FIT 2099 Assignment 1A

Lab 7 Team 1

Lab Tutor: Dr. Tan Choon Ling

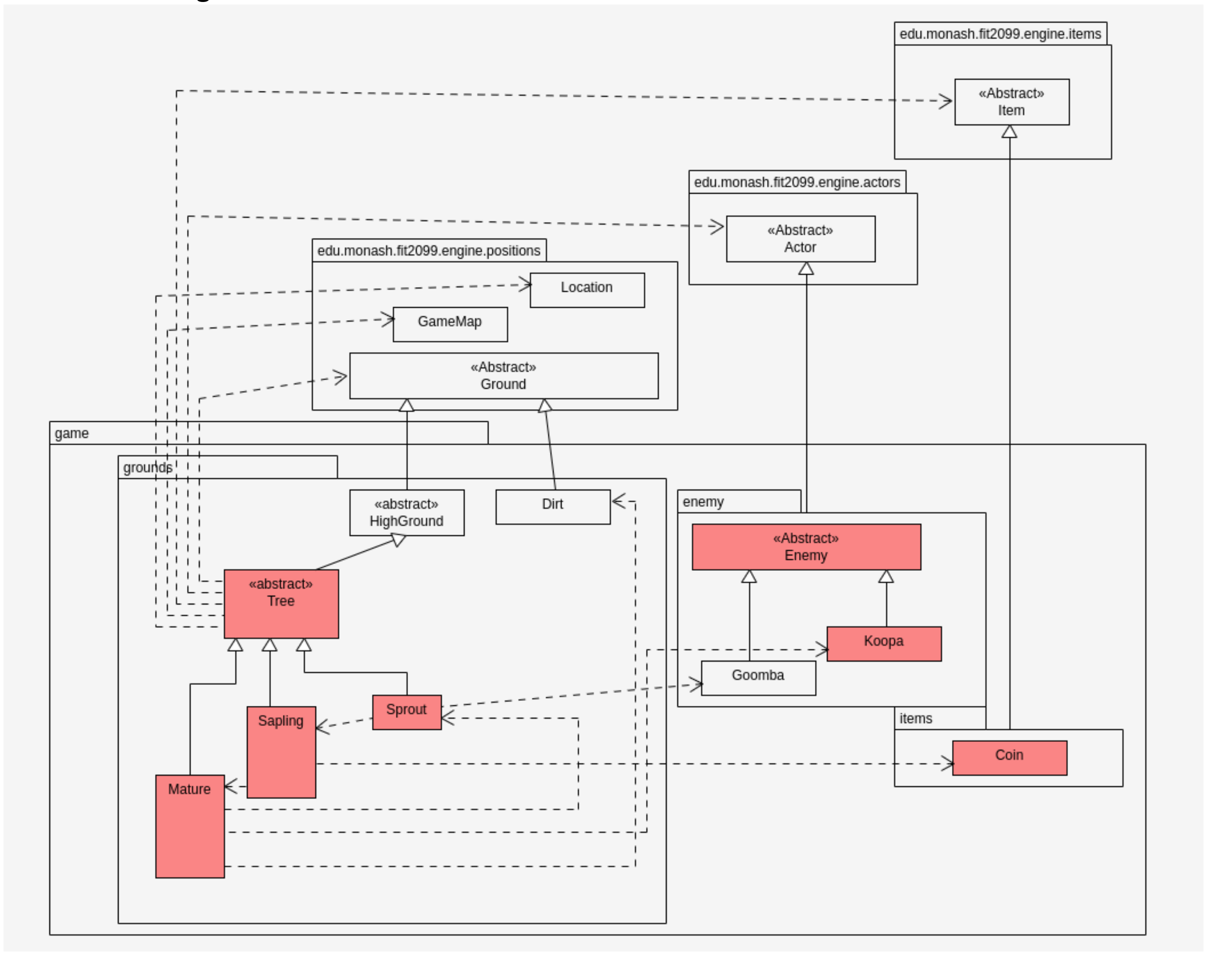
Group Members:

Khor Jia Shin 32356595

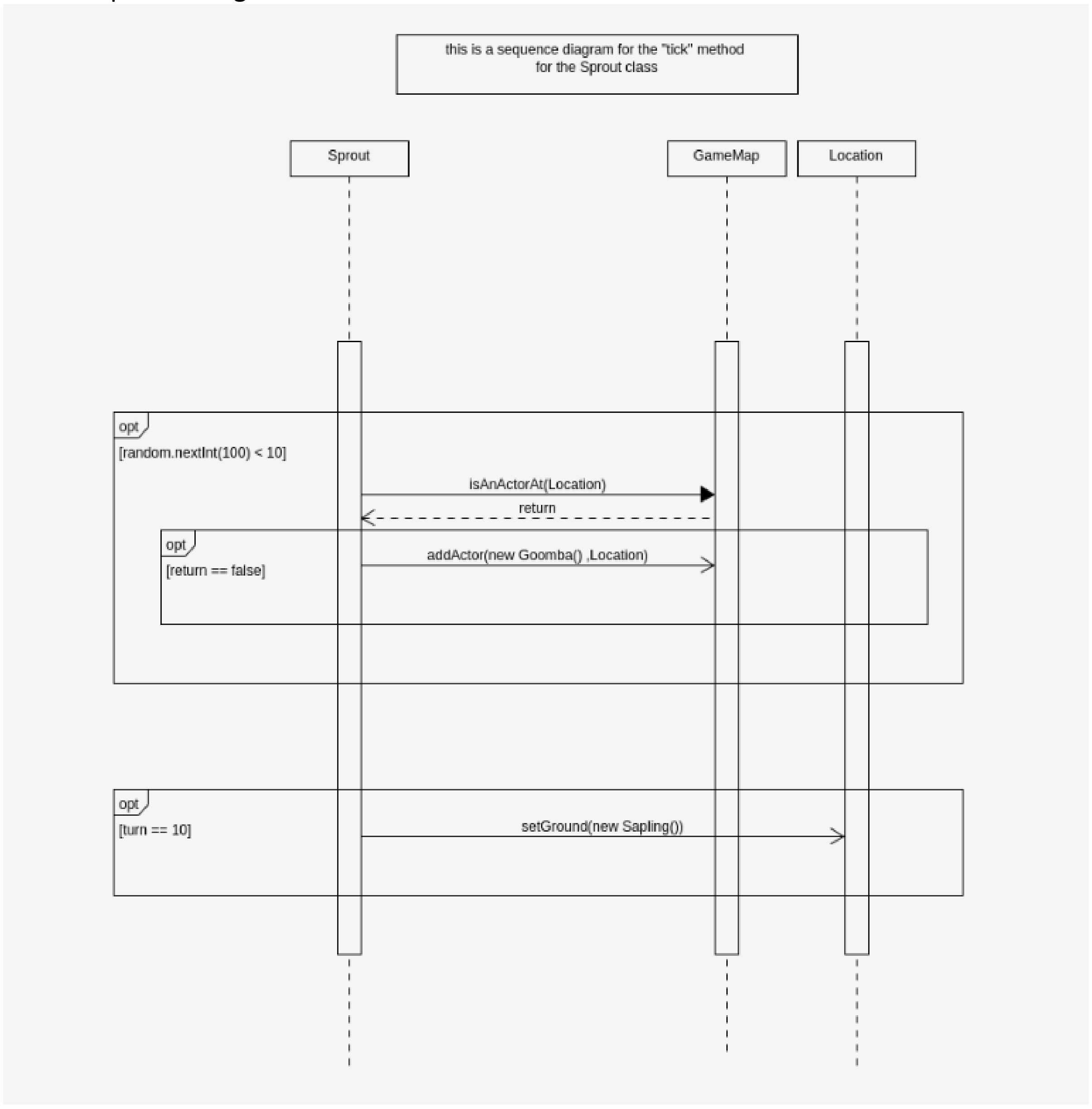
Lim Hong Yee 32455836

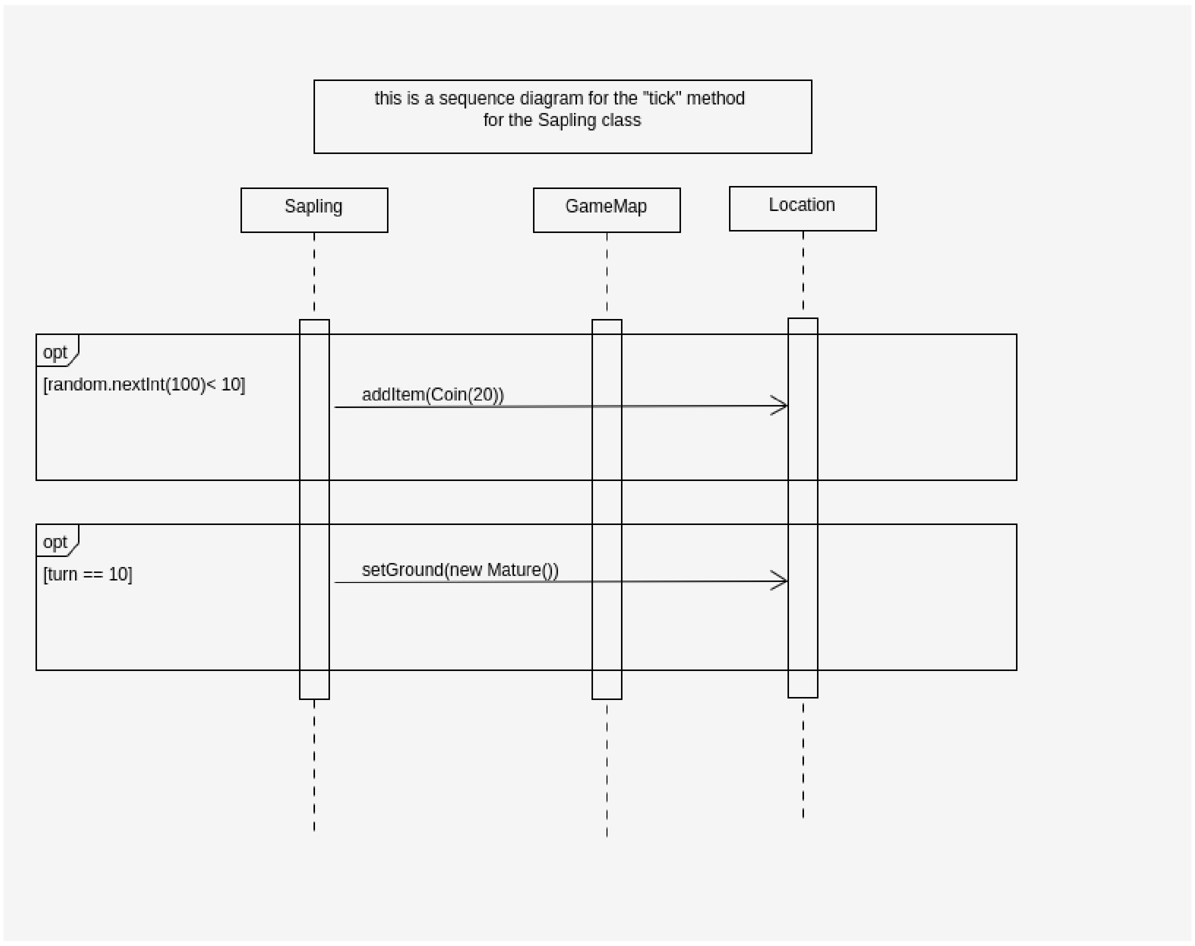
Fadi Alailan 31844936

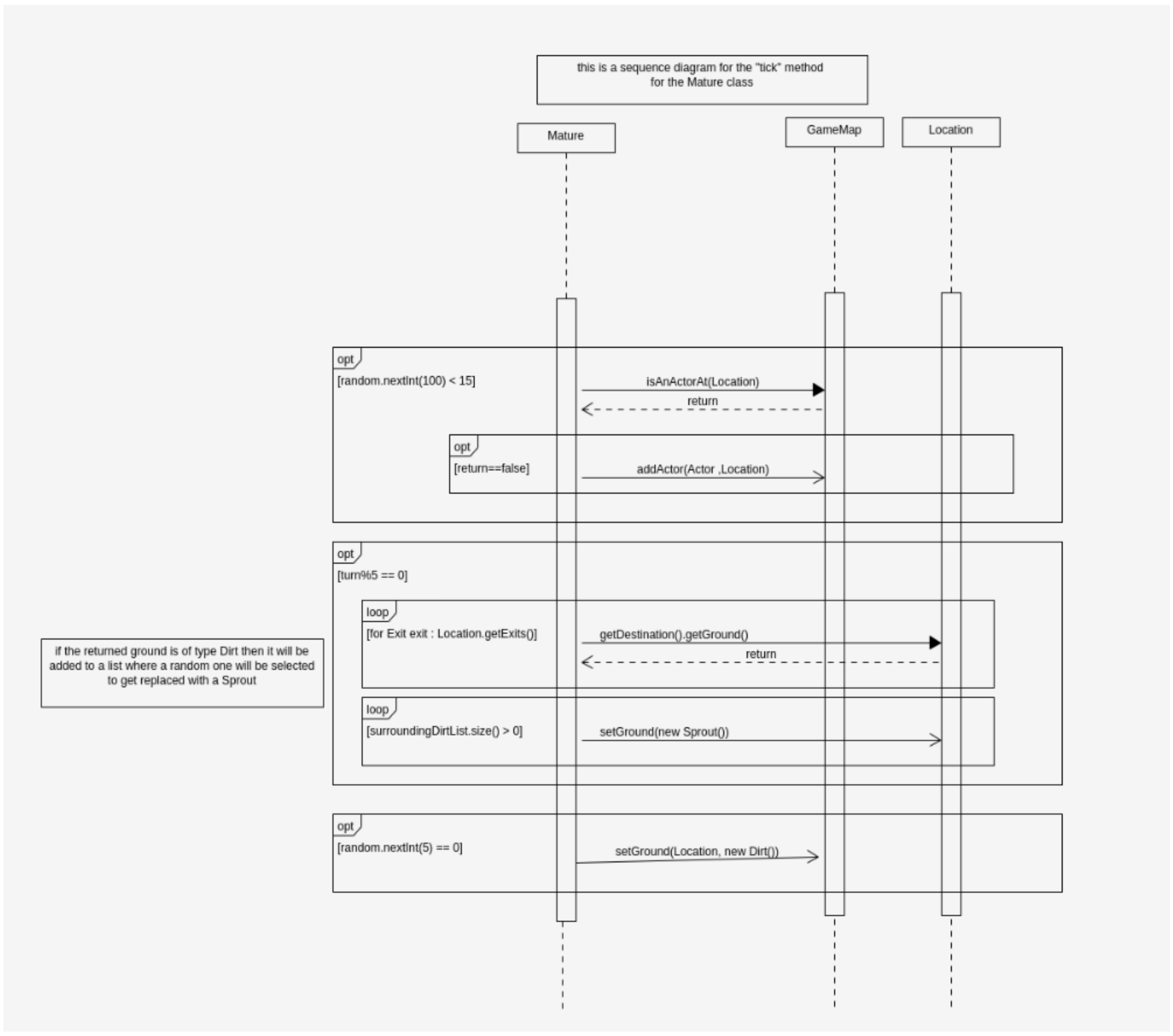
REQ1 Class Diagram



REQ1 Sequence Diagram


this is a sequence diagram for the "tick" method for the Sprout class
Sprout
GameMap
Location
opt
[random.nextInt(100) < 10]
isAnActorAt(Location)
return
opt
[return == false]
addActor(new Goomba() ,Location)
opt
[turn == 10]
setGround(new Sapling())


this is a sequence diagram for the "tick" method for the Sapling class
Sapling
GameMap
Location
opt
[random.nextInt(100)< 10]
addItem(Coin(20))
opt
[turn == 10]
setGround(new Mature())


this is a sequence diagram for the "tick" method for the Mature class
Mature
GameMap
Location
opt
[random.nextInt(100) < 15]
isAnActorAt(Location)
return
opt
[return==false]
addActor(Actor ,Location)
opt
[turn%5 == 0]
loop
[for Exit exit : Location.getExits()]
getDestination().getGround()
return
if the returned ground is of type Dirt then it will be added to a list where a random one will be selected to get replaced with a Sprout
loop
[surroundingDirtList.size() > 0]
setGround(new Sprout())
opt
[random.nextInt(5) == 0]
setGround(Location, new Dirt())

REQ1 Design Rationale

Enemy:

an abstract class that will be added in REQ3, added here forconsistency

HighGround:

an abstract class that will be added in REQ2, added here forconsistency

Tree:

Tree is an abstract class that is meant to be used to group the 3 main Tree types (Sapling, Sprout, Mature). The main reason for this class is to not violate the "SingleResponsibility Principle" and to make the code more organized. Trees are part of the terrain, so the Tree class inherits from Ground (HighGround in REQ2), it depends on Location because we plan to use the "tick" method and it depends on GameMap because of it will need to access the Location's GameMap. It depends on actor anditem because its children will need to spawn these objects.

Sprout:

It is a class that inherits from Tree because it is one of the 3 stages of a Tree's life (the other 2 being Sapling and Mature). It will spawn Goomba ,so it will depend on them, and it will depend on Sapling because it wil turn into one after 10 turns

Sapling:

it is a class that inherits from Tree because it is one of the 3stages of a Tree's life (the other 2 being Sprout and Mature). It will spawn Coins ,so it will depend on them, and it will depend on Mature because it will turn into one after 10 turns

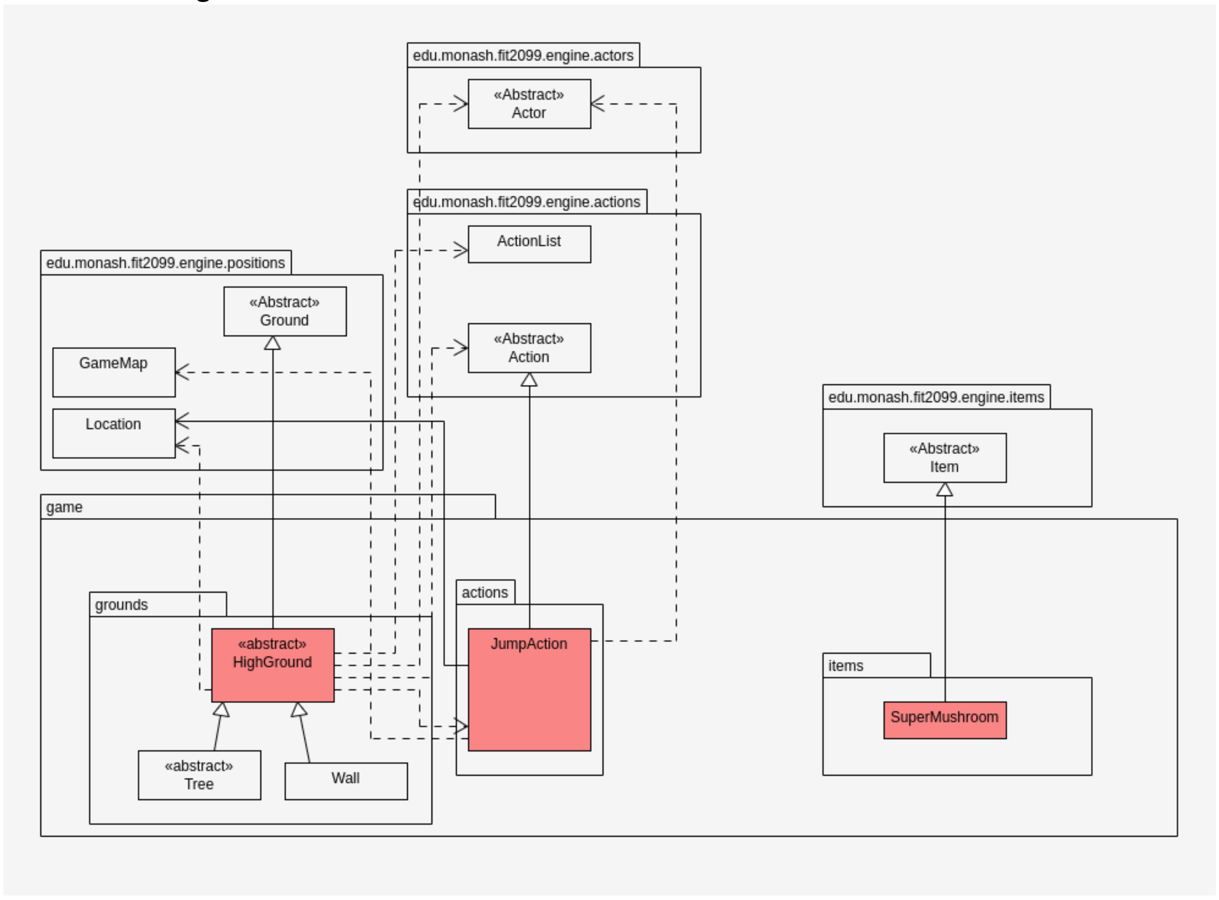
Mature:

It is a class that inherits from Tree because it is one of the 3 stages of a Tree's life (the other 2 being Sprout and Sapling). It depends on Koopa, because it will spawn them, and it will depend on Sprout because it will spawn them every 5 turns in a random surrounding fertile ground, Dirt is one of these fertile grounds, so it will depend on it and another reason why it depends on Dirt isbecause it has a 20% chance to turn into one each turn

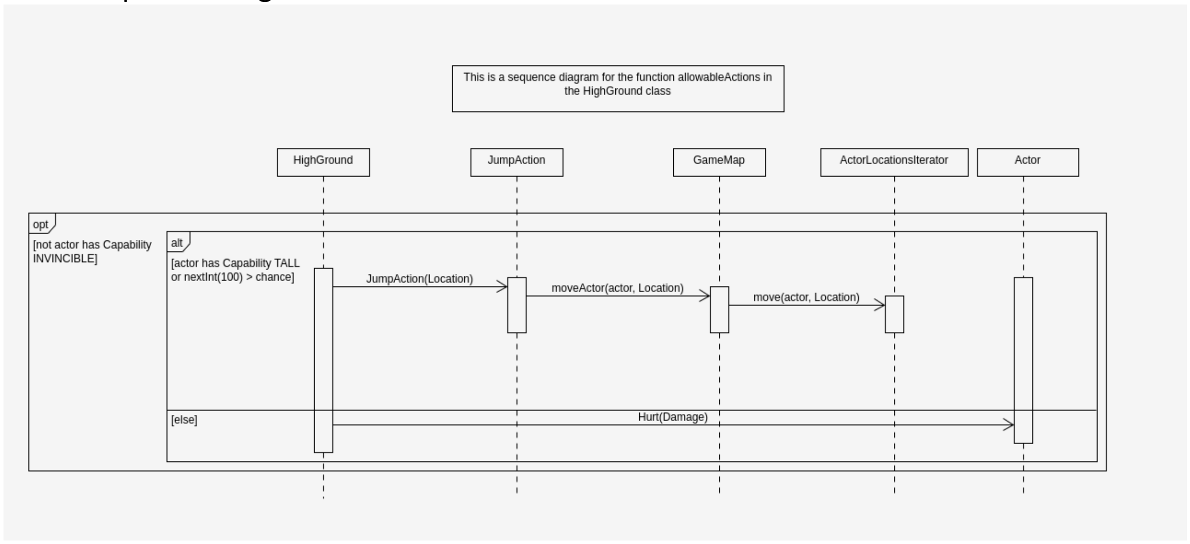
Coin:

an Item that if collected gives the player money, it is an Item that can be picked up so it seems natural that it inherits from Item

REQ2 Class Diagram



REQ2 Sequence Diagram


This is a sequence diagram for the function allowableActions in the HighGround class
HighGround
JumpAction
GameMap
ActorLocationsIterator
Actor
opt
[not actor has Capability INVINCIBLE]
alt
[actor has Capability TALL or nextInt(100) > chance]
JumpAction(Location)
moveActor(actor, Location)
move(actor, Location)
[else]
Hurt(Damage)

REQ2 Design Rationale

JumpAction:

This is an action that the player can use when next to a HighGround, it will allow the player to jump to said high ground (move to that ground) even if that ground type blocks actors, it is an action, so it inherits from Action. It will need to move the player, so it needs to work with the Location class, so it will depend on it. Because of the method "public String execute(Actor actor, GameMap map)" that it inherits from Action and we need to override, it will depend on Actor and GameMap

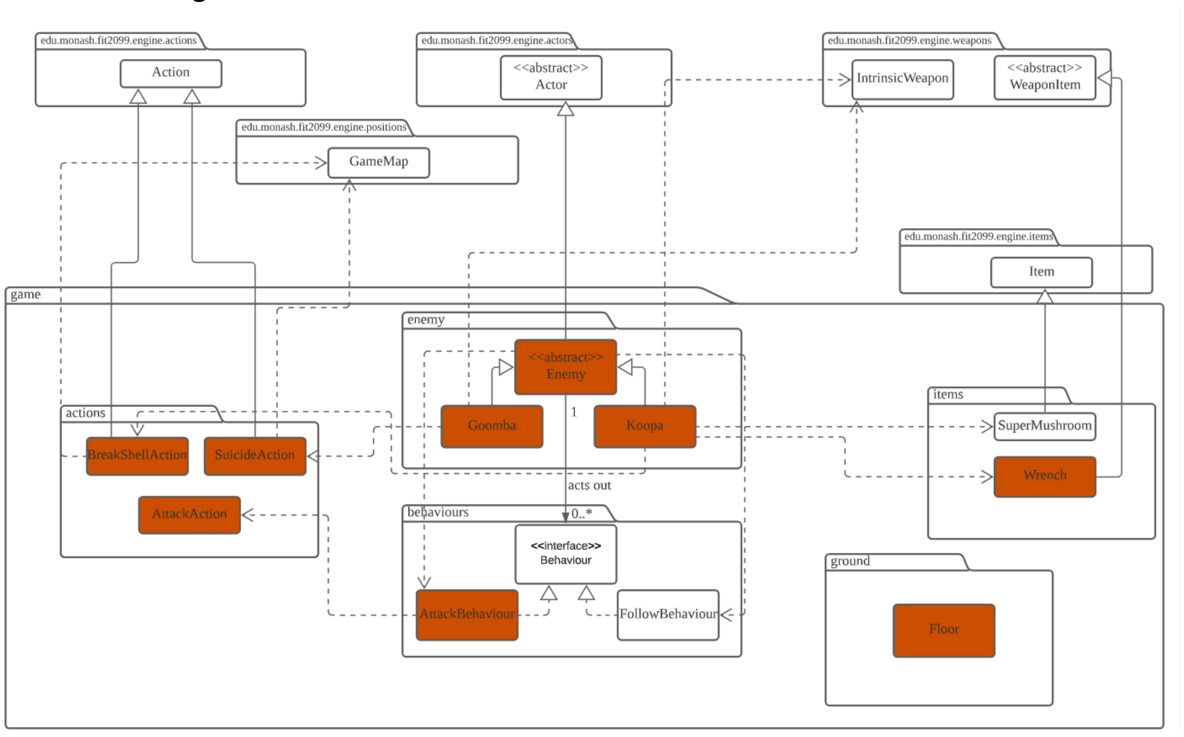
HighGround:

The HighGround Class is an abstract class that is meant to be used by Ground types that require the player to jump to them, it is meant meanly to make sure to follow the DRY principle. Because of the method "public ActionList allowableActions(Actoractor, Location location, String direction)" it will need to depend on Actor, Location, Action and ActionList. The method will return an ActionList containing the JumpAction, so it will need to depend on that as well

SuperMushroom:

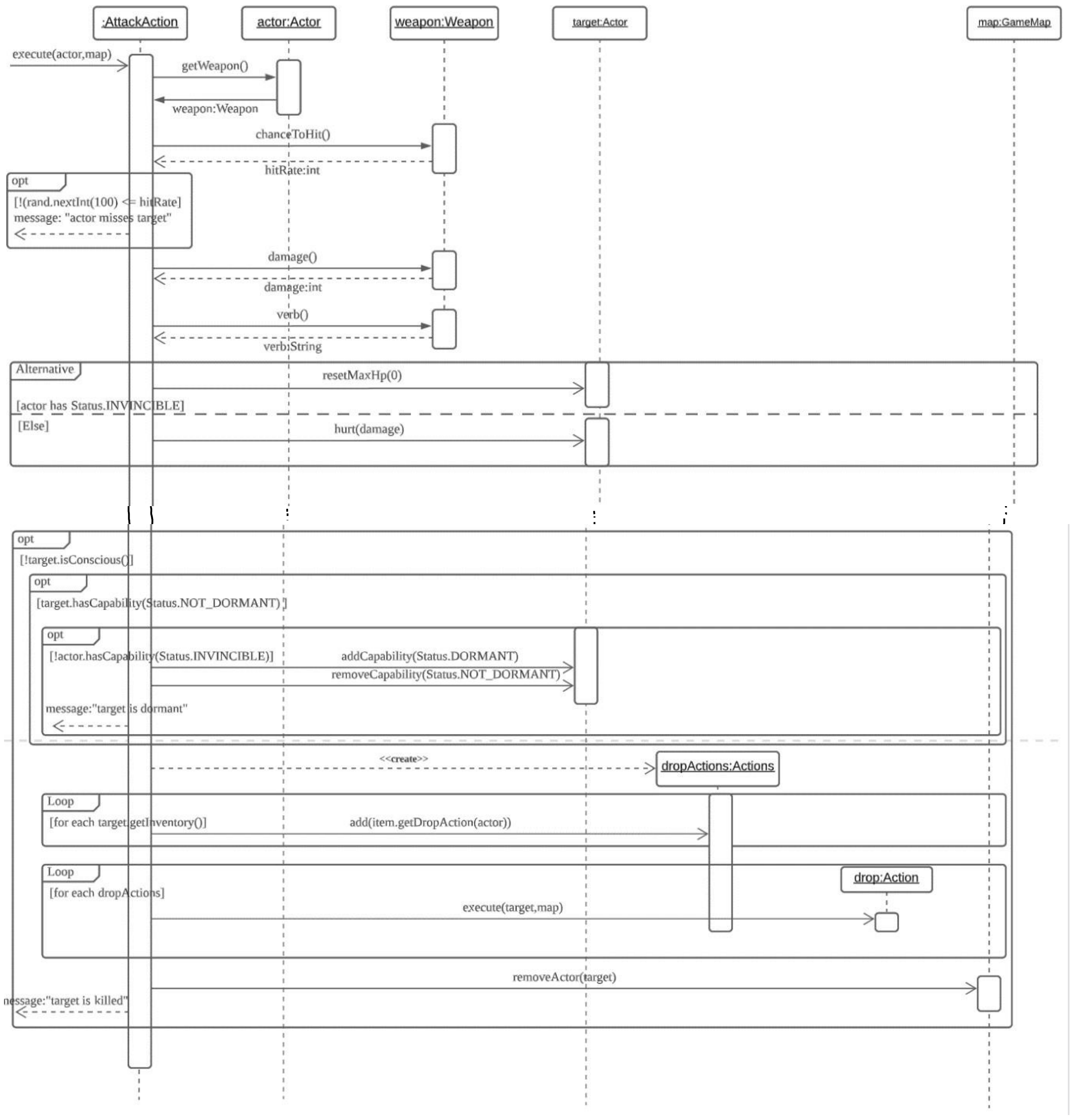
It is an item that if picked up, will allow the player to have a100% jump chance (it has other effects but they are not important for REQ2), we plan to make it give the player a status when consumed, so it won't depend or associate with any class except for an Enum class. it will only inherit from Item because it will be an item that the player can pick up and store in its inventory

REQ3 Class Diagram

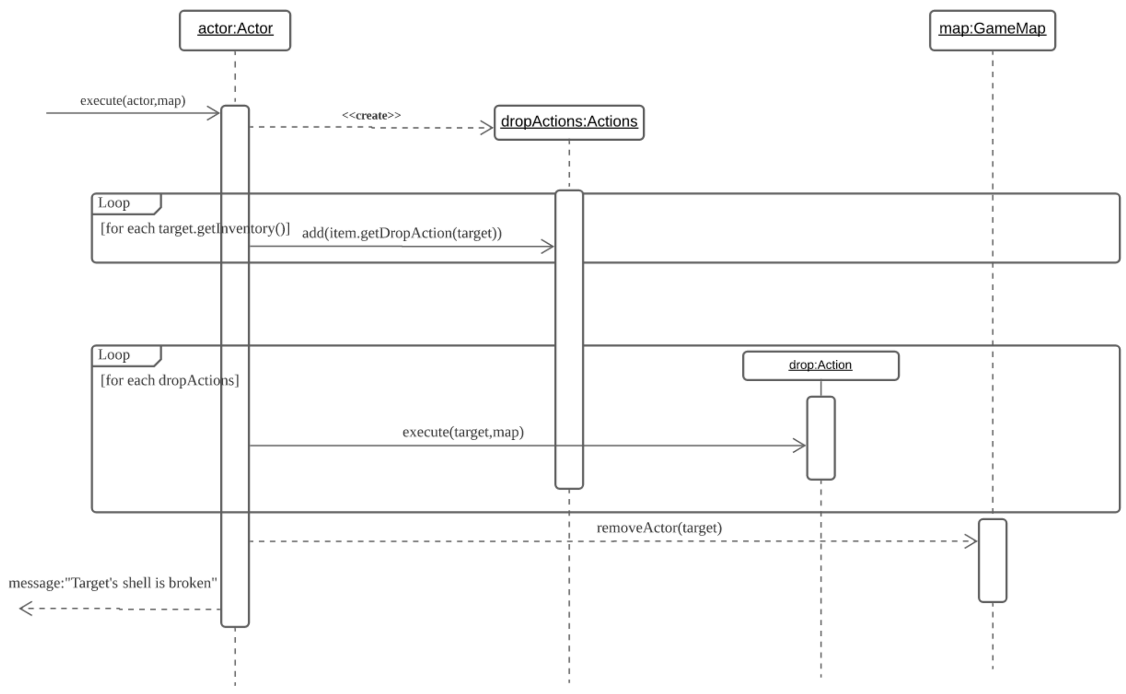


REQ3 Sequence Diagram

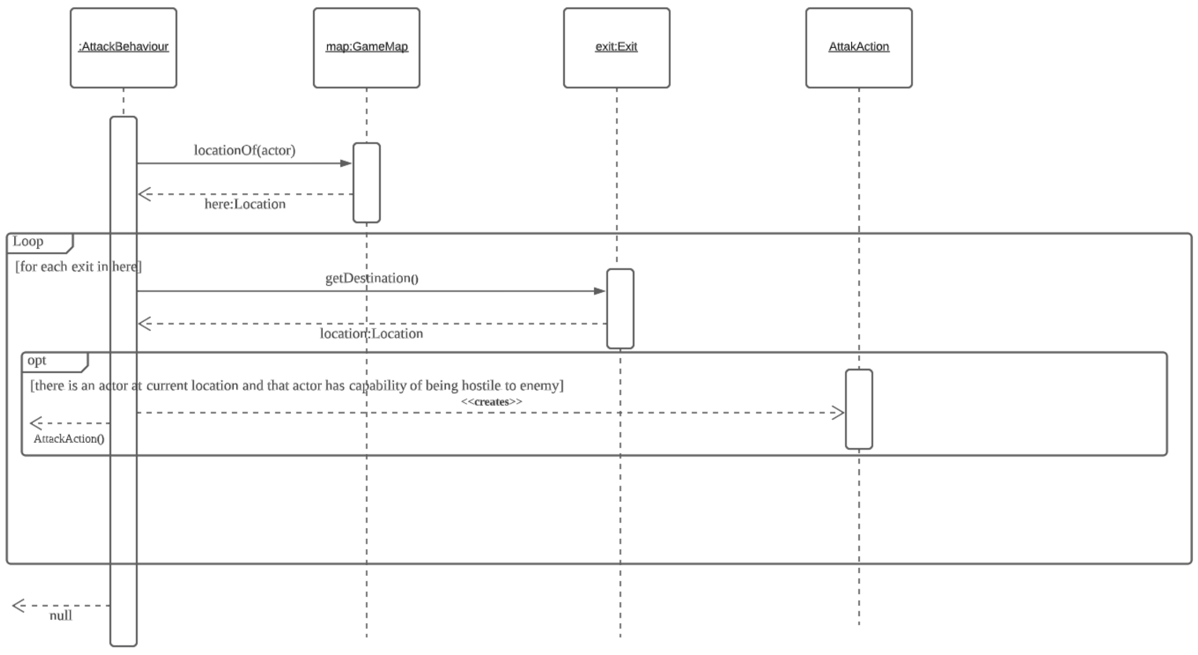
Sequence diagram of AttackAction.execute()


:AttackAction
actor:Actor
weapon:Weapon
target:Actor
map:GameMap
execute(actor,map)
getWeapon()
weapon:Weapon
chanceToHit()
hitRate:int
opt
[!(rand.nextInt(100) <= hitRate]
message: "actor misses target"
damage()
damage:int
verb()
verb:String
Alternative
resetMaxHp(0)
[actor has Status.INVINCIBLE]
[Else]
hurt(damage)
opt
[!target.isConscious()]
opt
[target.hasCapability(Status.NOT_DORMANT)]
opt
[!actor.hasCapability(Status.INVINCIBLE)]
addCapability(Status.DORMANT)
removeCapability(Status.NOT_DORMANT)
message:"target is dormant"
<<create>>
dropActions:Actions
Loop
[for each target.getInventory()]
add(item.getDropAction(actor))
Loop
[for each dropActions]
drop:Action
execute(target,map)
removeActor(target)
message:"target is killed"

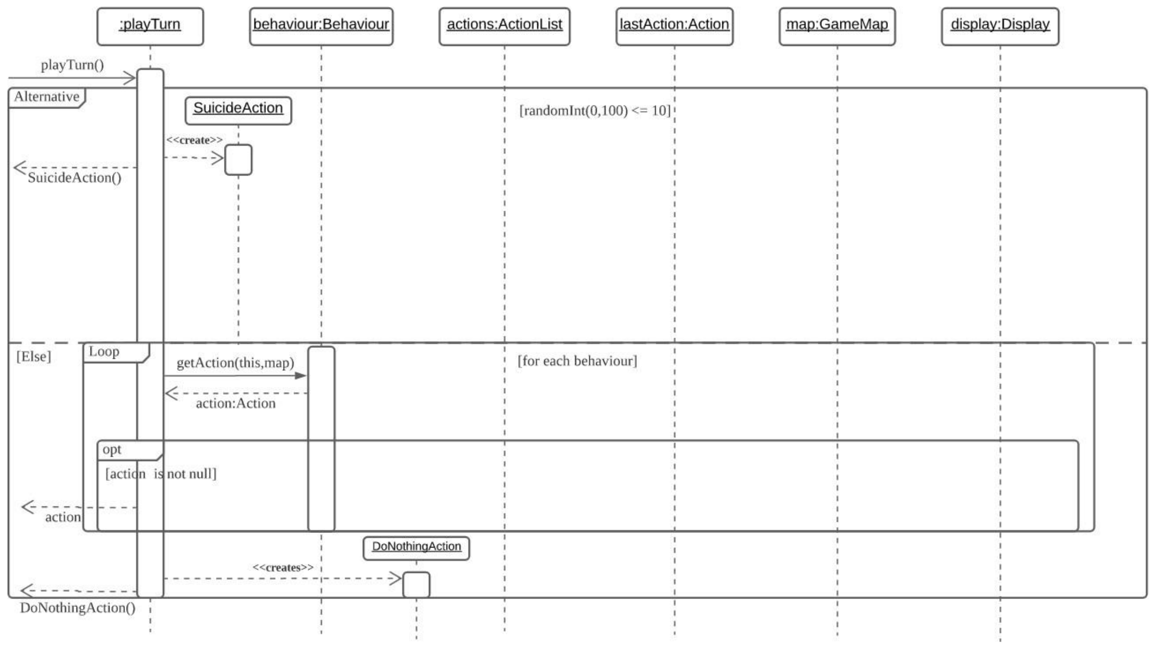
Sequence diagram of BreakShellAction.execute()



Sequence diagram of AttackBehaviour.getAction()



Sequence diagram of Goomba.playTurn()



REQ3 Design Rationale

NEWLY CREATED/MODIFIED CLASSES WITH NEW RELATIONSHIPS

Enemy

We made an abstract Enemy class that extends Actor class. Then, Goomba and Koopa will extend Enemy class. At first, we thought about just having Goomba and Koopa extend Actor class itself, since it seemed straight forward. However, after looking through the instructions, there are plenty of features that Goomba and Koopa share with one another, such as them being hostile to the player, or how both have similar behaviours which they can act on. So, that's why we decided to create an abstract parent class of both of them called Enemy, in order to not violate DRY(don't repeat yourself) principle. The biggest advantage by doing so is that in the future, we will assume many more actors will be hostile towards the player will be added, so we can make them extend enemy in order to prevent redundant codes. Responsible for handling all hostile enemies. Has association with Behaviour since every enemy will be able to act out behaviours.

Goomba

A class that extends Enemy class. Responsible for representing a Goomba enemy.

Koopa

A class that extends Enemy class. Responsible for representing a Koopa enemy.

AttackBehaviour

A class that implements the Behaviour interface. Responsible for knowing when it is appropriate to launch an attack to the player.

BreakShellAction

A class that extends Action class. Responsible for executing the action to break Koopa’s shell.

SuicideAction

A class that extends Action class. Responsible for executing the action to cause Goomba to suicide.

AttackAction

Modified to adhere to the feature of Koopa going into dormant state and not dying, and from REQ4’s feature that if player is invincible, can one-shot Koopa even without a wrench

Wrench

A class that extends Item class. Responsible for representing a Wrench weapon.

Floor

Modified to adhere to the features of enemies not being able to enter it

<u>Basic features of enemy</u>

"Once the enemy is engaged in a fight, it will follow the Player"

After looking through ED, Dr Riordan stated that as long as the enemy is in the vicinity(one of the 8 exits) of the player, it will start to follow the player. So, this can be done by adding a new FollowBehaviour to the enemy's allowableActions method, where if the otherActor has capability of being hostile to enemy, then it will start following them.

"The unconscious enemy must be removed from the map."

This is already implemented in the original source code given in AttackAction.

"All enemies cannot enter the Floor."

For this, we can override canActorEnter method in Floor class, to check if actor is hostile to player, false will be returned, true otherwise

<u>Features of Goomba</u>

"Goomba starting with 20 HP"

In the constructor itself, we can initialize Goomba with 20 hitpoints

"Goomba attacks with a kick that deals 10 damage with 50% hit rate."

This is easily achievable by overriding getIntrinsicWeapon method and changing the damage value and verb when creating new IntrinsicWeapon. Hence, the dependency from Goomba to IntrinsicWeapon

"In every turn, Goomba has a 10% chance to be removed from the map (suicide)."

This can be done in Goomba's playTurn method, where we could generate 100 numbers ranging from 0 to 100 exclusive, if the number generated is less than 10, a SuicideAction will be returned, else process Goomba's behaviour like normal. In SuicideAction's execute method, we can utilize GameMap's removeActor method. Originally, we thought of creating a SuicideBehaviour, but we thought it wouldn't make sense since behaviours are something that an actor depending on the priority of behaviours and the current condition, return an action. But for suiciding, it is completely random, hence that's why we scrapped the idea of having a SuicideBehaviour. Hence, the dependency from Goomba to SuicideAction.

<u>Features of Koopa</u>

"Koopa starts with 100 HP"

In the constructor itself we can initialize Koopa with 20 hitpoints

"Koopa attacks with a punch that deals 30 damage with a 50% hit rate."

This is easily achievable by overriding getIntrinsicWeapon method and changing the damage value and verb when creating new IntrinsicWeapon. Hence, the dependency from Koopa to IntrinsicWeapon

"When defeated, Koopa will not be removed from the map. Instead, it will go to a dormant state (D) and stay on the ground"

Koopa will be initialized with Status.NOT_DORMANT in the constructor. In AttackAction, when target is not conscious, we can check if the target has capability of not being dormant, and if the player does not have invincibility(from req4), Koopa target will be given the capability of being dormant, and the not dormant capability will be removed. All of Koopa's current behaviours will be removed as well since it can't do anything in dormant state. For the display character, we can override getDisplayChar method, and check if Koopa is in dormant state or not, if yes then get the capital letter of the current display character.

"Mario needs a Wrench (80% hit rate and 50 damage), the only weapon to destroy Koopa's shell."

In Koopa's allowableActions, if the player is holding a wrench(checked using getWeapon method), hence the dependency from Koopa to Wrench, and Koopa has the capability of being dormant, then the player can do a new action to it called BreakShellAction. Hence, the dependency from Koopa with BreakShellAction. It will get every item in Koopa's inventory and drop it, and then picked up by the player, and lastly utilize GameMap to remove the dormant Koopa from the map. Originally, we thought of just having more if statements to check if player is using a wrench and if it's dormant then kill it, but we think it violates the OOP principle of Single Responsibility Principle, as the AttackAction will have too many responsibilities at this point as it will bear the responsibility of attacking actors as well as breaking Koopa’s shell.

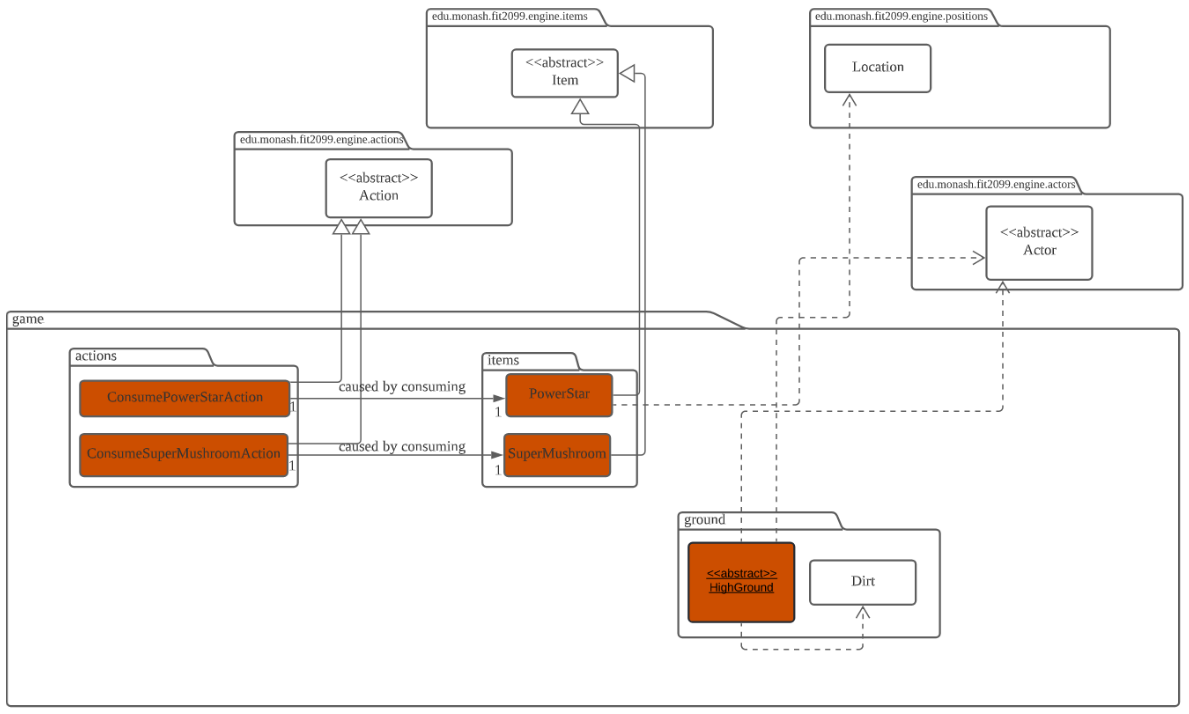
"Destroying its shell will drop a Super Mushroom."

In the constructor, a new SuperMushroom object will be made and added to Koopa's inventory. Hence, the dependency from Koopa to SuperMushroom

<u>AttackBehaviour's design</u>

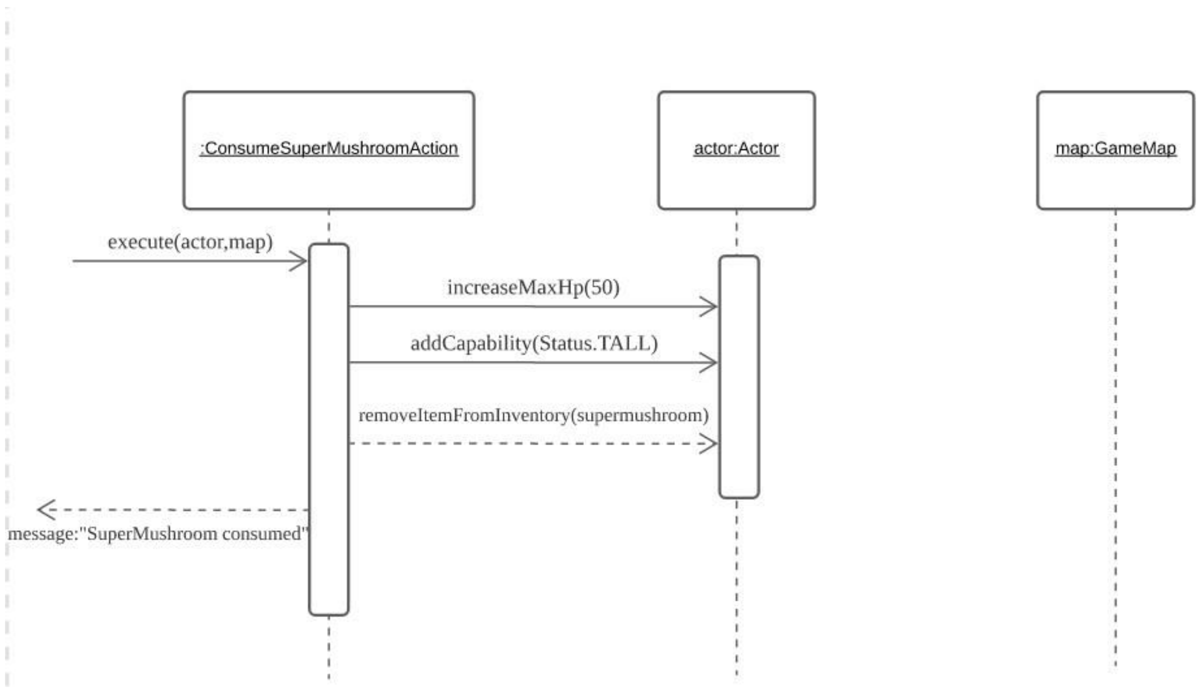
AttackBehaviour will be added to every enemy's behaviour list during construction, hence the dependency from Enemy to AttackBehaviour. Firstly, location of enemy will be obtained using GameMap's locationOf method. For that location, we will get all its exits using getExits method. For each exit, we will get its location using getDestination method. For each of those locations, we will check if there is an actor at a current location and whether that actor has capability of being hostile to enemies, if so, then create an AttackAction at that direction.

REQ4 Class Diagram

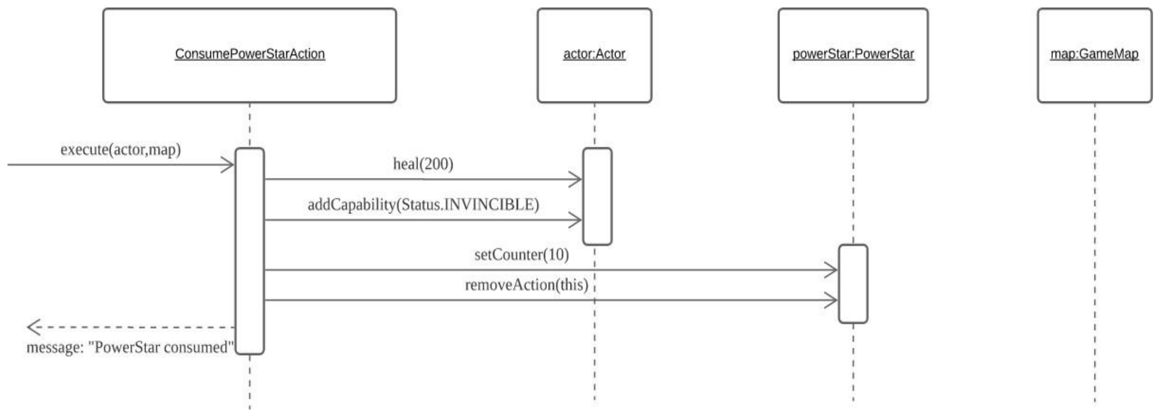


REQ4 Sequence Diagram

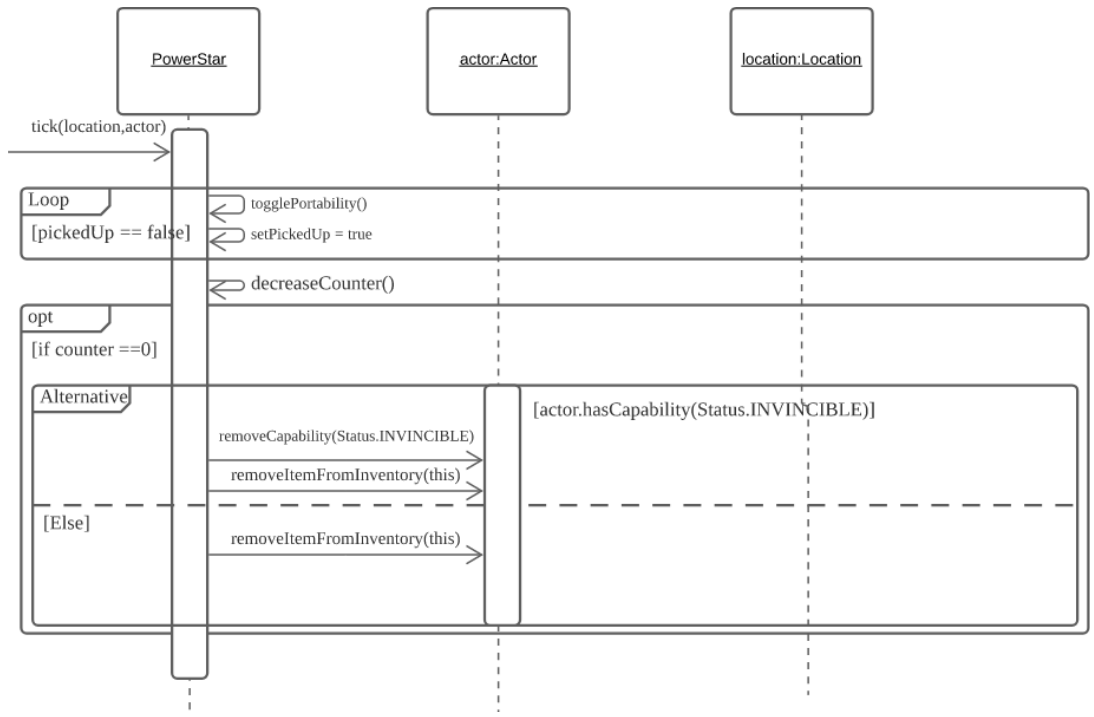
Sequence diagram of ConsumeSuperMushroomAction.execute()


:ConsumeSuperMushroomAction
actor:Actor
map:GameMap
execute(actor,map)
increaseMaxHp(50)
addCapability(Status.TALL)
removeItemFromInventory(supermushroom)
message:"SuperMushroom consumed"

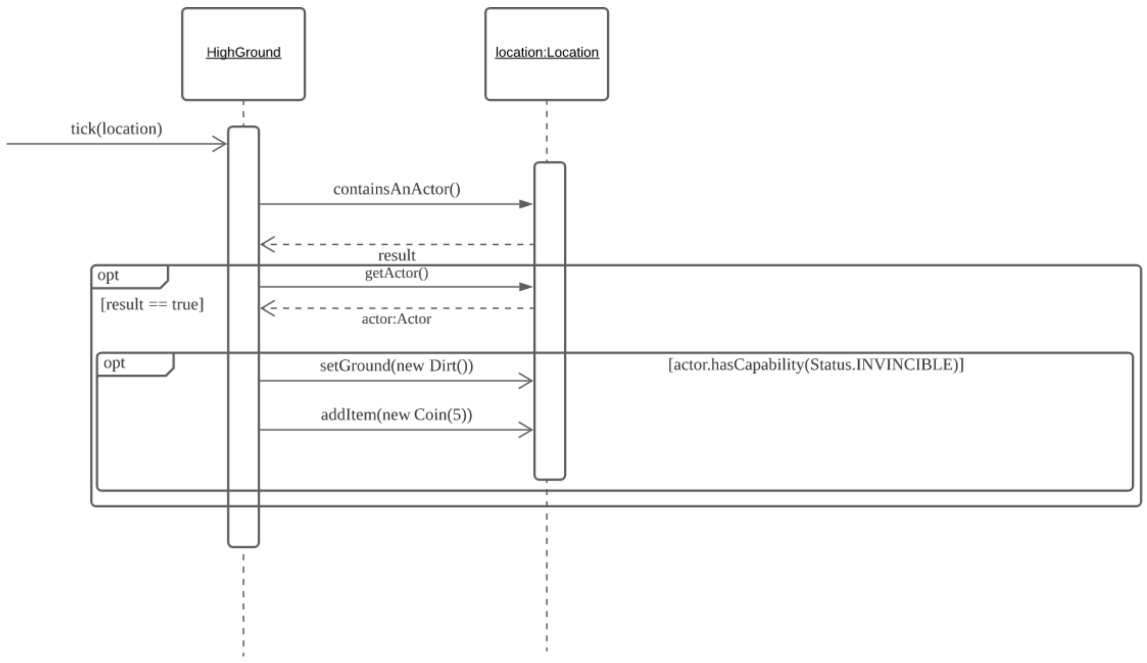
Sequence diagram of ConsumePowerStarAction.execute()



Sequence diagram of PowerStar.tick() when it is in player's inventory



Sequence diagram of HighGround.tick()



REQ4 Design Rationale

NEWLY CREATED/MODIFIED CLASSES WITH NEW RELATIONSHIPS

SuperMushroom

A class that extends Item. Responsible for allowing us to consume it to get its effects. We plan to create a new action for consuming the super mushroom, called ConsumeSuperMushroomAction.

ConsumeSuperMushroomAction

A class that extends Action. Responsible for giving us the effects of consuming a super mushroom.

Has an attribute of the mushroom consumed by the player. Hence, there is an association relationship between SuperMushroom and ConsumeSuperMushroomAction

PowerStar

A class that extends Item. Responsible for allowing us to consume it to get its effects. We plan to create a new action for consuming the super mushroom, called ConsumePowerStarAction.

ConsumePowerStarAction

A class that extends Action. Responsible for giving us the effects of consuming a power star.

Has an attribute of the power star consumed by the player. Hence, there is an association relationship between PowerStar and ConsumePowerStarAction.

HighGround

This is changed because of some changes that is needed to be done to its allowableActions method. And, it can be destroyed now after having the effects of consuming a power star.

Effects of consuming a super mushroom

"Incease max HP by 50"

We can utilize actor's increaseMaxHp method to heal players by 50 hitpoints in ConsumeSuperMushroomAction’s execute

"The display character evolves to the uppercase letter"

This is already done for us in original source code for Player class, in getDisplayChar method

"It can jump freely with a 100% success rate and no fall damage"

Players will be given the status of TALL. Jumping freely with a 100% success rate can be done in HighGround's allowableActions method, where we can implement a check to see if actor doing the jump have TALL status or not, if it is then JumpAction can be executed without even checking for percentages.

After all these is done, the SuperMushroom that was consumed will be removed from the player's inventory, in ConsumeSuperMushroomAction's execute using removeItemFromInventory method.

"The effect will last until it receives any damage"

We can override hurt method in Player class, where if player do have TALL status, then remove it.

Properties of Power Star

"It will fade away and be removed from the game within 10 turns (regardless it is on the ground or in the actor's inventory)"

For PowerStar, we plan to have a new attribute to it called counter with a value of 10. We can override both tick methods from Item class, the first tick method is the one with only location as parameter. This means that it is on the ground, so in every tick the counter will decrease by 1. If the counter reaches 0, we can utilize location's removeItem method to clear the PowerStar from the map. The second tick method is the one with location and actor as parameter, this means that it is in player's inventory. Like the first tick, counter will be decreased by 1. If counter reaches 0, we check if actor has capability of invincible, if it does, then remove it and remove the item from actor's inventory. If not, just remove the item from actor's inventory.

<u>Effects of consuming a power star</u>

"Anyone that consumes a Power Star will be healed by 200 hit points"

We can utilize actor's heal method to heal player by 200 hit points

" Will become invincible"

We can use addCapabilities to give the player Status.INVINCIBLE

“The actor does not need to jump to higher level ground (can walk normally)”

We can override canActorEnter method in HighGround, checking that if actor has capability of invincible, it will just return true, while normally it returns false.

“If the actor steps on high ground, it will automatically destroy (convert) ground to Dirt. For every destroyed ground, it drops a Coin ($5).”

In HighGround’s tick method, we will first check if the location contains an actor using Location’s containsAnActor method. If so, we will get the actor in that current location, using getActor method from Location, and check whether the actor has invincible capability or not. Therefore there’s a dependency between HighGround and Actor. If so, we can utilize Location’s setGround method to set current location to dirt and add a coin of value $5 using addItem method. Therefore there’s a dependency between HighGround and Location, as well as Dirt.

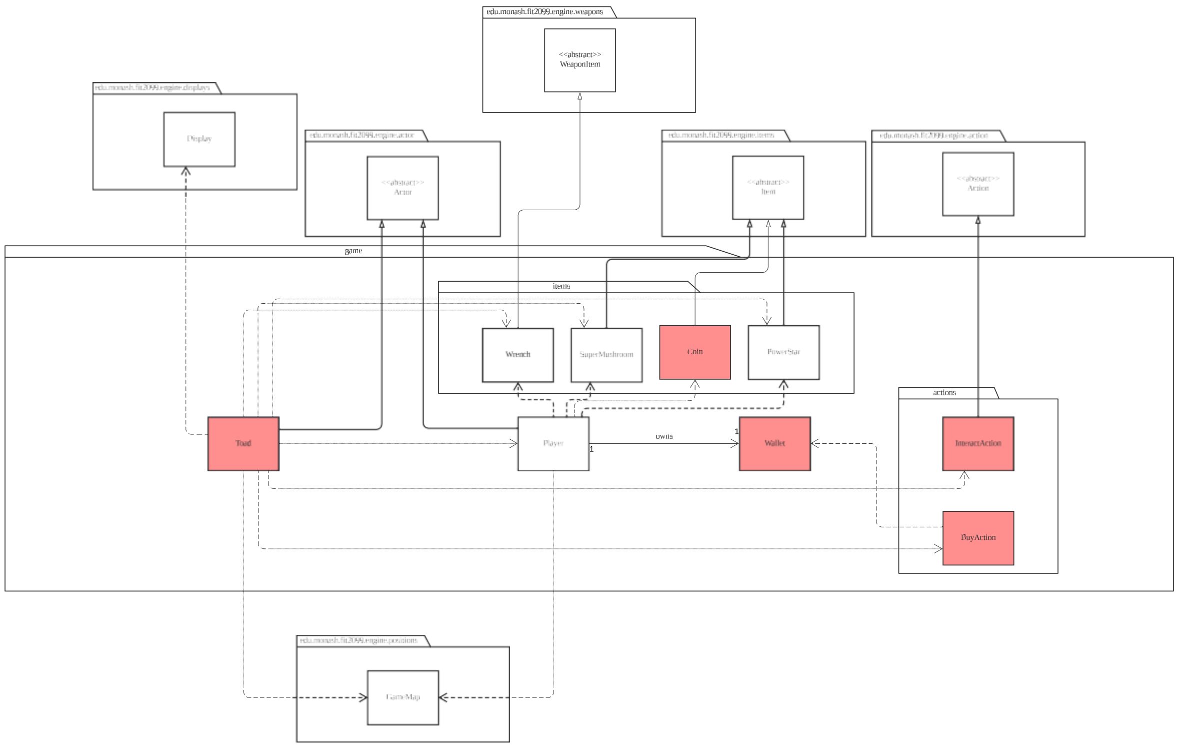
“All enemy attacks become useless (0 damage).”

Override player’s hurt method, if player has invincibility status, hitPoints -= 0

“When active, a successful attack will instantly kill enemies.”

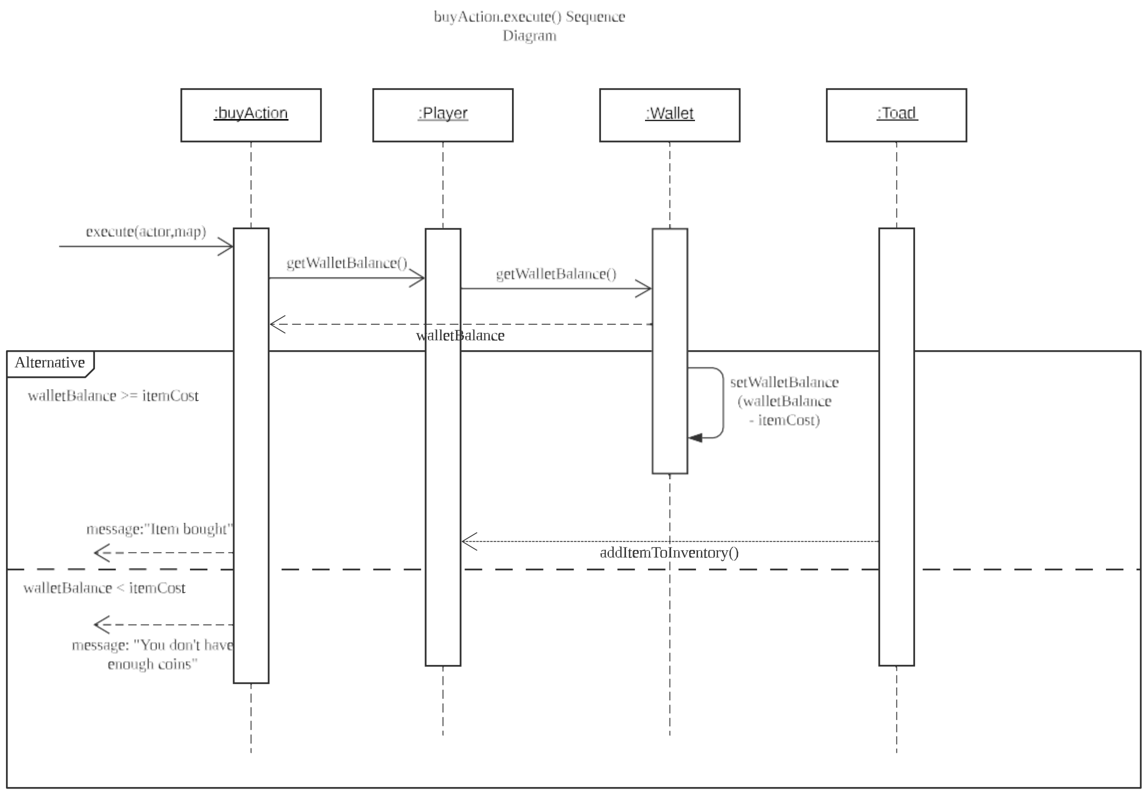
Referenced in REQ3’s AttackAction sequence diagram, we can check if actor has capability of invincibility. If so, reset the target’s max hp to 0 using resetMaxHp method.

REQ5&6 Class Diagram



edu.monash.fit2099.engine.weapons
<<abstract>> WeaponItem
edu.monash.fit2099.engine.displays
Display
edu.monash.fit2099.engine.actor
<<abstract>> Actor
edu.monash.fit2099.engine.items
<<abstract>> Item
edu.monash.fit2099.engine.action
<<abstract>> Action
game
items
Wrench
SuperMushroom
Coin
PowerStar
actions
InteractAction
BuyAction
Toad
Player
owns
1
1
Wallet
edu.monash.fit2099.engine.positions
GameMap

REQ5 Sequence Diagram



REQ5&6 Design Rationale

"The coin($) is the physical currency that an actor/buyer collects.A coin has an integer value (e.g., $5, $10, $20, or even $9001)."

Coin:

- A class that extends Item.
- Has a constructor that will initialise the value of the coin
- Whenever a player picks up an item, it will be automaticallybe in the player's inventory, but for coin, we do not want it to be in the player's inventory. We want it to add to player's wallet when it is picked up. Our implementation is, we are going to override item's tick method and remove the cointhat is picked up from the inventory and add it into player's wallet.

"Coins will spawn randomly from the Sapling (t) or from destroyed high grounds."

- Implemented in req2 and req5.

"Collecting these coins will increase the buyer's wallet (credit/money)."

Wallet:

- A new class to keep track of the amount of money that the player has.
- When a player picks up a coin, it will automatically be added into the player's wallet.

"Using this money, buyers (i.e., a player) can buy the following items from Toad:,Wrench: $200,Super Mushroom: $400,Power Star: $600"

"Toad (O) is a friendly actor, and he stands in the middle of the map (surrounded by brick Walls)."

"You will have an action to speak with Toad. Toad will speak onesentence at a time randomly:"

Toad:

- This is a class for the Toad actor that extends actor class.
- Toad has a dependency with InteractAction and BuyAction because Toad is

the object that gives the other actor(Player) an action to buy and interact. Alternatively, we can create BuyAction inside Wrench, Power Star and Super Mushroom.

- However, doing so will require the us to implement multiplemethods that are the

same inside different classes. This would violate the Don't Repeat Yourself(DRY) principle in object-oriented programming. Moreover, our design will also align with Reduce Dependency Principle because we break the coupling between Wrench, BuyAction and Player.

- Contains an arraylist for the items he sells.

InteractAction:

- Class that extends Action class, is in charge of having a conversation with a

player

- In this game, the ideal way to interact with the object is by attaching appropriate

action to its corresponding object. Therefore, this action will have a dependency with Toad insteadof Player.

- An action that will be called when player is at toad's surrounding.

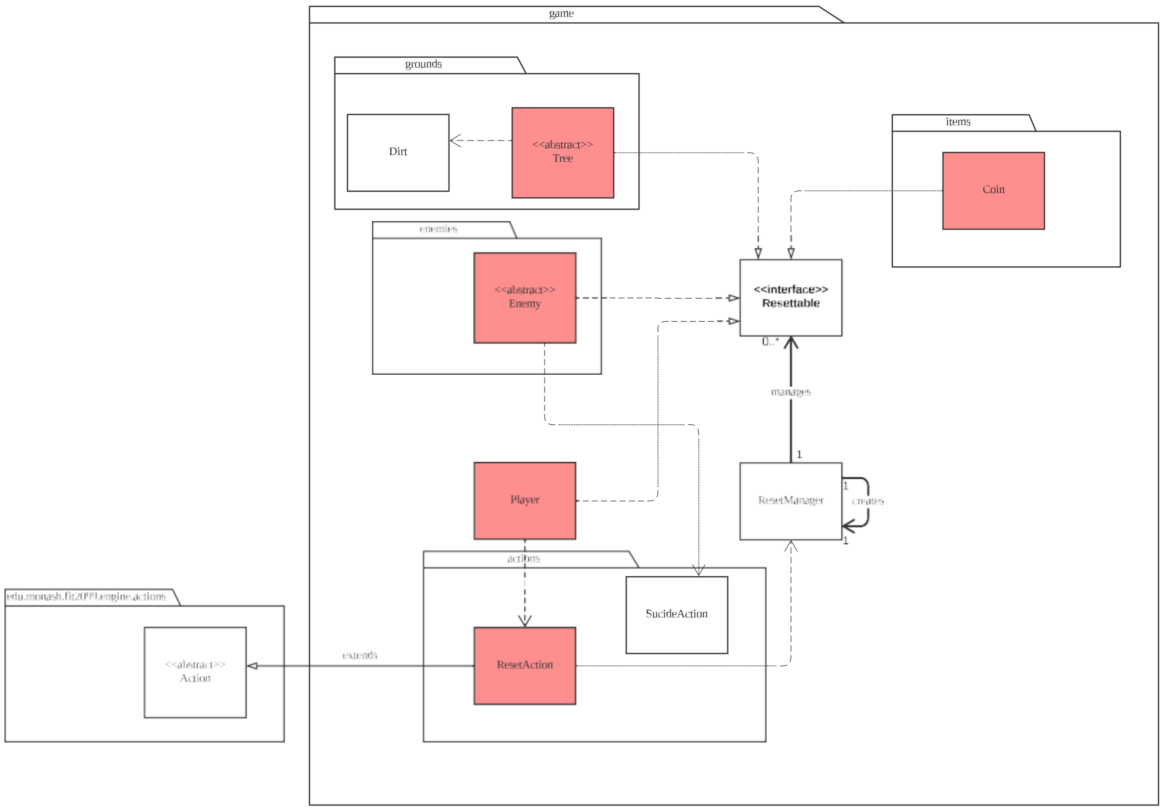
- Toad will check player's inventory if it contains wrench. If it contains wrench, it wont say first phrase

- If player contains capability of invincible, by using the hasCapability method to check, it wont say second phrase. If it contains both, it will only say third and fourth phrase.

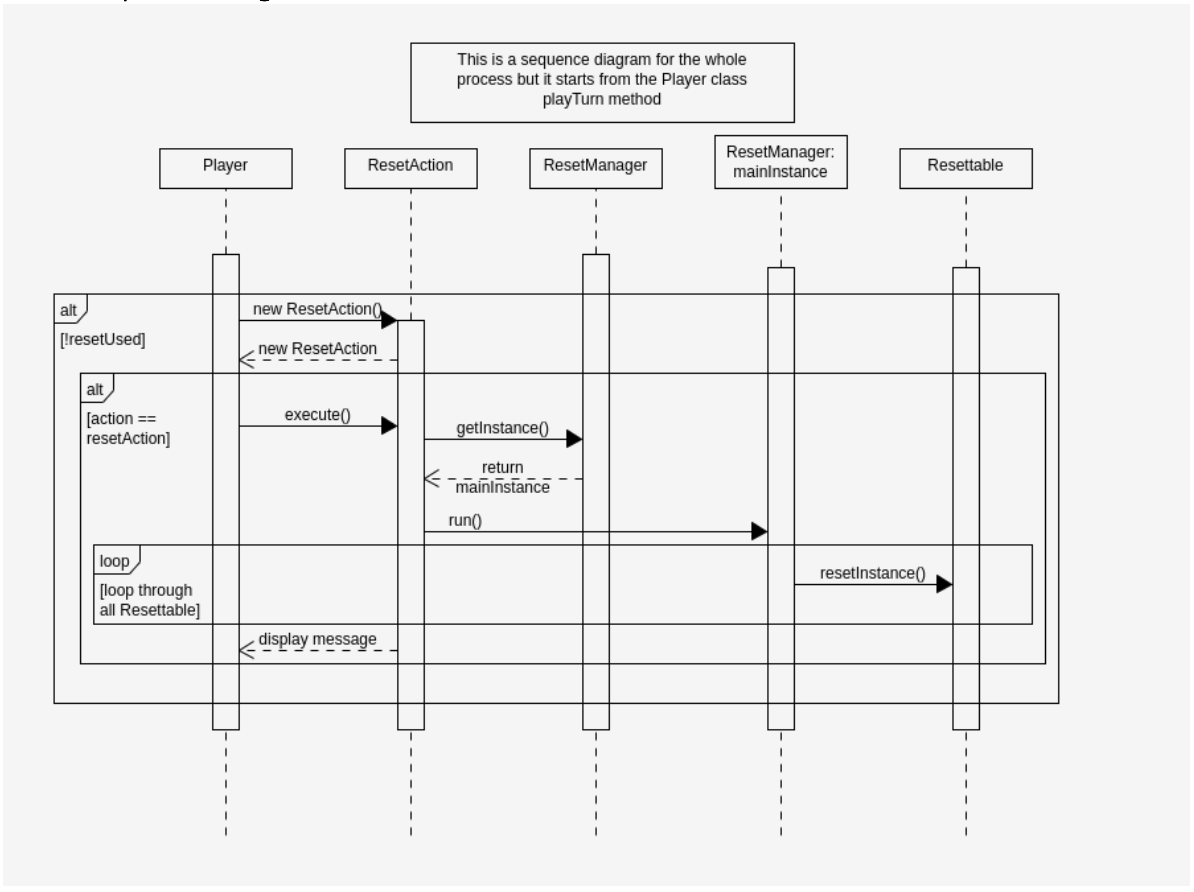
BuyAction:

- A class that extends Action class, in charge of selling itemsto the player.
- In this game, the ideal way to interact with the object is byattaching appropriate action to its corresponding object. Therefore, this action will have a dependency with Toad insteadof Player.
- An action that will be called when player is at toad's surrounding.
- When a player wants to buy an item, toad will check the player's wallet if it has sufficient balance to buy the item. If the player has sufficient balance, it will deduct the balance and add item to the inventory. If it doesn't have sufficient balance, it will tell the player that it does not have enough money.

REQ7 Class Diagram



REQ7 Sequence Diagram



REQ7 Design Rationale

NEWLY CREATED/MODIFIED CLASSES WITH NEW RELATIONSHIPS

ResetAction

A class that extends Action class. Responsible for doing a soft reset on the current game.

Tree

Modified to implement Resettable interface. In constructor, calls upon RegisterInstance method to register the current resettable instance of the Singleton manager.

Enemy

Modified to implement Resettable interface. In constructor, calls upon RegisterInstance method to register the current resettable instance of the Singleton manager.

Coin

Modified to implement Resettable interface. In constructor, calls upon RegisterInstance method to register the current resettable instance of the Singleton manager.

Player

Modified to implement Resettable interface. In constructor, calls upon RegisterInstance method to register the current resettable instance of the Singleton manager.

Process of resetting the game

We will have an attribute in Player class called resetUsed and will be set to false. In player's playturn, if resetUsed is false, we can call upon ResetAction to soft reset the game.

ResetAction's execute

We will first get the ResetManager's instance by using the getInstance method. Then, we will implement the run function, which will loop through all resettable objects and call resetInstance. Then, the attribute resetUsed will be set to true, so this action can no longer be done anymore in this current world.

<u>Tree’s resetInstance()</u>

Tree will have an attribute called resetUsed that will be defaulted to false upon construction. In resetInstance method, if it is called, it will set resetUsed to true. Then, in tick method for Tree, if resetUsed is true, it will generate 2 random numbers 0 and 1. If number chosen is 0, then we can utilize location’s setground method and set it to dirt.

<u>Enemy’s resetInstance()</u>

Enemy will have an attribute called resetUsed that will be defaulted to false upon construction. In resetInstance method, if it is called, it will set resetUsed to true. In Enemy’s playTurn, if resetUsed is true, then it will return a SuicideAction, thus removing them from the map.

<u>Player’s resetInstance()</u>

We can loop through player’s capabilities list and remove whatever status the player has, except for being hostile to enemy. We can also use heal the player with using heal method with parameter of getMaxHp, so the player will always be full health after soft resetting the game.

<u>Coin’s resetInstance()</u>

Coin will have an attribute called resetUsed that will be defaulted to false upon construction. In resetInstance method, if it is called, it will set resetUsed to true. Then, in tick method for Coin, if resetUsed is true, it will utilize location’s removeItem method to remove the coin from the location.

WBA

Task: REQ1 implement Tree
Teammate responsible: Fadi Alailan

Task: REQ2 implement Jumping
Teammate responsible: Fadi Alailan

Task: REQ3 Updating Goomba and Koopa properties
Teammate responsible: Khor Jia Shin

Task: REQ4 Adding Super Mushroom and Power Star class
Teammate responsible: Khor Jia Shin

Task: REQ5 and 6 Added Toad, Wallet and a few new actions
Teammate responsible: Lim Hong Yee

Task: REQ7 make sequence diagram
Teammate responsible: Fadi Alailan

Task: REQ7 make class diagram
Teammate responsible: Lim Hong Yee

Task: REQ7 make Design Rationale and explain our implementation
Teammate responsible: Khor Jia Shin

I accept this WBA(Hong Yee)
I accept this WBA(Fadi Alailan)
I accept this WBA(Khor Jia Shin)