Philip Warton

CS 162

February 9, 2020

Assignment 3 Design Document

Understanding the Problem:

Summary. We are tasked with creating a game where you play as a zoo manager, purchasing animals and profiting off of their exhibitions. The game goes month by month, with each month having to pay certain fees, acquire certain revenues, and deal with other events. If the player runs out of both money and animals, then the game is over. We are asked to use object-oriented design practices such as inheritance, operator overloading, and the big three as we write this program, with a great focus on inheritance specifically. There are three different animal types, and we must dynamically allocate an array that can contain each type.

Assumptions:

- I am assuming that players are not allowed to let animals die if they have enough money to pay for treatment
- I am assuming that the player must feed all animals every month, without having the option to let them starve
- I am assuming that animals can only give birth to babies of that animal type

Design:

(see next page)

Design

Animal
~~Zoo~~ Class $\longrightarrow$ Child Classes

- Tiger
- Sea Otter
- Bear

protected:
- birth rate
- age
- cost
- monthly food cost
- monthly revenue
- string animal type

public:
- constructor
- destructor
- getters
- setters

- print animal
- is_baby
- is_adult
- iterate month

Zoo Class

private:
- Animal ** array
- array length
- month
- special event
- money

public:
- iterate month
- special event
- sickneess
- babies
- purchase animal
- pay feeding

Program Flow

```
Program Starts
       ↓
  Zoo object created
       ↓
    player must buy at least
    one animal
          ↓
      while (money > 0) {
              iterate month {
                     special event
                     buy animal
                     etc..
              }
      }   ↓
      array of animal pointers
      memory freed
         ↓
    zoo object destroyed
        ↓
    program ends.
```

Testing:

Function	*Case*	*Case Type*	*Outcome*
Animal::is_baby()	Age = 0	Good	Return true
Animal::is_baby()	Age = 6	Edge	Return false
Animal::is_baby()	Age = 48	Bad	Return false
Zoo::purchase_animal(std::string “Tiger”)	Money > Tiger.cost	Good	Add Tiger to animal array. Decrease money by Tiger.cost
Zoo::purchase_animal(std::string “Tiger”)	Money = Tiger.cost	Edge	Add Tiger to animal array. Decrease money to 0.
Zoo::purchase_animal(std::string “Tiger”)	Money < Tiger.cost	Bad	Cout << “You do not have enough money to purchase this animal.” << endl;
Zoo::iterate_month()	Player has no money and an animal gets sick	Bad	Animal dies and zoo.money does not change
Zoo::iterate_month()	Player has enough money when animal gets sick	Good	Money is taken away, and animal is alive
Zoo::iterate_month()	Animal is 6 months old and money = 2 * cost	Edge	Money is taken away and animal is alive