FIT2099 Assignment 3 Work Breakdown Agreement

Team 5 Lab 6

1. Goh Kai Yuan (30881919)
2. Jonathan Koh Tong (32023146)
3. Ting Yi Xuan (32577346)

WBA (Work Breakdown Agreement)

We hereby agree to work on FIT2099 Assignment 3 according to the following breakdown:

- **Class diagrams:**
  Completion: 22/5/2022
  - **REQ1: (Author)** Ting Yi Xuan, **(Reviewers)** Jonathan Koh, Goh Kai Yuan
  - **REQ2: (Author)** Jonathan Koh, **(Reviewers)** Ting Yi Xuan, Goh Kai Yuan
  - **REQ3: (Author)** Goh Kai Yuan, **(Reviewers)** Jonathan Koh, Ting Yi Xuan
  - **REQ4: (Author)** Everybody, **(Reviewers)** Everybody
  - **REQ5: (Author)** Everybody, **(Reviewers)** Everybody
  -

- **Interaction diagrams:**
  Completion: 22/5/2022
  - **REQ1: (Author)** Ting Yi Xuan, **(Reviewers)** Jonathan Koh, Goh Kai Yuan
  - **REQ2: (Author)** Jonathan Koh, **(Reviewers)** Ting Yi Xuan, Goh Kai Yuan
  - **REQ3: (Author)** Goh Kai Yuan, **(Reviewers)** Jonathan Koh, Ting Yi Xuan
  - **REQ4: (Author)** Everybody, **(Reviewers)** Everybody
  - **REQ5: (Author)** Everybody, **(Reviewers)** Everybody

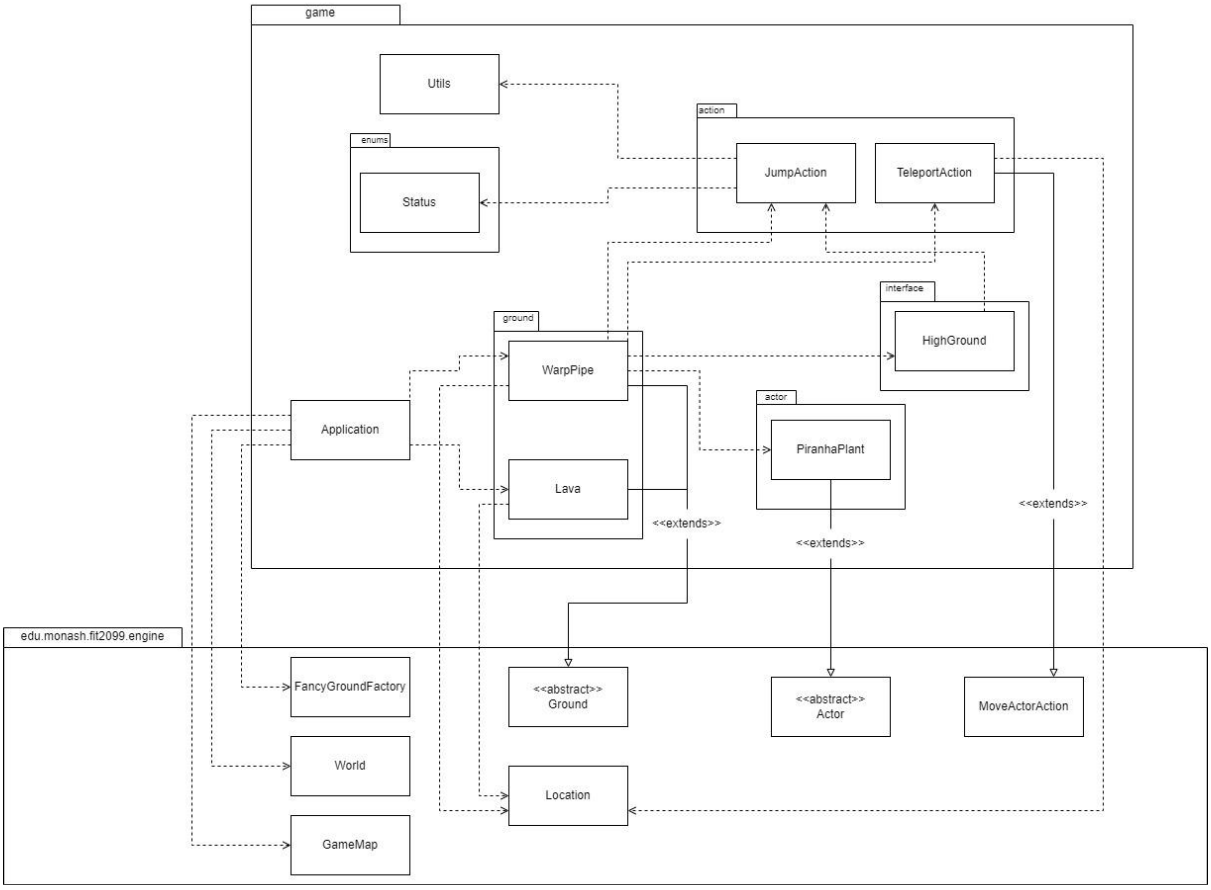
- **Design rationale:**
  Completion: 22/5/2022
  - **REQ1: (Author)** Ting Yi Xuan, **(Reviewers)** Jonathan Koh, Goh Kai Yuan
  - **REQ2: (Author)** Jonathan Koh, **(Reviewers)** Ting Yi Xuan, Goh Kai Yuan
  - **REQ3: (Author)** Goh Kai Yuan, **(Reviewers)** Jonathan Koh, Ting Yi Xuan
  - **REQ4: (Author)** Everybody, **(Reviewers)** Everybody
  - **REQ5: (Author)** Everybody, **(Reviewers)** Everybody

Signed by:

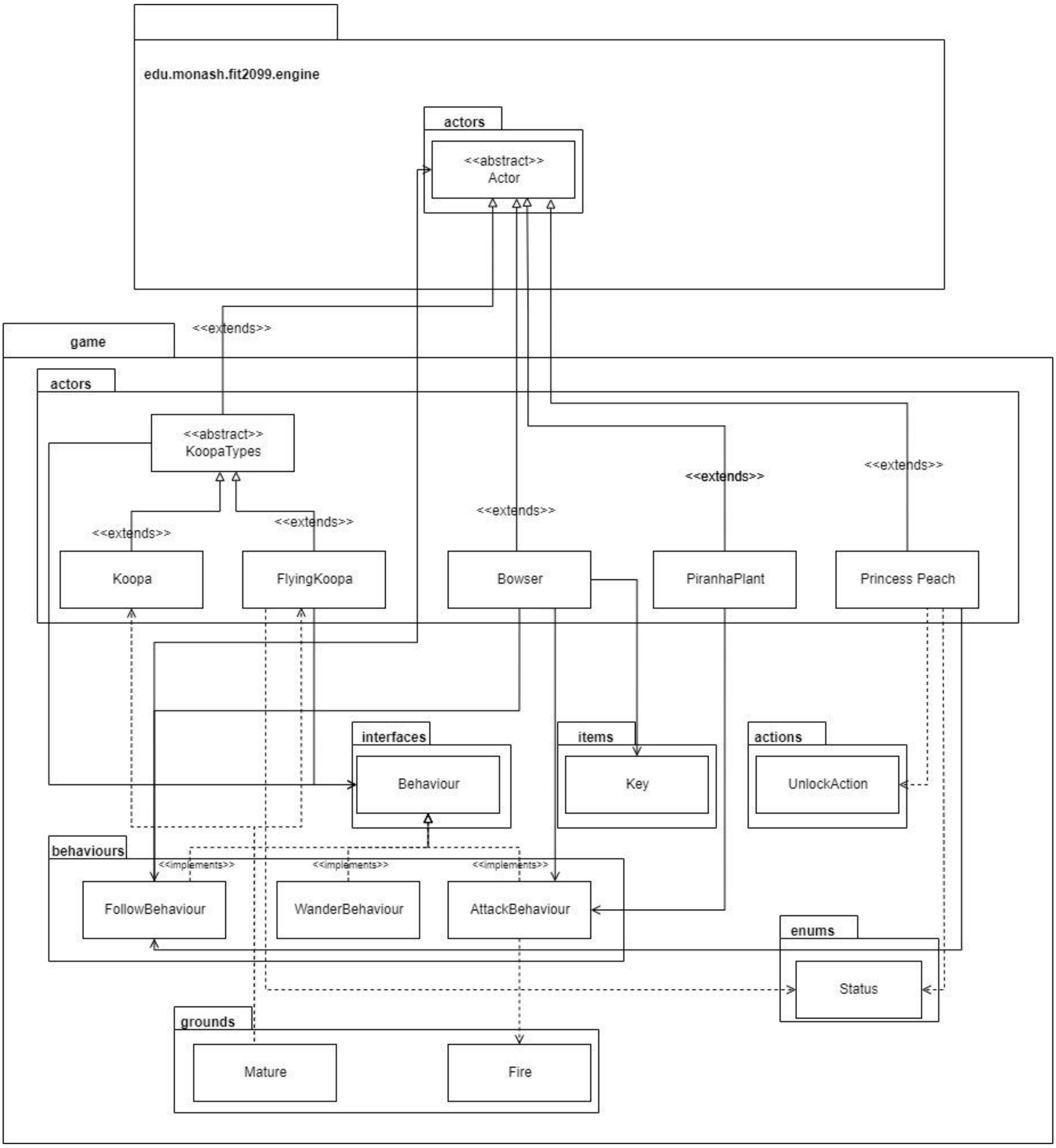
I accept this WBA – Goh Kai Yuan (16/05/2022)
I accept this WBA – Ting Yi Xuan (16/05/2022)
I accept this WBA – Jonathan Koh Tong (16/5/2022)

Class Diagram

REQ1

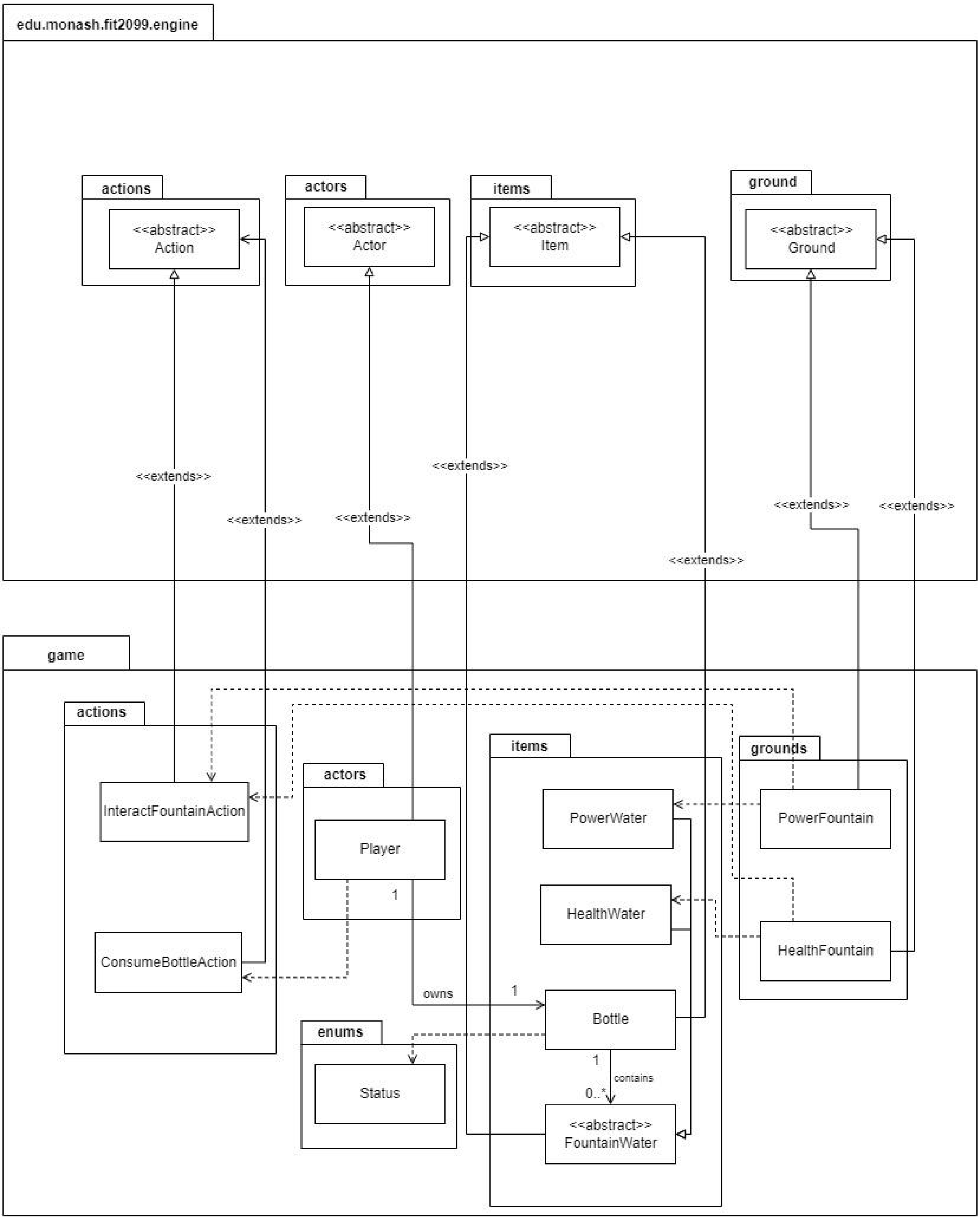


REQ2

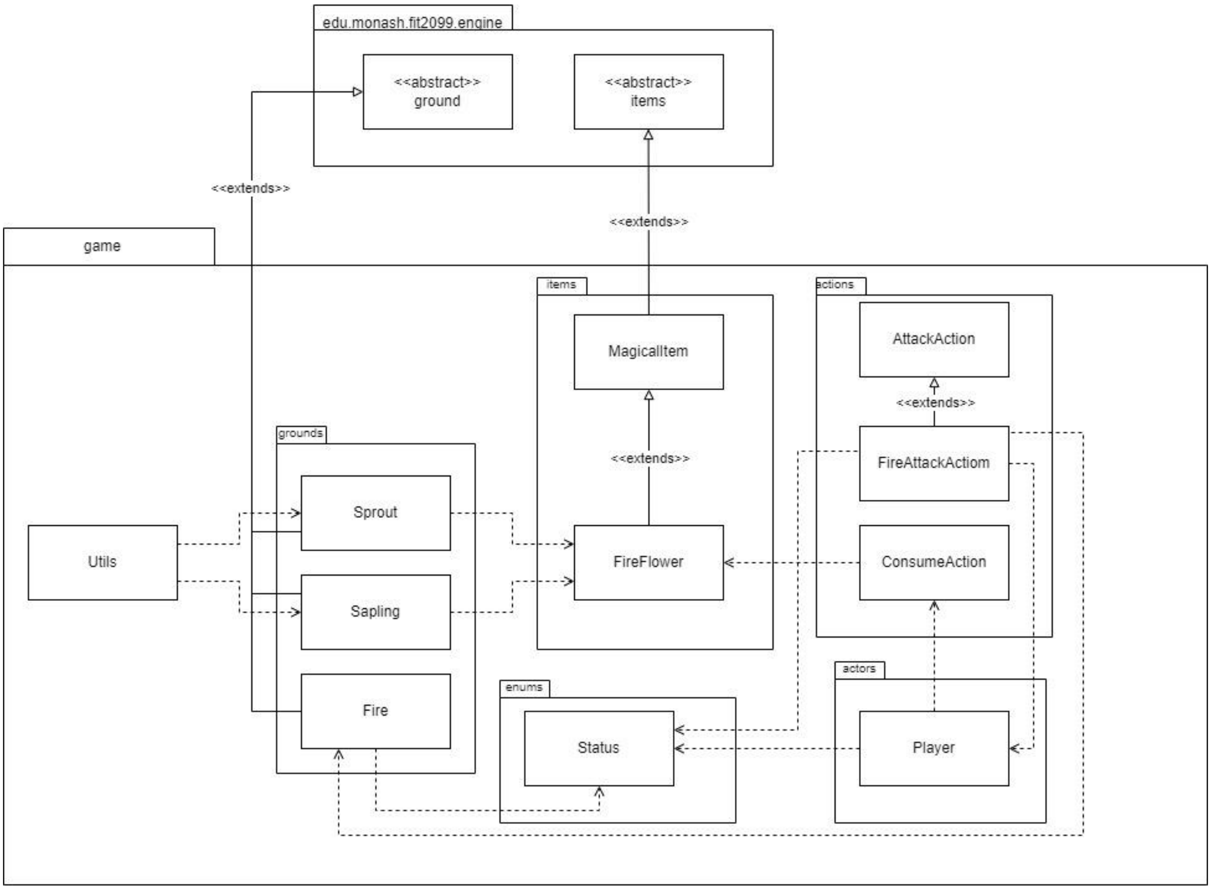


edu.monash.fit2099.engine
actors
<<abstract>>
Actor
game
<<extends>>
actors
<<abstract>>
KoopaTypes
<<extends>>
<<extends>>
<<extends>>
<<extends>>
<<extends>>
Koopa
FlyingKoopa
Bowser
PiranhaPlant
Princess Peach
interfaces
Behaviour
items
Key
actions
UnlockAction
behaviours
<<implements>>
<<implements>>
<<implements>>
FollowBehaviour
WanderBehaviour
AttackBehaviour
enums
Status
grounds
Mature
Fire

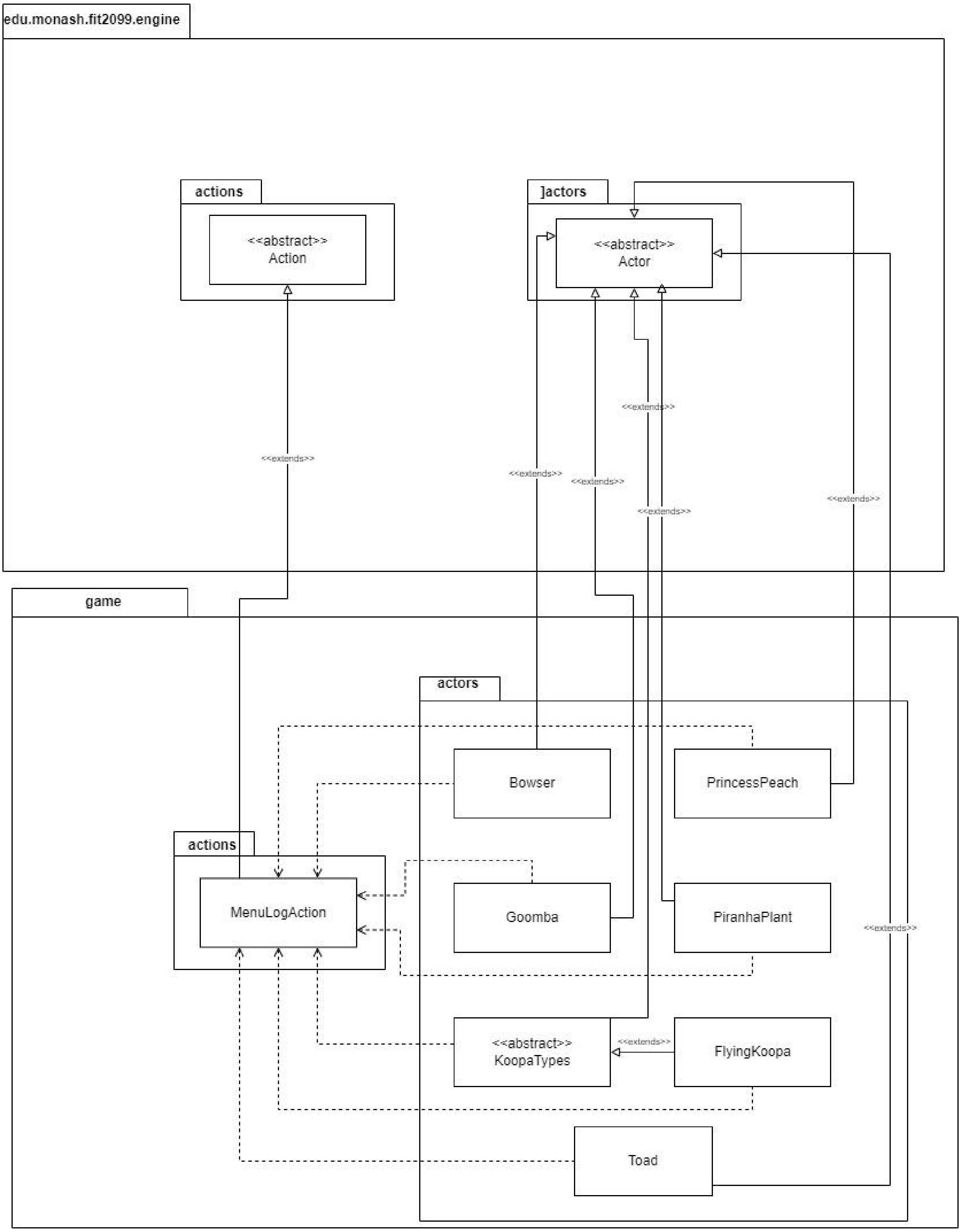
REQ3


edu.monash.fit2099.engine
actions
<<abstract>>
Action
actors
<<abstract>>
Actor
items
<<abstract>>
Item
ground
<<abstract>>
Ground
<<extends>>
<<extends>>
<<extends>>
<<extends>>
<<extends>>
<<extends>>
<<extends>>
game
actions
InteractFountainAction
ConsumeBottleAction
actors
Player
1
owns
1
items
PowerWater
HealthWater
Bottle
1
contains
0..*
<<abstract>>
FountainWater
grounds
PowerFountain
HealthFountain
enums
Status

REQ4


edu.monash.fit2099.engine
<<abstract>>
ground
<<abstract>>
items
<<extends>>
<<extends>>
game
items
actions
AttackAction
MagicalItem
<<extends>>
grounds
<<extends>>
FireAttackActiom
Sprout
Utils
FireFlower
ConsumeAction
Sapling
actors
enums
Fire
Status
Player

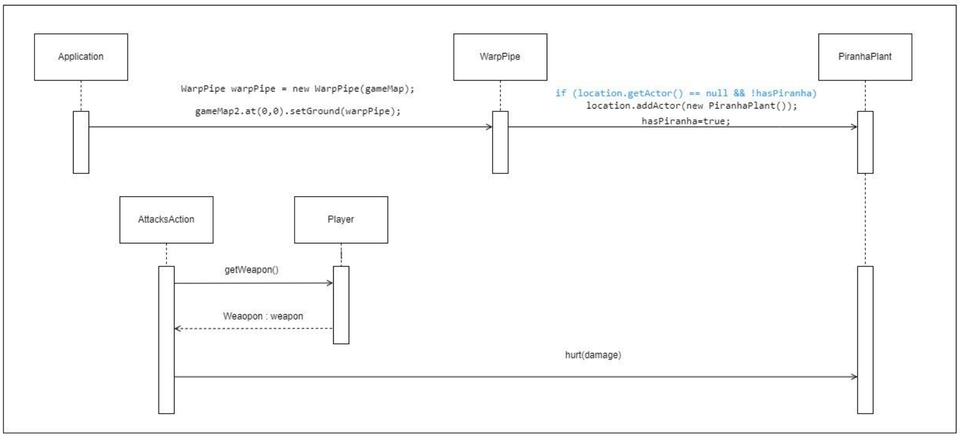
REQ5


edu.monash.fit2099.engine
actions
<<abstract>>
Action
]actors
<<abstract>>
Actor
<<extends>>
<<extends>>
<<extends>>
<<extends>>
<<extends>>
<<extends>>
<<extends>>
game
actors
Bowser
PrincessPeach
actions
MenuLogAction
Goomba
PiranhaPlant
<<extends>>
<<abstract>>
KoopaTypes
<<extends>>
FlyingKoopa
Toad

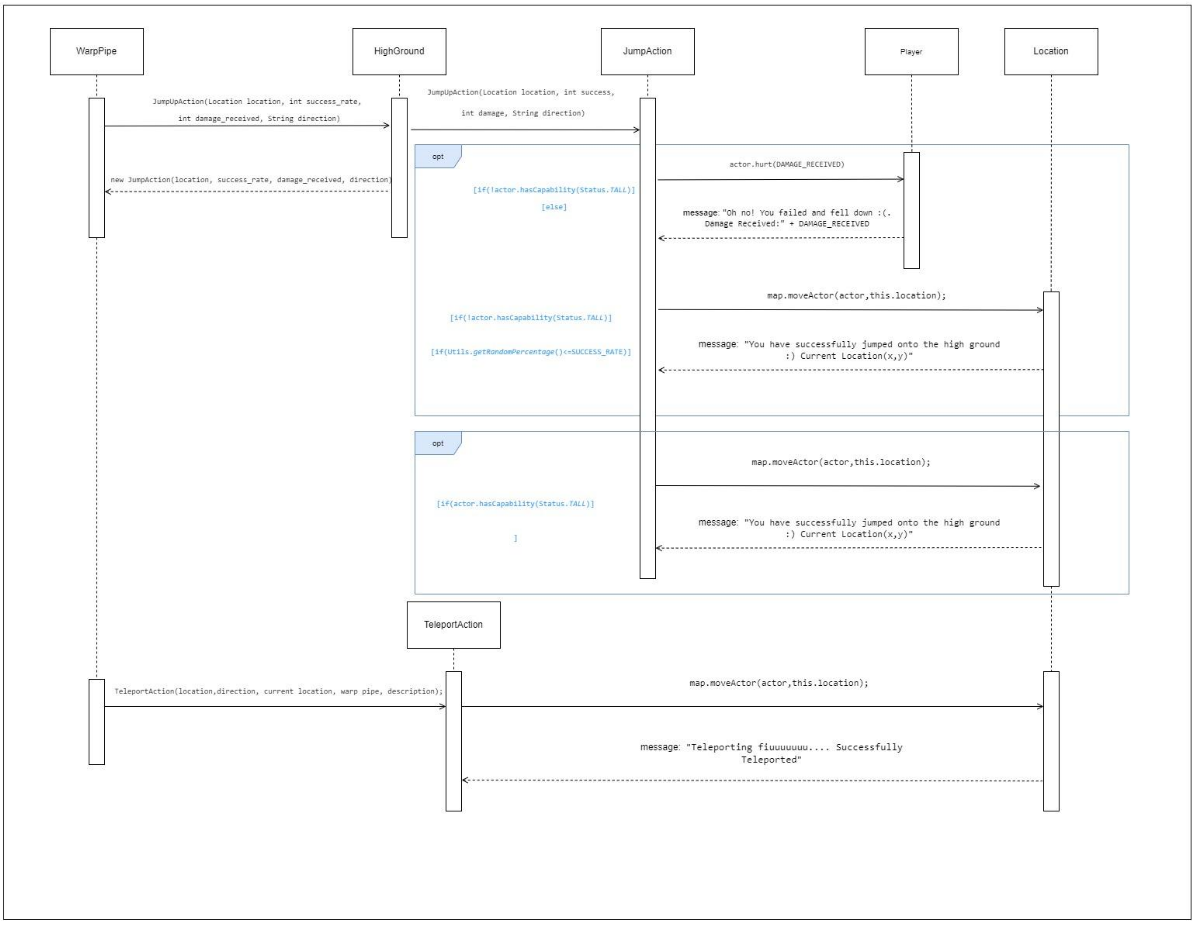
Interaction Diagram

REQ1

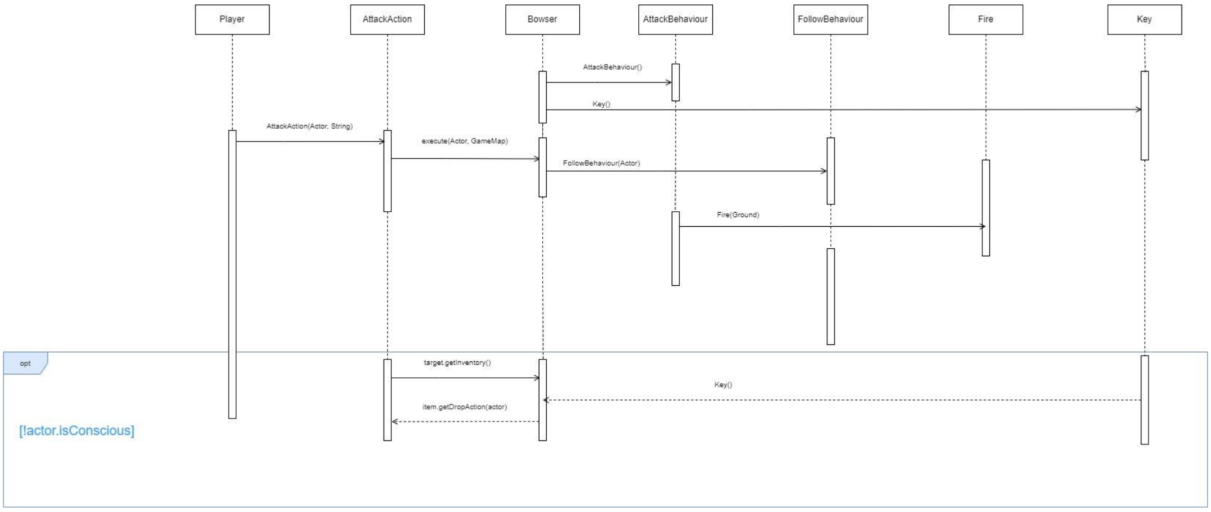
Interaction Diagram for WarpPipe spawns PiranhaPlant and Player attacks PiranhaPlant


Application
WarpPipe
PiranhaPlant
WarpPipe warpPipe = new WarpPipe(gameMap);
gameMap2.at(0,0).setGround(warpPipe);
if (location.getActor() == null && !hasPiranha)
location.addActor(new PiranhaPlant());
hasPiranha=true;
AttacksAction
Player
getWeapon()
Weaopon : weapon
hurt(damage)

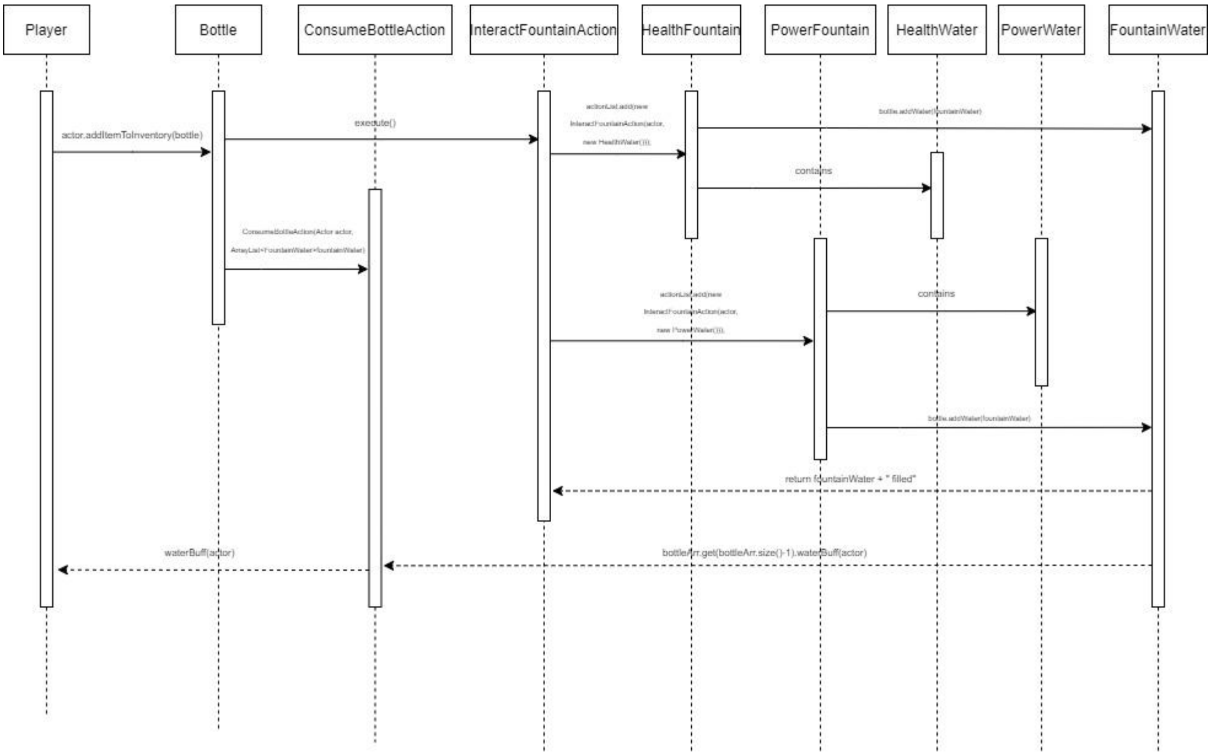
Interaction Diagram for Player jumps to WarpPipe and Player teleports to another map by using the WarpPipe


WarpPipe
HighGround
JumpAction
Player
Location
JumpUpAction(Location location, int success_rate,
int damage_received, String direction)
JumpUpAction(Location location, int success,
int damage, String direction)
new JumpAction(location, success_rate, damage_received, direction)
opt
[if(!actor.hasCapability(Status.TALL)]
[else]
actor.hurt(DAMAGE_RECEIVED)
message: "Oh no! You failed and fell down :(.
Damage Received:" + DAMAGE_RECEIVED
map.moveActor(actor,this.location);
[if(!actor.hasCapability(Status.TALL)]
[if(Utils.getRandomPercentage()<=SUCCESS_RATE)]
message: "You have successfully jumped onto the high ground
:) Current Location(x,y)"
opt
map.moveActor(actor,this.location);
[if(actor.hasCapability(Status.TALL)]
]
message: "You have successfully jumped onto the high ground
:) Current Location(x,y)"
TeleportAction
TeleportAction(location,direction, current location, warp pipe, description);
map.moveActor(actor,this.location);
message: "Teleporting fiuuuuuuu.... Successfully
Teleported"

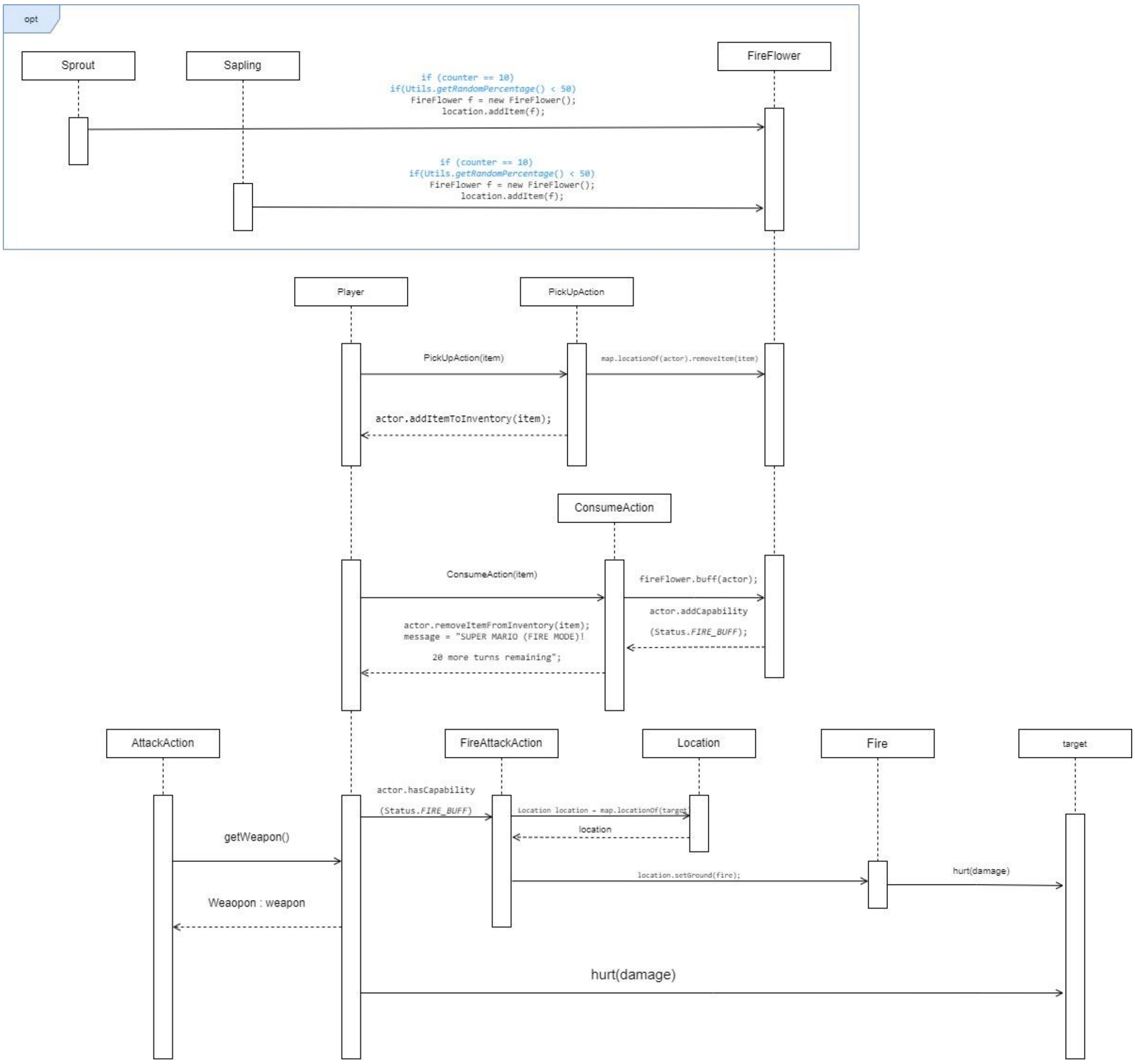
REQ2


Player
AttackAction
Bowser
AttackBehaviour
FollowBehaviour
Fire
Key
AttackBehaviour()
Key()
AttackAction(Actor, String)
execute(Actor, GameMap)
FollowBehaviour(Actor)
Fire(Ground)
opt
target.getInventory()
Key()
item.getDropAction(actor)
[!actor.isConscious]

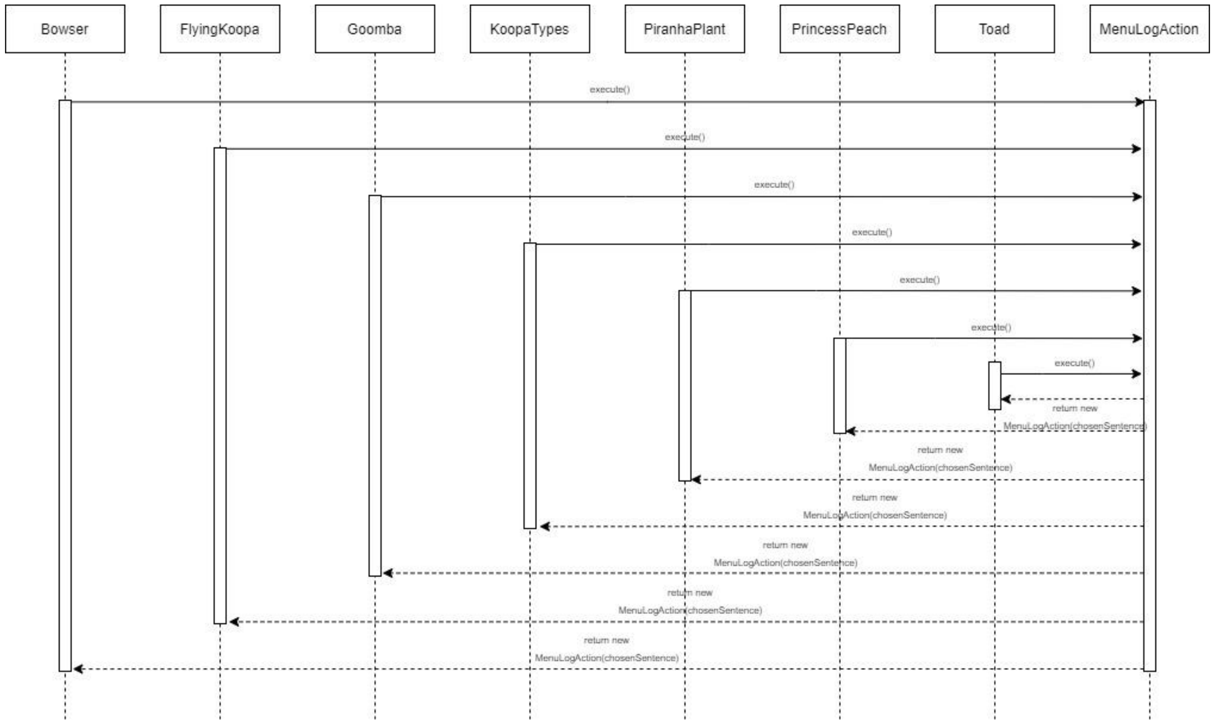
REQ3


Player
Bottle
ConsumeBottleAction
InteractFountainAction
HealthFountain
PowerFountain
HealthWater
PowerWater
FountainWater
actor.addItemToInventory(bottle)
execute()
actionList.add(new
InteractFountainAction(actor,
new HealthWater()));
bottle.addWater(fountainWater)
contains
ConsumeBottleAction(Actor actor,
ArrayList<FountainWater>fountainWater)
actionList.add(new
InteractFountainAction(actor,
new PowerWater()));
contains
bottle.addWater(fountainWater)
return fountainWater + " filled"
waterBuff(actor)
bottleArr.get(bottleArr.size()-1).waterBuff(actor)

REQ4


opt
Sprout
Sapling
FireFlower
if (counter == 10)
if(Utils.getRandomPercentage() < 50)
FireFlower f = new FireFlower();
location.addItem(f);
if (counter == 10)
if(Utils.getRandomPercentage() < 50)
FireFlower f = new FireFlower();
location.addItem(f);
Player
PickUpAction
PickUpAction(item)
map.locationOf(actor).removeItem(item)
actor.addItemToInventory(item);
ConsumeAction
ConsumeAction(item)
fireFlower.buff(actor);
actor.addCapability
(Status.FIRE_BUFF);
actor.removeItemFromInventory(item);
message = "SUPER MARIO (FIRE MODE)!
20 more turns remaining";
AttackAction
FireAttackAction
Location
Fire
target
actor.hasCapability
(Status.FIRE_BUFF)
Location location = map.locationOf(target)
location
getWeapon()
location.setGround(fire);
hurt(damage)
Weaopon : weapon
hurt(damage)

REQ5


Bowser
FlyingKoopa
Goomba
KoopaTypes
PiranhaPlant
PrincessPeach
Toad
MenuLogAction
execute()
execute()
execute()
execute()
execute()
execute()
execute()
return new MenuLogAction(chosenSentence)
return new MenuLogAction(chosenSentence)
return new MenuLogAction(chosenSentence)
return new MenuLogAction(chosenSentence)
return new MenuLogAction(chosenSentence)
return new MenuLogAction(chosenSentence)
return new MenuLogAction(chosenSentence)

Design Rationale

REQ1

Firstly, I created a whole new map(Lava Zone) in Application class with some Lavas in it. Next, I created a Lava class and I restricted other actors from walking on lava (except the player itself) by using Status.FLOOR_LAVA to check the actor's capability. Player will reduce 15hp every turn if he's standing on lava.

For the teleport part, I created a WarpPipe class and TeleportAction class. Warp Pipe is a high ground, we need to jump onto warp pipe so WarpPipe class is implemented HighGround interface and call getJumpAction in that object's class to override the allowableActions list in the same way we did previously without having to make any changes to the Jump Action class specifically. This demonstrates that the class follows the Open-Closed Principle (OCP), which states that software components should be extensible but not modifiable.

In addition, I created a new actor class called PiranhaPlant class. The warp pipe will spawn a piranha plant on it in the second turn. The player hasto kill the piranha plant so that he is able to jump onto the warp pipe because the warp pipe is blocked by a piranha plant. The warp pipe will only spawn the piranha plant once (just in the second turn).

Furthermore, I set some warp pipes in both of the maps with a given parameter "GameMap gameMap" in WarpPipe class's constructor. This is because we need to know the gameMap that the player will teleport to if he uses the warp pipe. For example, if the player uses the warp pipe in gameMap1, he will be teleported to the gameMap2 (Lava zone). If he uses the warp pipe in gameMap2, he will be teleported to the warp pipe in gameMap1 that he used to teleport to gameMap2.

Moreover, TeleportAction class is a moveActorAction class which will teleport the actor to another map. When the user clicks the teleport action's hotkey, it will teleport the actor to the warp pipe in another map. If the warp pipe has a piranha plant on it, it will remove the piranha plant instantly. As a result, the Single Responsibility Principle (SRP) is satisfied since the teleportAction class is one of the action classes and will be in charge of the game's teleport function.

REQ2

For the first map, there will be 2 new enemies which are PiranhaPlant and FlyingKoopa. Piranha Plant will extend from Actor abstract class since it has characteristics on an Actor. Piranha Plant will not wander nor follow, therefore it will only have AttackBehaviour, thus the instance of AttackBehaviour will be added. Since FlyingKoopa have very similar characteristics as Koopa, therefore a KoopaTypes abstract class is created. FlyingKoopa and Koopa will both extend from KoopaTypes abstract class. This adheres to the Open-closed Principle(OCP) as the definition of the principle is that a class should be open for extension and closed for modification. Therefore, instead of creating another FlyingKoopa by modifying the previous Koopa class, we can straight away inherit Koopa characteristics from KoopaTypes abstract class. The only difference between FlyingKoopa and Koopa is that FlyingKoopa has 150 health and it can fly through any terrain, therefore its capability will have IGNORE_TERRAIN status.

For the second map, there will be 1 new actor and 1 new enemy. The new enemy is Bowser. Bowser will drop a key when it is dead, which is required to save PrincessPeach, thus an instance of class Key will be added into Bowser's inventory. Bowser has similar characteristics as other enemies, but Bowser will not wander around and it will land fire when an attack is successful. Therefore, only AttackBehaviour and FollowBehaviour will be added to Bowser. Since Bowser will land fire if attack is successful, thus AttackBehaviour class is modified in a way that if Bowser attacks successfully, a new Fire object will be created on Mario's location. A counter in Fire will count the number of turns left for Fire to be removed from the map since Fire will last on the location for 3 turns. If a new Fire is landed on the same location where the previous Fire has not been removed, the new Fire will replace the previous Fire, thus counter starts from 0 again and Fire will last for 3 more turns.Fire will burn anyone that steps on it by decreasing 20 hit points every time an actor is still on it. This adheres to the Single Responsibility Principle(SRP) where a class should have only one reason to change, meaning that a class should have only one job. Fire class only has the responsibility to count the turns left and to burn anyone who steps on it, it is not its responsibility to land itself on the location. The new actor, PrincessPeach, will stand beside Bowser and not move until she is freed from her captivity. Thus, PrincessPeach's capability will have LOCKED status. Whenever Mario tries to unlock her, a message will pop out to indicate that a key is required to unlock her. When Mario has the key and chooses to unlock her, a message will pop out indicating that she's saved and FollowBehaviour will be added to PrincessPeach so that she will follow Mario. Unlock is made possible by the UnlockAction class which has a dependency relationship with PrincessPeach class. This also adheres to the Single Responsibility Principle(SRP) since UnlockAction class is solely responsible for unlocking actions and nothing else.

REQ3

In order for the player to drink the fountain water (Power Water or Health Water), the player must have a bottle. With that, I have implemented a Bottle class for player. On the first round, the player will obtain the bottle because it did not have a bottle and will not get a second bottle. This bottle can refill power water and health water and player is not allowed to drop this bottle.

I have implemented HealthFountain, H and PowerFountain, A in the map. Mario has to stand on top of these fountains to interact with them. With that, I return true for canActorEnter method so that any actors can go on top of it. With allowableActions method, fountain can only be interact with the player. Furthermore, I have created InteractFountainAction so that player is able to refill the fountain's water into their bottle.

The way how water can be filled into the player's bottle is that I have implemented an abstract class FountainWater and extend Item to hold the water. With that, i created another two classes, PowerWater and HealthWater that extends FountainWater class and each of the water buff my actor in different way in order to fulfills the Single Responsibility Principle. After that, I put the respective water into the bottle that the player is currently holding with FountainWater.

The ConsumeBottleAction will always go in a stack mode as water refilled last will be drank first by the player. Once they drink, they will different water buff from different fountain. Health Fountain that produces HealthWater will heal player for 50 points. Power Fountain that produces PowerWater will increase player by 15 base damage.

REQ4

For the first map, a new item called FireFlower is added. Since FireFlower has similar characteristics as PowerStar and SuperMushroom where it can be consumed, buff and debuff Player, FireFlower will then extend from the MagicalItem abstract class. This adheres to the Open-closed Principle(OCP) as the definition of the principle is that a class should be open for extension and closed for modification. Therefore, we do not need to manually create a new item called FireFlower by extending from the Item abstract class and creating its own methods all again, but straight away extending from the MagicalItem abstract class and inheriting MagicalItem class methods. Since there's a 50% chance of spawning FireFlower in every growing stage of the tree, Sprout, Sapling and Mature class will be modified such that it has a 50% chance of spawning FireFlower. Since the effect of FireFlower only lasts for 20 turns, therefore a counter is made in FireFlower class to count the number of turns left until effect wears off.

After consuming FireFlower, Mario will not only attack but also land fire on the enemies' location. After consuming FireFlower, a FIRE_BUFF status will be added to Mario's capability, thus Mario can conduct fire attacks. The FireAttackAction class will be created to cater for Mario's fire attack. Since fire attack is basically attack plus fire, therefore FireAttackAction class will extend from AttackAction class. This adheres to the Open-closed Principle(OCP) where FireAttackAction inherits from AttackAction class, without creating a whole new FireAttackAction class and implementing its method all over again. Similar to Bowser's attack, when a fire attack is successful, it will create a new Fire instance on the enemies' location. A counter in Fire will count the number of turns left for Fire to be removed from the map since Fire will last on the location for 3 turns. If a new Fire is landed on the same location where the previous Fire has not been removed, the new Fire will replace the previous Fire, thus counter starts from 0 again and Fire will last for 3 more turns.Fire will burn anyone that steps on it by decreasing 20 hit points every time an actor is still on it. This adheres to the Single Responsibility Principle(SRP) where a class should have only one reason to change, meaning that a class should have only one job. Fire class only has the responsibility to count the turns left and to burn anyone who steps on it, it is not its responsibility to land itself on the location.

REQ5

Bowser, Toad, Princess Peach, Goomba, all types of koopas, Flying Koopa and Piranha Plant have their own respective monologue. The idea of these is that, we implement a java array called “sentences” and store all of the pre-recorded monologue strings into this method. After that, since these actors will be outputting these monologues once per two turns, we implement a counter in playTurn method and override this method. We check if the turn is even and then output the monologue for that actor class.

To ensure that I have fulfill the Single Responsibility Principle, I have decided to create another action class called MenuLogAction to return the monologue in the menu. This action class get the monologue from each classes and output the monologue from each class on the menu.