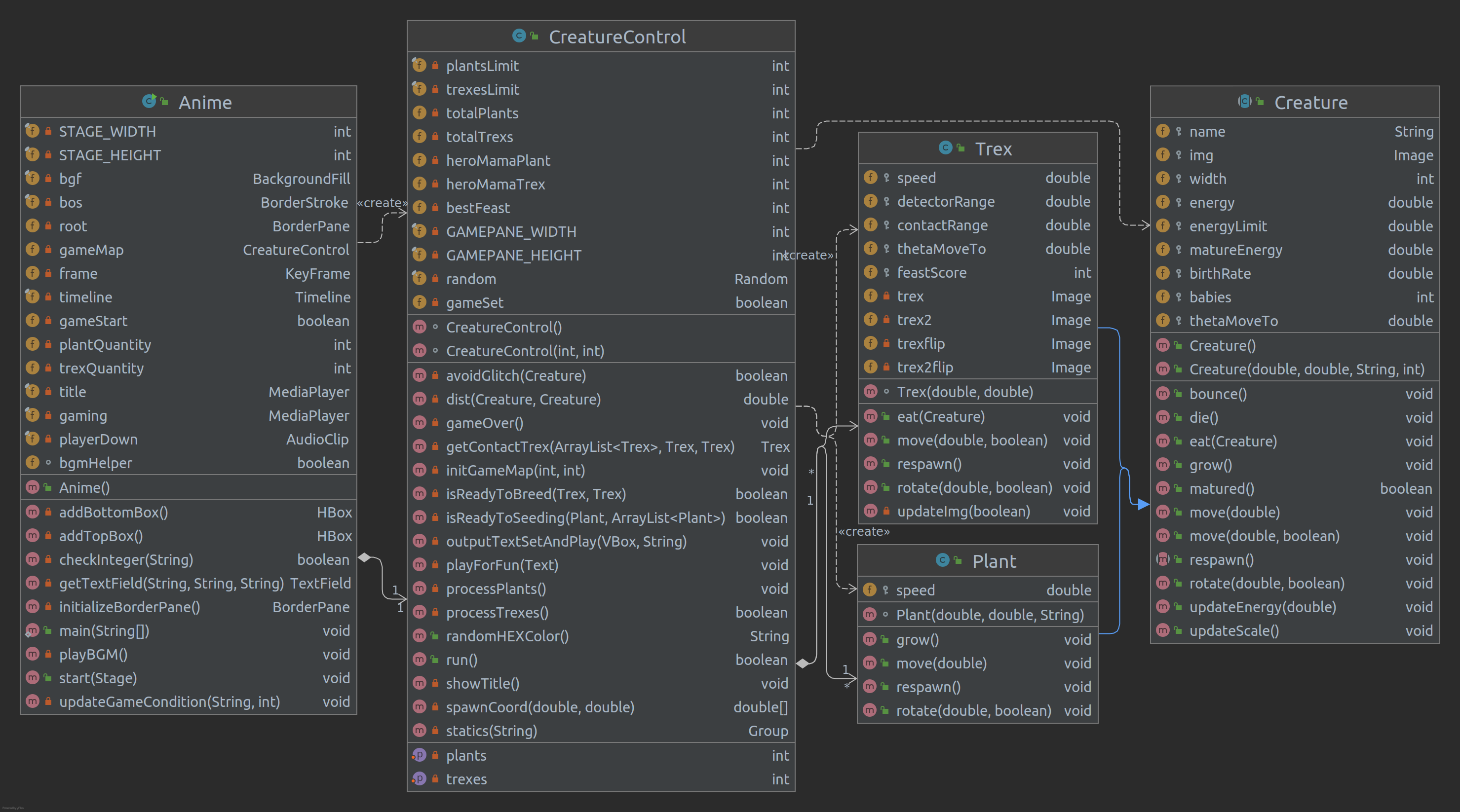
|  |  |  |
| --- | --- | --- |
|  | Looking for | comments |
| Implemented Features | **Number game:**  þ User can make guesses  þ Feedback to user (high/low)  þ Feedback to user (success)  þ Used UI elements:  þ Used a Layout  **Bug World:**  þ Bugs  þ Plants  þ Animated  þ The bugs have different behaviours  þ Images instead of circles  þ Control the animation  þ Other | Input is limited to Integer (0-100) by checking method.  BorderPane, ImageView, TextField,  Button,  1 type,  1 type,  6 different images  Start Stop Pause  Slider control of music and animation  Described in design part below |
| Quality of code | **OOP:**  þ What is the abstraction of your code  þ How have you used inheritance and/or composition  þ Encapsulation of the code  **Testing:**  **When marking we will consider these quality aspects of the code. You do not need to explain this but be aware of it.**  - Indentation is correct  - variables have understandable names  - variable are named consistently  - Avoiding repetition of code  - Used appropriate comments | How did you plan and design the programs?  Described in design part below  What were your considerations about encapsulation?  Described in design part below  Write a short comment about how you tested the code. (test method, test cases, boundary cases etc.)  Described in testing part below |
| Understanding of issues and challenges | We are not expecting comments for all of these questions. Use approximately 300 words and focus on where you have the most important things to comment on.   * How did you overcome issues? * Are there still known issues? Describe them and explain how they can be overcome if you had more time. * Are there aspects what you have done that you are particular proud of? * Are there design aspects you would like to change now that you have implemented it? |  |

Would you please watch my video demonstration: [Link to Youtube](https://youtu.be/BS9ejetgwts). Thank you for watching!

**Design description:**

* **Class Diagram:**



* **Program Structure:**
* <Anime>, User Interface, the main class:

Only UI elements are placed in this class.

Encapsulated from back end, designed only two portals:

1. the Constructor,
2. run() method in <CreatureControl> class, acting as a Boolean to control the animation.

One Main BorderPane,

Top, menu bar: assembled by addTopBox()

* 3 Buttons: play(play new round from title screen, resume pause), pause, reset(reset to title at any status). Play button is also a **portal** to start the controller and read the feedback of its status.
* 1 Label: shows status information
* 1 Slider: speed control of gaming BGM and animation

Bottom, user input area: assembled by addBottomBox()

* 2 TextFields: each hold by one HBox, showing prompt text, used to define the initial creature number for next round, all input will be automatically **restricted** to Integer only (0 - 30) by designed text field listener.

Center, **portal** to constructor of <CreatureControl>

Other features of this UI:

3 BGM for **different status**: Title music, Gaming music, Game ending sound clip. All 3 music precisely controlled by the logic inside playBGM().

Gif background of a turning island.

Gaming related UI information will be generated in next controller class.

* <CreatureControl>, Controller class:

The Class **extends from Pane**, contains creatures and their behaviors controller.

The Constructor reads input data about the creature quantity from the UI and setup 1 list of Plant and 1 list of Trex.

run() method starts the processing of all creatures and return as a **Boolean** about the gaming status.

2 Creature class, Plant and Trex have own controller following Rules.

<Plant>: Death, Breed, Grow, Move randomly

<Trex>: Death, Eat, Breed, Tracing, Move randomly

Generate animated **color and shale shifting** Title and Game over statics by information updated during the gaming process.

* <Creature>, super class of all creatures, **extends from ImageView**:

Shared attributes and behavior control methods for all creature

* <Plant>, sub class of <Creature>:

It has 4 types of plant images.

Plant can grow up (**scale up at the same time**), breed randomly close by itself when it is matured and passed birth control, or move randomly.

* <Trex>, sub class of <Creature>:

It has 4 types of images, 2 are mirror flip of the other 2, used for **animated effect** and **rotate** to moving direction.

Trex can **trace** the plant within a detection range, eat them, to grow up (**scale up at the same time**). When it meets other Trex, and if specific conditions are fulfilled (both matured and pass the eco-balancing birth control), breed the baby Trex. Or move randomly. Some mechanisms are designed to **prevent dumb** Trexes tracing one same plant to death. When the health energy is goes lower, the Trex can **shrink and fade out** by changing scale and opacity.

* **Conclusions:**

At beginning of the week, in my computer there is not a runnable BugWorld program at all. After few tries, I found it was extremely difficult to implement any UI before there is a logic ready program, so I decided to start all over again from sketch. The biggest difficulty I met is how to bend my mind to believe that the <Creature> class can practically be an <ImageView>, instead of emerging an <Image> or <ImageView> into the field of the <Creature>.

Rest part is the bricking up the construction of the game world along with the coding, some of them I am so happy that I learned, and had successfully implemented, including some testing:

1. Finish a tracing function. When testing the implementation, I found that multiple Trex would trace a same target, so some mechanism was patched to the tracing function to prevent them chasing a same target for too long before its health is in danger.
2. Understand the running timeline can be controlled by the EventHandler inside the frame. Learned how to animate by KeyValues.
3. Rotate the animated characters to the direction they are marching to, flip them when turning left / right.
4. Music player done, and the 3 tracks can be played or stopped according to the game status. Debugged for a while, until I write down all the logic flow 1 by 1, then cleaned up the codes and finished a fix.
5. Watched many videos teaching JavaFX, don’t have enough time for all, I still wish to know if there is a most efficient approach to pick up a new tool.
6. I removed Stop button from the first design and replaced it with a Reset button. It can be more reasonable to have a reset function rather than stop.
7. Built TextFieldfor user to input data for game constructor which is set with a Listener of value change. I learned an interesting way of using try catch to limit the input of user. try *{* return Integer.*parseInt(*str*)* <= 30; } catch *(*NumberFormatException e*) {* return false;}
8. How to tune the attributes of the creatures and make the Eco-system more sustainable.

* **Issues still exist:**

1. I tried to avoid objects move over right and bottom boundary, but every approach I made would glitch the game. If there is enough time, I will learn more about setX(), setTransferX() and setLayoutX(), maybe manipulate them using any simple Pane or Box from the most beginning.
2. Also I would like to make buttons much fancier with CSS.

* **Questions:**

1. How to use the MVC design pattern, and what is the right way to design a good program?
2. Is this encapsulated enough by my approach, leave only the constructor public and all other method in <CreatureControl> private to protect the data.
3. In all other classes those link to <CreatureControl> (exclude the main <Anime> class), I made some of the fields and methods protected or public. Is this safe? Because in my opinion, users can not touch from main class?