Recommendations for change to the game engine

- **Application, World, FancyGroundFactory, Ground**

First of all, the addition of the moon map requires us to add in actors on the moon map too. These actors are added into the same actorLocation hashmap data structure in world (or the extended world class) even though they are in different maps. It has led to displaying the actions taken by all actors in both maps. A user might face confusion as the map displayed is only the map the player is in.

Furthermore, the getConstructor() used in FancyGroundFactory doesn't accept parameters hence limiting the game to have the same symbol for each ground object of the same type. The public constructor required in its subclasses seems unnecessary without the ability to take in parameters to set different symbols.

> The untouchable design of the engine package where list of possible/allowed actions is first created in World and passed into the method playTurn() of each actor limits us from changing the list of actions passed to player using a class that may extend from others aside from Player. This may also be due to Player class being inside the engine package. Interception in between could not have been possible as calling playTurn() for the player would mean skipping other actors' turn and calling playTurn() for all actors would need the list of actors. This list of actors can only be created during initialisation in Application and may well be retrieve by a getter method in the Ninja class after performing a successful attack action but would not have the same reference to those added into the actorLocation hashmap data structure in world leading to errors. Additionally, creating a subclass of world would not work as we cannot set actor location twice or it would mean throwing away world completely as non of the methods could be used using super call without the reference of actorLocation.

- **World, Player**

The above comes from our design rationale under the section StunnedPlayer. This could be further elaborated as the World class in the engine package has now been modified. Those attributes which were private has been made protected which allows subclasses to access those attributes with a super call. However, this doesn't mean that we can abandon the previous implementation of the stun effect using the subclass.

Should the stun effect be implemented in this subclass of world, we would be overriding two important and backbone methods of world. It would again mean throwing away world completely but, this time it is due to using this new subclass

completely and using world add actors and map to the game only. Cases below are some examples:-

- The stun period would have to be calculated in run method and then makes a call to processActorTurn accordingly.
- Ninja would have to inform world to trigger stun.
- Since the quit option should appear in the menu in every turn, every turn would use the processActorTurn method in the subclass rather than the superclass to add it into the action list; throwing away the method in the superclass completely.
- The above point would make the first point invalid as it'll call processActorTurn *every turn* instead of accordingly.
- Then, the processActorTurn method would need a condition to check if the actor is a player and the stun effect is still going on to change the action(s) allowed.

In the end, the stun effect will still need to be implemented by extending Player and yet this is not what the assignment has in mind.

At the same time, this also leads to the problem where the code will get more and more complicated and no longer atomic. Take this assignment for example, the level of oxygen have to decrease for each level. The stunnedPlayer() class was already created to only makes sure that the player can do nothing when it is stunned. Since there is only one player on the map, the restrictions of not allowed to change the engine code leave me no choice but to decrease the oxygen level each round when the player is on the moon. The code will soon become unmanageable when in the future more items and feature is added and the player have to take care of everything in one single playTurn() method. My suggestion is, the engine code should introduce Mode() class. The actor can have many Mode(). In this case, the player can have stun mode and when it is on the moon, it will be on the spacesuit mode(means it is wearing spacesuit and the oxygen level decrease for each round). The playTurn() method get so messy as we are doing multiple task in one method due to the restrictions that the engine code is not allowed to change.

- **AttackAction**

The getWeapon() call in AttackAction prioritizes an actor's weapon as the medium of attack limiting the user's option to choose. Yet we might not want to waste the water pistol on the other enemies since it becomes empty regardless of hitting or missing. For this option to be added, unnecessary subclasses of AttackAction need to be created; one for weapon attack and another for intrinsic weapon attack for it to appear in the menu differently.

Plus, it always takes the first instance of a weapon which reduces player's damage to enemies should the weapon has no power (ammo). It again, reduces the player's option to choose between weapons in inventory. Extending the class just to allow this options seems to be throwing away the AttackAction class as it will need to override the whole execute method; the backbone of the class.

Since world loops through an actor's inventory to get allowable action(s) of an item, it could be utilise to gives the user an opportunity to choose between weapons, for eg: ChoosingWeaponAction. The next turn of attacking will then require usage of weapon of choice.

Also, other enemies would be allowed to attack each other even though they are on the same side since this action is added from the world class in the engine code. Difficulty of modification has already been mentioned above. Should we disabled that ability, it requires changes to all enemies which is not time-effective.

- **Item, DropItemAction, PickUpItemAction, Actor**

Next, the newInventoryItem method is supposed to help us create an item which allows DropItemAction only giving us the effect of it to be inside an actor's inventory. It's static nature prevents us from creating a new copy of the same item (good practice). But the method creates the item inside of it by accepting the necessary attributes as parameters which would restricts the item type to be only Item.

Thus, when creating a new inventory item of type Item and a subclass, we would need to go through the hassle of adding/removing actions using getAllowableActions for each item added to keep its instance nature. Having DropItemAction added into the list of actions allowed also causes non-player actors to drop mission items (key/rocket engine/rocket body) randomly as their actions are chosen at random which should be dropped only when they are knocked out.

Subsequently, PickUpItemAction existing in the list of actions allows non-player actors to pick up items. This reduces the chance of player winning the game as the difficulty is quite hard considering the consecutive stun effect after one period is over, empty water pistol after one shot and low cap of oxygen level. Items such as plans, weapons, sleeping actors, etc. shouldn't be allowed to be picked up by a non-player actor.

My recommendations is first of all, to avoid the non-player actor drop item randomly, the system shouldn't randomly pick one action to perform for each round. To avoid this from happening, we could make two types of allowable actions. In this case, one is when the player is around and one is when the player is not. In this case, we can prevent the enemies from attacking each other.(enemies attacking each other will be

fun but not when npc is dead so the player can’t get the hint or help from npc to proceed the game). The problem that the enemies picking up item randomly will be tackled as well by doing this.