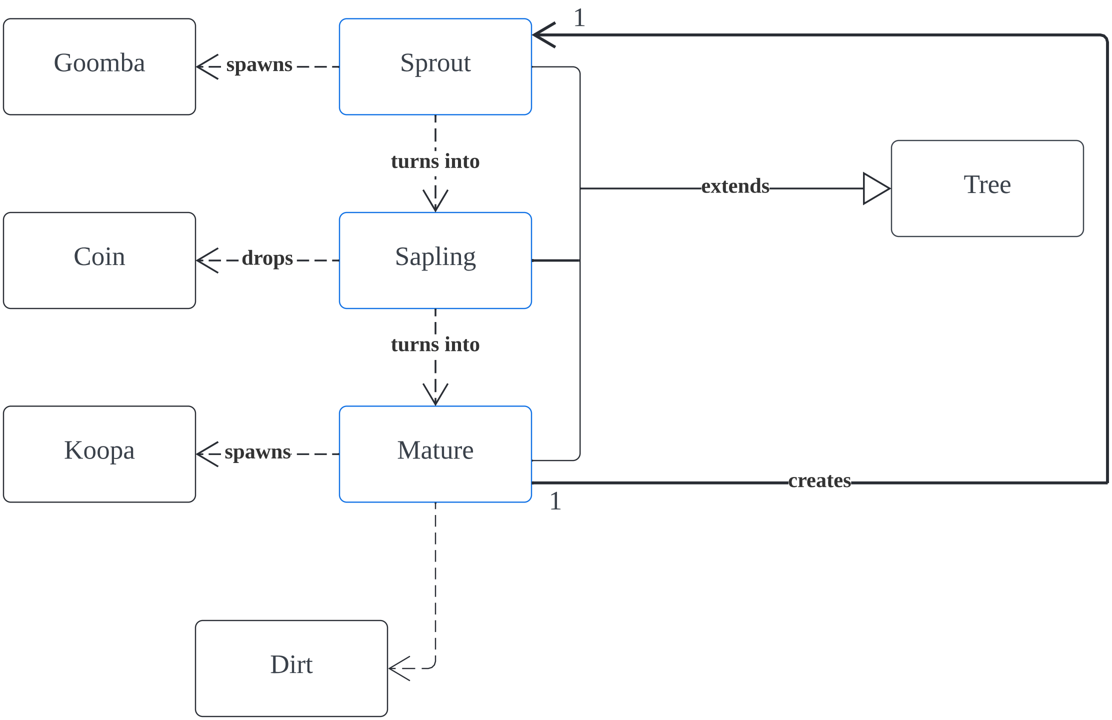
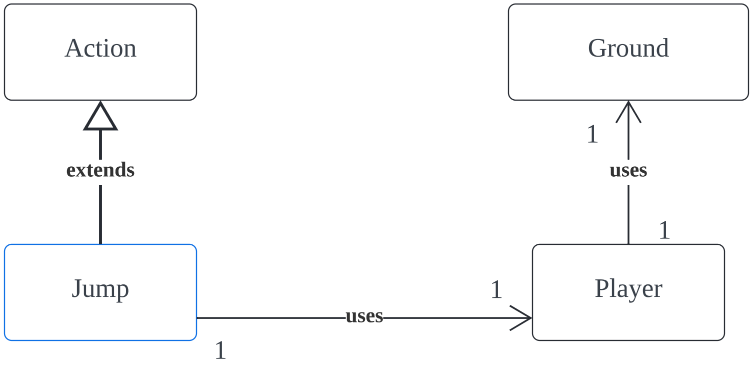
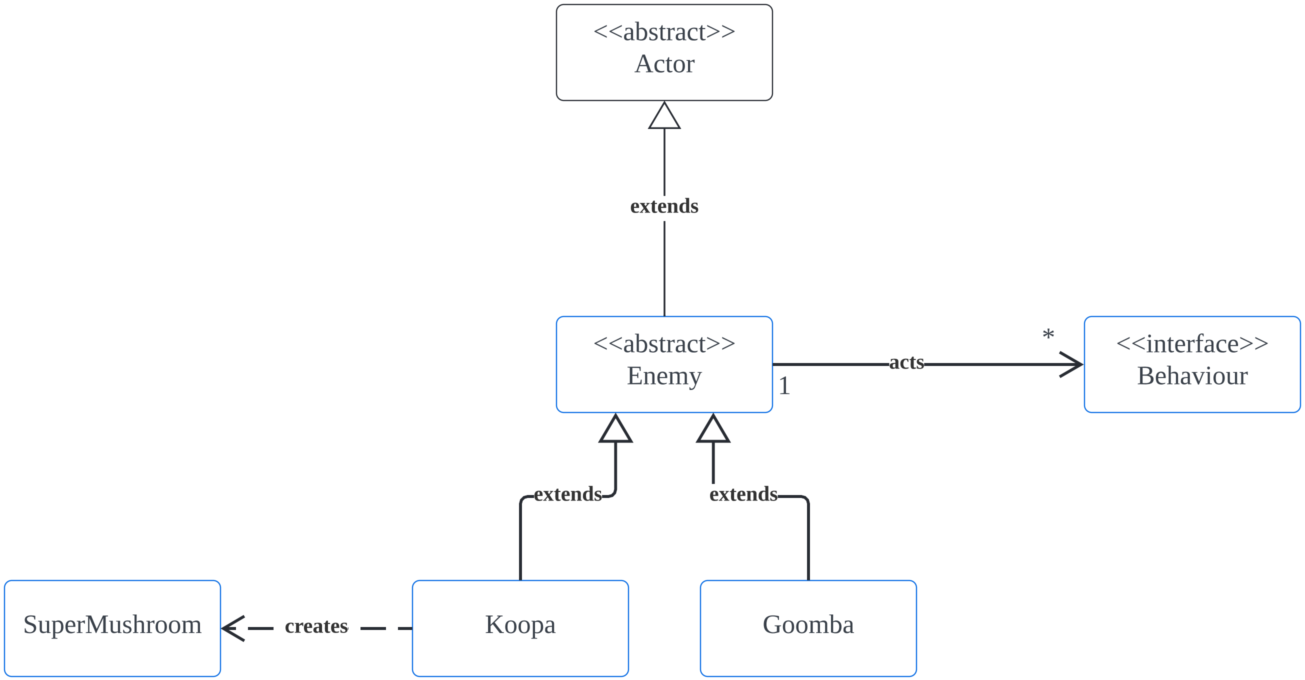
Req1


Goomba
spawns
Sprout
turns into
Coin
drops
Sapling
turns into
Koopa
spawns
Mature
Dirt
Tree
extends
1
creates
1

Req2

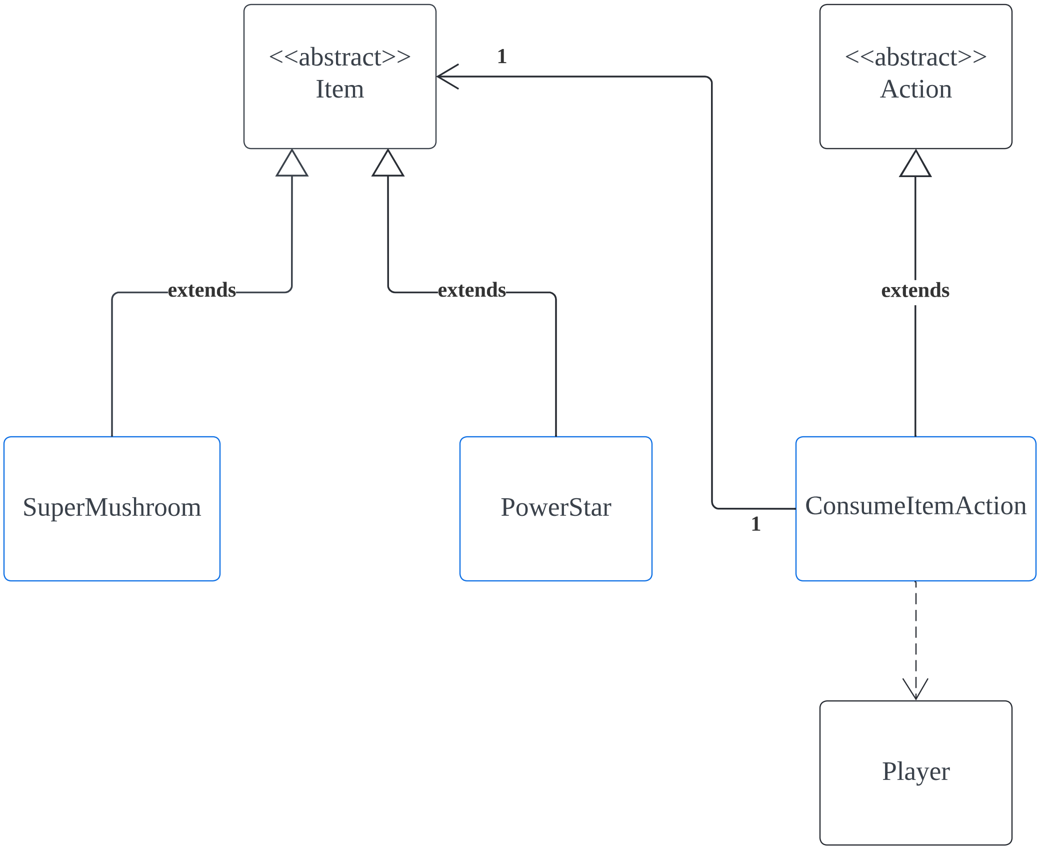


Req3



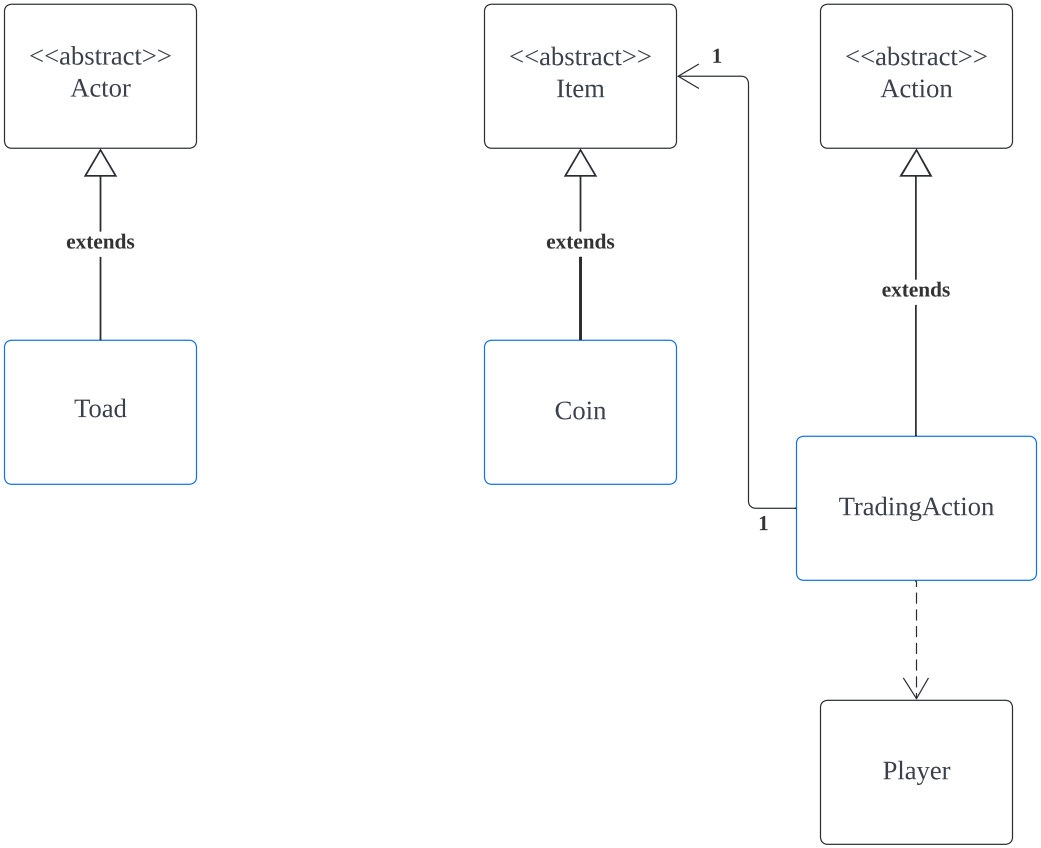
<<abstract>>
Actor
extends
<<abstract>>
Enemy
acts
1
*
<<interface>>
Behaviour
extends
extends
SuperMushroom
creates
Koopa
Goomba

Req4



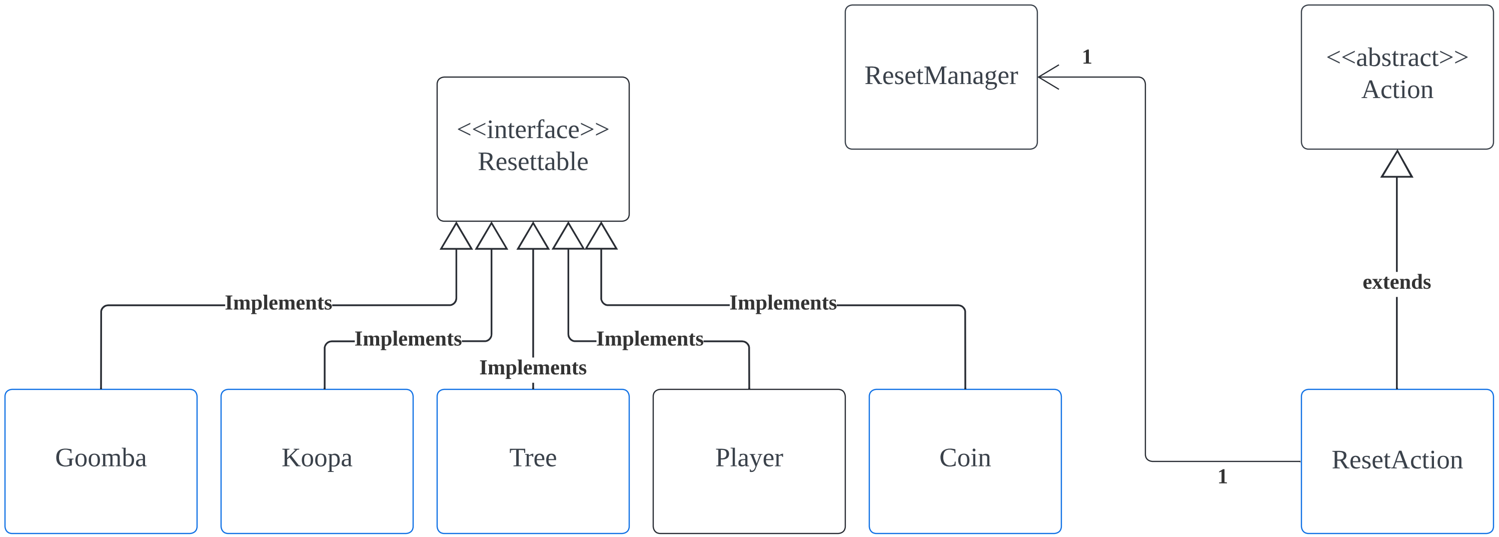
<<abstract>>
Item
<<abstract>>
Action
1
extends
extends
extends
SuperMushroom
PowerStar
ConsumeItemAction
1
Player

Req5



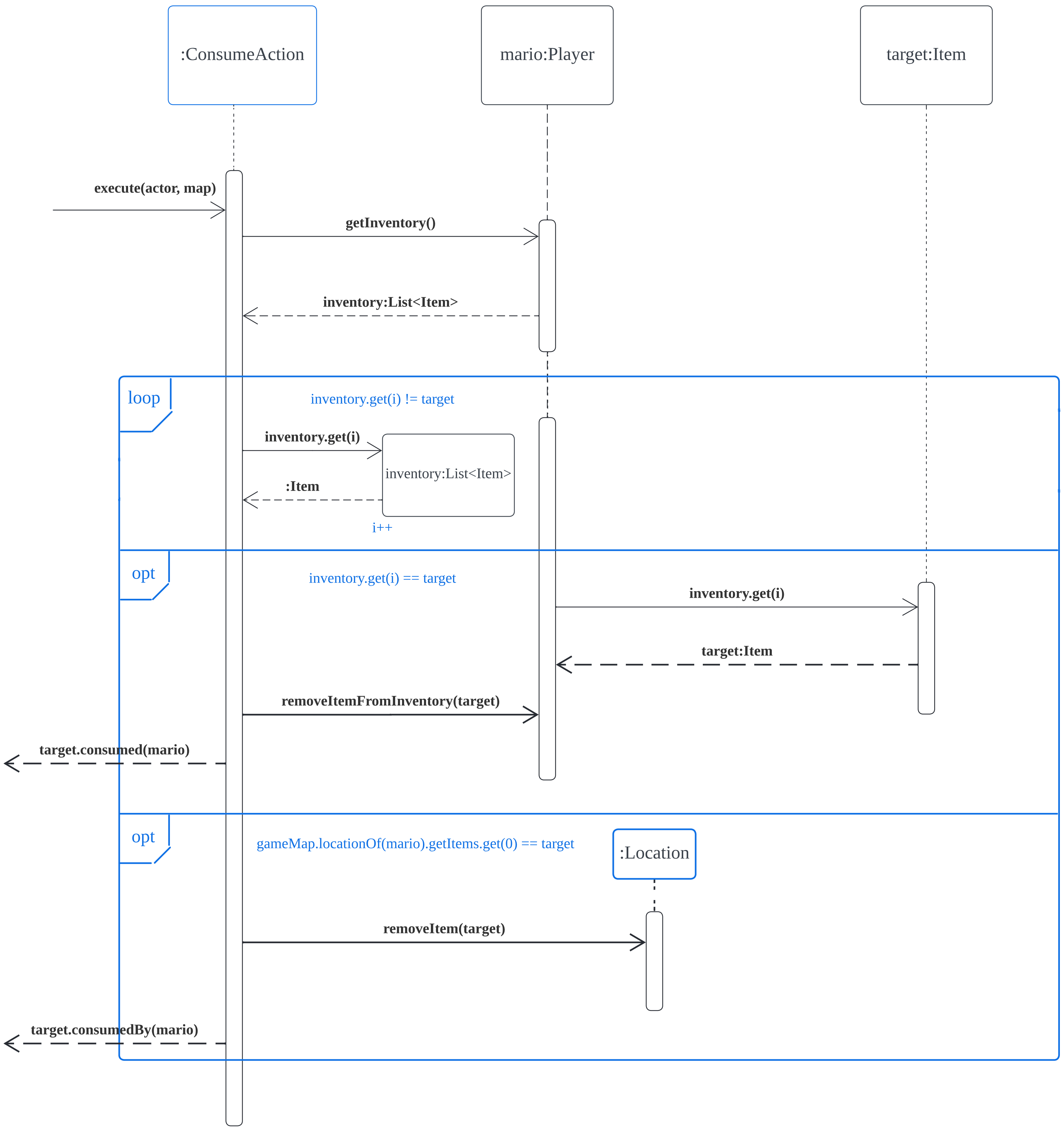
<<abstract>>
Actor
<<abstract>>
Item
<<abstract>>
Action
1
extends
extends
extends
Toad
Coin
TradingAction
1
Player

Req7

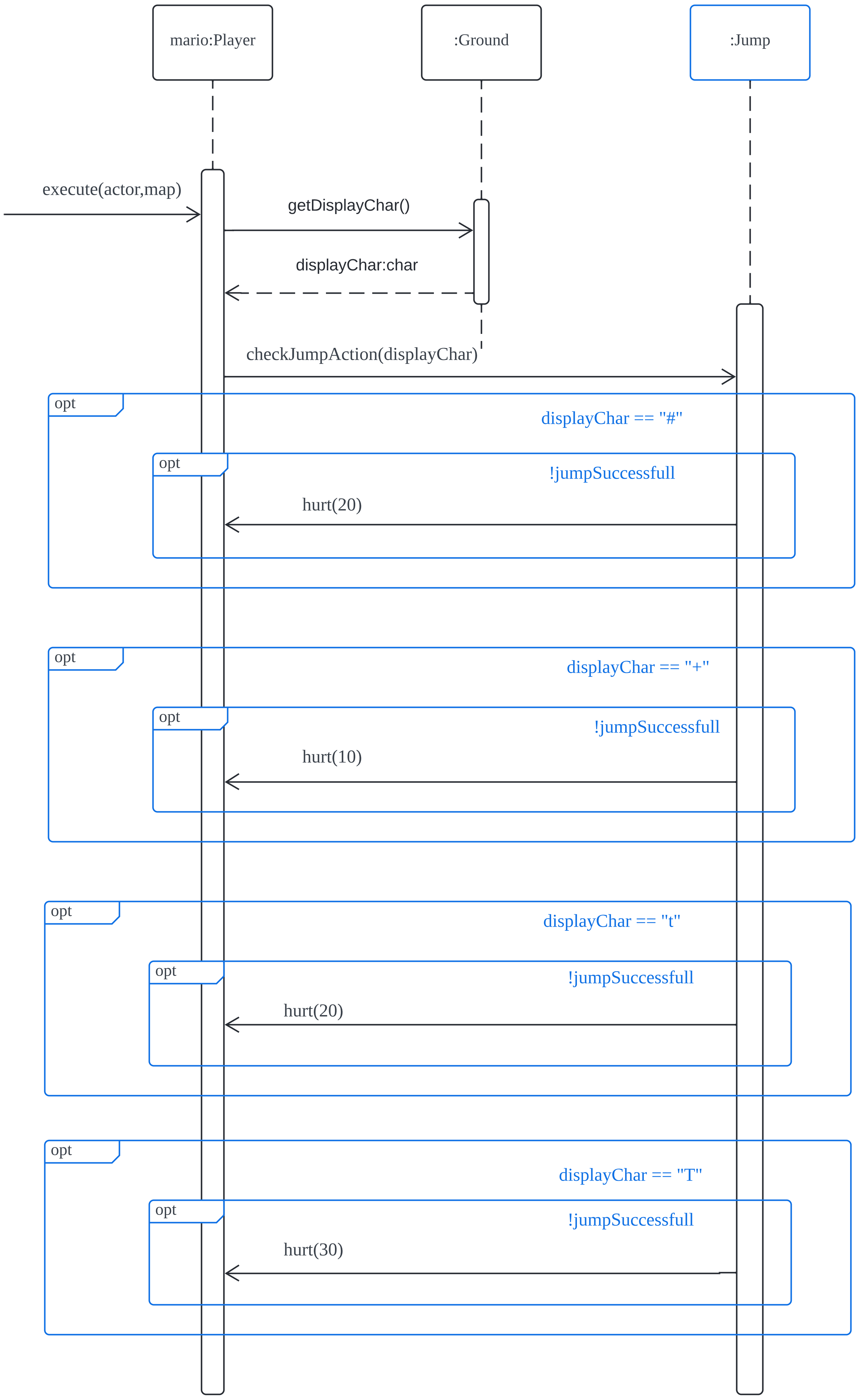


<<interface>> Resettable
ResetManager
<<abstract>> Action
1
1
extends
Implements
Implements
Implements
Implements
Implements
Goomba
Koopa
Tree
Player
Coin
ResetAction

Consume - Interaction Diagram



Jump - Interaction Diagram



FIT2009 Assignment

Design Rationale

Lab 11 Team 4

Teammate:
Subhan Saadat Khan - 32268513
Mingze Zheng - 32433786

REQ1

As mentioned in the requirement the Tree has 3 stages Sprout, Sapling and Mature. This indicates that these stages will act as child/base classes where parent is Tree class, following this I have created 3 new classes denoting each stage of the tree. These new classes inherit the Tree class as the fundamental behaviour of a tree remains the same at any stage.
Furthermore, for the new class Sprout I have given two dependencies 1) with Goomba as stated in the requirement Sprout has a chance to spawn Goomba 2) with Sapling as a Sprout eventually turns into sapling after 10 turns.
For the new class Sapling I have given 2 dependencies 1) with Coin as stated in the requirement that sapling has a chance to drop a coin 2) with Mature as a Sprout eventually turns into mature after 10 turns.
For the new class Mature I have given 2 dependencies 1) with Koopa as stated in the requirement that Mature has a chance to spawn Koompa 2) with Dirt as a Mature has a chance to die and become Dirt. Furthermore there is an association between Mature and Sprout as Mature is responsible for creating new Sprout.

REQ2

In order to execute the jump feature I created a new class Jump. Jump will be responsible for handling the jump mechanism for the player. Moreover the Jump class will have methods handling the success and failure consequences based on the success rate of jump on every type of high ground, for example as mentioned in the requirement the player will receive damage if the jump is unsuccessful. Keeping this in mind I have given an association with the Player because the success or failure of jumps will directly affect the Player. To instantiate the Jump class we need to know what grounds are next to the player as jumps can only take place when there is a high ground. To achieve this I have given an association from Player to Ground, so that the player could find out the types of grounds next to him and use this information to find if he is eligible to jump by calling the Jump class, due to this reason Player has a dependency with Jump class.

REQ3

In this requirement we were supposed to implement two enemies Goomba and Koopa. Since both of these are enemies having moreover the same fundamental attributes of an enemy, thus I created a new Abstract class called Enemy. The Enemy class will serve as a base class for common attributes in both enemies. Following this Goomba and Koopa will inherit from Enemy class. Since we know an enemy is also an actor thus the Abstract class Enemy will inherit the Abstract class Actor. Furthermore we know that every enemy has certain behaviour thus, Enemy class has an association with Interface Behaviour. Lastly, as mentioned in the requirement, destroying koopas shell will drop a super mushroom. Following this I have given a dependency of Koopa with the New SuperMushroom class.

REQ4

For the design of magical items, from the definition and description of super mushroom and power star, it is obvious that we need to create two new classes that inherit the Item class. Therefore, we create the two classes SuperMushroom and PowerStar and make them inherit class Item. From class Item, we can see that the attributes and methods in the Item class contain the common attributes and methods that SuperMushroom and PowerStar need. From the functionalities of SuperMush and PowerStar, we may need to add new statuses to Enum

Status to make the implementation easier. To consume magical items, we need a new action. Obviously, the new action is a subclass of the abstract class Action. We create a new class called ConsumeAction which inherits Action. In ConsumeAction, we need to use the item that will be consumed and the consumer of the item. As some other actions do, I choose to declare Item as an instance variable in ConsumeAction. And the consumer will be a parameter of the overridden method execute().

REQ5

As we can see from the description, a coin is also a type of item. It has the attributes that an item should have. So we make the new class Coin a subclass of Item. “Toad is a friendly actor”, Toad should inherit the Actor class from the description. Purchases from the toad, as we can imagine, is also an action. So we need to create a new class called TradingAction (it is also a subclass of action since trading is a type of action). As other actions do, we will make Item an instance variable of TradingAction, and buyer (Player) will be a parameter of the method execute(). So the relationship between TradingAction and Item is association and the relationship between TradingAction and Player is dependency.

REQ7

If we want to reset the game, from the base code and the description, the best design is to make the classes that need to be reset implement the interface Resettable. Then we will need to finish the methods in class ResetManager. To use ResetManager, we need to make reset an action. Therefore, we will create a new action called ResetAction. In the ResetAction class, there will be an instance of ResetManager to control the reset.