Design Rationale

Requirement 1:

Assignment 1:

For requirement 1 Sprout, Sapling and Mature classes were created, and 3 of these classes inherits the Tree class as they share the same methods and attributes with each other. This is because the 3 classes have their own different functionality. For example, Sprout takes 10 turns to grow into a Sapling while Sapling also shares a similar functionality of taking 10 turns to grow into a Mature. This differs for Mature as it takes 5 turns to grow a new Sprout of the surrounding fertile squares, and it has a chance to become Dirt.

A method will be created to have a chance to drop objects, which are Goomba, Coin and Koopa objects for Sprout, Sapling and Mature respectively. Therefore, 3 of these classes share the same functionality of dropping objects.

Assignment 2:

For Sprout, Sapling and Mature classes, the tick method was created for these classes to experience the flow of time, or it can also represent each turn of these classes. For example, Sprout and Sapling can grow where Mature has a 20% chance of withering. The ability to spawn items or enemies was also implemented where Sprout can spawn a Goomba, Sapling can spawn a coin and Mature can spawn a Koopa on each of their turn.

Requirement 2:

Assignment 1:

For requirement 2, a “JumpAction” class will be created. This JumpAction class inherits from the Action abstract class. This JumpAction class will be dependent on the Ground abstract class because the damage and the success rate done to the player depends on the type of Ground that the player jumps onto. For example, the Wall object has an 80% success rate with 20 fall damage, while the Dir and Floor will have a 100% success rate with no fall damage. However, this JumpAction class also depends on the SuperMushroom item as if the player has consumed a SuperMushroom, the player has a 100% success rate of jumping to the higher ground without taking any damage. In the World class, it contains the position of the Actor which indicates if the JumpAction is required by the Player. MoveActorAction class depends on the Actor’s position to indicate if the Actor is moving to a lower ground.

Assignment 2:

The canActorEnter method and allowableAction method was added to determine if the actors can jump onto these Grounds where it depends on the fall damage, the jump success rate, the type of actor and the status of the actors. For example, Sprouts, Saplings, Mature and Walls each have their own success rate and fall damage. Some conditions were also implemented for these classes where if the actor has consumed the SuperMushroom or PowerStar, it has a different success rate and fall damage for the actors.

Requirement 3:

Assignment 1:

For requirement 3, Goomba and Koopa classes have been created, and two of these classes inherit the Enemy class, which also inherits the Actor class. The Enemy class was created as two of these enemies share the same attributes in terms of HP, and behaviours. The AttackAction class is used by Goomba and Koopa to target the Player, while the Player uses the AttackAction class to target the Goomba and Koopa.

Assignment 2:

Goomba and Koopa have been created, Those 2 inherit Enemy Class which inherits Actor class. The Enemy class was created as two of these enemies share the same attributes in terms of HP, and behaviours.The AttackAction class is used by Goomba and Koopa to target the Player, while the Player uses the AttackAction class to target the Goomba and Koopa. If the Koopa is killed it will become dormant and turn into a shell.

Requirement 4:

Assignment 1:

The Items, Power Star and Super Mushroom, will spawn on the ground the same way we spawn the player. When the player walks on them he will consume them giving him certain power ups. When consuming the Super Mushroom the player Hp will increase and the player will have perfect Jumps, you lose these effects when you get hit.when consuming the Power Star he player will become invincible, will no longer need to jump and all the ground he walks on will become coins, you lose these effects after a certain amount of time.

Assignment 2:

The Items, Power Star and Super Mushroom, will spawn on the ground using .addItem(). When the player walks on them he will be prompt to consume them giving him certain power ups. When consuming the Super Mushroom the player Hp will increase and the player will have perfect Jumps, you lose these effects when you get hit.when consuming the Power Star he player will become invincible, will no longer need to jump and all the high ground he walks on will become coins, you lose these effects after 10 turns.

Requirement 5:

Assignment 1:

A Wallet class was created to store the Coin objects for the Player to purchase items. Using these Coins, players can buy items such as Super Mushroom, Power Star and Wrench from the Toad. The Coin class also has a dependency towards Sapling and Ground as Saplings and destroyed Grounds will also randomly spawn Coins.

Assignment 2:

A Wallet class was created to store the Coin objects for the Player to purchase items. Using these Coins, players can buy items such as Super Mushroom, Power Star and Wrench from the Toad. The Coin class also has a dependency towards Sapling and Ground as Saplings and destroyed Grounds will also spawn Coins.If player is near a Toad actor the game will promt the actor to buy a super mushroom or wrench or power star.

Requirement 6:

Assignment 1:

A SpeakAction class will be created for the Player to interact with the Toad. Therefore, SpeakAction will be dependent on the Toad. SpeakAction will also be dependent on Wrench and PowerStar as the monologue generated by the Toad depends on the presence of the Wrench and PowerStar.

Assignment 2:

A SpeakAction class will be created for player to interact with Toad. Therefore, SpeakAction will be dependent on the Toad. SpeakAction will also depend on Monologue Class which has all the Monologues that Toad can say. Monologue Class depends on Wrench and Power Star to remove certain lines from Monologue list.

Requirement 7:

Assignment 1:
Resettable will have specific methods for the coin class as it must be removed from the ground when resetting. Tree will also be changed to dirt when the game is resetting. Class inheriting enemies will also be removed from the game when it is resetting. So for all the class that is being affected when resetting, it must realise the resettable interface. The resetAction inherits Action abstract class and should be only used once so it has a one to one relationship with ResetManager.

Assignment 2:

Resettable will be implemented for sprout, sapling, mature, player, and wall. Player class can cover reset HP, and remove status such as TALL and INVINCIBLE. Sprout, sapling, and mature are all trees which will have a 50% chance to wither. Ground will never have coins on it so we don't need to implement resettable. ResetManager class will be used by ResetAction class to help reset all instances of the objects above to their reset state. Status.RESET capability will also be removed so player will not be able to reset.

Assignment 3

Requirement 1:

For requirement 1, a new gameMap that consists of strings is created, with a new groundFactory that is responsible for creating all the suitable grounds that should be in the new Map called “Lava Zone”. The new gameMap is then added into the World to be displayed. For Mario to be able to teleport to the new gameMap, some WarpPipes that are guarded with the PiranhaPlant are created where if Mario kills the PiranhaPlants and jumps onto the WarpPipe, Mario can proceed to the new gameMap. This is done by setting some exits at the location of the WarpPipe.

Requirement 2:

For requirement 2, some new classes that extend Actor are created. Princess Peach extends the Actor class, while Bowser, Piranha Plant, and Flying Koopa extend the Enemies abstract class as they are capable of attacking Mario. All the enemies are created as Goomba and Koopa, where their hitpoints, damage and hit rate are defined in the class. Where all the enemies have the capability to attack Mario, all these enemies have an attack behaviour. However, as only Piranha Plant cannot move around, Piranha Plant does not have a follow behaviour where all the other enemies do. Bowser also can drop a Key item when it is defeated, and the Key can later be used to interact with Princess Peach.

Requirement 3:

Two fountains are created which are Health Fountain and Power Fountain as grounds. The two fountains are placed in the game by the setGround method at a desired location. When Mario stands on a fountain and if Mario has a Bottle item, it can fill the bottle depending on the type of fountain that Mario is standing on. For example, Healing water can be added into the bottle when Mario fills from the Health Fountain, and Power water when Mario fills from the Power Fountain. This is done by creating a new list for the Bottle, and Mario can consume the water from the bottle from the last of the list of the Bottle item. The type of water also grants different perks for Mario, where a consumeAction class is created to check for the last water in the bottle. The Healing water increases Mario hit points by 50, and Power Water increases Mario attack damage by 15.

Requirement 4:

A FireFlower class is created as an Item as Mario is able to consume the Fire Flower to gain a special perk. A FireFlower item will be added to the ground of the Sprout or Sapling if either of these grounds grow to the next stage. When Mario stands on the ground that contains the FireFlower item, Mario is able to consume the FireFlower directly. After Mario has consumed the FireFlower, a method then checks if the Mario has the capability of a FireFlower. If Mario has that capability, a timer of 20 turns starts to count. After 20 turns, the capability will be removed. As Mario attacks an enemy, a Fire item will be added to the ground where if any actor stands on the fire, it will be damaged for 20. However, the fire only lasts three turns.

Requirement 5:

Each of the listed characters in this requirement can speak anywhere on the map at every second turn. Therefore, a timer for each of these characters' classes is created to count if the turn is even or not. If the turn of that character is even, a random dialogue will be printed out on the console.