**Recommendations for change to the game engine**

At first, when our group was given the game engine, we were overwhelmed by the number of classes and the engine class diagrams modelling the interactions between the classes. After getting familiar with the classes and their usages over three assignments, whether it was planning or implementing our features, we managed to come across many problems and frustrations in using the engine. However, the game engine has also made it easier and more convenient for certain tasks to be completed.

One of the problems we faced is that we could not access the ActorLocations attribute used in the GameMap class. The ActorLocations attribute contains all the locations of the Actors on each GameMap instance and is used in multiple tasks, such as finding Actors on the GameMap to be aimed or shot at with the Sniper Rifle. The ActorLocations attribute is protected and can be accessed by inherited classes. However, it cannot be accessed outside of its inherited classes because there are no accessor methods. This makes it hard to obtain the locations of every Actor on every GameMap. As an alternative, we had to search each Location on the maps to look for Actors.

We propose to add a class solely to govern the Locations of all Actors. From this, we can inherit this class to be used in the game package freely as ActorLocations is a protected attribute. It would make it easier to access the ActorLocations without causing any privacy leaks. Furthermore, it would reduce duplicated code, as we would not need to repeat the process of looping through every Location in the GameMap to find the Actors. However, it would increase dependencies as it would mean that this new class depends on the GameMap to update its ActorLocations. Besides that, it would go against the design principle of grouping elements that must depend on each other together inside the encapsulation boundary of a class.

Another problem we encountered is that the ActorLocations attribute in the GameMap class contains the Locations of every Actor from all maps in the World class, instead of only a particular GameMap. This makes it hard to keep track of Actors in certain GameMaps for tasks like ending the game, where we could not end the game because all the Humans specifically in the compound are killed. To work around this, we refactored the compound GameMap to keep track of its own Zombies and Humans to implement this feature.

We propose to change the way ActorLocations is designed, only to include the Locations of Actors within a certain GameMap. Therefore, each GameMap is responsible for its own ActorLocations, and keeps track of Actors within its own map. The advantage of this design is that it would be easier to keep track of map-specific events involving Actors, such as the one mentioned previously which effectively reduces redundant code such as the way we changed GameMap to keep track of its own Actors. However, it creates a disadvantage that there would be no way to track all Actors at once without having access to all the different GameMaps, which could be a problem for different tasks.

Besides that, another problem that we had while implementing the features is that there is no way to access an area of Locations besides using Exits. For example, in the Shotgun task, the only way we could scan the area to be blasted by the Shotgun is to use the Exits of each individual Location one-by-one, like a domino effect. This created complicated nested loops as each direction shot would produce different Exits and involved using a lot of Exits and their methods to obtain the next Location to be added to the area affected by the blast.

A design change we propose to address this problem is to add a method or class that can help to traverse between Locations easily. A class similar to the Exit class can be used to access adjacent Locations in an direction or area beside the selected Location. This would make it easy to obtain areas of the map to apply certain effects and reduce code complexity by removing the need for nested loops and many Exits. Besides, it would reduce repeated code if obtaining the area is used in multiple features. However, this design change could be very difficult to implement and it would be redundant as Exits would be enough in most cases.

There were also many aspects of the engine made developing the game easier. It helped prevent us from reusing code and creating unnecessary convoluted code. The use of Interfaces for widely used classes such as Actor, Ground, Action and Item helped us easily add additional helper methods to be used by instances of these classes. This saved us time and helped us reduce the redundancy implementing these methods individually to every class. It also helped us to avoid downcasting when using methods that inherit the engine.

The use of the tick method by a variety of classes in the engine such as Item, Ground and GameMap, helped implement features that require the passage of time. This helped us create useful counters in many of the classes for the required features such as Crop and Mambo Marie.

The Location and Exit classes helped use obtain specific areas that are required for implementing features. Since Location instances can obtain Exits and Exit instances can obtain Location, we did not need to create new instances to access the required class. This especially helped us in developing the AI for the behaviour of Actors. The Farmer is able to know condition and suitability of their surrounding to select their most suitable Action from their behaviours. Humans are able to check for useful items on the ground such as food.

The NumberRange class was very useful as we did not have to individually count and store the range, minimum and maximum values of the x and y values of the map. By getting the ranges of x and y values from NumberRange, we are able to iterate through to the map. This helps us to implement features such as Mambo Marie spawning zombies at random locations in the map.

The use of the Capabilities class in the engine was especially useful in differentiating the actors on the map. We did not need to create new methods on checking for actors or have to perform downcasting on the actors. This was especially helpful in finding targets for actions that involve attacking or targetting. This also helped us in keeping track of the number of the number of actors for each team, which was required to help us end the game.

In conclusion, the studying and usage of the classes in the engine that we have did has helped us in applying the design principles we have learnt in this unit. We are more conscious on applying design principles when designing and developing programs. We have also improved our understanding on concepts such as inheritance and encapsulation. This has helped use improve our skills in using object orientated programming.