Recommendations We Could Give For Changes In The Game Engine

One of the problems that we run into when doing these tasks is one that is seen in this Assignment 3 , which is the lack of options given to us to end the game properly without having the player die. This is because the run() method in World class uses a while loop that checks if the player is still on the map in order for the whole game to function. This means that we can only end the game if the player dies if we do not create a new class that extends World engine class, and also the stillRunning() method has a poor design decision as it has to rely on only a single variable which could have adverse effects if other programmers want to implement more complex features to the game. For this, we recommend the stillRunning() method in World to check for all other ZombieActors as well, such as whether there are still humans or zombies on the map. The advantage from this is that we are able to reduce the dependency on the player variable, which is according to the design principles that we learned in our lecturers. However, the disadvantage is that anytime if a programmer wants to add new Zombie or Human class, they would have to modify the stillRunning() method as well to keep up the date with all ZombieActor classes. This also applies for any application that uses a player Actor as the variable in order to run the game.

Other than that, another problem that we run into is that the player is only able to attack ZombieActor's which have the ZombieCapability of UNDEAD and not anything else. This means that whenever a programmer wants to add an enemy ZombieActor that is not of ZombieCapability of UNDEAD, such as a UNDEADBOSS type, then the player would not be able to attack it, which also implies that this violates the DRY design principle that is covered in lectures. Fortunately, we recommend coding in processActorTurn() method to create a new AttackAction for any Actor that is not of type human or any ZombieActor's that are deemed as friendly NPCs by the user using a for loop. This could also be implemented in other applications that have friendly and enemy types. Also, one advantage to this implementation is that we are able to conserve the DRY design principle, however it would also make the code in the method more complicated to understand for future programmers when they see the code.

Moving on from that, problems that we have faced also include having a way where the user is able to return the main menu. This is because the application does not support having submenus and returning to the main menu if they are able to call submenus, thus we have to add a null into the actions list so that it would be able to return the previous menu when it meets a null. Our recommendation for this problem is to add a new method in Menu that is able to call back the previous menu that was used before so that it is easier to add more aim and shoot actions for different weapons and we would not have to add new classes for certain functionalities which would make it harder for other programmers to understand the application in general.

Finally, although our experience with the engine brought forth many problems that we had to face, it has also brought a few positives for us. This is because these problems allow us to make up more creative solutions in order to solve the task at hand since we are not able to modify the game engine. One such example would be using the Interface package such as ActorInterface in order to write methods for all ZombieActor without adding it into the game engine. With this, we are able to add methods which were not previously added by the parent class in the game engine

into its subclasses, the subclasses would also show errors if they have not implemented the methods that were written in the interface package, which also helps so that we would not forget to implement them. In conclusion, I believe we still benefitted from doing this assignment even with or without these recommendations in place.