Tristan Santiago
February 4, 2018
CS-162-400
Project 2 Reflection

The design plan was written at the very onset of the project. It was just a simple sketch-up class definitions and variable declarations. The design reflects my linear thought process in how this program would actually work. I took the very same approach used with the previous assignment in this project. I started with a very small piece, built it, tested it, and made sure it functioned properly, before moving on to another small piece and doing the very same. I repeated this process until I had a skeleton that I thought would be rather easy to fill with some of the tougher logic associated with this program, namely dynamically-allocated arrays.

Simple Design Plan

Classes
Animal Base Class
- Age
- Cost
- # of Babies
- Base Food Cost
- Payoff

Tiger
- Cost = $10,000
- # of Babies = 1
- Base Food Cost = $50
- Payoff = 20%

Penguin
- Cost = 1,000
- # of Babies = 5
- Base food cost = $10
- Payoff = 10%

Turtle
- Cost = 100
- # of Babies = 10
- Base food cost = $5
- Payoff = 5%

Custom(name = N)
- Cost = C
- # of babies = B
- Base food cost = F
- Payoff = P

Zoo
- Dynamic array per animal (each animal's array starts at 10, but will double in size, if more animals are needed)

Food Type
- Cheap: half as expensive, sickness(random event) twice as likely

Generic: behaves normally
Premium: twice as expensive, sickness half as likely to occur
Function that passes multiplier.

StartGame
Starting Numbers:
Money: 100,000
Choose quantities of animals to buy:
Tigers:
1 or 2
Subtract from 'Money'
Set age to 1(day old)
Penguins:
1 or 2
Subtract from 'Money'
Set age to 1(day old)
Turtles:
1 or 2
Subtract from 'Money'
Set age to 1(day old)
Ask what kind of food to buy
Cheap
Generic
Expensive
(subtract food cost)
One random event (w/ message in separate txt file to display) rand (1-4)
Sickness
Animal dies(remove animal)
Boom in attendance
Generate random bonus from 250-500
Baby animal born
Random animal has a baby
Check to see if animals are >= 3 days old
If selected animal isn't >=3, try different animal
If no animals are old enough, then run 4
Nothing happens
Calculate profit(payoff*animal)
Add in bonuses if any
Ask player if they'd like to buy adult animal(>3 days old)
What kind of animal?
Subtract cost from bank

ALL WITHIN THE GAMEPLAY LOOP.
End turn(day)
Add one to animals age
Subtract food cost

Ask user if they want to play again or not
Output current info
If money is <= 0, game over automatically and end game.

As stated previously, the initial design was not very sophisticated. The final program's design was only marginally different from the initial design. I attempted no extra credit (as there simply was not enough time for me to implement it) for this assignment. The major problems greatly interfered with the implementation of extra credit.

Problems Encountered During the Programming Phase

The first major error I encountered in this project was with bad allocation:
terminate called after throwing an instance of 'std::bad_alloc'
what(): std::bad_alloc
This error occurred when the random event triggered sickness and tried to kill animals that did not exist.
My first attempt at correcting the issue was something along the lines of:

```
int i;
    if(randomChoice == 1 && TigerCage->pa[i]) // Attempt to evolve the logic for random sickness.
```

However, this code resulted in no animal ever dying, which is not what I wanted. Eventually, I figured out that I was not pointing to the right array, because I wasn't properly allocating an array for each animal type. This led me to revisit the logic I used for my dynamically allocated arrays. Initially, I was only creating one array and placing all of the animal objects into it. While trying to figure out what the problem was with the bad allocation, I came across my second hurdle: memory leaks.

```
==5087== LEAK SUMMARY:
==5087==    definitely lost: 48 bytes in 3 blocks
==5087==    indirectly lost: 60 bytes in 3 blocks
==5087==      possibly lost: 0 bytes in 0 blocks
==5087==    still reachable: 72,704 bytes in 1 blocks
```

It took me a very long time to draw the conclusion that having 72,704 bytes still reachable after the program terminated was not something that I should be concerned with addressing. However, the other lost bytes could not be ignored, as they were generating all sorts of errors with valgrind. One I remember in particular was "invalid write size of 4 when 8 was allocated" or something to that effect. After so many hours of frustration with debugging, I cannot exactly recall how I ended up stumbling onto the solution, but instead of doing "Zoo *PenguinCage = new Zoo;" I changed it to "Zoo PenguinCage;" This generated dozens more errors mostly because of the ->'s I was using, so I simply replaced all of the "->" with ".", resulting in compiling with no memory leaks. Debugging took an incredible amount of time to finally resolve with no errors, significantly hamstringing the programming process, but somehow I managed to figure it out time in time. Overall, I am pleased with the final version of this program and how closely it matched my original design.