Work Breakdown Agreement for FIT2099 Assignment 2

Team 3 Lab 10

1. Mikael Andrew Susanto (ID: 32332270)
2. Looi Teck San (ID: 32439180)
3. Tan Kah Hong (ID: 31110126)

We will separate the work into three parts:

1. All codes for REQ 1 & 2, codes for REQ 4
2. All codes for REQ 3 & 7, codes for REQ 4
3. All codes for REQ 5 & 6, codes for REQ 4

Mikael will be responsible for Part 1 and its design rationale, Reviewer: All of the members, Completion: Wednesday, 30 April 2022

Teck San will be responsible for Part 2 and its design rationale, Reviewer: All of the members, Completion: Wednesday, 30 April 2022

Kah Hong will be responsible for Part 3 and its design rationale, Reviewer: All of the members, Completion: Wednesday, 30 April 2022

Signed by
I accept this WBA. - Looi Teck San
I accept this WBA. - Tan Kah Hong
I accept this WBA. - Mikael Andrew Susanto

Design Rationale

Requirement 1

Changes:

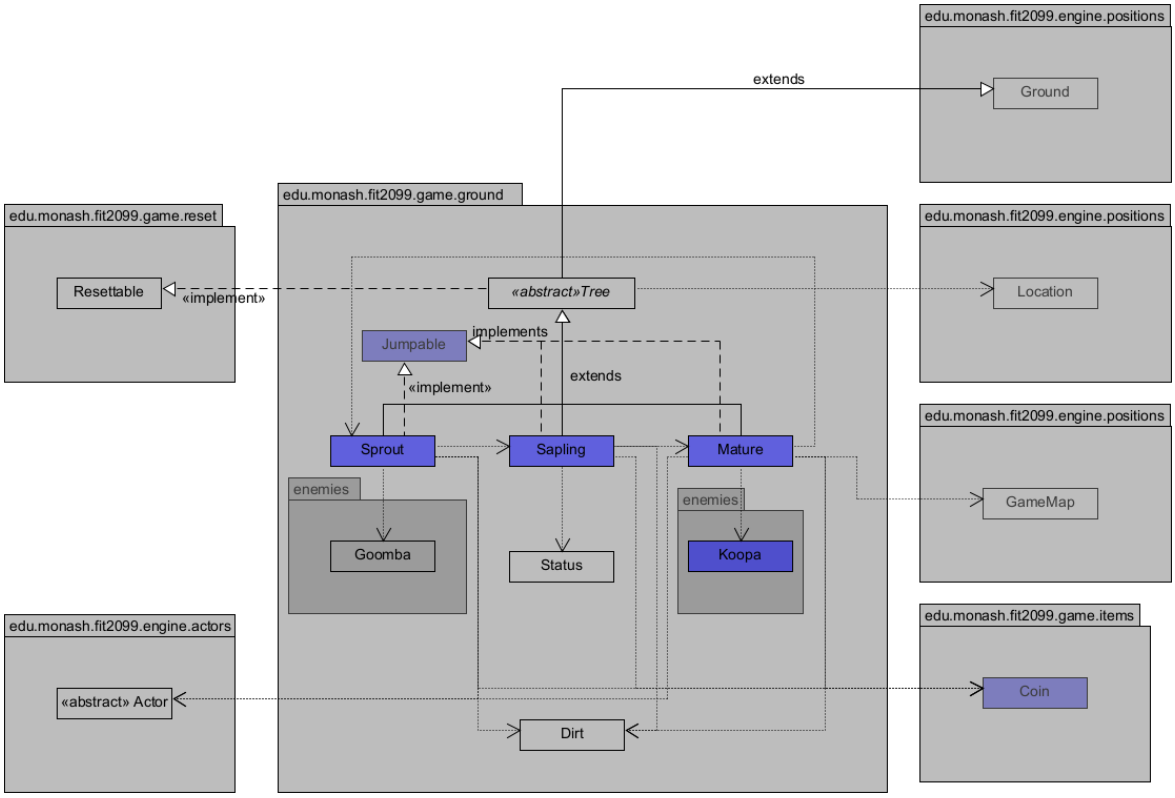
- All the class that inherits from the Tree and Tree itself implements Resettable.
- Sprout, Sapling and Mature implements Jumpable since it is considered high ground.
- Wall also implements Jumpable.
- Sprout doesn't associate to Goomba rather than depends since it doesn't have any attribute of Goomba.
- Sapling doesn't associate to Coin, but depends on Coin since it only creating Coin object inside of its method and rather than depends on Sprout, should be Sprout that depends on Sapling.
- Mature does not associated to Sprout and Koopa, but depends on them.

Principles:

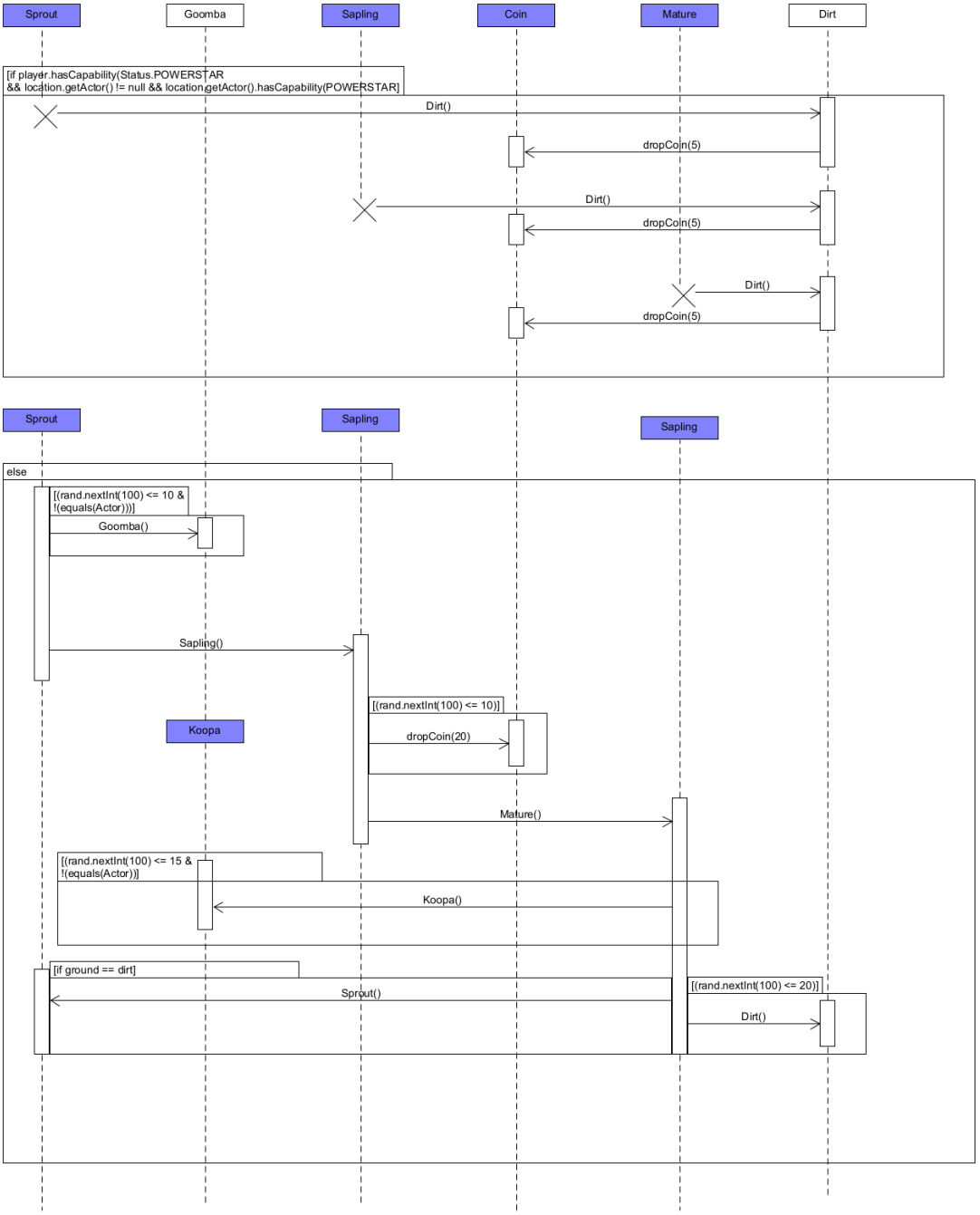
LSP:

- Substituting Sprout, Sapling and Mature with the parent class Tree will not result in any unexpected behaviour.

Class Diagram



Sequence Diagram



Design Rationale

Requirement 2

Changes:

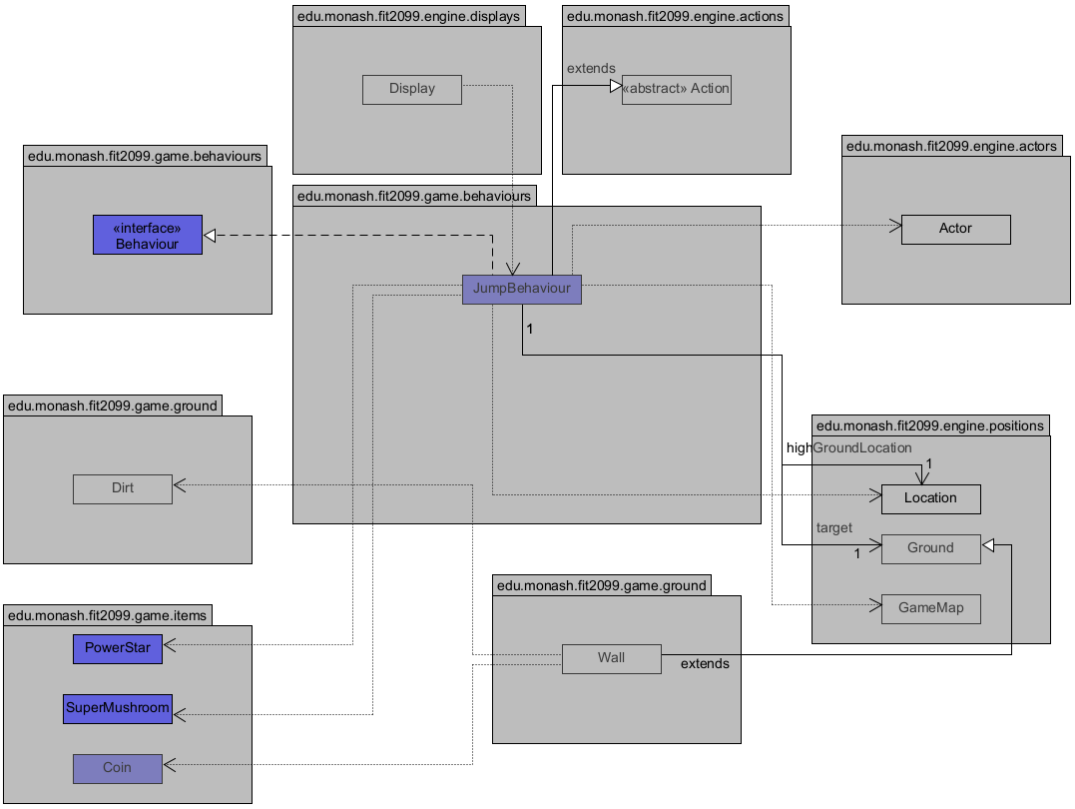
- JumpBehaviour will extend from Action class rather than Actor.
- JumpBehaviour is associated to Ground, Location and Jumpable since it is needed to keep track of the Ground target and its Locaiton which is a high ground that the player wants to jump onto and Jumpable to get the high ground attributes
- Using the Jumpable Interface adheres to OCP..
- Rather than MagicalItem, it depends on PowerStar and SuperMushroom.
- it will also depend on coin since if the Player has the capability of PowerStar then the Player can directly walk to the high ground. It will destroy it to a Dirt and drop a Coin with a value of 5.

Principles:

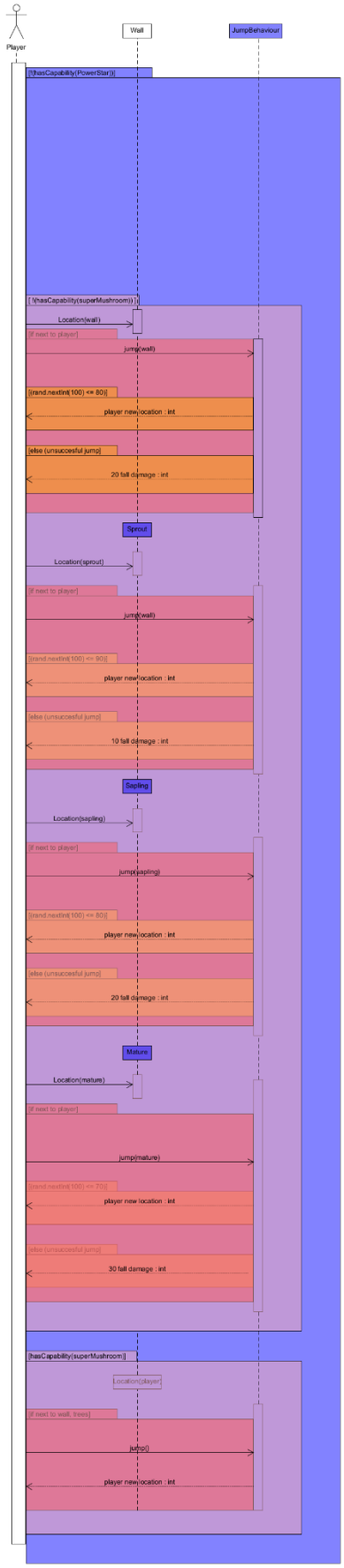
OCP:

- Interface Jumpable implements getSuccessRate(), getFallDamage(), getName() methods for each Jumpable high grounds. Each high ground only needs to be registered to become a Jumpable ground with no changes in the JumpBehaviour class. Hence, there can be more Jumpable ground with no modifications in JumpBehaviour.

Clasas Diagram



Sequence Diagram



Design Rationale

Requirement 3

Changes

Enemies abstract class

- Created Enemies abstract class and Koopa and Goomba will inherit the class so that during attack action we can avoid using instance of which violate SOLID principles.

AttackBehaviour

- Added the class to manipulate and let the enemies to attack player automatically once player entered their surroundings

Shell

- Added the class to create shell object

BreakShellAction

- Added the class to break shell

SOLID principles

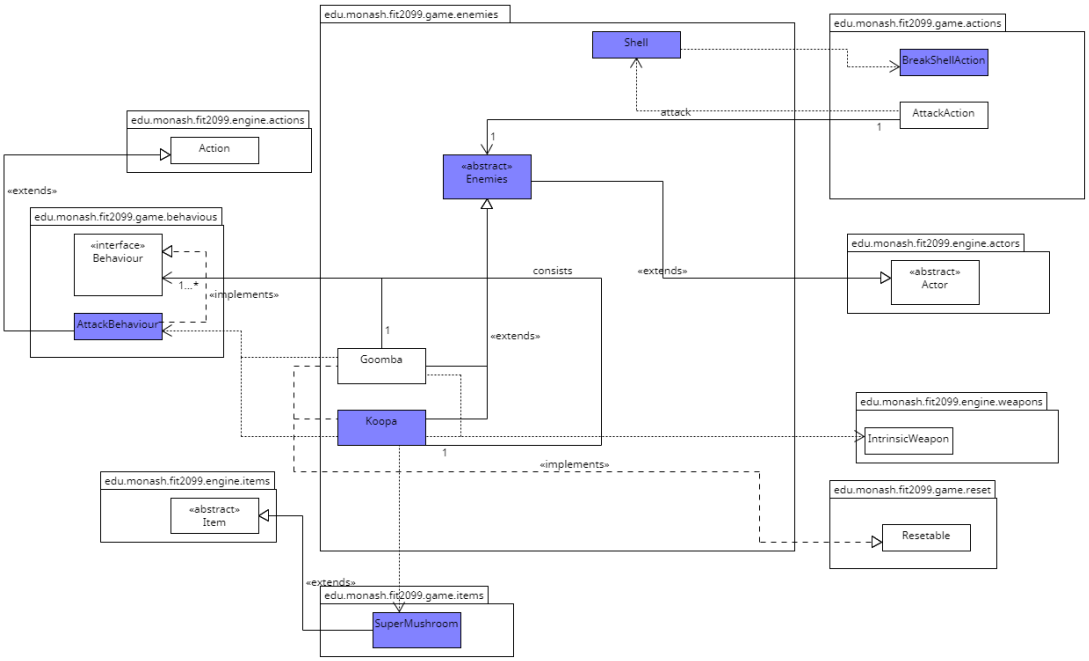
Single Responsibility Principle

- **Koopa,Goomba, AttackBehaviour**
    - The Koopa and Goomba class in only for creating Koopa and Goomba only, while to let them to attack the player AttackBehaviour will handle it.
- **Shell,BreakShellAction**
    - The action to break shell and shell object are different class, Shell class will be used only for creating the shell object to break the shell, I created another class BreakShellAction to do it.

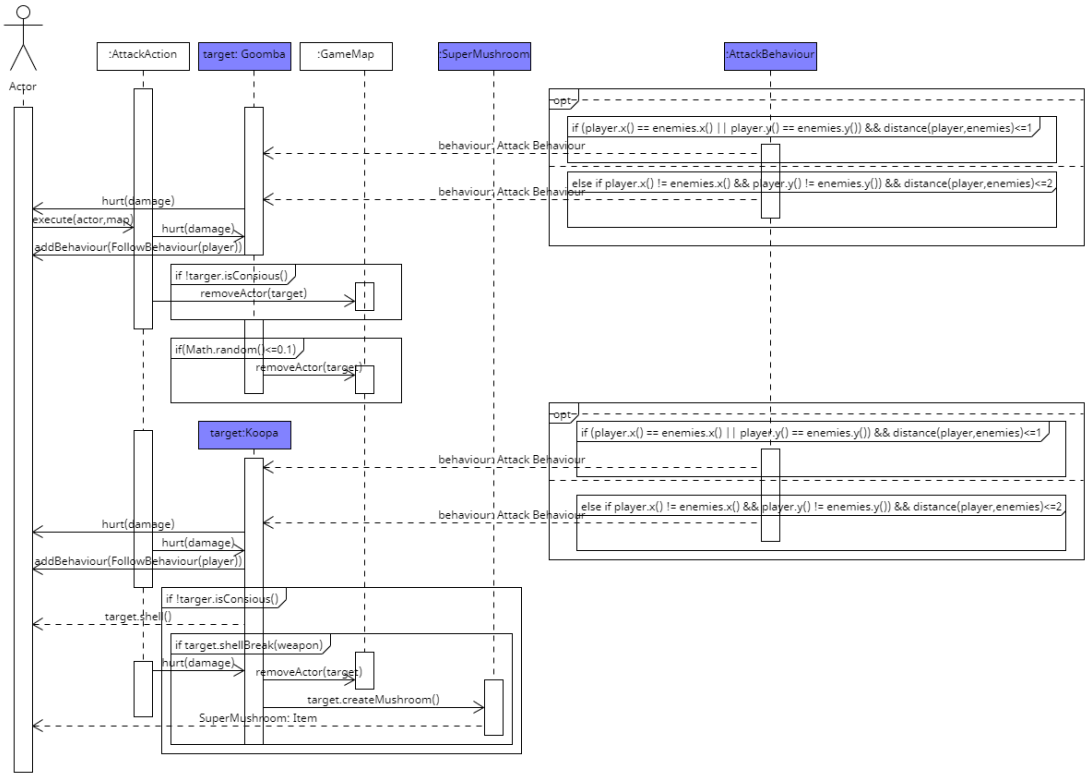
Open-closed principle

- **Enemies abstract class**
    - Created so that if in the future we would like to create more enemy we would not need to modify the class besides it also avoid us from using instanceof in AttackAction class

Class Diagram



Sequence Diagram



Design Rationale

Requirement 4

Changes:

- There is no WalkBehaviour, instead the functionality already provided is used. There is no need for WalkBehaviour.
- JumpBehaviour also inherits Action, because it is an Action that the Player can do. Also, it will use MoveActorAction to move the Player to the targeted high ground.
- ConsumeSuperMushroomAction and ConsumePowerStarAction are new Action for handling the consumption of the items.
- The adding of the effects to Player are now done by two methods: addPowerStarEffect() and addSuperMushroomEffect().
- Coin has no more relationship with PowerStar, to better follow separation of concerns. Current implementation for consuming items and Player gaining the effects:
- Have a Consume<Item>Action for each consumable item.
- This will trigger addSuperMushroomEffect() or addPowerStarEffect() in Player.
- A new Consume<Item>Action will have to be made for every new consumable item, which follows SRP but is tedious. Another implementation was considered to solve this (see below) but was not implemented.
- Printing the action to menu will be done by Items.addAction().
- There might unfortunately be multiple identical prints to the menu.

Considered implementation for consuming items and Player gaining the effects: (We have considered a better implementation that follows more principles, but doing so it was not possible to fulfil one of the requirements.)

- Have an interface Consumable, implementing a method consume().
- All consumable items would implement this interface.
- The method consume() will handle all the item's effects of consumption to Player, and be implemented in the item's class itself, hence following SRP.
- A ConsumeAction(ConsumableItem) will handle the consuming for all ConsumableItems. Calls ConsumableItem.consume() and removes the item from Player's inventory.
- This follows OCP, as there is no need to add any new code should more ConsumableItems be added.

Due to Actor.setDisplayChar() being protected and final, it can only be used in Player and hence this implementation was not possible.

Principles:

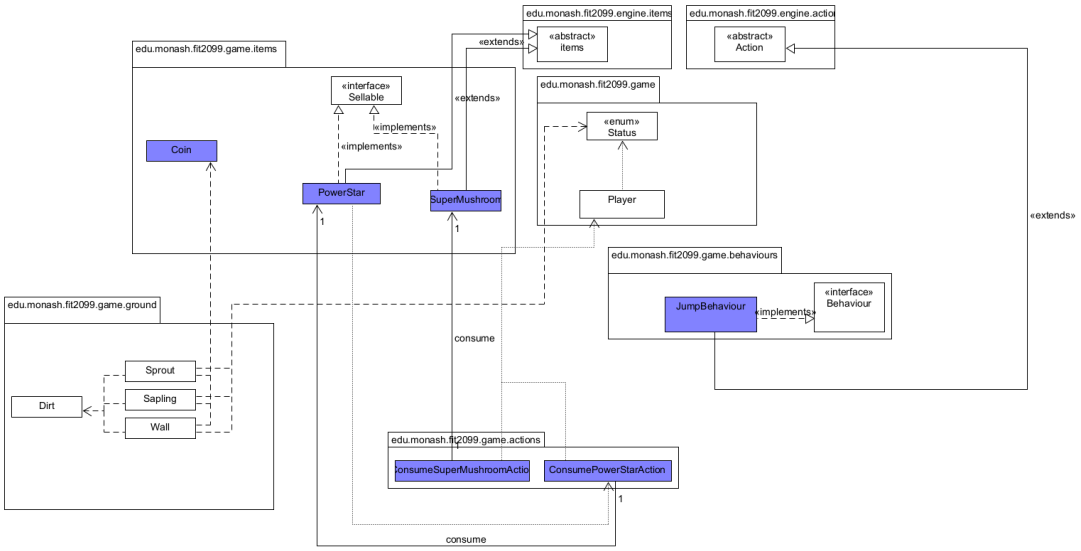
SRP:

- ConsumeSuperMushroomAction and ConsumePowerStarAction both only handle their own respective items' consumption and its effects on Player.

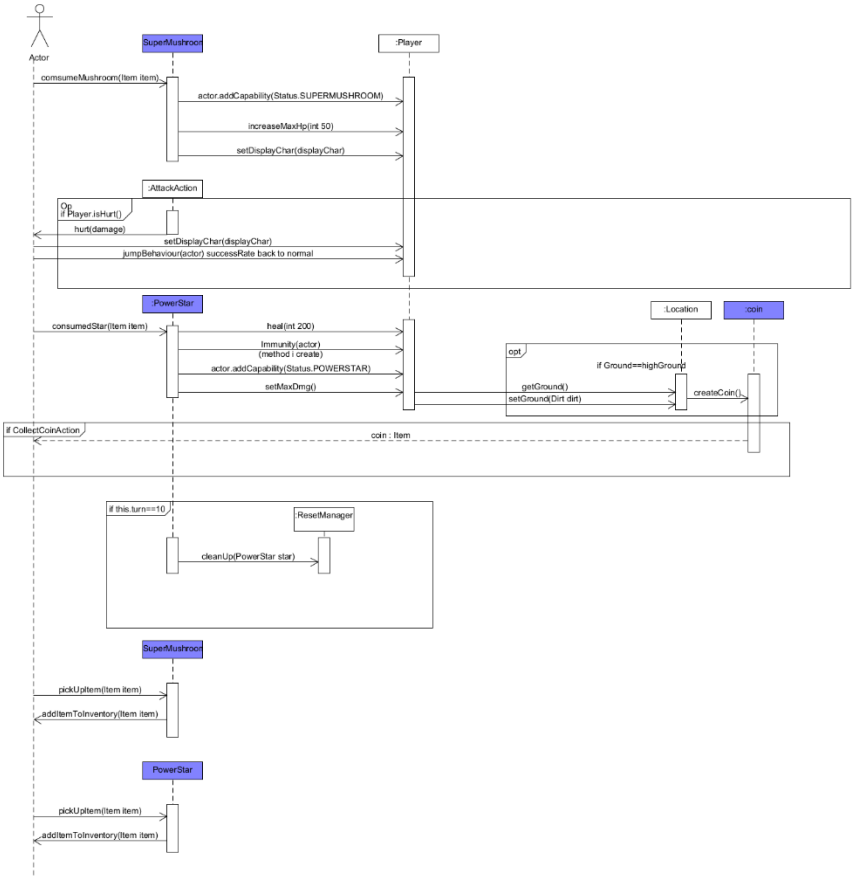
LSP:

- In Player, the hurt() method overrides its super and calls a method to remove SuperMushroom effects. However, the return type is still the same. It will not cause any unexpected behaviour if Player is replaced by Actor.

Class Diagram



Sequence Diagram



Design Rationale

Requirement 5

Changes:

- Toad now has dependencies on each item to be sold, instead of a manager class for all Sellable items.
- TradeAction was replaced with BuyAction
- This is to simplify printing of the action and buying of each item
- Each BuyAction handles the purchase of one item
- There is no longer a manager class that stores instances of all Sellable items
- The original plan was to store a list of Sellable items, iterate through all of them to display to the menu, much like the ResetManager.
- Every new Sellable item would only need to register as one, with no need for Toad to add another dependency on the new item. Hence, this complied with OCP.
- However, this was not implemented for simplicity, and because there are only 4 sellable items.
- CollectCoinAction was made collect coins, instead of using PickUpItemAction. If PickUpItemAction was used, each Coin would be added to the Player's inventory. This would be a waste of memory.
- CollectCoinAction only works on Coins, and adds the value directly to Wallet, which keeps track of the Player's total value of Coins collected.
- Wallet is not a static class but a class with static attributes and methods

Principles:

OCP:

- Interface Sellable implements getPrice() method for each Sellable item. Each item only needs to add one method to become a Sellable item, with no change in BuyAction. Hence, there can be more Sellable items with no modification to BuyAction.

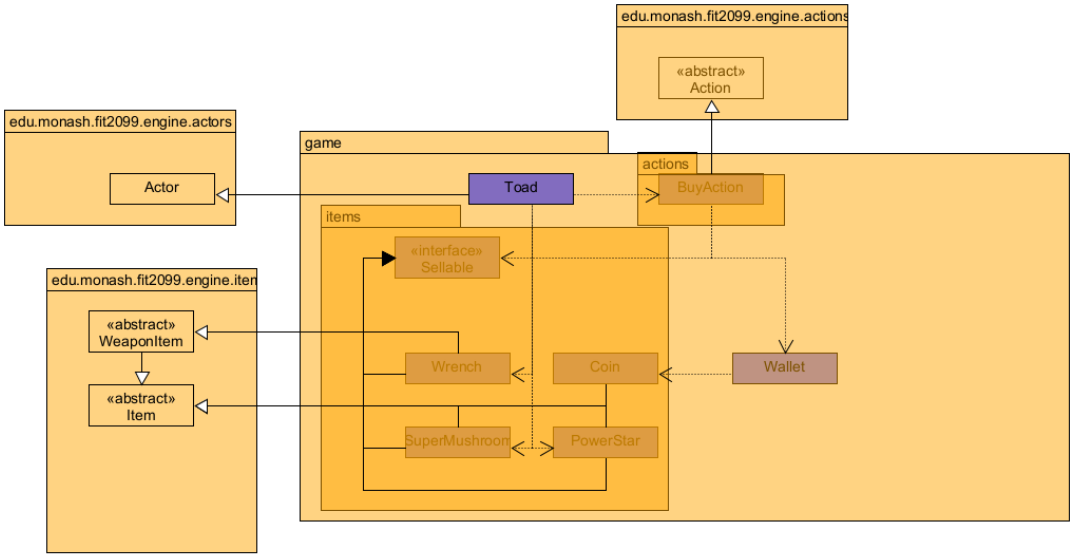
SRP:

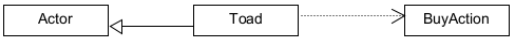
- CollectCoinAction was made specifically to handle the collection of coins.
- As separate classes from PickUpItemAction, because of the key difference that this Action only works for coins, and collected Coin value goes to Wallet.
- BuyAction was made to handle paying and transaction of items
- Wallet was made to keep track of Coin balance.
- Rather than having to keep track of the balance of total Coins in inventory.
- Only has a single purpose - to store the total balance. No other functionis implemented in this class.

LSP:

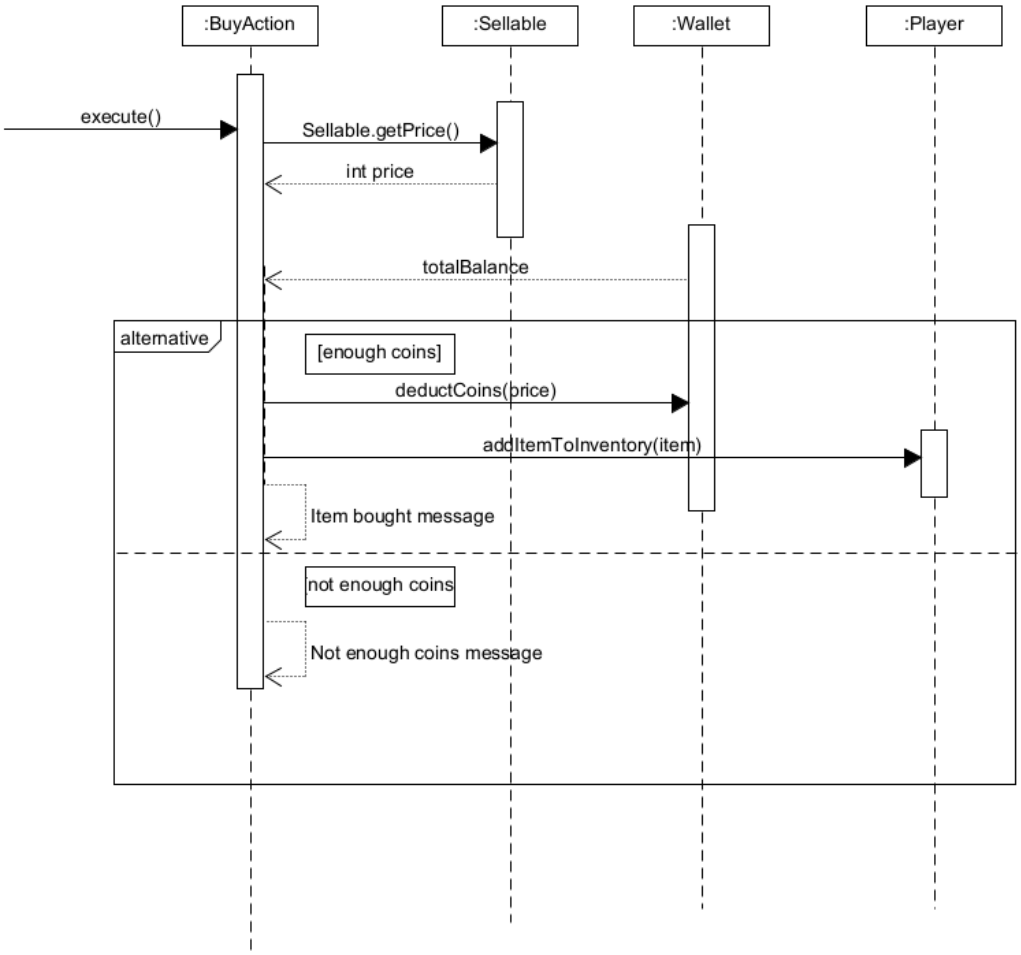
- Substituting BuyAction or CollectCoinsAction with the parent class Action will not result in any unexpected behaviour.
- For CollectCoinsAction, it was considered to override Item.getPickUpAction() with CollectCoinsAction for Coins. But the return type for Item.getPickUpAction() is PickUpItemAction, and changing that would mean Coin cannot be replaced by parent class Item without unexpected behaviour, thereby violating LSP. So, this was not implemented.

Class Diagram





Sequence Diagram



Design Rationale

Requirement 6

Changes:

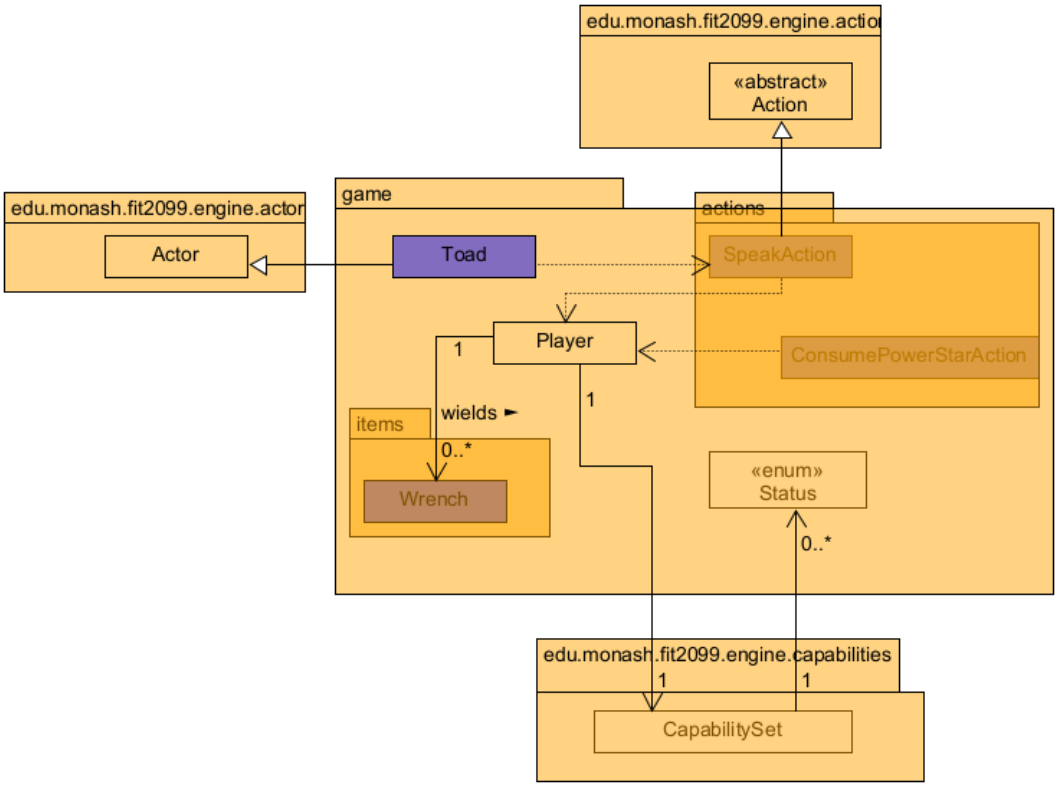
- Nothing much changed, except having the POWERSTAR status triggered by ConsumePowerStarAction
- In the sequence diagram, minor changes for having actual method names.

Principles:

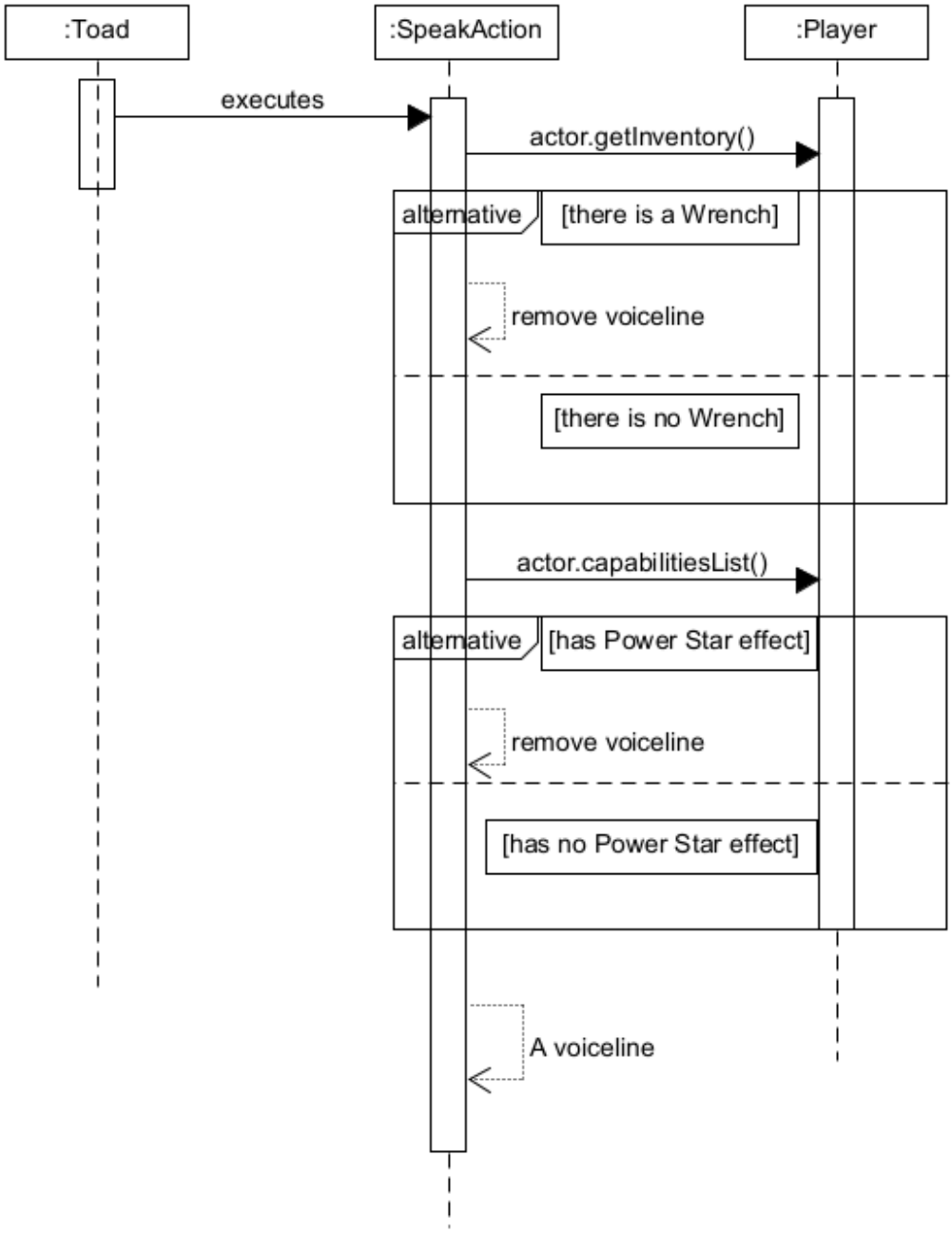
SRP:

- SpeakAction is a class to handle the speaking to Toad.

Class Diagram



Sequence Diagram



Design Rationale

Requirement 7

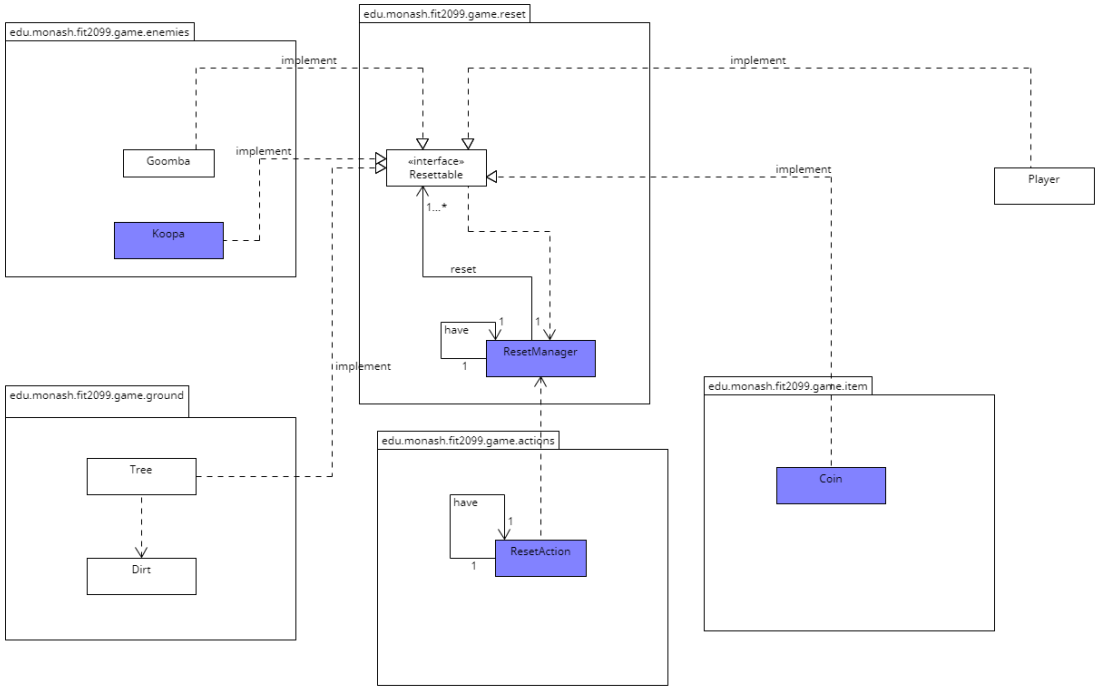
Changes

- Nothing much changed

Dry principles

- Used interface and all resettable item will impelement the interface, therefore we do not need to repeat ourselves to reset each item in each classes one by one, all the resettable item will be reset once the ResetManager's run method is called.

Class Diagram



Sequence Diagram