Design rationale

1. How will the proposed system will work (class roles)
   - The roles and responsibilities of any new or significantly modified classes
     - Firstly, initialisation step:
       Door class inherits from Ground class that is represented on the GameMap as a special symbol (e.g. "+"). And Key class which inherits from Item that would be implemented with a use of HashMap that, each different key would map to a special type of room that only able to unlock that type of room. When the player is on the location of door (entering process), check if player have the type of they key matches to the location.

     - Secondly, the Enemy abstract class that inherited from Actor class would provide main skeleton attributes and methods for the other children classes (Ninja, Grunt, Goon and DoctorMaybe). In these children classes, FollowBehaviour class is changed to implement new functionality (Following player and insulting at the same time) for Goon class, and StunBehaviour is created for special functionality for Ninja to be able to stun player (which implements ActionFactory interface).
       Class Q inherits from Actor class as a special NPC that provides ExchangeBehaviour that allows player to obtain rocket body, where ExchangeBehaviour also implements ActionFatory interface that execute actions.

     - Then three main rocket item parts (RocketPlan, RocketBody, RocketEngine) classes are constructed by extend item class, RocketPlan would be placed at the locked location randomly that waited to be collected, RocketBody (which is an item of Q and in dependency relation with Q) is exchanged while player meeting Q with RocketPlan, and the last part RocketEngine (which is an item of DoctorMaybe and in dependency relation with Q) is dropped when player defeats DoctorMaybe.

     - Lastly, RocketPad class is created (which extends from Location class) as a final stage to check if player is winning or not.

   - How these classes relate to and interact with the existing system
     - Item like RocketPlan, RocketBody, RocketEngine that are inherited from Item class from existing system can use methods and functionalities such as DropItemAction, PickUpItemAction, so that the desired functionality can be fulfilled.
     - Actors like Q, Enemy and DoctorMaybe can execute actions from the existing system such as AttackAction method, so that the desired functionality can be fulfilled.
     - Location like RocketPad that are inherited from Location class from existing system can use methods and functionalities such as addItem methods and specify the location on the map so that the desired functionality can be fulfilled.

2. Why we implemented the new classes
    - New classes are created and implemented to fulfil the functionality the current existing systems cannot provide. As for the whole game, elements in the game are existed as same-types elements but executing (tend to execute) with different functionalities.
    - For example, for Q and Enemy, Q is not able to attack whereas Enemies can attack and follow player, but both of they were existed as Actor in the original game system, so that the functionalities of the specific character can not be displayed/executed. Inside Enemy abstract class, 4 different classes are created as each of them has different functionalities (Ninja has stunning skill where as the other 3 enemies do not, DoctorMaybe do not have FollowBehaviour whereas Grunt and Goon so that they are separated class)
    - Secondly, Rocket parts are divided into 3 classes (RocketPlan, RocketBody, RocketEngine) so that it is easier to distinguish which specific part of rocket does the character in the game processes (including player).
    - Finally, RocketPad are created, inherited and separated from other Locations is because of the fact that it is going to determine whether the player is qualified to win or not. (To check the items that the player processes)
    - Door class is inherited (from Ground) to distinguish the wall.
    - Key class is inherited from Item class is based on the idea that our team intend to implement HashMap to map different types to key to unlock specific types of doors)

Assignment 2 – updated Rationale

Changes:

1. Enemy class removed -- As we found that the enemy class does not need to add any methods and all the subclass should inherited Actor class

2. PlayerStatus class added – To inherit from Player that player actions can be modified and controlled (specifically useful for the effect of stun power – make player SkipTurnAction() for two rounds
3. Change RocketPad class: this class should inherit Ground class

4. Created openDoor class: this class should inherit Action class because this the action let actor to use the key and open the Door. When the door opened, it show remove the item Key and replace the new Door() char to new Floor().

5. Created DotocterBehavior class: this class should inherit ActionFactory and including the DoctorMaybe attack behavior and skipTurn behavior also when it is knocked out, should drop a RocketEngine.

6. Change Key class: it should independent to Ninja class, Grunt Class, Goon class because Key will be assigned when the actor was created. Also it will be added on GameMap class when the DropAction is processed.

7. Change RocketBoby class, RocketEngine class, RocketPlan class should independent to GameMap class. It will be added on GameMap class when the FropAction is processed.

8. Communication and Exchange classed are added – as they are different actions that Q would make use of, and these actions are not like FollowBehaviours as they have return type in string to allow direct user interaction and display. And better suits the functionality of Q