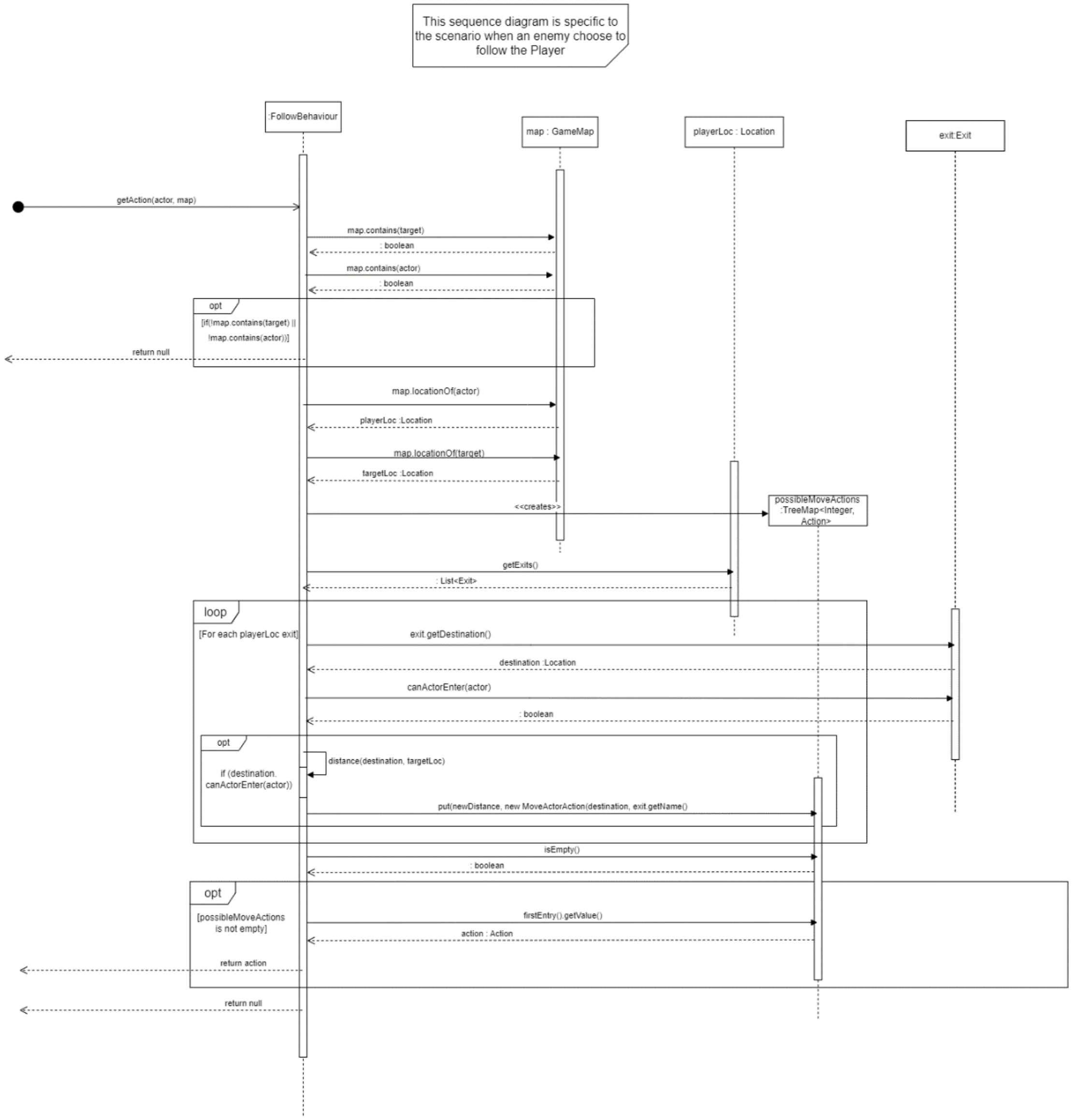
SEQUENCE DIAGRAM

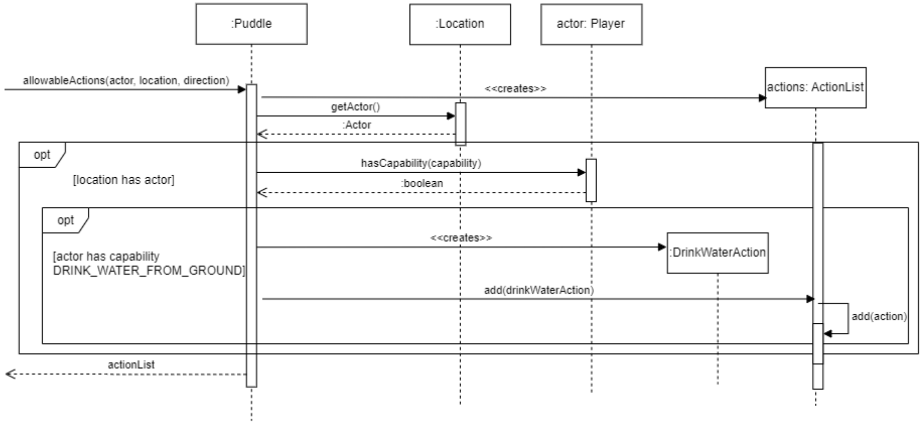
Requirement 1 (Raynen)


This sequence diagram is specific to the scenario when an enemy choose to follow the Player
:FollowBehaviour
map : GameMap
playerLoc : Location
exit:Exit
getAction(actor, map)
map.contains(target)
: boolean
map.contains(actor)
: boolean
opt
[if(!map.contains(target) || !map.contains(actor))]
return null
map.locationOf(actor)
playerLoc :Location
map.locationOf(target)
targetLoc :Location
<<creates>>
possibleMoveActions :TreeMap<Integer, Action>
getExits()
: List<Exit>
loop
[For each playerLoc exit]
exit.getDestination()
destination :Location
canActorEnter(actor)
: boolean
opt
distance(destination, targetLoc)
if (destination. canActorEnter(actor))
put(newDistance, new MoveActorAction(destination, exit.getName()
isEmpty()
: boolean
opt
[possibleMoveActions is not empty]
firstEntry().getValue()
action : Action
return action
return null

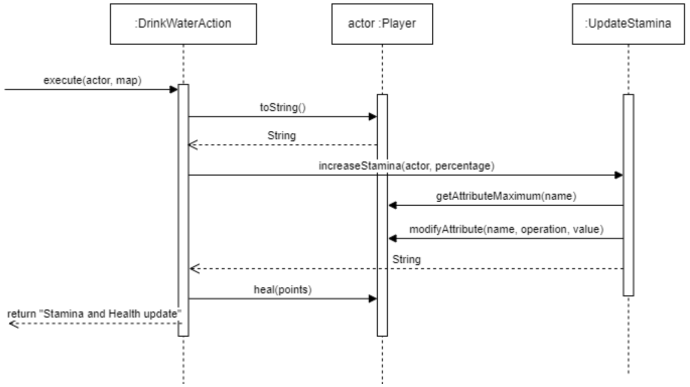
SEQUENCE DIAGRAM

Requirement 2 (Choong Lee Ann)

This sequence diagram is specific to the scenario when the actor is on the Puddle and the alloableActions of Puddle us called for the Action that the Player can get from Puddle

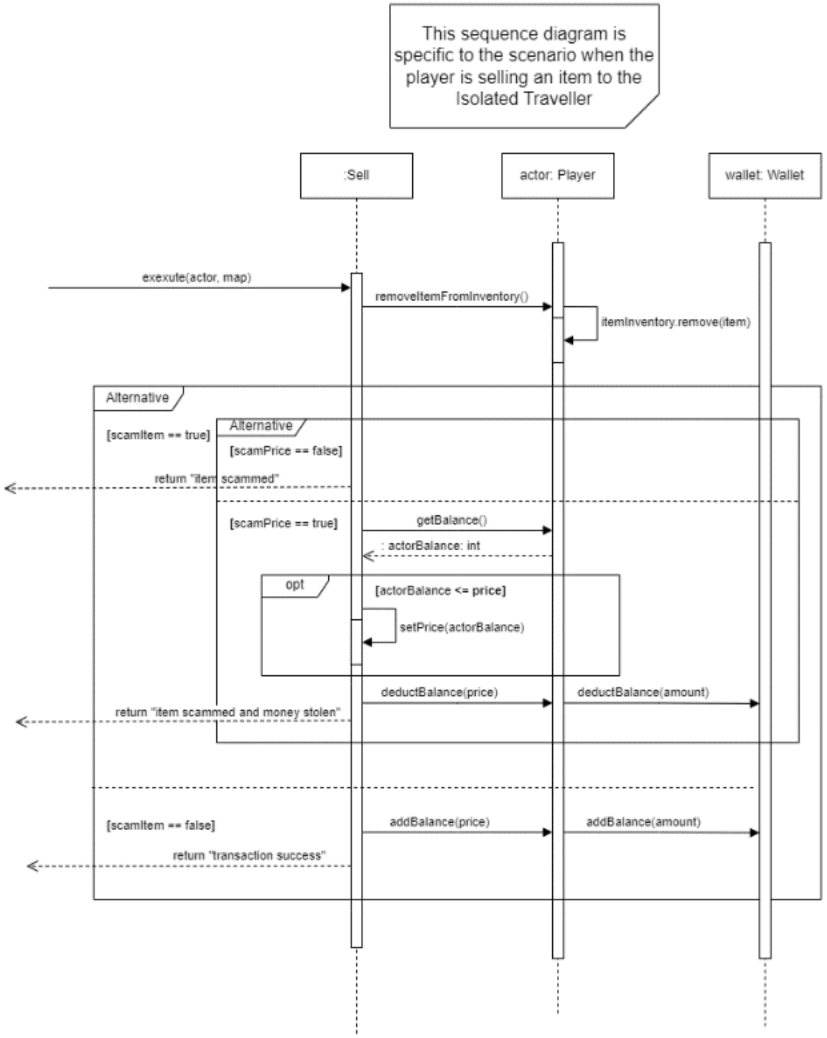

:Puddle
:Location
actor: Player
actions: ActionList
allowableActions(actor, location, direction)
<<creates>>
getActor()
:Actor
opt
[location has actor]
hasCapability(capability)
:boolean
opt
<<creates>>
:DrinkWaterAction
[actor has capability
DRINK_WATER_FROM_GROUND]
add(drinkWaterAction)
add(action)
actionList

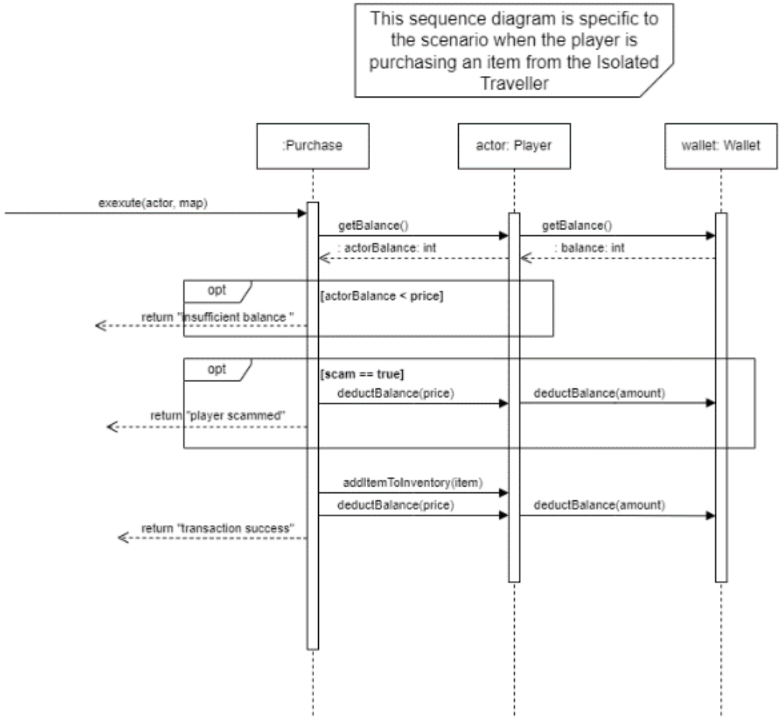
This sequence diagram is specific to the scenario when the actor chooses to Drink Water from the Puddle


:DrinkWaterAction
actor :Player
:UpdateStamina
execute(actor, map)
toString()
String
increaseStamina(actor, percentage)
getAttributeMaximum(name)
modifyAttribute(name, operation, value)
String
heal(points)
return "Stamina and Health update"

SEQUENCE DIAGRAM

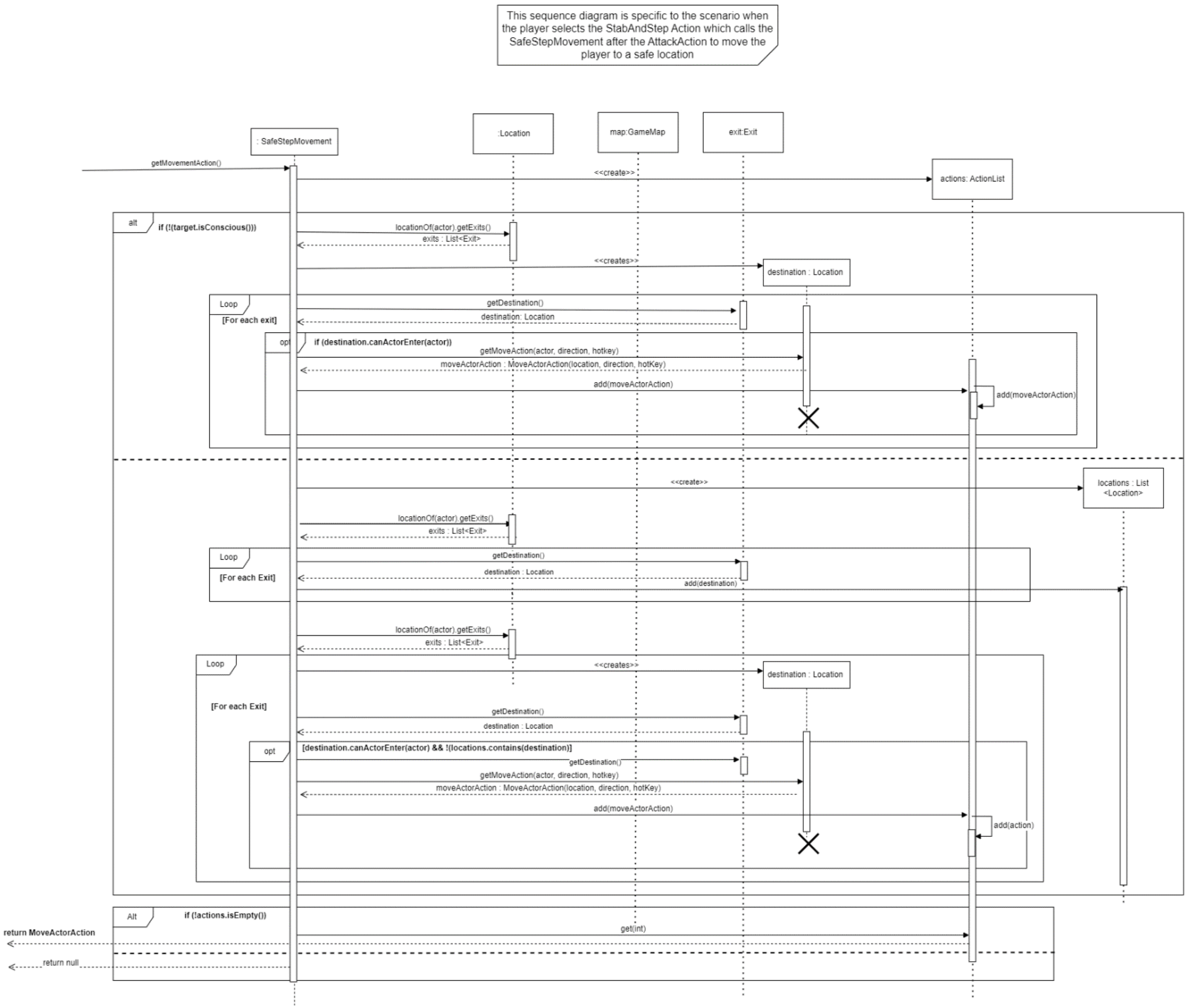
Requirement 3 (Sarviin Hari)


This sequence diagram is specific to the scenario when the player is selling an item to the Isolated Traveller
:Sell
actor: Player
wallet: Wallet
exexute(actor, map)
removeItemFromInventory()
itemInventory.remove(item)
Alternative
[scamItem == true]
Alternative
[scamPrice == false]
return "item scammed"
[scamPrice == true]
getBalance()
: actorBalance: int
opt
[actorBalance <= price]
setPrice(actorBalance)
deductBalance(price)
deductBalance(amount)
return "item scammed and money stolen"
[scamItem == false]
addBalance(price)
addBalance(amount)
return "transaction success"


This sequence diagram is specific to the scenario when the player is purchasing an item from the Isolated Traveller
:Purchase
actor: Player
wallet: Wallet
exexute(actor, map)
getBalance()
getBalance()
: actorBalance: int
: balance: int
opt
[actorBalance < price]
return "insufficient balance "
opt
[scam == true]
deductBalance(price)
deductBalance(amount)
return "player scammed"
addItemToInventory(item)
deductBalance(price)
deductBalance(amount)
return "transaction success"

SEQUENCE DIAGRAM

Requirement 4 (Jun Hirano)

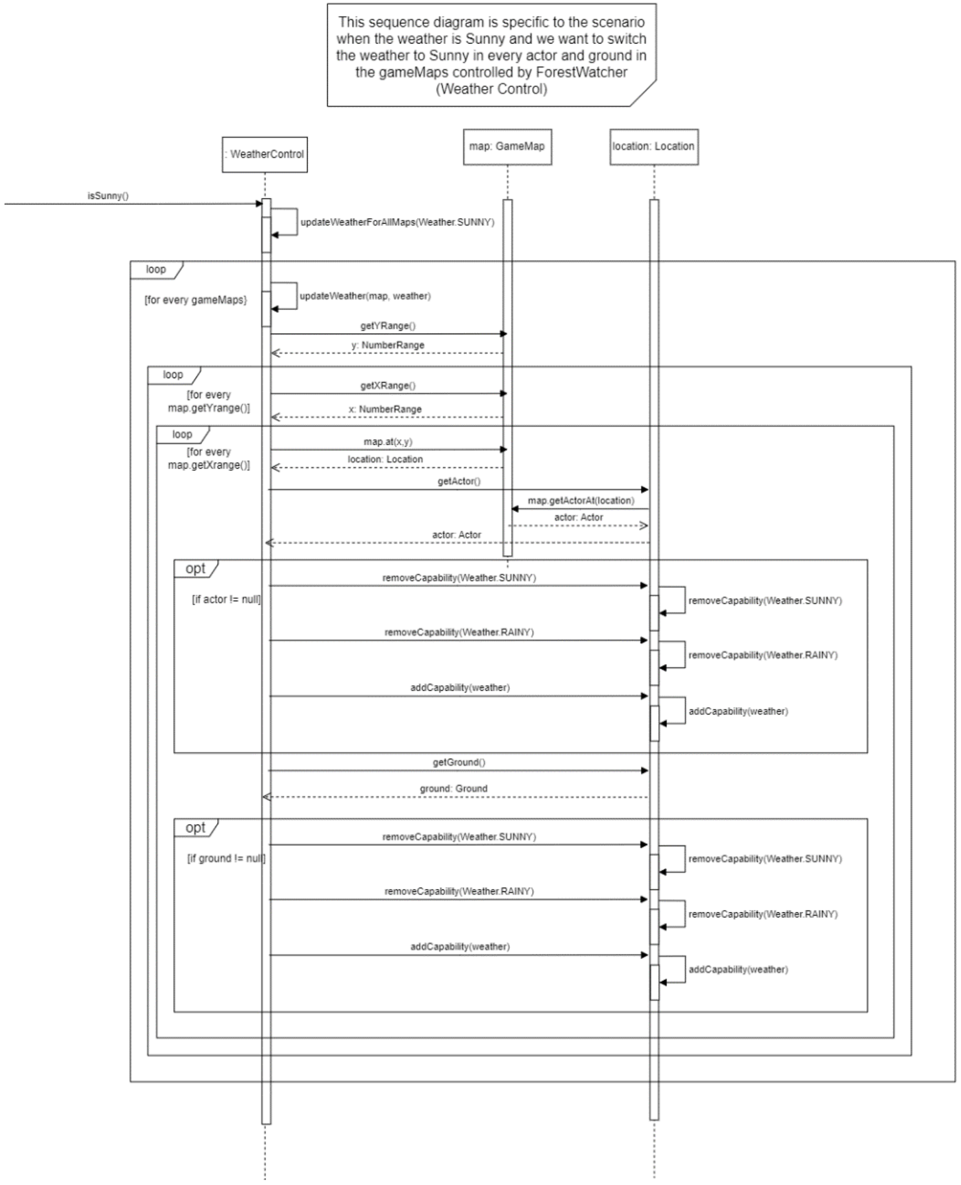


This sequence diagram is specific to the scenario when the player selects the StabAndStep Action which calls the SafeStepMovement after the AttackAction to move the player to a safe location

: SafeStepMovement
:Location
map:GameMap
exit:Exit
actions: ActionList
getMovementAction()
<<create>>
alt
if (!(target.isConscious()))
locationOf(actor).getExits()
exits : List<Exit>
<<creates>>
destination : Location
Loop
[For each exit]
getDestination()
destination: Location
opt
if (destination.canActorEnter(actor))
getMoveAction(actor, direction, hotkey)
moveActorAction : MoveActorAction(location, direction, hotKey)
add(moveActorAction)
add(moveActorAction)
<<create>>
locations : List <Location>
locationOf(actor).getExits()
exits : List<Exit>
Loop
[For each Exit]
getDestination()
destination : Location
add(destination)
locationOf(actor).getExits()
exits : List<Exit>
Loop
<<creates>>
destination : Location
[For each Exit]
getDestination()
destination : Location
opt
[destination.canActorEnter(actor) && !(locations.contains(destination)]
getDestination()
getMoveAction(actor, direction, hotkey)
moveActorAction : MoveActorAction(location, direction, hotKey)
add(moveActorAction)
add(action)
Alt
if (!actions.isEmpty())
get(int)
return MoveActorAction
return null

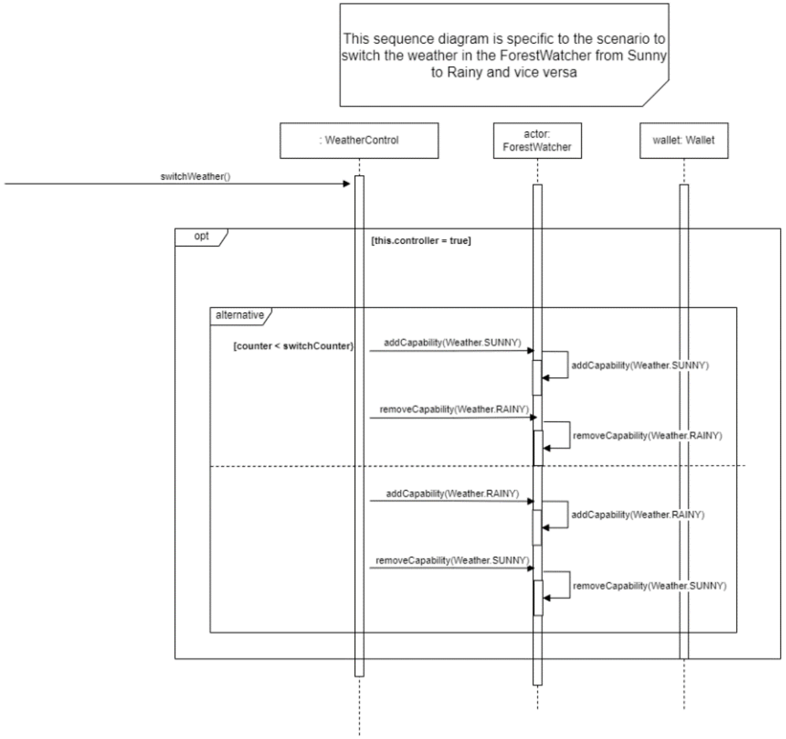
SEQUENCE DIAGRAM

Requirement 5 (Everyone)



This sequence diagram is specific to the scenario when the weather is Sunny and we want to switch the weather to Sunny in every actor and ground in the gameMaps controlled by ForestWatcher (Weather Control)

: WeatherControl
map: GameMap
location: Location
isSunny()
updateWeatherForAllMaps(Weather.SUNNY)
loop
[for every gameMaps]
updateWeather(map, weather)
getYRange()
y: NumberRange
loop
[for every map.getYrange()]
getXRange()
x: NumberRange
loop
[for every map.getXrange()]
map.at(x,y)
location: Location
getActor()
map.getActorAt(location)
actor: Actor
actor: Actor
opt
[if actor != null]
removeCapability(Weather.SUNNY)
removeCapability(Weather.SUNNY)
removeCapability(Weather.RAINY)
removeCapability(Weather.RAINY)
addCapability(weather)
addCapability(weather)
getGround()
ground: Ground
opt
[if ground != null]
removeCapability(Weather.SUNNY)
removeCapability(Weather.SUNNY)
removeCapability(Weather.RAINY)
removeCapability(Weather.RAINY)
addCapability(weather)
addCapability(weather)



This sequence diagram is specific to the scenario to switch the weather in the ForestWatcher from Sunny to Rainy and vice versa

: WeatherControl
actor: ForestWatcher
wallet: Wallet
switchWeather()
opt
[this.controller = true]
alternative
[counter < switchCounter]
addCapability(Weather.SUNNY)
addCapability(Weather.SUNNY)
removeCapability(Weather.RAINY)
removeCapability(Weather.RAINY)
addCapability(Weather.RAINY)
addCapability(Weather.RAINY)
removeCapability(Weather.SUNNY)
removeCapability(Weather.SUNNY)