# Game Engine Comments and Review

**Extremely accessible and reusable methods:**

The game engine allowed for access to a plethora of pre-existing methods which made reusing code easy and preferable. As an example, the ability to utilise map.locationOf(actor) allowed for a highly reusable way to determine or perform operations utilising the location of one or multiple actors. This is further demonstrated by the capability methods which universally allowed all terrain, item and actor based objects of the code to use simple enum constants to represent a state of being or a capability thus allowing for a cleaner and less dependent related method of allowing two classes or objects to interact through downcasting or other means.

**Encapsulation of classes:**

The game engine provided extremely explicit classes pertaining to even slightly unique objects such as a weapon and intrinsic weapon. As opposed to essentially giving a single weapon class god capabilities over all variants of a weapon, creating both a weaponItem class and an intrinsicWeapon class to encapsulate different kinds of weapons allowed for greater ease in making useful distinctions between the two kinds of weapons when programming how they would function within the game.

This encapsulation is further shown in differentiating a single action from actions. The game engine provided two very useful classes action and actions which aided in not only allowing for easier creation of additional actions but also to alter potentially which actions may be allowed at a certain point of the game such that the Actions class could be utilised. Because of this encapsulation of these different classes, it was easier to treat an action and an allowable action as different things.

**Abstraction of code and helpful abstract classes and interfaces:**

The game engine contained much complicated code in areas such as the Actions class, FancyGroundFactory and other classes. However, these classes were designed in such a way that they could be instantiated in a simple way such as simply entering ground types into the FancyGroundFactory and performing all the complicated actions in the background of the code.

The use of abstract classes was also very helpful in reducing repetition within extension code as much of a child classes' attributes and methods would be present within the abstract class. In the case of the Actor class, many of an actor's core methods do not even need to be referenced within a subclass such as a stegosaur or player because these have been placed within the game engine's abstract actor class allowing for DRY code and a lack of repetition. Additionally, these methods were easy to override and were not rigid in their design such that alterations to a

method such as the consciousness behaviour was made easy upon needing to add thirstpoints as a factor in unconsciousness.

The interfaces were also designed in such a way that they could be implemented to manage the behaviour of a class which would implement them. However, it is probably worth keeping these within the engine as use of the interfaces frequently causes interface pollution as the interfaces are non-specific (as they are probably meant to be) and thus do not entirely pertain to the convention of interface segregation on a level that would reduce pollution. For example, implementing a method for trees and bushes would force the use or inclusion of said method in all other types of terrain which is both unnecessary and violates DRY by causing additional repetition.

**Criticism on game engine's menu description for actions function:**
Each action that can be made has a description that can be placed on the game menu. However, the way in which this function is constructed contradicts the requirement to have actor locations displayed with actions. The method only takes the actor as a parameter and thus restricts the ability to give that location unless a new constructor is made. The problem is easily resolvable, but in most cases results in an unused constructor appearing in action classes for this method.