Design Rationale

Requirement 1:

- For the first requirement we will add an enumeration class that will represent the tree status, Sprout(+), Sapling(t), and Mature (T).
- For the Wall and Tree class, they both are an abstraction of the ground abstract class.
- The abstract class is from the engine package.

Requirement 2:

- We will be creating a JumpingAction class that represents the super jump that the player will have.
- JumpingAction is an abstraction of the abstract class Action, as we assume that the jumping action is one of the actions that the user can insert, just like moving from left to right up or down.
- The JumpingAction will have a dependency that goes from the class JumpingAction to both classes of tree and wall; because the class JumpingAction increases the success of overcoming these classes when they are represented as obstacles in the game environment.
- Wall is represented as (#) , while the tree is represented as either (+,t,T).

Requirement 3:

- We assumed that to create an enemy, Koopa, should be an abstraction of the abstract class Actor.
- Koopa should implement some behaviours from the behaviours interface, because koopa has wander behaviour and attack behaviour that implement the behaviour interface.
- The Koopa class has a dependency with the DropItem from the engine package, because Koopa will drop a coin when it's eliminated from the game.
- The Goomba enemy class has an association with the IntrinsicWeapon, because it possesses a kick as its intrinsic weapon given in the engine package.
- The Koopa enemy class has an association with the IntrinsicWeapon, because it possesses a punch as its intrinsic weapon.

Requirement 4:

- To implement the magical item feature, we will be creating a MagicalItem abstract class in the game package that is considered as the abstraction of the Item class in the engine package.
- This MagicalItem class will be an abstract class that has two classes that are abstracted from it, SuperMushroom and PowerStar classes.
- The MagicalItem also has a dependency with the Player class, because it will change some of the player’s attributes when both are interacted with each other. Hence, it knows about the players attributes and is able to increase their value.

Requirement 5:

- We will create a class the represents the coin in the game,
- We assumed that coin is introduced to the game in two ways, one of which will be through the tree's when the tree has a status of sapling(t) or from the destroyed high grounds. Hence, an association is created between the tree and the coin class, and ground and the coin class.
- Since requirement 5 has a trade theme in item we will be creating a trading interface.
- The Player class should be able to implement all of the trading methods.
- We will also create a new class that represents the wallet in the game that will be used as an inventory for the coins and it will have a dependency with trade, because we assume that the player won't be able to trade without a wallet in hand.
- The trade interface will have a dependency with both of the MagicalItems that we will create inside the game package and the WeaponItem class that is already created inside the engine package.

Requirement 6:

- To create the monologue feature, we assume that we must start it with creating a Toad class that will represent the toad NPC in the game.
- Toad class is the abstraction of the Actor class because he is an actor in the game.
- We also created a SpeakAction Class that will represent the action of speaking from both toad and the player, however the player's speaking action is dependent on the player location in regards to toad.
- The player will be capable of purchasing a wrench or a power star when it speaks to a toad requesting one of the two things.

Requirement 7:

- Resetting the game requires the changes in all of the action options that are available in the menu for the player, and the ground related to that stage.
- As for the ground, the ground will be resetted to the original I/O console.
- When the game is reset, the map will be restored to a randomised state (the initial state of the game) as created by the algorithm.