CENG443 – Homework 1 Design Document

Name: Orçun Başşimşek
ID: 2098804

1-) In this homework, I implemented a Zombie-Soldier simulation task with in fixed size environment. For administrative tasks about environment, I used SimulationController class provided by you. This class were responsible for implementing the methods for adding or removing any entity to environment and simulating all current entities's behaviors for one step. For the entitites that may take part in the environment, I used SimulationObject abstract class as a top most anecstor for any entity. According to my design, this class is reponsible for including all common fields for our entities and helper methods that can be commonly used by them. Our entities were Zombies, Soldiers and Bullets for this homework, and I created Zombie, Soldier and Bullet classes for them respectively. Of course, each of them extends the SimulationObject class (i.e. they are child classes of SimulationObject). Among them, I created Bullet as normal class because it is a real-world object for our environment and it can be initiliazed at Soldier related classes while a soldier firing a bullet. However, I created Zombie and Soldier classes as abstract class because they are not actually real-world concrete objects for our case. They are just generalization of any zombie or soldier and these two classes do not need to be initiliazed. For concrete kinds of zombies and soldiers, I used Commando, Sniper and RegularSoldier classes that extend my Soldier abstract class and I also used SlowZombie, FastZombie and RegularZombie classes that extend my Zombie abstract class. With Bullet class, these classes constitutes any real-world object in our environment. Apart from these, I also used enum classes that were provided by you. I did not change anything for ZombieState and SoldierState classes since they are fixed and easily usable for our tasks. Thus, I used them as is in my implementations. However, for ZombieType and SoldierType classes, I needed to add a double type field(named "value") and proper construction and getter methods to these classes since we need to keep zombie's and soldier's type as constant double variable as stated in Homework text. Thus, I associated ZombieType and SoldierType instances with this double value, and initiliazed my concrete soldier and zombie classes with this specific double value in their constructors. If it is needed to add new zombie or soldier type in the future, its name can be added to ZombieType or SoldierType classes with different specific double value as an instance. And then, when a concrete class is created for this new type of zombie or soldier, its zombieType or soldierType field(which is inside Soldier/Zombie abstract class) should be initiliazed with that specific double value. After that, of course, this new type of zombie or soldier can use any helper method defined in Soldier,Zombie or SimulationObject abstract classes without a problem. It just will need to implement abstract step() method inherited from top most ancestor SimulationObject class.

2-) If I thought Object Oriented Design Principles generally, I did not use any composition although its usage is encouraged. I think for our homework, inheritance is much more easier to handle tasks, provide reusability and create class relations. In my implementation, all my concrete entities (Bullet, Commando, Sniper, Regular Soldier, Slow Zombie, Fast Zombie and Regular Zombie) can easily use same inherited helper methods without a problem. Moreover, while creating these inheritances, I think I obeyed the Coad's Rules as much as possible. My inheritance layers are always a special kind of their parents. My classes will not transmute in the future, just new classes can be added. There was no overriding or nullifiyng in my any class. I also did not inherit from a utility class, of course. I also did not use any interface for this project. However, with abstract SimulationObject class and its abstract step() function (which is core method of our homework), I realized the advantages of interfaces also. The principle that mostly worries me about my implementation is OCP. I could not be sure whether my implementations are agree with OCP or not. I think, my common helper methods in my abstract classes (Soldier, Zombie and SimulationObject) are compatible with OCP generally. They are responsible for specifically one task mostly and I

think, they can be used with future added zombie or soldier types without a problem since they are mainly work with SimulationObject and SimulationController parameters, and because newly added types will also be a child of SimulationObject class, I think there will not be a problem. When there is a need to add new requirements to the project, new methods can be added to these abstract classses, of course. About ZombieType and SoldierType enum classes, I could not understand that adding new instances for newly added types to these classes disrupts OCP or not. Finally, I think my implementations highly agree with Liskov Substition Principle because my methods generally do not know what is the SimulationObject's real type while performing tasks. I also heavily sticked to Don't Repeat Yourself principle in my codes. I tried to not write any piece of code again somewhere in my all project.