REQ5: Trading

Overview:

To achieve this feature, there will be seven new classes (SuperMushroom, PowerStar, BuyAction, Wallet, Coins, Toad and Wrench) in the extended system and two modified existing classes (Player and Tree). The design rationale for each new or modified classes will be shown below.

Tree class

The tree class represents tree in the game. Sapling (t) will randomly spawn coins(with different integer values) to be collected by players. Tree class will not be implemented in this REQ (its implementation will be considered and focus on in REQ1. Hence the design rationale for this class can be found in the REQ1 section.

Toad class

This class acts as the vendor in the game. Toad class sells weapons (wrench) and magical items (Super Mushroom and Power Star) to players by taking in coins. It is also used to upgrade player’s attributes. Toad class extends actor as it is an actor and it is associated with BuyAction as players are allowed to purchase items from it. There is dependency between Toad class and the PowerStar, SuperMushroom and Wrench class because we need to add a new instance of those items into the player’s inventory.

Description of attributes:

- SUPER_MUSHROOM_PRICE is a public static integer attribute with value of 400 that indicates the amount of coin to be deducted from player’s wallet when player purchases the Super Mushroom
- POWER_STAR_PRICE is a public static integer attribute with value of 600 that indicates the amount of coin to be deducted from player’s wallet when player purchases the Power Star
- WRENCH_PRICE is a public static integer attribute with value of 200 that indicates the amount of coin to be deducted from player’s wallet when player purchases the Wrench

```
    private static int SUPER_MUSHROOM_PRICE=400;
    private static int POWER_STAR_PRICE=600;
    private static int WRENCH_PRICE=200;
```

Description of method:

- tradeItem method that subtract a certain amount of coins from player’s wallet and provide items to player depending on item purchased by players. If item is a Power Star, subtract 600 coins from player’s wallet; if it is a Super Mushroom, subtract 400 coins player’s wallet; if it is a Wrench, subtract 200 from player’s wallet. Wallet balance will be set to the subtracted amount and purchased item will then be added to player’s inventory.

```
public void tradeItem(Wallet wallet,Item item, int price,Actor actor){
        int currentBalance=wallet.getBalance();
        currentBalance-=price;
        wallet.setBalance(currentBalance);
        actor.addItemToInventory(item);
    }
```

Why i choose to do it that way:

Since Toad is an actor, functions in the Actor class are required and hence need to extend the Actor class. I created this class in order to enable player to use coins to trade for weapons and magical items.

Advantages:

Excessive use of literals was also prevent by declaring SUPER_MUSHROOM_PRICE, POWER_STAR_PRICE and WRENCH_PRICE as private static attribute. This prevents confusion during coding process. Furthermore, if the value of HEALED_HIT_POINTS needs to be change, changes only need to be done at one place, which is at the line where that attribute is declared instead of going through entire code and changing their values . This minimise possibilities of producing errors too. Open Close Principle can also be implemented as this class extends the Actor class, by adding functionality to the Actor class without modifying its already available functionalities, in a way that does not change the way we use existing code in the Actor class. This enables the Actor class to support new functionalities as well as being added new methods easily. For example, super() can be called set the characteristic of the toad instead of recreating constructors, getters and setters

```
public Toad(String name, char displayChar, int hitPoints) {
        super(name, displayChar, 0);
    }
```

Disadvantages:

N/A

BuyAction class

BuyAction is an action that allows the player to use coin to purchase items from the toad. It is trigger by the Toad class when players use buy action to purchase items from the toad. This class has dependency relationship with the Toad class as a new instance of Toad will be created each time this action is carried out.

Description of method:

- purchaseItem method checks if actor’s wallet has enough money, if yes, enable player to trade with the toad using the tradeItem method . Else, inform player that balance is insufficient

```
public void purchaseItem(int price,Item item,Actor actor){
        if (wallet.getBalance()>=price){
            toad.tradeItem(wallet,item, price,actor);
        }
        else{
            System.out.println("insufficient balance");
        }
    }
```

Why I choose to do it that way:

Since BuyAction is an action, functions in the Action class are required and hence need to extend the Action class. I created this class in order to enable player to purchase items from the toad.

Advantage:

This class is created using the Single Responsibility Principle where it does not need to take extra responsibility. In case of need to change responsibility, all pieces needed will be there. This makes the system easier to maintain and extend. Therefore, this class is only responsible for allowing player to trade with the Toad if wallet balance is sufficient. This is because in the future there may be more items available for player's to purchase. The Liskov Substitution Principle can also be fulfil as the initial meaning of the purchaseItem method behaviour from the Action class.

Disadvantage:

N/A

Wallet class

This class is used to store and keep track of the amount of coins players have. The Player class is associated with this class because there is a wallet attribute in the Player class constructor to ensure a wallet belongs to a particular player.

Description of method:

- getBalance and setBalance method returns the current amount of coins in player's wallet and set the new wallet balance

```
public int getBalance() {
        return balance;
    }
public void setBalance(int balance) {
        this.balance = balance;
    }
```

- receiveCoin method to add coin value into the wallet's balance

```
public void receiveCoin(Coin coin){
        int currentBalance=getBalance();
        int newBalance= currentBalance + coin.getValue();
        setBalance(newBalance);
    }
```

Why i choose to do it that way:

I created this class to keep track of the amount of coins players have and implement a getWalletBalance method to check if players balance is sufficient to purchase magical items and to increase player’s wallet balance whenever player receives coin.

Advantage:

This class is created using the Single Responsibility Principle where it does not need to take extra responsibility. In case of need to change responsibility, all pieces needed will be there. This makes the system easier to maintain and extend. Therefore, this class is only responsible to constantly being update about player’s coin amount and return the amount of coins players have if needed

Disadvantage:

N/A

Wrench class

This class is a subclass of items. It represents the weapon players use in the game to defeat enemies, such as the Koopa’s shell. It has 80% hit rate and 50 damage.

Description of method:

- updateStatus method that add capabilities (ability to attack) to the player. This method will be call when player gets a wrench.

```
public void updateStatus(Player player){
        player.addCapability(Status.HAVE_WRENCH);
    }
```

Why i choose to do it that way:

Since Wrench is an item, functions in the Item class are required and hence need to extend the Item class. I created this class in order to store information about this item, such as the hit rate and damage

Advantage:

Open Close Principle can also be implemented as this class extends the Item class, by adding functionality to the Item class without modifying its already available functionalities, in a way that does not change the way we use existing code in the Item class. This enables the Item class to support new functionalities as well as being added new methods easily. For example, the Wrench can use the addCapability method from the Item class to add capability (HAVE_WRENCH) to players.

Disadvantage:

N/A

Coin class

This class is a subclass of Items. It is currency the player uses to trade for magical items. Coins will spawn randomly from the Sapling (t). The collected coins can be traded with Toad for Wrench, Power Star or Super Mushroom. A coin has an integer value (e.g., $5, $10, $20, or even $9001).

Description of method:

- getValue and setValue method is to return the value of coin when needed to add coin into the player’s wallet and to set the value of coin upon spawning by the sapling

Why i choose to do it that way:

Since Coin is an item, functions in the Item class are required and hence need to extend the Item class. I created this class in order to store information about this item, such as the name, display character, portability and value.

A ‘value’ parameter is also added into the constructor as coins spawned will contains different values hence we need a setValue to set the value of the coin.

Advantage:

This class is created using the Separation of Concern principle where the program is separated into sections with its own responsibilities, by having well- defined concerns and as little overlapping as possible. In this class, the class is only responsible for storing and returning information about the Coin. Open Close Principle can also be implemented as this class extends the Item class, by adding functionality to the Actor class without modifying its already available functionalities, in a way that does not change the way we use existing code in the Item class. This enables the Item class to support new functionalities as well as being added new methods easily. For example, super() can be called set the characteristic of the coin instead of recreating constructors, getters and setters. Extra characteristic (value) can be added to the Coin too.

```
public Coin(String name, char displayChar, boolean portable,int value) {
        super("Coin", '$', false);
       setValue(value);
    }

public void setValue(int value) {
        this.value = value;
    }
public int getValue(){
        return this.value;
    }
```

Disadvantage:

N/A