Recommendations for change to the game engine

Problems Perceived

Handler subclasses being required, to reduce code duplication and to introduce new functionality into existing classes, is an issue.

As the engine was not allowed to be modified, certain functionality that could have been implemented in one class had to be spread across two classes, a base class and a handler class, to add new features

Examples from our code implementation:

Eg -

- WeaponHandler
- UtilityGameMap
- GameHandler (subclassed from World)

All of the functionality in these classes can be implemented within one class cohesively, if the engine allowed access to key methods and attributes.

WeaponHandler was written to reduce repetitive code, as each weapon class needed to separately have the same code, because the engine could not be modified. The creation of extra classes increases complexity, unnecessarily, and also leads to classes with a certain amount of coupling.

Suggested changes

Classes, methods and attributes that are helpful and make the code more the codebase to be extensible, need to be more accessible. While the engine must encapsulate the internal workings of the game and provide an interface for the game code to access the low level code from a high level, certain classes such as WeaponItem, or certain actions such as DoNothingAction, do not belong in the engine, per se and if they are in the engine, they should provide greater access to modify the code and make it extensible.

Advantages and Disadvantages

Advantages:
Functionality can be achieved in one cohesive class, separating related concerns to itself. Reduces unnecessary encapsulation boundaries and reduces code complexity

Disadvantages:
More access to an Engine class might lead to classes being privacy leaks and potential violations of the single responsibility principle as properties of a class might be passed easily to another. Also, This might cause an impact on the Dependency Inversion Principle, as low level details can be exposed at a higher level.

Problems Perceived

The usage of interfaces to prevent downcasting of code, and encapsulating all actions under a common method of a class useful. However, as interfaces enforce that all subclasses of a class should implement the method, and because the code that is implemented, is implemented at the highest levels of the inheritance hierarchy, various classes are enforced to contain methods that are somewhat irrelevant to the class.

Eg - Item Interface has an “asCraftableItem” method that checks if an item is craftable. This method needs to be implemented for all items, including corpses (which are a portable item). These were set to return null.

These were required to be implemented at the highest level in the inheritance hierarchy as various classes used common methods to achieve functionality

Eg - target.death() for killing all forms of actors. Therefore, for the compiler to not return an error, the relevant method needs to be accessible to all of the classes in general.

Suggested Changes

The engine enforces that certain exact methods are required to achieve functionality. As various classes are subjected to the same method, all of them need to implement the said method, even though the method is somewhat irrelevant and affects the cohesiveness of the class. Due to this interfaces are required to be used, to enforce these methods.

Therefore, a suggested change would be to produce different pathways to achieve functionality, rather than one single dedicated method.

This will help make the code cohesive.

Advantages and Disadvantages

At the existing implementation, it is easy to implement a general case for all scenarios. Also, code runs through a well defined interface consisting of methods. This makes the code clearer.

However, as various classes use more or less the same interface, different concerns are not well separated.

Problems Perceived

Constraints due to existing code structure. The engine package produces a well defined interface for achieving certain tasks.

Eg - Action classes for actions
PlayTurn methods for dedicated turns

Menu taking in an actions list

At most places the well defined interface abstracts tasks really well and allows easy extensibility of code. However, in order to adhere to the existing structure in achieving certain tasks, classes had to be created for very trivial code.

Eg - For anything to be displayed in the Menu provided in the engine, it must be an action class. So, all menu actions however trivial needs to be written as a dedicated class

This leads to requiring more classes than is necessary.

Suggested Changes

Introduce other pathways that reduce unnecessary complexity, but retain encapsulation. However, the existing implementation, abstracts the code really well.

Advantages and Disadvantages

Modular code structure is helpful for extensibility. The existing implementation is helpful in this aspect.

However, it introduces encapsulation boundaries for achieving trivial tasks.