Project 2 Oliver Solorzano

"Zoo Tycoon" CS 162

Design

1. Game
    a. Have user start with 100,000 dollars
        i. Needs to buy three types of animals
            1. Each type should have a quantity choice of either 1 or 2
        ii. Newly bought animals are set to be 1 day old (initialize in constructor)
    b. Each turn is 1 day
        i. At beginning, increment animal age by 1 day (variables that hold animal age)
        ii. Pay feeding cost of each animal (cost function)
            1. Have user choose from different feed types
                a. Cheap
                    i. Half as expensive
                    ii. Sickness 2x more likely
                        1. Create rand bias
                b. Generic
                    i. Normal
                c. Premium
                    i. Twice as expensive
                    ii. Sickness 0.5x less likely
                        1. Create rand bias
            2. Subtract from total
            3. If animal is not fed it dies
        iii. Random event takes place
            1. Sickness occurs
                a. Random animal dies
                    i. Remove one animal of type from dynamic array holding animals
            2. Attendance boom
                a. Generate random bonus between 250 and 500 dollars for each tiger in the zoo for that day (rand function with range 250-500)
                    i. Add to total payoff of day as a reward
            3. Baby animal is born
                a. Random animal has baby (choose rand from array)
                    i. If animal chosen is older than 3, add babies depending on type of animal
                    ii. If animal chosen is not older, choose again (loop)
                        1. Break loop if no animals are older than 3
                    iii. Baby animal starts at age 0
            4. Nothing happens
            5. Have output message that lets user know what occurred
    c. End of day
        i. Calculate profit based on number of each animal and specific payoff
            1. Add bonus if applicable

ii. Ask user if they would like to buy an adult animal
    1. If yes, subtract cost from total
    2. Adult starts at 3 days old
    3. Add animal to array
        a. Increment animal total variable
iii. Ask user if they would like to keep playing
    1. If user has no money (total <= 0) or selects no, tell user, end game

2. Zoo Class
    a. Has access to animal classes
    b. Holds dynamic array that manages animal count
        i. Holds ten initially
            1. Create capacity variable
            2. Create current animal count variable
        ii. Array resizes by doubling capacity if full (check if animal count == capacity)
            1. Function takes current array and capacity(size) variable
            2. Dynamically allocates new array
            3. Copies current animal objects to new array
            4. Deletes old array
            5. Returns new array
    c. Functions that calculate total and subtract based on specific animal classes and costs
3. Animal Class (specific animals are inherited from this class)
    a. Age
        i. Adult >= 3 days
        ii. Baby < 3 days
    b. Cost
        i. Tiger cost $10,000
        ii. Penguin cost $1000
        iii. Turtle cost $100
    c. Number of babies
        i. Tigers have 1 baby
        ii. Penguins have 5 babies
        iii. Turtles have 10 babies
    d. Food Cost
        i. Set constant $10
            1. Tigers are 5x
            2. Penguins are 1x
            3. Turtles are 0.5x
    e. Payoff
        i. Tigers are 20% of cost per tiger
        ii. Penguins are 10% of cost per penguin
        iii. Turtles are 5% of cost per turtle
    f. Create User animal
        i. Variables

1. Name
2. Cost
3. Number of babies
4. Food cost
5. Payoff

ii. Have user input values for each variable
1. Initialize with constructor

Test Table

Test Case	Input	Expected	Observed
User enters valid option for initial quantity of animals	Enters int value within range (1-2)	Option selected successfully, progresses to next input(creates those objects and stores them in animal array)	Option selected successfully, progresses to next input(creates those objects and stores them in animal array)
User enters invalid quantity option	Enters value != int, or not within range	Tells user entry is invalid, tries again	Tells user entry is invalid, tries again
Cost is accurately subtracted each day	End of day, check to see if cost is removed based on current events and animals	Cost is accurately subtracted	Cost is accurately subtracted
Payoff is accurately added each day	End of day, check to see if payoff is added based on current events and animals	Payoff is accurately added	Payoff is accurately added
Random bonus is accurately added for each tiger	Count the number of tigers and multiply by random bonus	Bonus payoff is accurately calculated and added for each tiger on top of regular payoff	Bonus payoff is accurately calculated and added for each tiger on top of regular payoff
Random event prints to user	Check to see if random events matches message output	Message output and random event are the same	Message output and random event are the same
User enters invalid traits for custom animal	Enters value != int or string,	Tells user entry is invalid, tries again	Tells user entry is invalid, tries again

Reflection

My original design for the Zoo Tycoon program was overly ambitious to say the least. I originally wanted to implement the extra credit options, but found that I was running out of time to program them. I began by programming the Animal class. I created the member variables and made them public for the subclasses to inherit them. I proceeded to create the tiger, penguin, and turtle subclasses. I had them call the base constructor and pass their specific values to it when creating instances of the subclasses. The Zoo class was the most complex of this program. It managed all aspects of the game and manipulated the animal subclasses. As I was programming this class I found that my original design was missing many crucial points and details.

The biggest issue I faced was dynamic allocation of the animal objects. I had a hard time conceptualizing how I would implement this in the program. I decided to create a pointer to a pointer array containing the addresses of the animal objects. Whenever I would create an a new instance of the animal class I would store the address of that new variable in the a pointer. I would then access it by dereferencing the pointer pointing to the address of that object.

Memory leaks were a huge issue also. I went through multiple design functions for resizing the animal objects array. I decided to pass the old array into the function and then create a new temporary array with an increased capacity. I would then transfer the old values to the new array and then delete the old array. I then changed the address of the old array to point to the new address where the newly allocated array was held. I found that the memory leaked whenever I would increase the array size. I found that the issue was the way the parameter was passed into the function. I wasn't passing the array by reference so it was creating new arrays, but not deleting the originals.