Changes:

1. I decided to get rid of NPC abstract class and AttackableNPC abstract class in my implementation, because I realized that the FriendlyNpc class, PassiveNpc class and AggressiveNpc class do not actually have enough properties in common for it to be worthwhile to have two extra layers of abstraction between these classes and the Unit class. Having the two extra layers of abstraction would only add unnecessary complexity to the project.

2. There is only one EntityManager to manage all entities in my initial design. I ended up having a standalone manager for each type of entities, i.e. FriendlyNpcManager, PassiveNpcManager, AggressiveNpcManager and ItemManager. Currently these four manager classes all have very similar functionalities, namely updating and rendering the entities they manages. The reason why I did not create one manager class and have four instance of this class is mainly each different type of entities need to be updated with different parameters, and having only one manager class would result in many unused parameters in the code. And that just felt dirty to me. Besides, having four separate manager classes would also improve the extendability of my code.

3. Inventory class and HealthBar class in the initial design are not present in the implementation. Inventory can simply be achieved through an ArrayList, and due to the way I handle item pickup, there is no need for complex operations on the inventory. In the end, it is not really worth it to create a class for inventory management. As for health bars, the only functionality I need is to render the health bars. I ended up putting the render method in each type of unit class.

4. Added CSVReader class to help with reading data from csv files. FontLoader class for better font.

Difficulties:

1. The most challenging aspect of this project for me is to deal with cooldown/timer. Attack, dialogue display, fleeing, they all require slightly different behavior in their timers. So it’s been especially challenging for me to design a general purpose Timer class to facilitate all required functionalities. Another reason why I found cooldown was hard to deal with is that when I implement a game, I always think of each problem in a frame by frame manner, whereas cooldown required me to think of each problem in an interval of time. The different paradigms of thinking created the obstacles for me. I found it really helpful to have flag variables to indicate the change in certain states when a timer reaches a key point.

2. Another challenging aspect of the project for me is to delegate responsibilities to each class. I found it really hard to delegate responsibilities properly when I have

multiple layers of abstraction. I would say I actually spent half the time on designing the responsibilities of each class. In the end, I took a top-down approach, trying to keep common responsibilities in the ancestor classes as much as possible.

What I Would Do Differently:

1. I actually wasted a lot of time thinking about optimization at the design stage, which yielded little result in the end. I realize now that it is easier to consider and actually do optimization when you have an initial implementation than to have optimization in mind right from the start.

2. Start implementation early. It is significantly less enjoyable to do a project when you have three other projects due in the same week.