**Derek Yang**

**Document (Design + Reflection)**

Program Design

Problem: Create a zoo tycoon game where there are three types of animals. You must use inheritance with these animal types. The following are necessary conditions of the game:

* Create three animals tiger, turtle, and penguin which inherit from the animal class
* Create three arrays and make sure to deallocate them
* If an animal is above three years of age, they are an adult and can have babies
* There are different costs and requirements for each animal listed
* User will start with 100000 in the bank
* A random event will take place everyday

The program will loop everyday and ask the user if they wish to continue playing or if they lost the game.

Animal Class – Parent

* This will be the main parent class for tiger, turtle, and penguin.
* Virtual functions: set information
* Print what type of class

Tiger Class – Child

* This will inhert from the main class and use the virtual function setinfo to set current information about the animal

Turtle Class – Child

* Same conditions as tiger class

Penguin Class – Child

* Same conditions as tiger class

Zoo Class

* This will hold the array of three animal types
* For each function, I will need to provide a method for all three animal types
* I will need helper classes to: access the arrays, delete the arrays, resize the arrays by 2, create new baby animals, kill animals, get total payoff, get total feed cost, get total profit cost, and get total tiger bonus.
* I will need member variables to keep track of: total payoff, total bonus, total feed cost, total profit, total animals, total baby sizes, keep track of current array position, keep track of if number of babies exceeds current array size, and if animals have been killed.

Game Class

* This will play the game
* Loop through each day in the following order
  + Increase all animals age by 1
  + Pay the feeding cost of each animal
  + Random event:
    - Sickness occurs and animal dies
    - Bonus for tigers
    - Baby animal is born
    - Nothing happens
  + Calculate the profit and payoff for each animal and add it to the bank
  + Ask the user if they would like to buy an adult animal (3 days old)
  + Prompt user if they would like to keep playing the game
    - End game if user wants to quit

Main

* This should only have two or three functions to play the game

Input Validation

* The user will be asked mainly integers in the menu options. The cases are usually within a range of answers. Therefore, I will create a function that checks for an int, and then if that int is within a specified range.
  + Ex: choose 1 to 3 will ask for an int within range 1 to 3
  + Ex2: choose starting grid width will be min 1 and max int

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Input Values** | **Functions** | **Expected Outcomes** | **Observed Outcomes** |
| Input too low | Input is less than the min | Main()  Validation()  If input < min | Loop back and ask the user to re-input | Continually asks user to enter a higher int |
| Input too high | Input is more than the max | Main()  Validation()  If input > max | Loop back and ask the user to re-input | Continually asks user to enter a lower int |
| Input in correct range | Input is within max and min | Main()  Validation()  If min>input>max | Accepts the correct answer and exits the while loop | Loop stops and accepts correct int |
| Input is not an int | Input is a string, bool, float or other variable | Main()  Validation()  If input != int | Loop back and ask the user to re-input an int | Loops until an int is entered within the designated range. |
| User chooses adult tiger, turtle, or penguin | 1 for tiger | Main(0  Validation()  Tiger()  checkArrSize()  bank() | The user will buy an adult tiger and money will be subtracted from the bank | The array will need to be re-sized if it reaches max. 1st time gave an error . Second time resized array and error was gone. |
| User has babies for random event | No input random event | Main()  makeBabies() | Babies will appear in the zoo | Need to check for if type can have babies, then create the babies if so. Will loop through all types to find an adult. Increase the array size if needed and adds babies to the zoo |
| Bonus for tigers | No input, random event | Main()  Tigers()  increaseBonus() | Tigers will receive a random bonus per tiger | Needed to bypass dead tigers. Used a dead function to identify these tigers and only create bonuses for those who are alive. |
| Animal dies | No input, random event | Main()  randomDead() | Random animal dies. | Looped through until random animal died. IF the animal type chosen did not die, the function will keep looping. Needed to split function into 2. |
| User stops playing | 2 to stop playing | Main()  Game()  Validation() | The user enters 2 to stop playing | The bool flag triggered and user was escaped from the loop of the game. |

Reflection

Wow, this was a hard one. I learned a lot from this project regarding initializing objects in arrays. I began researching initializing objects in vectors but quickly realized that the instructions said specifically not to. I understand the importance of using dynamic arrays for objects because then we can re-size the array if needed. Most of the time this can be seen in GUI applications where we really do not know the user’s input. I found that the hardest part for me in this project was figuring out all the conditions. There were so many conditions to the project like looping through until a particular animal was killed. On the surface this sounds easy, however I quickly realized that I needed to write a bunch of helper functions to do this process. I had to create a condition where I could not use previous random outputs because of this. Then, I had to check if certain conditions were met, such as if the animal died or if an animal could bear children. Putting multiple conditions on functions clearly made me appreciate the use of helper functions.

Having a proper naming convention for getters, setters, destructors, and constructors was essential in keeping my plan on track. My original plan was way off compared to what I ended up with. I had to put a bunch of tracker variables or flag variables to keep track of things. I felt like this project really taught me how to think within nested functions and the nitty gritty of programming. I had to write many things down to remember what I did. Using comments greatly helped me to create and utilize functions. I always wrote a summary of the function so that way I am able to review later on quickly if I need to utilize it. However, this came at the cost of me creating a bunch of functions, some of which I only used in testing. I left in a lot of unused functions because I would be able to reference them later on for my own knowledge. However this also leads to sloppy code when other people are reviewing. I found that I need to create an order to how I declare my functions for use.

This project really made me see how meticulous one really needs to be in the planning phase. Also, how important comments are. I couldn’t imagine if someone wrote a zoo tycoon program without comments and another programmer had to read it. It would take so much longer to understand what was going on in the document. I found that because I had so many files to go through, I really needed to summarize and comment. This also helped me to keep track of what I had done. Completing one part and coming back to it the next day really slowed me down. However, time was a crunch and is a crunch for us students. Overall I really enjoyed this project even though it gave me a run for my money.