1 REFLECTION

The reason why I have a substantial number of classes is because I always try to follow the software development principle called DRY (Don't Repeat Yourself). I believe that the less code you have then the less work you have to do in implementing and maintaining the program. As a result, I use multilevel inheritance and have a number of abstract classes in my program to provide the required functionalities for the derived classes. The way I designed it is that if two classes have the same attributes then I would make a parent class that contains the attributes. This would allow me to eliminate duplication of accessors and mutators for that attribute.

I spent a large number of my time in the design phase. The design phase started with me coding the Monster class. I started with the Monster class because Player class has a reference to monster and evolved monster. Therefore, I started with the Monster class and move on to building the evolved monster and boss monster. However, both BossMonster and EvolvedMonster class have the same attribute which is the special power modifier. Therefore, in order to eliminate duplication for the special power modifier accessor and mutator, I created an abstract class called HigherRankMonster. This design allows me to simplify my BossMonster and EvolvedMonster class by passing the appropriate type, skill level, health, and special power modifier to the HigherRankMonster constructor.

After finishing all the monster classes, I move on to coding the Player class. I encounter the same problem with Player and Monster class that is the two classes have the same attributes (skill level and health). Therefore, in order to eliminate duplication for the skill level and health accessors and mutators, I decided to create another abstract class called Character. At this stage, I have not created the game logic because I am more focused on making sure that each classes perform its job properly. This is because it would probably get harder to debug the application if you already make the game logic and one of the classes does not work properly.

When I was coding the game logic in the GameController class, I wanted to use polymorphism to hold the content of the dungeon. This is mainly because my lecturer and tutor said that it would be much easier to implement the game if you use polymorphism. With my current design, I cannot use polymorphism. This is because I want to implement an Item class for the extra functionalities. My current design will not allow me to use polymorphism because the Character class can only refers to Player and Monster while the Item class can only refers to item. In order to solve this issue, I started to make another abstract class called GameObject. The Item and Character then inherit from the GameObject class which then allows me to use polymorphism to store the content of the dungeon. It definitely made my work easier because I can now refer to any derived classes from the GameObject class. I could also downcast from GameObject type to any derived class

type and invoke their functions. In addition, in order to respond correctly to a particular object, I create a protected class level variable called ObjectType in the GameObject class. This would allow me to properly detect and respond to a particular object.

Finally, I am pretty happy with my application design for assignment 3. This is because my design allows me to reuse code and reduce redundant code which increase the efficiency. Furthermore, the use of polymorphism made it easier for me to implement my application. In addition, if I have more time to improve my application, I would make the GameController class as a “Singleton”. This is mainly because I only need one instance of it to play the game.