Questions:

5) Must TOAD also have a wallet?

REQ1

First, we want to enrich the world with some greens (World 1-1). A Tree has three stages/cycles and each stage has a unique spawning ability. It takes 10 turns to grow into the next stage.

* Sprout (+)
  + It has a 10% chance to spawn Goomba on its position in every turn. If any actor stands on it, it cannot spawn Goomba.
  + It takes 10 turns to grow into a small tree/Sapling (t)

Sprout extends Tree class – a type of Ground

It has 10% chance to spawn Goomba on its position in every turn

* + For every turn in ground.tick() method, using probabilities to see if should spawn goomba
    - * public boolean getBiasedRandom(int bias) {
      * int c;
      * Random t = new Random();
      * // random integers in [0, 100]
      * c=t.nextInt(100);
      * if (c>bias){return false;
      * }
      * else{return true;}
      * }
  + HOW TO SPAWN? (SPAWN IN TICK() method)
    - Spawn on its position (replace sprout by Goomba)
      * Get location of ground (the argument given in the tick() function)
      * Location.addActor(new Goomba());
  + When an actor stands on it, cannot spawn a Goomba
    - In ground.tick() method, check if the current location has an actor. (hasAcActor method)

Takes 10 turns to grow into a small tree/sapling (t)

* Add attribute of numTurns
* In tick() method increment numTurns & once 10 turns detected, destroy sprout & instantiate sapling
* Growable <<interface>>
* Have a grow() method in interface that c

**OR**

Tree class has attribute: “stage” of enum TreeCycleStage, which represents the tree’s current stage of the cycle.

Create enum class TreeCycleStage.

- has methods that are implemented based on the Tree’s “stage” attribute (ie: sproutTick(), saplingTick(), matureTick() to be called based on stage attribute in overriden ground.tick() method)

- modify Tree constructor such that Tree when instantiated starts with a display char of “+”.

- When instantiated, it is a sprout

SPROUT (+)

It has 10% chance to spawn Goomba on its position in every turn

* + For every turn in ground.tick() method, using probabilities to see if should spawn goomba
    - * public boolean getBiasedRandom(int bias) {
      * int c;
      * Random t = new Random();
      * // random integers in [0, 100]
      * c=t.nextInt(100);
      * if (c>bias){return false;
      * }
      * else{return true;}
      * }
  + HOW TO SPAWN? (SPAWN IN TICK() method)
    - Spawn on its position **(ASK!! replace tree sprout by Goomba? OR can multiple things be on the same GROUND location??)**
      * Get location of ground (the argument given in the tick() function)
      * Location.addActor(new Goomba());
  + When an actor stands on it, cannot spawn a Goomba
    - In ground.tick() method, check if the current location has an actor. (hasAcActor method)

Takes 10 turns to grow into a SAPLING (t)

* + Add attribute of turnsPlayed in tree class
  + In tick() method increment numTurns & once 10 turns detected, setDisplayChar(“t”); & setStage() to set stage attribute of Tree to be a SAPLING.
  + Once 20 turns detected, setDisplayChar(“T”); & setStage() to set stage attribute of Tree to be a MATURE & resetNumTurns to 0.

SAPLING (t)

It has a 10% chance to drop a coin ($20) on its location in every turn.

* + For every turn in ground.tick() method, using probabilities to see if should drop a coin on the location.
  + HOW TO SPAWN? (SPAWN IN TICK() method)
    - Spawn on its position **(ASK!! replace tree sprout by Koopa? OR can multiple things be on the same GROUND location??)**
      * Get location of ground (the argument given in the tick() function)
      * Location.addActor(new Koopa());

It takes another 10 turns to grow into a tall tree/Mature(T)

* + In tick() method increment numTurns & once 10 turns detected, setDisplayChar(“T”); & setStage() to set stage attribute of Tree to be a MATURE & resetNumTurns to 0.

MATURE (T):

It has a 15% chance to spawn Koopa in every turn. If an actor stands on it, it cannot spawn Koopa.

* + For every turn in ground.tick() method, using probabilities to see if should spawn a Koopa.
  + HOW TO SPAWN A COIN? (drop in TICK() method)
    - Drop coin on top of the ground its own
      * Get location of ground (the argument given in the tick() function)
      * Location.addItem(new Coin());
  + When an actor stands on it, cannot spawn a Koopa
    - In ground.tick() method, check if the current location has an actor. (hasAcActor method)

For every 5 turns, it can grow a new sprout (+) in one of the fertile squares randomly. If there is no available fertile square, it will **stop spawning sprouts (ASK: STOP SPAWNING FOR EVERY 5 TURNS AFTER IT??)**. At the moment, the only fertile ground is Dirt.

* + For every turn in ground.tick() method:
    - Increment numturns by 1 & check of numturns%5 == 0. If yes, GROW A NEW SPROUT (tree with stage “SPROUT”) in one of the fertile squares randomly.
      * HOW TO FIND FERTILE SQUARES (DIRT) in game map?
    - Else do nothing

It has 20% to wither and die (becomes Dirt) in every turn.

* + For every turn in ground.tick() method, using probabilities to see if should WITHER & DIE (turn into Dirt)
    - Get location of ground
    - Location.setGround(new Dirt());

REQ2

REQ 5

**UML Diagram**

Diagram

Description automatically generated

**Justifications and further explanation of classes & methods.**

Approaches:

For the Wallet System –

**1.** have a wallet item class for **only players**, this wallet item would be a subclass of the item abstract class. This wallet would have allowable actions tied to it relating to being purchased.

The wallet shall:

- not be portable,

- be added to **player's** inventory in **player's** constructor

- contain Coins, meaning that Coins are added to the Wallet item **instead of** the player’s own inventory when it is picked up by the player

- Attributes:

private int totalBalance,

private ArrayList<Coin>,  
- Constructor:

create 3 instances of tradeAction (with trade amount),

one for each of wrench, SuperMushroom & Powerstar

add these tradeActions to the list of allowable actions for the Wallet item

- Methods:

private getBalance() to return the balance

private addCoin() to add coin to its arraylist

private useCoins() to use coins to purchase an item

2. have a

WALLET

PLAYER  
1. have a wallet instance which references the wallet  
2. have a getWallet() instance to return the wallet item

COIN  
1. can be picked up from ground but cannot be dropped - so override abstract item class's getDropItemAction method,  
2. when picked up from ground & added to inventory it must be instead added to the wallet INSTEAD of the actor's inventory, so modify item class's getPickItemAction() method such that it instantiates a obj of addToWalletAction class instead & returns it  
3. addToWalletAction extends pickUpItemAction class just that the EXECUTE function needs to be changed to access the wallet & add the coin item into the actor's wallet item  
2. attributes  
- int value

TRADEACTION  
1. attrubutes:  
- item itemToTrade  
- Toad toad  
2. methods:  
- execute method:  
    uses player, its wallet, and Toad -> to trade itemToTrade with Toad,  
    reduces wallet balance of player,  
    removes coins from player's wallet arraylist,  
    if walletbalance not enough, print the not enough message in console

REQ 5

**UML Diagram –**

Diagram

Description automatically generated

## Design Rationale

**tradeAction extends Action.** All concrete classes that inherit abstract Action class has the necessary methods to execute some functionality with the interaction execute. This allows for the action to purchase an item to be done. This creates an is-a relationship that would allow for reusability of code & extensibility of tradeAction class.

Wallet extends Item. All concrete classes that inherit abstract Item class has the necessary methods to provide add actions (such as tradeAction) to the wallet item. This would allow an actor to use their Wallet item to perform this tradeAction action.

In this game, the ideal way to interact with the object is by attaching appropriate action to its corresponding object (aligns with the meaning of "object-oriented").

Thus, this is shown with this: **Wallet ---<<create >>---> TradeAction, and Player ---<<stores>>---> Wallet.** The Wallet is the item object that gives the Actor (i.e., the Player) an action to purchase an item.

Alternatively, we can add an attribute called totalBalance in Player class to account for the total amount he has received from collecting coins. However, doing this will require the Player to instantiate a tradeAction in its “playTurn” method in order to add it to the list of actions it can do. By doing so, this would generate an additional dependency for Player to TradeAction, and also “playTurn” would have to do added functionality such as checking if the balance is enough for purchase. This goes against the principle of “Reduce Dependency” as well as “single-responsibility”, as it would have more dependencies and the Player class would be responsible for purchasing an item, which it shouldn’t be as it shouldn’t be responsible for the functionality of the coins/items it has.

Hence, we discard this alternative. Our final design has now aligned with the Reduce Dependency Principle and Single-Responsibility Principle.

**The application checks for actions allowed for the Wallet item stored in Player’s inventory, instead of Player adding the trading action to the action list inside the Player's own playTurn() function**. Both designs are okay and do not break any object-oriented principles. We decided to create a Wallet item and add it to the Player's inventory because we ensure that the meaning of each class is clear and they can be reused and allows extensibility in other scenarios in the future (ie: if there is another actor who can store a Wallet item other than Player) (adheres to the DRY principle & SRP principle).

**Justifications and further explanation of classes & methods –**

TOAD

Singleton class - can have only one object (an instance of the class) at a time  
1. methods  
    - sayMonologue()  
2. attributes  
    - public static final arraylist of monologues

TALKWITHTOADACTION  
1. attrubutes:  
- item itemToTrade  
- String hotkey (for menu description command)  
2. methods:  
- execute method:  
    pick & print a monologue based on player's inventory,  
        if player's inventory has wrench: either sentence 2,3,4  
        if powerstar effect is there: either sentence 1, 3, 4  
        else randomly pick from 1,2,3,4  
3. create instance of TalkWithToadAction in every Ground item's construtor & add it to Ground item's allowable actions