Design Rationale

REQ1: Let it grow!

An abstract class Tree that extends Ground abstract class was created for the 3 classes of Sprout, Sapling and Mature to tidy up to code. The Sprout class has dependencies with Goomba and Sapling class. This is because the Sprout class will spawn Goomba on chances every turn and will create a Sapling instance after 10 turns to replace itself. The Sapling class has dependencies on Coin and Mature class as it will spawn Coin instances on chances and create a Mature instance after 10 turns. Finally, the Mature class has dependencies on Koopa, Sprout, Dirt class and Fertile interface. This is because the Mature class will create Koopa, Sprout or Dirt instances on chance every turn. At the same time, to create Sprouts instances, the Mature class will check for surrounding land if it implements the Fertile interface. However, at the current moment, only Dirt class implements the Fertile interface. Koopa and Goomba are classes that extend the Actor abstract class and Coin class extends the Item abstract class.

REQ2: Jump Up, Super Star!

The JumpAction has an association with the HighGround abstract class as it stores the target HighGround to jump to as an attribute. The HighGround abstract class is extended by Wall and Tree that requires the player to jump onto it. The JumpAction class extends the abstract Action class and has a dependency on the Actor class as it checks for the Actor capabilities to change the chances of success. It also has dependency on Dirt and Coin class as it destroys Walls and Trees if the actor has Invincible capabilities.

REQ3: Enemies

An abstract class Enemy was created for Goomba and Koopa. This is because all Enemy instances have behaviours. Behaviour is an interface implemented by WanderBehaviour, FollowBehaviour and AttackBehaviour. WanderBehaviour and FollowBehaviour control the pattern of which the Enemy instance moves and has a dependency on MoveActorAction to move the Enemy instances. AttackBehaviour has a dependency on AttackAction as it controls when the Enemy executes an AttackAction on the Player. The AttackAction, AttackBehavior, and FollowBehaviour has an actor attribute of the target of the action. AttackAction checks for the presence of a Wrench instance of the actor executing the action, which is an extension of WeaponItem, in the actor’s possession in order for the actor to destroy the Koopa’s shell.

REQ4: Magical Item

SuperMushroom and PowerStar class extends the abstract Item class and implements the Consumable interface while ConsumeAction class extends the abstract Action class. The SuperMushroom and PowerStar class offer the ConsumeAction that will offer the actor to consume each of the items. SuperMushroom and PowerStar on the Player as it adds capabilities to the Player using methods in the Player class, as SuperMushroom and PowerStar provides the Player with a specific buff. ConsumeAction has an attribute of

Consumable as only objects that implement Consumable would be able to be targeted by ConsumeAction.

REQ5: Trading

Player and Toad class both extend the Actor abstract class. Toad has dependency on TradeAction as it creates TradeAction instances in its method to show that it allows the Player to Trade with it. TradeAction has an association with Tradable interface as it has an attribute of an Tradable item (Item object that implements Tradable) to trade with the Player. TradeAction also has a dependency with Player as it checks the remaining money on the Player in its attribute to check if Player has enough balance to perform TradeAction.

REQ7: Reset Game

The abstract Tree class, abstract Enemy class, Player and Coin class implements the Resettable interface class. This is because the instances of the Enemy class will be destroyed (killed), the Tree instances have a chance of converting into dirt, the Player's health and status will reset and the Coin instances will be removed. Therefore, all these instances implementing Resettable interface will be added to the ResetManager. Thus, when the ResetAction, an extension of Action abstract class, is executed, it can use the ResetManager to know what instances are to be resetted.