Recommendations for change to the game engine – Assignment 3

Group: 404

1) Nur Farhanah binti Baharudin (29969470)
2) Morad Abou Shadi (29799260)
3) Lai Khairenn (29959381)

Bad points about the engine:

1) **Problem**: Many classes in the engine have protected attributes instead of private attributes, and other classes access these protected attributes directly instead of using accessors which can cause privacy leaks. For example, in the World class in the addGameMap method, the actorLocations of the gameMap is set by directly accessing the attribute. Since most instance variables in the engine do not have accessors, there is also another related problem where code in the game package is unable to access important attributes when needed. For example, in our ShootingSubmenu class, we had to get all the enemy zombies on the map so that the player can choose one as a target for the Sniper, but the actorLocations attribute in the GameMap has a protected access modifier so we could not access it directly but it also does not have any accessor, so we were forced to loop through each location on the map to find any enemies.

   Solution: Change the access modifier for all instance variables to be private, and set public or protected accessors and mutators for them.

   Advantage: We will be able to access information from outside the engine package which can help simplify the code like in our case, we could just get a collection of all the actors and find any enemies using a for loop, instead of looping through the whole map which can be a problem with big maps. This also reduces the scope of these attributes and prevents unauthorized changes to them, since we can check for preconditions in the mutators before making changes to the actual instance variable itself, which follows the design by contract principle.

Disadvantage: Not all the attributes in the engine package will require public mutators because they will only be used within the engine package itself once, or even not used at all, like the item instance variable in the PickUpItemAction class. In cases like this, the attribute should just be set to private with no accessors and mutators so the developer will have to exercise discretion when choosing which attributes to set mutators for. Setting getters and setters for attributes can also introduce privacy leaks in the code, especially when the getter is returning an object type of another class. If the getter does not return a copy, but instead returns a direct reference to the original object, then any changes made to the object in the caller will also be reflected in the original, which can be unwanted in some cases.

2) **Problem**: The engine World class does not have a method to modify the ending conditions of the game and also does not have an attribute that keeps track of whether the player wins or loses the game. These two functionalities are essential because every game will have different ending conditions, and the game is also either won or lost when it ends. In our assignment, in order to modify the end game conditions and also have different endings for when the player wins or loses, we had to create a whole new MyWorld class that inherits from the engine World class just so that we can override the existing method to set new ending conditions and also introduce the new attribute to record whether the player wins or loses when the game ends so that an appropriate message can be displayed.

   Solution: Create a new method in the World class that can be called from outside to end the game, with the method input parameter being a boolean that represents whether the game is won or lost.

   Advantage: Future developers using this engine will definitely have their own game ending conditions because all games are different and this method will give the developer easy control over the game ending conditions with the correct message being displayed when the player wins or loses without having to inherit from the World class and repeat the same code every time. This also allows the developer to set interesting game ending conditions, such as a time attack game where the player has to complete an objective within a time limit or navigate to a certain location like a rescue helicopter before he can win.

Good points about the engine:

One point that I liked a lot about the engine is the Action base abstract class that we can use to represent all the actions that can be done by the characters in the game. This class allows us to encapsulate all the special functionality of the actions of the actors inside the specific Action class and helps us to avoid creating a god class by cramming all the actions of the actor as methods inside the actor class itself. This also reduces a lot of repeated code because the characters in a game often have common overlapping actions that they can execute, so having an Action class for each action allows different actors to carry out the same action without coding it many times. The Action class can also be used nicely in conjunction with Behaviours to make NPCs in the game easier to implement because we can encapsulate the checking of whether the actor can execute a certain action inside the behaviour. This also prevents the actor classes from becoming a god class because they do not have to know about the conditions of when certain actions can be executed, and the Action class also just contains information about how to execute the action and not whether the action is valid.