CS 162 - Lect 1 - Rec 16
February 8, 2019
Nick Olson

Assignment 3, Zoo Tycoon Game

Understanding the Problem:

Of the things that are not explicitly stated, I'm going to assume that the user can choose whether or not to buy the medication for their animals, but not whether or not to feed their animals, that each element of the zoo that makes money or acts as an expense should be listed so the user can see exactly what decisions they've made that are leading them ever on toward ultimate failure. I'm also going to assume that all that is needed is some basic outputs on what's going on and how much money they have left in the console, but because I'm awful and apparently hate myself, I'm going to work way too hard to build a system like curses and jpanels from scratch. Users will only give input on what kind of food they want to buy, whether they'll spend on medication, hitting enter to advance menus, etc. To implement good food that decreases chance of sickness, I think I'll randomize uniformly and give them one second chance if they're chosen, so I don't need to mess with an array with multiple instances of some but not others that I'd have to refresh every once in a while.

Program Structure:

The zoo_tycoon.cpp file will create a Zoo object and hold the main game loop that runs the day method until game over has been reached, but all logic for the game will be found in the Zoo class. The Zoo class will have arrays for each type of Animal. Animals themselves don't have much internal logic other than how they age and making sure each of their fields are accurate to their current state. The Zoo class will handle all calculations of how they interact and how the attributes of each animal impact the game. Each Animal's class will have very little unique functionality as the biggest difference is how they are constructed.

Classes:

- Animal
    - Has:
        - Age in days
        - Name of the animal
        - Species of the animal
        - Cost (multiple of base cost)
        - Number of babies it has
        - Cost of food (multiple of base cost)
        - Revenue multiplier for species
        - Whether animal is currently sick
        - What age-group the Animal is in (baby, adolescent, adult).
    - Does:
        - Accessors for each of those fields
        - Get revenue animal produces in a day
            - Pre-conditions:
                - none
            - Post-conditions:
                - Returned revenue the animal produced that day
            - Pseudo-code:
                - If animal is a baby
                    - Return 2 * (animal's cost) * (animal's revenue multiplier)
                - Otherwise
                    - Return (animal's cost) * (animal's revenue multiplier)
        - Adjusts all fields after a day passes

- Pre-conditions:
    - Day has ended and animal must ‘age’
- Post-conditions:
    - Animal’s fields are all accurate to the new age
- Pseudo-code:
    - Age = age + 1
    - If age < 30 days
        - Set age group to baby
    - Else if age < 3 years
        - Set age group to child
    - Else
        - Set age group to adult

- Gets sick
    - Pre-conditions:
        - Animal was chosen to get sick by special event
    - Post-conditions:
        - Animals is sick
    - Pseudo-code:
        - Set whether animal is sick to true
- Takes medicine
    - Pre-conditions:
        - Animal was given medication to cure sickness
    - Post-conditions:
        - Animals is no longer sick
    - Pseudo-code:
        - Set whether animal is sick to false

- Monkey
    - Has:
        - Everything from Animal class but all Monkeys have:
            - Species of Monkey
            - Cost of 15,000
            - Number of babies of 1
            - Food cost multiplier of 4
            - Revenue multiplier of .1
    - Does:
        - Calculate revenue from boom random event
            - Pre-conditions:
                - Attendance boom occurs, boom revenue generated
            - Post-conditions:
                - This monkey’s boom revenue has been returned
            - Pseudo-code:
                - Generate random integer from 250 to 500, inclusive
                - Return this random integer
- Otter
    - Has:
        - Everything from Animal class but all Otters have:
            - Species of Otter
            - Cost of 5,000

                - Number of babies of 2
                - Food cost multiplier of 2
                - Revenue multiplier of .05
        - Does:
            - Everything from Animal class
- Sloth
    - Has:
        - Everything from Animal class but all Sloths have:
            - Species of Sloth
            - Cost of 2,000
            - Number of babies of 3
            - Food cost multiplier of 1
            - Revenue multiplier of .05
    - Does:
        - Everything from Animal class
- Zoo
    - Has:
        - Array of Monkeys
        - Array of Otters
        - Array of Sloths
        - Number of day
        - Today's revenues
        - Today's expenses
        - Today's boom revenue
        - Base food cost
        - Balance of player's bank account
    - Does:
        - Sets up day context (events, etc)
            - Pre-conditions:
                - Day has not started yet
            - Post-conditions:
                - Day is ready for user input
            - Pseudo-code:
                - Set boom revenue for day to 0
                - Call function to setup special events
                - Call function to update base food cost
                - Call function to update revenue and expenses
        - Ends day and updates values
            - Pre-conditions:
                - All choices for a given day have been made and zoo is ready for closing, takes whether the user is paying for medication that day or being a heartless monster.
            - Post-conditions:
                - All revenues and expenses from that day have been committed, unmedicated animals have died, etc; returns whether the player has lost (gone bankrupt).
            - Pseudo-code:
                - Day number = day number + 1
                - Store running total of revs and exps, starting at total = 0
                - Call function to update revenues and expenses

        - For each species
            - Total = total + (species' revenues) - (species' expenses)
            - For each animal in the species
                - Call function of animal to make day pass
                - If animal is sick
                    - If player is paying for meds
                        - Medicate animal
                    - Else
                        - Kill animal (remove it from the list)
        - Player's funds = player's funds + total
        - If player's funds are less than or equal to zero
            - Return true (game is over)
        - Else
            - Return false (not game over)
- Allows purchase of new animal
    - Pre-conditions:
        - Day logic has started and player wants to purchase a new animal
    - Post-conditions:
        - If player had the money to and decided to buy an animal, a random animal of that species is added to the animal lists and that cost is removed from funds, else nothing has changed.
    - Pseudo-code:
        - Get user input on which species to buy or if they want to exit
        - If input is invalid (not long enough, not a valid option)
            - Throw error, ask for re-input
        - Else
            - If user wants monkey and has >= 15,000
                - Create monkey with random name (3 years old)
                - Add monkey to monkey list
            - If user wants otter and has >= 5,000
                - Create otter with random name (3 years old)
                - Add otter to otter list
            - If user wants sloth and has >= 2,000
                - Create sloth with random name (3 years old)
                - Add sloth to sloth list
            - Else
                - Yell at user for invalid input or not having enough money, ask for re-input (including opportunity to leave).
- Update revenues and expenses
    - Pre-conditions:
        - Fields are correctly initialized and boom revenue value is accurate for the day (call day setup before calling this); takes whether user is paying for meds
    - Post-conditions:
        - Revenue and expenses fields are accurate to current context
    - Pseudo-code:
        - Set revenue and expenses to 0
        - For each species in zoo
            - For each animal of this species

- Revenues = revenues + (result of animal's get revenue function)
- Expenses = expenses + (animal's food mult) * (base food cost)
- If the animal is sick
    - If the user is paying for medication
        - Expenses += (animal's cost) / 2
- Revenues for monkeys += boom revenue for the day

- Update base food cost
  - Pre-conditions:
    - Day is being setup and food cost must be updated
  - Post-conditions:
    - Base food cost is a new value within 25% of the prior value
  - Pseudo-code:
    - Generate random number between 75% of current base food cost and 125% of current base food cost
    - Set base food cost to this random number
- Handle special events
  - Pre-conditions:
    - Currently in day setup function, day is being setup and this is the first time this function is being called for the day
  - Post-conditions:
    - Any effects of the random special event have been applied and will impact the rest of this day
  - Pseudo-code:
    - Generate random number from 0 to 3
    - If random number is 0 and user has adults (baby is being born)
      - For a random species
        - Choose random adult
          - Get that adult's species
            - Create and add new animal of that species with a random name (based on species)
    - Else if random number is 1 and user has animals (sickness)
      - For a random species
        - Choose random animal
          - Make that animal sick (call function)
    - Else if random number is 2 (attendance boom)
      - For each monkey
        - Add result of each monkey's boom revenue function to the zoo's boom revenue variable for the day
    - Else if random number is 3 (nothing happens)
      - Nothing happens
- Run logic for a single day
  - Pre-conditions:
    - An interface is calling for a new day to be presented to the user, the game has been initialized and the program is ready
  - Post-conditions:
    - A day has been simulated and fields have been adjusted to the accurate values after the day has run its course; returns whether the player went bankrupt at the end of that day

- Pseudo-code:
    - Call day setup function to handle special events, etc
    - If an animal got sick as part of random event, ask user if they want to pay for medication, store that response
    - Store result of calling end of day function, passing in the user's response
    - If that result is true (game is over)
        - Print user's funds + day number
        - Return true
    - Else
        - Ask if player wants to go to the store
        - If yes
            - Call function to handle purchase of animal
    - Return false

Testing:

Test Cases				
Variable	Type	Value	Expected Result	Actual Result
Whether player wants to pay for medication	edge	<empty>	Ask for re-input (no empty inputs)!	TBD
	bad	"yope"	Interpret as 'y', assume yes	TBD
		"nEEEP"	Interpret as 'n', assume no	TBD
		"x"	Ask for re-input (invalid input)!	TBD
		0.5	Ask for re-input (invalid input)!	TBD
	good	"y"	Account for med in expenses	TBD
		"n"	Kill animal, don't account for meds	TBD
What food user wants to buy for animals (Cheap, Regular, Premium)	edge	<empty>	Ask for re-input (no empty inputs)!	TBD
	bad	"chJASDasd"	Interpret as 'c', assume Cheap	TBD
		"Rdansds"	Interpret as 'r', assume Regular	TBD
		"Predasd"	Interpret as 'p', assume Premium	TBD
		"yep"	Ask for re-input (invalid input)!	TBD
		9	Ask for re-input (invalid input)!	TBD
		0.5	Ask for re-input (invalid input)!	TBD
	good	"c"	Set fields for cheap food option	TBD
		"reg"	Set fields for regular food option	TBD
		"Premium"	Set fields for premium food option	TBD
Whether player wants to 'go to the store' to buy a new animal	edge	<empty>	Ask for re-input (no empty inputs)!	TBD
	bad	"yope"	Interpret as 'y', assume yes	TBD
		"nEEEP"	Interpret as 'n', assume no	TBD

		0.5	Ask for re-input (invalid input)!	TBD
	good	"y"	Send user to buy animal function	TBD
		"n"	Don't send user to buy animal	TBD
What species of animal the user wants to buy from the store (Monkey, Otter, Sloth)	edge	<empty>	Ask for re-input (no empty inputs)!	TBD
	bad	"moasdasd"	Interpret as 'm', assume Monkey	TBD
		"ooooooooh"	Interpret as 'o', assume Otter	TBD
		"sHHHHHASD"	Interpret as 's', assume Sloth	TBD
		"yep"	Ask for re-input (invalid input)!	TBD
		9	Ask for re-input (invalid input)!	TBD
		0.5	Ask for re-input (invalid input)!	TBD
	good	"m"	Adds randomly named monkey to zoo	TBD
		"ott"	Adds randomly named otter to zoo	TBD
		"Sloth"	Adds randomly named sloth to zoo	TBD