WBA:

WBA for Assignment 2 - FIT 2099 Team: Lab 14 Team 3

Members: Jun Heng Tan (3280 5217), Dayan Perera (3224 2026), Kuan Yi Xuan (3202 3227)

REQ 1: Code and JavaDoc-->
Managed by Kuan Yi Xuan (3202 3227).
- Will be reviewed by Dayan Perera (3224 2026) & Jun Heng Tan (3280 5217)

REQ 2: Code and JavaDoc -->
Managed by Jun Heng Tan (3280 5217).
- Will be reviewed by Dayan Perera (3224 2026) & Kuan Yi Xuan (3202 3227)

REQ 3: Code and JavaDoc -->
Managed by Jun Heng Tan (3280 5217).
- Will be reviewed by Dayan Perera (3224 2026) & Kuan Yi Xuan (3202 3227)

REQ 4: Code and JavaDoc -->
Managed by Dayan Perera (3224 2026).
- Will be reviewed by Jun Heng Tan (3280 5217) & Kuan Yi Xuan (3202 3227)

REQ 5: Code and JavaDoc -->
Managed by Dayan Perera (3224 2026).
- Will be reviewed by Jun Heng Tan (3280 5217) & Kuan Yi Xuan (3202 3227)

REQ 6: Code and JavaDoc -->
Managed by Kuan Yi Xuan (3202 3227).
- Will be reviewed by Jun Heng Tan (3280 5217) & Dayan Perera (3224 2026)

REQ 7: Code and JavaDoc -->
Managed by Jun Heng Tan (3280 5217).

- Will be reviewed by Kuan Yi Xuan (3202 3227) & Dayan Perera (3224 2026)

Jun Heng Tan: I accept this WBA

Dayan Perera: I accept this WBA

Kuan Yi Xuan: I accept this WBA

Changes Made to Design Rationale for Assignment 2:

REQ 4: Conversion of ground to Dirt and and dropping coin when Player has capability INVINCIBLE no longer done by moveActorAction. Instead managed by Player class playTurn method. Will check if this.hasCapability(Status.INVINCIBLE). If true check if this current location is not dirt or floor. If true, proceed to set ground to dirt and drop a coin of value 5 on the current location.

REQ 5: Player class, modification of method player PlayTurn() method, specifically method of adding value of collected coins in inventory to instance variable wallet. All coins's value in inventory will be added to the wallet first and to a list name toRemove. A loop will then be utilised to remove every coin in the player's inventory.

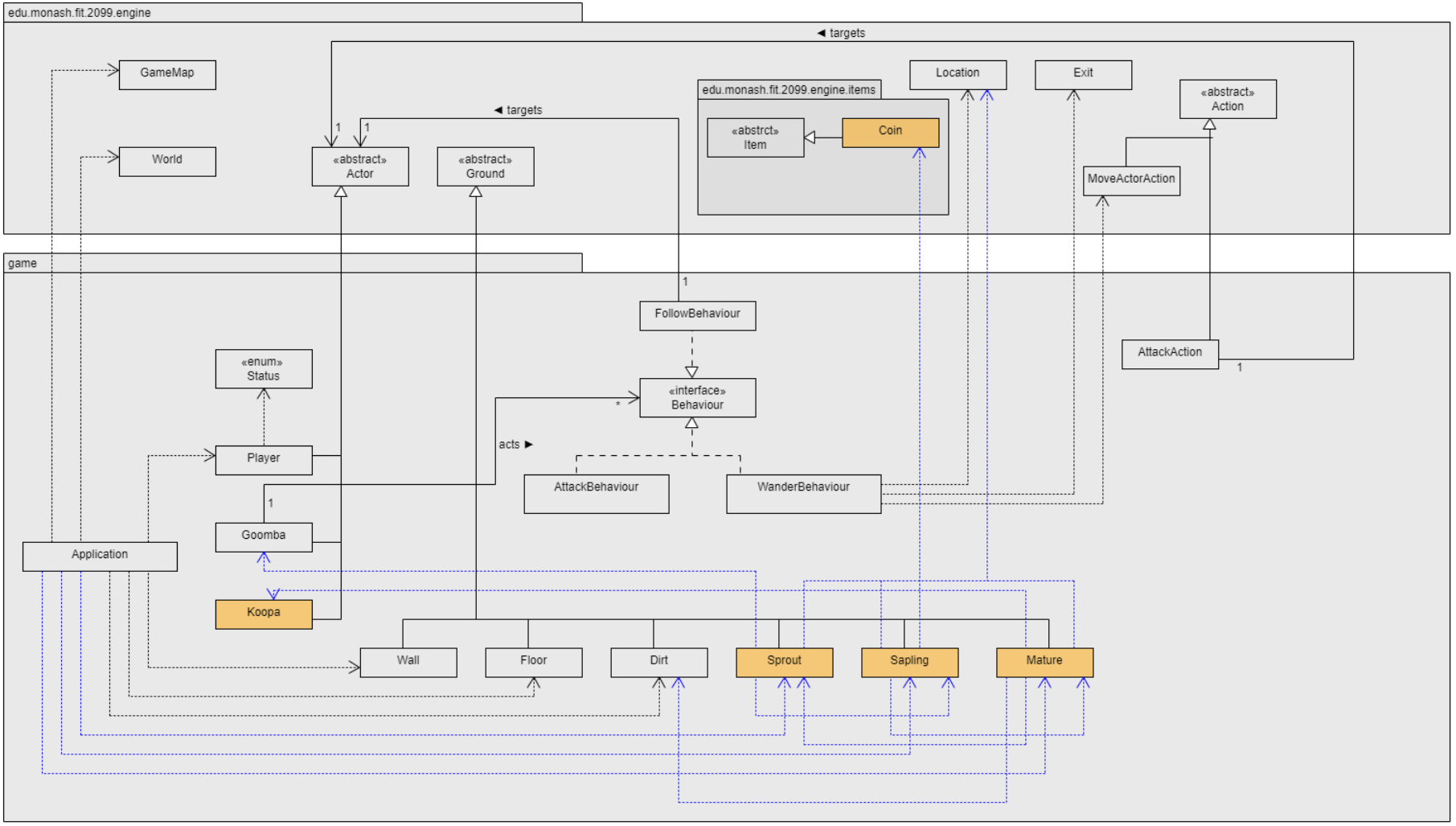
REQ 6: New means of deciding which string to output for the SpeakAction. Prevents a possibility of an infinite loop.

REQ 7: Use of a boolean instance variable to check if an object has resetted, instead of incrementing a variable storing the reset count.

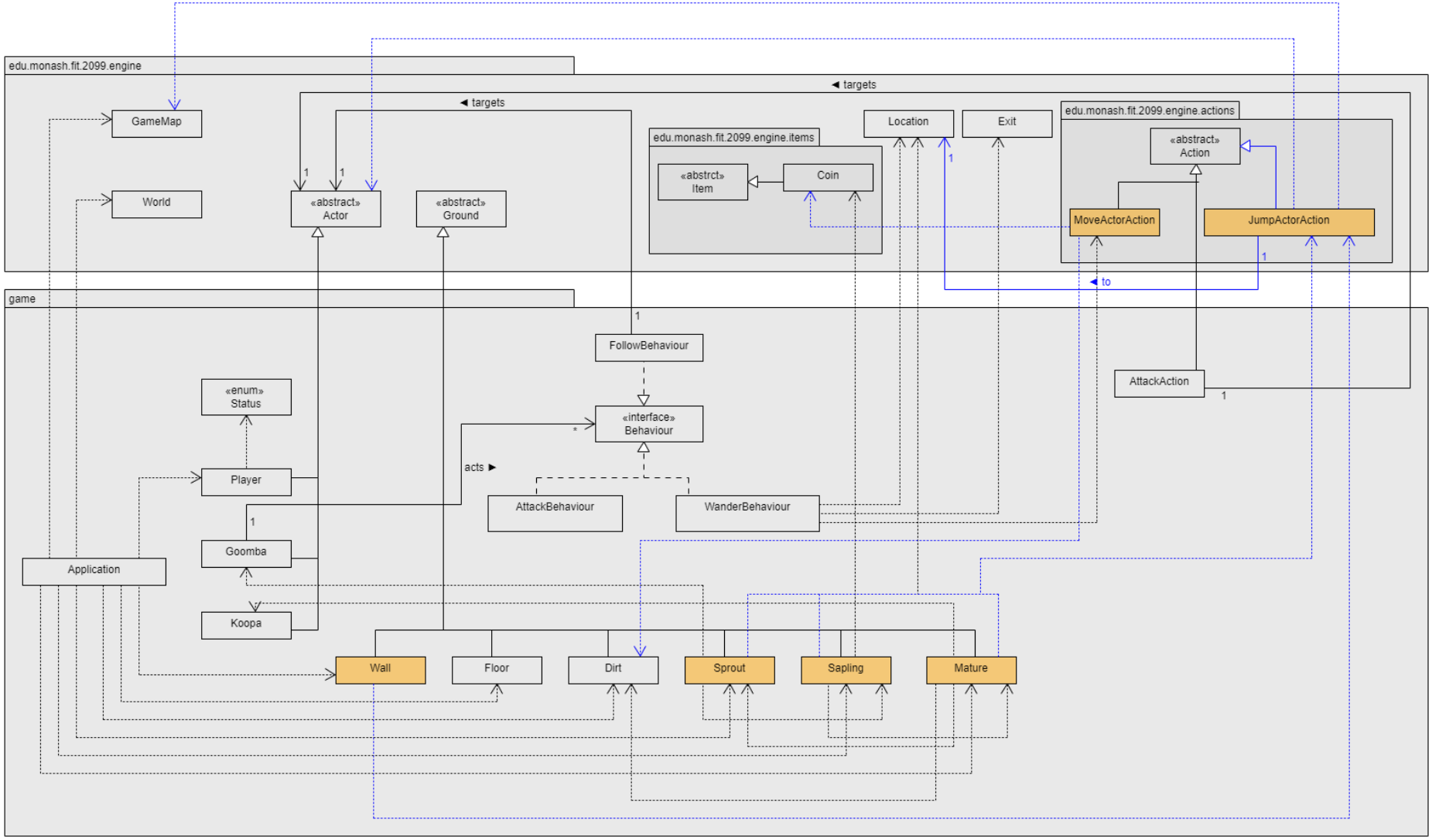
REQ 1 - UML Class:

---

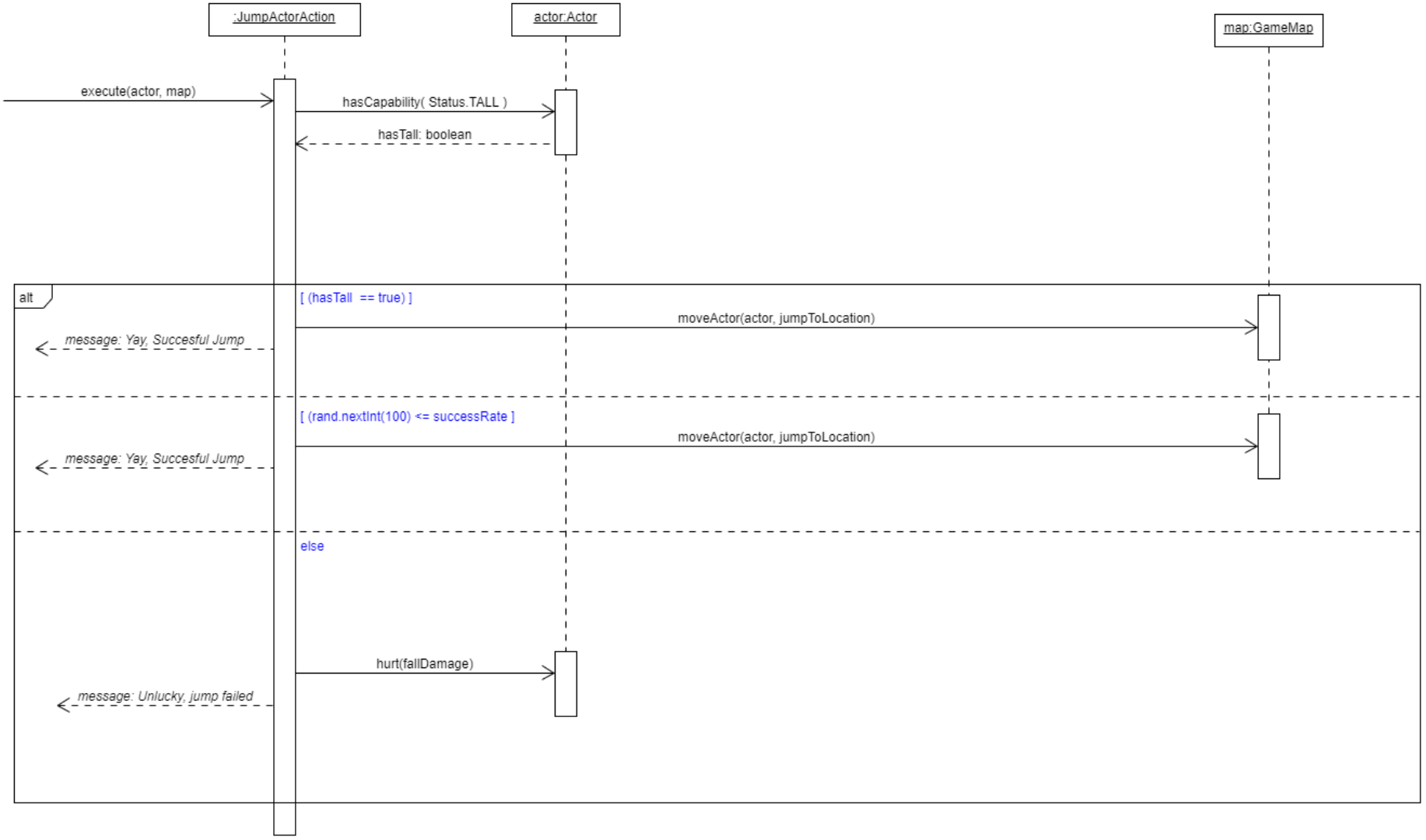
* Orange classes are new classes added and blue lines are new relationships.



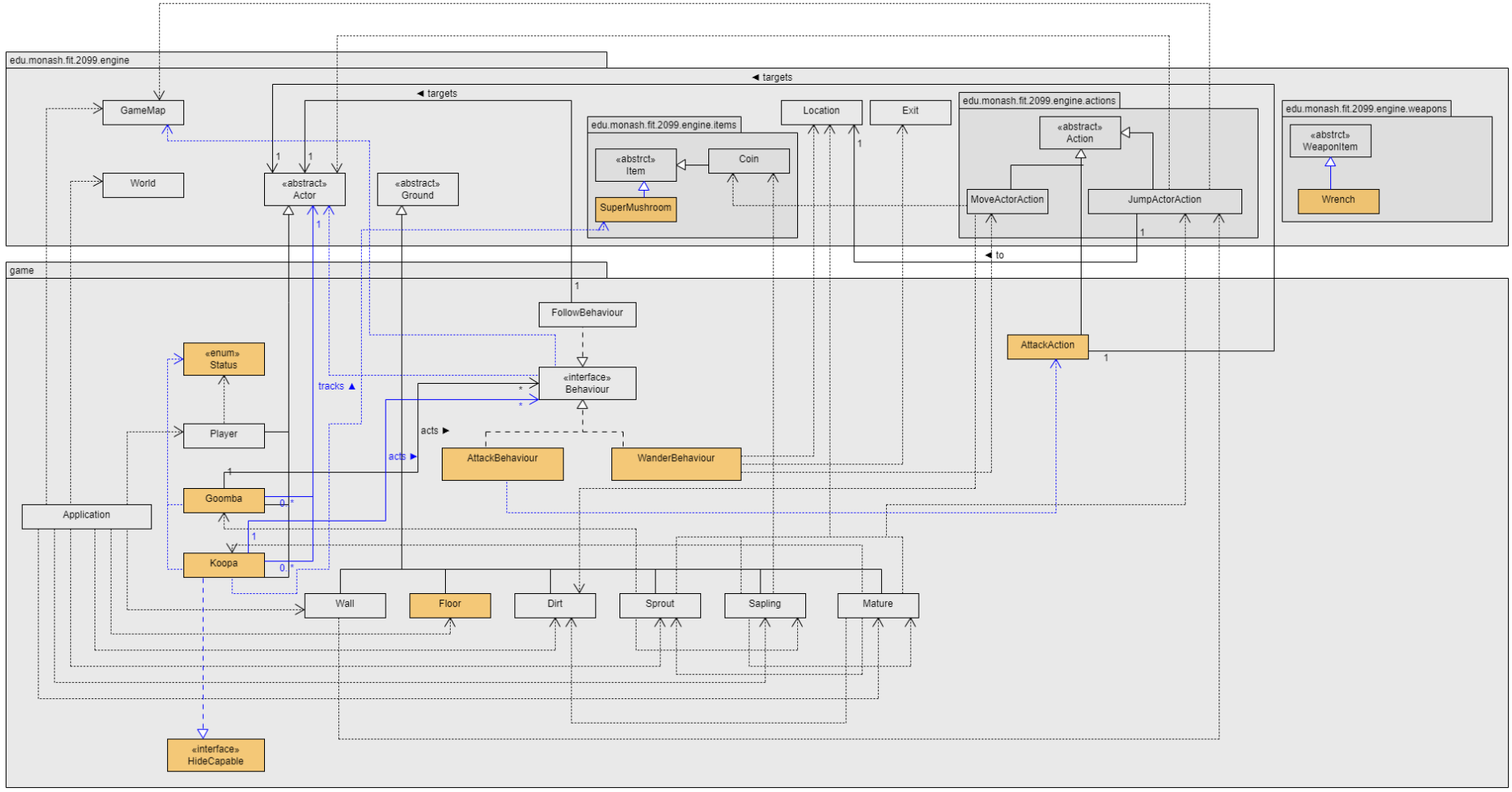
REQ 2 - UML Class:



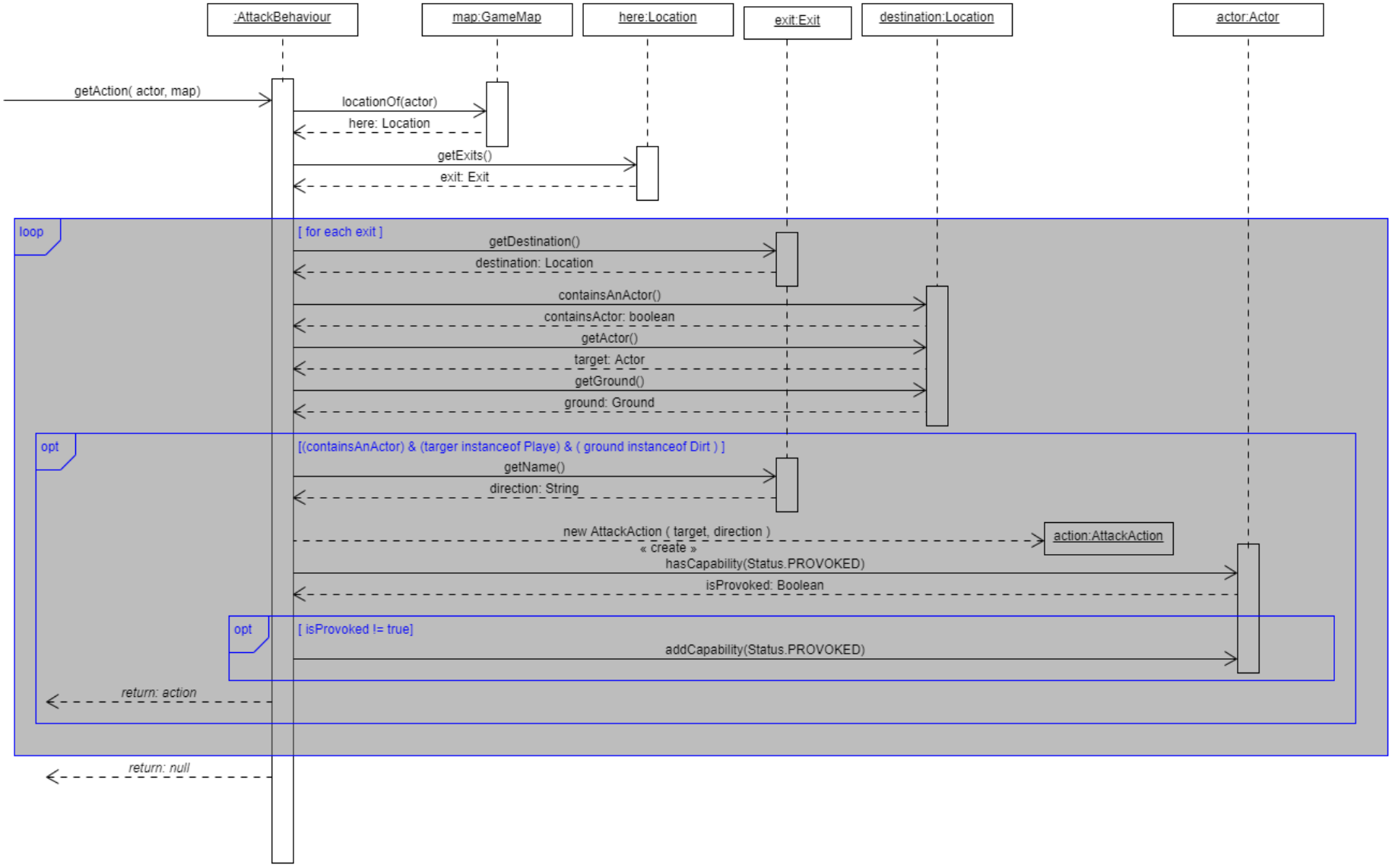
execute.JumpActorAction:



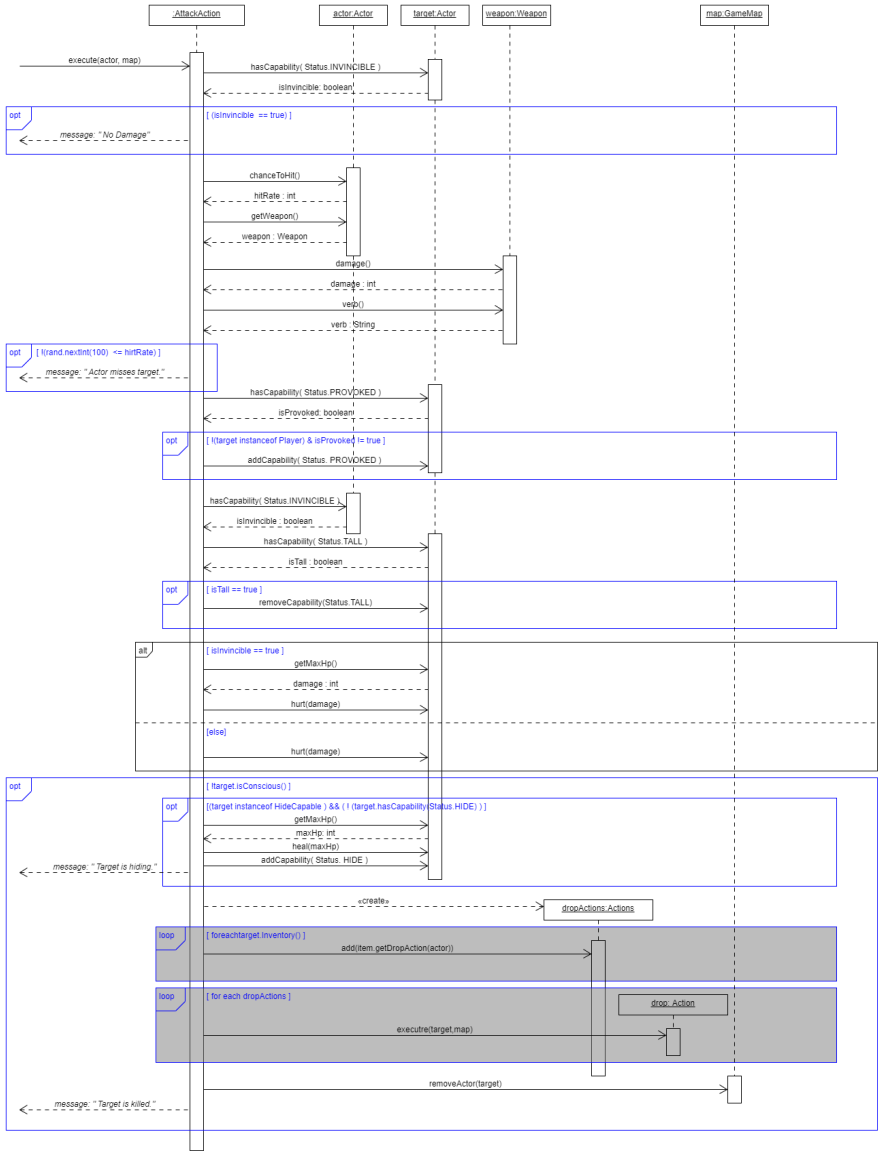
REQ 3 - UML Class:



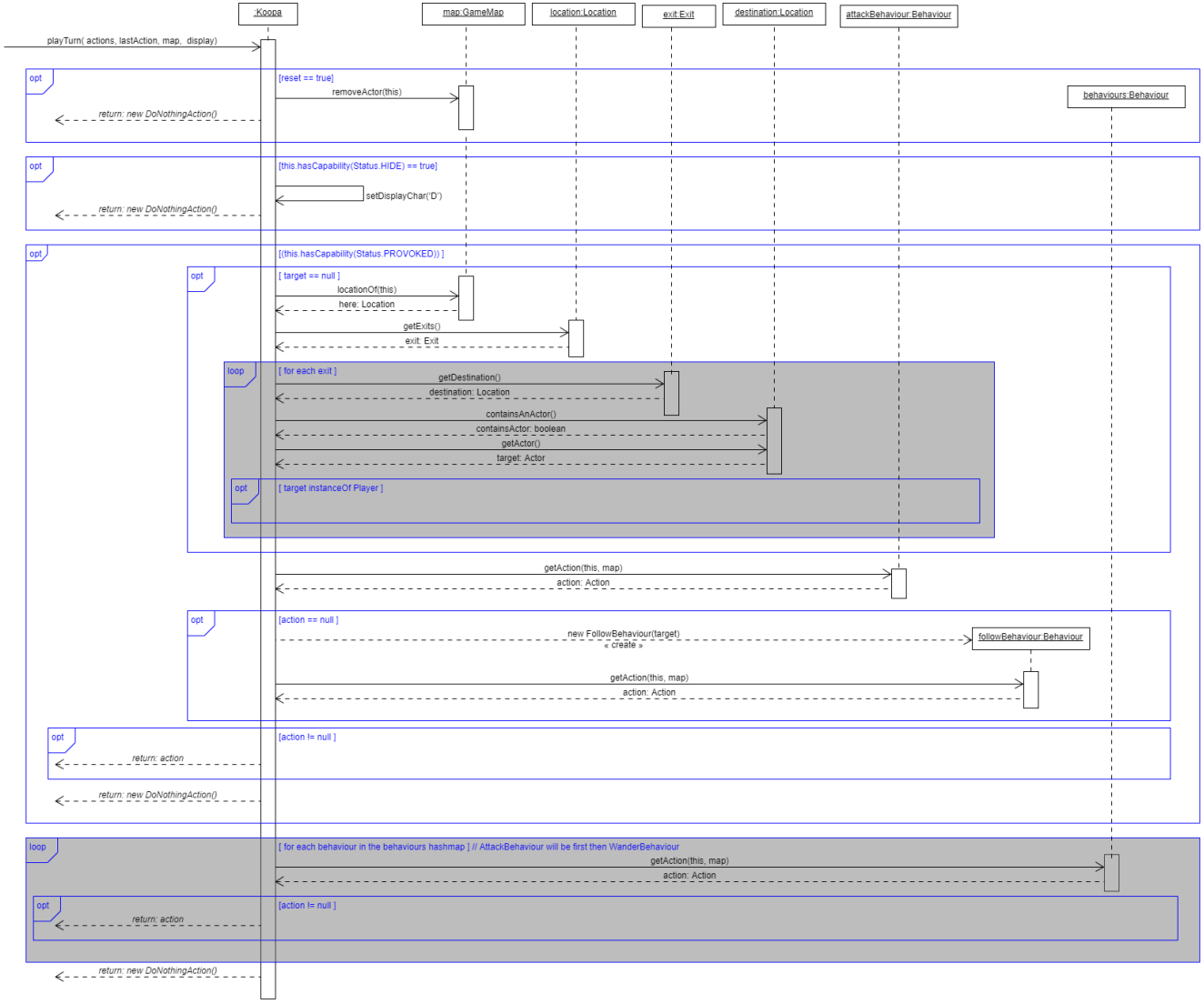
AttackBehaviour.getAction:



execute.AttackAction:

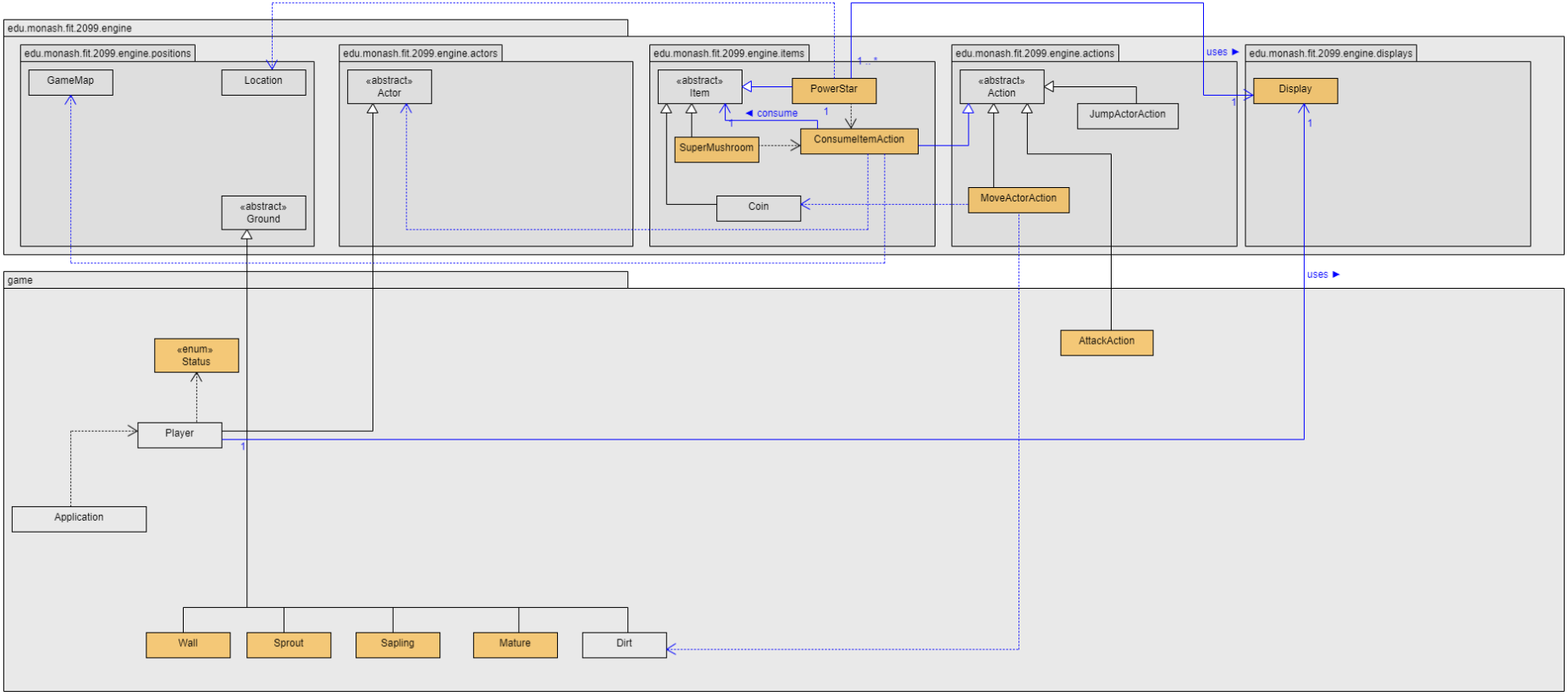


Koopa.playTurn:

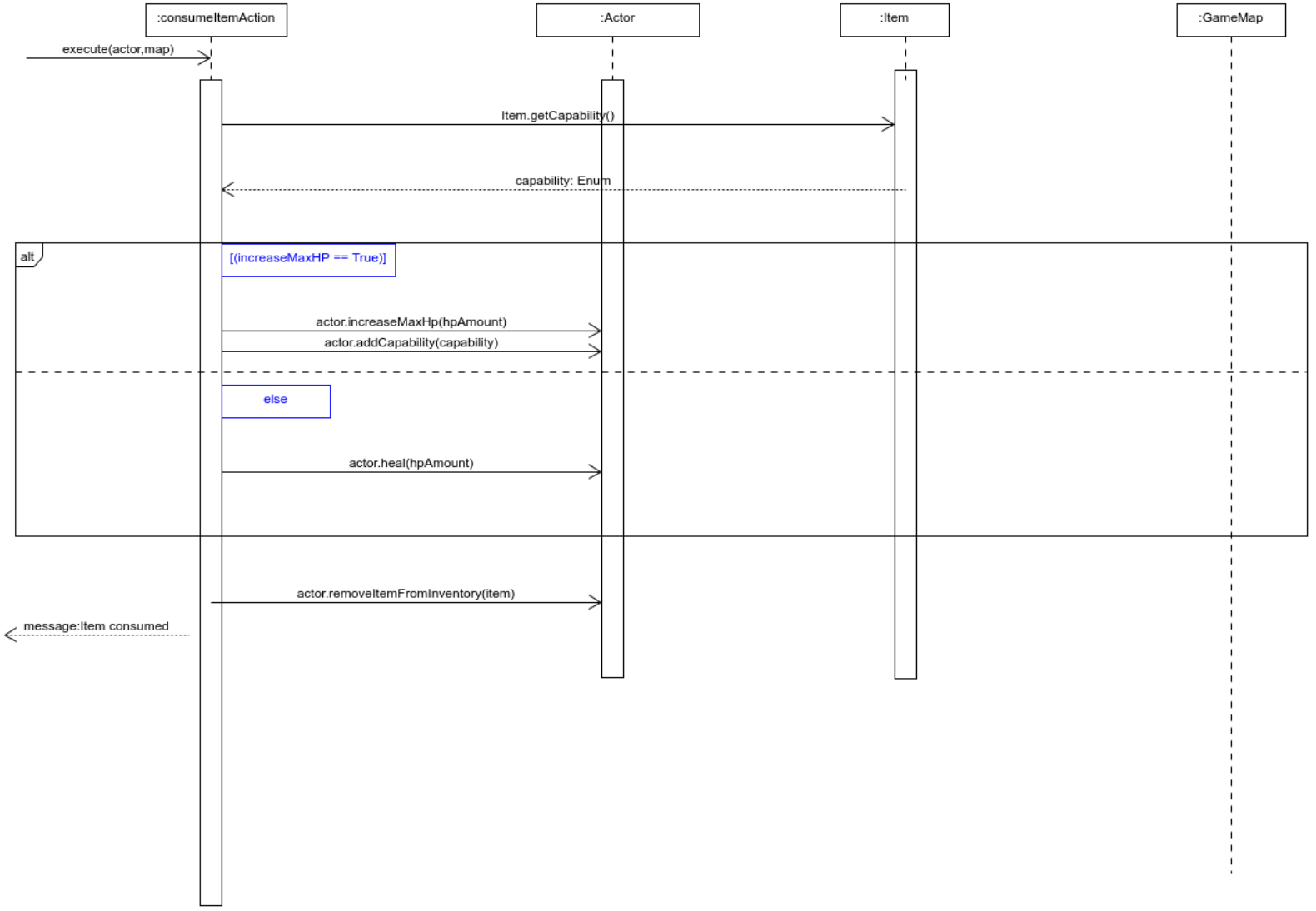


* For Goomba replace 2nd opt with 10% suicide calculation. Will removeActor(this) if true, then return a new DoNothingAction.

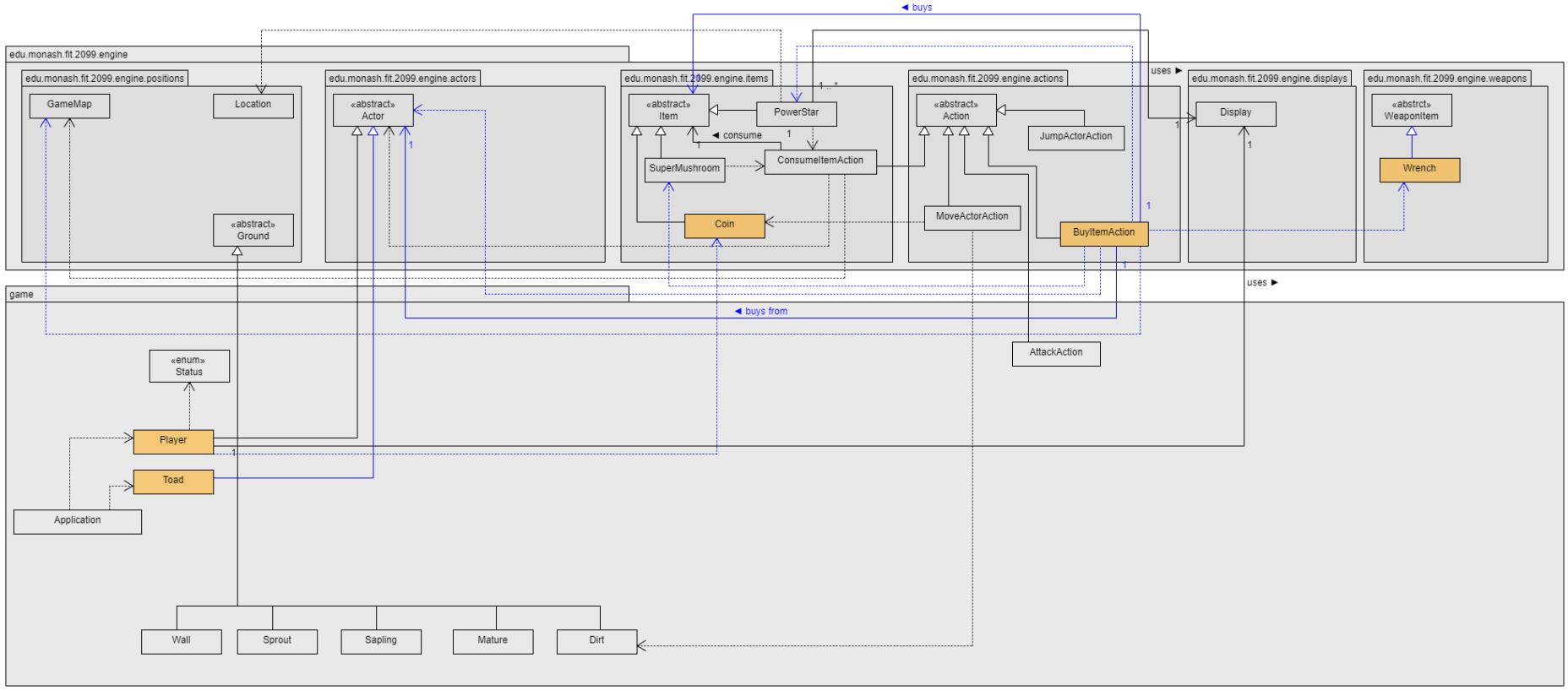
REQ 4 - UML Class:



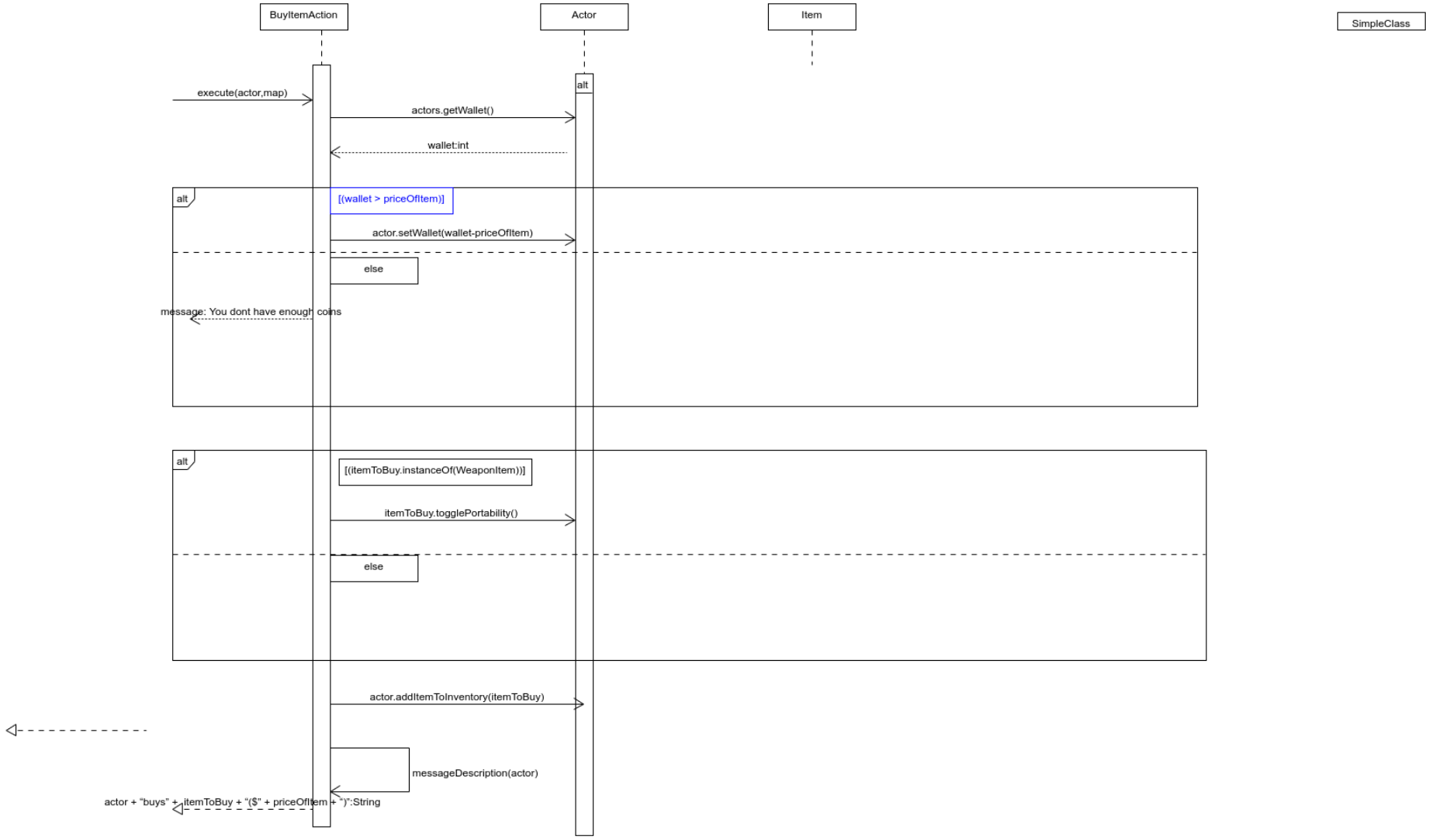
execute.ConsumeItemAction:



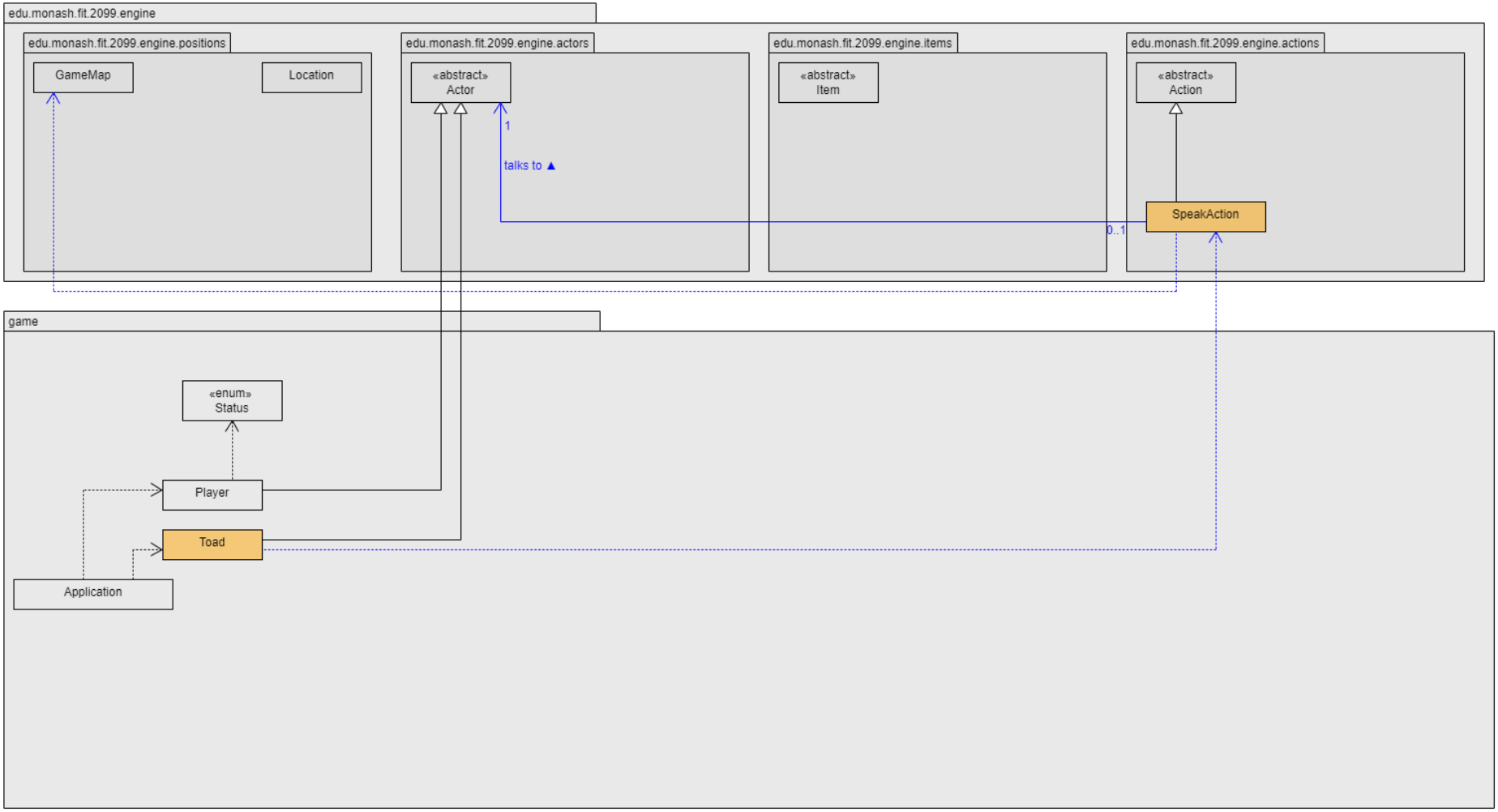
REQ 5 - UML Class:



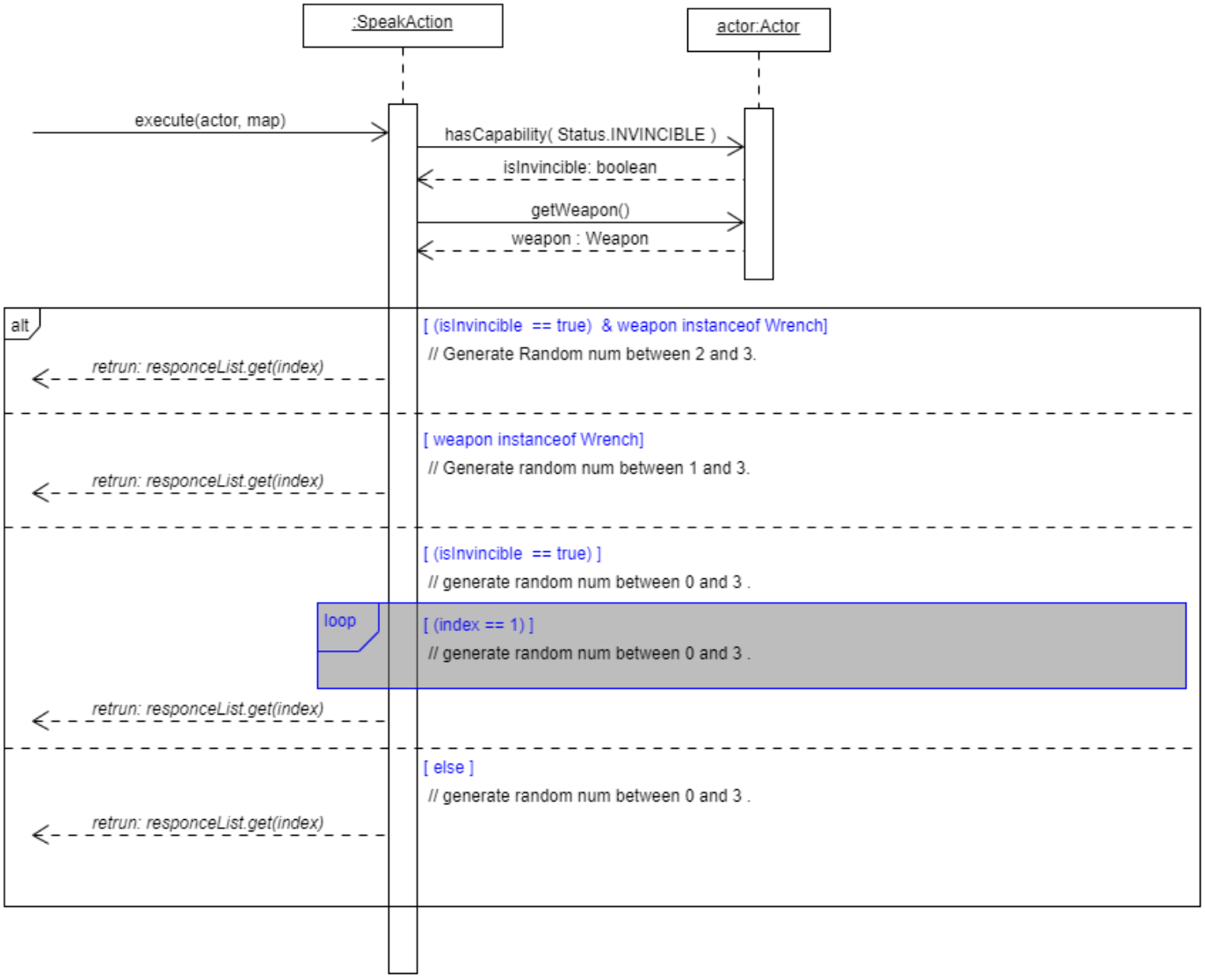
execute.BuyItemAction:



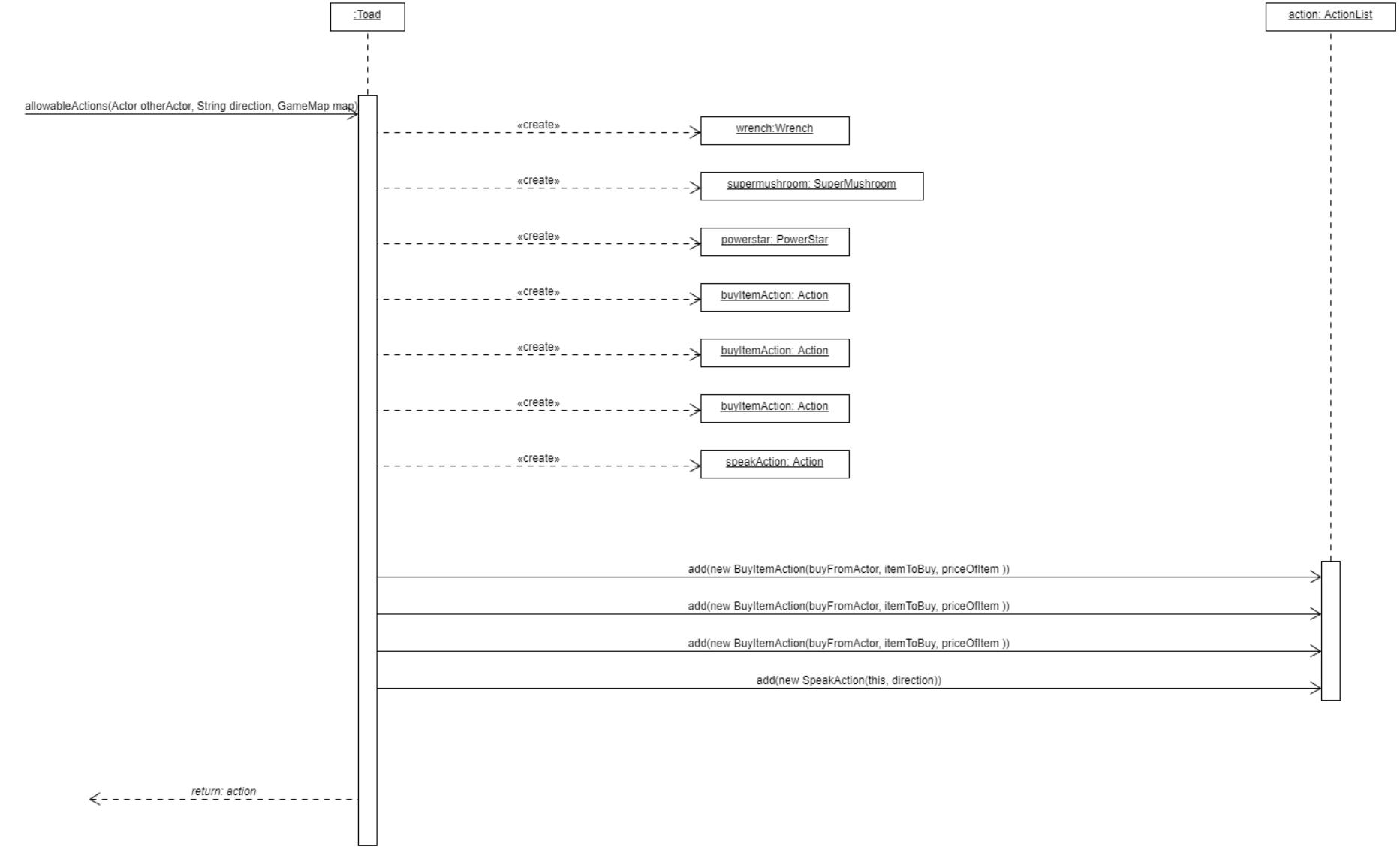
REQ 6 - UML Class:



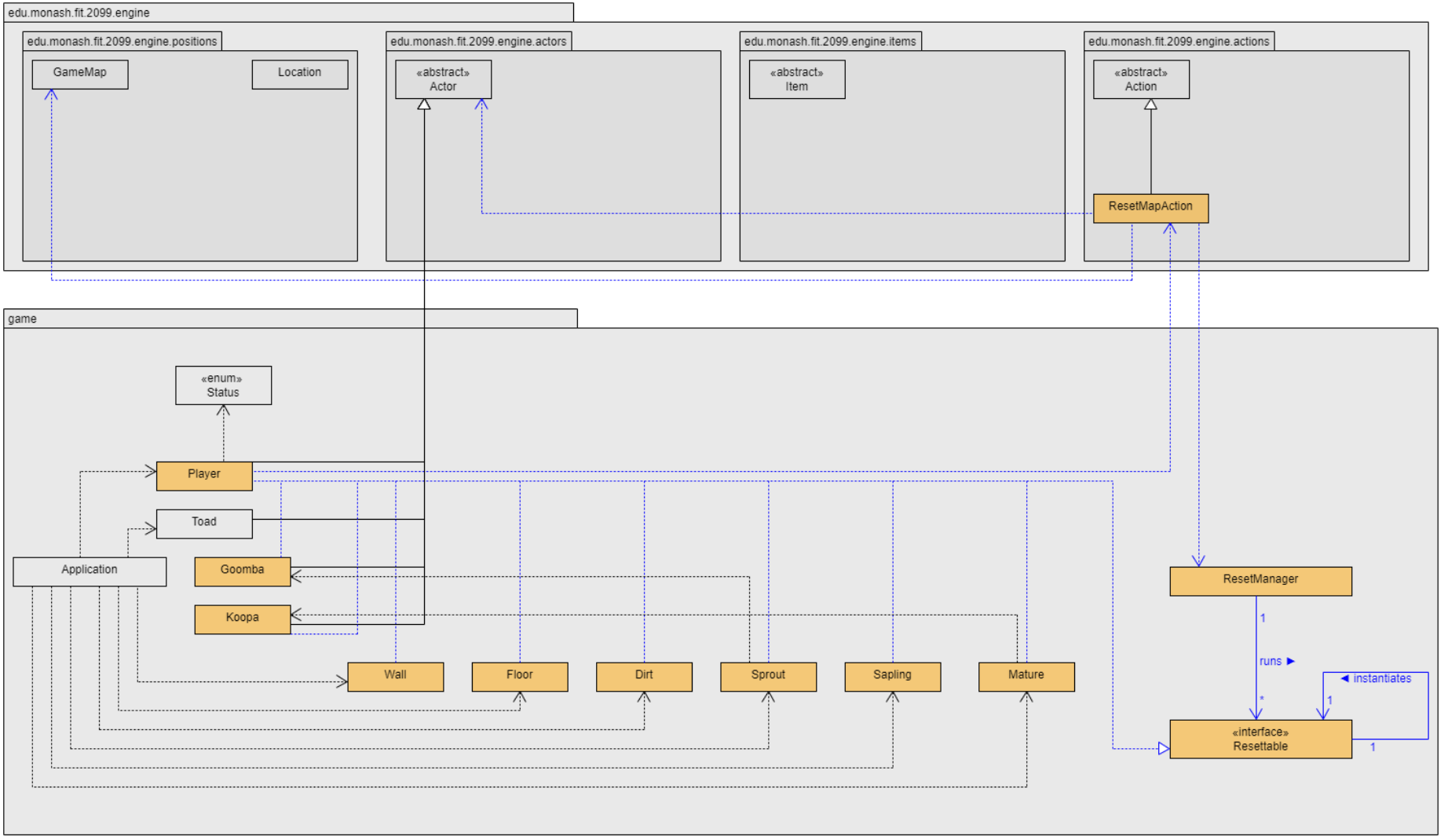
execute.speakAction:



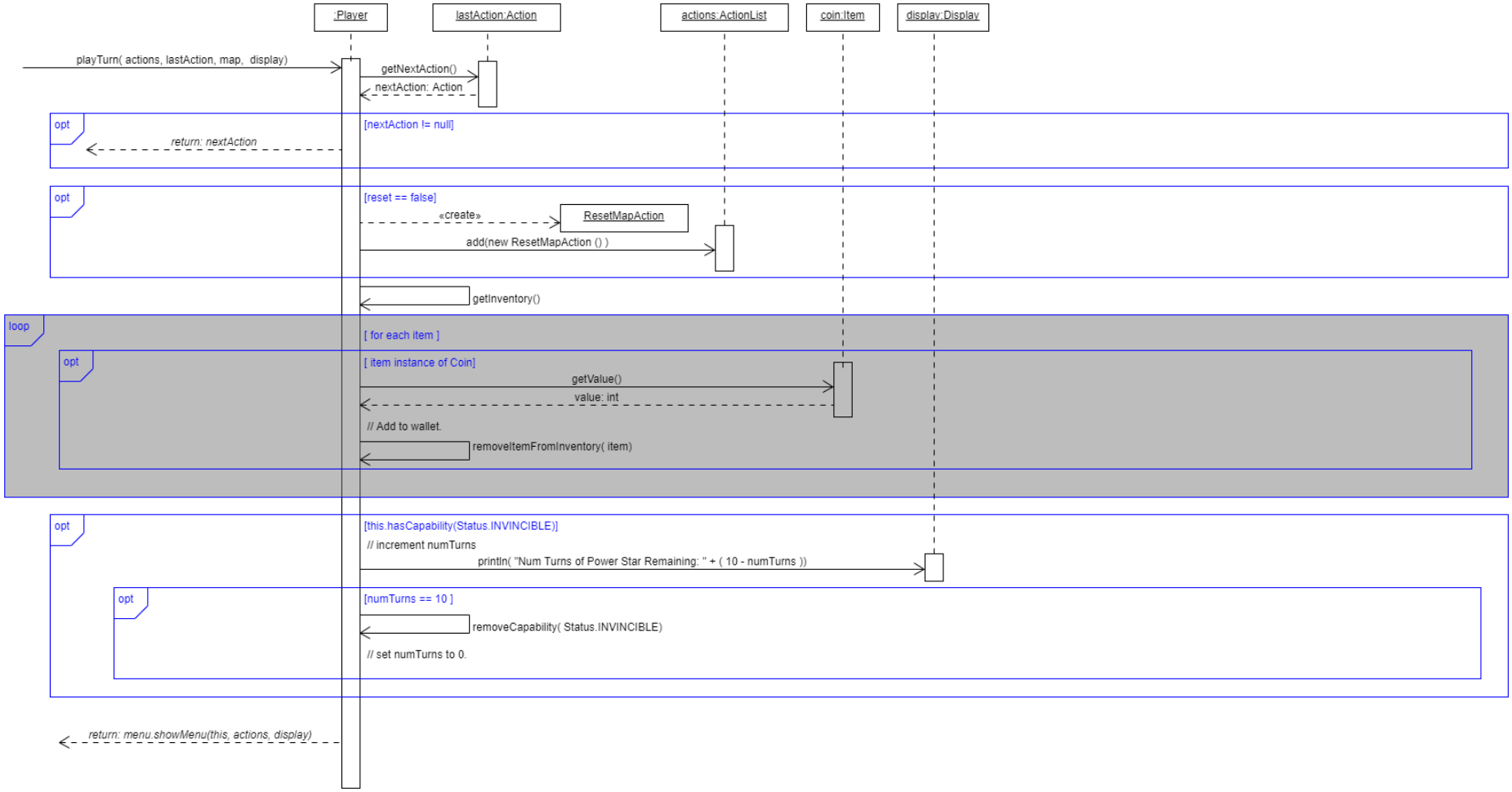
toad.allowableActions:



REQ 7 - UML Class:



Player.playTurn:



REQ1: Let it grow!

- Create new concrete classes of Sprout, Sapling & Mature. All extending Ground as its parent class.
    - Tree class will thus be removed, although it can be made to an abstract class and be extended by the newly generated classes. It is unnecessary as abstract class Ground will contain all the needed abstract methods and methods to run the newly generated classes. Meaning abstract class Tree will just be an empty class.
    - With new types of ground, changes will need to be made in the Application class when initialising the FancyGroundFactory object to include the new grounds.
    - For each class, a new private instance variable name NumTurns will be declared, utilised in recording the number of turns that have passed. It will be initialised to 0 in their respective constructors.

- All 3 classes will have one integral method tick() in their respective classes. Overriding the original method definition in the abstract class Ground. It takes in a Location object as its parameter.
    - Responsible for recording the number of turns that have passed. Done through increasing the private attribute numTurns by 1 each time the method is called.
    - Responsible for central decision making. Will check if criteria for certain actions are met, and if met called those actions. (eg. Sprout class → Call spawnGoomba() method, if the 10 % chance is met)
    - Actions are not performed in method tick() to minimise method responsibility and to ensure the Single Responsibility Principle is followed.

- New Concrete Class Koopa & Coin will be created (methods in them will be explained later down this document)
    - Class Coin will extend abstract class Item.
    - Class Koopa will inherit from abstract class Actor.

- java.util.Random will be utilised in calculation of percentages
    - Eg. for 10 % → if (random.nextInt(100) <= 10) …

Sprout Class:

Methods → Sprout(), tick(Location location), spawnGoomba(Location location), becomeSapling(Location location)

- Sprout() ⇒ Constructor for Sprout class. Set displayChar to appropriate character. Initialise numTurns to 0.
- tick(Location location) ⇒ Increments numTurns by 1. Check conditions for spawnGoomba() and becomeSapling() methods to be called.
- spawnGoomba(Location location) ⇒ Spawns Goomba at current location. Takes in a Location object (currentLocation) as a parameter to be utilised in its function.
    - location.containsAnActor() == false
    - location.map().at(location.x(), location.y()).addActor(new Goomba())
- becomeSapling(Location location) ⇒ Sets ground to become a Sapling. Takes is a Location object (currentLocation) as a parameter to be utilised in its function.
    - location.setGround(new Sapling())

Sapling Class

Methods in Sapling Class → Sapling(), tick(Location location), dropCoin(Location location) , becomeMature(Location location)

- Sapling() ⇒ Constructor for Sapling class. Set displayChar to appropriate character. Initialise numTurns to 0. Initialise a new Coin object to instance variable coin of type Coin.
- tick(Location location) ⇒ Increments numTurns by 1. Check conditions for dropCoin() and becomeMature() methods to be called.
- dropCoin(Location location) ⇒ Will add a new Coin item to the current location. Will take in a Location object (currentLocation) as a parameter to be utilised in its function. Can append multiple coin objects to a single Location.
    - location.addItem(new Coin(int value))
- becomeMature(Location location) ⇒ Sets ground to become Mature. Takes is a Location object (currentLocation) as a parameter to be utilised in its function.
    - location.setGround(new Mature())

Mature Class:

Methods in Mature Class → Mature(), tick(Location location), spawnKoopa(Location location) , growSprout(Location location), becomeDirt(Location location)

- Mature() ⇒ Constructor for Mature class. Set displayChar to appropriate character. Initialise numTurns to 0.
- tick(Location location) ⇒ Increments numTurns by 1. Check conditions for spwanKoopa(), growSprout() and becomeDirt() methods to be called.
- spawnKoopa(Location location) ⇒ Spawns Koopa at current location. Takes in a Location object(currentLocation) as a parameter to be utilised in its function.
    - location.containsAnActor() == false
    - location.map().at(location.x(), location.y()).addActor(new Koopa())
- growSprout(Location location) ⇒ grows a sprout at a random surrounding fertile square. Takes in a Location object (currentLocation) as a parameter to be utilised in its function.
    - Determines possible grow locations by checking Exits of current Location is of type Dirt. If true, store it in a Ground arrayList.
        - location.getExits()
    - Check if Ground arrayList.size() is greater or equal to 1. If true …
        - Select random destination of exit by producing a random number in range of the size of the Ground arrayList. Getting that location and setting its ground to a newSprout.
    - numTurns will be reset to 0. Allowing Mature to continue growing Sprouts.
- becomeDirt(Location location) ⇒ Sets ground to become Dirt. Takes is a Location (currentLocation) object as a parameter to be utilised in its function.
    - location.setGround(new Dirt())

REQ2: Jump Up, Super Star!

- Grounds that have to be jumped to enter will have new instance variables int successRate, int fallDamage, & String groundName. All passed when new JumpActorAction called.
    - Double successRate stores jump action success rate for that type of ground
    - int fallDamage stores damage to be dealt to Actor if jump fails
    - String groundName stores name of ground. E.g. “ Wall”
    - Ensure open closed principle, by removing the need to check instanceof Wall/ Sprout … in JumpActorAction to determine successRate and fallDamage to be dealt.

- Grounds that have to be jumped to enter will have overridden/ modified canActorEnter() methods. (i.e. Wall, Sprout, Sapling, Mature)
    - returns true if the Actor does have the capability of INVINCIBLE else false.
        - MoveActorAction returned if INVINCIBLE instead of JumpActorAction. The benefit of consuming PowerStar.

- No changes needed to be made for the enemy Goomba and Koopa classes. Will not be able to jump.
    - PlayTurn() method in their respective classes prevents this.

- allowableActions() method in grounds (e.g. Wall, Sprout, Sapling, Mature) that can be jumped over will overridden to change implementation inherited from abstract class Ground.

<u>allowableActions(Actor actor, Location location, String direction):</u>

- If Actor has capability INVINCIBLE it returns an empty Action list. - (Ensures that no JumpAction options are in menu)
- Else Creates and Returns an ActionList object which contains a single new JumpActorAction().
    - Check if location is the Actor's current location before creating JumpActor Action. This is done to prevent the option for the Actor to jump to its current location.
        - if (location != location.map().locationOf(actor)) {}
    - If location != Actor’s current location. It will actions.add(new JumpActorAction (Location jumpToLocation, String direction, int successRate, int fallDamage, String groundName))

The cause of the issue is a result of a line in class World, method processActorTurn().
- actions.add(here.getGround().allowableActions(actor, here, ""))
- Which checks for allowable actions at the Actor’s current location

- New concrete class JumpActorAction created inheriting from the Action class. All abstract methods will be overridden

<u>JumpActorAction Class:</u>

Instance Variables → Location jumpToLocation, String direction, int successRate, int fallDamage, String groundName
- jumpToLocation stores location to jump to.
- direction stores direction of jump. (eg. North, NorthWest …)
- successRate stores probability of successfully jumping to that Ground
- fallDamage stores damage to be dealt if jump fails
- groundName stores name of Ground that is being jumped to.

Methods in JumpActorAction Class → JumpActorAction(...), execute(Actor actor, GameMap map), menuDescription(Actor actor)

- JumpActorAction(…) ⇒ Constructor for JumpActorAction class. Constructor will include parameters (Location jumpToLocation, String direction, int successRate, int fallDamage, String groundName) used to initialise all instance variables.
- execute(Actor actor, GameMap map) ⇒ Evaluates the result of the jump.
    - Check if Actor has TALL capability. If true map.moveActor(actor, jumpToLocation). Return appropriate String.
    - Elif check if [rand.nextInt(100) <= successRate]. If true map.moveActor(actor, jumpToLocation). Return appropriate String.
    - Else, actor.hurt (fallDamage). And return appropriate message.
- menuDescription(Actor actor) ⇒ menuDescription will output the required menu display.
    - “ Player jumps” + direction + “to” +groundName

REQ3: Enemies

- Override CanActorEntor method for class Floor. Will only return true if Actor instanceof Player else false.

- New class Wrench created, it will extend abstract class WeaponItem
    - Contains a 0 parameter constructor. Which initialises name, displayChar, damage, verb & hitRate to appropriate values.

- New class SuperMushroom created, it will extend abstract class Item.
    - Will contain one 0 parameter constructor. Which initialises name, displayChar & portable to appropriate values.

- Modification to Enum class Status, adding enumeration of PROVOKED and HIDE.
    - Once PROVOKED Goomba will immediately retaliate in the same turn.
    - Once PROVOKED Koopa will immediately retaliate in the same turn.
    - HIDE used to indicate that Koopa has been defeated and now Dormant.

- Interface HideCapable created and implemented by Koopa class. Will be utilised as a marking for Koopa. Ensuring when Koopa's HP == 0, it won't implement getDropAction and output that Koopa is killed. Instead resets Max Hp and sets it to have capability Status.HIDE. This capability will then setup Koopa to become Dormant, done in Koopa's playTurn().
    - Implemented to ensure Open-closed Principle.

Interface HideCapable: No methods and variables.

- AttackBehaviour class will be modified. Will be called when Goomba/ Koopa is PROVOKED or Player is in its surroundings.
    - getAction(Actor actor, GameMap map) ⇒ Will be overridden.
        - For every exit of the Actor's location. Check if an Actor is there & if the Actor is on ground Dirt & instanceOf Player. If yes, return new AttackAction(). Will use Exits of Actor's location to get the target and its direction for the new AttackAction.
            - map.locationOf(actor).getExits()
            - destination = exit.getDestination()
            - Action action = newAttackAction(target, direction)
        - Before returning attackAction. Check if Actor hasCapability(Status.PROVOKED).
            - If false, actor. addCapability(Status.PROVOKED)
        - Return null otherwise. If it is not possible to have return AttackAction.

- FollowBehaviour implements interface Behaviour. Utilised by both Goomba and Koopa.
    - Implements code to follow a target (Player).
    - Only implemented by both Goomba and Koopa if PROVOKED. Additionally, will only be implemented if AttackBehaviour get action returns null, meaning that player is not within striking distance. This check is performed in Goomba & Koopa's respective playTurn() method.

- Modification of execute() method in AttackAction class.
    - Check if the target hasCapability (Status.INVINCIBLE). If true returns " No Damage"

- Check if the attack was successful [!(rand.nextInt(100) <= hitRate)]. Return appropriate statement if missed.
- If the attack was successful. Method will perform a check for if target is not an instanceof Player & if target does not have Capability(Status. PROVOKED)
    - If true, meaning target != instance of Player and does not have Status. PROVOKED. It will addCapability(Status. PROVOKED)
- Check if the target has Capabilities TALL. If true, remove that capability.
- Check if the Actor has INVINCIBLE Capabilities. If yes, deal full damage to target by getting target.getMaxHp and dealing the MaxHp as its damage.
    - hurt(maxHp)
- Else deal the target with the weapon's damage.
    - hurt(weapon.damage())
- Check if (!target.isConscious()).
    - If true, then proceed to check (target instanceof HideCapable) && (! (target.hasCapability(Status.HIDE)).
        - If true, heal to maxHp
        - addCapability(Status.HIDE)
        - Return appropriate message about target hiding.
        - Check for target having capability Hide is done to ensure Koopa can only hide once.
- Continue Previous implementation.

- Modification to Goomba class

Goomba Class:

Instance Variables →

- Map<Integer, Behaviour> behaviours = new HashMap<>() → Stores mapping of behaviour and their priority.
    - Only contains <1, AttackBehaviour > and < 10, WanderBehaviour >.
- Actor target = null → Stores the Actor object (Mario/ player) that attacked it. It is utilised to track location of player/ target, allowing the implementation of FollowBehaviour(). playTurn() method will be used to initialise it.

Methods → Goomba(), getIntrinsicWeapon(), playTurn(…), allowableActions(…)

- Goomba() ⇒ Constructor for class Goomba. Will set name, displayChar. hitPoints, initialise possible behaviours
    - this.behaviours.put(1, new AttackBehaviour())
    - this.behaviours.put(10, new WanderBehaviour())
- getIntrinsicWeapon() ⇒ Creates and returns an intrinsic weapon. It is an Overridden method inherited from the Actor class.
    - “Kick”, 10
- playTurn(ActionList actions, Action lastAction, GameMap map, Display display) ⇒ Overrides the method in Actor class. Figures out what to do next.
- allowableActions(…) ⇒ No modifications made

PlayTurn(ActionList actions, Action lastAction, GameMap map, Display display):

- Method returns an action to be executed.
- Check if the 10% suicide possibility is met. If true.
    - map.removeActor(this)
    - return new DoNothingAction()

- Checks if Status.PROVOKED. If it is true, it will then check if the target instance variable != null. If == null, will initialised it by looping through Exits and finding the target (the instance variable will be empty only for the initial attack, thus target will be located in one of Goomba's adjacent squares)
    - It will then implement AttackBehaviour.
        - Action action = behaviours.get(1).getAction(this, map) ;
    - Check if action != null, if true, return action. This ensures if Goomba can hit the target it will do it instead of following target.
    - Else FollowBehavior will be implemented.
        - Behaviour behaviour = new FollowBehaviour(target)
        - Action action = behaviour.getAction(this, map)
    - Check if action == null. If true, a new DoNothingAction() will be returned, else action obtained by WanderBehaviour.getAction() will be returned.
- Else will loop through behaviours HashMap to get action.
    - So either AttackBehaviour or WonderBehaviour will be implemented. If both behaviours return no actions, a new DoNothingAction() will be returned. AttackBehaviour will be called first, so Goomba will always hit the target if it is in its surroundings instead of Wondering. If AttackBehaviour.getAction() returns an action, add capability PROVOKED to this Goomba object.

- Creation of new concrete class Koopa which extends abstract class Actor and Implements Interface HideCapable.

<u>Koopa Class implements HideCapable:</u>

Instance Variables →

- Map<Integer, Behaviour> behaviours = new HashMap<>() → Stores mapping of behaviour and their priority.
    - Only contains <1, AttackBehaviour > and < 10, WanderBehaviour >.
- Actor target = null → Stores the Actor object (Mario/ player) that attacked it. It is utilised to track location of player/ target, allowing the implementation of FollowBehaviour. playTurn() method will be used to initialise it.

Methods → Koopa(), getIntrinsicWeapon(), playTurn(…), allowableActions(…)

- Koopa() ⇒ Constructor for class Koopa. Will set name, displayChar. hitPoints, initialise possible behaviours and add a SuperMushRoom to its inventory.
    - this.addItemToInventory(new SuperMushroom())
    - this.behaviours.put(1, new AttackBehaviour())
    - this.behaviours.put(10, new WanderBehaviour())
- getIntrinsicWeapon() ⇒ Creates and returns an intrinsic weapon. It is an Overridden method inherited from the Actor class.
    - "Punch" , 30
- playTurn(ActionList actions, Action lastAction, GameMap map, Display display) ⇒ Overrides the method in Actor class. Figure out what to do next.
- allowableActions(…) ⇒ Actions that can be performed on Koopa.
    - Check if otherActor.getWeapon() instance of Wrench & this.hasCapability(Status.HIDE) & (otherActor.hasCapability(Status.HOSTILE_TO_ENEMY). If true actions.add(new AttackAction(this,direction)).
        - Allows attack on Koopa with Wrench when Koopa is Dormant.
    - Check if this.hasCapability(Status.HIDE) == false & (otherActor.hasCapability(Status.HOSTILE_TO_ENEMY). If true If true actions.add(new AttackAction(this,direction)).
        - Prevents attack with weapons other than wrench when Koopa is Dormant.

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

- Method returns an action to be executed.
- Checks if this.hasCapability(Status.HIDE). If it does…
    - this.setDisplayChar('D')
    - return new DoNothingAction()
- Checks if Status.PROVOKED. If it is true, it will then check if the target instance variable != null. If == null, will initialised it by looping through Exits and finding the target (the instance variable will be empty only for the initial attack, thus target will be located in one of Koopa's adjacent squares)
    - It will then implement AttackBehaviour.
        - Action action = behaviours.get(1).getAction(this, map) ;
    - Check if action != null, if true, return action. This ensures if Koopa can hit the target it will do it instead of following target.
    - Else FollowBehavior will be implemented.
        - Behaviour behaviour = new FollowBehaviour(target)
        - Action action = behaviour.getAction(this, map)
    - Check if action == null. If true, a new DoNothingAction() will be returned, else action obtained by FollowBehaviour.getAction() will be returned.
- Else will loop through behaviours HashMap to get action.
    - So either AttackBehaviour or WonderBehaviour will be implemented. If both behaviours return no actions, a new DoNothingAction() will be returned. AttackBehaviour will be called first, so Koopa will always hit the target if it is in its surroundings instead of Wondering. If AttackBehaviour.getAction() returns an action, add capability PROVOKED to this Koopa object.

Req4: Magical Items

- A SuperMushroom class and PowerStar class will be created.
    - Both will extend abstract class Item and implement interface Printable and Capable

- Add new enum to Status named INVINCIBLE.
    - TALL = capability gained from the consumption of SuperMushroom.
    - INVINCIBLE = capability gained from the consumption of PowerStar.

- New Class ConsumeItemAction which extends abstract class Action.
    - Allows an Actor to consume an item.
    - Option will pop out on the menu console when the Actor is directly on top of the item or the item is in Actor's inventory.
    - Result of code in Class World method ProcessActorTurn().

- No modification to DropItemAction & PickUpItemAction. Which allows an item to be picked or dropped.
    - World class adds a DropItemAction for every item in the inventory.
    - World class adds PickUpItemAction for every item at an Actor's Location.

<u>SuperMushroom Class:</u>

Attributes → int hpAmount, boolean increaseMaxHP = true
- hpAmount stores hp amount to be healed or increased.
- increaseMaxHP stores whether the passed HP should be used to increase max HP or heal.

Methods → SuperMushroom()

- SuperMushroom() ⇒ Constructor for SuperMushroom class. Will add a new ConsumeItemAction to its allowableActions ActionList. Will set hpAmount to 50. Add capability TALL to the item as well.
    - super("SuperMushroom", '^', true)
    - this.addAction(new ConsumeItemAction(itemTobeConsumed, hpAmount, increaseMaxHP))
    - hpAmount = 50
    - this.addCapability(Status.TALL)

Changes below are shown in previous requirements(except for ConsumeItemAction) / already implemented:

Changing display chart to an uppercase letter is managed by the Player Class method getDisplayChar().
Increasing max HP by 50 will be managed by ConsumeItemAction.
Jumping freely with a 100% success rate managed by JumpActorAction.
Effect lasting until it receives any damage managed by AttackAction.

PowerStar Class:

Attributes → int numTurns, int hpAmount, boolean increaseMaxHP = false
- numTurns acts as the fading turn's ticker
- hpAmount stores hp amount to be healed or increased.
- increaseMaxHP stores whether the passed HP should be used to increase max HP or heal.

Will have a static Display display = new Display()
- Utilised in outputting the number of remaining turns

Methods → PowerStar(), tick(Location currentLocation), tick(Location currentLocation, Actor actor)

- PowerStar() ⇒ Constructor for PowerStar class. Will add a new ConsumeItemAction to its allowableActions ActionList. Will set hpAmount to 200. Add capability INVINCIBLE to the item as well.
    - super("Power Star", '*', true)
    - this.addAction(new ConsumeItemAction(itemTobeConsumed, hpAmount, increaseMaxHP))
    - hpAmount = 200
    - this.addCapability(Status.TALL)
- tick(Location currentLocation) ⇒ Method will implement the fading feature of Powerstar when item is not in Player's inventory. numTurns will be incremented each time the method is called. Will further check if numTurns == 10, if true it will remove the item from location. Will also have a display.println() call, used to output the number of remaining turns before PowerStar fades away.
    - location.removeItem(this)
- tick(Location currentLocation, Actor actor) ⇒ Method will implement the fading feature of Powerstar when item is in Player's inventory. numTurns will be incremented each time the method is called.Will further check if numTurns == 10, if true it will remove the item from location. Will also have a display.println() call, used to output the number of remaining turns before PowerStar fades away.k
    - So regardless if item is on the ground or in the actor's inventory, item will fade away and be removed from the game within 10 turns
    - actor.removeItemFromInventory(this)

Changes below are shown in previous requirements(except for ConsumeItemAction) / already implemented:

Being able to walk to highgrounds is managed by all high Ground class methods CanActorEnter() and AllowableActions().
Convert to coins for every destroyed ground managed by MoveActorAction class.
Automatically destroy (convert) ground to Dirt is manage by MoveActorAction class
Immunity, all enemy attacks becoming useless is managed by AttackAction class.
Attacking enemies. When active, a successful attack will instantly kill enemies is managed by AttackAction class.

ConsumeItemAction Class:

Attributes → Item toConsume, int hpAmount, boolean increaseMaxHP
- toConsume stores item to be consumed.
- hpAmount stores hp amount to be healed or increased.
- increaseMaxHP stores whether the passed HP should be used to increase max HP or heal.

Methods → ConsumeItemAction (Item Item toConsume, int hpAmount, boolean increaseMaxHP), execute(Actor actor, GameMap map), menuDescription(Actor actor)

- ConsumeItemAction (Item Item toConsume, int hpAmount, boolean increaseMaxHP) ⇒ Used in initialising all attributes.
- execute(Actor actor, GameMap map) ⇒
    - check item's capability and add it to the actor.
    - check if increseMaxHP == true. If true actor.increaseMaxHp(hpAmount). Else actor.heal(hpAmount).
    - Remove item from inventory and location. Remove from location because Actor can directly consume item without storing it in inventory first.
    - Return appropriate String message.
- menuDescription(Actor actor) ⇒ outputs action to menu. (actor + " consumes " + toConsume)

- Modification of the Player class.
    - New Attribute int numTurns = 0. Used to keep track of the number of turns remaining after consumption of PowerStar.
    - Will have a static Display display = new Display(). Used in displaying the appropriate messages.
    - Modification of playTurn method.
        - Add a new if statement checking if player has capability INVINCIBLE. If yes it will increase numTurns by one. And display.println() the number of turns remaining for using the capability till it fades. Inside the if statement will be another conditional statement where it checks if numTurns == 10. If true, it will this.removeCapability(Status.INVINCIBLE) and reset numTurns to 0 for further use of capability in the future.
        - Will continue the previous implementation after this.

Req5:Trading

- Modification of Player class.
    - New instance variable int wallet. Stores the total value of all coins collected by Player. Initialised to 0 in its declaration.
    - Modification of playTurn method. Will be used as a means of adding coins in inventory to the wallet and removing those coins. When playTurn() is called, it will now loop through all items in Player's inventory checking if they are an instanceof class Coin. If yes, the Coin's value will be obtained using ((Coin) item).getValue() and added to the player's wallet. The coin will also be removed from inventory. Will continue previous implementation after this.
    - New methods getWallet() and setWallet().

- Class Coin extends abstract class Item implements Printable, Capable
    - Will have an instance variable int value. Which stores the value of this Coin object.
    - Will have a constructor method. Will take one parameter, that being the value of the Coin. Sets name, displayChar, portability and value of Coin
        - super("Coin", '$', true)
        - this.value = value;

    - New method, getValue(). Returns value of Coin object.

- New Class Toad which extends abstract class Actor.
    - Toad is placed in the centre of the map, surrounded by walls which are manually placed. So changes in application will need to be made.
        - Creating a new Actor toad placing it in the centre of the map.
        - Adding walls by changing map's string

- New BuyItemAction class extends the abstract class Action.

- Class Wrench will extend abstract class WeaponItem.
    - Will have a constructor. Takes in no parameters. Will initialise appropriate values to name, displayChar, damage, verb, hitRate.

Toad Class:

Instance Variables →

Methods → Toad(), allowableActions(...), playTurn(...)

- Toad() ⇒ Constructor of class Toad. Initialises name, displayChar and hitPoints to appropriate values.
- allowableActions(...) ⇒ Explained below.
- playTurn(...) ⇒ returns a new DoNothingAction.

---

allowableActions(Actor otherActor, String direction, GameMap map):

- Initialise ActionList actions = new ActionList().
- actions.add 3 new BuyItemAction.
    - Item itemToBuy = new SuperMushRoom()
    - new BuyItemAction (Actor buyFromActor, Item itemToBuy, int priceOfItem)
- Return actions.

BuyItemAction Class:

Instance Variables → Actor buyFromActor, Item item, int priceOfItem
- buyFromActor records Actor in which Player will buy from.
- item records Item to be bought
- priceOfItem records the price for buying the item.

Methods → BuyItemAction(...), execute(...), menuDescription(...)

- BuyItemAction(Actor buyFromActor, Item item, int priceOfItem) ⇒ Constructor for BuyItemAction. Will initialise all instance variables
- execute(Actor actor, GameMap map) ⇒
    - Checks actor's wallet(using wallets getter), see if it is > priceOfItem. If not, return String "You don't have enough coins!"
        - ((Player) actor).getWallet()
    - else. Deduct priceOfItem from Player's Wallet and set it.
        - ((Player) actor).setWallet()
    - Check if itemToBuy is not an instanceOf WeaponItem. If true ensure that it can't be dropped using itemToBuy.togglePortability()
    - Add itemToBuy to Player's inventory. (For PowerStar fading effect will not start until item is in Location's inventory or and Actor's because of tick() methods, thus will have 10 turns remaining).

- Return actor + " bought " + itemToBuy + " from " + target
- menuDescription(Actor actor) ⇒ what to display for the menu.
  - Return actor + "buys" + itemToBuy + "($" + priceOfItem + ")"

fading duration of PowerStar will be managed by PowerStar class.

REQ6: Monologue.

- Modification of Toad Class.
  - Modification of AllowableActions(). Adding a new SpeakAction to actions ActionList after adding the 3 BuyItemAction().
    - actions.add(new SpeakAction(Actor talkToTarget))

- New class SpeakAction extends abstract class Action.

SpeakAction Class:

Instance Variables → Actor talkToTarget, Random rand = new Random()
- talkToTarget stores the Actor in which player will talk to

Methods → SpeakAction(...), execute(...), menuDescription(...)

- SpeakAction (Actor talkToTarget) ⇒ Constructor for SpeakAction class. Will set instance variable talkToTarget.
- execute(Actor actor, GameMap map) ⇒ Returns appropriate message.
  - Stores the 4 Strings into an arrayList response..
    - 0: "You might need a wrench to smash Koopa's hard shells."
    - 1: "The Princess is depending on you! You are our only hope."
    - 2: "Being imprisoned in these walls can drive a fungus crazy :("
    - 3: "You better get back to finding the Power Stars."
  - Check if actor has Capability INVINCIBLE and actor.getWeapon instanceof Wrench.
    - If true. Generate random index between 1 and 2. And return response.get(index)
  - Elif check if actor.getWeapon instanceof Wrench
    - If true. Generate random index between 1 and 3. And return response.get(index)
  - Elif check if actor has Capability INVINCIBLE
    - If true. Generate random index between 0 and 2. And return response.get(index)
  - Else generate random index between 0 and 3. And return response.get(index)
- menuDescription(Actor actor) ⇒ what to display for the menu.
  - Return actor + " talks with" talkToTarget.

Random index between a range generated by index = rand.nextInt(max - min + 1) + min
- max = maximum value and min = minimum value.

REQ7: Reset Game

- Interface Resettable will be implemented by all resettable objects.

- ResetManager Class manages all resettable objects.Will only have one instance of it for the whole game.
  - Modification of run() ⇒ loop through all ressetable objects in the resettableList and call their resetInstance() method

- Modification of appendResetInstance(Resettable reset) ⇒ Add ressetable objects to resettableList
    - resettableList.add(reset)
- Modification of cleanUp(Resettable resettable ⇒ Remove ressetable objects from resettableList
    - resettableList.remove(resettable)

- Create new ResetMapAction Class which extends abstract class Action. Action to be executed when the player wishes to reset game.

ResetMapAction Class:

Attributes → String hotKey
- hotKey stores hot key for implementing the action

Methods → ResetMapAction(), execute(Actor actor, GameMap map), menuDescription(Actor actor)

- ResetMapAction() ⇒ Constructor for ResetMapAction Class. Will have no parameters. Initialises hotKey to "r"
- execute(Actor actor, GameMap map) ⇒ Executes action. Will call ResetManager.getInstance().run(). And return appropriate String stating Game has been reset.
- menuDescription(Actor actor) ⇒ method for returning appropriate message on menu.
    - Return " Reset the game."

- Modification of Player Class. Will now implement interface Ressetable.
    - New instance variable boolean reset = false. Stores if the Player object has been called to reset.
    - Modification of Constructor ⇒
        - Will now include this.registerInstance(), which adds this player object to the ressetableList in ResetManager
    - Modification of playTurn() method ⇒
        - Check if (lastAction.getNextAction() != null). If true return lastAction.getNextAction();
        - Check if reset == false. If true, add a new ResetMapAction to the passed ActionList actions. This allows the ResetMapAction to be displayed on the menu.
        - Then continue the previous implementation.
    - Override resetInstance() method ⇒
        - Set instance variable reset to true.
        - Remove the capability of TALL and INVISIBLE from the player.
        - getMaxHp and heal(MaxHp)

- Modification of Sprout, Sapling & Mature Classes. Will now implement interface Ressetable. Changes will apply and be the same for all.
    - New instance variables. boolean reset = false & boolean hasResseted = false.
        - reset stores whether resetInstance() method have been called.
        - Stores if this object has already been reset
    - Modification of Constructor ⇒
        - Will now include this.registerInstance(), which adds this object to the ressetableList in ResetManager
    - Override resetInstance() method ⇒
        - Sets instance variable reset to true.
    - Modification of tick(Location location) method ⇒

        - When the method is now called. It will now start by checking if [(this.reset == true) & (!hasResetted)]. If true …
            - Check if (random.nextInt(100) <= 50). If true, set current location to Dirt.
                - location.setGround(new Dirt())
            - Loop through all Item objects stored at the current location's inventory. And check if they are instanceOf Coin. If true, remove the Item object from location's inventory.
            - Set hasRessetted to true. This ensures reset of objects are only done once.
        - Then continue previous implementation.

- Modification of Dirt, Floor and Wall clases. Changes will apply and be the same for all.
    - New instance variables. boolean reset = false & boolean hasResseted = false.
        - reset stores whether resetInstance() method have been called.
        - resetCount stores num of time resetInstance() method have been called.
    - Modification of Constructor ⇒
        - Will now include this.registerInstance(), which adds this object to the ressetableList in ResetManager
    - Override resetInstance() method ⇒
        - Sets instance variable reset to true.
    - Override tick(Location location) method from abstract class Ground ⇒
        - Check if [(this.reset == true) & !hasResetted)]. If true …
            - Loop through all Item objects stored at the current location's inventory. And check if they are instanceOf Coin. If true, remove the Item object from location's inventory.
            - Set hasRessetted to true. This ensures reset of objects are only done once.

- Modification of Goomba and Koopa class. Changes will apply and be the same for both.
    - New instance variables. boolean reset = false
        - reset stores whether resetInstance() method have been called.
    - Modification of Constructor ⇒
        - Will now include this.registerInstance(), which adds this object to the ressetableList in ResetManager
    - Override resetInstance() method ⇒
        - Sets instance variable reset to true.
    - Modification of playTurn(ActionList actions, Action lastAction, GameMap map, Display display) ⇒
        - When the method is now called. It will now start by checking (this.reset == true). If true …
            - remove actor from map.
                - map.removeActor(this)
            - return new DoNothingAction()
        - Then continue previous implementations.