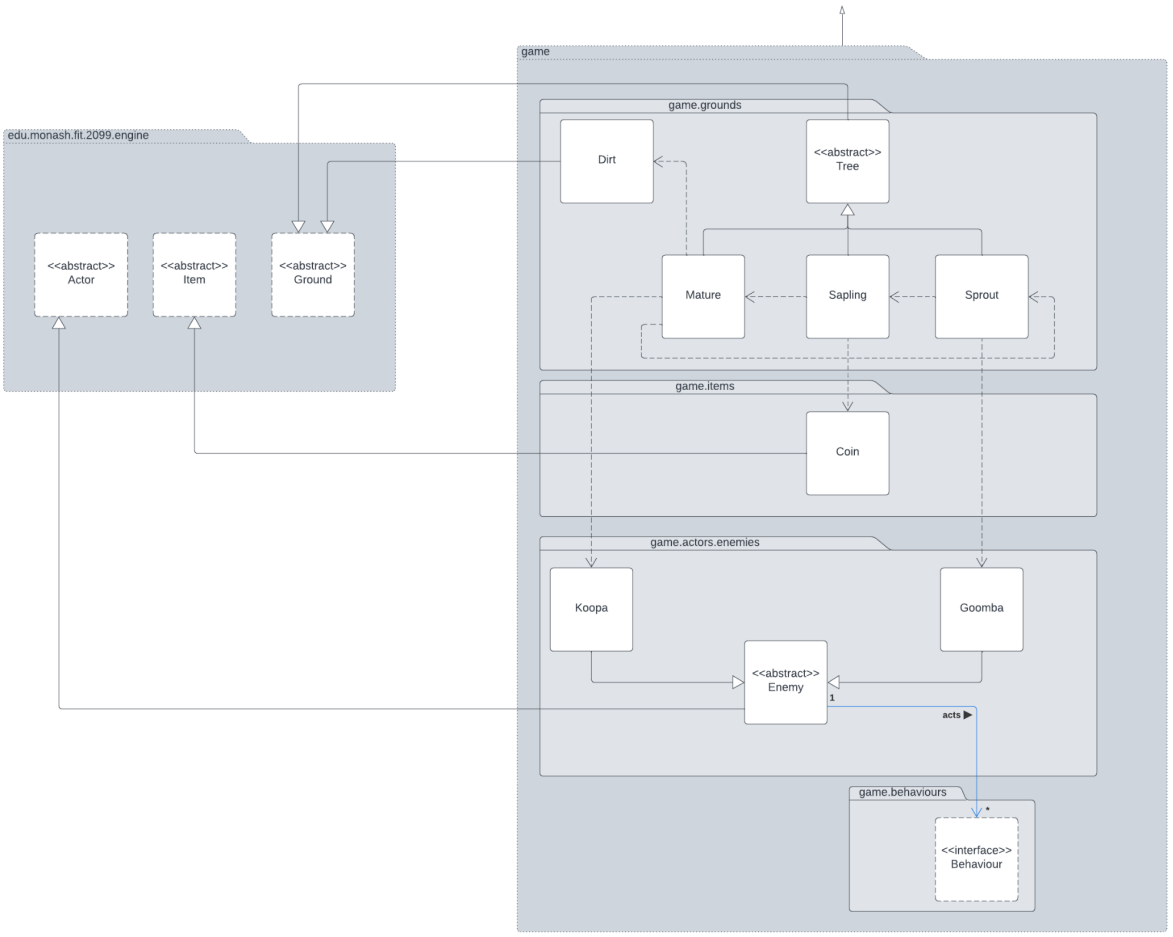
Lab 19 Team 3

FIT2099 Assignment 1

Lab 19 Team 3

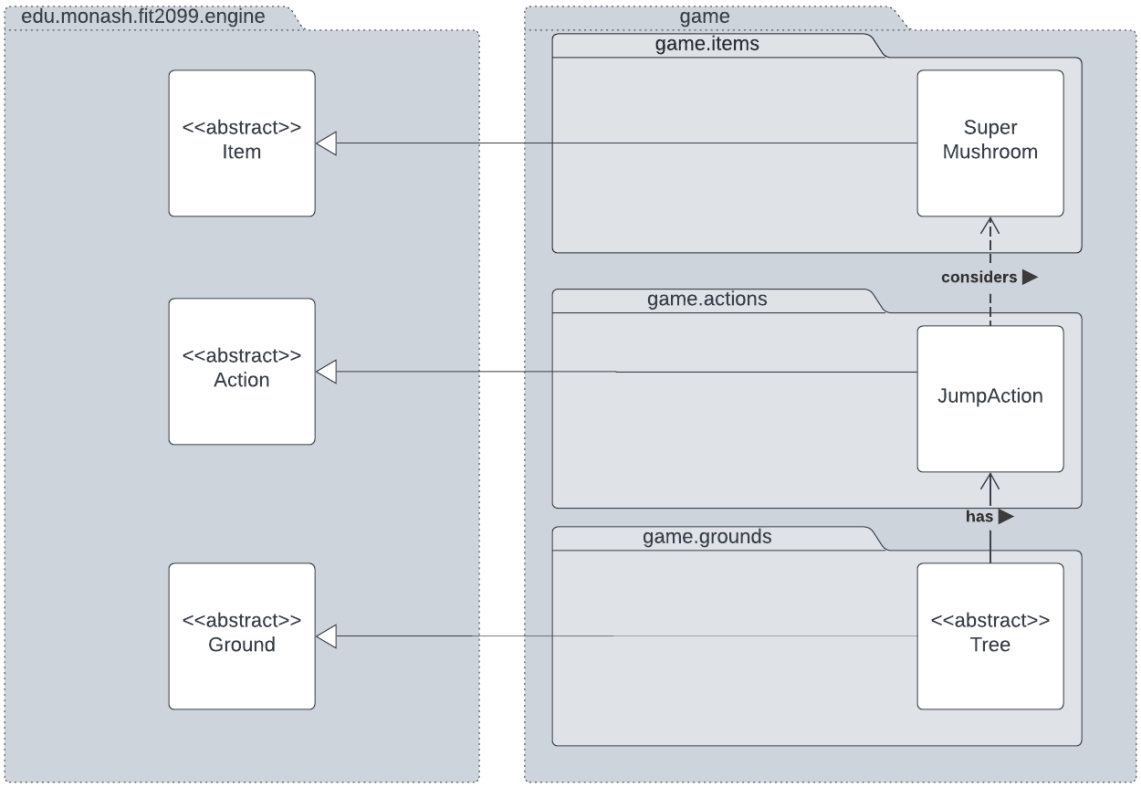
REQ1



In requirement 1, we decided to have the Tree class be abstract, and extend it's functionality in 3 subclasses (Mature, Sapling, Sprout). This allows us to put any shared functionality into the Tree class, and implement any differences into the respective classes.
The Sprout and Mature will directly create new Goomba and Koopa trees (respectively), which resulted in that association.
The Sapling will generate coins, which differ from Mature and Sprout who spawn Actors. Coins will inherit from the abstract Item class given by the engine. Instead of going into the Actor's inventory, it will go into a *wallet* which will hold a simple integer value, representing the Actor's money (this functionality is not included in the Actor class, so will need to be added to the Player class - which extends Actor)
Sprouts grow into Sapling, which grow into Mature, which can create new Sprouts in adjacent dirt. Due to this functionality, there is a 'cyclic' association between these 3 classes.

The Mature tree will also have a chance to reduce to dirt each turn. This results in an association to the Dirt class, as it needs to create new Dirt objects.

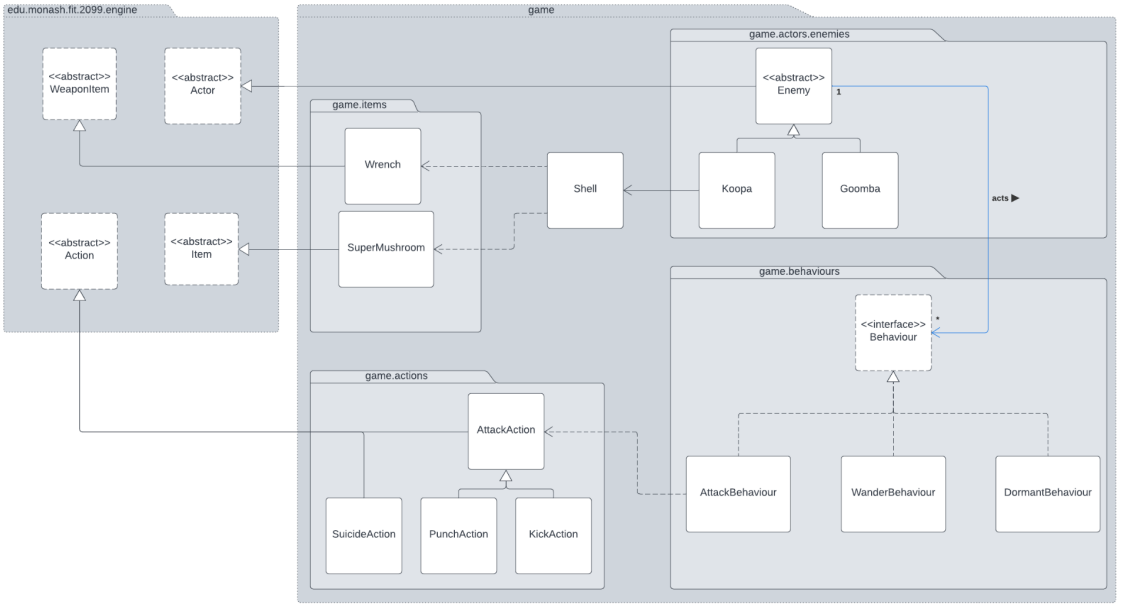
REQ2



We will create a new JumpAction class, which inherits from Action, and will be added as an attribute of the Tree class, which created a dependency.
The JumpAction class inherits from the engine's Action class, which already has an association to the Actor class (in it's execute(Actor, GameMap) method). Using this, we can check the Actor's capabilities to see if it has an active Super Mushroom. (See req 4)

Lab 19 Team 3

REQ3



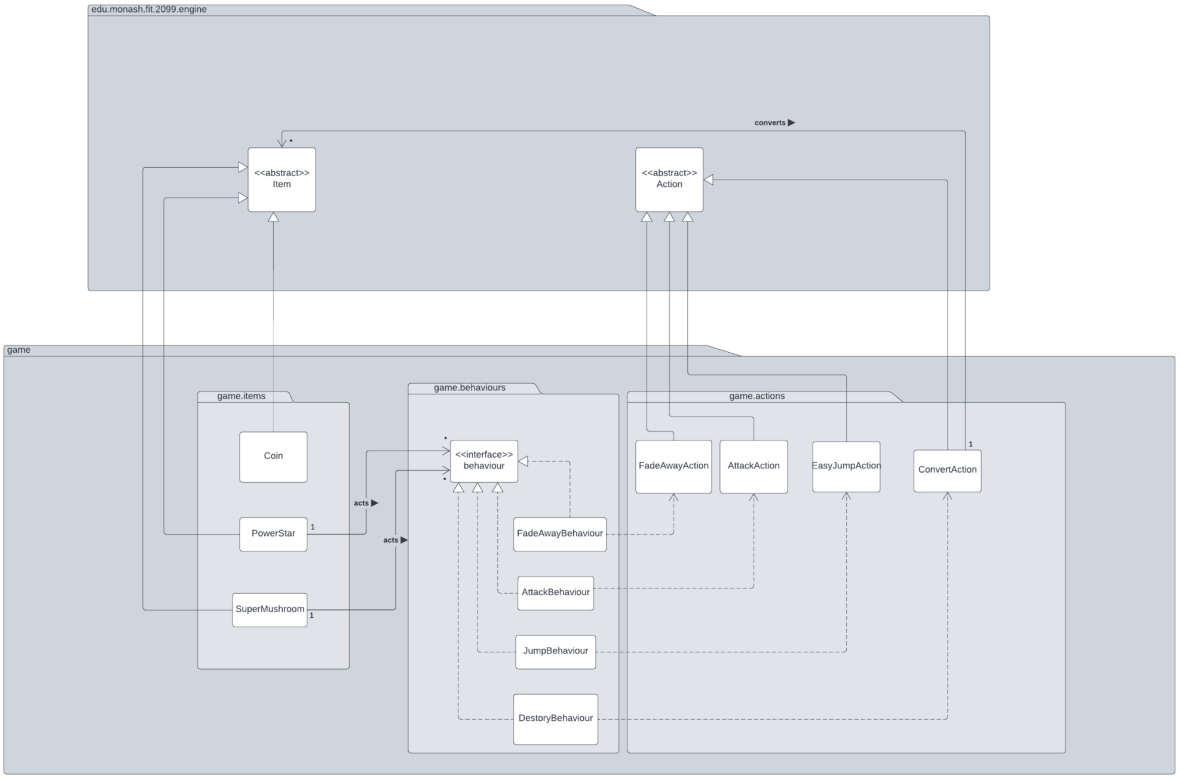
The game will have two enemies - Goomba & Koopa. Both inherit from Enemy, an abstract class that inherits from the Actor class provided by the engine. Both enemies will implement Behaviours, depending on the state of the game. The Dormant Behaviour state is also included among the Behaviours and will activate when the enemy Koopa has lost all its HP points. The choice of the Behaviour to be separated from AttackBehavior and WanderBehaviour is so that when the Koopa undergoes the DormantBehaviour, it will not be able to follow, attack or wander around.

To break the Koopa's shell, the player must possess a Wrench. Wrench implements the Capable interface as shown in the engine UML, and will provide the player with the capability to break the shell, when equipped. The shell will depend on the wrench, so that it is the only weapon in the game which can produce a super mushroom when the shell is broken.

The enemy class actors are able to access the Punch and Kick Actions as well as the SuicideAction. The choice of separating SuicideAction was because it was deemed that SuicideAction will not do damage to the Player and hence should be separated from AttackAction. However, because it is still an action that can be done by the enemy actor, AttackAction and SuicideAction still have some relations.

Lab 19 Team 3

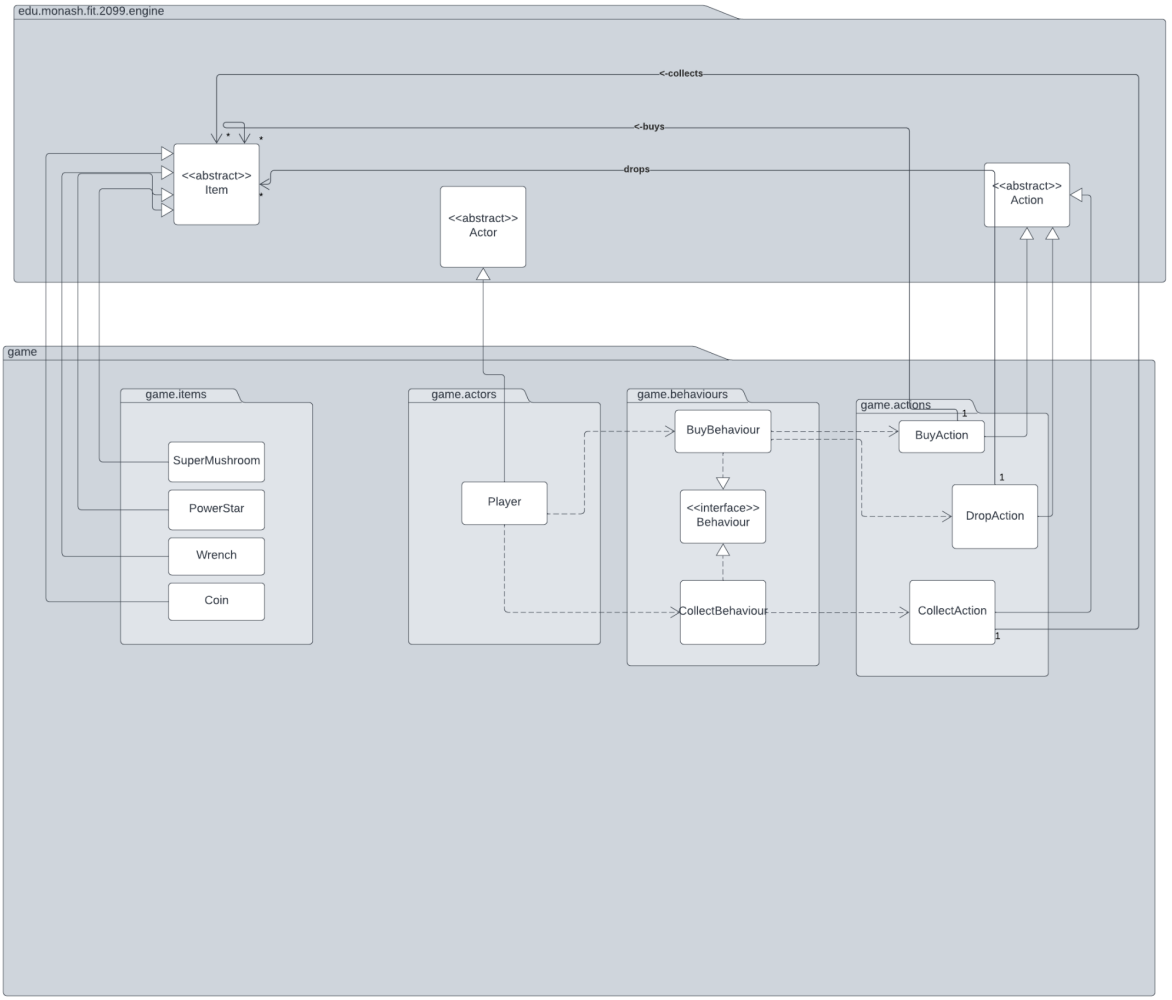
REQ4



We'll need two magical items PowerStar and SuperMushroom in the game, Both of them are the type of Item with their own properties and behaviours. So we design them as subclasses of the Item superclass where we can record what's in common in the superclass and what's different in the subclass. The magical items can have various behaviours, and each behaviour is associated with one or more actions. Some actions don't have a target but some actions(e.g.ConvertAction or AttackAction) will have a target.

Lab 19 Team 3

REQ5



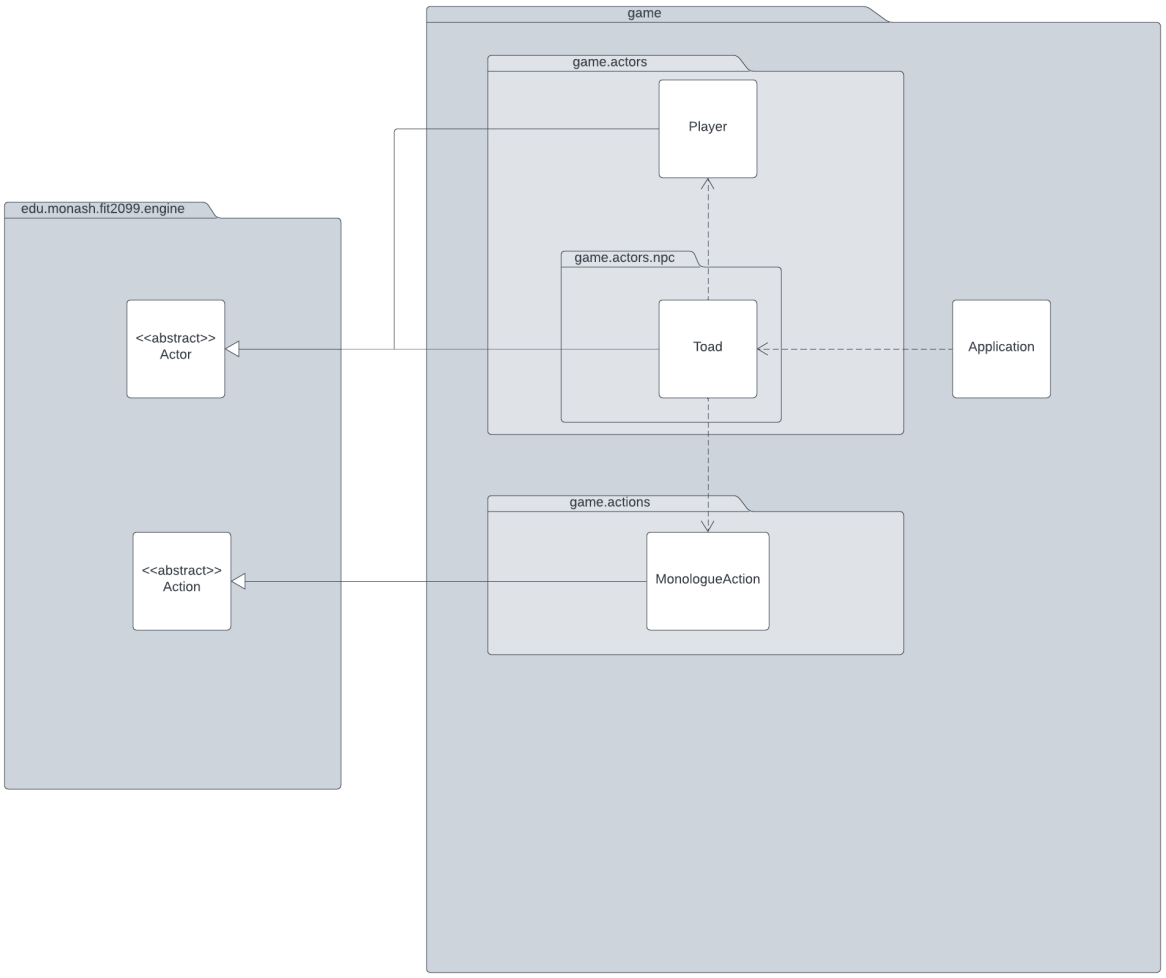
Players can collect coins through CollectBehaviour by performing CollectAction. We choose to implement the Behaviour interface in CollectBehaviour class so that we can keep all behaviours in the system to have some common methods.

BuyBehaviour can be achieved by two actions - BuyAction and DropAction. Both of the actions inherit the Actions class. BuyAction will provide the player with the capability to buy items and DropAction can let the player drop an item. We have an interface called Droppable where the Wrench item implements this interface. When the items are PowerStar or SuperMushroom, they are not droppable; therefore, they don't need to implement this interface.

Wrench, SuperMushroom, and PowerStar are all types of items so they all inherit the Item class in the engine package.

Lab 19 Team 3

REQ6



Toad will have SpeakAction in it's allowable actions (an inherited method from Actor), which created an association between Toad and Speak

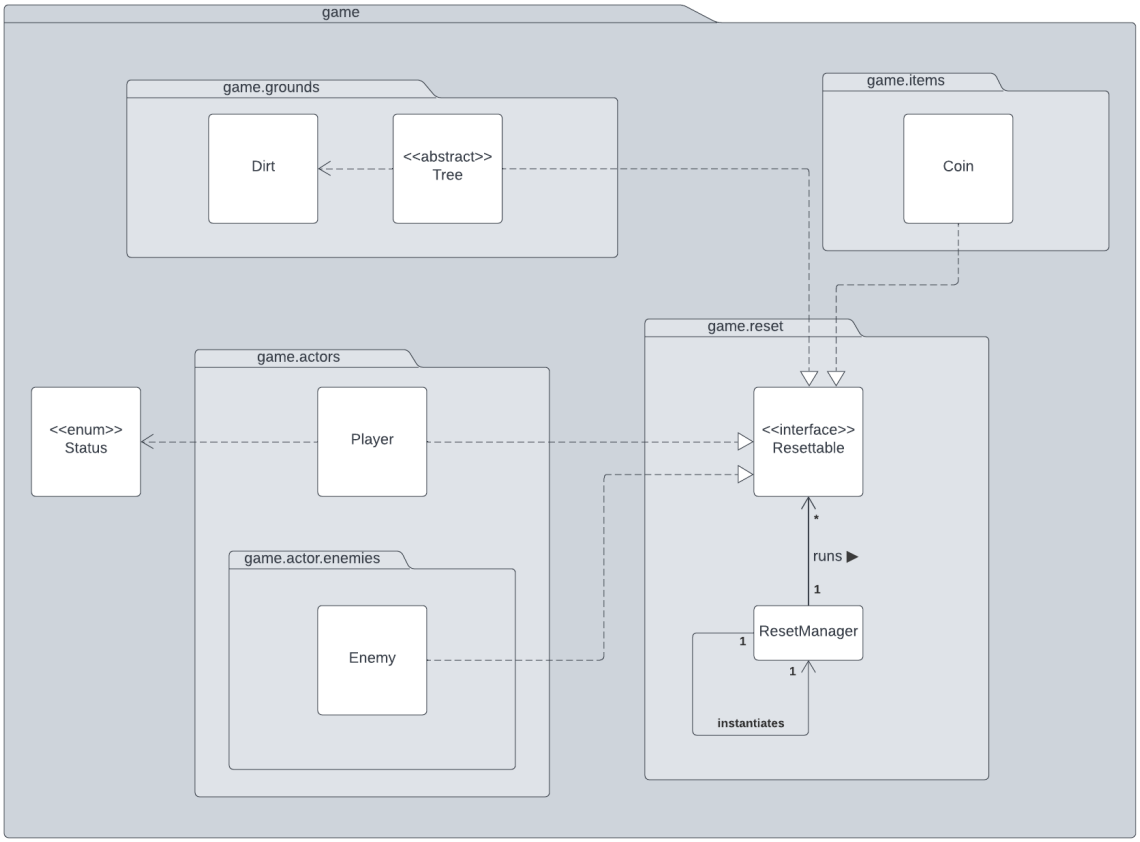
This will allow the player to speak with the Toad when the toad is in one of the player's exit locations.

The Toad needs information about the player, in order to check its inventory and see if it holds a Wrench. This created an association between Toad and Player.

The logic for what message the toad should send to the player would be inside the MonologueAction.execute(Actor, GameMap) method, which can check the player's inventory through the game map.

In order to place a Toad actor into the Game Map, Application.java needs to know about Toad, which created an association between Application and Toad.

REQ7



Player Mario is able to inherit the resettable interface which allows access to the ResetManager. The reset manager controls all of the execution when resetting all entities as the Tree class (which holds sprout, sapling and mature - see REQ1) allows the trees to be converted back to Dirt. Enemy class also inherits the resettable interface so as to kill all enemies using the ResetManager. The player for the same reason, uses the reset manager to reset the status of the SuperMushroom and PowerStar on the player if these are active during time of reset. Item coin also inherits the Resettable interface which allows for coins to be removed from the ground. Coins class does not include the Super Mushrooms and Power Stars, hence, these items will not be removed from the ground.

Lab 19 Team 3

Sequence Diagram

Buying action