Initial Design

Ok so initial design - the best part. First, I'm going to have several files (with their respective header files as well) below:

- main.cpp
- animal.cpp
- zoo.cpp
- menu.cpp
- event.cpp
- validation.cpp

Main.cpp will be a simple file that includes the menu function and starts the simulation. I'm condensing most of the information into their respective header/function files and keeping main.cpp rather clean and tidy.

Menu.cpp will generate a introduction menu that notifies the player of how much starting income they have ($100,000). It then asks the user to select how many turtles, penguins, and tigers they want to initially purchase (1 - 2 initially). We should have the three class arrays generated beforehand, so when the user inputs all the necessary information (Animal Type and amount) it calls a function which creates 1 or 2 additional objects and appends those to the respective array. I can wrap all of the initial process into an initial() function that generates all of the necessary information. Afterwards, the user is only ever going to add 1 animal per day.

After we've generated all of the animals, we can just loop through each array and decrement the money we have in the bank. Or, I can subtract from the bank each time an animal is purchased (before it's added to the array). Thinking about it, it might be easier to just determine the totals of each animal via for loop and just determine the value from there. I'll determine what to do once I get to that crossroad.

Event.cpp is going to contain the random events that can occur throughout the day. I will be using rand() to determine what occurs to the animals in the zoo. For every option but the "business boom" option, I will have the function randomly determine which array it will be targeting. So:

1. Sickness
    1. Randomly select the animal array
    2. Using rand() we generate a random index number
    3. We use that random index number and delete that animal from his/her array

2. Bonus
    1. Determine the length of the tiger array
    2. Multiply the amount of tigers in the array by $500 and store that in a variable
    3. Increment the payoff total for the day by the stored variable

3. Birth (tricky)
    1. Randomly select the animal array
    2. Loop through that array and determine if there is an animal >= age 3
        1. If no animal is eligible we should break out of that if statement and select a different array to test for age.
        2. If none of the animal arrays have a valid candidate for giving birth, we simply return a string saying that there was no eligible candidate and move on.
    3. Once we've determined that there's an eligible animal we can just run a for loop and add however many objects (children) to that same object array. We don't need to worry about assigning a parent, so there's no need to fret over WHICH animal over age 3 is the one giving birth, we just need to know whether or not the array is valid across the board.
4. Nothing Happens
    1. Return a string simply stating that the day was uneventful and nothing happened.

Progression and Changes (Post-Completion)

I reached and climbed so many walls to finish this project that I'm pretty sure even Donald Trump began to question whether walls were a good idea.

I ended up realizing that I would need to invest a lot of time into this because the concept of pointers/references is still something that confuses the hell the out of me when I think too hard about it. Unlike my Die game project which I didn't build in chunks, I opted to build the Zoo Tycoon project in chunks so that I could test arrays, functions, etc and make sure that everything was tip-top before moving on to the next step.

The project turned into hell on earth for me. Between misallocation errors at the very end of my program **(fig. 1)**, and array access issues I was starting to get sucked into a quagmire of troubles.

Fig. 1 - Hell on earth

```
0x100306ce0
0x100306f70
0x100306f90
0x100306fb0
0x100306dd0
0x100306dc0
0x100306de0
0x100306e00
0x100306e20
0x100306e40
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0x0
0x0
PENGUINS FINAL COUNT 18
TURQUINS FINAL CAGE SIZE 20
proj2(41274353,0x1001213c0) malloc: *** error for object 0x100306ce0: pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
(lldb)
```

I managed to resolve the malloc issues after a code review with Harlan Waldrop. We went through a few of my delete arrays and he immediately saw some minor problems. We fixed those up and the moment I got off the line with him, I hit another malloc issue - this time related to resizing the array. What was the problem here? I was basically running code like this:

```
for (int i = 0; i < oldSize; i++) {
    delete turtles[i];
}
delete [] turtles;
```

inside of my resize function. This was, in theory, supposed to clear the initial array's index, but I ended up deleting a bunch of pointers that had never been allocated to begin with (I used a function to set them all to null each time I called resize and the first time I created the arrays). Removing that for loop and only deleting the array completely resolved my issues.

Afterwards I started to bump into a *bunch* of issues regarding rand(). Rand() can be a tricky little piece of garbage because the modulo operator isn't particularly accurate when you're hitting bigger value ranges. It also has a side effect of returning values on the lower range of the spectrum. However, it did it's job for smaller values like finding a random number between 1 - 4. In another section of my code (when I randomly determine which animal gives birth), I had to use int_distribution libraries in order to provide a more *"random"* number. Rand() simply would not work inside of my while loop and I had to use a different means of feeding a random value into my switch expression.

Once all of these issues were resolved it was a very straight forward process. I organized three functions called beginDay(), randomEvent(), and endDay(). I ran these functions inside of a run() function which was called by the simulation class (created inside main.cpp). My function file for zoo.cpp became really big so I opted to move functions related to simulating the game from the zoo.cpp function file into the simulation.cpp file. This was sort of an accidental discovery because I was originally going to make Simulation functions that could access information from zoo and animal, but I was unable to figure it out in a timely manner. So, I opted to just move functions into simulation.cpp and maintain the class reference. You'll see this within simulation.cpp where I have 3 functions prefixed by **zoo::** .

I ran the project through flip a bunch of times and managed to reach this **(fig. 2)**:

Fig. 2 - Pure Bliss after a week of work

```
==8949==
==8949== HEAP SUMMARY:
==8949==     in use at exit: 0 bytes in 0 blocks
==8949==   total heap usage: 35 allocs, 35 frees, 2,464 bytes allocated
==8949==
==8949== All heap blocks were freed -- no leaks are possible
==8949==
==8949== For counts of detected and suppressed errors, rerun with: -v
==8949== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
flip2 ~/cs162/proj2 140%
```

Originally, I was up to my neck in misallocations and errors. Now, the program runs perfectly without any issues. It's quite verbose, but I managed to crawl through this project mostly on my own and with the help of some kind students on piazza and a TA. Indeed, this was an incredible learning process for me.

There's a lot more that I want to do for this program which I'll try to do on Sunday evening if there's time after completing Lab 4. Namely, I'd like to tackle some of the extra credit challenges and *perhaps* implement templates so that I don't need to have 3 different add, resize, and remove functions for each object array.

Testing Plan

Test Case	Input Values	Driver Functions	Expected Outcome	Observed Outcome
Add first tigers to array	No input / Running a function	addTiger();	Tiger is added to the array	Tiger is added to the array
Add second tiger to array	No input / Running a function	addTiger();	Tiger should skip over the first index and get added into the next empty index	Tiger is added to the array and and skips over to the next available index.
Resize Tiger array	Add 11 tigers to array.	addTiger() + resizeTiger();	10 tigers are added and the resize function is called when an 11th tiger is added.	Resize function runs successfully but runs into a malloc error
Resize Tiger array post fix	Add 11 tigers to array	addTiger() + resizeTiger()	Same as above	Resize works without any malloc errors.
Test Random Events #1 - An Animal Dies	randomEvents with a roll value set to 1	randomEvents();	Animal is removed from the array if it exists - otherwise a message is displayed in its stead saying that nothing happened.	Animal from the last index (equal to total animals) is popped off and count is decremented. A message is displayed otherwise when an animal doesn't exist in the selected array.
Test Random Events #2 - A boom in business	randomEvents with a roll value set to 2 Tigers are in the array	randomEvents();	A business boom occurs and the tigers generate a random amount of money.	Money is generated properly and added to a bonus int. Value produced is out of bounds - resolved with a stricter rand() boundary.
Test Random Events #3 - Birth	randomEvents with a roll value set to 3 All three animal types are present with an age of at least 3 in the array.	randomEvents()	An animal is randomly selected and gives birth.	Animal is selected, gives birth, and exits the loop.
Test Random Events #3 -Birth x2	same as above except there is only 1 eligible animal at index 2	randomEvents()	An animal at index 2 (turtles) gives birth	Stuck in a while loop - modulo operator is working incorrectly and not seeding a random number. Resolved with random_direction (et al) libraries.
Test Random Events #4 - Nothing Happens	randomEvents with roll value set to 4	randomEvents()	Nothing happens as per random event	Nothing happens
Age animals up at beginning of new day	no input	beginDay()	When the function runs the animals should be aged up by one year.	Existing animals are aged up by one year.
Animals are fed at the beginning of the day and the bank is decremented by the total food cost	no input	beginDay()	Function runs and the animals, if they exist, are fed.	Function runs and determines which animals are available to be fed. Animals are then fed and the money is subtracted from the bank.
Bonus and payoff values are incremented to the bank at the end of the day	no input	endDay()	Function runs and adds the bonus (if eligible) and payoff values for each animal (that exists) to the bank.	Bonus and payoff values are incremented to the bank and the bank displays its end of day total.
User is prompted for if they want to add animals (Run twice)	Input corresponding menu item	endDay()	Menu prompts user. If user inputs a bad value it asks them again. If user says yes it lets them select an animal - otherwise it moves on to the next menu.	Menu prompts user. Out of bounds inputs are ignored and user is asked to input again. User can successfully choose an animal and move on. User can choose to not add an animal and it does so successfully and moves on to the next menu item.
User is asked if they want to continue or quit the program.	Input corresponding menu item	endDay()	Menu prompts user to see if they want to continue or quit. Hitting continue will continue the game and loop through beginDay(), randomEvent(), and endDay(). Hitting quit will end the simulation.	Bad values redirect user for re-input. Hitting quit will exit the game as expected. Continue causes the game to loop again and repeat the day.