Recommendations for Extensions to Game Engine –

Lim Shir Yin (31059546), Tan Jie Yen (31171184)

Recommendations for Extensions

In our opinion, we think that the existing game engine does not need to be changed as there are a lot of methods that we can reuse from the class we extend. For instance, the classes that extend from Actor class in the game engine have overrode the playTurn method which will be called in World class at every turn. Other than that, in order to return a collection of the Actions that the other actor can do to the current actor, the getAllowableActions method in the Actor class has also been overridden by the child classes to perform the actions.

The Location class has provided us the methods that allowed us to do modifications to the ground. To look for a list of adjacent locations of current location, we are able to call the getExits() method in the child class that extends from the Location class, instead of creating the list by ourselves which is very convenient to use. The MoveActorAction class in the game engine package has also allowed us to move to the destinations that we desired.

By applying the design principles, the classes that exist in the game engine are able to extend and allow us to reuse all the overridden methods.

Justifications on Design Principles

We have realized the importance of the design principle throughout all the assignments. First of all, we have 4 different types of dinosaurs which share some common attributes and methods such as foodLevel, waterLevel, die, layEgg and more. The behaviours such as Eat Behaviour, Drink Behaviour that will be performed by each type of dinosaur will be implemented in the Dinosaur abstract parent class instead of implementing them into the child classes which will cause repeating code. According to the **Don't Repeat Yourself principle** (DRY), a Dinosaur abstract parent class which extends from Actor class in the game engine has been created to reduce repeated code and code redundancies. Instead of using accessor and mutator methods to get attributes for each dinosaur, it has been

implemented into the abstract parent class and it allows the child classes to be inherited from it. This principle is also working efficiently for the Food abstract parent class. For instance, there are different kinds of food to be implemented and they share some common attributes and methods such as foodLevel and price.

There are different types of action for dinosaurs to perform in the game such as BreedAction, CatchFishAction, DinosaurAttackAction, DrinkWaterAction and EatManyAction. These child classes extend from the DinosaurAction abstract parent class which extends from the Action class in the game engine. It is a very bad approach to combine all the different responsibilities in one class only. This is because each action class has its own responsibility which handles only a single action and the responsibilities might be changed over the time. Therefore, they have been implemented in different child classes which extends from the Action parent class. One of the advantages of implementing this way is that if an error occurs in any of the action classes, it will be so much easier to find out what the error is. Hence, this implementation has obeyed the **Single Responsibility Principle (SRP)**.

Liskov Substitution Principle states that the objects of a parent class could be replaced with the objects of its child classes without interrupting the application. Overridden methods of a child class accept the same input parameter of the parent class. For instance, Dinosaur class which is a child class of Actor class in the game engine have overridden methods from the parent class such as name, displayChar, hitPoints and maxHitPoints. Since the Dinosaur class is one of the subtypes of the Dinosaur class, the input parameter values are able to be replaced by the Dinosaur class. The child classes inherited from the parent class have maintained a proper and correct inheritance hierarchy. This is because they do not enforce stricter rules on the overridden methods from the parent class. Hence, this implementation has been shown to obey the **Liskov Substitution Principle**.

Since we have implemented the Liskov Substitution Principle, this extends the Open-Closed Principle. This extension has allowed us to implement new functionality to each subclass. Instead of implementing “instance of” an object, a new child class has been created for the object to override the methods from the parent class. A Food class is created which extends from the Item class in the game engine. Instead of implementing “instance of” Food inside the Item class, a Food class has been created to override the methods in Item class. Besides that, we have also created a few child classes such as Fruit, MealKit, Fish implemented by

extending from Food class to avoid the use of "instance of" in the Food class. This concludes that this implementation has obeyed the **Open-Closed Principle**.

In conclusion, we have realised the importance of applying design principles in this assignment. We are able to reuse the methods that extend from the parent class and add new functionality in each subclass that we have created. We have understood more about the design principles and how they can be applied throughout the assignments we have done. We believe that the design principles are very important and useful in our future projects.