Design Principles used in Design Rationale:

- Don't Repeat Yourself [DRY]
- Avoid Excessive Use of Literals [AUL]
- Declare things in the tighest possible scope [TPS]
- Classes should be responsible for their own properties [ROP]

Design Rationale:

Enemy

- Extends from Actor class, so will inherit methods from Actor
- With additional addBehaviour() method to add a behaviour into Enemy's actionFactories
- addBehaviour is protected so that every subclass of Enemy can use the method without repeating the same code [DRY]
- Override playTurn method so that if subclass does not override this method, it will try to execute behaviours in actionFactories first, if the Action in actionFactories are not executable (return null), it will return a random Action from actions list.

Grunt

- Extends from Enemy class, so it inherits methods from Enemy (as well as Actor class) if the methods are not override [DRY]
- Grunt will have 50 hitpoints, represented by 'g' and has priority of 3, so it is the 3rd Actor who performs action in each turn of the game
- Using protected addBehaviour() method from Enemy class, we can add FollowBehaviour to Grunt。 FollowBehaviour is constructed when it is added [TPS]
- Create a key Item when Grunt is cronstructed and store the key Item in both Grunt's inventory and Door object. This follows design principle declare things in the tightest possible scope to reduce dependencies since we only construct a key Item when it is needed in constructor of Grunt
    - So that Grunt will have key and will drop the key through DropItemAction when they are defeated
    - Door can check if Actor has these key Item in order to perform OpenDoorAction (refer to OpenDoorAction)
- Overriding playTurn method (polymorphism) so that during Grunt's playTurn:
    - DropItemAction is removed from Grunt's actions to prevent Grunt from dropping key at random
    - Grunt will prioritise behaviours in actionFactories, where it will try to performs its behaviour in actionFactories if any of them is performable, else they will randomly choose an action from actions
    - Grunt will have IntrinsicWeapon which allows it to slap other Actor with 10 damage

Goon

- Almost the same as Grunt except that Goon's damage is double of Grunt's - reimplement getIntrinsicWeapon and change its value
- Goon has one additional behaviour than Grunt: InsultBehaviour

InsultBehaviour

- Extends Action class and implements ActionFactories [ROP] - ensure all necessary methods from Action class and ActionFactory are implemented
- Override getAction method to get 10% probability of insulting Actor
    - Generate a random number within 10 as local variable [TPS]
    - If the number is 1 (10%) then return this action so it will go to execute() method and perform this action, else return null
- Override execute method:
    - A local variable to store a random number [TPS]
    - Use the random number to choose which insult words to be thrown towards Actor

OpenDoorAction

- Extends from Action class, override abstract class methods to ensure these necessary methods are implemented
- Check if Actor has key Item in their inventory, if the Actor has key Item then remove the key Item from Actor's inventory and replace the Door with passable Floor

Door

- Extends Ground class so that we don't need to implement same methods again [DRY]
- Has key Item added into a list so that Door can check if the Actor has the key Item needed to open the Door.
- Override getAllowableAction so that if the Actor has key Item then the Actor will have OpenDoorAction in their menu/ actions

Q

- Extend from Enemy class, so it inherits methods from Enemy(as well as Actor class) if the methods are not override.
- Q will have 100 hitpoints, represented by "Q" and has priority of 6, so it is the last actor who performs action in each turn of the game.
- when Q is created, that must be a rocketPlan that corresponding to the rocketBody passed in via the constructor.
- Q will create a rocketBody and store the rocketBody item in both Q's inventory and RocketPad object
- Using protected addBehaviour() method from Enemy class, we can add StandStillBehaviour to Q
- Using the concept of polymorphism, Q class override the getAllowableActions from Actor class.
    - otherActorActions, TalkAction and GivePlansAction object is created and assigned inside the class as we want to declare things in the tightest possible scope.
    - Q will check if the otherActor has rocketPlan in their inventory and add GivePlansAction to the otherActor Actions if it does.

GivePlanAction

- Extend from Action class, so it inherits methods from Action abstract class hence all the method exist in the Action abstract class need to override in GivePlanAction class [ROP]
- The constructor of this class including the actor and the subject as well as rocketPlan
- Execute class
    - The method will first check if the actor's inventory item has the same item with the rocketPlan [AUL], If yes the actor will execute removerItemFromInventory (which indicates RocketPlan has passed to subject for exchange).
    - Then subject will excute the DropItemAction and drop all the InventoryItem(including rocketBody)
    - The map will then remove the subject from the map.

TalkAction

- Extend from Action class, so it inherits methods from Action abstract class hence all the method exist in the Action abstract class need to override in GivePlanAction class [ROP]
- The constructor of this class including the actor and the subject as well as rocketPlan
- Execute class
    - The method will first check if the actor's inventory item has the same item with the rocketPlan [AUL]. If yes, the flag which is declare as false will set to true.The result will be set based on the result of the flag.
    - To move the subject to a random place, we create a random number and map the x and y to the map by using map.at(x,y), if the corresponding ground is canActorEnter, then map.move the subject to that location, else create a new random object to get new random x and y inside the method [TPS].
    - Then subject will excute the DropItemAction and drop all the InventoryItem(including rocketBody)
    - The map will then remove the subject from the map.

DoctorMaybe

- Extend from Enemy class, so it inherits methods from Enemy(as well as Actor class) if the methods are not override.
- DoctorMaybe will have 25 hitpoints, represented by "D" and has priority of 5, so it is the 5th who performs action in each turn of the game.
- DoctorMaybe will create a rocketEngine Item and store the rocketEngine item in both DoctorMaybe 's inventory and RocketPad object
- Using protected addBehaviour() method from Enemy class, we can add StandStillBehaviour to DoctorMaybe
- By using the concept of polymophism, we override the playTurn method from the Actor class.
    - AtttackAction is the only actions that can be done by DoctorMaybe
    - Then for the actionFactories, DoctorMaybe will perform the StandStill behaviour as that is the only action that add into the actionFactories List.
- DoctorMaybe will have the InterisicWeapon with damage of 5 hitpoints to other actor.

Ninja

- Extend from Enemy class, so it inherits method from Enemy(as well as Actor class) if the methods are not override.
- DoctorMaybe will have 50 hitpoints, represented by ‘n’ and has priority of 4, so it is the 4th who performs action in each turn of the game.
- Using protected addBehaviour() method from Enemy class, we can add ThrowStunPowderBehaviour and StandStillBehaviour to Ninja.
- By using the concept of polymophism, we override the playTurn method from the Actor class.
    - By looping the actionFactories, DoctorMaybe will perform the ThrowStunPowderBehaviour or StandStillBehaviour. However, ThrowStunPowderBehaviour is being prioritised as it is first added to the actioFactories.

RocketPad

- Extend from Ground class, so it inherits methods from Ground class if the methods are not override.
- A static Item ArrrayList (rocket) is created, as every instance of any class will use the same ArrayList.
- AddRocket Parts
    - The rocketBody(parameter) which is instance of Item will be added to the rocket ArrrayList when the rocket parts created.
- hasCompleteRocket
    - It will check if the Item in actor’s inventory list contains the Item same of the rocket ArrayList
    - If the actor’s inventory list contains the two same Item with the rocket ArrayList, return true otherwise false.
- allowableActions
  This method is override from the Ground class
    - It will first check if the actor have both rocket body and rocket engine by calling (hasCompleteRocket). If return is yes, new FlyToMars action is created and return to actor. Else, return the allowableActions from superclass.

FlyToMars

- Extend from Action class, so it inherits methods from Action abstract class hence all the method exist in the Action abstract class need to override in FlyToMars class [ROP]
- The map will remove the actor from the map and return the prompt (actor ”flew to Mars happily ! Bye Bye")

ThrowStunPowderBehaviour

- Extends Action and implements ActionFactory [ROP] - ensure necessary methods are overriden
- Override getAction() from ActionFactory to get Action to be performed
    - If there is a Ground object that blocks performing Actor from seeing target Actor, it will return null (not performing this Action

    - If the distance between performing Actor and target Actor is within five squares, which is the x-coordinate and y-coordinate of performing Actor and target Actor are both less than 6, then performing Actor can perform this Action so it will execute this Action through execute() method
        - If target Actor could not be detected, it will do nothing
    - execute() method is overriden:
        - Local variable to store probability of stun powder hit on target Actor [TPS]
        - If probability is true (50% chance), stun powder will miss target player
        - If probability is false, check if target Actor is stunned,
            - If it is stunned, return a message to tell user that stun powder has no effecr
            - Else target Actor will be stunned for 2 turns (refer to NewPlayer)

StandStillBehaviour

- Extends SkipturnAction and implements ActionFactory [ROP] - ensure necessary methods are overriden
- The menuDescription is override to return empty string so the skipTurnAction prompt will not be printed out.

NewPlayer

- NewPlayer extends Player class, so inherits public and protected methods from Player hence duplicated code can be avoided [DRY]
- Private fields isStunned and counter is only visible within the class, hence we have getter and setter to retrieve and set the values
- During a NewPlayer's playTurn:
    - Check if the NewPlayer is stunned
    - If it is stunned, update the counter to know how many turns has the player been stunned and return SkipTurnAction. SkipTurnAction is only constructed when it is needed [TPS]
    - If the NewPlayer is not stunned, return showMenu so that NewPlayer can choose Action to be performed from menu