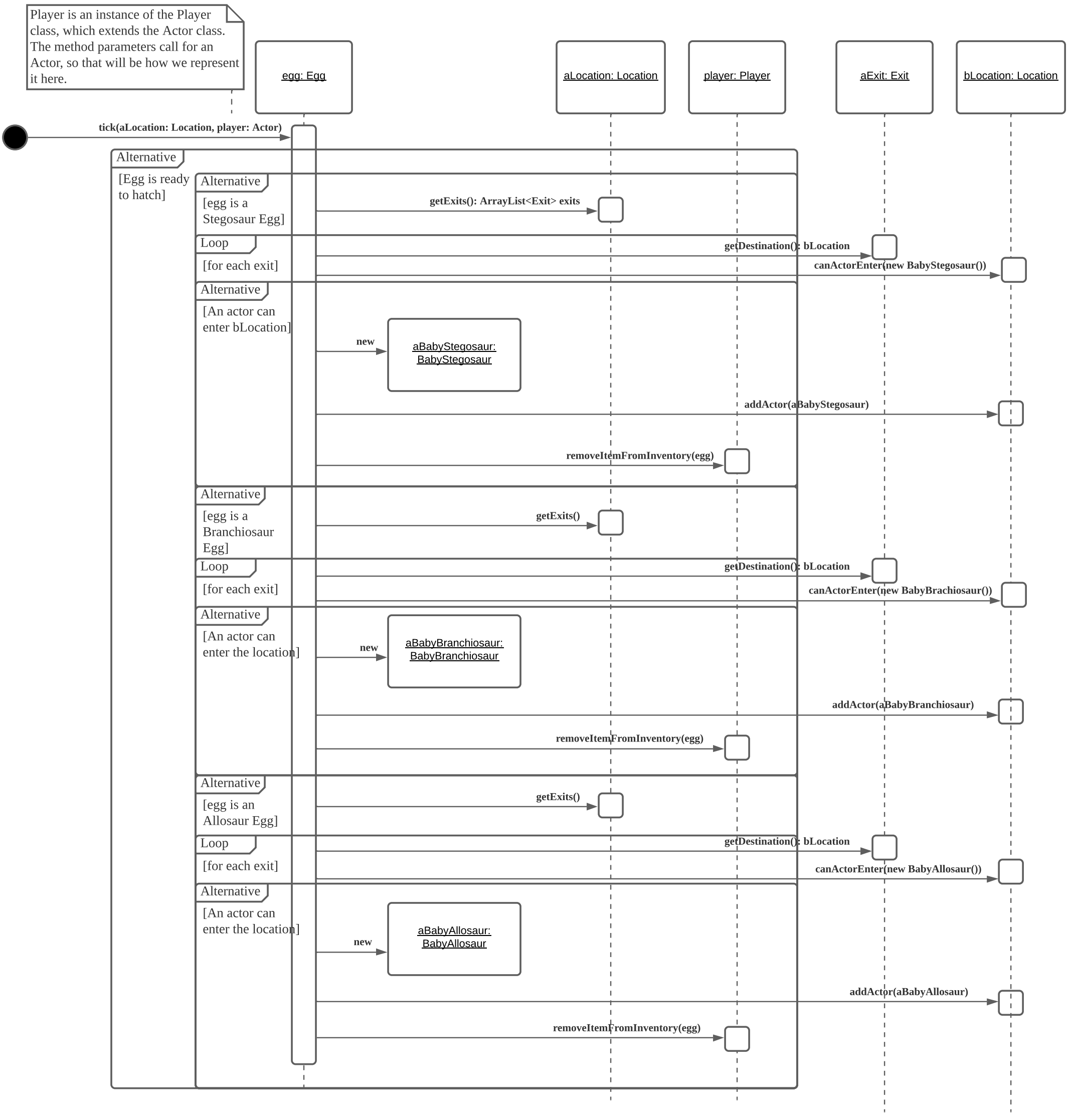

Player is next to a vending machine and buys an item
Actor
: World
: Menu
<<UI>>
vendingMachine: Ground
itemPrice is stored as a value in a Map in vendingMachine, with item as the key
player: Player
allowableActions(player, : Location, : Exit.getName(), :Map)
getEcoPoints()
Loop
[for each item: PortableItem]
Alternative
[player has enough eco points]
new(item, itemPrice)
buyAction: BuyAction
actions: ArrayList(buyAction, ...)
[Else]
We're assuming the last Action doesn't carry over into this turn
playTurn(actions, lastActionMap.get(player): HashMap, : Map, : Display)
showMenu(player, actions, :Display)
Selects to buy an item
buyAction
buyAction
execute(player, :Map)
deductEcoPoints(: int)
addItemToInventory(item)

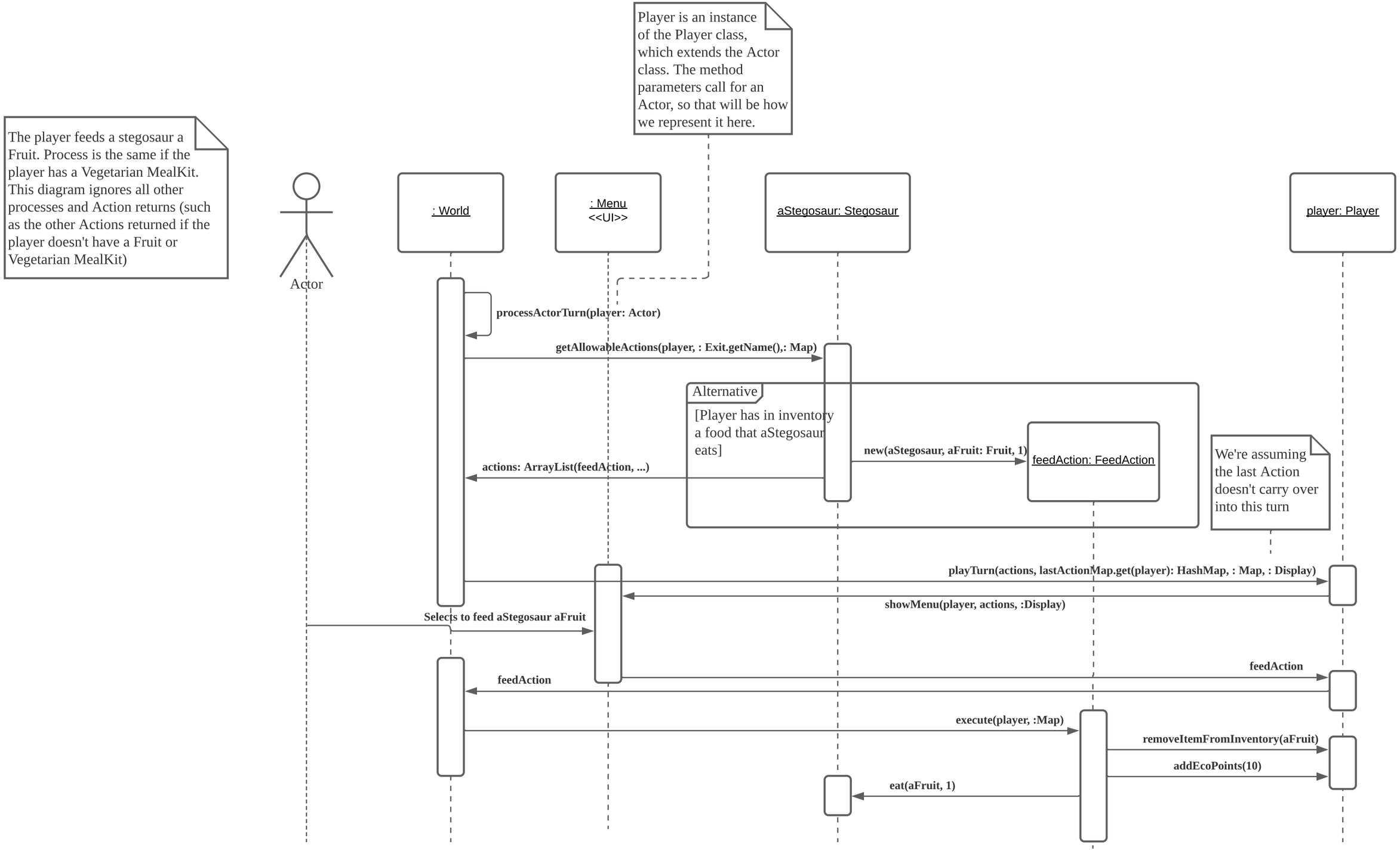
Bush produces and drops a fruit
(The process with a Tree is the same, just using different probabilities)

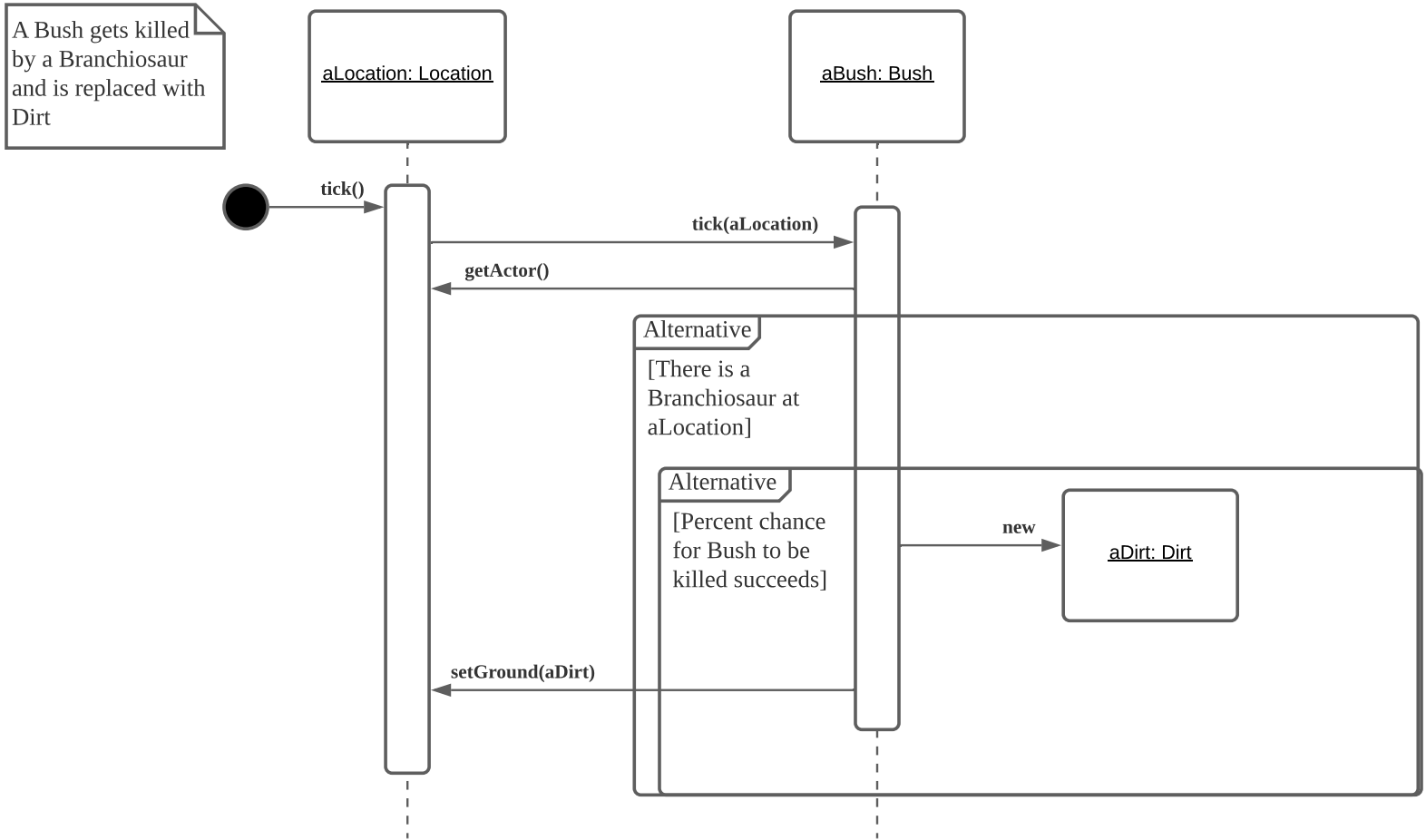


Process for a stegosaur egg in the player's inventory hatching. (Process for an egg on the ground is largly the same except for the : Egg.tick() method called.

Player is an instance of the Player class, which extends the Actor class. The method parameters call for an Actor, so that will be how we represent it here.




The player feeds a stegosaur a Fruit. Process is the same if the player has a Vegetarian MealKit. This diagram ignores all other processes and Action returns (such as the other Actions returned if the player doesn't have a Fruit or Vegetarian MealKit)
Player is an instance of the Player class, which extends the Actor class. The method parameters call for an Actor, so that will be how we represent it here.
Actor
: World
: Menu
<<UI>>
aStegosaur: Stegosaur
player: Player
processActorTurn(player: Actor)
getAllowableActions(player, : Exit.getName(),: Map)
Alternative
[Player has in inventory a food that aStegosaur eats]
new(aStegosaur, aFruit: Fruit, 1)
feedAction: FeedAction
actions: ArrayList(feedAction, ...)
We're assuming the last Action doesn't carry over into this turn
playTurn(actions, lastActionMap.get(player): HashMap, : Map, : Display)
showMenu(player, actions, :Display)
Selects to feed aStegosaur aFruit
feedAction
feedAction
execute(player, :Map)
removeItemFromInventory(aFruit)
addEcoPoints(10)
eat(aFruit, 1)


A Bush gets killed by a Branchiosaur and is replaced with Dirt
aLocation: Location
aBush: Bush
tick()
tick(aLocation)
getActor()
Alternative
[There is a Branchiosaur at aLocation]
Alternative
[Percent chance for Bush to be killed succeeds]
new
aDirt: Dirt
setGround(aDirt)