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 the 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 the 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, so we can make them extend the enemy class 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 a hostile actor.

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

Basic features of enemy

"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 the 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 the canActorEnter method in Floor class, to check if the actor is hostile to the player, false will be returned, true otherwise.

Features of Goomba

"Goomba starting with 20 HP"

In the constructor itself, we can initialise Goomba with 20 hitpoints.

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

This is easily achievable by overriding the 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 utilise 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.

Features of Koopa

"Koopa starts with 100 HP"

In the constructor itself we can initialise Koopa with 100 hitpoints.

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

This is easily achievable by overriding the 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 initialised 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 its dormant state. For the display character, we can override getDisplayChar method, and check if Koopa is in dormant state or not, if yes then replace the char with 'D'.

"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. It will get every item in Koopa's inventory and drop it, and then picked up by the player, and lastly utilise 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 BreakShellAction, a SuperMushroom item will be created and added to the location where Koopa's shell is broken.

AttackBehaviour's design

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 the getExits method. For each exit, we will get its location using the getDestination method. For each of those locations, we will check if there is an actor at a current location and whether that actor has the capability of being hostile to enemies, if so, then create an AttackAction in that direction.