Rocket Class

A new class named Rocket is created. This class inherits Item class which allows rocket object item to be created anywhere else in system. This class has a constructor that accepts a string to represent the rocket item name, and also a displayChar, a character to represent to the Rocket item when they are outputted to the main User interface. We chose to create a separate Rocket class and creating a rocket object items through it, rather than creating a normal Item object item because it helps to set apart these type of items from other items. It is Important because throughout the application, different Rocket object items are needed to coordinate actions across the system and to achieve the goal in this game. 3 Rocket object item were created through Rocket class, which are Rocket plans, Rocket Body and Rocket Engine. Each importance of the Rocket object will be explained in other classes.

This Class has no method because we can reuse all the method from Item class through inheritance. We could choose to copy-paste all the methods items and modify some variables to make them part of Rocket’s class method but this would violate the design principle “Don’t Repeat Yourself”. Our Main objective here is to keep the code from duplicating as much as possible and adhere to the design principle. The Player’s item inventory can be easily accessed to check for the existence of the Rocket items just by using Rocket object instances.

Key Class

A New Class called Key is created to represent the Key type Item. The concept of creating and implementing this class is similar as Rocket class. Since Key class is a sub class of Item, all the Item class method can be reused.

This class contain only one method called newInventoryItem. This method is originally from Item class, but we choose to override this method from the superclass to give it a new behaviour. Initially, this method return an Item type item but through overriding, this method now return a Key type item object. This helps verification purpose to be carried out more efficiently, such as when unlocking a door, it is better to search for a Key type object instances rather than an item type object instances.

Key object instances are declared in application itself and added to each related enemies. This ensure easy testing and modification of code throughout the implementation.

Inheriting Item class also this enable the object of this class to be placed inside the list of Item class type named inventory in Actor.java.

Door Class

This class contains a constructor which determine the symbol that represents a door.

We decided to use ‘|’ character to show a door to allow easier readability for the user of this system.

This class inherited Ground class to overwrite the canActorEnter method to determine whether an actor can enter the room by checking the existence of Key object instance in the Actor’s item inventory list. If the Key instance exists, this method will return a Boolean value of True and allow the actor to enter. If the actor does not have any key in the inventory, Boolean value of False will be returned and actor will not be able to enter. Since new types of enemy are created for this system, and enemies holding a key is also part of the specification, that means that any actor that posses the key can enter a locked room, not player alone can enter.

Goon Class

This class represent one of the new enemies called Goon. The implementation and designing of this class is very similar to Grunt class that was given to us in the game package.

This class inherits Actor Class to allow efficient creation of Goon object instance through the constructor, because it was already shown to us the way to set name, a display character, priority and hit points of an actor just by reusing the same constructor.

A method called getIntrinsicWeapon from the Actor class is reused and overridden to modify the damage that goon can deal. Grunt can deal a damage of 5, since Goon has twice the damage of Grunt, goon can deal a damage of 10 through invoking the method.

Goon also has a FollowBehaviour which allows it to follow the player around the map. We also reused the FollowBehaviour class given initially. Main objective is to also keep the duplication of code to the minimum.

Goon has another method called Insult. Every turn it has a 10% chance to shout an insult to the player. We chose to create this as a method rather than a class to reduce dependency of Goon class to another class. Since it was one of the first class that we implemented, we chose to keep Goon’s actions inside one class to avoid errors that are hard to be detected

The playTurn method is similar as the one in Grunt class. Since a key should be dropped from Goon once it is defeated, we removed the DropItemAction from the actions list of Goon. This is because by default, the engine had set a way for actors to carry out random allowable action. To prevent Goon from randomly dropping items while moving, it was the best option to remove it rather than coming up a new way to stop goon from dropping item randomly.

MiniBoss Class

MiniBoss is also a new class to represent one of the new type of enemies. This class inherits Actor class. The methods from Actor will be reused to set the name, symbol displayed on the map, priority, hit point and maximum hit point of Miniboss.

MiniBoss has half of the damage and hit points of Grunt. We reused the getIntrinsicWeapon to return a new damage for MiniBoss (same way as Goon Class). Halves of the damage of Grunt is 2.5, but this method only accepts Integer value. This is why we rounded off the damage for MiniBoss to 3. Hit points of MiniBoss is changed through the constructor, no extra method is needed.

We added a StayStill behaviour to MiniBoss because it should remains at the same position throughout the entire game. How the StayStill behaviour works will be explained more clearly under StayStill class

The method distance is to calculate the distance between MiniBoss and player.

The method near is to check whether player is 1 step away from Miniboss. We could have just coded the literal “1” in the if else statement (for distance check) in playTurn, but this would mean using literals and violate the design principle “Avoid excessive use of literals”. This is why we decided to create a new method called near for validation.

In this context, (while also naming a method “near”) near is considered 1 step away.

The playTurn method in MiniBoss is significantly different from other actors. We do not want MiniBoss to do anything unless a player is near. When player is not near Miniboss, it will stick to its StayStill behaviour. But when a player is near, the allowable action will be returned to Miniboss randomly. Knowing that, we had created another method called returnValidAction, that skips through MoveActorAction, SkipTurnAction and also DropItemAction. This is to ensure MiniBoss doesn’t move and drop item randomly, and only attack when the player is near.

MiniBoss has a Rocket Engine. It will only be dropped when it is defeated. We hardcoded MiniBoss inside a locked room so that player would not be able to access it without having a key.

In this class, we tried to adhere to the design principle “Declare things in the tightest possible scope”. Such example is shown in method playTurn by declaring local variables rather than attributes.

StayStill Class

This class is a new behaviour that allow an actor to stay at the same position for the entire game, unless overridden by another action. This class implements ActionFactory. We override the getAction method in ActionFactory and returns a new instantiated SkipClassAction, which override all the actor actions and does anything. Main objective is to keep the code simple while achieving the desired results, without coding a bunch of unnecessary code.

Talk Class

This class represent a new action called Talk. This class inherits from Action and all the methods from Action are reused. The constructor consists of a parameter “subject” that represent Q. The purpose of this class is to allow communication between player and character Q, by letting player talks to Q.

Method execute has a loop that checks the inventory of player for Rocket plans using getClass() and toString() method of comparison. This double verification ensure valid Rocket Plans item is return. When the method execute is executed while the player is not holding any rocket plans, this method will return a string “I can give you something that will help, but I’m going to need the plans”, but if the player does hold the rocket plans, this method will return a string of “Hand them over, I don’t have all day!”. String returned indicates the replies from Q, and will be printed on the UI.

The method menuDescription will print a string “Player talks to Q” on the menu.

The method hotKey will print a string “9” as the hotkey to represent Talk action.

GivePlans Class

This class represent a new action called GivePlans. This class inherits from Action and all the methods from Action are reused. The constructor consists of a parameter “subject” that represent Q, “item1” represent rocket plans and “item2” represent a rocket body. The purpose of this class is to let player exchange an item with Character Q. When the method execute is executed while the player is holding rocket plans, this method will caused the rocket plans to be removed from player and replaced by a rocket body. After the exchange, Q will then be removed from the game, and return a String “Q disappears with a

cheery wave, and player obtained Rocket Body.”

The method menuDescription will print a string “Player give Rocket Plans to Q” on the menu.

The method hotKey will print a string “10” as the hotkey to represent GivePlans action.

WanderAround Class

This class is a new behaviour that allow an actor to wander around the map for the entire game, unless overridden by another action. This class implements ActionFactory. We override the getAction method in ActionFactory and move the actor in the random valid direction by returning a new instantiated MoveActorAction with the direction.

The idea of algorithm to give the actor a random direction to move towards is taken from the method processActorTurn in World class. We decided to remodify this algorithm into a better one that fits the term “wander”, while having a full understand of the original one.

Q Class

This class represent one of the NPC in this class called Q. This class inherits Actor class.

The method and constructor of Actor will be reused to set the name, symbol displayed on the map, priority, hit point and maximum hit point of Q.

We added a WanderAround behaviour to Q because it moves randomly on the map.

The method playTurn is the same as the one given in Grunt originally because we intended Q to WanderAround throughout the entire game. Thus overriding actions such as SkipTurnAction, DropItemAction and AttackAction.

The method checkItemList checks if the item list of player contains rocket plans and return a Boolean value.

A method getAllowableAction that overrides the method in Actor Class is added. A check loop is added to check the inventory of player for Rocket plans using getClass() and toString() method of comparison. This double verification ensure valid Rocket Plans item is return. The purpose of this method is to add two new actions to player, which is Talk and Give plans. This method only add the Talk action if player does not possess any rocket plans, but if player does possess rocket plans, both action will be added to the allowable action of player.

We hardcoded rocket plans to be in a locked room so that player will need a key to obtain it.

Ninja Class

This class inherits Actor class instead of inheriting any of the other enemy classes because it has a unique behaviour.

The method of Actor will be reused to set the name, symbol displayed on the map, priority, hit point and maximum hit point of Ninja.

This class contains setOriLocation method to save the player starting position to allow tracking of player movement of the entire game.

This class overrides the Actor class playTurn method to allow Ninja to determine whether the player is within 5 squares from them. If Ninja detected the player is within the range, it will perform move a step away action and have a probability of 0.5 to stun the player for 2 rounds. This overridden method will also keep track of player’s previous and current position to allow the stun action to be performed.

Another method of locationCoordinate is included to return the actor location in the list format of [x-Coordinate,y-Coordinate].

Stun method is added to allow the stun action to be performed which remove the player from the current position and add the player back to the previous position where the player was previously stunned.

RocketPad Class

This class inherits Ground class to allow it to reuse the superclass’s constructor to specify the display character.

The method of canActorEnter is overridden to prevent player from entering the RocketPad’s location.

Another method named allowableActions is overridden as well to check whether the player’s item inventory contains Rocket Engine and Rocket Body. If both parts existed, a new action of build Rocket is added to the player’s allowable actions. This action will invoke the BuildRocket Class which is responsible for performing building rocket action.

BuildRocket Class

This class inherits Action class to allow it to override the execute method of it’s superclass. The execute method is overridden to remove Rocket Body and Rocket Engine from the player’s item inventory and add the rocket object to the player’s item inventory if the player has chosen the action of building rocket which directly invoked the execute method of this class.

The menuDescription and hotkey methods are overridden as well to generate unique string and hotkey to be shown on the console for player to select the build rocket action.