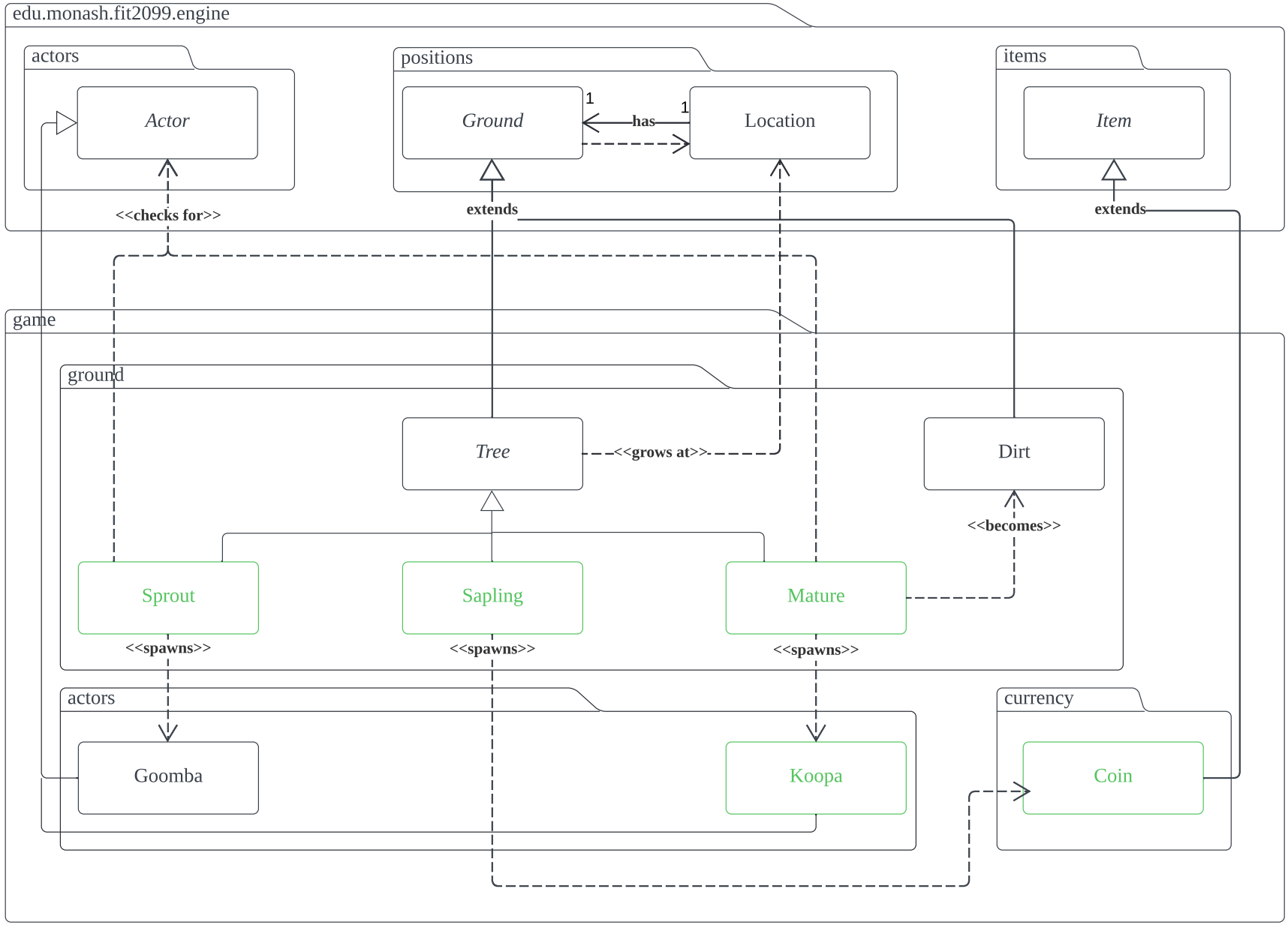
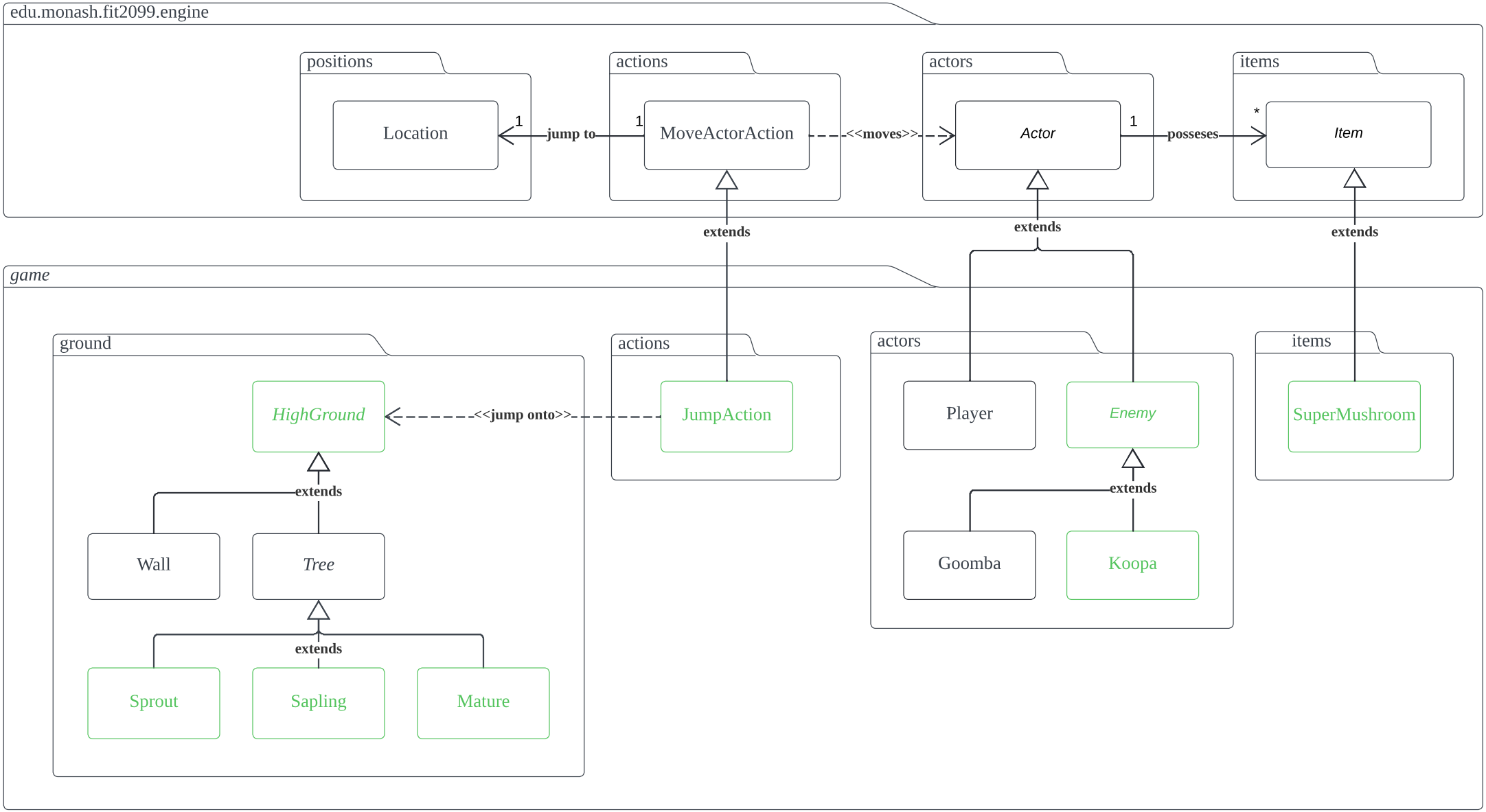
UML DIAGRAMS

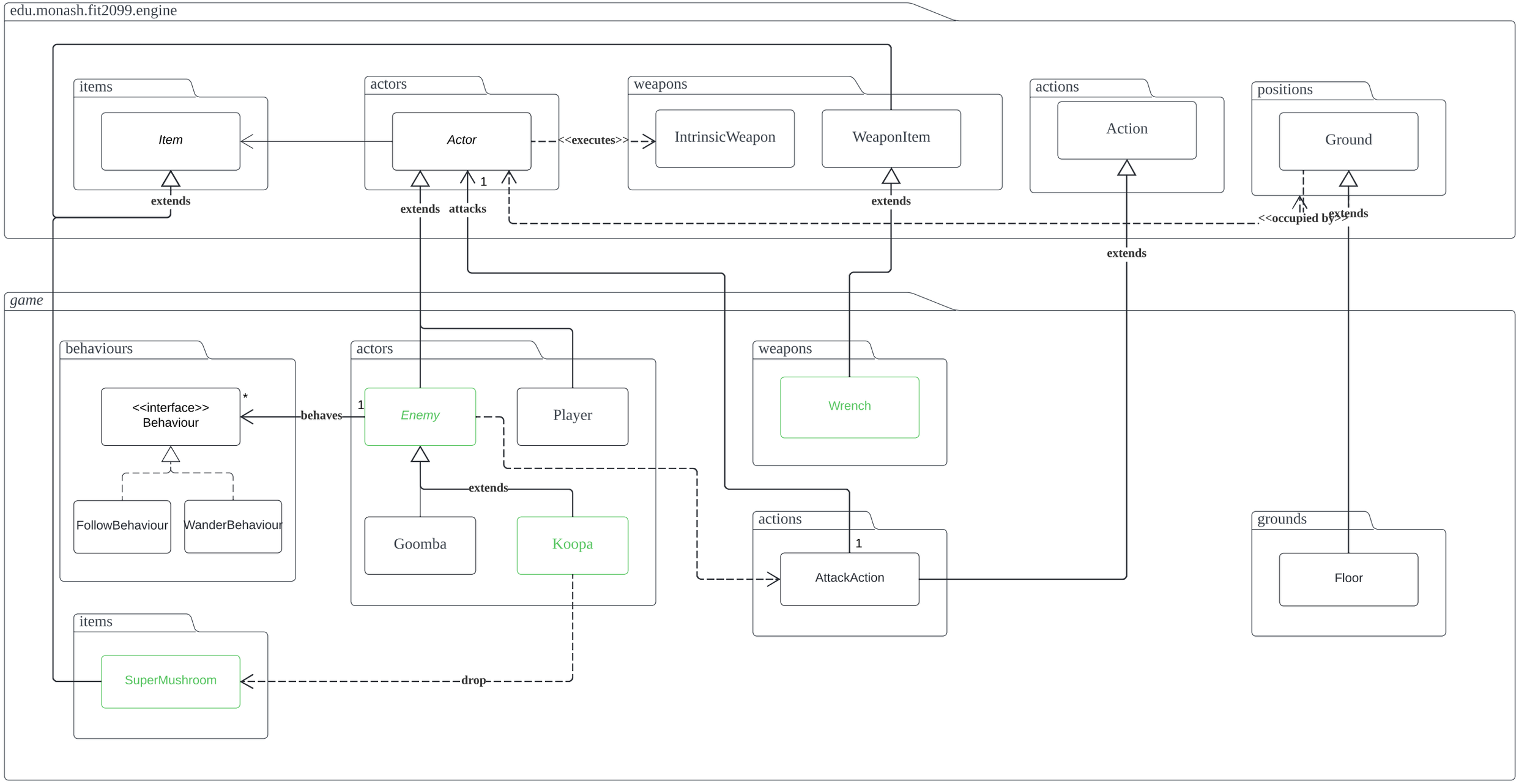
REQ1: Let It Grow



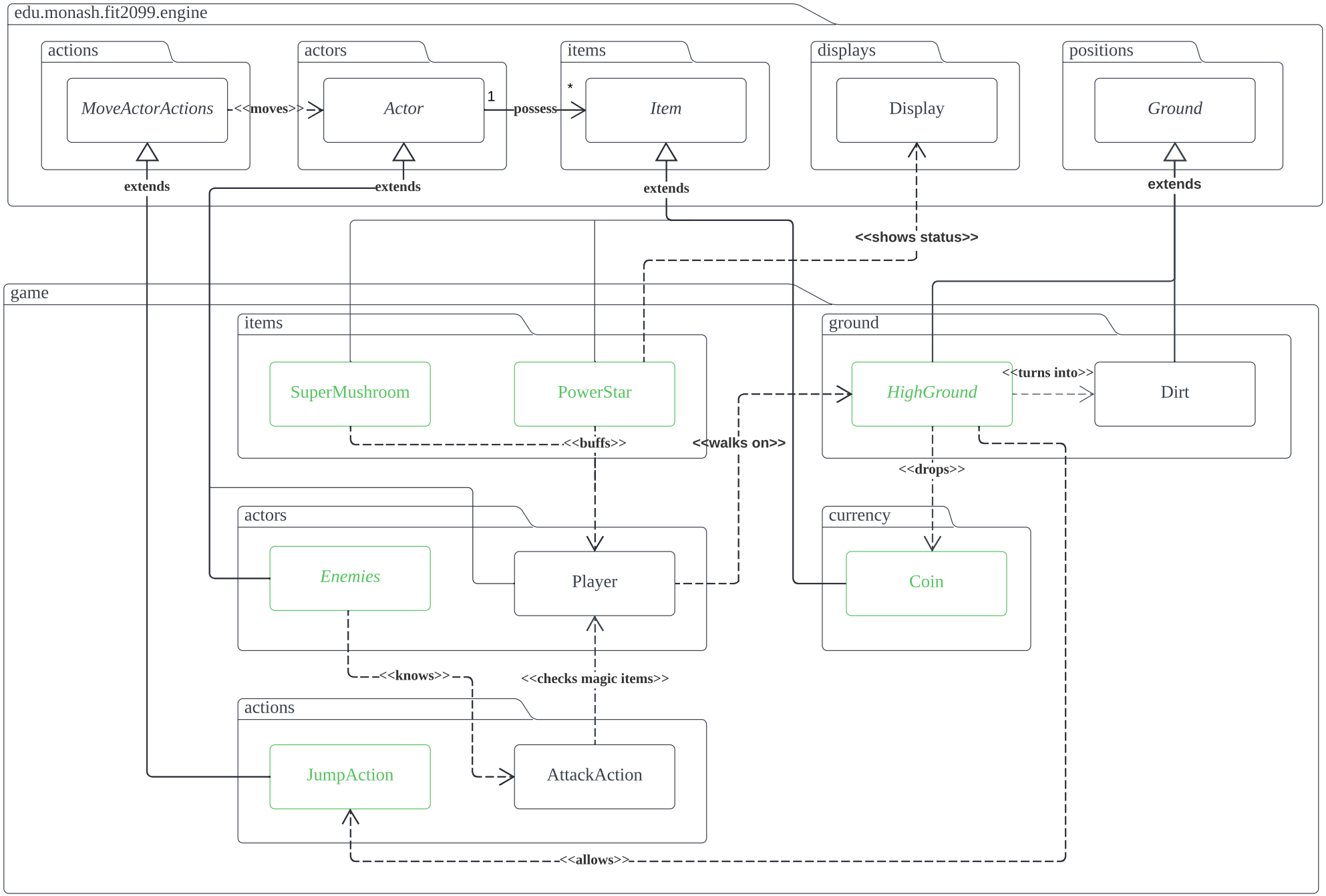
REQ2: Jump Up, Super Star



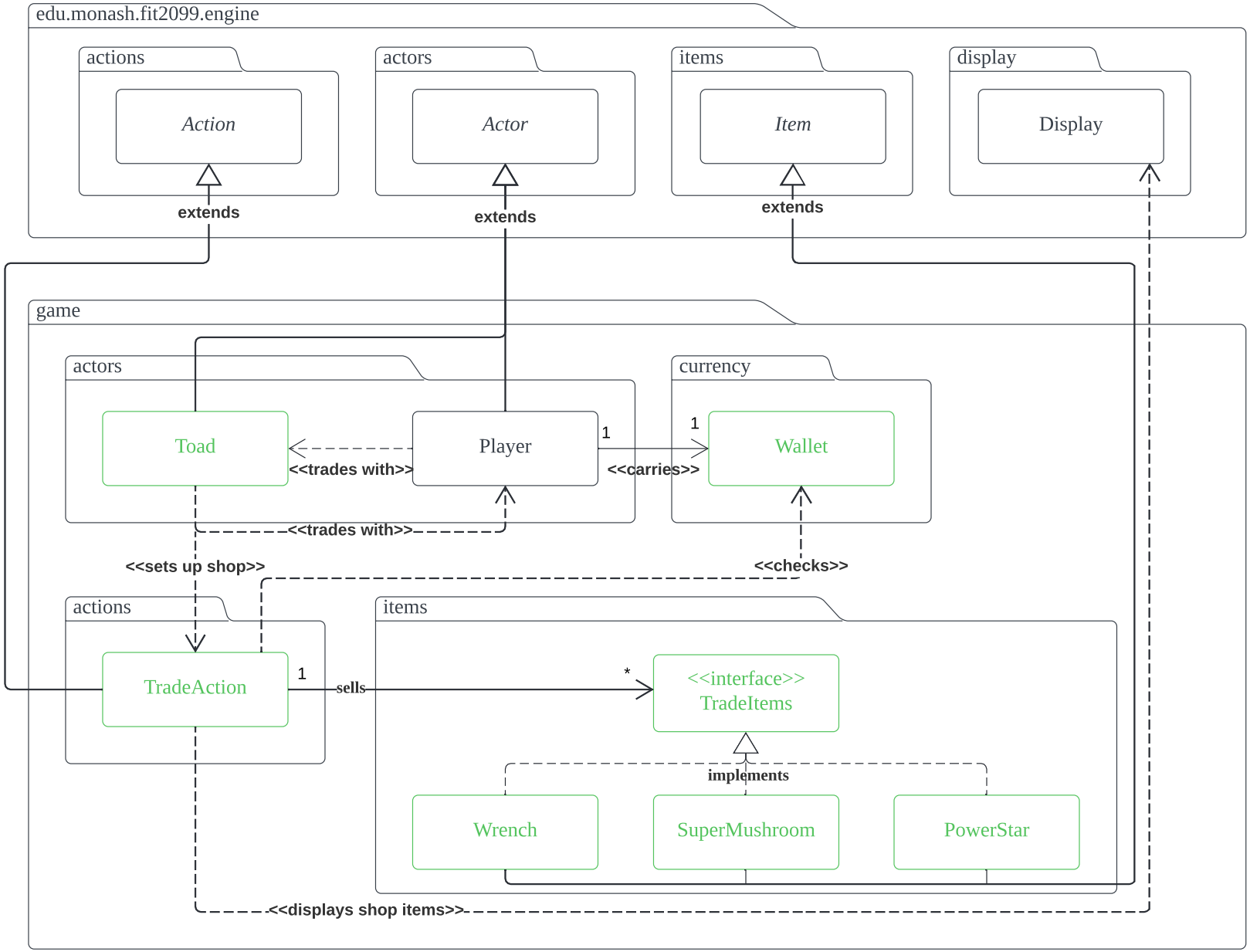
REQ3: Enemies



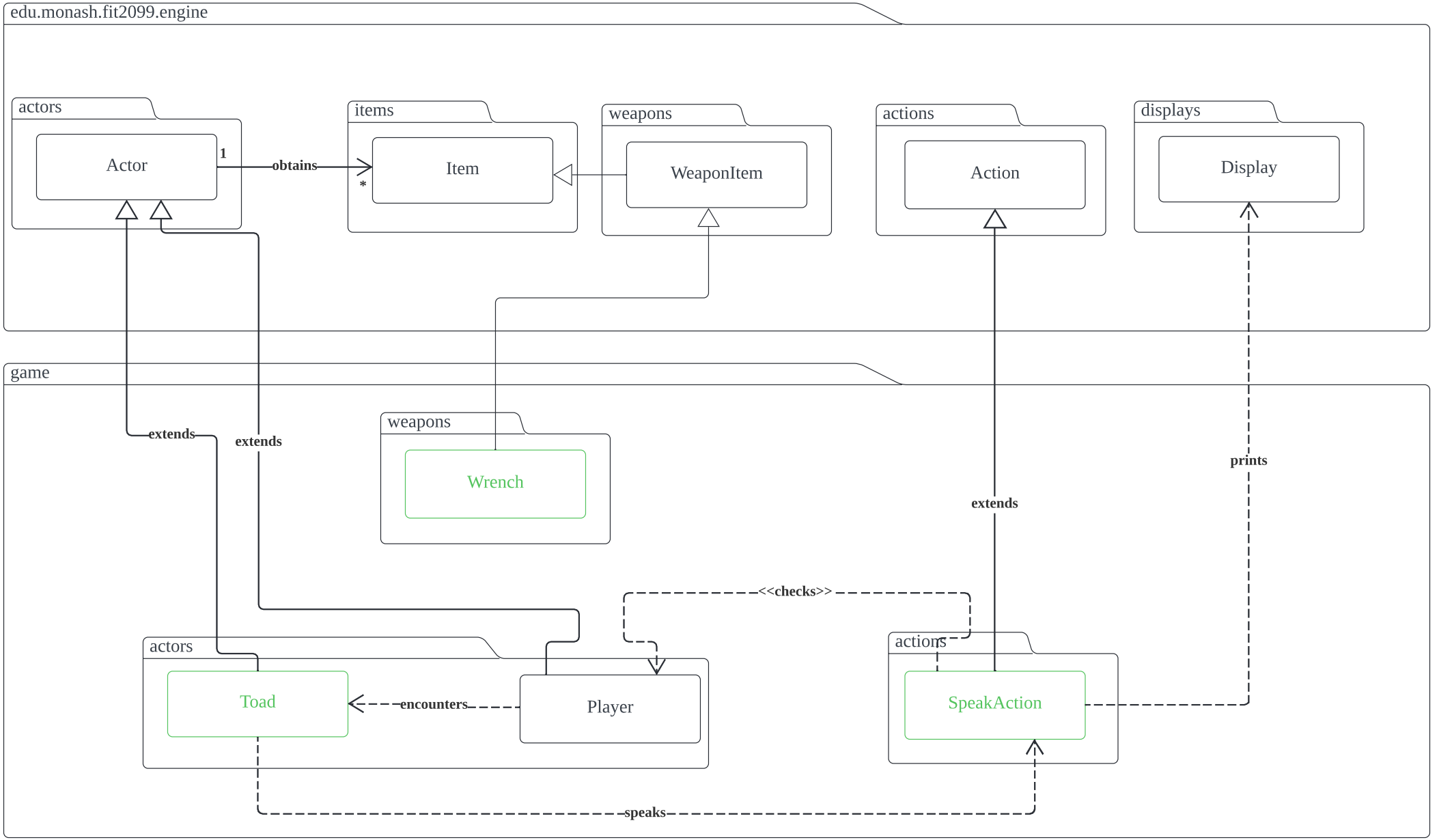
REQ4: Magical Items



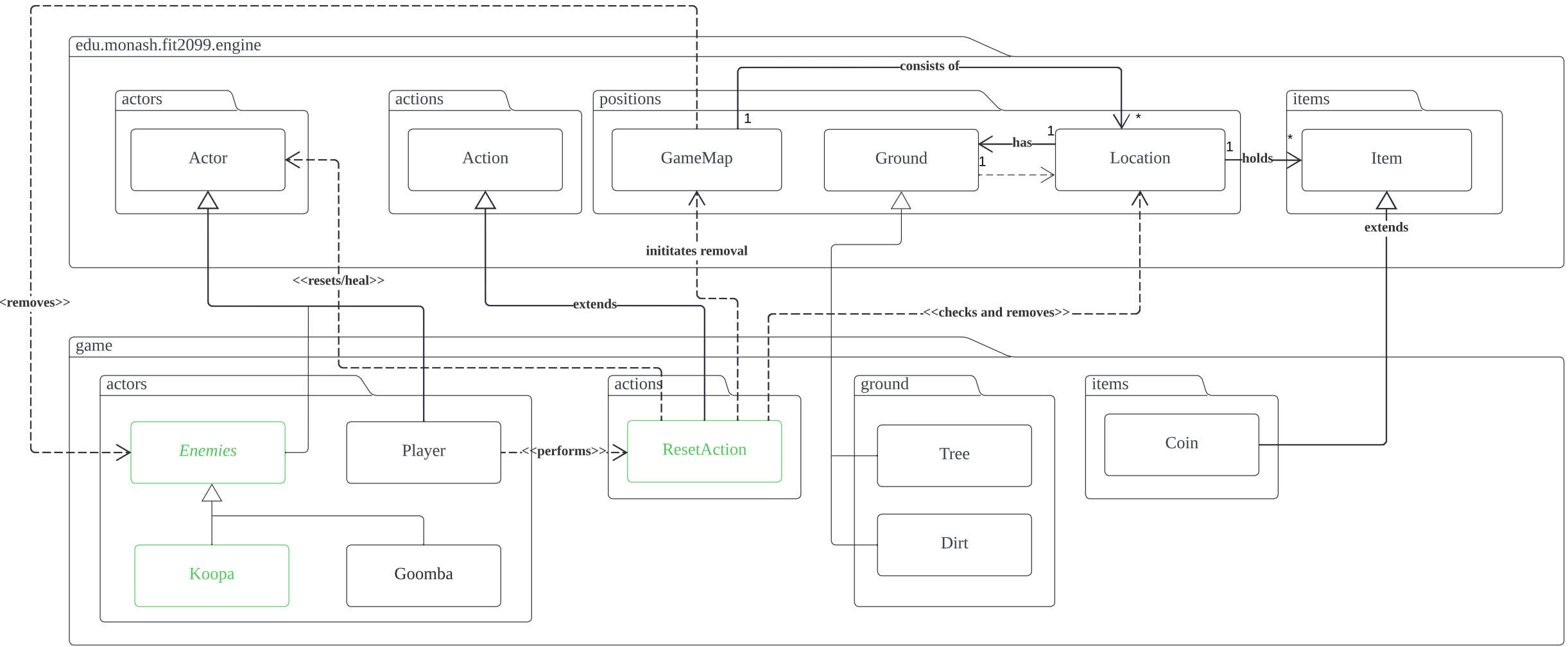
REQ5: Trading



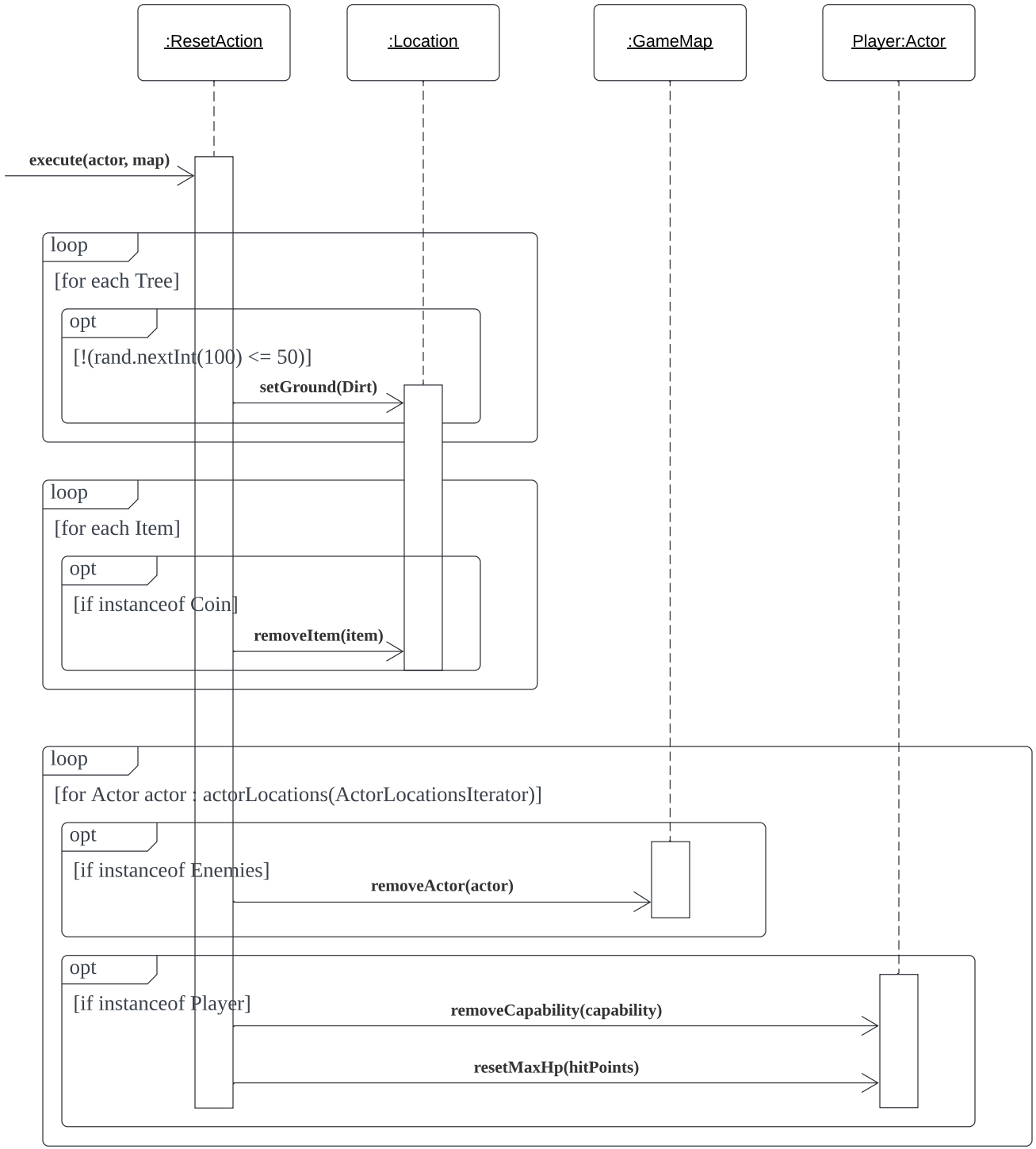
REQ6: Monologue



REQ7: Reset Game



REQ 7 UML Sequence Diagram



DESIGN RATIONALE

REQ1: Let it Grow!
Written by: Matthew Tan, Kar Weng Choong, Jing Ee Lim

Class	Responsibilities
Tree	- The parent class of the Sprout, Sapling and Mature classes
Sprout	- The first growth stage of a Tree. - 10% chance to spawn a Goomba per turn. - Grows into a Sapling after 10 turns
Sapling	- The second growth stage of a Tree. - 10% chance to drop a $20 coin every turn. - Grows into a Mature after 10 turns.
Mature	- The third and final growth stage of a Tree - 15% chance to spawn a Koopa every turn - 20% chance to convert into Dirt every turn
Goomba	- The enemy actor that will be spawned by a Sapling - Will not spawn if an actor is standing on top of the Sapling
Koopa	- The enemy actor that will be spawned by a Mature. - Will not spawn if an actor is standing on top of the Mature
Dirt	- The state of a Tree before it becomes a Sprout. - Mature converts into Dirt, thus beginning the growth cycle of a Tree.
Coin	- The item that will be dropped by a Sapling

Note: The Location class in the engine is responsible for growing a Tree into its respective stages due to the functionality of setGround(ground) method.

Design Rationale (REQ1)

Requirement 1 was conceptualised around the **Open/Closed Principle** by extending three new classes from an abstract Tree class to represent its three growth stages. This will allow us to extend new growth stages for a Tree (such as Death) if such a need would arise in the future, without having to modify any single existing class. **Dependency Inversion Principle** can also be seen applied as we have Tree abstract class being extended by the Sprout, Sapling and Mature. Reason being each subclass of Tree may be located at a specific location and doing so allows the Location class to only depend on the abstract Tree class instead of multiple concrete classes.

REQ2 : Jump Up, Super Star!
Written by: Jing Ee Lim

Class(es)	Responsibilities
JumpAction	- Jump is a class that allows an actor to move onto higher ground. - An extension of the MoveActorAction class
Location	- It represents the coordinates of the map. - Determines where the Player will jump onto.
HighGround	- An abstract class that represents all types of high grounds in the game, which are ground types meant for a Player to perform a JumpAction onto. High grounds include: a) Wall b) Tree - Sprout, Sapling, Mature
Wall, Tree	- Wall and Tree are types(extensions) of High ground. Meant to be jumped onto. - They each have different jump success rates and fall damage: a) Wall: 80% success rate, 20 fall damage b) Tree: (a parent class, look at child class for details)
Sprout, Sapling, Mature	- Sprout, Sapling, Mature are parts of a process of a Tree in the game. - They determine the success rate and fall damage of the jump action performed by the player: - Sprout: 90% success rate, 10 fall damage - Sapling: 80% success rate, 20 fall damage - Mature: 70% success rate, 30 fall damage
Actor	- An abstract class that represents the characters in the game.
Player	- Player is an actor of the game - Player might carry items that can assist gameplay/enhance actions. Thus, the player's performance will depend on items that they have.
Enemy	- Enemy is an Actor in the game. - An abstract class that contains base attributes and methods of an enemy.
Goomba, Koopa	- Goomba and Koopa are enemies of the game. - They are not allowed to perform jump actions. They are not allowed onto high grounds.
Ground	- An abstract class that represents the Grounds in the map.
Floor	- A ground type that is only accessible by Player and not Enemies.
Item	- An abstract class that represents collectable items in the game.
SuperMushroom	- SuperMushroom is an Item to be collected throughout gameplay that will allow 100% jump success rate for the player without fall damage.

Design Rationale (REQ2)

JumpAction extends the MoveActorClass since it is an action that moves an actor (only player) in the game. It must know a location to jump onto and is dependent on the highgrounds and items carried by the player to determine the success rate and fall damage on the player. **Open/Closed Principle** and **Dependency Inversion Principle** were applied in several cases. For example, the game takes the growth process(Sprout, Sapling, Mature) of a Tree into account throughout gameplay since they each have different jump success rates and fall damage. It can also be seen by the abstract enemy class since there are more than 1 kind of enemy. Abstract class Tree and Enemy aligns with the **Open/Closed Principle** since they allow extensions of the class without having to go through modification. **Dependency Inversion Principle** is achieved by having the JumpAction class point to an abstract class, HighGrounds instead of multiple concrete classes such as Wall, Sprout etc. Moreover, these examples obey the rules of **DRY Principle** since the content in the Tree class and Enemy class need not to be repeated for each subtypes.

REQ3: Enemies
Written by: Jing Ee Lim

Class	Responsibilities
Actor	- An abstract class that represents the characters in the game.
Player	- Player is an Actor of the game. - The Player may collect weapon items that can be used to fight against enemies of the game etc. Helping them survive
Enemy	- Enemy is an Actor in the game. - An abstract class that contains base attributes and methods of an enemy. - Each Enemy has an IntrinsicWeapon to be executed to attack the player
Goomba	- Goomba is a type of enemy of the game.
Koopa	- Koopa is a type of enemy of the game. - Koopa will drop a SuperMushroom when its shell is destroyed by the player. An instance of SuperMushroom will be created then.
Behaviour	- An interface class that represents the behaviour of an enemy
FollowBehaviour	- A class that represents a type of behaviour (follow behaviour).
IntrinsicWeapon	- A final class that represents an attack by an actor unarmed.
Item	- An abstract class that represents collectable items in the game.
SuperMushroom	- SuperMushroom is an Item to be collected throughout gameplay - It is dropped by a Koopa when its shell is destroyed
WeaponItem	- An abstract class representing items to be collected throughout the game to help perform attacks
Wrench	- Wrench is a weapon item that can be collected by the player to fight enemies
Action	- An abstract class that represents the action of an actor.
AttackAction	- An action to be performed by an actor.

Design Rationale (REQ3)

Actors of the game each have an intrinsic weapon to perform an attack. A player however may collect additional weapon items along the game and use them for attacks. A wrench is a weapon item and can be used to attack a Koopa, upon destroying a Koopa's shell will produce a SuperMushroom which is also a collectable item. An enemy will have a behaviour, follow behaviour. Enemies also have an attack action on actors of the game. **Open/Closed Principle** can be seen applied. One of few examples include the Wrench class extending the WeaponItem abstract class since we know an actor may possess multiple items one of them being a wrench. Another principle, **Dependency Inversion Principle** can be seen by having the behaviour interface class responsible for all behaviours of the enemy, in this case a follow behaviour. This prevents enemies from depending on multiple concrete classes. Instead it just points to an interface class.

REQ4: Magical Items
Written by: Kar Weng Choong

Class	Responsibilities
Player	- The main Actor that obtains the magical items SuperMushroom and/or PowerStar and their benefits. - Stores these items in their inventory until they are consumed or expire. - Gains a status effect when consuming a SuperMushroom and/or PowerStar.
SuperMushroom	- A magical item, provides a Player with buffs: a) increased max HP, b) better jumps, and also c) changes the display character of a Player.
PowerStar	- A magical item, heals a Player and provides them with buffs: a) can walk on higher grounds without jumping (higher grounds turn to Dirt and drops a Coin), b) immune to enemy attacks, c) instant-kill attack on enemies. - Keeps track of its own lifetime in the game (max of 10 turns). - Displays its remaining time in the game map or inventory, after which it is removed entirely if not consumed.
Enemies	- An abstract class that represents all Enemies in the game. - Instantly killed if a Player has an active PowerStar status. - Allows a Player to attack it through AttackAction.
AttackAction	- An action performed by actors to attack another actor on the map. - Checks if an actor is under the status of a magical item, then when that actor is attacked: a) removes the status for an active SuperMushroom status, b) inflicts no damage for an active PowerStar status. c) Inflicts normal damage if there are no magical buffs.
JumpAction	- An action performed by actors to “jump” onto high grounds on the map at varying success rates. - Checks if an actor is under the status of a SuperMushroom. - Under the effect of an active SuperMushroom status, all jumps onto all high grounds will have a 100% success rate.
HighGround	- An abstract class that represents all types of high grounds in the game, which are ground types that require a Player to perform a JumpAction in order to reach that location. - Under the effect of an active PowerStar status, any high ground that the player walks on will be destroyed: a) turning into Dirt, and b) dropping a Coin.
Dirt	- A type of ground that represents the surface level of the game map.
Coin	- An item that represents the currency of the game.

Design Rationale (REQ4)

Requirement 4 was conceptualised with the **Open/Closed Principle** in mind, specifically for the status effects of the PowerStar, which allows an enemy to be instantly killed if an attack is successful, and for high grounds to be destroyed into Dirt (and drop a Coin). Hence, two new abstract classes are implemented, namely Enemies and HighGround such that any new enemy or high ground added to the game in the future can inherit the effects of the PowerStar, instead of adding the new classes into the PowerStar class. This also aligns with the **Do Not Repeat Yourself (DRY) Principle**, as it removes the need to repeat similar code in every individual enemy and high ground class.

REQ5: Trading
Written by: Kar Weng Choong

Class	Responsibilities
Player	- The main Actor that trades Coins with Toad for Items. - Carries a Wallet with them to store their Coins.
Toad	- A peaceful Actor that remains stationary in the middle of the map, selling useful Items in exchange for Coins: a) Wrench, b) Super Mushroom, and c) Power Star.
TradeAction	- An action performed by actors to trade with another actor on the map. - Prints the items available for sale by Toad. - Checks if a Player's Wallet has sufficient balance to purchase an Item. - Adds the item to their inventory and deducts the Wallet's balance if it is sufficient. - If not, prints a message in the UI indicating insufficient balance.
Wallet	- An inventory to store a balance of coins collected in the game. - Increases the balance when a Player picks up a Coin. - Decreases the balance when a Player purchases from Toad.
TradeItems	- An interface representing items that can be sold by Toad. - Contains methods to be implemented by each Item sold.
Wrench	- A weapon item sold by Toad, costs 200 Coins.
SuperMushroom	- A magical item sold by Toad, costs 400 Coins.
PowerStar	- A magical item sold by Toad, costs 600 Coins.

Design Rationale (REQ5)

Requirement 5 was conceptualised with the **Dependency Inversion Principle** and the **Open/Closed Principle**, which can be observed through the implementation of an interface, TradeItems, that is associated with TradeAction. This not only prevents TradeAction from depending on multiple concrete classes (the items sold by Toad), but allows for the addition of more items for sale in the future without the need to overhaul TradeAction.

REQ6: Monologue
Written by: Matthew Tan Yau Koon

Class	Responsibilities
Player	- The main actor who will be interacting with the Toad. - The Player will interact with Toad, who will initiate the monologue.
Toad	- The actor who will be performing the SpeakAction class. - Will speak one of four lines by random, and certain lines are excluded depending on whether the Player has a Wrench in its inventory or is in a powered up state.
SpeakAction	- The main class will contain the necessary methods and attributes to perform the act of speaking. - The class will check the Player's inventory and the Player's state to identify which dialogue to exclude. - Uses the Status enum to identify if the Player is currently under the effects of a Power Star. - Prints one of the four possible lines that Toad will speak.
Wrench	- The WeaponItem that the Player may be holding. - If the Player currently owns a Wrench, certain dialogue will be excluded from the monologue.

Design Rationale (REQ6)

Requirement 6 was designed while taking into account the **Open-Closed Principle**. This can be seen with the creation of a new class named *SpeakAction*. The *SpeakAction* class inherits/extends the *Action* class, which abides to the Open-Closed Principle whereby the class has been extended without modifying the source code. Additionally, the **Dependency Inversion Principle** was also taken into consideration while designing the system. This can be seen in the fact that the *SpeakAction* class is extended from the *Action* class, and the *Toad* and *Wrench* class are extended from the *Actor* and *WeaponItem* class respectively.

REQ7: Reset Game
Written by: Matthew Tan Yau Koon

Class	Responsibilities
Player	- The main actor that will invoke a method from the ResetAction class. - The Player can reset the game at any point, but this can happen only once per playthrough.
Enemy	- An abstract class for the Goomba and Koopa class. - Generalises the Goomba and Koopa class into one to make it easier to remove Enemies from the map.
Goomba	- The enemy Actor that will be killed off from the game.
Koopa	- The enemy Actor that will be killed off from the game.
ResetAction	- The main class that will initiate the process of resetting the game. a) Has a chance to convert Trees into Dirt b) Kills off all enemies in the game c) Removes all Coins in the map d) Resets Player status to default/neutral e) Heals the Player to maximum - The ResetAction class will invoke a method from the Location class to perform the operation of converting the Tree into Dirt.
Tree	- The terrain object which has a 50% chance of being converted back into Dirt when a reset occurs.
Dirt	- The terrain object that will be converted from a Tree when a 50% chance is successfully rolled.
Coin	- The item in the map that will be cleared off when a reset occurs.

Design Rationale (REQ7)

While designing this extended system, the **Open-Closed Principle** is taken into account once again. This is because the *ResetAction* is a subclass of an abstract class Action, present in the engine package. This shows that the system has added functionality without modifying existing code. Moreover, the use of abstraction and inheritance in turn will abide by the design principle **DRY (Don't Repeat Yourself)**. Using abstraction/inheritance reduces redundancy in code, increasing readability of the code. Lastly, the **Dependency Inversion Principle** is also observed in this extended system. The creation of the abstract Enemy class generalises the Goomba and Koopa classes, removing the need to call multiple classes when the game removes all enemies.