Computing IV Sec 203: Project Portfolio

Senny Lu

Spring 2025

Contents

Time to Complete Portfolio: 6 hours

1 PS0: Hello SFML

1.1 Discussion

This was the first assignment where we created a movable sprite within a window. This is also where I was first cast into a sea of knowledge with graphics from SFML, and where I first learned how to finally swim—by first holding my breath. This was the starting point and an introduction into a world of learning.

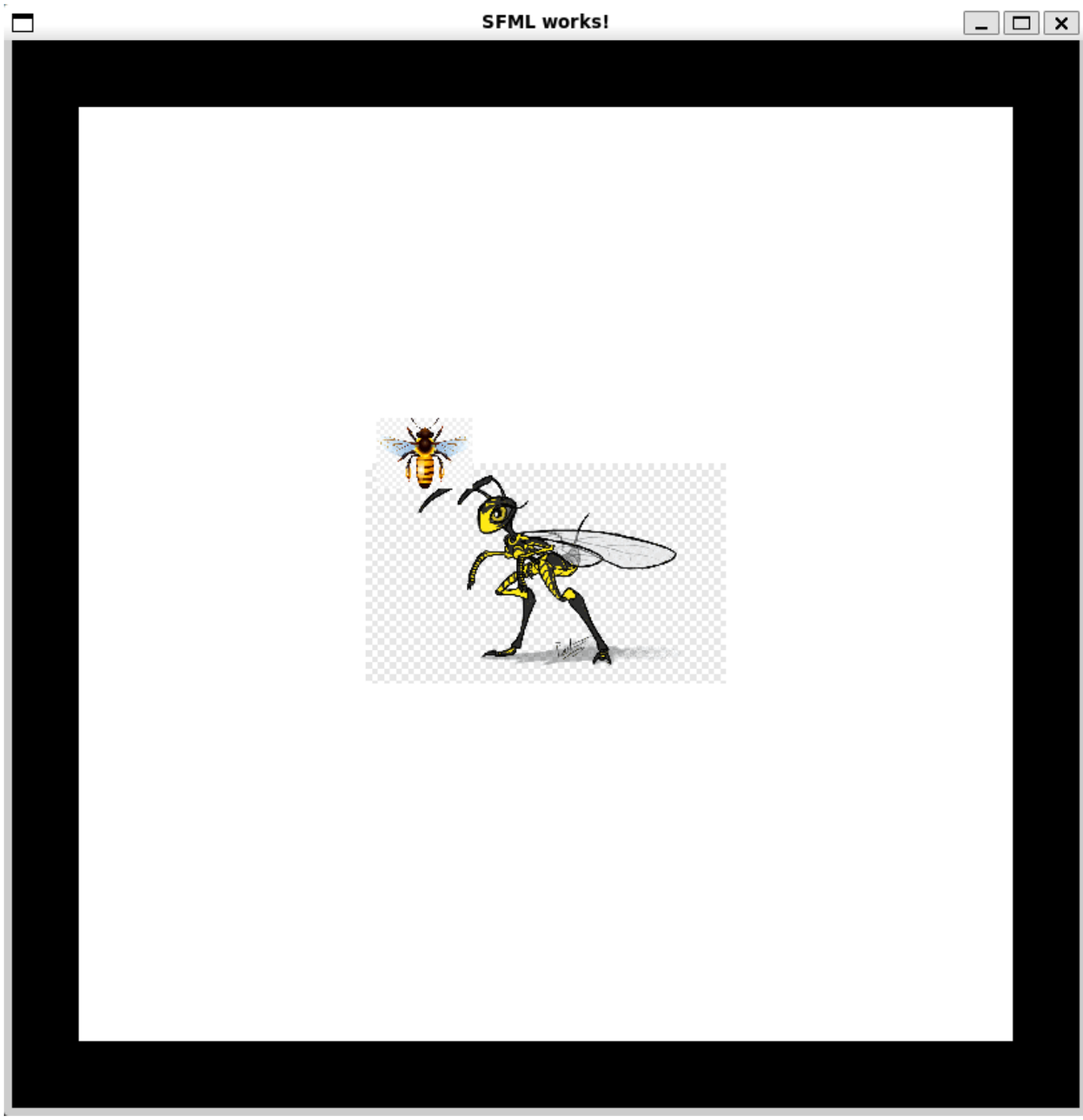

Figure 1

1.2 What I accomplished

I accomplished creating a window with a sprite that can move using arrow keys or WASD.

1.3 What I already knew

Nothing.

1.4 What I learned

I learned about SFML libraries and how to create a window with movable parts.

1.5 Challenges

The main challenge was finding the information needed for the SFML types.

1.6 Codebase

2 PS1: LFSR/PhotoMagic

2.1 Discussion

In this project I created a FibLFSR class with functions that shift the LFSR (Linear Feedback Shift Register). This allowed the class to create a random LFSR. Using this LFSR, I am able to decrypt and encrypt an image by taking the pixels of the image and excrpting them using the LFSR class using a given seed. Next by using the same seed, we are able to decrypt the image back to its original form. This project was interesting because it gave a deeper preview into both SFML libraries and LFSRs. This uses a string for LFSR due to it becing easier to represent the data.

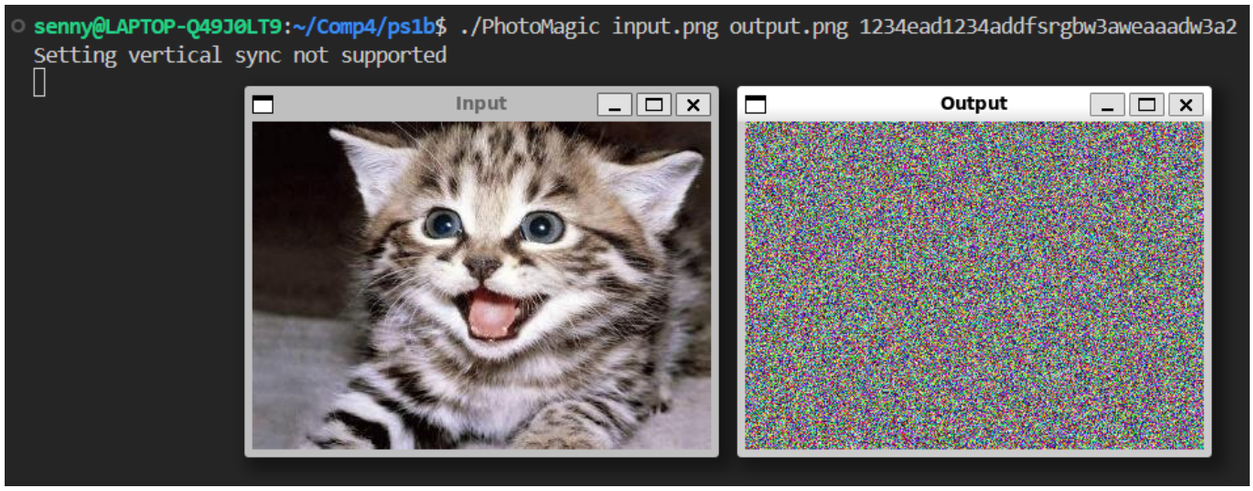

Figure 2

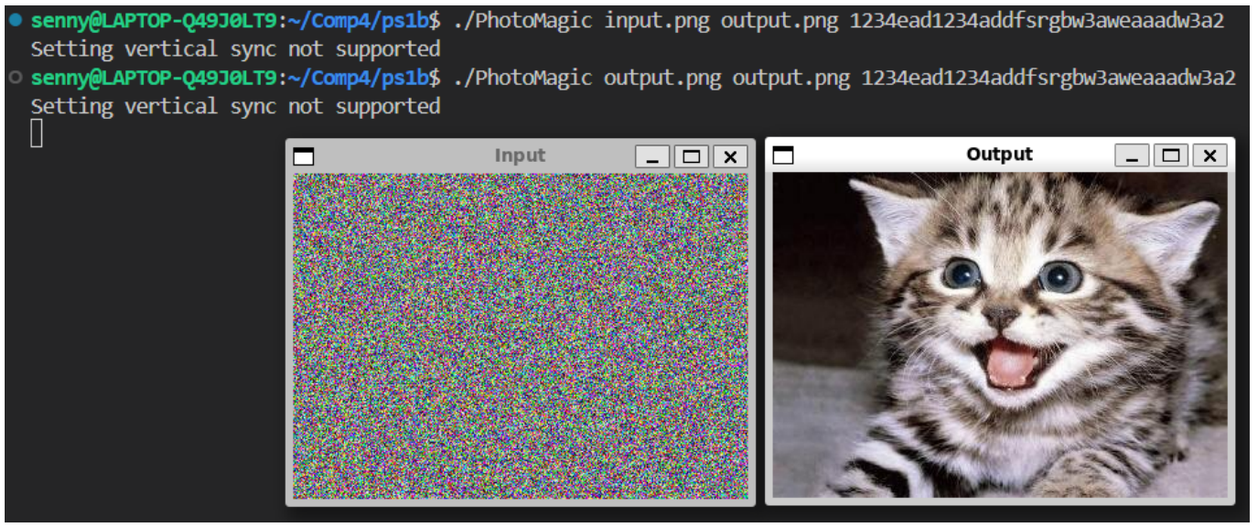

Figure 3

2.2 What I accomplished

I accomplished creating an encoder and decoder for image files, and created a LFSR class that is able to randomize numbers.

2.3 What I already knew

I already knew how to create the windows for SFML from ps0.

2.4 What I learned

I learned a lot about SFML libraries, especially about displaying the images and chaning the size of the images and writing out to an image. I also learned about LFSRs and how

they are able to create completely random numbers, and how they are used to decode and encode data.

2.5 Challenges

One major challenge for me was trying to write the LFSR first by hand because I want to experience how the process would be like. This was especially difficult because I made small mistakes that iterate through the entire exersice.

2.6 Codebase

3 PS2: Triangle Fractal

3.1 Discussion

This assignment required me to create a triangle fractal. This was an interesting project because it required me to use SFML libraries and a good bit of math to create a beautiful triangle fractal that repeats itself depending on the iterations I requested. I used `CircleShape` to represent the triangles so that I have access to the radius; this allowed easy creation of the triangle itself.

The radius is equal to

$$\frac{\text{length}}{2 \cdot \cos\left(\frac{\pi}{6}\right)}$$

I used the length of the triangle to determine the radius of the circle of the triangle. With the radius, it was easier to find the center point of the triangle, which I used for the locations of the triangles and their child triangles. Because the origin of my triangles is the center point, I was able to compute the positions of the 3 child triangles by adding a multiple of length to the x-coordinate and a multiple of radius to the y-coordinate. I also computed the position of my triangle and its child triangles in the window itself using a geometric series to determine the max width and length of my window based on the length of the triangle.

The max window length is $4.5 \times$ radius, and the max window width is $2.5 \times$ length.

- Top child triangle = $(\text{length}/4, -1.25 \times \text{radius}) + \text{location of parent triangle}$
- Left child triangle = $(-0.75 \times \text{length}, 0.25 \times \text{radius}) + \text{location of parent triangle}$
- Right child triangle = $(0.50 \times \text{length}, 1 \times \text{radius}) + \text{location of parent triangle}$

Each triangle has the origin set at the center point of the triangle. This allowed me to create a window that perfectly fits the triangle fractal.

3.2 What I accomplished

I accomplished creating a triangle fractal recursively with functions and used math to generate the triangle at the right locations.

3.3 What I already knew

I already knew most of how the math worked.

3.4 What I learned

I learned a lot about position, rotation, and alignment in the SFML libraries. I also learned that recursion can take a long time depending on the function.

3.5 Challenges

One challenge was making the math work inside the code. This was especially difficult because I needed to rotate the triangles to create the fractals.

3.6 Codebase

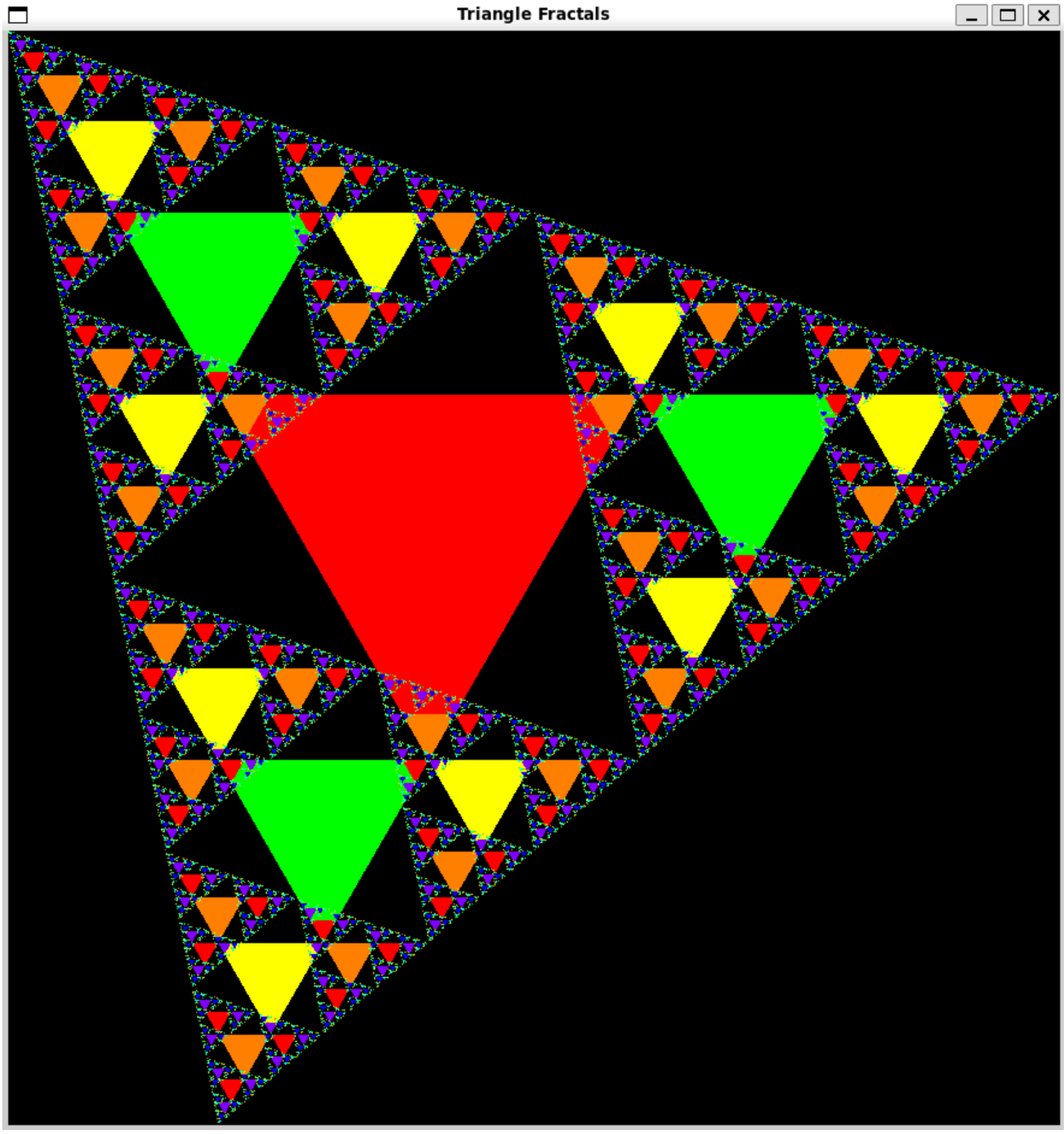

Figure 4

4 PS3: NBody Simulation

4.1 Discussion

This assignment required me to create a Nbody simulation of a set of particles. These particles were things such as the sun, earth, and moon. I was tasked with creating the simulation of them moving according to their velocities and their gravitational pull towards other planets. This project is fascinating in that it creates an amazing real world example of useful code to create an animation. I used additional member variables for the name of the program and shared pointers to store both the textures of the CelestialBodies in CelestialBody and used a vector of shared pointers of CelestialBodies inside of Universe to access each CelestialBody. I created setter functions for Universe and CelestialBody, and the insertion operator for Universe uses the insertion operator for CelestialBody for each shared pointer of CelestialBody in the vector of shared pointers. I used a vector to store the velocity changes which I calculated once then used it to calculate the positions and velocities of the particles. I followed the instructions to calculate net force of each particle first then calculate the accelerations using the net force then the new velocities using the accelerations then the positions from the new velocities and finally updating everything. I used a vector of shared ptrs of CelestialBody in Universe so that the shared ptrs of each texture of each CelestialBody does not need to be copied when pushing the CelestialBody onto the vector of Universe

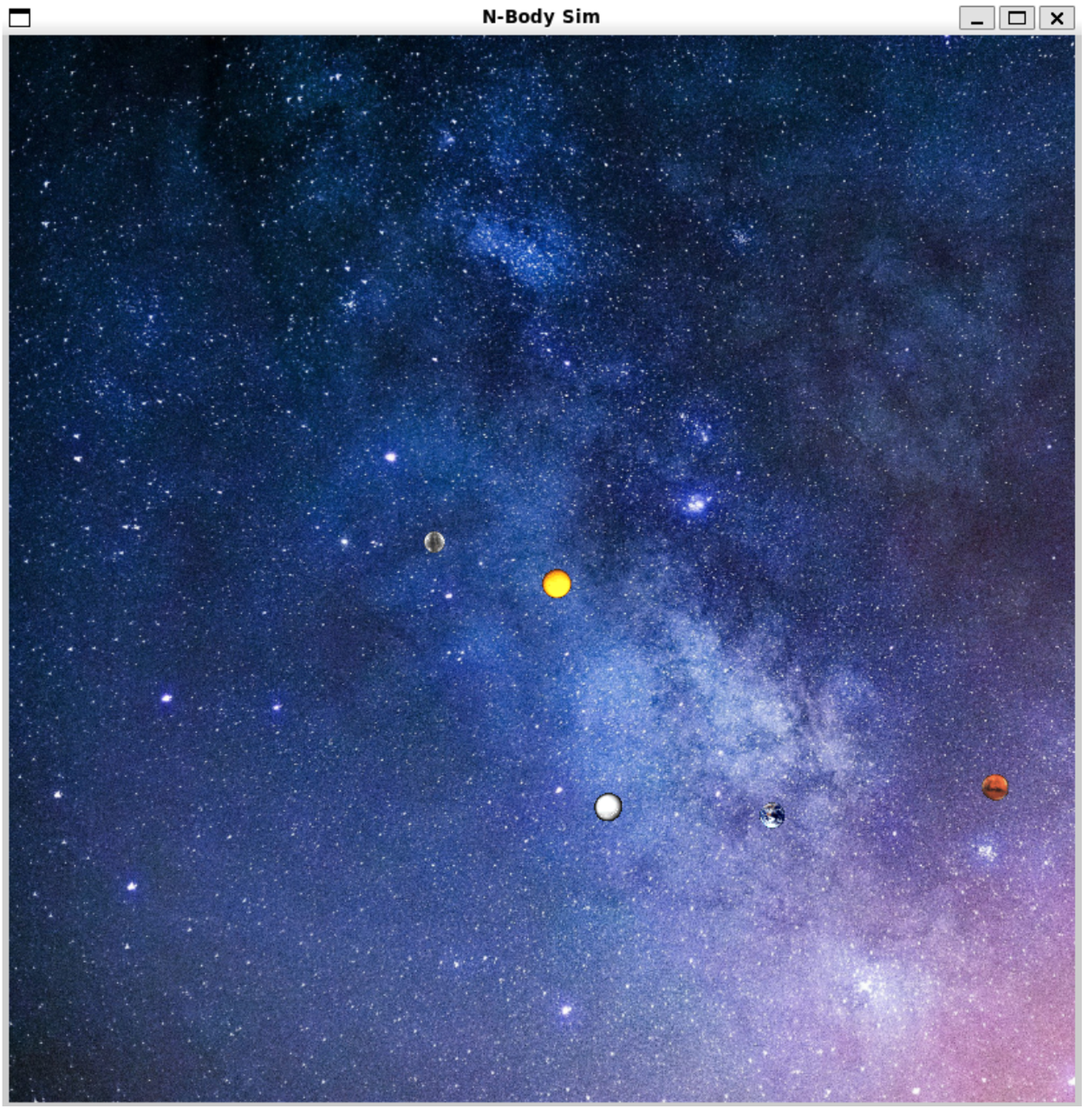

Figure 5

4.2 What I accomplished

I accomplished creating a simulation that animates the movements of particles using their relationship with other particles.

4.3 What I already knew

I already knew how to create the window and drawing each element of the particles.

4.4 What I learned

I learned a lot about smart pointers and while I don't fully understand why I needed to use them, I can understand that it makes the simulation run faster without constantly reloading them.

4.5 Challenges

I do not understand fully why I needed to use shared pointers. I had a working version of this program without using shared pointers, instead using a vector of CelestialBodies, but I guess it would be slower if they were copied each time a CelestialBody was pushed onto the vector. It does not work with unique pointers since they cannot be copied

4.6 Codebase

5 PS4: Sokoban

5.1 Discussion

This assignment tasked me with recreating an old but nostalgic Sokoban game. This was my favorite assignment because it brought the joys of creating a game, both with the frustrations of bugs and the satisfaction of solving the issues. I used a one-dimensional array because it's easier to iterate through. I created a separate vector of characters to store the initial level for resets, created the boolean data for whether the player is standing on top of a teleporter or a storage location, and essentially created a copy of all data for the reset command with an additional vector of characters to store the initial level and a 'Vector2u' for the initial player location. The collisions were checked using many predicate functions to make sure a player movement is valid. This was done by looking at the spot the player is going to move into and changing the effect based on whether it was a wall, empty space, crate, teleporter, or storage location. If the player's next location is a crate, the 'movePlayer' function will call a function that checks if the block ahead of the player can move. That function will check if the block can move to its corresponding location. I used a vector to store the level data as characters. Similar to the NBody simulation, I used smart pointers so that I only needed to load each texture once. In the 'otherTeleporterIndex' function, I used the 'find_if' function from '<algorithm>' and a predicate lambda as one of the parameters to the 'find_if' function to find the index of the other teleporter.

Figure 6

5.2 What I accomplished

I accomplished creating a childhood game that works exactly like the old game, and I added teleporters to give the game a bit of a new edge.

5.3 What I already knew

I already knew how to draw the windows and how to create the structures.

5.4 What I learned

I learned a lot about what I enjoy. I learned a lot about how bugs can appear in video games, especially in places where you would least expect them. I had a bug with movement because I was using a one-dimensional array to store the data of the locations.

5.5 Challenges

The main challenge came with locating where the bugs took place. For one of the bugs I encountered, it was when the player would walk off the screen from one side and teleport to the other side. This had to do with how I was storing the player locations. This took me quite some time to find.

5.6 Codebase

6 PS5: DNA Alignment

6.1 Discussion

This assignment required me to use either the Needleman-Wunsch algorithm or Hirschberg's algorithm to find the lowest-cost alignment of two sequences such that we don't use a recursive solution that would consume excessive memory. This was an interesting assignment because there are a lot of real-world applications of this type of data that can have real impact. I used a two-dimensional vector of ints to hold the matrix of cost data and strings to hold the two sequences. First, I populated the bottom and right sides of the matrix starting with 0 cost at the bottom-most right-most point, then filled the matrix up and to the left from the corner, increasing by 2 each time to represent the cost of inserting a gap. Next, I used a double for-loop with the sizes of both strings, starting at the upper limit: index sizeX - 1 and sizeY - 1. Then I inserted the lowest total cost from either the right cost, below cost, or diagonal cost. For the alignment, I went back through the matrix starting from the top-right and picked any possible path—either right, down, or diagonal. For the path to go right or down, the cost needed to be exactly 2 less, and for it to go diagonal, I needed to calculate the penalty and make sure the cost was correct. For each movement of the path, the alignment is printed.

Here is the data I collected using the Needleman-Wunsch algorithm:

data file	Calculated (MB)	Measured (MB)	Difference (%)
ecoli2500.txt	25.020004MB	24.06MB	3.99004%
ecoli5000.txt	100.040004MB	95.72MB	4.51316%
ecoli7000.txt	196.056004MB	187.37MB	4.63574%
ecoli10000.txt	400.080004MB	382.2MB	4.67818%
ecoli20000.txt	1600.160004MB	1526.7MB	4.81168%
ecoli28284.txt	3200.1649MB	3,053.02MB	4.81965%

6.2 What I accomplished

I was able to accomplish creating the Needleman-Wunsch algorithm for DNA alignment, and I compared and contrasted the data from different machines to find the time taken and memory used.

6.3 What I already knew

I already knew most of how the math worked with the Needleman-Wunsch algorithm, and how the data structures for this assignment worked.

6.4 What I learned

I learned more about space—more specifically, memory—when it comes to doing big calculations in these programs. I learned that I should be more open to working with a group mate because that could've saved me more time, since I struggled greatly with some portions.

6.5 Challenges

The main challenge came with the 'min3' function that I created. There are many edge cases that could break my 'min3' function, and I am amazed by how few bugs appear in the daily apps I use when I couldn't even find the simple bugs that broke my 'min3' function. I also was unable to figure out how to implement Hirschberg's algorithm. I understand that it allows us to use much less memory by only using a portion of the Needleman-Wunsch algorithm, but I was unable to code it into my program because I couldn't fully understand how it works.

6.6 Codebase

7 PS6: RandWriter

7.1 Discussion

For this assignment, I created a random writer that used k-grams of order k to create the frequency and probability table for data. This assignment was interesting in that we could generate entire texts that could be read and somewhat interpreted by humans, due to it using something similar to large language models in how it predicts what the next word or character is going to be. This project gave me a deeper glimpse into how LLMs like ChatGPT and DeepSeek work—how they piece together words that are usually correct together to form, or at least attempt to form, the correct answer. I ended up creating the constructor of RandWriter by generating each k-gram one by one. I initially tried to populate the map by using only one string of length k and removing the first element of the string and appending the next character, but I ran into an issue with the wrapping of the string when it reached the end and with empty characters, which I did not know how to solve.

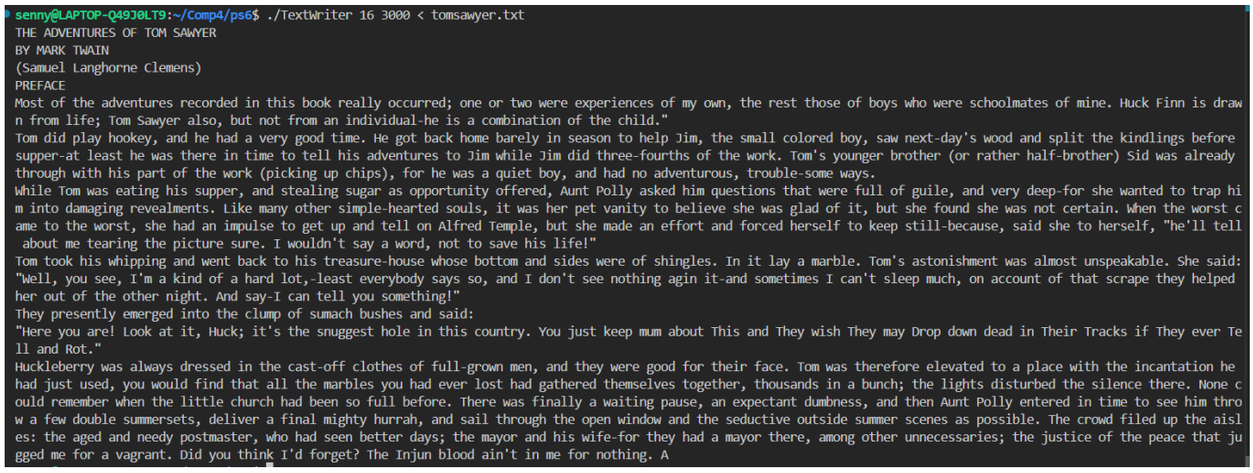

Figure 7

7.2 What I accomplished

I accomplished creating a random character writer that can generate random characters depending on the previous k characters to create readable text.

7.3 What I already knew

I already knew how to use lambdas in the ‘for_each‘ algorithm.

7.4 What I learned

I learned a lot about how code can generate random output that can mimic humans. It’s really cool to see how similar code output can resemble human output. This reminded me of the infinite monkey typewriter Shakespeare thought experiment—where an infinite number of monkeys could eventually create the entire works of Shakespeare just by randomly typing for an infinite amount of time. If they were able to incorporate this, wouldn’t they finish it in a finite amount of time?

7.5 Challenges

The main challenge I faced was the creation of the maps. I used a map of a map like ‘std::map<std::string, std::map<char, int»‘. This allowed me to store the frequencies of the characters that came after each k-gram. Using those frequencies, I could calculate the frequencies of the k-grams themselves. The issue I ran into was when I initially tried to construct the data by using one string of a k-gram and deleting the first character, then appending another character. I wasn’t able to get it to allocate all the data correctly.

7.6 Codebase

8 PS7: Kronos Log Parsing

8.1 Discussion

This assignment required me to use regex to find the start and end points of programs in given files and output the data into a separate file. This assignment is very useful in that this would most likely be the kind of work I will need to do in the future—in that I need to analyze data and find the important pieces of information. Looking through the given log files, it was interesting to see that it can be quite hard to piece together some of the information. They all had some sub-data with varying names or numbers to represent the data itself. I remember seeing a 'log.c.177 server started' while our project required us to find 'log.c.166 server started'.

I used the Boost libraries for the time calculations. It seemed easier to just do it with arithmetic, but actually using the Boost library for time did make things easier than calculating by hand, because the libraries themselves provided a lot of the math required to do the conversions.

I split the program accordingly for each part I needed. In this project, I needed to find each time the server started, each time it ended, and the time in between. I did this with 3 regular expressions: one to find lines with the string (log.c.166) server started and another with the string oejs.AbstractConnector:Started SelectChannelConnector.

In each of these lines, there are timestamps given as year, month, day and hour, minute, second, which I used to calculate the total time between start and stop.

This is the output of the program with one device boot start and completion: Device Boot Report
InTouch log file: device1_intouch.log
Lines Scanned: 443838
Device boot count: initiated = 6, completed = 6
=== Device boot ===
435369 (device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759 (device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
Boot Time: 183000ms

8.2 What I accomplished

I was able to accomplish parsing through the data in the given files to find the important information and output it to a file.

8.3 What I already knew

I already knew what I didn't know, and what I needed to learn to finish this project.

8.4 What I learned

I learned a lot about regex and how they integrate with C++ strings, and I learned that real-world data doesn't need to be human-readable because we, as humans, can use programs to create readable formats. Even though the given files were not very readable, they were packed with important (and sometimes unimportant) information that could be sorted with regular expressions to get the parts we want.

8.5 Challenges

The main challenge was finding the correct start and end points in the files. There were multiple 'server started' entries, and it was important to find the correct one that corresponded to the data we wanted—specifically 'log.c.166'. Another challenge (sort of) was creating the capture groups correctly so that I could use indexing with 'smatch' to get the values.

8.6 Codebase