Project 2: Data Structure with User Defined Data

Andrew G., Cody N., Caleb R.

Professor Mehri

December 14, 2019

Capital Technology University

CS-200-D01

Abstract

Introduction

Procedure

1. To start, we had a meeting to discuss the topic of the project. We decided to do a zoo app as the banking app has a lot of messy code and just a lot of code in general. It was general consensus that it would be much more efficient and faster to make a new program that we can set the requirements for. From there, we discussed the features of the program and came up with a feeding schedule for animals as well as providing information on the animals. To add even more functionality, we decided to add in an option to add an animal.
2. Next was the AGILE planning. To start with, we came up with a list of requirements we would have to complete in order to complete this project.

Creating Initial Menu

Create Classes

Read Files

Write Files

Adding new Animals

Feeding Schedule

The feeding schedule covers the data structure requirement. Reading and writing to the files covers file i/o and polymorphism as we are using a pointer with a virtual print function to display file contents. There will be three classes with Animal being the parent and Avian and Reptile being children.

1. The first UML diagram we created to plan the project details was the Class Diagram (Fig. X). The class diagram makes programing the class files themselves significantly easier in addition to helping us better visualize the final project’s structure. each class has unique attributes. Animal has food type, gender, weight, and name. Reptile has number of legs and venomous. Avian has wingspan and beak size. All attributes have a get and set function. Animal has a virtual print function and the other two have regular print functions. The process of the code starts with a menu with three options. 1. Animal Info, 2. Add an animal, and 3. Feeding Schedule. When 1 is selected it will print out the information from a file for an animal that was asked for. It will have an option to add it to the feeding schedule afterwards. The feeding schedule prints the top of a queue that has animals in it. 3 will use class functions to make a new file for the new animal. This is shown in the Use Case Diagram and Sequence Diagram (Fig. X and X)
2. AGILE estimation is a must once the design is complete but before any actual programming begins. After determining the tasks needed, we estimated their story points and value points so that we can calculate the BFTB points (Fig. X) which will help us determine what tasks we should prioritize. The story points will also help when planning sprints since they will act as a guideline for how long each task will take.
3. Once the planning was complete, we began working on the code itself. We split off and began working based on what we had determined through AGILE estimation to be the best places to start. We began with the class files since, despite their low BFTB value, they are required to effectively test the rest of the program; as well as the basic structure of the main method.
4. To start off in the main(), an if loop was created to hold the three main options that were mentioned previously. There is a 4th option to exit the program. To avoid the possibility of having declaration problems, we used if else loops for all the menus. The queue of Sched is made right before the menu is posted. It reads from a Schedule.txt file and adds them into a queue. It will close the file right after. Option 3 to print the feeding schedule was done at the same time as this. The feeding schedule is inside of a loop that checks if the queue is empty or not. If it is not, then it will read the front and ask if the animal was fed. If no, it will tell you to go feed it and will repeat the question. If yes, it will pop it and read the next front element. Once it is empty, it will clear the feeding schedule txt. This can be seen in Figure X.
5. The process of selecting an animal to perform actions on requires the user to input the name of the animal they desire. From there the program looks for a file that matches that name and reads the data to recreate the Animal object. Since the program won’t know what type of Animal the user selected until runtime we used an Animal pointer that points to a reference of whatever child class the user’s animal is. After the file is parsed and the child reference is created and pointed to by the Animal pointer the user is able to view the Animal’s print function and perform actions such as add it to the feeding queue or change its weight. If the user changes the Animal’s weight the Animal’s file is overwritten so that the change is persistent.
6. To keep the code organized the process of making a new animal was divided into separate methods (Fig X). The methods return either an Avian or Reptile, based on the user input, which is set to an Animal pointer, so the Animal’s print just has to be output to a file for the Animal to become persistent.

Analysis

Conclusion

Figures

A screenshot of a cell phone

Description automatically generated

*Figure X*. UML Class Diagram

A screenshot of a cell phone

Description automatically generated

*Figure X*. Agile Estimation

A screenshot of a social media post

Description automatically generated

*Figure X*. Method for Adding New Reptile

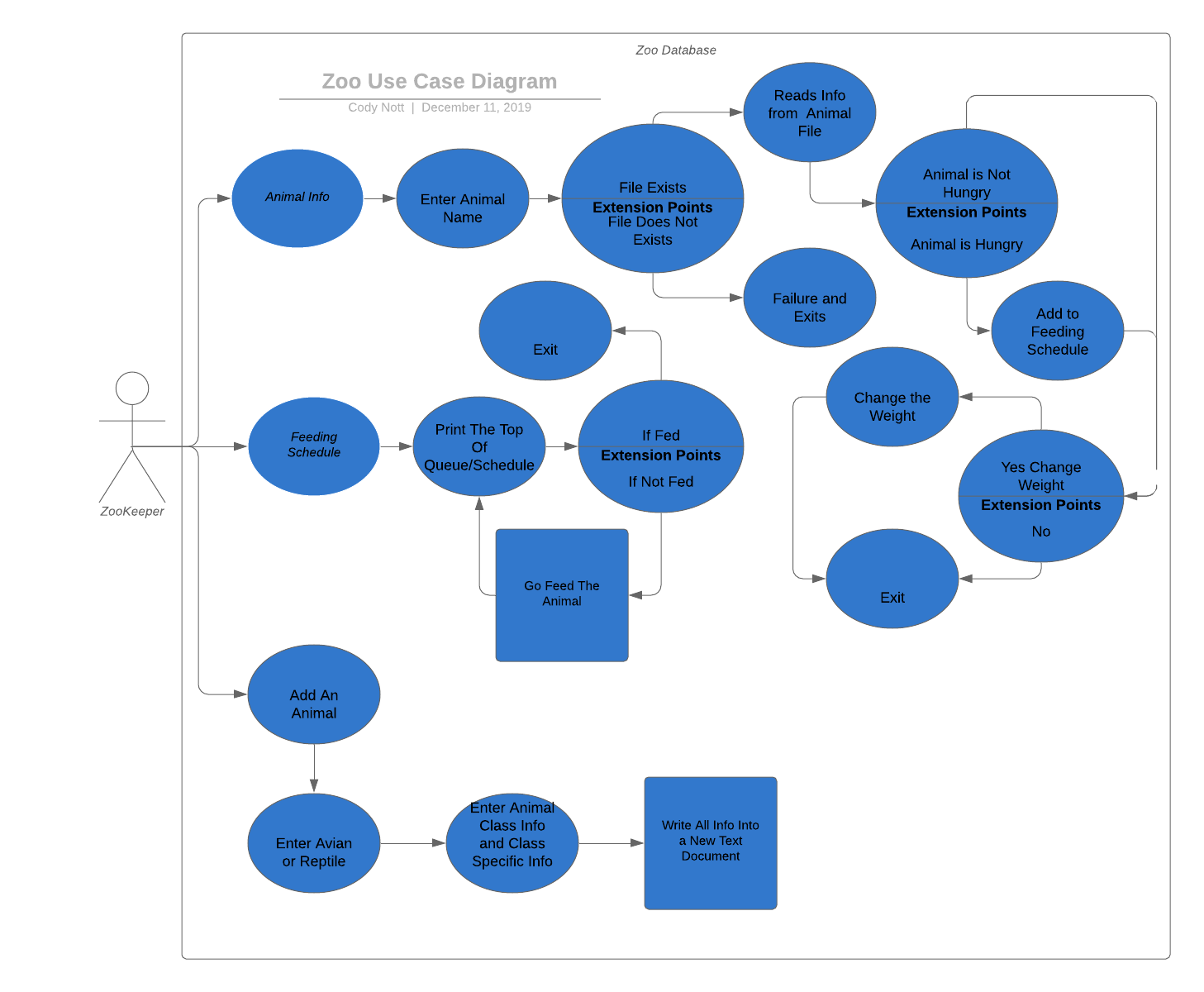


Figure X. Use Case Diagram

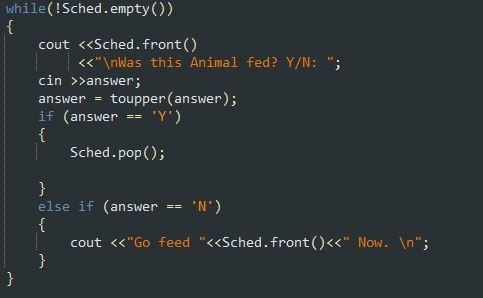


Figure X. Queue Code

References