REQ6: Monologue

Overview:
To achieve this feature, there will be two new classes (i.e., SpeakAction and Toad) created in the extended system, and three existing classes (i.e., Application and Status) will be modified. The design rationale for each new or modified class is shown on the following pages.

1) SpeakAction Class

SpeakAction class is a class allows the actor to have conversation with toad(friendly NPC) to get some useful information. It is a new class that extends Action class. In this class, there is 1 attribute which is the toad object.

Description of class attribute:

- toad is an instance of Toad class. It will used to have conversation with actor.

The SpeakAction class overrides the execute and menuDescription method from its parent class.

Description of methods:

- For the execute method, there are 4 sentences may be print out:
  1. "You might need a wrench to smash Koopa's hard shells.",
  2. "You better get back to finding the Power Stars.",
  3. "The Princess is depending on you! You are our only hope.",
  4. "Being imprisoned in these walls can drive a fungus crazy :("

  There will be an if-else statement. When actor is close to the toad:
  If the actor didn't consume power star AND have wrench (i.e., check the actor's capability), the toad will speak one sentence from that four sentences at a time randomly.
  Else if the actor consume the power star, the toad will only speak one sentence from sentences(1,3,4) at a time randomly.
  Else if the actor buy a wrench, the toad will only speak one sentence from sentences(2,3,4) at a time randomly.

- menuDescription will return a string that will appear on the command list in the console. This string states that, actor speaks to toad.

Why I choose to do it that way:
Since SpeakAction is an action and we need the functions in the Action class, and thus we let SpeakAction class extend Action class. However, since we need to ensure that when SpeakAction is called, the actor will have a conversation with toad, hence we will need to override the execute method to ensure our game logic works properly.

Besides that, I had also override the menuDescription method, the reason to do so is to let the user have a better understanding on what will happen when he/she lets the actor perform a SpeakAction to toad.

Advantage:
According to the design above, we are adhering to the Single Responsibility Principle as the SpeakAction class only focuses on the action that will be executed when the actor speak to the toad. Besides that, we fulfil the Liskov Substitution Principle as SpeakAction preserves the meaning of execute and menuDescription method behaviours from Action class.

Disadvantage:
N/A

2) Toad Class

Toad is a class that represents the NPC Toad in the game map. It is a class that extends from the Actor class. According to the assignment specification, toad is a NPC which is friendly (0 hits point) and it will have conversation with the actor when actor is beside it.

For this class I created an arraylist to store the 4 sentences of toad. By using the indexation I can display the correct sentence according to the actor's capabilities.

In addition, Toad class has overridden 2 methods from its parent class which are playTurn() and allowableActions(). Besides that, I created 3 new methods to display the conversation between toad and actor which are giveRandomTalk(), noTalkPowerStar() and noTalkWrench().

- playTurn() states the behavior of the Toad. If the toad stay at its own position(didn't move around), it return a string: toad does nothing
- allowableActions will return a SepakAction instance if the actor is currently one step beside the Toad else return an empty ActionList. By doing so, we can make sure the actor will not be able to speak to the toad if the actor is currently far away from the toad.
- giveRandomTalk will return a sentence randomly from the four sentences.
- noTalkPowerStar will return a sentence randomly from the three sentences. ("You better get back to finding the Power Stars." will be remove from the arraylist since the actor have power star buff already)
- noTalkWrench will return a sentence randomly from the three sentences. ("You might need a wrench to smash Koopa's hard shells." will be remove from the arraylist since the actor have wrench already)

Why I choose to do it that way:
In this Toad class I create a arraylist for the four sentences, rather than array I think arraylist is easier to modify and get the exactly output I want. By doing so, I can edit the arraylist content for different method, which will not have so many redundant code.

Advantage:
By creating an arraylist for all the sentences of toad instead of creating an array, I can add/remove the sentence from the arraylist easily since size of arraylist is not fixed. According to the assignment requirement, when the actor consume the power star (have power star buff), toad will not remind the actor again to get a power star, so I just remove the sentence regarding to power star from the arraylist by using arraylist.remove(). Same goes to wrench, when actor have wrench, toad will not ask the actor to get a wrench, so I just remove the sentence regarding to wrench. It fulfill the “dry”(don’t repeat yourself).

Disadvantage:
N/A

3) Application Class

Application class is the driver class of the whole game, it contains the default game map. In this class I need to initialize the new actor Toad into the game map since he stands in the middle of the map(surrounded by brick Walls) and never move around.

Why I choose to do it that way:
In this class I just initialize the instance of Toad to create a new toad object and make it appears on the game map when game is start. I can also specify its position when I add it into players.

Advantage:
By initializing the toad here, it will be easy for me to edit its position and displayChar. I can modify it anytime I want.

Disadvantage:
N/A

4) Status Class

Status is an enumeration class that gives 'buff' or 'debuff'. It is also useful to give a 'state' to abilities or actions that can be attached-detached. It is an existing

enumeration class with some additional modification. In this class, I had added two statuses, which are POWER_STAR_BUFF and HAVE_WRENCH.

Description of constant:

- POWER_STAR_BUFF indicates that the actor has consumed a power star, so the toad don't need tell the actor to find a power star.
- HVE_WRENCH indicates that the actor has bought a wrench, so the toad don't need tell the actor to get a wrench.

Why I choose to do it that way:
Instead of creating class attributes to indicate whether a particular actor has a wrench or consume power star, it is better to make use of the enumeration class to make our code easier to read. Besides that, I also can utilize the engine code provided to check the actor's capability to simplify our code

Advantage:
With this design, we will not use any more literal to specify whether a Player/Actor has a particular capability in another class as we can make use of these constants. Hence, we are adhering to the "avoid excessive use of literals" principle. This kind of implementation also provides us more flexibility for future development as we can simply add additional capability when needed.

Disadvantage:
As the game develops, more and more capability will be added to this status class. If there are too many statuses within this class, it might cause confusion when debugging.