# Introduction:

In the Game Engine package, the codes and entities are designed in a way that could guide us get design ideas from it and help us meet the requirements in the assignment specifications. Nevertheless, the system that we are trying to design should be extended and updated based on clients’ needs and demands. In this report four possible improvement to the system has been explained. In addition, some positive points about the game engine package has been demonstrated along with the design principle which can justify these positive points.

# RECOMMENDATIONS: 1. Having a getter method for actor’s hit point (e.g. getHitPoint()). There are some ways that we can do the quite same implementation but it will not be a good design. For instance, creating a static variable in the Actor class and try to use it in other classes. For future implementation it may be possible for performing some action from actor side the hit point of an actor is necessary. E.g. actions like health booster if the hit point of the actor is less than 30.

# 2. The world class tells the allowable action for the player to be updated if the item is in the ground or if the item is in the player’s inventory. Since for some actions such as CraftAction() is only performed when the player got limbs in its inventory. If instead of one allowableAction we add two allowable actions for Items, one for items in the ground and another when it goes to the player’s inventory, some of the Action implementation will be easier by using allowable action.

# 3. In the Engine package a ground can have a sub-ground (background) process. In this case actors (invisible actors) can process their turns in the background which their process cannot be seen by the user since the top ground will only be visual for the user. One of the benefits of this improvement is for some actors who has the same feature of mambo Marie and they get vanished and invisible sometimes and using this implementation their health will be stored instantly without saving it in any variables.

4. Having a class that handles all the maps and current actors and current items on the map would have made life easy. Methods like *getAllActorsOnMap(GameMap map)* or *getAllItemsOnMap(GameMap map)* that returns actor objects and item objects respectively would have helped greatly.

# Positive views:

1. Some classes in the engine package (e.g. World and GameMap) have protected attributes and methods which they can be used in those classes’ sub classes and using those attributes new public methods can be made. In these cases, the proper encapsulation has been used because if those attributes and methods were private we couldn’t use them in the sub classes. Also, since these attributes are meant to be used in sub classes not anywhere else, the attributes should not be public too. In addition, providing protected attributes in the engine package classes gives programmers this ability to inherit from their classes (since the classes in engine classes cannot be modified.)
2. Another good design which has been used in the engine package classes is using privacy leaks for some getters methods in engine classes which return a private attribute from the class. Such as getInventory() method in Actor class which returns the unmodifiable list of the inventory. If a programmer wants to modify the private attribute from a getter method the system calls an error. For avoiding the error other classes should create copy of that attribute of and modify the copy to fulfill their needs from the classes private attribute.
3. Some of engine classes has implemented the interfaces inside the interface package. Interfaces should not contain too many information (code) but they can be used for some simple but useful functions for instance in the CraftAction can only be performed if the Actor is a crafter since the Actor class does not provide such information and since a Boolean function can decides who is a crafter (public boolean isCrafter() method) and this method can be used in the ActorInterface interface.
4. The way that engine package has been implemented which it is unmodifiable can be demonstrated as Open/Closed Principle (OCP). This principle has been implemented since the package classes should only be extended for adding entities to it which means it is opened for extension. However, it is closed for modification (General rule for not modifying the engine classes directly.)
5. In some of the engine package classes Abstraction principle has been. And when the subclasses of those abstract classes are created, the Override of the abstract methods should be created as well. The advantage of using this principle is that the developer only need to know about the abstract method and they don’t need to know about other methods. For example, in the Action abstract class has two abstract method(menuDescription() and execute()) and when a developer create a new Action class such as CraftAction() (CraftAction is a sub class of Action) the developer just need to override abstract methods. And other methods such as hotKey() will automatically be generated (in other classes).
6. The way interfaces were implemented for major components of the game was really nice, even though it was a rule that we were not allowed to changed anything in the engine package (I’m assuming it’s because we don’t know what we’re doing and if we changed something, we might screw the whole program up) we were still given freedom to an extent to still be able to implement methods that we’d add to the engine classes.