Group Project: Pet Simulator

Aoutul Nabi Purna#1, Shishir Dhar#2, Thauhid Ferdhous#3, Md. Rahat Sarkar#4

#*Department of Computer Science and Engineering, North South University  
Plot#15, Block-B, Bashundhara R/A, Baridhara, Dhaka-1229, Bangladesh*

1aoutul.purna.241@northsouth.edu

2shishir.dhar.241@northsouth.edu

3thauhid.ferdhous.241@northsouth.edu

4rahat.sarkar.241@northsouth.edu

***Abstract*— This document contains information regarding the group project assigned to the members of Group 2. The paper talks about the work the group members have done and the procedure that has been followed. An overall understanding has been provided for the different methods that have been implemented.**

1. Introduction

This document is a report on our group project. An electronic copy can be downloaded from our github page[1]. Any questions or inquiries can be through our email or through our social media handles. Our source code is also available via the same github page[1].

1. Our Approach

After the project was assigned, we began to think about what the project as a whole would look like. Multi-threading and file input-output has been implemented in the project. We have separated individual project elements into different classes and have emphasized on the usage of methods and built-in java functionalities. We have also tried to turn commonly used parts of our codes into methods. Since the project is on pets, we have an Animal class with the required fields and methods. We can use inheritance to create various forms of pets to add diversity. The custom methods and the GUI have been assigned individual classes to organize the code and make it more functional. A separate Settings class has been created to store the default values for settings. Multithreading has been used to concurrently run GUI elements alongside pet behaviour.

1. *Coding*

We have currently implemented our idea of the basic multithreading and file I/O.We have created a parent class for the various pets. A way to show animation via the generation of frames has been implemented. A working main menu and a skeleton Game GUI with multiple layers have been implemented so that it can hold other sprites such as pet toys.

1. *Others*

There were some other miscellaneous tasks that needed to be completed/sorted, such as, easy ways to share our code, collaborate on topics and think of further improvements that can be done. We’ve regularly used github[2], cl1p[3] and messenger[4] to share code and communicate whenever it was necessary.

1. Interesting Parts of our Code
2. *Animate()*

A form of animation has been created at a relatively constant FPS. The method takes in current system time and uses 30 FPS by default but it can be changed in the future using the Settings Menu. Our method involves an infinite loop checking whether the difference between current time and last time the frame was updated exceeds the amount of time each frame should be visible for under the current FPS, updating to a new frame once it does exceed the value. A progress bar updates as well using this method. The bar shows happiness and hunger states.

1. *CustomMethods*

We’ve created many customized wrapper methods of some common Java methods here to abstract away the parts we think are going to be used a lot in a specific way in our project. We basically simplified things and hid them.

1. *Generics*

To prevent constant method overloading, we are implementing generics so that any input can be used as different method parameters.

1. *Try and Catch*

We have used try and catch to prevent any unforeseen errors from crashing the program. We have implemented the try catch in file handling since file access and read permissions vary across devices. Try catch allows us to keep the code running in places where otherwise it would have crashed the program.

1. *Menus*

There are different menus for different purposes and a GUIHandler class that allows a smooth transition between the menus.

1. *Main Menu*

This menu currently has buttons that initiate New Game, Load Game (from SaveFile), and Settings.

1. *JLayeredPane*

Since in Java, multiple elements added to the frame are displayed according to when they were added, we are using JLayeredPane for using multiple layers for the various different sprites so that the UI remains uniform and it is easier for us to modify without fearing that some parts may break.

1. *Resources*

Currently we are using AI-generated images as well as public domain artwork for our resources. So far, the AI-generated images are being used as the main backdrop for the pets in the Game UI.

1. *CustomMethods*

Using the java.io package’s ObjectInputStream and ObjectOutputStream classes we save and load from and to the current instance of the gameData object.

1. *Multi-threading*

Since our program involves two infinite loops under normal circumstances, we could not have created a functioning program with such ease as it would be stuck on the first infinite loop or we’d have to go through a high magnitude of complex logic and code in a single loop. Multi-threading allows us to operate two infinite loops simultaneously and more. This makes it easier to break down our code into smaller and more manageable modules, free of convoluted logic focused on a single part of the program.

1. *Save and Load*

Saving information regarding current play helps load the data and continue from where the player left off.

1. *Percentage Based Structuring*

The GUI elements have been placed using percentage of the screen height and width.

1. *Right Indexed Structure*

The GUI elements have been placed from the right side of the screen using screensize-componentsize-right, where one can put 0 in right to hug the screen’s right.

1. *Bottom Indexed Structure*

The GUI elements have been placed from the bottom side of the screen using screensize-componentsize-bottom, where one can put 0 in bottom to hug the screen’s bottom.

1. *Random Sickness*

A tiny chance of the pet getting sick every time the pet ages. The player has to solve a trivia question to continue playing.

1. Problems And Solutions

We sometimes ran into the issue where multiple threads used the same variable hence the changes were not reflected on other threads. We fixed it by using the volatile modifier.

Having an infinite loops nearby frame.setVisible(true) causes a bug where the frame is not display. We changed our approach to the infinite loops by giving them their own separate thread.

1. Future Plans

Our future goals are:

* Many more animal subclasses
* Keyboard inputs
* Pet animations
* Audio support
* In-game time with regards to system time.
* Ability to own multiple animals

References

1. https://github.com/micyclebobert/Pet\_Simulator
2. https://github.com/
3. https://cl1p.net/
4. https://www.messenger.com/
5. Template used for writing this report: https://docs.google.com/document/d/0B-9sQq9x70jLOU95VE1XaVRUZTA/edit?resourcekey=0-gQLvMUP3P-6I5gJO-QDYcQ