Recommendations for change to the game engine

Overall, the game engine package allows us to fulfil the requirements needed for assignment 1 & 2 as well as assignment 3. While implementing the features throughout the assignments, we managed to complete them by working around the engine package. Therefore, we did not find any classes that require a huge change.

There are several abstract classes which are significant such as Ground, Action and Item. They are basically base classes which contain abstract methods that need to be implemented should we have a subclass that inherits from them. For example, DoNothingAction or DropItemAction extends Action. Both are required to implement execute and menuDescription method. As every Action, or rather, every class that inherits from Action is required to implement those abstract methods. This basically reduces our time needed to manually code out certain methods since Eclipse would just tell us to create the unimplemented methods and automatically generate the method header for us. This adheres to the design principle DRY (Don't Repeat Yourself). The abstract methods that are to be implemented also allows us to implement different functionalities, for instance, PickUpItemAction and DropItemAction, one's execute method picks up an item, the other one drops an item.

The Action classes also adhere to the Single Responsibility Principle, which is obvious. Every subclass of Action does one specific task like the previously mentioned PickUpItemAction/DropItemAction. Action subclasses in the game package are also the same. This makes modifying codes easier, if we were to change the task/functionality of an Action subclass, we would only have to focus on that specific subclass, nothing else. For instance, any kind of attack that needs to be modified, depending on what Actor, can only be done in AttackAction.

Another principle which is the Open/Closed Principle. This principle states that classes/objects/methods should be open for extension but closed for modifications. We can add new methods/attributes, in a way where we do not need to rewrite/heavily modify old methods/attributes. Such an example is in the class Location. The Location class basically has methods which are related to a specific position/coordinate on the Game Map. If we could modify it, we can easily add new methods/attributes without even touching the old methods.

Furthermore, the classes in engine package also adhere to the Liskov Substitution Principle. This principle that defines a superclass object should be replaceable with a subclass object, meaning that the subclass object is required to behave exactly like the superclass object. While coding throughout the assignment, a lot of features are related to the Item class. A simple example is the Mace class. It represents a Mace that extends WeaponItem which then extends Item. In the Action subclass PlayerCraftAction where the Player is required to craft a weapon, a Mace class is created, which is then stored inside the Player's inventory and at the same time, a Zombie Leg is then removed from it. As we know that addItemToInventory and removeItemFromInventory both accept a parameter of type Item, both the Mace/Zombie Leg object are successfully passed into the methods, as they are a subclass of WeaponItem, which is just a kind of Item. They behave like an Item object, thus there will not be any kind of error.

Another very useful feature that adheres to the Interface Segregation Principle is the Capabilities feature. In the interface Capable, there are three methods has, add and remove Capability. Any class that implements the interface would be required to implement them, which in this case is done in the Actor class, as Actor implements Capable. This then allows any subclass of Actor, such as the Farmer to be able to have a capability added, or removed, which are necessary methods when we are modifying an Actor's capability.

In conclusion, the classes in engine package adhere to the design principles which were taught in this unit, which is why there weren't a lot of issues while implementing the features, other than us misunderstanding certain methods/classes when we were first introduced to the package.