edu.monash.fit2099.engine

Ground

Location

Exit

MoveActorAction

Item

game

Actor

Player

game.dinosaurs

game.behaviours

1..4

<<Interface>>
Behaviour

FeedAction

Dinosaur

1

1

AttackBehaviour

HungryBehaviour

HornyBehaviour

WanderBehaviour

FollowBehaviour

Egg

BabyDino

Stegosaur

Brachiosaur

Allosaur

BabyStegosaur

BabyBrachiosaur

BabyAllosaur

BreedAction

edu.monash.fit2099

engine

WeaponItem

Item

Action

Ground

game.actions

BreedAction

BuyAction

PickFruitAction

EatFromGroundAction

EatFromPlantAction

game.grounds

game.items

VendingMachine

LaserGun

PortableItem

<<interface>>
hasFood

Fruit

Tree

Bush

Dirt

Corpse

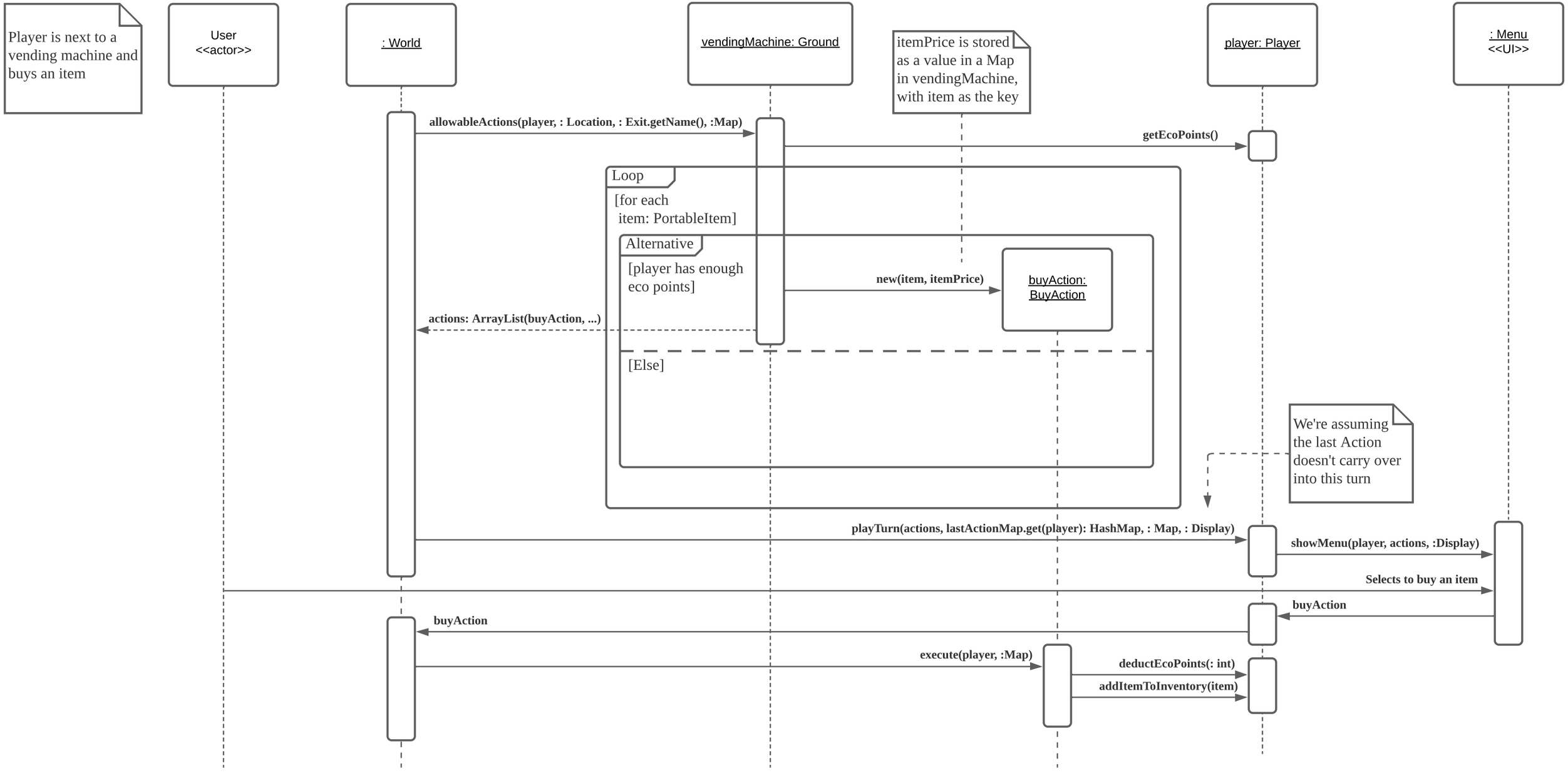
Egg

MealKit

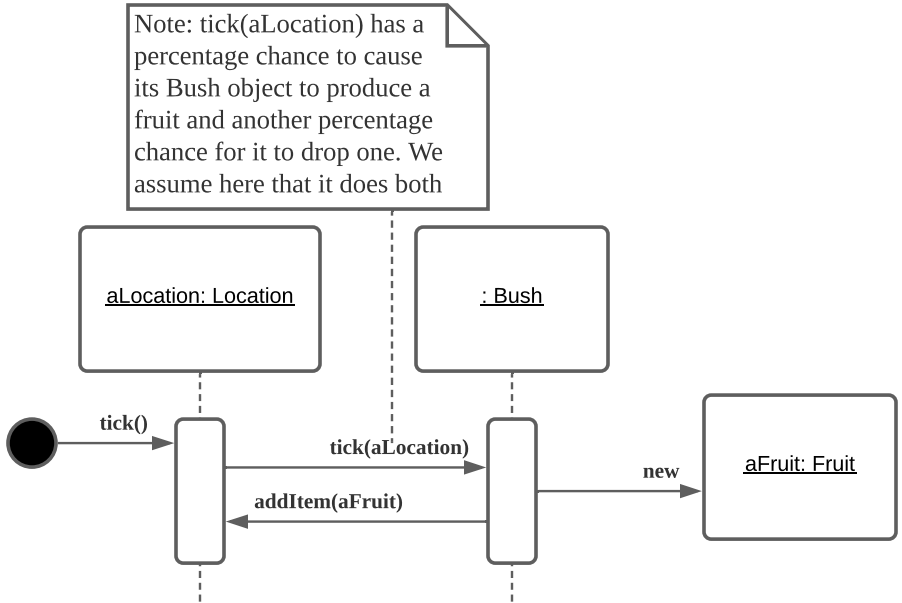
game.dinosaurs

Dinosaur

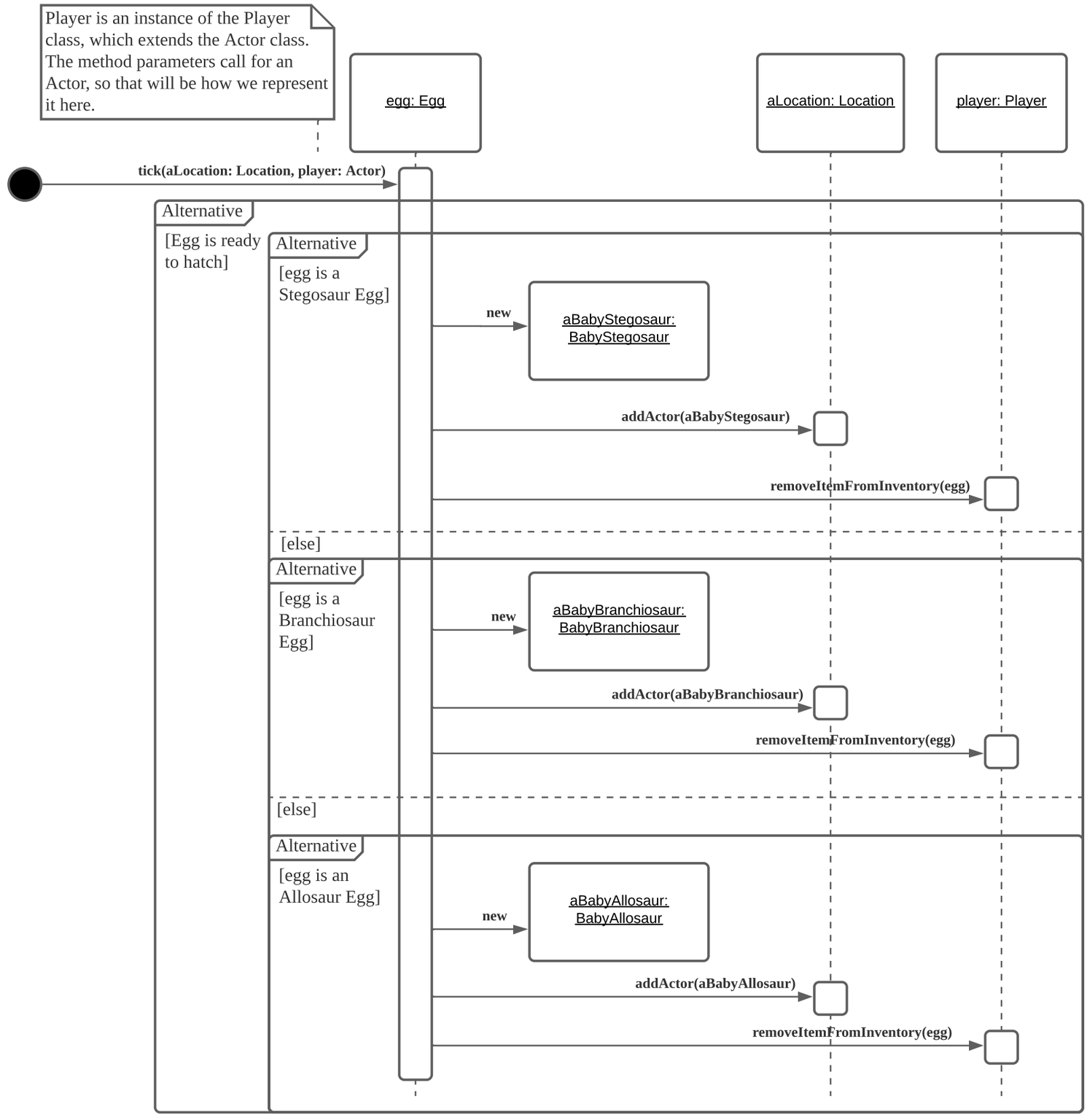
Branchiosaur

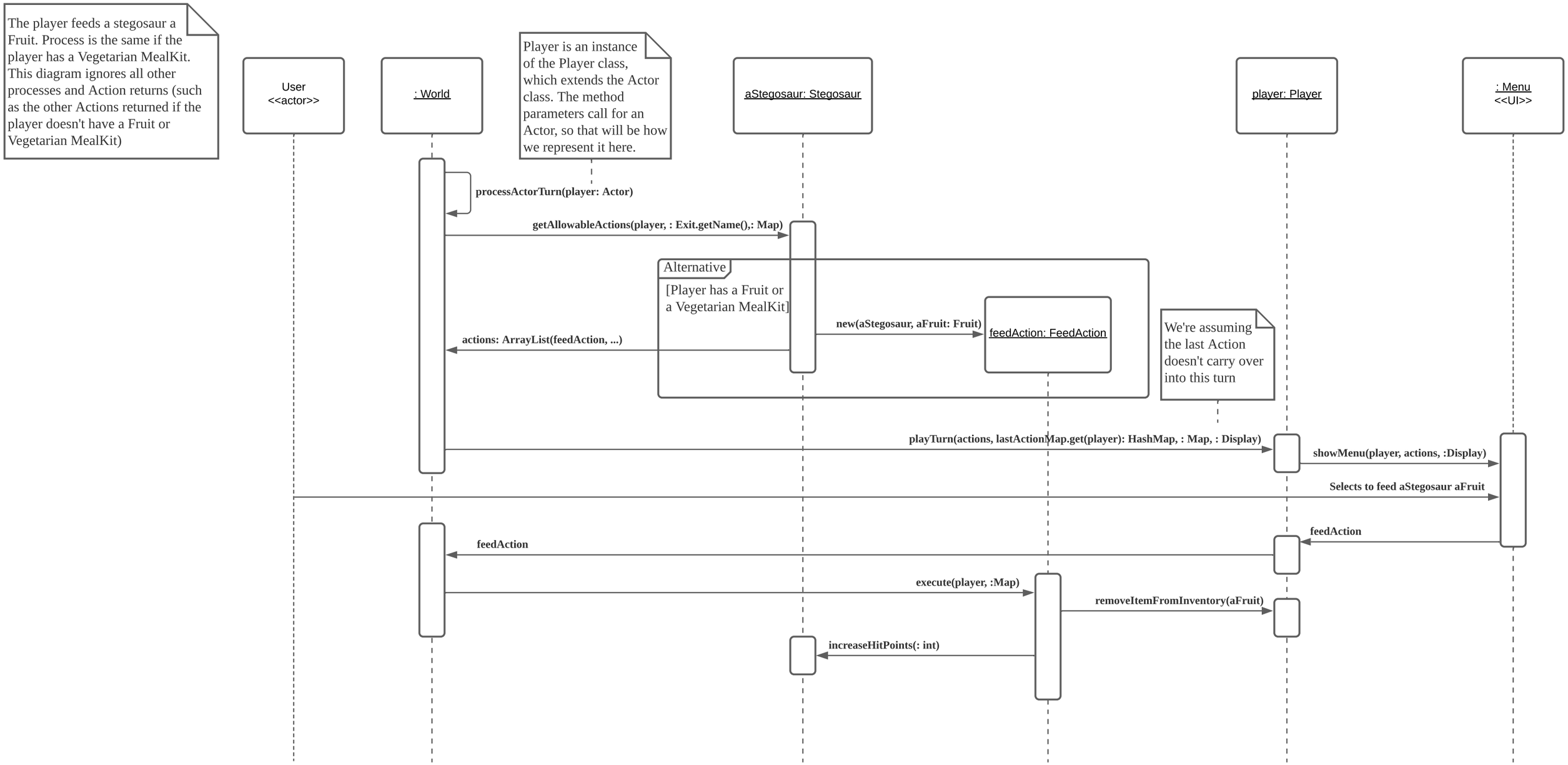

Player is next to a vending machine and buys an item
User
<<actor>>
: World
vendingMachine: Ground
itemPrice is stored as a value in a Map in vendingMachine, with item as the key
player: Player
: Menu
<<UI>>
allowableActions(player, : Location, : Exit.getName(), :Map)
getEcoPoints()
Loop
[for each item: 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)

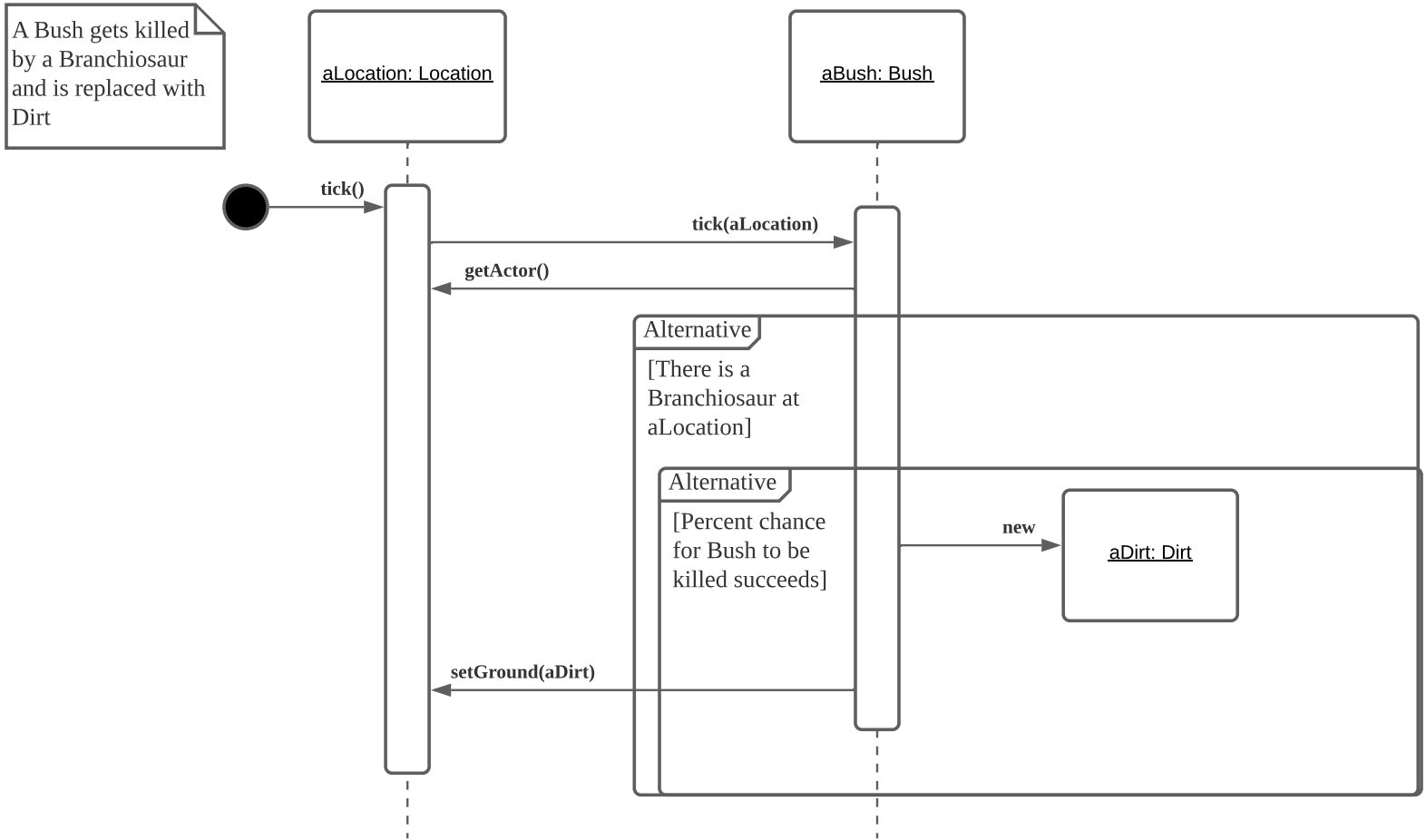
Bush produces and drops a fruit
(The process with a Tree is the same, just using different probabilities)



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.




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)
User
<<actor>>
: World
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.
aStegosaur: Stegosaur
player: Player
: Menu
<<UI>>
processActorTurn(player: Actor)
getAllowableActions(player, : Exit.getName(),: Map)
Alternative
[Player has a Fruit or a Vegetarian MealKit]
new(aStegosaur, aFruit: Fruit)
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)
increaseHitPoints(: int)


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)

Details Allosaur's AttackBehaviour scenario

:World
:Allosaur
:AttackBehaviour
:GameMap
:Location
:Exit
:AttackAction
:HornyBehaviour
:HungryBehaviour
:WanderBehaviour

playTurn(Actions actions, Action lastAction, GameMap map, Display display)

nearStegosaur(GameMap map)

Alternative

if there is a Stegosaurus nearby

getAction(Actor actor, GameMap map): Action

getNearbyStegosaur(Actor actor, GameMap map): Actor

locationOf(Actor actor): Location

getExits(): List<Exit>

Loop

for (Exit exit : here.getExits())

getDestination(): Location

getActor(): Actor

Alternative

if destination actor is a Stegosaur

return Actor

if destination actor is not a Stegosaur

return null

getOffLimitsStegosaurs(): ArrayList<Stegosaur>

ArrayList<Stegosaur>

Loop

for (Stegosaur offLimitsStegosaur : offLimitsStegosaurs)

Alternative

if offLimitsStegosaur contains stegosaur

AttackAction(Actor target)

return AttackAction

heal(int points)

addOffLimitsStegosaurs(Stegosaur stegosaur)

return AttackAction

if destination actor is not a Stegosaur

return null

if hitPoints > 50 and there's a possible mate

getAction(Actor actor, GameMap map): Action

if hitPoints < 90 and there's a possible food source

getAction(Actor actor, GameMap map): Action

else

getAction(Actor actor, GameMap map): Action

return Action

Details Stegosaur HornyBehaviour scenario

stegosaur :Stegosaur | :HornyBehaviour | :GameMap | :Dinosaur | :Location | :Exit | :BreedAction | :FollowBehaviour

Text

playTurn(Actions actions, Action lastAction, GameMap map, Display display)

getAction(Actor actor, GameMap map): Action

getNearestTarget(Actor actor, GameMap map): Actor

getXRange(): NumberRange

getYRange(): NumberRange

Loop
for (int x : xNumberRange)

Loop
for (int y : yNumberRange)

at(int x, int y): Location

getActorAt(Location location) : Actor

Alternative
if target and actor are of the same class

isFemale():Boolean

Alternative
if actor is female XOR target is female

Alternative
if actor is female and not pregnant OR target is female and not pregnant

add Location to a list of possible target locations

[Else]

[Else]

[Else]

locationOf(Actor actor) : Location

Loop
For locations in possible target locations

Alternative
If distance > newDistance

closestActorLocation = location

[Else]

getActorAt(Location location):Location

return Location

isTargetInExit(Actor target, Actor actor, GameMap map)

getExits():List<Exit>

Loop
For list of exits

getDestination() :Location

Alternative
destination contains target

return True

none of the exit destinations contain target

return True

Alternative
if isTargetInExit returns true

BreedAction(Actor actor, Actor target)

return BreedAction

if isTargetInExit returns false

getAction():Action

return MoveActorAction

Details a HungryBehavior scenario

: World
aDinosaur: Dinosaur
: HungryBehavior
map: GameMap
aLocation: Location
exit: Exit
aGround: Ground
bLocation: Location
bGround: Ground

playTurn(Actions actions, Action lastAction, GameMap map, Display display)

Alternative
[if hungry and no possible mate]

getAction(aDinosaur, map: GameMap)
locationOf(aDinosaur): aLocation
getExits(): ArrayList<Exit> exits

Loop
[for each exit in exits (Store output in an ArrayList exitLocations)]
getDestination(): Location

Loop
[for each aLocation: Location]
getItems(): items
getFood(): food

Alternative
[if items contains an instance of food]
new
aAction: EatFromGroundAction
trturn aAction

Loop
[for each aLocation: Location]
getGround(): aGround
getEatsFrom(): eatenFrom

Alternative
[if aGround implements hasFood() and is of the same class as eatenFrom]
isEmpty(): truth

Alternative
[if truth is False]
new
bAction: EatFromPlantAction
return bAction

getXRange(): widh
getYRange(): height

Loop
[for x in range 0 to width]

Loop
[for y in range 0 to height]
at(x,y): bLocation
getItems(): items

Alternative
[if items includes an instance of food: add bLocation to an ArrayList locations]
getGround(): bGround

Alternative
[if bGround implements hasFood() and is of the same class as eatenFrom]
isEmpty(): truth

Alternative
[if truth is False: add bLocation to the ArrayList locations]

getShortestDistance(locations): cLocation

Loop
[for each aLocation in exitLocations]
canActorEnter(): truth

Alternative
[if truth is True]
diestance(aLocation, cLocation)

Alternative
[if aLocation is closer to cLocation than current Location]
new
cAction: MoveActorAction
reurn cAction