**FIT2009 ASSIGNMENT 3**

**ENGINE UPDATE RECOMMENDATIONS**

A broader range of interfaces would be preferable. Interfaces can be used to extend the range of options a developer has in using the engine, without risk of breaking fundamental engine code through direct alteration. We would suggest adding interfaces for the Menu, Location, WeaponItem and GameMap classes.

The Menu class in the engine is restrictive in its implementation, and could be expanded to accommodate a much broader range of menu options. Currently, the menu only accepts Action subclasses as parameters to display, and returns only those Action subclasses once a choice is made. This works well when the Menu is used only to determine what single action a Player should make in a turn, but when more complicated turns are being developed that involve one or more sub-menus, this narrow range of possible choices restricts other design choices across the platform. This problem could be remedied with a Menu interface, allowing the developer to overload the showMenu class and allow the user to implement String or Boolean returns from a Menu choice. This would also not need to include extensive re-writing of existing code, as showMenu could be refactored to call another menu class, which would contain any code not specifically reliant on the types of objects passed as menu choices.

The Location class could similarly benefit from additional interfaces, as some features may need to interact with actors or items at specific locations in unique or unpredictable ways. While this is not called for in our design currently, it is easy to imagine and just as easy to resolve.

WeaponItem could benefit strongly from a usable Interface; doing so would allow more unique or unconventional weapons to be implemented in the engine, without strictly requiring a new abstract subclass of Weapon. While WeaponItem does have an interface Weapon, it is in the Engine package, and so cannot be used or modified by students.

GameMap could benefit from having an interface, as this could mean if a developer in the future wanted to add methods to this class, they could do so. Many possible features in any Roguelike game will interact with the world map, either by needing to know the locations of specific things in that map or to change the properties of certain tiles, and so allowing these features to be more easily implemented would be beneficial.

Interfaces would be useful not only for usability, but to refactor the engine to more closely adhere to the Dependency Inversion Principle. Having more classes such as GameMap and World depend on an abstraction between one another would be a positive design choice. This has already been implemented in several areas, such as the Actor abstraction separating dependencies between its subclass and all other classes, but there remains some room for improvement.

Another area for improvement lies in the implementation of the Item allowableActions method, and its use in World’s processActorTurn. In its current implementation, an Actor can use any allowable Action associated with an Item simply by standing on top of the item. There is no distinction between standing on an item and holding it in terms of the Item’s allowable actions, which is unintuitive and restrictive. This problem also cannot be resolved through using the Item interface, as the code which checks for allowableActions in items at a user’s feet is in the inaccessible World class. Implementation of Firearms could have been made much easier if their actions could be added as allowableActions, as adding those actions to the player menu would be already built into the game code. However, it’s impossible to properly fire a gun by standing on it, so this could not be used to implement Firearms.

There should be a method in the GameMap class that allows us to efficiently and safely retrieve the ActorLocations iterable of all the actors that are currently on the map as a collection that we can use, currently we have to traverse the map and manually do this, which is inefficient. This can be useful in implementation of the end of the game, as we should be able to simply call a method such as getActorLocations() for example. In the use case of checking whether the player has lost, this can be useful as we can simply traverse the (much shorter) ActorLocations iterable to check if there are any humans remaining on the GameMap rather than traversing through the entire map meaning that the number of loops will be to the power of 2 (traverse through each x,y coordinate).

The tick() method that is defined in many abstract classes such as Item and Ground that is executed on each turn could be improved by having a basic incrementing value that tracks the time. Even though this may serve any useful purpose, it could be useful in providing a guide for people who are implementing this method in child classes of how they could use this method to implement a feature involving the tracking/passage of time provided in the Assignment specifications.

Classes in the engine package could benefit from being categorised into various different sub packages, such as an Action, DoNothingAction, DropItemAction, MoveActorAction, etc. in a subpackage with all the engine Actions. If we apply this method of breaking the engine up into subpackages, it would be easier for a person new to the engine to familiarise themselves with it.

The name of the ‘Capable’ interface should be changed so that it is adhering to the naming convention applied for the rest of the starter code, and therefore should be renamed something similar to ‘CapableInterface.’ This will ensure that if any changes are made to the engine in the future (probably not by a student) that it will be easier to see that it is in fact an interface. This also applies to other interfaces defined in the engine package such as Printable and GroundFactory.

The engine was moderately easy to understand, as the code in the engine was documented with descriptions for each class, the class’s attributes, interface, and each method and it’s parameters and result type. However, the descriptions for classes, constructors, methods, etc. could have been benefited by explaining how they interacted with other components of the system in more detail. For example, “Base class for Actions. These represent things that the player can do.” Is inherently correct, but is still quite vague as a Zombie can also perform Actions, however Zombie(s) are not the same as Player.

Overall, the engine did not have any bugs and was very usable. However, after implementing the changes stated above, the engine will be easier to use for developers.