

**FIT2099 Assignment 1: Design Rationale**

**with Preliminary Design**

**Documentation**

Team:  **Tute03Team100**

Team members:

|  |  |
| --- | --- |
| **Student Name** | **Student ID** |
| Tan Ke Xin | 30149258 |
| Marcus Lim Tau Whang | 30734819 |

- *To explain the choices we made, must explain both how your proposed system will work and why you chose to do it that way*

- *preliminary design documentation: explain how you are going to add the specified new functionality to the system*

*\*Classes in the UML Class diagram without any packages are part of the edu.monash.fit2099.game package.*

To ensure our implementation works as expected, we have created a few new classes and interfaces. Having said that, a **DinosaurInterface** has been added.

This interface will then be implemented by classes such as **Stegosaur, Brachiosaur and Allosaur.** As we know, an interface works as a protocol/contract that classes implement this interface must follow all methods in the interface. Here, in our case, the dinosaurs have some additional features that a Player Actor does not have. For example, food level, hunger and breeding ability. With this, we could add additional features/attributes to only some particular kind of Actor instances (the dinosaurs), but still maintaining the shared attributes of all Actor instances (e.g.: hitPoints, displayChar). By doing so, we successfully achieved the **‘Reduce dependencies’ (ReD)** design principle. We now will have more flexibility in switching between the functionalities. Indirectly, **‘Polymorphism’** is achieved as well, since we are now able to pass different data types to the main class.

Besides that, by extending classes (parent-child relationship), eg the three dinosaur classes extending Actor class, Tree class extending Ground class etc, we can greatly reduce repetitive code for methods that are similar, in each of the child classes, and just implement them only once in the parent class. Only when needed, we could override/overload the methods in the children class by changing the method signature and body. With this, we’ve just achieved the **‘Don’t Repeat Yourself’ (DRY)** design principle.

Looking into the class diagram, new classes such as Brachiosaur, Allosaur, Bush, Fruit, VendingMachine, LaserGun, MealKit, Egg and Corpse are created. We can also see that a dependency relationship is maintained between class VendingMachine and classes LaserGun, MealKit, Egg and Fruit, since VendingMachine only needs to return new instances of these classes, and do not need to store them as attributes.

For the Corpse class, due to its portability, it will extend the PortableItem class. When a dinosaur dies, a new Corpse instance would be created according to the type of dinosaur. Thus, depending on the dinosaur’s type, each corpse will have a different displayChar and will remain in the game for different periods of time (unless picked up by Player and stored in inventory). We’ve also decided Stegosaur corpses will remain for … turns, Brachiosaur … turns, and Allosaur … turns in the game, if not picked up or eaten.

----------------------------------------------------------------------------------------------------------------

**Corpse class**

The Corpse class would extend the PortableItem class, since it is portable. When a dinosaur dies, a new Corpse instance would be created. Also depending on the type of the dinosaur, the corpse will have different displayChar, and will remain in the game for different periods of time (unless picked up by Player and stored in inventory). We’ve decided that Stegosaur corpses will remain for … turns, Brachiosaur … turns, and Allosaur … turns

haven’t finish

Fail-fast principle

don’t repeat yourself

liskov design principle

command query principal

We created:

- **Classes**: Brachiosaur, Allosaur, Bush, Fruit, VendingMachine, LaserGun, MealKit, Egg, Corpse

- **Interface**: DinosaurInterface

*\*Classes in the UML Class diagram without any packages are part of the edu.monash.fit2099.game package.*

Explanation:

**DinosaurInterface interface (implemented by class Stegosaur, Brachiosaur and Allosaur)**

We decided to create this new interface, as the dinosaurs have some additional features that a Player Actor doesn’t have, for example food level, hunger and breed ability. With this, we could add additional features/attributes to only some particular kind of Actor instances (the dinosaurs), but still maintaining the shared attributes of all Actor instances (eg hitPoints, displayChar),

**VendingMachine and some classes**

Dependency relationship among class VendingMachine and classes LaserGun, MealKit, Egg and Fruit, since VendingMachine only needs to return new instances of these classes, and do not need to store them as attributes.

**Corpse class**

The Corpse class would extend the PortableItem class, since it is portable. When a dinosaur dies, a new Corpse instance would be created. Also depending on the type of the dinosaur, the corpse will have different displayChar, and will remain in the game for different periods of time (unless picked up by Player and stored in inventory).

We’ve decided that Stegosaur corpses will remain for … turns, Brachiosaur … turns, and Allosaur … turns