Kenneth Hall
CS 162 400 W2018
2-4-2018

Design + Reflection

Animal class: age, cost, numberOfBabies need to be able to me modified by derived classes, so they are protected instead of private. The project required that food cost be defined as a constant in terms of what the Penguin ends up costing, so the implementation of calculating food cost for each animal ends up being a little "hacky" when implemented in the Zoo class
getAge(), I discovered, is critical in the copying over of animals from one array to another, since the only thing making an animal "unique" is its age
incrementAge() is called to make the animals age every day
I initially had built setCost() before I reread the specs on the assignment and found that the food cost had to be in terms of a constant. It's just kind of there, like a vestigial tail
setNumberOfBabies() is an important mutator for the derived classes, letting them define how many babies they generate per birth event

Tiger/Penguin/Turtle class:
Constructor takes age as a parameter; the game has animals introduced to the zoo at various ages, so it is important for age to be assigned during object initialization.
Accessor functions for food cost and pay off let the zoo know how much the animal costs to feed and how much they pay out, respectively

Zoo class:

Has four menu objects to hold all of the menus the game needs. All of the user input in the game occurs in the context of a menu, which allows me to leverage the input validation I've implemented in the Menu class. I got docked a few points previously for allowing certain floats and chars through on previous assignments, but I think I tweaked the method enough that it should kick back anything that isn't an int and prompt for a new entry.

feedChoice is used to track what feed option the user has currently selected; and is used in feed cost calculating and sickness likelihood

playerMoney keeps track of the player's money…

int numOf[Derived Animal] keeps track of the population of each exhibit. This number is important in determining when it is time to double the zoo's capacity for an exhibit, and for knowing the number animals eligible for getting sick or having babies or getting older.

[Derived Animal]ArrayCapacity keeps track of how big the exhibit is. Compares to numOf[Derived Animal] to determine when it is necessary to expand, and how much more expansion is needed

[Derived Animal]Pointer: used in the creation of new objects

arrayOf[Derived Animal] & copyOf[Derived Animal]: animal exhibits and a place to copy them to while the exhibit is being upgraded. This part definitely gave me the most trouble in implementation. I was originally trying to just copy the pointers from one array to the other, and then what I called delete down on the pointer in the old array, I found out that was a good way to cause some really weird things to happen to my animals, since the pointer in the updated array may or may not point to something resembling an animal anymore.

ageTheAnimals has each animal call its incrementAge function

feedTheAnimals has the player choose a feed type and then deducts money from the player's playerMoney

selectRandomEvent rolls a random event, with likelihood of events influenced by the feed choice. Normal feed yields a ¼ chance for sickness, premium feed is ⅛ and cheap feed yields ½.

sicknessEvent randomly selects an animal type at random until it finds a type with at least one animal, then randomly selects an animal out of that exhibit to die. The exhibit is copied over to the copy array, sans the "dead" animal, and then copied back. In my original design, I would get a seg fault if a sickness event occurred and all of the animals were dead, so I added a quick check at the top to make sure that if the zoo was empty, the sickness event didn't try to do anything.

boomInAttendance gives the player extra money, +251-500 per Tiger

babiesAreBorn selects an animal type at random, if there is no adult in that type roll again. If there is an adult in that type, add the appropriate amount of age 0 animals to that exhibit

incomeForDay tallies the amount each animal generates and adds it to the player's playerMoney

endOfDayBuyOption gives the player the opportunity to buy an adult animal at the end of the day

quitGamePrompt is called at the end of each day to see if the player would like to keep playing. Also kicks the player out of the game if they no longer have enough money to keep playing.

gameSetupPhase walks the player through the menus where they select how many of each animal they would like to begin with

gameStartMenuPrompt: called by game setup phase to access the Menu object for choosing starting animals

showStats: mostly for debugging/logic validation. Displays the amount of money the player has at the end of the day, and displays the number of animals in each exhibit, as well as the age of each animal in the exhibit

add[Derived Animal] handles the addition of new animals to the exhibits, as well as handles the upgrading of exhibits when a new animal is to be added and the exhibit is at capacity

main() just creates a zoo object and then calls the zoo's member functions until the game is over.

All of the interaction in my game was couched in the enforceValidInput() Menu object member function, so all of my tests can be described as either a test of the game's logic, or a test of valid user input.

TESTING

Condition	Expected Result	Actual Result	Notes
Player runs out of money	player kicked out of game at end of day	player kicked out of game at end of day	
tiger dies	tiger removed from zoo	tiger removed from zoo	
penguin dies	penguin removed from zoo	penguin removed from zoo	
turtle dies	turtle removed from zoo	turtle removed from zoo	
animal "dies" but zoo is empty	event skipped and error message displays, letting user know that an animal was supposed to die	event skipped and error message displays, letting user know that an animal was supposed to die	

tiger birth	one tiger added to zoo	one tiger added to zoo	
penguin birth	five penguins added	five penguins added	
turtle birth	ten turtles added	ten turtles added	
extra money event	+251-500$ each tiger	2 tigers on event -> 687$, 3 tigers -> 1172$	
player buys [Animal]	3 day old [Animal] added	3 day old [Animal] added	
player tries to pass string to menu	reprompt	reprompt	
player tries to pass float to menu	reprompt	reprompt	
player tries to pass int outside of bounds	reprompt	reprompt	

player passes appropriately valued int to menu prompt	record value and advance	record value and advance	

REFLECTION:
I really struggled with implementing a way to copy the array over, and once I got over that hurdle, the rest of the zoo design came together really nicely. I learned it was really important to have an accessor function for everything that makes an animal unique, in this case that's just its age, so that a *brand new* animal that is not discernible from the old animal can be built anew into the copy array and then you have to do the same thing to move the animal object back to the regular array.
I also got a really weird error that took me forever to figure out because the error message that I was getting didn't seem to point me in the right direction as far as I could tell. The problem ended up being that I had put an & character instead of % when I was getting random numbers with rand(). Tracing through 700 lines of zoo.cpp code to find that bugger was NOT fun. Maybe there was a way that I could have broken up some parts of the Zoo class into smaller chunks to make everything a little easier to sift through.