Group 5

Goh Kai Yuan (30881919)
Jonathan Koh Tong (32023146)
Ting Yi Xuan (32577346)

REQ 1

First of all, I set my Tree class to abstract class and I created 3 children classes (Sprout, Sampling and Mature) which extends from Tree class. Next, I created two new classes which are Koopa class and Coin class. Lastly, I create an abstract class called Wallet Class

In Tree Class:

- Import java.util.Random

In Sprout Class:

- Generate a random number and use the if else statement to check whether we need to spawn Goomba in this turn or not. (e.g. if(int random.nextInt(100) < 10){ Goomba.addGoomba();})
- Add Goomba by using add method in ActorLocationIterator and set a new instance of Goomba (In every turn, it has a 10% chance to spawn Goomba on its position)
- Use the if-else statement and isAnActorAt method in ActorLocationsIterator class to check whether any actor stands on it, if any actor stands on it, it cannot spawn Goomba.

In Sapling Class:

- Generate a random number and use the if-else statement to check whether we need to drop coins in this turn or not. (e.g. if(int random.nextInt(100) < 10){coin.addCoin();})
- Add Coin by using addCoins method in Coin class and increase the coins amount (set attribute int wallet in Player Class) in player's wallet balance if player picks up the coins (In every turn, it has a 10% chance to drop coin on its position)
- Use the if else statement and isAnActorAt method in ActorLocationsIterator class to check whether any actor stands on it, if any actor stands on it, player will pick up the coins

In Mature Class:

- Generate a random number and use the if else statement to check whether we need to spawn Koopa in this turn or not. (e.g. if(int random.nextInt(100) < 15){Koopa.addKoopa();})
- Add Koopa by using add method in ActorLocationIterator and set a new instance of Koopa (In every turn, it has a 15% chance to spawn Koopa on its position)
- Use the if else statement and isAnActorAt method in ActorLocationsIterator class to check whether any actor stands on it, if any actor stands on it, it cannot spawn Koopa.

In Goomba Class:

- Attributes: Location location
- Check it's position in the GameMap by using isAnActorAt method in ActorLocationsIterator class

In Koopa Class:

- Attributes: Location location
- Check it's position in the GameMap by using isAnActorAt method in ActorLocationsIterator class

In Coin Class

- Attributes: int amount, Location location
- addCoin(amount, location) (method): Create coins ($20) on its position, the coins will increase if the player picks up the coin.

In Wallet Class:

- Attributes: int money
- Use setMoney(money) to increase our player's wallet money by picking up coins from the ground
- Extends by Player class and Coin class

In Player Class:

- Check it's position in the GameMap by using isAnActorAt method in ActorLocationsIterator class

In Application Class:

- There are some sprouts there in the game map. If we need to add more sprouts, we can manually put "+" into the game map or set a new instance of Sapling and add it in List<String> map

- Set a new instance of Sapling (It will grow a sapling tree in every 10 turns) and add it in List<String> map (same position). We have to use tick method from Ground class to check if it's the turn for Sprout grows into a Sapling

- Set a new instance of Mature (It will grow a mature tree in every 10 turns) and add it in List<String> map (same position). We have to use tick method from Ground class to check if it's the turn for Sapling grows into a Mature

- Use the if-else statement to check whether there is any Dirt surrounding the specific Mature in the map, if there is no Dirt, it will stop growing sprouts.
    - Set a new instance of Sprout (For every 5 turns, It can grow a new sprout in one of the surrounding fertile squares, randomly) and add it in List<String> map (same position). We have to use the tick method from Ground class to check if it's the turn for Mature to grow a new sprout in one of the surrounding fertile squares, randomly.

- Generate a random number and use the if-else statement to check whether we need to turn mature to dirt in this turn or not.
    - Set a new instance of Dirt (It has 20% to wither and die (becomes Dirt) in every turn and add it in List<String> map (same position).

- Add Goomba, Koopa and coin into the map list by using add() in ActorLocationsIterator, addCoin()

REQ 2

Firstly, I created a JumpAction class which extends from the Action class (abstract class) in the engine file and a SuperMushroom class.

In JumpAction class:

- I create five methods which are isADirtAt(location), isASproutAt(location), isASaplingAt(location), isAMatureAt(location), isAWallAt(location) in their respective classes (Dirt class, Sprout class, Sapling class, Mature class and Wall class)

- Use the if-else statement, the methods I mentioned above to check which type of high ground. Also, we will use isAnActorAt(location) to check whether the actor is standing next to the high ground.

    - If it's a wall, the player will have a 80% success rate to jump up to the wall and if the player fails to jump up, he will receive 20 fall damage which means the player's hitPoints will reduce 20 and the player will stay on the current ground

    - If it's a sprout, the actor will have a 90% success rate to jump up to the sprout and if the player fails to jump up, he will receive 10 fall damage which means the player's hitPoints will reduce 10 and the player will stay on the current ground

    - If it's a sapling, the player will have an 80% success rate to jump up to the sapling and if the player fails to jump up, he will receive 20 fall damage which means the player's hitPoints will reduce 20 and the player will stay on the current ground

    - If it's a mature, the player will have a 70% success rate to jump up to the mature and if the player fails to jump up, he will receive 30 fall damage which means the player's hitPoints will reduce 30 and the player will stay on the current ground

    - If the player consumes Super Mushroom, he can jump freely with a 100% success rate and no fall damage. We will use the hasCapability method from Actor class to check whether the player has consumed Super Mushroom or not.

    - All of the probability calculations will use random.nextInt(100) to try.

- It will print the coordinates of the current high ground (e.g. Mature(20,4)). I will create a new method called getHighGroundAt(type, location) in Wall class and Tree class so that I can get the type and location of the high ground.

- If the player jumps to the high ground successfully, it will put the player on the target high ground by using the MoveActorAction constructor from MoveActorAction class and print a message (e.g. Player jumped up successfully. Current location Wall(8, 9))

- If the player jumps to the high ground unsuccessfully, The player will stay on the remaining ground and we will use the hurt(damage received) method from actor class to reduce the actor's hitPoints. Lastly, we will print a message (e.g. Player failed to jump up. Damage Received: 20 (from the Wall). Current location Dirt(8, 8))

In Player class:

- Only Player Class can use jump action so we will implement the JumpAction method in Player Class.
- Use hurt method from actor class if the player failed to jump up to the high ground to reduce the player's hitpoints
- Koopa and Goomba Classes won't use the JumpAction method because enemies can't jump to high ground.
- After a successful jump, if the player wants to jump to other high grounds, I will execute JumpAction again.
- All actors can walk down to lower ground by using the MoveActorAction constructor from MoveActorAction class

REQ 3

Firstly, the player and enemies (Goomba and Koopa) will be created through the Actor abstract class. Goombas will be spawned from sprouts therefore Sprout class has an association relationship with Goomba class. Status enum class is to indicate the current status of the Actors. Player needs a Wrench to destroy Koopa's shell, therefore PLayer class has an association relationship with Wrench class and Wrench class is extends from WeaponItem abstract class.Since players and enemies can attack and have their own type of way to attack, therefore the AttackAction class has an association relationship with Actor abstract class. Since all enemies have their own behaviours, therefore a Behaviour interface is created and is implemented by AttackBehaviour and WanderingBehaviour class. Whenever a fight is engaged between the player and the enemy, the enemy will follow the player, therefore the FollowBehaviour class is added and it implements Behaviour interface since it is part of the enemies' behaviour. Since player and enemies will also move around, therefore MoveActorAction class is created to enable player and enemies to move around. Therefore, Player class and WanderingBehaviour class will have dependency relationships with MoveActorAction class. Since the location will need to be updated when player and enemies move around, therefore MoveActorAction class will have a dependency relationship with Location class

If Goomba is unconscious, it will be removed from the map, therefore no additional class is needed. However, when Koompa is unconscious, it will go into dormant state, therefore the display method will display D instead of K, and the rest of the method will change accordingly when isConscious becomes false. When the shell of Koopa is destroyed, it will drop SuperMushroom, therefore Koopa class knows about SuperMushroom class and SuperMushroom class will have dependency relationship with Player class since it will update attributes of Player class.

REQ 4

SuperMushroom class and PowerStar class is created from Item abstract class and will extend Player class because both will have their unique way of changing the Player's attributes. Both SuperMushroom and PowerStar will have an association relationship with RemoveMagicalItem abstract class because both magical items will be removed once consumed by the Player, however, unlike SuperMushroom, PowerStar will also be removed from the map after 10 turns.

SuperMushroom class has an association relationship with JumpAction class because the Player can jump freely and no fall damage will be applied to the player. SuperMushroom class also has an association relationship with AttackBehaviour as the SuperMushroom effect will wear off once it is attacked by an enemy.

PowerStar class has an association with AttackAction class because the Player will instantly kill the enemy if an attack is successful. PowerStar class also has an association with AttackBehaviour class because the enemies' attack will deal 0 damage to the Player that has consumed PowerStar as the Player will become invincible for 10 turns. PowerStar then has dependency relationships with Tree class and Wall class as once a Player that has consumed PowerStar steps on higher grounds like Tree and Wall, it will automatically convert them into Dirt, therefore Tree class and Wall class has an association relationship with Dirt class. Dirt class has a dependency relationship with Player class because once a new Dirt is created, the Player's total coin amount will be increased.

REQ 5

Firstly, a toad is always present in the middle of the map and that we can hardcoded the toad to be always in the middle of the map. With that, the toad has to be surrounded by walls and that we need to call our Ground abstract class where later we can access to Wall class. This means that any Actor class cannot have access to the trader Toad, unless the wall has been broken down or the actor has the capability to jump on the wall. If the actor/player can break the wall, it means that the wall part will leave a dirt after actor/player walk away from the position. If a player jump on to the wall, then the wall is still present. JumpAction class is created and this class will determine if the player is able to jump onto the wall or no.

I will make my Item class as an abstract class as it will be extended to three other classes. These three other classes will be Wrench, SuperMushroom and PowerStar. Three of these items can be “picked up” or bought from the toad trader. However, SuperMushroom and PowerStar cannot be dropped. Therefore, once a player has picked up those items, the portable attributes in Item class will be set to false so that it is not droppable.

SuperMushroom can be consumed anytime but PowerStar will be fading over time. With that, we have to use tick method from Item class to check if the PowerStar will be faded over time.

REQ 6

Firstly, Toad will be in its wall. Therefore, toad class and ground class will be used. Player class has to implemented to talk with the Toad and 4 monologues will be chose from to be output onto the console.

A new class called Monologue will be created to hard code the 4 sentences that will be output when player interacts with the Toad. A MessageAction class is created so that we can display the message to the user.

Wrench class will be used as the monologue will prevent us from getting the first monologue when a player has a wrench and talk to the toad.

Likewise, SuperMushroom will be used as well. With the presence of SuperMushroom in the Inventory of the player, the second monologue will be prevented from displaying.

REQ 7

- Generate a random number and use the if else statement to check whether we need to turn mature to dirt in this turn or not.
    - Set a new instance of Dirt (It has 50% to wither and die (becomes Dirt) in every turn and add it in List<String> map (same position).
- Use remove method from ActorLocationsIterator to remove all enemies (Goomba and Koopa) in the game map.
- Use resetMaxHp method from Actor class to heal the player to maximum
- Add a method removeCoin(location) and use a for-loop to check through the game map. If there is a coin, use removeCoin(location) method to remove it.
- Using the removeCapability methods from actor, we can remove the capabilities that have in the actor.