REQ3: Enemies

Overview:
To achieve this feature, there will be three new classes (i.e., Koopa, PlayerManager and AttackShellAction) created in the extended system, and six existing classes (i.e.,Player, Goomba, Status, Floor, AttackAction and AttackBehavior) will be modified. The design rationale for each new or modified class is shown on the following pages.

1) Player

Player class is a class represents the main player(Mario) in this game. It is a class that extends from the Actor class. According to the assignment requirements, player can move around in the game map, speak to Toad(friendly NPC) and fight with enemies with weapons.

For this class I modified it with adding PlayerManager in its constructor to append a player in PlayerManager's instance, which to make sure we can access to the actor player easily in each round of game. Because we need to point out which actor is to be attacked/followed in the enemies behavior, it's hard for us if we use Player class only, so we add PlayerManager to get a player for us to determine which player is going to be the target.

2) PlayerManager

PlayerManager is a class used to return an actor player in the game for us easy to use it in the enemies behaviour, so the enemies knows who to attack and who to follow. In this class I create a player instance, by appending the player into its instance, we can get the player whenever we need by PlayerManager.getInstance().returnPlayer().

Why I choose to do it that way:
By adding PlayerManager to get a player for us to determine which player is going to be the target because we need to point out which actor is to be attacked/followed in the enemies behaviour, we couldn't do it with just the Player class.

Advantage:
By using this class, it does not required to create a new object each time they're invoked. That will nice to us since we don't need create a player each time when we need to attack/follow. It makes our code clear and logic.

Disadvantage:
N/A

3) Goomba

Goomba class is a class represents the enemies in this game. It is a class that extends from the Actor class. According to the assignment requirements, goomba can move around in the game map but cannot enter floor. Once goomba is engaged in a fight (the Player attacks the enemy or the enemy attacks player -- when the player stands in the enemy's surroundings), it will follow the Player. It causes 10 damages to player with 50% hit rate. To make sure the map is clean and not too overcrowded, goomba will has a 10% chance to suicide each round of this game.

In addition, enemies should have some behaviors: attackbehavior, followbehavior and wanderbehavior. By adding all the behaviors into behavior hashmap, we can make sure that the goomba wil attack the player automatically when player is in the enemy's surroundings. When goomba no longer able to attack/follow the player(i.e player enter floor which goomba is not allowed to), goomba will wandering around again.

Goomba class has two methods which is override from its parent class Actor: allowableActions and playTurn.

- allowableActions() is a method used to make goomba can be attacked by Player.
- playTurn() is a method used to figure out what to do next.

Why I choose to do it that way:
For goomba class, we involve a if loop in the method. Because goomba will has a 10% chance to suicide each round of this game, so by adding the if loop, a random number which less than 10 will be pick from 100 numbers, means 10% chance the goomba will be removed from the map.

Advantage:
By doing this, the game map will not being overcrowded which our game map will be clean.

Disadvantage:
It will be a possibility that all the goomba been removed from the map, so at that time the game map will not have enemies anymore.

4) Koopa

Koopa class is a class represents the enemies in this game. It is a class that extends from the Actor class. According to the assignment requirements, koopa can move around in the game map but cannot enter floor. Once koopa is engaged in a fight (the Player attacks the enemy or the enemy attacks player -- when the player stands in the enemy's surroundings), it will follow the Player. It causes 30 damages to player with 50% hit rate and koopa has the same behaviors with goomba.

But when koopa died, it will not be removed from the map directly. When koopa is not conscious(means it is defeated), it will hide inside its shell, and its character will change from ’K’ to ‘D’. Player cannot attack it anymore, and all the behaviors will removed from koopa(attack/follow/wander).

Goomba class has two methods which is override from its parent class Actor: allowableActions and playTurn.

- allowableActions() is a method used to make koopa can be attacked by Player. In this method, we have a if & else if loop inside. Not same as goomba, when koopa is not conscious(means it is defeated), it will hide inside its shell, at this time we cannot attack it with normal weapon anymore. When it hide in its shell, only wrench can break it. So when the koopa is still conscious(not defeated), we can attack it as usual until it is defeated and hide inside a shell(‘K’ -> ‘D’), at this time we need to use wrench to smash its shell.
- playTurn() is a method used to figure out what to do next. In this method we add a if loop inside to detect whether koopa is defeated. In each round of game, if koopa is defeated, the character of it will change to ‘D’, all the behaviors of koopa will be removed but remove the key in hashmap behaviors.

Why I choose to do it that way:
For koopa class, I created a class attribute isDefeated and initialized it as false at beginning. When the koopa is not conscious, I assigned true to the isDefeated, means koopa hide into its shell already, so we can easily define the condition after koopa is defeated.

Advantage:
By doing this, once the koopa is defeated, we can detected it, change its character and remove all the behaviours of it which is responsive and efficient.

Disadvantage:
N/A

5) Status

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 IM_GOOMBA and IM_KOOPA.

Description of constant:

- IM_GOOMBA indicates that the Actor goomba. By adding this capability to Goomba, we are able to add specific attack behavior to it.

- IM_KOOPA indicates that the Actor koopa. By adding this capability to Koopa, we are able to add specific attack behavior to it.

Why I choose to do it that way:
Instead of creating class attributes to indicate whether this actor is goomba or koopa, 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 this actor is goomba or koopa 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.

6) Floor

Floor class is the class used to represents the floor inside a building. It extends from Ground class. In this class, we override canActorEnter() method from its parent class. By modified this method to return a boolean, we add a if - elseif loop inside. For the actor has capability HOSTILE_TO_ENEMY(means player), he can enter floor. For the enemies who has capability IM_GOOMBA or IM_KOOPA, they cannot enter floor which make sure that the player has somewhere safe to stay.

7) AttackShellAction

AttackShellAction is a class which used to samsh the koopa’s shell. It extends from its parent class Action. When the koopa is defeated and player has capability HAVE_WRENCH, player can use and only this action to smash koopa’s shell in order to get a super mushroom.

AttackShellAction class has two methods which is override from its parent class Action: execute() and menuDescription.

- execute() is the method to perform the action (smash the shell). This method get the current location of the target first, then remove it from the map. After that create a super mushroom there and return a string: "Koopa is gone, pick up the Super Mushroom!"

- menuDescription is the method returns a descriptive string in the menu, which give player a choice to smash the shell.

Why I choose to do it that way:
Because when the Koopa is defeated, player cannot attack it as usual. Only when player is stand beside it and with a wrench, so I create an attackshell action for attacking the shell only to make sure that our code is easy to understand and clear.

Advantage:
With this design, we will not use any more literal to specify whether this actor is goomba or koopa 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.

8) AttackAction

AttackAction is a class which used by player to attack the enemies, it extends from its parent class Action class. In this class, player can attack enemies with weapon with a corresponding hit rate. If the target(enemies) no longer conscious, all the item of that target will drop and it will be removed from the map(except Koopa). I modified this class by adding a new if loop in it, if the target is not Koopa, then remove from the map. In the end, there is a string will be printed to show how much you hurt the enemies.

9) AttackBehaviour

AttackBehavior is the class which developed and use it to attack the player automatically. This class extends from its parent class Action and implements Behavior class. By adding this behavior into hashmap behavior, the enemy will attack the player automatically if player is beside it.

Why I choose to do it that way:
Instead of creating two attack method/behavior separately for Koopa and Goomba, we put two actions(kick and punch) together in one AttackBehavior class. By doing so, there will be less repeating code.

AttackBehavior class has three methods which is override from its parent class Actoion: execute(), getAction() and menuDescription().

- execute() is the method to perform the action(kick or punch). In this method an if - else if loop will be use to determine whether the actor is Goomba(kick) or Koopa(punch).
- getAction() is the method use to attack the player automatically. When the player is at enemies' surrounding, will return this attackBehavior to attack player.
- menuDescription() is the method returns a descriptive string in the menu, but for this class enemies will attack player automatically, so an empty string will be returned.

Advantage:
With this design, we will not use any more classes to specify whether this actor is goomba or koopa as we can make use of these capabilities. 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 actions when needed.

Disadvantage:
N/A