Assignment 3 Changes Recommendations & Opinions

– Tony Nguyen & Owen Austin

Tony:

In my opinion the engine package has been really well made and does what it should from the perspective of what a roguelike game should be and just testing/playing around with it. With that said however, from the perspective of someone that has made implementations to the Game I felt that some parts (especially blocking any change of the engine) made it so I had to break some principles to create the implementations. Other than that, I really liked the encapsulation boundaries that kept related information as well as the usage of Actor as an Abstract class. Finally, I think it should be stated that I mostly had a positive experience with the engine package but there are still some things I would recommend changing.

Owen:

In my opinion the engine is quite well built. It perform it tasks well on every part of this game. Moreover, it is very easy to extend or implement changes or making new elements on the game due to the open principle applied on the engine. The engine has a good but moderately complicated upon seeing in the first time. I personally find it hard to understand when making assignment 1 and 2 but almost totally comprehend it near the ending of assignment 3 where I quite enjoy using the engine. I think that this engine might need some changes so that it will fully facilitate developer when developing the game to the next stage and version.

*Problem*

Tony:

One of the problems I encountered was that I sometimes felt like I was going against some of the principles with my implementations, but they were necessary. I feel like this often happened because I could not make any change to the package or even extend what we currently had in it. To use an example to explain my point my TraverseAction implementation would’ve been more optimised to follow the reduce dependencies across encapsulation boundaries should it just be in the Engine package alone. Traverse action needs Actor and GameMap along with Location which are 3 dependencies across the boundary, I could reduce it to 1 if Traversection were in Engine.

*Proposed Fix*

I think you should allow for people to be only able to add onto Engine package (Not modify the current but at least add onto). Although a small change it would help in certain design choices and also allow for us to maintain the principle of keeping related info together.

Owen:

There are some problems that I faced which are hard to check due to the policy of cannot change the engine. In one of my problems when developing assignment three, I’m not sure why a certain zombie can’t be killed. It’s hard checking it because there is no way of the game to know a hit point of the actor because there is no getter for the healthpoint. There are a lot of similar problems when testing the game while developing it.

*Proposed Change*

I think that we should be able to add more things to the engine that help future editor (adding more, not changing or reducing). The engine helped a lot when making the extensions but it is not helping much when checking or debugging. Instead of having the game printing and know like what when wrong, we need to debug line per line.

Tony:

Pros –

* It would allow for us to adhere to the principles including and are related to reducing dependencies
* Allow for us to demonstrate that we do know our principles (aka it’d be easier to point out that this person knows the principles in the lectures)
* Simplify potential future tasks and provide a sort of hierarchy structure (With engine at the top)

Cons –

* More difficult to mark as we’d have to point out what we added to engine
* People could potentially get away with changes to the engine class that the marker won’t notice (Although I should specify that I only want to add onto the package not make changes to the current)
* Simple Relationships between certain classes could look complicated on a UML (E.g. Action class has a lot of subclasses but is easily viewed since they are seperated)

Tony:

Positive Experiences

Other than that recommendation I can’t really think of anything else, I really liked seeing how the encapsulation boundary not just in packages but even the classes themselves kept the related information together for their respective needs. Basically, I think I liked how each class were kept to a single task so that only the passage of information goes to what was needed. I believe this shows that each class has been in line with the single responsibility principle.

Along with this at multiple times was I thankful the abstract classes were exact and concise as I would often extend them. What I mean by this was I am glad that the classes that were given to us were following the Open/Closed principle, specifically I am thankful for the open part. I know that an abstract class is meant to be exactly that, Open for extensions but they could’ve given us nothing or at least told us you couldn’t extend them. So for that I am grateful that the abstract classes look so clean and easy to understand.