Design Rationale

Design Choice

The project would be divided into 4 main parts: Item, Ground, Actions and Actor. The newly added classes will inherit from those 4 classes above. The project was designed in this way because polymorphism can make the development and maintenance of this project easier since many of the new classes that are of different types are in the same catalogue. That means they can be managed by the same parent class. So the class "Q", "Goons", "Ninjas" and "DoctorMaybe" would inherit from the class Actor. This is because they are all NPCs that were created by the system and cannot be manipulated by the user. The class "KeysItem", "RocketItem", "RocketPlanItem", "RocketBodyItem" and "RocketEngineItem" would inherit from the class "Item" because they are the objects that can be obtained by the actors and cannot move on their own. Inheriting from "Item" also it possible for the actors to put these items into their inventory. Class "SpeakAction", "GivePlanAction", "buildRocketAction" and "TalkAction" would inherit from the class "Action". Those classes are actions and behaviors that can be performed by the actors. So we can use "ActionFactory" to generate them and decide which of them should be performed by the actor in this turn later on. An interface "ActionFactory" would be called later on to generate a list of actions, so the "InsultBehaviour" and "MoveAwayBehaviour" will implement to generate actions that an actor should perform. The classes "Doors" and "RocketPad" would inherit from the class "Ground". This is because "Doors" and "RocketPad" share the same property with the parent class "Ground" and added a few more methods to make them identical. Being a child class of "Ground" can also let them to show properly on the map. A class that inherit from class "Player" would also be created so we can add new methods and attributes inside to make it perform correctly when the player is stunned.

In the assignment3, some new classes were added to implement the required functionality. Class SpaceSuit, OxygenTankItem, WaterItem, WaterPistolItem and ExoskeletonItem were inherited from the class Item so they can be added into actors' inventory and can be carried by the actors. Class OxygenDispenser, LunarSoil and Pool were inherited from the class Ground so they can be read into the map and use all the functionality from the class Ground. Since they don't need to be modified or moved by the actors, so they don't need to be subclasses of class Item. Yugo Maxx will be a subclass of the class Actor so he can carry items and perform actions. His initial way to attack and how he could get hurt can also be implemented by overriding the functions from the class Actor. Several classes that inherit from the class Action are also added to the game including WinAction, LoseAction, QuitGameAction, SquirtingWaterAction, AddWaterAction and makeOxygenAction. These classes were made into subclasses of the class Action so they can be added into the action list and performed by the actors. Also, this make display easier as the result of the execution can just be returned and the world will handle the display part. This is also convenient for adding some of them into the places so the actors can only perform that at some specific locations. A new class called NewWorld that inherits the class World was also added to this game so the world can be modified according to the needs. This class was added because the current world cannot record the result of execution so it's hard to determine when the program should end. Inheriting from the class World makes it convenient to reuse the functionality in World while new functionality can be added to satisfy the requirements.

Class roles

This section is to explain the functionality of each class and the purpose of creating each class.

Goons:

The class Goons would have 3 main behaviors: Attack, Insult and Follow. It would use "ActionFactory" to generate an action list at the beginning and perform these actions accordingly. This class would inherit from the class Actor.

Ninjas:

The class Ninjas would have 3 main behaviors: Attack, StunBehaviour and MoveAwayBehaviour. It would also use an "ActionFactory" to generate an action list. It would check the condition first before

performing the actions. This class would also inherit from the class Actor. This was made into a class since we are going to have several Ninjas and making a class would make the creation easier.

DoctorMaybe:

The class DoctorMaybe would have 2 main behaviors: Attack and DropItem. Before performing any actions, there would be a condition to check whether the door is open or not. This was made into a class so the actions can be added easily and this object can be modified easily.

Q:

The class Q would have 2 main actions: TalkAction and GivePlanAction. The GivePlanAction action will replace the plan in the user's inventory with the item RocketBody. This was made into a class so it can have the same format with other NPCs and is easy to identify.

KeysItem:

The class Keys will inherit from the class Item. So the players can put these keys into their inventory because of polymorphism. The keys will be initially on the NPCs, and NPCs would drop them after their hitpoint goes to 0. This was made into a class so the user can have multiple items in their inventory that are of different types.

RocketPlanItem

This class will also inherit from the class Item. The plan would be in a locked door so the doors objects would have a check method to see if the plan is inside or not. This is made into a class because RocketPlan can inherit from class Item so the player can put it inside the inventory.

RocketBodyItem:

This class will be a subclass of class Item so the program can replace the RocketPlan with RocketBody easily. If the player has the RocketPlan and talked to Q, RocketPlan would be removed from the inventory and add RocketBody instead.

RocketEngineItem:

This class will also inherit from the class Item. So it is easy to put it into the user's inventory and check if RocketEngine and RocketBody are both in the player's inventory or not. I.e. If the player has won this game or not.

InsultBehaviour, StunBehaviour,MoveAwayBehaviour:

These classes will implement the interface ActionFactory so these behaviors can be used to generate actions and decided which one to perform according to the conditions. It is also beneficial for the actors as they can add them easily into an arrayList and decide which one to perform in the playTurn method.

Doors:

This class will inherit from the class Ground since it shares some same properties with class Floor and added some methods to make sure its functionality. After the Doors were opened by the user, it would change it's property so it acts the same as the floor.

RocketPad:

This class is a child class of the class Ground since it needs to show on the map and decide whether an actor can enter or not. It will have an action that justify whether the actor can build a rocket or not. If the actor can build a rocket, it will replace the rocket body and rocket engine in the actor's inventory with the rocket itself.

TalkAction,GivePlanAction,BuildRoctetAction,SpeakAction:

These classes are the action classes. These were made into a class so when creating the NPCs, they can be added into the ActionFactory so the NPCs can perform these actions using the ActionFactory. They all inherited from the class Action so they are easy to manage and show on the menu.

StunnedPlayer:

This is the class that inherit from the class Player. It was made into a class like this because it needs to perform all the behaviors a player can perform if the player was not stunned. If the player is indeed stunned, this class will only returns a SkipTurnAction so the player doesn't need to do anything. Making it as a child class of the Player class let the performing of the player's normal actions possible while it would do nothing while stunned.

How System Work and Why Chose to do this way?

Firstly, the game system would run a constructor in the Doors class to set the door's character so that it would appear in the user interface. Secondly, the game system uses a canActorEnter method to return a Boolean value to judge whether the door would be opened or not. In the canActorEnter method's if condition, game system uses a checkKey method which in the Keys class to return a Boolean value that checks whether a player has the correct key for the door or not. If the player has the correct key, this method will return a true. Then the canActorEnter method's if the condition will execute and return true to open the door for the player. Since one key corresponds to only one door, the game system needs to use the Keys class to check whether the key is the right one, and then use the return in the checkKey method of the Doors class to decide the status of the door. Furthermore, the game system uses a blocksThrownObjects method to stop anything go through the door when it is closed. In the Goons class, game system creates a constructor to set the actor goon's name, display character, priority and its hit points. Since game system need to create a new object in the Application class and initialize the attributes for game using. Inside this class, addBehaviours and playTurn methods perform the actions of the goon, such as insult and attack the players. Moreover, goon need to check whether the player has been stunned or not since goon cannot do anything before the player wakes up.

The InsultBehaviours will be added to the goon's array list using the actionFactories and executed the methods. In the Ninjas class, a Ninjas' object will be set in the Application class by using the constructor of Ninjas. Firstly, the distance between Ninjas and the player should be accessible because Ninjas can only see the player when the player is within 5 squares of them. Only when the Ninjas' can see the players, getAction and playTurn would be called. So ninjas would throw the stun powder towards the player and move one space away from the player. Also, the StunPowder

need to be added into Ninjas' action factories. In the Q class, game system uses a constructor to show the character of Q in the map. Then, the checkPlan method would be called to check whether the player has a rocket plan or not. If the player has the plan, it will call getRocketBody method from GivePlanBehaviours class. The player needs to use the rocket plan to exchange the rocket body. TalkwithQ be called anyway. There is a check condition inside it so if the player doesn't have the rocket plan, it would return false and would do nothing to the user's inventory. Else it should replace the rocket plan.

Assignment 2 fixed part:

Firstly, system will run a new class BulidRocket for RocketPad class. The BuildRocket behaviour called by RocketPad class's allowableAction method. Because player need to put them rocket body and rocket engine on the rocket pad. The BuildRocket class will remove player's rocket body and rocket engine item from player's inventory. Moreover, if player inventory has both item, the execute method will return a string that shows build successful. If player only has one of those two item, execute method will return a string that shows player do not have enough part.

Secondly, system run new class MoveAwayBehaviour, It made for ninjas, because ninjas will move one step away from player when finished attack. In the MoveAwayBehaviour, getAction method will get location of player and ninjas and calculate the distance between ninjas and player. In the end of this method, system will choose a farthest distance to move ninjas. This will cause ninjas move away every turn when player far from ninjas 5 steps. Fuethemore, there is a distance method for calculate the distance between player and ninjas.

Thirdly, when executing the StunBehaviour, it would check if the actor can see the target or not. If the target cannot be seen, i.e. the target is 5 squares away, the actor would do nothing. If the target can be seen, the method would get the status of the target and see if the target has already been stunned. If the target is not stunned or the stunned turns has become 2 turns, the method would generate a random number to see if the target would be stunned in this turn. Then the behaviour would return the method from MoveAwayBehaviour to move the actor away from the target.

Finally, System also run a new class that called SpeakAction. It used for goons talking. Because goons need to insult player. SpeakAction called by InsultBehaviour. It use SpeakAction print insult sting on the user interface.

Assignment 3:

WaterPistolItem: Player needs a water pistol item to destroy the yugomaxx's exoskeleton, but the water pistol needs to be filled of water. The water pistol should be put 5 squares away from rocket and the player needs to pick it up and put it into inventory. Player also needs to walk to the pool to fill water into the water pistol. Every time player attacks yugomaxx, the water pistol should be filled of water again.

Pool: The pool exists in the earth map to provide water to be filled in the water pistol. The player cannot stand at the location of the pool because the canActorEnter method would return a false. This was designed into a ground so it can have actions that can only be performed at this location and actors cannot enter it.

WaterItem: Player could get the water item from the pool. But player can only carry one unit of water item in inventory. Once the player runs out of water, the player should go back to the pool and get water again.

SquirtingWaterAction: This action is used to attack the yugomaxx with water pistol. The possibility of the yugomaxx's exoskeleton being destroyed is 70%. This action will only generate damage to the exoskeleton, therefore, player should check whether the yugomaxx has exoskeleton before attacking with the water pistol.

QuitGameAction: This action was added inside the stunnedplayer class's action. The player can decide to quit the game or not once a new round starts. If player choose to quit the game, it will return a statement to notify the world. After the world detected the user wants to quit, it will

break the loop and the program would stop.

WinAction/LoseAction: These 2 actions will be called if the player has won the game or lost the game. They both inherited from the class action so they can be managed easily and the user can perform it in the function playTurn. Once the wining or losing condition has been met, they will return messages and break the loop in the world so the program would end immediately without processing the next actor.

NewWorld: This class was inherited from the class World so we can customize the world to perform the needed functionality. It has an additional attribute to record the result after each actor played their turns. If the result means the player wants to quit, they lost or they won, the loop inside its function run() would be broken and the game would end immediately. It was designed to inherit the class World so all the functions in the world can be reused and modified to fit the requirements.

StunnedPlayer: The game system needs to determines whether the player is win or lose.
The playTurn method in StunnedPlayer class contains the condition of the player existing in the earth map and getting unconscious yugomaxx body in inventory. If the player satisfies both conditions, the system shows “Player win” and be quit. Oppositely, if the player is beaten by others, it will shows “player losses” on screen.

OxygenTankItem, ExoskeletonItem, SpaceSuit: These classes were made into subclasses of class Item so they can be carried and modified by actors. Player can pick up oxygen tank and space suit to so that they can have then required equipment to go to moon. The exoskeleton would be added to YugoMaxx’s inventory so the player have to destroy it first before they can actually hurt him.

OxygenDispenser: Since the player need a place to make oxygen tanks, so this class was made as a subclass of class Ground. Then it can have action list so the player can only make oxygen at its location. It can also be added into the map since it’s a subclass of the class Ground.

YugoMaxx: This class is a subclass of the class actor so Yugo Maxx can have the exoskeleton in his inventory. And it is also convenient to override the functions in its parent class so he can have his own ways to attack and he cannot be hurt if he has the exoskeleton. His pick up item action and drop item action were removed since he shouldn’t pick up or drop anything during the process.

AddOxygenAction: This action was added into the class OxygenDispenser so actors can only make oxygen if they are at that location. This action would have a condition to check whether this place has already has an oxygen Tank or not. If there is an oxygen tank, it would not produce a new one until the old one was removed by the player. This was made into a subclass of the class Action so it can be easily added into the action list.