Design Rationale : FIT 2099

**Door Class:**

Class Responsibility:

This class holds the ability to allow a player to pass through it only if the player has a key(an instance of the Key class) in their inventory. We decided to have unique keys for each door, therefore to enter a particular room we need its equivalent key(1 Door has 1 Key relationship)

Relationships With  Other Classes:

We observed that the Door class would require attributes and methods from the Ground class(which is a generalized class of terrain types- eg: Floor class & Wall class) as it has all of the methods(Need to Override) we require for our Door class. This way, we follow the DRY principle and eliminate the need to have the same code.

The door class will therefore extend(inherit methods and attributes) Ground class present in the game engine.

Methods/Attributes:

- Inherited attributes from Ground Class

+ Inherited Methods from Ground Class

Overridden Methods:

+canActorEnter(Actor actor) : Boolean

         If key is present in inventory, and the keyID of key in  inventory matches the doorID:

                   Returns True

         Else:

                   Return False

+blocksThrownObjects() : Boolean (Made True)

Addon Methods/Attributes:

-doorID: String

+isDoorOpen(): bool

If the door has been opened by the player before,  Return True. So it doesn’t need to be opened again

Else: Return False (Indicating the Door has **not** been unlocked yet)

+keyMatches(): bool

If the keyID attribute of the key in the player’s inventory matches the doorID: True

Else: Player is not allowed to pass through the door (“Seems like you have obtained the wrong key, come back when you have the right one”),  Return False.

**Key Class:**

Class Responsibility:

In order to obtain a key, the player must defeat an enemy. After the enemy has been knocked out, the enemy will drop a key. This key is used to open a door. Each Key, which is an object of the Key class, has a attribute called the keyID which is used to uniquely identify each key. The keyID matches with a specific doorId so that, that key can be used only to open that particular Door. Once the key is used, it should be removed from the players inventory.

Relationships With  Other Classes:

The key class inherits from the item class because when we thought about the implementation we realised that we needed all the attributes and methods used in item class, mainly, the displayChar attribute, the allowable actions attribute, the getAllowableActions method and the newInventoryItem method. This is because when we went through the code given for the Item class, the getAllowableActions and the newInventoryItem  method already implements a way for the player or enemy to pick up an item and drop an item . This will be useful to us when player defeats an enemy (Grunt,Goon,Ninja) and drops the key into the location where the enemy was. By inheriting methods and attributes from Item class, we are yet again avoiding duplicated code(DRY principle)

The Key class has a one to one relationship with the door class and it is a dependency relationship because the door (a locked one) cannot exist without a key according to our understanding and our plans of implementation of the game.

Methods/Attributes:

- Inherited Attributes from the Item class.

+ Inherited Methods from the Item class.

Addon Methods/Attributes:

+ keyID: String

**Goon Class:**

Class Responsibility:

The Goon Class is used to implement an enemy character. The Goons can either attack the player with twice as much damage as Grunt or the Goons have a 10% chance of shouting insults at the player. The Goons also follow the player like Grunt.

Relationships With  Other Classes:

The Goon class will extend(inherit methods and attributes) Actor as it is an enemy with all required attributes of the Actor class. The Goon class also requires all the methods (like isConscious, hurt, getIntrinsicWeapon) in Actor class to be implemented properly in the game.

The Goon has two behaviors insultBehaviour and FollowBehaviour. The way FollowBehaviour is implemented for Goon is the same way that it is implemented for Grunt so it is not shown in our class diagram. However, the insultBehaviour is a new Behaviour so we showed it in our class diagram. The Goon class uses ActionFactory interface to generate actions to perform, we show this using the simple association relationship. The Goon Class also can hold a key which is shown by a simple association relationship. The Goon Class is also dependant on Action class because the Goon Class cannot exist without an Action to perform.

Methods/Attributes:

-Inherited attributes from the Actor Class

+ Inherited methods from the Actor Class

Overridden Methods:

+playTurn() - currently this method randomly chooses an Action to be performed by the user. This method should be overridden in a way that an appropriate action is performed at an appropriate time. Eg. Either follow the player,attack the player or insult the player depending on the players location.

+getIntrinsicWeapon() -  this method should be overridden so that the Intrinsic Weapon’s damage points and the name of it can be changed so that the damage of the Goon is twice of Grunt.

**InsultBehaviour Class (Implements ActionFactory interface):**

Class Responsibility:

The InsultBehaviour class is used to print an insult only with a 10% chance at each turn. The way we decided to implement InsultBehaviour is that it has an attribute called insults which is an arraylist of strings with different insults. Every time the InsultBehaviour class is called, we will obtain a random insult by getting a random number between 0 to len(arraylist) and using it as index to get insult  through the arraylist of insults and print it.

Relationships With  Other Classes:

The insultBehaviour class inherits from Action class as it is an Action that can be performed by the Player and it has all the required methods for us to implement the class. The class InsultBehaviour, implements ActionFactory so that it can use the getAction() method.The getAction() method is overridden in order to return steps to achieve our behaviour., in this case throwing insults at the player. We show that InsultBehaviour implements ActionFactory by using a interface realisation arrow.

Methods/Attributes:

- Inherited Attributes from the Action Class.

+ Inherited Methods from the Action Class.

Overridden Methods:

+ execute() - override this method so that at each call to InsultBehaviour a random insult from the arraylist of insults is printed on the console.

+getAction() - override the method so that when the player is within the range(will be decided later) of the goon then the goon calls the execute() method which prints an insult on the console.

**Ninja Class:**

Class Responsibility:

The Ninjas have the ability to stun the player but there is only a 50% chance of that happening. When the player is stunned he cannot perform any actions for two turns. After throwing stun powder at the player, the ninja moves back one step.

Relationships With  Other Classes:

The relationships of Ninja Class are the same relationships of the Goon class as they are both enemies the only difference is that Ninjas don’t have FollowBehaviour so we omit that relationship for Ninja. We plan on overriding the method of the Actor class for Ninja so that it is able to spot a player if player is within 5 squares of Ninja and Stun the player.

Methods/Attributes:

- Inherited Attributes from the Actor Class.

+ Inherited Methods from the Actor Class.

Overridden Methods:

+playTurn() - currently the playTurn method chooses a action randomly from a list of possible actions for the actor. We need to override it for Ninja in a way that there is only one action which is only performed if the player is within 5 squares of Ninja. There are two actions, throw stun powder and move one step back but it should be implemented as one for the sake of simplicity.

**StunBehavior Class: (Implements ActionFactory interface)**

Class Responsibility:

The stunBehaviour class is used to stun player with only 50% chance of stunning the player. The way we decided to implement stunBehaviour is, there is another class called stunPowder which is a weaponItem object which is present in the inventory of Ninja. Each time the stunBehaviour class is called it checks the Ninjas inventory for stunPowder and stuns the player for two turns but the probability of that happening is only 50%

Relationships With  Other Classes:

The stunBehaviour class has the same relationships with the classes from the engine as the insultBehaviour class so we choose not to waste time explaining about it. However, we are explaining the relationship of the stunBehaviour class with non-engine classes below.

The stunBehaviour class is dependant on the stunPowder class because if Ninja doesn't have stunPowder he cannot perform stunBehaviour. This is shown in the diagram as a dependency relationship with a normal open arrow with dotted lines. Ninja has stunBehaviour which is shown as a simple association.

Methods/Attributes:

- Inherited Attributes from the Action Class.

+ Inherited Methods from the Action Class.

Overridden Methods:

+execute() - this is overridden in a way to show a description that the Ninja stuns the player.

+getAction() - this is overridden in a way that the methods gets the allowable actions of the player and sets it to null for two turns.

**Stun Powder Class**

Class Responsibility:

The objects of the Stun Powder class are held in the the Ninjas inventory. The Stun Powder has no damage but it Stuns player for two turns.

Relationships With  Other Classes:

The Stun Behaviour Class is dependant on the Stun Powder class because the Ninja cannot perform Stun Behaviour without the Stun Powder. The Ninja holds instances of the Stun Powder class in its inventory which is shown by an simple association relationship.

Methods/Attributes:

- Inherited Attributes from the Action Class.

+ Inherited Methods from the Action Class.

Overridden Methods:

+execute() - this is overridden in a way to show a description that the Ninja stuns the player.

+getAction() - this is overridden in a way that the methods gets the allowable actions of the player and sets it to null for two turns.

**Q(NPC) Class:**

Class Responsibility:

The Q actor is a NPC (Non-Player Character). The Q class is an important class as it communicates with player(Talk Behaviour) and it gives the Rocket Body in turn for the Rocket Plans.

Relationships With  Other Classes:

The Q class inherits from Actor class because it is also a character with hitpoints, a display character and all the other attributes any other actor has. It also uses the methods of the actor class therefore it inherits from Actor class. The givePlansAction class is dependent on the Q class because the givePlansAction cannot take place unless the Q is there to receive the plans.

Methods/Attributes:

- Inherited Attributes from the Action Class.

+ Inherited Methods from the Action Class.

Overridden Methods:

+execute() - this is overridden in a way to show a description that the Ninja stuns the player.

Note:

Q inherits from actor class and the playturn method should be overridden so that the actions list for Q is a list of MoveActorActions chosen randomly. And the talk action and give plans action should be in the actions list accessed by the getAllowableActions  method which should be overridden as well.

When ever position of player changes we need to get allowable actions of player from depending on the surroundings. When actor moves to location of item the actor picks it up. When an actor is defeated the actor drops the item.

Description

Inherited Methods/Attributes:

+

Overridden Methods:

+

Addon Methods/Attributes:

+

**Doctor(Doctor Maybe) Class:**

Description

Inherited Methods/Attributes:

+

Overridden Methods:

+

Addon Methods/Attributes:

+

**RocketPlans Class:**

Class Responsibility:

The RocketPlans is an object of RocketPlans class, which needs to be given to Q(the Non-Player Character) in order to obtain the “Rocket Body”.

Relationships With  Other Classes:

The RocketPlans class inherits from the item class because when we thought about the implementation we realised that we needed all the attributes and methods used in item class. This makes it very similar to the key class(with regards to methods such as getAllowableActions, newInventoryItem method etc.) except for the fact that the plans will be held inside of a room. This will be manually positioned in the application class of the game which is used to initialize the characters/terrains of the game. We decided to do it this way because we can make changes to any code in the Game Package and cannot make changes to the Engine Package.

The reason we decided to inherit from the item class is the same reason we decided to inherit the item class for key class. There are attributes and methods present which already implement a way in which player can pickup or drop an item.

Methods/Attributes:

- Inherited Attributes from the Item Class.

+ Inherited Methods from the Item Class.

**RocketEngine Class:**

Class Responsibility:

The RocketEngine which is an object of the RocketEngine class is initially given to the doctor(stored in the Doctors Inventory) at the start of the game.

Relationships With  Other Classes:

The RocketEngine class inherits from the item class because when we thought about the implementation we realised that we needed all the attributes and methods used in item class. This makes it very similar to the RocketPlans class(with regards to methods such as getAllowableActions, newInventoryItem method etc.). Doctor Maybe must hold the RocketEngine and it is shown by a simple association relationship.

Methods/Attributes:

- Inherited Attributes from the Item Class.

+ Inherited Methods from the Item Class.

**RocketBody:**

Class Responsibility:

The RocketBody which is an object of the RocketBody class is initially given to Q(stored in Q’s Inventory) at the start of the game.

Relationships With  Other Classes:

PLEASE DO THIS

Methods/Attributes:

- Inherited Attributes from the Item Class.

+ Inherited Methods from the Item Class.

**RocketPad Class:**

Class Responsibility:

This class is used to build the rocket! This is done when the player has both the Rocket Engine and the Rocket Body in his inventory upon entering the RocketPad which is a location in the map. The RocketPad is initialized at the start of the game by giving it a location.

Relationships With  Other Classes:

Do this

Methods/Attributes:

- Inherited Attributes from the Location Class.

+ Inherited Methods from the Location Class.

**Rocket Class:**

This class