Engine recommendations

Jiten Verma and Ayesha Ali

Engine well-performed

SOLID principles

All the classes in Engine follow the SOLID principles. For example, for each functionality a class was formed whenever possible in accordance with the SRP like making a single class to keep track of the locations of all the items and actors on the map, a separate class for the map and so on. The use of LSP is also evident as we can override methods in any Engine class without compromising its functionality.This has been done multiple times by overriding methods of Action in its subclasses and IntrinsicWeapon overriding Weapon.The the use of interfaces for each extensible feature like separate interfaces for Behaviour, Action and Actor allows functionality to be extended based on our needs, following ISP. It also follows OCP as these interfaces cannot be modified but extended based on our needs. It also follows DIP as the use of interfaces ensures that changes in the sub classes do not affect the base classes like the changes in a subclass of Actions would not affect the interface.

Advantages
The usage of SOLID principles ensures that code follows DRY principle and also allows the code to be extended easily for further features without changing the important sections of the code. Also it allows us to know which classes are to be extended to implement a certain functionality.
Also SRP helps with low coupling and high cohesion as details related to a certain functionality is stored in a single class which ensures that it has to interact with limited classes only.
Disadvantages
The SRP may cause formation of multiple classes which can make the system harder to maintain and understand.

Encapsulation

The engine has been able to encapsulate the data quite well. Most of the classes use private attributes, and some have been set to protective which the designers believed would have provided some aid to developers extending the Game package. Although the attributes have been given private or protected access modifiers, privacy leaks may still occur if getters returned the attribute or object themselves, as they can then be accessed outside of the given scope and modified. However, the engine package has ensured that the getter methods prevent the unwanted privacy leaks by returning copied attributes or objects which ensures the scope given by the access modifiers remains true.

Advantages
The correct display of encapsulation ensures that the engine code is safe from any unwanted privacy leaks and ensures that attributes and objects are able to remain within the scope given by the access modifiers. This also helps to prevent bugs within the code, for example an attribute or object is being modified by mistake, and this causes further errors within the system.

Disadvantages
This may potentially cause some hardships for programmers as they must spend extra time and labour to gain access to or modify certain attributes or objects, however this would prevent bugs within the program.

Command-Query separation

The engine package is able to ensure that there is a clear command-query separation as all the methods either modify the state of objects or return an action. For example in World class separate methods are used to modify state of multiple objects (processActorTurn (actor)) and to return a variable(stillRunning()).

Advantages
It is useful in debugging as you can determine how the function modifies the system or how its return value effects the system

Disadvantages
This is not always possible to implement as a function may be required to both return a value and modify the system.For example, in the case of execute(actor, map) in Action, the method modifies the actor object but it also returns a string telling how the actor has been modifies which is necessary as well.

Polymorphism and Inheritance/Extensibility

The engine package has demonstrated really good use of inheritance, extensibility and polymorphism through the use of Item, Ground, Action and Actor classes. Each of these classes has subclasses which extend from the base class to represent different objects around the map. For example, the game package extends the Action class many times to represent the different actions which may be performed by the Actors. This sense of extensibility has allowed programmers to use one base class of Actions, which has clear methods defined for execute and menuDescription. Furthermore, the Action class also demonstrates polymorphism as we can easily represent a subclass of Action as an instance of Action, as its primary methods are execute and menuDescription.

Advantages
The advantages of using polymorphism, inheritance and extensibility is that abiding by other good coding practices becomes more easier and natural. The classes are able to be broken down which prevents from having a god class, this also helps make the code easier to

understand and follow. Polymorphism allows us to simply get an action, and perform it by running the execute method, regardless of which action it may be. This allows us to have one simple and central method of executing actions, rather than something specific for each action, hence it reduces the labour required to build such a system.

Disadvantages
The cost of reducing labour during the implementation implies that the system requires more time to be invested in planning and designing of such a system. The system must be very well designed to ensure that the system retains the mentioned abilities.

Engine improvement recommendations

Mutators (getters/setters)

The classes in the Engine do not have getters/setters for some of the useful attributes they contain. This was quite difficult to work with, as we sometimes required important detail stored within attributes, however we were unable to access them. For example, for the sniper weapon, we wished to use the protected attribute named actorLocation. However, the only way to access this within the game package would be through a getter or by creating a new class which extends GameMap and then converting the map to that class. The latter required high workload and could have caused potential unwanted dependencies or errors.

Proposed change
The proposed change would be to introduce getters and setters more often for attributes, especially for attributes that would be highly useful.

Advantages
Implementing the proposed change would allow developers to gain access to important attributes which would help reduce their workload at later stages.

Disadvantages
The disadvantage is that if all attributes have mutators then the purpose of private attributes would be defeated as we can easily get and modify the values of the attributes. This could leak to potential privacy leaks.