Breakout Group

- Jason Toh Zhern Wee
- Tah Wen Zhong

Recommendation for change to the game engine

First and foremost, the extensive restrictions that have been placed on the engine code has proven a challenge for our group as we have to make changes around the engine code but not directly as per assignment requirement. To solve and provide some of the functionality of the assignment specification. ***Some of the classes in the engine code lack the necessary methods needed for our code such as the attribute hitpoint in Actor class should have an accessor method.*** As a result of this, much of our earlier design plans have to be scraped of due to the lacking of methods provided in that class.

Secondly, due to inability of modifying engine code, we were forced to create subclasses of the particular engine code in order to override their methods. This was not optimal in a game development design as some methods have to be completely rewritten and possibly duplicated which violates the DRY principle. Furthermore, additional classes have to be created each with their functionality where we would need to implement their methods and override their methods just to change one line or two lines of code. ***To give an example, the World Class in the engine method is closed for modification and not open for extension as the run method does not accept parameters to their code*. Since ending a game is a vital part and should be a smooth process in any game, ending the game in the run method should be allowed for extension where there are additional ways to end a game, but it is closed only for when players die in the game. As such we have to extend the world class and completely rewritten their run method and it makes the whole process much more complicated than it should be if restrictions on engine code weren't placed. Hence, my recommendation is to relax restrictions on certain parts of the engine code such as classes where the students are more likely to use (such as Actor/Location) and place restrictions on classes least likely to be used (Printable/NumberRange). An ***advantage to this change is that the code will be less sophisticated than it should be and it explores overall understanding and fundamentals behind the project they are working on.***

Another design changes our group would like to highlight is to place or group classes that are linked together in the same folder in the engine code. Most of the time, our group had to revisit and examine the code as ***the linkage to each other classes were not clearly stated or outlined***. Furthermore, some parts of the code were confusing as little to none description was provided for it. It left our group to make assumptions as they were no way to test out the code for it. Hence, providing more descriptive comments will help greatly for students to understand the code. The advantage of this design change is that students are able to quickly what classes are linked together as they are in the same folder. The disadvantage is that since they are in a different folder which translates to different package, it would be more cumbersome to import all the packages to do so.

Another design challenges our group faced was the similar name of classes in the engine code. An example of this was the "Capabilities" and "Capable" classes. The issue with it was that one of them is an interface and the other is not. ***At first glance, both seem to be the same without any clear distinction, but after inspection of the code. It turns out "Capable" is an interface.*** I would recommend Adding a tag to the name of the class such as "CapableInterface" to allow easier distinction between the two classes. This would be an advantage to the design of the engine code. We believed no disadvantage will arise from this design change.

All the difficulties stated above was what our group faced during this whole project period. We understand it may be difficult to implement some of the changes above but doing so may ease the burden or challenges for students.