

**FIT2099 Assignment 3: Extended Edition Design Rationale**

Team:  **Tute03Team100**

Team members:

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

### **Introduction**

In Assignment 3, our team has discussed and made changes according to the feedback given by the teaching team. With that, our application now is an improvised and extended version of the previous system.

### **Updates from Assignment 2**

Since we defined laser gun’s display character as ‘~’, it contradicts with the newly added, specified lake instance’s display character. Therefore, we updated the laser gun’s display character from ‘~’ to **‘!’**

Besides, Change all condition checking usage of displayChar (if any) to capabilities,

Moreover, added itemType, CorpseType enum

eatAction now only has one constructor that takes in an Item instance and a boolean (to indicate whether fed by player), to avoid confusion on which constructor to use.

Wait wait **eatAction now has TWO constructors** lol, the other constructor takes in an actor instance and boolean as parameter. Mainly to handle allosaur eating live Pterodactyls.

Kx: yea i knw but since u wrote this down before tht so i just copy pasted here hahaha

### **Extension implemented for Assignment 3**

In order to develop a more interesting game, we have implemented the extension accordingly, mostly new classes are introduced to the system.

* Quick view of all newly added objects and their display character:

|  |  |
| --- | --- |
| **Map Objects** | Pterodactyl |
| **Display Character** | / |

|  |  |
| --- | --- |
| **Portable Item** | Fish |
| **Display Character** | h |

|  |  |
| --- | --- |
| **Ground** | Lake |
| **Display Character** | ~ |

* **DinosaurMap class**

Since every 10 turns, there is a probability of 20% that the sky might rain, we decided to implement this functionality in the DinosaurMap class. We overridden the **tick()** method and provided condition checking in it. To illustrate, if probability of raining is met, boolean variable **isRained** will be updated to **True**, otherwise remain as False.

* **Fish class** *(portableItems package)*

This is a class that represents a fish instance in the lake. It currently does not have any specific implementation.

* **Lake class** *(ground package)*

To ensure that there are enough lakes on the game map, a number of lakes has been added to the game map. Having said that, each horizontally 10 square away and vertically 5 square apart, there will be a lake. Since land-based creatures are **not allowed** to enter the lake, we overridden the **canActorEnter** method and provided our implementation in it. To handle the situation where the sky rained and added water to the lake, we used an if condition to tackle this. Each lake can hold a maximum of 25 fish and there is also a probability of 60% that a new fish is borned. If probability is met, a new Fish instance will be created and added to the current lake.

* Thirsty Dinosaurs *(dinosaurs package)*

A dinosaur now has a new functionality: it will become thirsty if it’s water level dropped below 40. Therefore, we made changes accordingly in the abstract Dinosaur class and all 4 dinosaurs classes.

In the dinosaur abstract class, changes we made in the **eachTurnUpdates()** method are:

* Increase unconsciousCount if a dinosaur’s water level gets to 0 OR hitpoints get to 0 (previously we only validate hitpoints).
* Deduct water level by 1 on every turn, for all dinosaurs.

In each respective dinosaur class, we also updated their **playTurn()** method. As what we’ve been implementing, playTurn() method will determine which action to be performed by the dinosaur. The **priority of actions/behaviours** are as follows (most prioritised to least):

Lay egg → Breed → Attack (only for Allosaurs) → Drink Water → Eat → Follow (follow another nearby, same specie dinosaur to prepare for breeding) → Search for nearest lake → Search for nearest food source → Wandering around → Do nothing

Each action/behaviour has its own class that inherits the Action class/implements the Behaviour class.

If a thirsty unconscious dinosaur remains unwatered for 15 turns and no rain occurs, it will die(DieAction is returned).

From the **priority of actions/behaviours** above, we can see that **drinking water is prioritised, instead of eating**. This is because the amount of water increased when the thirsty dinosaur found a nearby lake is quite high (as compared to eating). To be more specific, Stegosaur, Allosaur and Pterodactyl will increase their water level by 30 while Brachiosaur will increase by 80. Having said that, if the dinosaur is thirsty and hungry, once it finds a nearby lake and drinks water from it, it can then happily move around the map to find a suitable food source.

* **DrinkAction class** (*actions package)*

This class is created to handle situations where a Dinosaur is thirsty and found a lake to drink water from. We overridden the **execute()** method to update the water level of the respective dinosaur accordingly. Amount of water a dinosaur can consume is different for different types of dinosaurs. For example, a Brachiosaur will increase its water level by 80 if it found a lake nearby. Whereas for the other 3 dinosaurs(Stegosaur, Allosaur & Pterodactyl), it will increment their water level by 30 only. This action will be called in each dinosaur’s playTurn() method if the dinosaur’s water level is lower than its bare minimum and it finds a lake nearby.

* **SearchNearestLakeBehaviour class** *(game package)*

This behaviour class is a class that finds the nearest lake and moves the dinosaur actor one step closer to that particular lake. The getAction() method would return a MoveActorAction if there is a lake nearby, otherwise it would return null. This getAction method is also called from each of the 4 dinosaur classes (Stegosaur, Brachiosaur, Allosaur,Pterodactyl) in their playTurn() method.

* **Application driver class** (*game package)*

In order to fulfill the new requirements: a second map & a more sophisticated game driver, we made changes to the **Application** class. Now, the player has the ability to quit or choose which game mode(either Challenge or Sandbox) they want to play. This is done by using a switch case statement. Besides, we also added a new map that has the same size as the existing map in this class.

* **Player class** (*game package)*

If the player is at the most north of the first map(y=0), then he/she has the option to go to the south edge of the new map. This is done by **adding a new actions** to the player, leading it to display one more new option for the player to choose in the console.

* **Pterodactyl class** *(dinosaurs package*)

### **Engine Recommendations**

Firstly, we concluded that the introduction of abstract classes are very useful. For example, in the game engine, classes such as Actor,Item,Ground,WeaponItem are declared as abstract classes. With that, each implementation of the child-classes are hidden and only functionality of the abstract class is shown/known. To illustrate, we can easily extend the Actor class and create our own Dinosaur class and provide our own implementations by overriding the necessary methods. Since we can easily extend child classes, code reusability is promoted, the chances of creating repetitive methods that have the same functionality is greatly reduced, leading us to achieve the D.R.Y principle. Besides, the game is now extendable and maintainable is an easier manner.

However, we think there are also a few improvements that can be done in the game engine. In the **Display class**, instead of having only a **readChar()** method that returns the first character of the entered string, we should provide more flexibility. To illustrate, the Display class should also have a method that takes in numbers (eg: integers). Now, the system can read in both ‘char’ and numbers and do the necessary processing afterwards. An example of proposed usage can refer to Application class’s runGame method.

Moreover, we think the idea of introducing the **hotkey() method**(in the Action class) can be improved. Now, the system might encounter a situation where there are no free characters available to use anymore. Consequently,the game cannot be continued and affects the gaming experience. We are also aware that we should avoid using any magic numbers as well. This is discouraged as it might disrupt other programmers from understanding which literal numbers stand for. Not only that, it will be more problematic and harder to maintain if the magic number appears more than once in the system. Since we are the developers of this game, we can indeed define all the possible actions with a named constant.Having said that, we can now maintain the system in a better way. By doing so, we can also ensure that the system will be more user-friendly; ensuring all programmers who read the code will be able to identify quickly/have a better understanding of which constant represents which actions.

Duplicated methods in ActorLocation & GameMap (eg: isActorAt) （kx: im not sure but i think we never use ActorLocation there de? We always refers to the gameMap de isActorAt or blabla)

* what problems your proposed extra features would solve
* the design of your proposed extension
* the advantages, and any disadvantages, of the proposed change
* the new functionality, if any, that would be available to designers of new game clients