Steven Mellett
Prof. Dr. Yelena Rykalova
COMP IV Sec 204: Project Portfolio
Spring 2024

**Table of Contents:**



Time to complete: 15 hours

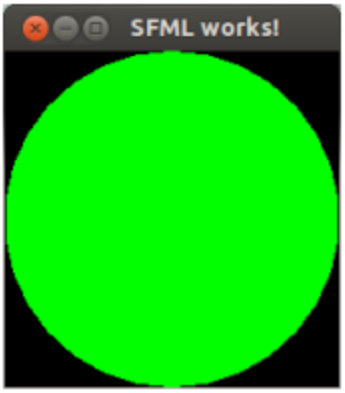
# 1 PS0: Hello World with SFML

## 1.1 Discussion

This is one of the first projects I coded in C++ so the majority of my time was spent learning the syntax around defining classes and how to specify their scope. Most of my prior knowledge was from previous Computing coursework that was mainly about the C language, so I had a good understanding of basic coding conventions that would allow me to complete this project without any major issues.

I was also not familiar with the Simple and Fast Multimedia Library before this assignment. The only confusing part about this library is the loop to open the window, which involves sf::Event and sf::RenderWindow objects of which I gained a better understanding of in later projects.

The goal of this project was to extend a demo program that creates this window:




The requirements are specified as below:

1. Draw a sprite (the technical term for a movable image that makes up a larger image).
2. Make the image sprite move.
3. Make the sprite respond to keystrokes.
4. Add another feature of your choice.

This involves the use of a couple core SFML objects. In order to have a sprite show in the window it needs a Texture object. So I started by initializing a sf::Texture and used the .loadFromFile to load this png:

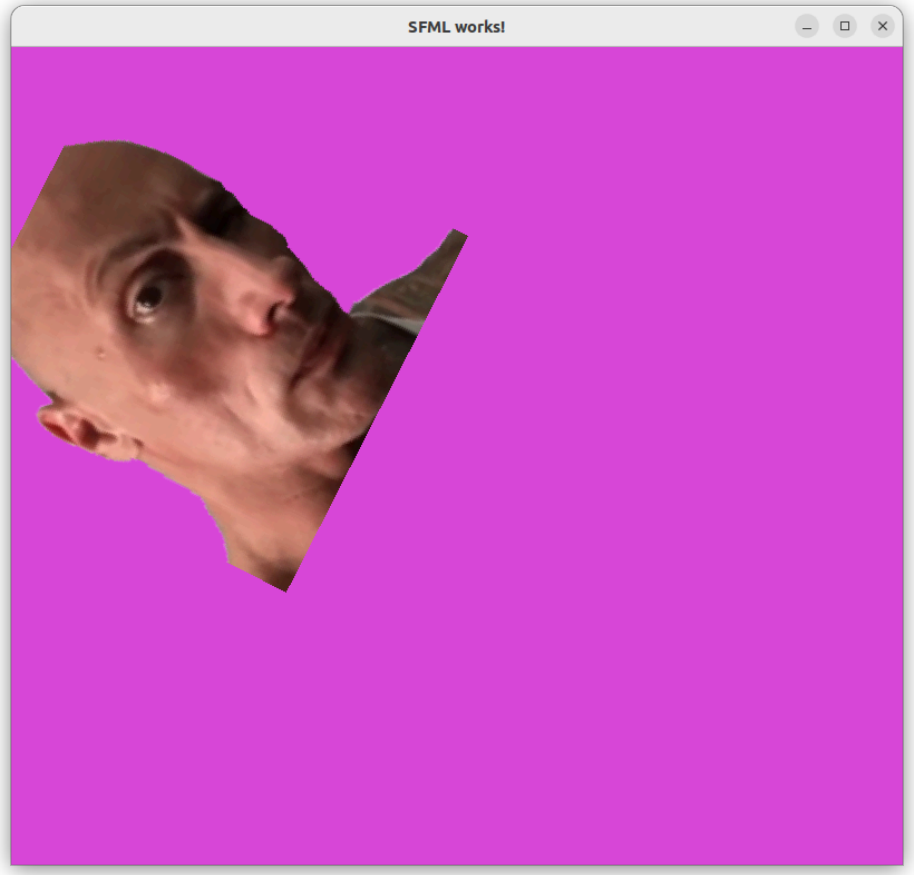
Next, to pair the texture loaded from the file to the sprite, I used the Sprites .setTexture(textureObj). Then to draw the Sprite to the window, I called the .draw(sprite) each iteration of the while loop. This fulfills the first requirement.

The next two requirements are to make the Sprite move and to respond to keystrokes. I combined these two steps to make the sprite move based on W, A, S, D. This was easier for me because I had experience coding basic games in C# using the Unity game engine. Getting access to the keyboard involved creating a sf::Keyboard object and using the simple syntax .isKeyPressed(key_constant). Then adding an if then to check if any of the W, A, S, D keys are pressed inside the while loop that refreshes the window, this allows me to change the position of the Sprite based on what key is pressed. The only issue with this solution is that each frame that the window refreshes it also checks for a key, this means when I'm holding down a key it will update the position every frame the key is pressed. This leads to extremely fast and uncontrollable movement. Some possible solutions to this is to decrease the amount of pixels of movement to something very low, the problem I foresaw with this solution is that the frame rate is variable based on many factors. So if the frame rate is low on one machine but not on another the functionality of the program would be drastically different, that is why instead I decided to limit the frame. Capping the frame rate at 20 ensures basic machines will have similar functionality; however, I will admit this solution is not something that would be sufficient for anything beyond the scope of a project like this. All in all, this comes together to fulfill the 2nd and 3rd requirements.

The final requirement is to add a unique feature of your choice. This involved experimenting with a couple features of the SFML. I found all Sprite movements are really easy to work with so I quickly added a small rotation in the while loop to the Sprite so that the Sprite is rotating in the window during the entire duration of the program. This was too easy so I decided to also figure out how to change the

color of the background. Using the .clear(sf::Color) syntax I changed the color of the background, I used simple R, G, B values that change every frame. The result is a cool color changing background.

The final product looks like this:




Pressing W, A, S, D moves the Sprite in that direction. The Sprite Rotates. The background cycles through many colors. Magnificent.

### 1.2 What I Learned

This assignment introduced me to C++ syntax which was very already familiar from my experience with C. It also introduced the use of libraries, specifically SFML. I gained a good grasp of the Window, Sprite, and Texture objects and am now confident in using them in different contexts.

### 1.3 Codebase

```
Makefile
```

```
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_ test_framework
4: # Your .hpp files
5: DEPS =
6: # Your compiled .o files
7: OBJECTS =
8: # The name of your program
9: PROGRAM = sfml-app
10:
11: .PHONY: all clean lint
12:
13: all: $(PROGRAM)
14:
15: # Wildcard recipe to make .o files from corresponding .cpp file 16:
%.o: %.cpp $(DEPS)
17: $(CC) $(CFLAGS) -c $<
18:
19: $(PROGRAM): main.o $(OBJECTS)
20: $(CC) $(CFLAGS) -o $@ $^ $(LIB)
21:
22: clean:
23: rm *.o $(PROGRAM)
24:
25: lint:
26: cpplint *.cpp *.hpp
27:
28:
```

main.cpp

```
1: // STEVEN MELLETT
2:
3: #include <SFML/Graphics.hpp>
4: #include <SFML/Window/Keyboard.hpp>
5:
6: int main() {
7: sf::RenderWindow window(sf::VideoMode(800, 800), "SFML works!"); 8:
window.setFramerateLimit(20);
9: sf::Texture the_rock_t;
10: the_rock_t.loadFromFile("./sprite.png");
11: sf::Sprite the_rock;
12: the_rock.setTexture(the_rock_t);
```

```
13: float x = 400;
14: float y = 400;
15: int r, g, b;
16: r = 0;
17: g = 55;
18: b = 0;
19: the_rock.setPosition(x, y);
20: float middle_R = 200;
21: the_rock.setOrigin(middle_R, middle_R);
22: float r_angle = 0;
23: sf::Keyboard K;
24: while ( window.isOpen() ) {
25: sf::Event event;
26: while ( window.pollEvent(event) ) {
27: if ( event.type == sf::Event::Closed ) {
28: window.close();
29: }
30: }
31: window.clear(sf::Color(r, g, b, 255));
32: window.draw(the_rock);
33: if ( K.isKeyPressed(sf::Keyboard::Key::A) ) {
34: the_rock.move(-5, 0);
35: }
36: if ( K.isKeyPressed(sf::Keyboard::Key::S) ) {
37: the_rock.move(0, 5);
38: }
39: if ( K.isKeyPressed(sf::Keyboard::Key::D) ) {
40: the_rock.move(5, 0);
41: }
42: if ( K.isKeyPressed(sf::Keyboard::Key::W) ) {
43: the_rock.move(0, -5);
44: }
45: if ( r < 255 ) {
46: r += 1;
47: } else {
48: r -= 255;
49: }
50: if ( g < 255 ) {
51: g += 1;
52: } else {
53: g -= 200;
54: }
55: if ( b < 255 ) {
56: b += 1;
57: } else {
58: b -= 55;
59: }
```

```
60: r_angle += 3;
61: the_rock.setRotation(r_angle);
62: window.display();
63: }
64: return 0;
65: }
```

# 2 PS1: Linear Feedback Shift Register and Image Encoding

PS1a: Linear Feedback Shift Register

## 2.1 Discussion part 1

Part one of this project is to create a pseudo random number generator through the simulation of a linear feedback shift register. In my program, this is accomplished by using a specific data structure provided by the C++ standard library called a Bitset. This data structure is an integral part in my solution for simulating a LFSR as it provides many useful functions and is a customizable size. Compared to using just a standard **int**, using the operator[] is a way better syntax to access specific bits than having to do a lot of convoluted binary operations to change a single bit. Moreover, the bitset's operator[] accesses the right-most-least-significant bit at [0] so I wouldn't have to remember to access the end rather than the beginning when implementing the LFSR.

In short, a linear feedback shift register is an initial seed of bits that are input to a linear function, which is usually a XOR between two or more bits (called taps), then shifts the set of bits once to the left and replaces the least-significant bit with the output of the function. The result is a seemingly random stream of new bits that you can append to the set, while the most-significant bit is dropped off.

A problem with this generator is that it will eventually wrap around to the starting set of bits and you will get a repetition of numbers. At the speed that computers operate at, it runs out of new random numbers in a very small amount of time. The amount and quality it can generate is very dependent on a few factors. First, the

initial seed of bits, some specific seeds have better performance than others, for example a seed of bits with mostly 0's will perform worse than a seed with an even amount of 0's and 1's. That also depends on the size of the seed, the larger the seed is the more values it can store as well as having more places to put a tap. Finally, the number of taps also affects number generation, in this project it is required to implement a Fibonacci LFSR that has 3 taps.

The goal of this project is to implement this API:

```
namespace PhotoMagic {
class FibLFSR {
public:
  // constructor to create LFSR with the given initial seed and tap
  FibLFSR(string seed);
  // simulate one step and return the new bit as 0 or 1
  int step();
  // simulate k steps and return k-bit integer
     int generate(int k);
private:
  // Any fields that you need
};
std::ostream& operator<<(std::ostream&, const FibLFSR& lfsr);

}
```

I defined the Bitset that's 16 bits in the private field.

The constructor was fast and simple to implement. I initialized a bitset of 16 bits of all 0's and then |= them to the string seed. This makes sure that the seed passed to the constructor will be converted to 16 bits while also maintaining most of its values.

The step function also didn't introduce any problems, set a temp integer variable equal to the XOR value of bits 15, 13, 12, 10. Then left shift the bitset stored in the class and then set the last bit equal to the temp variable and also return temp.

The generate function is a simple for loop with some clever arithmetic that was outlined in the project PDF.

Other than the few simple helper functions I added, all that overloading the operator<< involved was simply passing the bitset because it was already overloaded in the standard library.

After I got it working it was required I write 4 test functions using the new boost testing suite framework. My code passed the first example test so I used those as an outline for my new test. One of which makes sure that eventually the seed provided, after so many steps, will return to its original seed.

In this part of the project there is no specified main behavior so it just returns 0.

```
smellett@Ubuntu:~/CompIV/PS1/ps1a$ ./test
Running 6 test cases...
0000000000010001 Init with string less than 16 bits:
 Adds initialized seed value of 10001 to end of 16 bits 0 bitset
1010000000000000 Inits wiht string more than 16 bits:
 Drops the rightmost bits past 16 bits
Return to origin seed:
 Calls step function until it get back to the beggining seed

*** No errors detected
```

## 2.2 What I Learned part 1

In this assignment one of the more important things I learned was how to read documentation. Before this project I had no experience using the Bitset data structure but I was able to implement a working program with it by using the features described in its documentation. I am now more confident in being able to understand and decipher the documentation specifically on cppreference.com. Another part of this assignment I struggled with was the namespace. When I defined the original implementation my code would not compile because of naming and scope issues. I had to make sure the function definitions were also inside the namespace as well as making sure to have a **using** statement at the top of my main.cpp.

## 2.3 Codebase part 2

```
Makefile
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
```

```
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
test_framework
4: OBJ = PhotoMagic.o FibLFSR.o Test.o
5:
6: .PHONY: all clean lint PhotoMagic.a
7:
8: all: PhotoMagic test PhotoMagic.a
9:
10: PhotoMagic : PhotoMagic.o FibLFSR.o
11: $(CC) $(CFLAGS) -o lfsr PhotoMagic.o FibLFSR.o $(LIB)
12:
13: test : test.o FibLFSR.hpp
14: $(CC) $(CFLAGS) -o test test.o FibLFSR.o $(LIB)
15:
16: PhotoMagic.o : PhotoMagic.cpp FibLFSR.hpp
17: $(CC) $(CFLAGS) -c PhotoMagic.cpp
18:
19: FibLFSR.o : FibLFSR.cpp FibLFSR.hpp
20: $(CC) $(CFLAGS) -c FibLFSR.cpp
21:
22: test.o : test.cpp FibLFSR.hpp
23: $(CC) $(CFLAGS) -c test.cpp
24:
25: clean:
26: rm *.o lfsr test
27:
28: lint:
29: cpplint *.cpp *.hpp
30:
31: PhotoMagic.a:
32: ar rcs PhotoMagic.a FibLFSR.o PhotoMagic.o
33:
34:
```

```
PhotoMagic.cpp
1: // Copyright 2024
2: // By Steven Mellett
3: // main.cpp for PS1a
4: // updated 1/22/2024
5: #include <string>
6: #include <iostream>
```

```
7: #include <cstring>
8: #include <bitset>
9: #include "FibLFSR.hpp"
10:
11: using std::size_t;
12: using std::cout;
13: using std::endl;
14: using PhotoMagic::FibLFSR;
15:
16: int main(void) {
17: return 0;
18: }
19:
```

FibLFSR.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // FibLFSR.hpp for PS1a
4: // updated 1/22/2024
5: #ifndef FIBLFSR_HPP
6: #define FIBLFSR_HPP
7:
8: #include <iostream>
9: #include <cmath>
10: #include <cstdint>
11: #include <bitset>
12: #include <string>
13:
14:
15: using std::cout;
16: using std::bitset;
17: using std::string;
18: using std::ostream;
19:
20: namespace PhotoMagic {
21: class FibLFSR {
22: public:
23: explicit FibLFSR(string seed);
24: int step();
25: void left_shift(int amount);
26: int value();
27: int generate(int k);
```

```
28: bool operator==(const FibLFSR& r) const {
29: if (this->bits == r.bits) {
30: return true;
31: } return false;
32: }
33: private:
34: bitset<16> bits;
35: friend ostream& operator<<(ostream& out, const FibLFSR& lfsr);
36: };
37: } // namespace PhotoMagic
38: #endif // FIBLFSR_HPP
```

FibLFSR.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // FibLFSR.cpp for PS1a
4: // updated 1/22/2024
5: #include <iostream>
6: #include <cmath>
7: #include <cstdint>
8: #include <bitset>
9: #include <string>
10: #include "FibLFSR.hpp"
11:
12: using std::cout;
13: using std::bitset;
14: using std::string;
15: using std::ostream;
16: using PhotoMagic::FibLFSR;
17:
18: namespace PhotoMagic {
19: ostream& operator<<(ostream& out, const FibLFSR& lfsr) {
20: out << lfsr.bits;
21: return out;
22: }
23: } // namespace PhotoMagic
24: FibLFSR::FibLFSR(string seed) {
25: bits = bitset<16>("0000000000000000");
26: bits |= bitset<16>(seed);
27: }
28: void FibLFSR::left_shift(int amount) {
29: bitset<16> temp_bits;
```

```
30: temp_bits = bits << amount;
31: bits = temp_bits;
32: }
33:
34: int FibLFSR::step() {
35: bitset<16> temp_bits;
36: int temp;
37: temp = ((bits[15] ^ bits[13]) ^ bits[12]) ^ bits[10];
38: temp_bits = bits << 1;
39: bits = temp_bits;
40: if (temp == 1) {
41: bits[0] = 1;
42: }
43: return temp;
44: }
45: int FibLFSR::generate(int k) {
46: int temp = 0;
47:
48: for (int i = 0; i < k; i++) {
49: temp = (temp*2) + this->step();
50: }
51: return temp;
52: }
53: int FibLFSR::value() {
54: int temp = 0;
55: for (int i = 0; i < 16; i++) {
56: temp = (temp*2) + this->bits[i];
57: }
58: return temp;
59: }
```

test.cpp

```
1: // Copyright 2022
2: // By Dr. Rykalova
3: // Editted by Dr. Daly
4: // test.cpp for PS1a
5: // updated 1/8/2024 by Steven Mellett
6:
7: #include <iostream>
8: #include <string>
9:
10: #include "FibLFSR.hpp"
```

```
11:
12: #define BOOST_TEST_DYN_LINK
13: #define BOOST_TEST_MODULE Main
14: #include <boost/test/unit_test.hpp>
15:
16: using PhotoMagic::FibLFSR;
17:
18: BOOST_AUTO_TEST_CASE(testStepInstr) {
19: FibLFSR l("1011011000110110");
20: BOOST_REQUIRE_EQUAL(l.step(), 0);
21: BOOST_REQUIRE_EQUAL(l.step(), 0);
22: BOOST_REQUIRE_EQUAL(l.step(), 0);
23: BOOST_REQUIRE_EQUAL(l.step(), 1);
24: BOOST_REQUIRE_EQUAL(l.step(), 1);
25: BOOST_REQUIRE_EQUAL(l.step(), 0);
26: BOOST_REQUIRE_EQUAL(l.step(), 0);
27: BOOST_REQUIRE_EQUAL(l.step(), 1);
28: }
29:
30: BOOST_AUTO_TEST_CASE(testGenerateInstr) {
31: FibLFSR l("1011011000110110");
32: BOOST_REQUIRE_EQUAL(l.generate(9), 51);
33: }
34:
35: BOOST_AUTO_TEST_CASE(test_with_zero_Seed) {
36: FibLFSR l("0000000000000000");
37: BOOST_REQUIRE_EQUAL(l.step(), 0);
38: }
39:
40: BOOST_AUTO_TEST_CASE(init_with_string_less_than_16) {
41: FibLFSR l("10001");
42: std::cout << l << " Init with string less than 16 bits: " << std::endl;
43: std::cout << " Adds initialized seed value of 10001 to end of 16 bits 0
bi
tset" << std::endl;
44: BOOST_REQUIRE_EQUAL(l.step(), 0);
45: }
46:
47: BOOST_AUTO_TEST_CASE(init_with_string_more_than_16) {
48: FibLFSR l("1010000000000000001");
49: std::cout << l << " Inits wiht string more than 16 bits:" << std::endl;
50: std::cout << " Drops the rightmost bits past 16 bits" << std::endl;
51: BOOST_REQUIRE_EQUAL(l.step(), 0);
```

```
52: }
53:
54: BOOST_AUTO_TEST_CASE(returns_to_origin_seed) {
55: FibLFSR l("1010101010101010");
56: FibLFSR r("1010101010101010");
57: do {
58: l.step();
59: } while (!(l == r));
60: std::cout << "Return to origin seed: " << std::endl;

test.cpp Sun Jan 28 17:24:44 2024 2
61: std::cout << " Calls step function until it get back to the beggining
seed
" << std::endl;
62: BOOST_REQUIRE_EQUAL(l.step(), 0);
63: }
```



### 2.4 Discussion part 2

The goal of part 2 of this project is to use the LFSR I made in the previous part to encode an image with the random numbers generated.

The requirements for this part are as follows:

- Read three arguments from the command line: source image filename, output image filename, and FibLFSR seed.
- Use SFML to load the source image from disk and display it in its own window. Use your debugged FibLFSR class to encode (or decode) the image.
- Display the encoded/decoded image in its own window.
- Save the new image to disk.

In order to accomplish this I had to use the sf::Image object from the SFML specifically in tandem with the sf::Color object. The Image object stores useful data about the images size and color that I use in various ways in the project.

To encode the entire image I have to get the size using the .getSize() function that returns a sf::Vector2u. This makes the syntax to iterate through the whole image as simple as two nested for loops. When iterating through the image each pixel at position x, y has a sf::Color associated with it. I extract the R, G, B values by using temp variables and the syntax .r, .g, and .b from the Color object. Finally use the LFSR class to generate 3 8-bit “random” numbers then XOR each new random number with the R, G, B temp variables. This means I will have 3 random values I can assign the color of the pixel at x, y to.

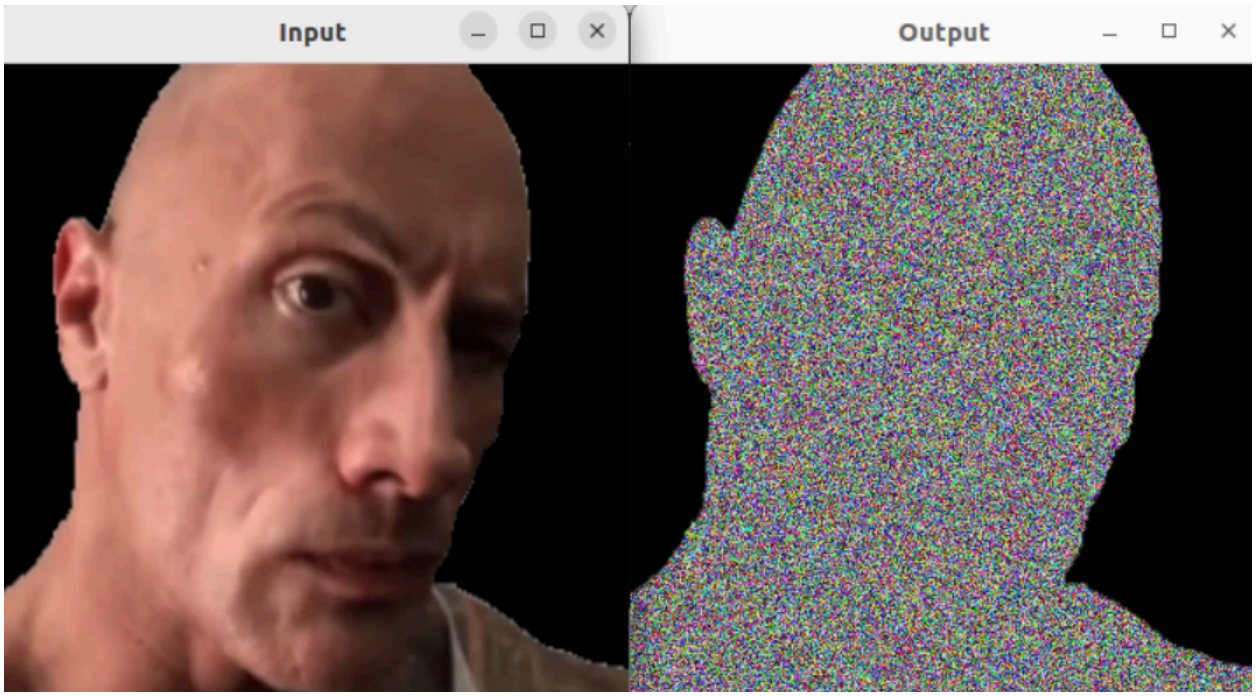
The behavior of this is simpler than trying to explain how it works. It takes in an image and randomizes the colors based on a seed. What’s cool about this is that you can use the same seed to unscramble the image. In my program I take input from stdin rather than the command line which I will discuss later.

So for example, if you use the .png image named “input.png”:

Then the command and input:

```
smellett@Ubuntu:~/CompIV/PS1/ps1b$ ./PhotoMagic
input.png
output.png
0100100101011110
```

I get the output:




After closing the windows, and .png is saved in the directory of the program. This also works in reverse, if I gave the program “output.png” first then the input.png image would be deciphered.

### 2.5 What I Learned part 2

In this part I had even more trouble with namespaces. It was difficult to get my program to find every function it needed to because they were in separate files but the same namespaces. I have since figured this out and have a better understanding but it is something that I still have to work on. At this point I’m very familiar with how SFML works and am confident in using it for more applications. One thing that I did not get right was accepting commands from the command line rather than stdin. This is something I do get in future projects but did not implement in this one.

### 2.6 Codebase part 2

Makefile

```
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
```

```
test_framework
4: OBJ = main.o FibLFSR.o test.o PhotoMagic.o
5:
6: .PHONY: all clean lint pdf PhotoMagic.a
7:
8: all: PhotoMagic test PhotoMagic.a
9:
10: PhotoMagic : main.o PhotoMagic.o FibLFSR.o
11: $(CC) $(CFLAGS) -o PhotoMagic main.o PhotoMagic.o FibLFSR.o $(LIB)
12:
13: test : test.o FibLFSR.hpp
14: $(CC) $(CFLAGS) -o test test.o FibLFSR.o $(LIB)
15:
16: main.o : main.cpp FibLFSR.hpp PhotoMagic.hpp
17: $(CC) $(CFLAGS) -c main.cpp
18:
19: FibLFSR.o : FibLFSR.cpp FibLFSR.hpp
20: $(CC) $(CFLAGS) -c FibLFSR.cpp
21:
22: PhotoMagic.o : PhotoMagic.cpp PhotoMagic.hpp
23: $(CC) $(CFLAGS) -c PhotoMagic.cpp
24:
25: test.o : test.cpp FibLFSR.hpp
26: $(CC) $(CFLAGS) -c test.cpp
27:
28: clean:
29: rm *.o PhotoMagic test
30:
31: lint:
32: cpplint *.cpp *.hpp
33:
34: pdf:
35: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
36:
37: PhotoMagic.a:
38: ar rcs PhotoMagic.a FibLFSR.o PhotoMagic.o
39:
40:
```

```
main.cpp
1: // Copyright 2024
2: // By Steven Mellett
```

```
3: // main.cpp for PS1b
4: // updated 1/22/2024
5: #include <string>
6: #include <iostream>
7: #include <cstring>
8: #include <bitset>
9: #include <SFML/System.hpp>
10: #include <SFML/Window.hpp>
11: #include <SFML/Graphics.hpp>
12: #include "FibLFSR.hpp"
13: #include "PhotoMagic.hpp"
14:
15: using std::size_t;
16: using std::cout;
17: using std::endl;
18: using std::cin;
19: using std::string;
20: using PhotoMagic::FibLFSR;
21: using PhotoMagic::transform;
22:
23: int main(void) {
24: sf::Image image;
25: sf::Image e_image;
26: string i_file;
27: string o_file;
28: string seed;
29: FibLFSR* p_e_key = nullptr;
30: cin >> i_file;
31: cin >> o_file;
32: cin >> seed;
33: FibLFSR e_key(seed);
34: p_e_key = &e_key;
35:
36: if (!image.loadFromFile(i_file)) {
37: cout << "Error returning -1: cannot open input file" << endl;
38: }
39: if (!e_image.loadFromFile(i_file)) {
40: cout << "Error returning -1: cannot open input file" << endl;
41: }
42: transform(e_image, p_e_key);
43: sf::RenderWindow window1(sf::VideoMode(image.getSize().x,
image.getSize().
y), "Input");
```

```
44: sf::RenderWindow window2(sf::VideoMode(image.getSize().x,
image.getSize().
y), "Output");
45:
46: sf::Texture texture1;
47: sf::Texture texture2;
48: texture1.loadFromImage(image);
49: texture2.loadFromImage(e_image);
50: sf::Sprite sprite1;
51: sf::Sprite sprite2;
52: sprite1.setTexture(texture1);
53: sprite2.setTexture(texture2);
54: while (window1.isOpen() && window2.isOpen()) {
55: sf::Event event;
56: while (window1.pollEvent(event)) {
57: if (event.type == sf::Event::Closed) {
58: window1.close();
59: }
60: }
61: while (window2.pollEvent(event)) {
62: if (event.type == sf::Event::Closed) {
63: window2.close();
64: }
65: }
66: window1.clear();
67: window1.draw(sprite1);
68: window1.display();
69: window2.clear();
70: window2.draw(sprite2);
71: window2.display();
72: }
73: if (!e_image.saveToFile(o_file)) {
74: cout << "Error saving image returning -1" << endl;
75: return -1;
76: }
77: return 0;
78: }
79:
```

PhotoMagic.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
```

```
3: // PhotoMagic.hpp for ps1b
4: // updated 2/4/2024
5: #ifndef PHOTO_MAGIC_HPP
6: #define PHOTO_MAGIC_HPP
7: #include <SFML/System.hpp>
8: #include <SFML/Window.hpp>
9: #include <SFML/Graphics.hpp>
10:
11: namespace PhotoMagic {
12: void transform(sf::Image& image, FibLFSR* key);
13: }
14: #endif // PHOTO_MAGIC_HPP
```

PhotoMagic.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // PhotoMagic.cpp for PS1b
4: // updated 1/22/2024
5: #include <string>
6: #include <iostream>
7: #include <cstring>
8: #include <bitset>
9: #include <SFML/System.hpp>
10: #include <SFML/Window.hpp>
11: #include <SFML/Graphics.hpp>
12: #include "FibLFSR.hpp"
13: #include "PhotoMagic.hpp"
14:
15: using std::size_t;
16: using std::cout;
17: using std::endl;
18: using std::cin;
19: using std::string;
20: using PhotoMagic::FibLFSR;
21: using PhotoMagic::transform;
22:
23: namespace PhotoMagic {
24: void transform(sf::Image& image, FibLFSR* key) {
25: sf::Color color;
26: sf::Vector2u i_size;
27: int r;
28: int g;
```

```
29: int b;
30: int num_xor1;
31: int num_xor2;
32: int num_xor3;
33: i_size = image.getSize();
34: for (unsigned int x = 0; x < i_size.x; x++) {
35: for (unsigned int y = 0; y < i_size.y; y++) {
36: color = image.getPixel(x, y);
37: r = color.r;
38: g = color.g;
39: b = color.b;
40: num_xor1 = key->generate(8);
41: num_xor2 = key->generate(8);
42: num_xor3 = key->generate(8);
43: r = r^num_xor1;
44: g = g^num_xor2;
45: b = b^num_xor3;
46: color.r = r;
47: color.g = g;
48: color.b = b;
49: image.setPixel(x, y, color);
50: }
51: }
52: }
53: } // namespace PhotoMagic
```

FibLFSR.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // FibLFSR.hpp for PS1a
4: // updated 1/22/2024
5: #ifndef FIBLFSR_HPP
6: #define FIBLFSR_HPP
7:
8: #include <iostream>
9: #include <cmath>
10: #include <cstdint>
11: #include <bitset>
12: #include <string>
13:
14:
15: using std::cout;
```

```
16: using std::bitset;
17: using std::string;
18: using std::ostream;
19:
20: namespace PhotoMagic {
21: class FibLFSR {
22: public:
23: explicit FibLFSR(string seed);
24: int step();
25: void left_shift(int amount);
26: int value();
27: int generate(int k);
28: friend bool operator==(const FibLFSR& l, const FibLFSR& r) {
29: if (l.bits == r.bits) {
30: return true;
31: } return false;
32: }
33: private:
34: bitset<16> bits;
35: friend ostream& operator<<(ostream& out, const FibLFSR& lfsr);
36: };
37: } // namespace PhotoMagic
38: #endif // FIBLFSR_HPP
```

FibLFSR.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // FibLFSR.cpp for PS1a
4: // updated 1/22/2024
5: #include <iostream>
6: #include <cmath>
7: #include <cstdint>
8: #include <bitset>
9: #include <string>
10: #include "FibLFSR.hpp"
11:
12: using std::cout;
13: using std::bitset;
14: using std::string;
15: using std::ostream;
16: using PhotoMagic::FibLFSR;
17:
```

```
18: namespace PhotoMagic {
19: ostream& operator<<(ostream& out, const FibLFSR& lfsr) {
20: out << lfsr.bits;
21: return out;
22: }
23: } // namespace PhotoMagic
24: FibLFSR::FibLFSR(string seed) {
25: bits = bitset<16>("0000000000000000");
26: bits |= bitset<16>(seed);
27: }
28: void FibLFSR::left_shift(int amount) {
29: bitset<16> temp_bits;
30: temp_bits = bits << amount;
31: bits = temp_bits;
32: }
33:
34: int FibLFSR::step() {
35: bitset<16> temp_bits;
36: int temp;
37: temp = ((bits[15] ^ bits[13]) ^ bits[12]) ^ bits[10];
38: temp_bits = bits << 1;
39: bits = temp_bits;
40: if (temp == 1) {
41: bits[0] = 1;
42: }
43: return temp;
44: }
45: int FibLFSR::generate(int k) {
46: int temp = 0;
47:
48: for (int i = 0; i < k; i++) {
49: temp = (temp*2) + this->step();
50: }
51: return temp;
52: }
53: int FibLFSR::value() {
54: int temp = 0;
55: for (int i = 0; i < 16; i++) {
56: temp = (temp*2) + this->bits[i];
57: }
58: return temp;
59: }
```

Test.cpp

```cpp
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly
// test.cpp for PS1a
// updated 1/8/2024

#include <iostream>
#include <string>

#include "FibLFSR.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

using PhotoMagic::FibLFSR;

BOOST_AUTO_TEST_CASE(testStepInstr) {
FibLFSR l("1011011000110110");
BOOST_REQUIRE_EQUAL(l.step(), 0);
BOOST_REQUIRE_EQUAL(l.step(), 0);
BOOST_REQUIRE_EQUAL(l.step(), 0);
BOOST_REQUIRE_EQUAL(l.step(), 1);
BOOST_REQUIRE_EQUAL(l.step(), 1);
BOOST_REQUIRE_EQUAL(l.step(), 0);
BOOST_REQUIRE_EQUAL(l.step(), 0);
BOOST_REQUIRE_EQUAL(l.step(), 1);
}

BOOST_AUTO_TEST_CASE(testGenerateInstr) {
FibLFSR l("1011011000110110");
BOOST_REQUIRE_EQUAL(l.generate(9), 51);
}

BOOST_AUTO_TEST_CASE(test_with_zero_Seed) {
FibLFSR l("0000000000000000");
BOOST_REQUIRE_EQUAL(l.step(), 0);
}

BOOST_AUTO_TEST_CASE(init_with_string_less_than_16) {
FibLFSR l("10001");
std::cout << l << " Init with string less than 16 bits: " << std::endl;
```

```
43: std::cout << " Adds initialized seed value of 10001 to end of 16 bits 0
bi
tset" << std::endl;
44: BOOST_REQUIRE_EQUAL(l.step(), 0);
45: }
46:
47: BOOST_AUTO_TEST_CASE(init_with_string_more_than_16) {
48: FibLFSR l("10100000000000001");
49: std::cout << l << " Inits wiht string more than 16 bits:" << std::endl;
50: std::cout << " Drops the rightmost bits past 16 bits" << std::endl;
51: BOOST_REQUIRE_EQUAL(l.step(), 0);
52: }
53:
54: BOOST_AUTO_TEST_CASE(returns_to_origin_seed) {
55: FibLFSR l("1010101010101010");
56: FibLFSR r("1010101010101010");
57: do {
58: l.step();
59: } while (!(l == r));
60: std::cout << "Return to origin seed: " << std::endl;

test.cpp Sun Jan 28 17:24:44 2024 2
61: std::cout << " Calls step function until it get back to the beggining
seed
" << std::endl;
62: BOOST_REQUIRE_EQUAL(l.step(), 0);
63: }
```

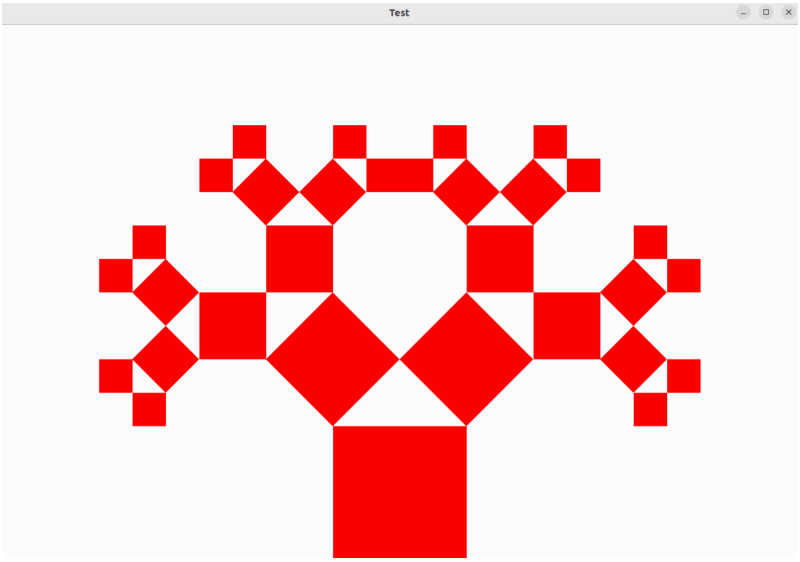
# 3 PS2: Pythagoras Tree

## 3.1 Discussion

The main point of this project is to write a recursive function that draws a Pythagoras Tree to an SFML window. First getting going on this project was very difficult as SFML square and rectangle objects are difficult to align when the origin of the window is at the top left corner. I would find that I could calculate one branch of the tree but those calculations would not apply to the other side. In the end I figured out that you need a separate recursive branch with different calculations on both sides.

I found myself having the same issues from the last project with taking commands from the command line so I opted to take input from stdin rather than not having a working program.

So in the end the output when you specify 200 size and 5 depth:




The main point of this project is the recursive function PTree, I chose to offload the resusion to a new helper function so I could pass in parameters instead of storing them somewhere in the class. I found this to make the class and function PTree simpler.

One problem with the recursion is that when the depth is too high, the program slows my machine down considerably. This is because I chose to call the draw function from inside the sf::Window loop. So every frame it re-recurses and recalculates all of that math. When using a virtual Ubuntu machine I can get to about depth 7 or 8 before the whole system becomes unresponsive. The solution to this would be moving it outside the while loop and only displaying one frame; however, I found it involves some new parameters for the sf::Window object that I decided I didn't have the time to learn and implement.

### 3.2 What I Learned

Before this project I was familiar with recursion and how/when to use it from Comp I & II so I was confident this project would be easy for me. However, I

found that it was a lot harder to keep track of the math aspect and work with the SFML. I now understand how Shape objects work and how important  it is to keep track of where the origin is considered on that shape but still struggle with actually using them in code.

Specifically I learned a way to keep track of variables like depth through passing them into the parameters of the next function call. It was way easier to pass in an integer and subtract one to keep track of the current position in the recursion.

### 3.3 Codebase

```
Makefile
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
test_framework
4: OBJ = main.o
5:
6: .PHONY: all PTree clean pdf lint
7:
8: all: PTree
9:
10: PTree : main.o PTree.o
11: $(CC) $(CFLAGS) -o PTree main.o PTree.o $(LIB)
12:
13: main.o : main.cpp PTree.hpp
14: $(CC) $(CFLAGS) -c main.cpp
15:
16: pTree.o: pTree.cpp PTree.hpp
17: $(CC) $(CFLAGS) -c PTree.cpp
18:
19: clean:
20: rm *.o PTree
21:
22: pdf:
23: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
24: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
25: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
26:
27: lint:
```

```
28: cpplint *.cpp *.hpp
29:
30:
```

main.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // main.cpp for ps2
4: // updated 1/24/2024
5: #include <iostream>
6: #include "PTree.hpp"
7:
8: using std::cout;
9: using std::endl;
10: using std::cin;
11:
12: int main(int argc, char* argv[]) {
13: double side;
14: int depth;
15: cout << "Please enter side length: ";
16: cin >> side;
17: cout << endl << "Please enter depth: ";
18: cin >> depth;
19: pTree(side, depth);
20: return 0;
21: }
```

PTree.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // pTree.hpp for ps2
4: // updated 2/12/2024
5: #ifndef PTREE_HPP
6:
7: #define PTREE_HPP
8: #include <iostream>
9: #include <SFML/Graphics/RectangleShape.hpp>
10: #include <SFML/Graphics.hpp>
11:
12: class sq_tree {
13: public:
```

```
14: explicit sq_tree(double side, int depth);
15: void draw(sf::RenderWindow& window, double side, int depth);
16: void draw_help(sf::RectangleShape& sq_b,
17: const int count, float rot, sf::RenderWindow& window);
18: private:
19: sf::RectangleShape sq;
20: double _side;
21: int _depth;
22: };
23: void pTree(double side, int depth);
24: // void recurse(double side, int depth,
25: // sf::RenderWindow& window, int x, int y, float rot,
26: // sf::RectangleShape sq, int it);
27:
28: #endif
```

PTree.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // pTree.cpp for ps2
4: // updated 2/12/2024
5: #include "PTree.hpp"
6: #include <cmath>
7: #include <iostream>
8: #include <SFML/Graphics/RectangleShape.hpp>
9: #include <SFML/Graphics.hpp>
10:
11:
12: sq_tree::sq_tree(double side, int depth) {
13: _side = side;
14: _depth = depth;
15: }
16: void sq_tree::draw(sf::RenderWindow& window, double side, int depth) {
17: sf::RectangleShape sqa(sf::Vector2f(side, side));
18: sqa.setOrigin(sqa.getSize()/ 2.f);
19: sqa.setPosition(window.getSize().x / 2.f, window.getSize().y -
(side/2.f))
;
20: sqa.setFillColor(sf::Color::Red);
21: // std::cout << " in draw" << std:: endl;
22: draw_help(sqa, depth, 0, window);
23: }
```

```
24:
25: void sq_tree::draw_help(sf::RectangleShape& sq_b,
26: const int count, float rot, sf::RenderWindow& windo
w) {
27: sf::Transform position = sq_b.getTransform();
28: // std::cout << "count is :: " << count << std::endl;
29: if (count <= 0) {
30: return;
31: } else {
32: window.draw(sq_b);
33: // std::cout << "in draw_help " << std::endl;
34: sf::RectangleShape new_left = sq_b;
35: new_left.setSize(sq_b.getSize() * static_cast<float>(sqrt(2.f) / 2.f));
36: new_left.setOrigin(0, new_left.getSize().y);
37: new_left.setPosition(position.transformPoint(sf::Vector2f(0, 0)));
38: new_left.rotate(-45);
39: draw_help(new_left, count-1, 0, window);
40: sf::RectangleShape new_right = sq_b;
41: new_right.setSize(sq_b.getSize() * static_cast<float>(sqrt(2.f) /
2.f));
42: new_right.setOrigin(new_right.getSize());
43:
new_right.setPosition(position.transformPoint(sf::Vector2f(sq_b.getSize(
).x, 0)));
44: new_right.rotate(45);
45: draw_help(new_right, count - 1, 0, window);
46: }
47: }
48: void pTree(double side, int depth) {
49: sf::RenderWindow window(sf::VideoMode(6 * side, 4 * side), "Test");
50: sf::RenderWindow& rWindow = window;
51: sq_tree sq(side, depth);
52: if (1) {
53: while ( window.isOpen() ) {
54: for (sf::Event event; window.pollEvent(event);) {
55: if ( event.type == sf::Event::Closed ) {
56: window.close();
57: }
58: }

PTree.cpp Tue Apr 23 15:22:16 2024 2
59: window.clear(sf::Color::White);
60: // std::cout << "kad;skjdlka" << std::endl;
```

```
61: sq.draw(rWindow, side, depth);
62: window.display();
63: }
64: }
65: }
```

# 4 PS3: Sokoban UI & Game

PS3a: Sokoban UI

### 4.1 Discussion part 1

This was one of the quicker projects that I worked on. I do a lot of my own projects outside of Comp IV on game development so I am familiar with some concepts like the tile map that I use in this project.

The hard part about this project is that you read in levels and the levels have changing sizes and layouts so my program is going to have to be robust enough to handle this. I will talk about some of the problems I had with this later on in this discussion.

The levels come in this format:

```
10 10
##########
#....a...#
#....A...#
#........#
#...##...#
#...##...#
#..@..A..#
#.......a#
#........#
##########
```

Each character represents another texture in the game with different functionality.

The first data we get is the level size. This helps quickly initialize how much memory and what window size to use. In this case, rather than make things too

complex, I decided to use a normal 64x64 tile size for each without scaling. So a 10x10 level's window would be 640x640.

That's the easy part though, I had to decide how to represent the actual level in a way the data would be easily accessible. I chose to represent the game as one long vector, this means that I would have to do some math when calculating rows and columns but I found this way to represent the data worked fine.

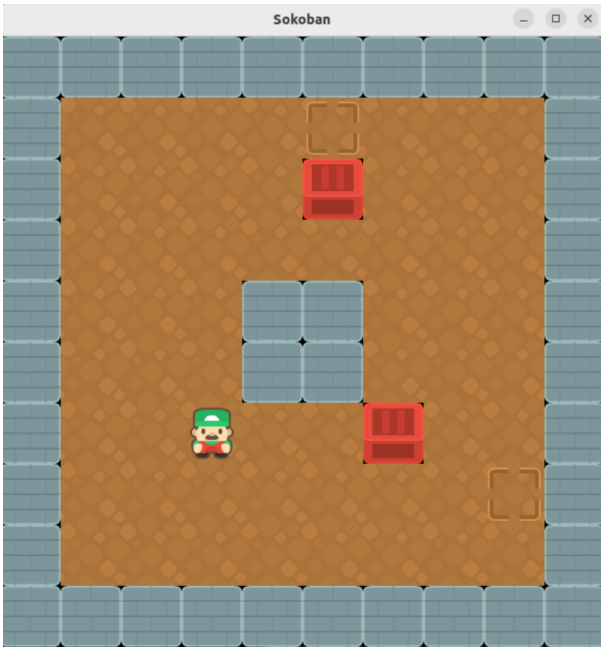
I did have some trouble reading the file into a string and using that data to make a new vector with sprite objects that would hold the correct texture. If I were to do this over again I would make the level reading into a string more robust because I run into problems with this when using different machines on certain levels later on.

One of the more exciting parts about this project was actually drawing the vector of sprites that I had created to the window. We were required to inherit from the sf::Drawable class so you could overload the virtual draw function. This was a fun way to explore the SFML forums and documentation as there were a lot of resources on the topic. This allows me to use the .draw(object) function of the window object. Other than that the way I was able to draw a single dimensional vector into a 2 dimensional window was using a second helper vector that stored every point in pixel space that corresponded to a tile in the window and assign each sprite with a new (x, y) pixel value based on the calculated points. So basically it acts as a tile map with position stored across a vector. This makes assigning the sprites location very easy and in the next part moving the sprites just as easy.

I finally did learn how to accept command line arguments and I will also talk about that later.

With the command:

Where level1.txt is the text document shown above, the output of this program is:




Going back to what I said earlier, one main issue with this program has something to do with reading the level file into a string at the beginning. Using a Ubuntu virtual machine I can compile and run my program with a variety of levels with no issues. However, for some reason on other machines it does not run and gives an out of bounds error. I realized my program had this problem too late and was not able to sort it out. If I were going to go back and fix it I would make sure reading into a string at the beginning is not the issue, rather than using the size of the level to tell how much to read I would check for EOF constant instead.

### 4.2 What I Learned part 1

I had to use and become comfortable with virtual functions from parent classes. Which was something new I had not implemented completely before. Overloading the specific virtual draw function is something I am now more confident in being able to do.

Also in this project I finally figured out how to take command line arguments, it was something I thought was harder than it was actually at first. The parameters of the main function are simply the amount of arguments passed as well as an array of what was passed. So you can check if arguments have been passed as well as what

the arguments actually are. I was eventually able to implement this in this project's main function.

### 4.3 Codebase part 1

Makefile

```
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
test_framework
4: OBJ = main.o
5:
6: .PHONY: all Sokoban clean pdf lint Sokoban.a
7:
8: all: Sokoban Sokoban.a
9:
10: Sokoban : main.o Sokoban.o
11: $(CC) $(CFLAGS) -o Sokoban main.o Sokoban.o $(LIB)
12:
13: main.o : main.cpp Sokoban.hpp
14: $(CC) $(CFLAGS) -c main.cpp
15:
16: Sokoban.o : Sokoban.cpp Sokoban.hpp
17: $(CC) $(CFLAGS) -c Sokoban.cpp
18:
19: clean:
20: rm *.o *.a Sokoban
21:
22: lint:
23: cpplint *.cpp *.hpp
24:
25: pdf:
26: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
27: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
28: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
29:
30: Sokoban.a:
31: ar rcs Sokoban.a Sokoban.o
32:
33:
```

main.cpp

```
1: // Copyright 2024 <Copyright Steven Mellett>
2: #include <iostream>
3: #include <fstream>
4: #include <string>
5: #include <SFML/Graphics.hpp>
6: #include "Sokoban.hpp"
7:
8: using std::ifstream;
9: using SB::Sokoban;
10:
11: int main(int argc, char* argv[]) {
12: std::string FILE_NAME;
13: if ( argc <= 1 ) {
14: FILE_NAME = "level1.txt";
15: std::cout << "Please enter level name: ";
16: std::cin >> FILE_NAME;
17: } else {
18: FILE_NAME = argv[1];
19: }
20: // std::cout << "Please enter level name: ";
21: // std::cin >> FILE_NAME;
22: std::cout << std::endl;
23: Sokoban game;
24: ifstream level_file(FILE_NAME);
25: if ( level_file.is_open() ) {
26: level_file >> game;
27: level_file.close();
28: } else {
29: std::cout << "Error opening file, or file doesn't exist" << std::endl;
30: return 1;
31: }
32:
33: sf::RenderWindow window(sf::VideoMode(game.width()*64,
game.height()*64, 6
4), "Sokoban");
34: sf::Event event;
35: while ( window.isOpen() ) {
36: while ( window.pollEvent(event) ) {
37: if ( event.type == sf::Event::Closed ) {
38: window.close();
39: }
40: }
```

```
41: window.clear();
42: window.draw(game);
43: window.display();
44: }
45: return 0;
46: }
```

Sokoban.hpp

```
1: // Copyright 2024 <Copyright Steven Mellett>
2: #ifndef SOKOBAN_HPP
3: #define SOKOBAN_HPP
4: #include <iostream>
5: #include <fstream>
6: #include <string>
7: #include <vector>
8: #include <SFML/Graphics.hpp>
9: #include <SFML/Graphics/Drawable.hpp>
10: #include <SFML/Graphics/Texture.hpp>
11: #include <SFML/Graphics/Sprite.hpp>
12:
13: using std::ifstream;
14: using std::string;
15:
16: namespace SB {
17: enum Direction { Up, Down, Left, Right };
18: class Sokoban : public sf::Drawable {
19: public:
20: Sokoban();
21: explicit Sokoban(ifstream& level);
22: int width() const { return y; }
23: int height() const { return x; }
24: void Load_level(ifstream& level);
25: sf::Vector2i playerLoc() const { return
sf::Vector2i(static_cast<int>(play
er.getPosition().x), static_cast<int>(player.getPosition().y)); }
26: void movePlayer(Direction dir) { return; }
27: int isWon() { return 0; }
28: friend ifstream& operator>>(ifstream& file, Sokoban& game);
29: // friend istream& operator>>(ifstream level_file, Sokoban& game);
30: private:
31: void Load_textures();
32: void Load_sprites();
```

```
33: void Calculate_points();
34: void move_sprites();
35: int player_pos;
36: int x = 0;
37: int y = 0;
38: string game_data;
39: std::vector<sf::Sprite> game_map;
40: sf::Sprite player;
41: sf::Texture block;
42: sf::Texture wall;
43: sf::Texture ground;
44: sf::Texture goal;
45: std::vector<sf::Texture> player_textures;
46: std::vector<sf::Vector2f> draw_points;
47:
48: protected:
49: virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const
{
50: for ( int i = 0; i < (x * y); i++ ) {
51: // sf::Sprite& sprites = game_map.at(i);
52: // game_map.at(i).setPosition(draw_points.at(i));
53: target.draw(game_map.at(i), states);
54: // std::cout << game_map.at(i).getOrigin().x << std::endl;
55: }
56: target.draw(player, states);
57: }
58: };
59: } // namespace SB
60: #endif
```

Sokoban.cpp

```
1: // Copyright 2024 <Copyright Steven Mellett>
2: #include "Sokoban.hpp"
3: #include <string>
4: #include <fstream>
5: #include <iostream>
6: #include <cctype>
7: #include <vector>
8:
9: using std::ifstream;
10: using std::string;
```

```
11: using std::cout;
12: using std::vector;
13:
14: namespace SB {
15: Sokoban::Sokoban() {
16: this->Load_textures();
17: // this->Load_sprites();
18: // this->Calculate_points(); // moved to end of level;
19: // this->move_sprites(); moved to end of level;
20: }
21: Sokoban::Sokoban(ifstream& level) {
22: this->Load_level(level);
23: this->Load_textures();
24: // this->Load_sprites();
25: // this->Calculate_points();
26: // this->move_sprites();
27: }
28: void Sokoban::Load_level(ifstream& level) {
29: cout << "Loading Level..." << std::endl;
30: string level_string;
31: // vector<char> level_vector;
32: char array_c[1000];
33: level.readsome(array_c, 1000);
34: // cout << array_c << std::endl;
35: char c = 'c';
36: char b = 'b';
37: int w = 0;
38: int i = 0;
39: c = array_c[i];
40: i++;
41: while (!isspace(c)) {
42: x *= 10;
43: int temp = c - '0';
44: x += temp;
45: c = array_c[i];
46: i++;
47: }
48: c = array_c[i];
49: i++;
50: while (!isspace(c)) {
51: y *= 10;
52: int temp = c - '0';
53: y += temp;
```

```
54: c = array_c[i];
55: i++;
56: }
57: // cout << x << ' ' << y << std::endl; works
58: // cout << array_c << std::endl;
59: // level >> level_vector;//////////////////////////////////////////////
60: // level >> game_data;
61: while (b != '#') {
62: b = array_c[w];
63: // cout << array_c[w];
64: w++;
65: }
66: for ( int p = 0; p < (x * y); p++ ) {
67: if ( !isspace(array_c[w]) ) {
68: game_data.push_back(array_c[w]);
69: // array_c[w];
70: } else {
71: p--;
72: }
73: w++;
74: }
75: // cout << game_data;
76: // ////////////////////////////////////////////////////////////////
77: // cout << "Game data is : " << game_data[0];
78: // cout << "size of string is " << game_data.size();
79: // int h = 0;
80: this->Load_sprites();
81: this->Calculate_points();
82: this->move_sprites();
83: }
84: void Sokoban::Load_textures() {
85: cout << "Loading Textures... " << std::endl;
86: block.loadFromFile("Textures/block.png");
87: wall.loadFromFile("Textures/wall.png");
88: ground.loadFromFile("Textures/ground.png");
89: goal.loadFromFile("Textures/goal.png");
90: sf::Texture p_u;
91: if ( !p_u.loadFromFile("Textures/up.png") ) {
92: cout << "Error loading texture" << std::endl;
93: }
94: sf::Texture p_d;
95: if ( !p_d.loadFromFile("Textures/down.png") ) {
96: cout << "Error loading texture" << std::endl;
```

```
97: }
98: sf::Texture p_l;
99: if ( !p_l.loadFromFile("Textures/left.png") ) {
100: cout << "Error loading texture" << std::endl;
101: }
102: sf::Texture p_r;
103: if ( !p_r.loadFromFile("Textures/right.png") ) {
104: cout << "Error loading texture" << std::endl;
105: }
106: player_textures.push_back(p_u);
107: player_textures.push_back(p_d);
108: player_textures.push_back(p_l);
109: player_textures.push_back(p_r);
110: }
111: void Sokoban::Load_sprites() {
112: cout << "Loading Sprites..." << std::endl;
113: // cout << "akjwl" << std::endl;
114: char c = 'v';
115: int i = 0;
116: for ( int w = 0; w < (x * y); w++ ) {
117: game_map.push_back(sf::Sprite());
118: // cout << "akjwl" << std::endl;
119: }
120: for ( int w = 0; w < (x * y); w++ ) {
121: if ( !isspace(c) ) {
122: switch (c) {
123: case '#':
124: game_map.at(w).setTexture(wall);
125: // cout << "DIFFERENT" << std::endl;
126: break;
127: case '.':
128: game_map.at(w).setTexture(ground);
129: break;
130: case 'A':
131: game_map.at(w).setTexture(block);
132: break;
133: case 'a':
134: game_map.at(w).setTexture(goal);
135: break;
136: case '1':
137: game_map.at(w).setTexture(block); // might need to change this
138: break;
139: case '@': // player
```

```
140: game_map.at(w).setTexture(ground);
141: player_pos = w;
142: player.setTexture(player_textures.at(1));
143: break;
144: default:
145: game_map.at(w).setTexture(wall);
146: }
147: } else {
148: w--;
149: }
150: c = game_data.at(i);
151: i++;
152: }
153: }
154: void Sokoban::Calculate_points() {
155: cout << "Calculating Points..." << std::endl;
156: for ( int w_y = 0; w_y < y; w_y++ ) {
157: for ( int i_x = 0; i_x < x; i_x++ ) {
158: // cout << i_x << std::endl;
159: draw_points.push_back(sf::Vector2f((64.f * i_x) + 32.f, (64.f * w_y)
+ 32.f));
160: }
161: }
162: }
163: void Sokoban::move_sprites() {
164: cout << "Moving sprites " << std::endl;
165: for ( int i = 0; i < (x * y); i++ ) {
166: game_map.at(i).setOrigin(32.f, 32.f);
167: game_map.at(i).setPosition(draw_points.at(i));
168: if ( i == player_pos ) {
169: cout << "Placing character" << std::endl;
170: player.setOrigin(32.f, 32.f);
171: player.setPosition(draw_points.at(i));
172: }
173: }
174: }
175: ifstream& operator>>(ifstream& file, Sokoban& game) {
176: game.Load_level(file);
177: return file;
178: }
179: } // namespace SB
```

### 4.4 Discussion part 2

This was one of the longest projects to implement and it took me a while to get the logic right. One of the reasons it took a while was because I had to get the logic to work with a 1 dimensional array and a changing level size. I also had to develop many more test functions to catch some bad implementations my tests were tested against.

The game implementation is to behave like the actual game, so W, A, S, D moves that character, the character can push boxes but can't push more than one, the character can't go through walls, and finally the game ends when all the boxes are pushed into a goal.

The logic with a 1 dimensional array is easy when the level size is constant and that's what I tested on mostly. I decided to get the logic to work with the basic level that was 10x10. This made the calculations easy, if the character moved up then you would subtract 10 from its position in the one dimensional vector. This would make the character's position change by one row. This is simple enough to implement for different sized levels, the harder part I found was calculating the bounds of different sized levels. When I would try a level that was 10x20 then the character would be able to go off screen and the game would crash. This was a long process of if statements to make sure the behavior of the game was as specified.

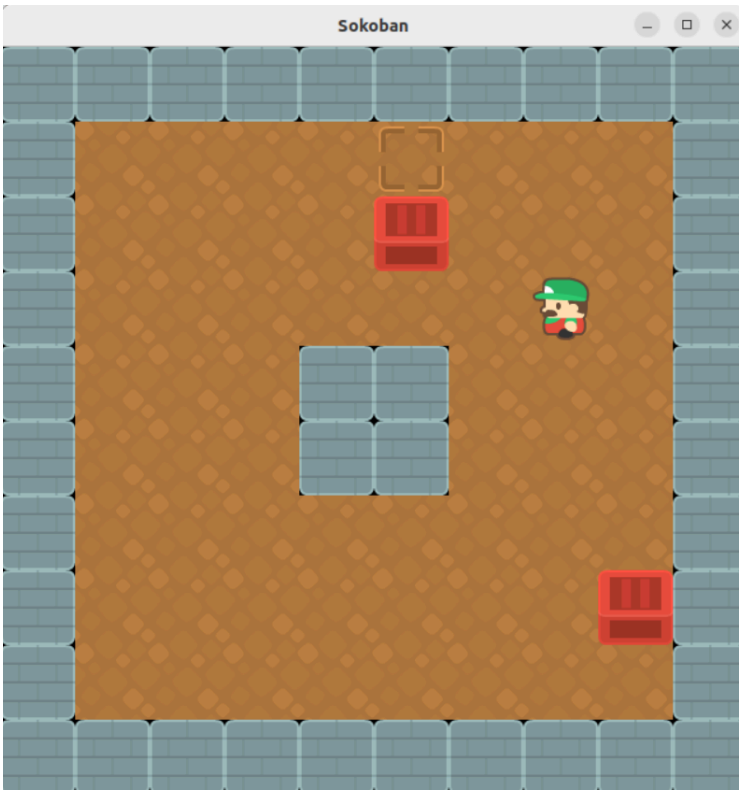
Once I got it to work correctly by validating it with the tests I wrote I decided to write more tests to specifically catch bad implementations the grader provided.

Some additional features that were required was a reset option, by pressing R then the entire level would reset. This was harder than expected but I ended up just initializing a new class with the same level. Last but not least, in order to tell if the level is beaten I had to revamp how I kept track of the boxes. I decided to put them in their own vector so I could keep track of their position. By doing this I could check if the level is won by iterating through the box vector and checking against

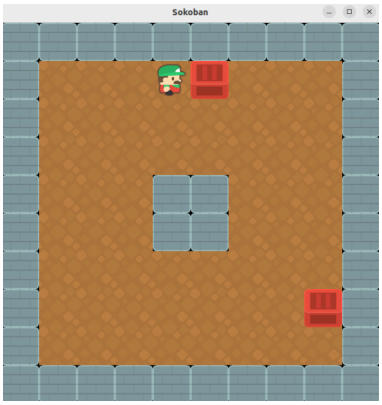
goal position. If at least one of them failed the check that would mean the game is still going.

For extra credit I made the character face the direction he is moving.

The final program looks like this:




The character moves around with keys W, A, S, D and when all boxes are in the right place:




```
smellett@Ubuntu:~/CompIV/Sokoban/ps3b$ ./Sokoban level1.txt

Placing character
YOU WON!
```

### 4.5 What I learned part 2

I think the most useful thing I learned from this project was accepting command line arguments. Other than that I'll need to take into more consideration how I'm reading in files in order to maximize compatibility with multiple machines.

By making extra tests to catch purposefully bad implementations gave me more experience in the Boost Testing Suite. I will be able to write more tests proactively rather than reactively like I have been doing.

Texture management is something I did in this project that will help me later on, the sf::Texture object is not carried along with the sf::Sprite object if they are moved or put into a vector. So assigning the texture after the sprite has been added to a vector is necessary.

### 4.6 Codebase part 2

Makefile

```
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
test_framework
4: OBJ = main.o
5:
6: .PHONY: all Sokoban clean lint pdf Sokoban.a
7:
8: all: Sokoban Sokoban.a test
9:
10: Sokoban : main.o Sokoban.o
11: $(CC) $(CFLAGS) -o Sokoban main.o Sokoban.o $(LIB)
12:
13: test : test.o Sokoban.o
14: $(CC) $(CFLAGS) -o test test.o Sokoban.o $(LIB)
15:
16: main.o : main.cpp Sokoban.hpp
17: $(CC) $(CFLAGS) -c main.cpp
18:
19: Sokoban.o : Sokoban.cpp Sokoban.hpp
20: $(CC) $(CFLAGS) -c Sokoban.cpp
21:
```

```
22: test.o : test.cpp Sokoban.hpp
23: $(CC) $(CFLAGS) -c test.cpp
24:
25: clean:
26: rm *.o Sokoban test
27:
28: lint:
29: cpplint *.cpp *.hpp
30:
31: pdf:
32: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
33: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
34: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
35:
36: Sokoban.a:
37: ar rcs Sokoban.a Sokoban.o
38:
```

main.cpp

```
1: // Copyright 2024 <Copyright Steven Mellett>
2: #include <iostream>
3: #include <fstream>
4: #include <string>
5: #include <SFML/Graphics.hpp>
6: #include <SFML/Window/Keyboard.hpp>
7: #include <SFML/System/Time.hpp>
8: #include "Sokoban.hpp"
9:
10: using std::ifstream;
11: using SB::Sokoban;
12:
13: int main(int argc, char* argv[]) {
14: sf::Time t1 = sf::seconds(0.175f);
15: sf::Time t2 = sf::seconds(1.f);
16:
17: std::string FILE_NAME;
18: sf::Keyboard K;
19: if ( argc <= 1 ) {
20: FILE_NAME = "level1.txt";
21: std::cout << "Please enter level name: ";
22: std::cin >> FILE_NAME;
23: } else {
```

```
24: FILE_NAME = argv[1];
25: }
26: // std::cout << "Please enter level name: ";
27: // std::cin >> FILE_NAME;
28: std::cout << std::endl;
29: Sokoban game;
30: ifstream level_file(FILE_NAME);
31: if ( level_file.is_open() ) {
32: level_file >> game;
33: level_file.close();
34: } else {
35: std::cout << "Error opening file, or file doesn't exist" << std::endl;
36: return 1;
37: }
38:
39: sf::RenderWindow window(sf::VideoMode(game.width()*64,
game.height()*64, 64), "Sokoban");
40: sf::Event event;
41: while ( window.isOpen() ) {
42: while ( window.pollEvent(event) ) {
43: if ( event.type == sf::Event::Closed ) {
44: window.close();
45: }
46: }
47: if ( game.isWon() ) {
48: std::cout << "YOU WON!" << std::endl;
49: window.clear();
50: window.draw(game);
51: window.display();
52: sf::sleep(t2);
53: return 0;
54: }
55: window.clear();
56: window.draw(game);
57: if ( K.isKeyPressed(sf::Keyboard::Key::A) ) {
58: game.movePlayer(SB::Left);
59: }
60: if ( K.isKeyPressed(sf::Keyboard::Key::S) ) {
61: game.movePlayer(SB::Down);
62: }
63: if ( K.isKeyPressed(sf::Keyboard::Key::D) ) {
64: game.movePlayer(SB::Right);
65: }
```

```
66: if ( K.isKeyPressed(sf::Keyboard::Key::W) ) {
67: game.movePlayer(SB::Up);
68: }
69: if ( K.isKeyPressed(sf::Keyboard::Key::R) ) {
70: ifstream reset_level(FILE_NAME);
71: if ( reset_level.is_open() ) {
72: game.reset(reset_level);
73: reset_level.close();
74: } else {
75: std::cout << "Error getting level data for reset." << std::endl;
76: return 1;
77: }
78: }
79: window.display();
80: sf::sleep(t1);
81: }
82: return 0;
83: }
```

Sokoban.cpp

```
1: // Copyright 2024 <Copyright Steven Mellett>
2: #include "Sokoban.hpp"
3: #include <string>
4: #include <fstream>
5: #include <iostream>
6: #include <cctype>
7: #include <vector>
8:
9: using std::ifstream;
10: using std::string;
11: using std::cout;
12: using std::vector;
13:
14: namespace SB {
15: Sokoban::Sokoban() {
16: this->Load_textures();
17: // this->Load_sprites();
18: // this->Calculate_points(); // moved to end of level;
19: // this->move_sprites(); moved to end of level;
20: }
21: Sokoban::Sokoban(ifstream& level) {
```

```
22: this->Load_level(level);
23: this->Load_textures();
24: // this->Load_sprites();
25: // this->Calculate_points();
26: // this->move_sprites();
27: }
28: void Sokoban::Load_level(ifstream& level) {
29: // cout << "Loading Level..." << std::endl;
30: string level_string;
31: // vector<char> level_vector;
32: char array_c[1000];
33: level.readsome(array_c, 100000);
34: // cout << array_c << std::endl;
35: char c = 'c';
36: char b = 'b';
37: int w = 0;
38: int i = 0;
39: c = array_c[i];
40: i++;
41: while (!isspace(c)) {
42: x *= 10;
43: int temp = c - '0';
44: x += temp;
45: c = array_c[i];
46: i++;
47: }
48: c = array_c[i];
49: i++;
50: while (!isspace(c)) {
51: y *= 10;
52: int temp = c - '0';
53: y += temp;
54: c = array_c[i];
55: i++;
56: }
57: // cout << x << ' ' << y << std::endl; works
58: // cout << array_c << std::endl;
59: // level >> level_vector;//////////////////////////////////////////////
60: // level >> game_data;
61: while (b != '#') {
62: b = array_c[w];
63: // cout << array_c[w];
64: w++;
```

```
65: }
66: w--;
67: for ( int p = 0; p < (x * y); p++ ) {
68: if ( !isspace(array_c[w]) ) {
69: game_data.push_back(array_c[w]);
70: // array_c[w];
71: } else {
72: p--;
73: }
74: w++;
75: }
76: // cout << game_data;
77: // ////////////////////////////////////////////////////////////
78: // cout << "Game data is : " << game_data[0];
79: // cout << "size of string is " << game_data.size();
80: // int h = 0;
81: this->Load_sprites();
82: this->Calculate_points();
83: this->move_sprites();
84: }
85: void Sokoban::Load_textures() {
86: // cout << "Loading Textures... " << std::endl;
87: block.loadFromFile("Textures/block.png");
88: wall.loadFromFile("Textures/wall.png");
89: ground.loadFromFile("Textures/ground.png");
90: goal.loadFromFile("Textures/goal.png");
91: sf::Texture p_u;
92: if ( !p_u.loadFromFile("Textures/up.png") ) {
93: cout << "Error loading texture" << std::endl;
94: }
95: sf::Texture p_d;
96: if ( !p_d.loadFromFile("Textures/down.png") ) {
97: cout << "Error loading texture" << std::endl;
98: }
99: sf::Texture p_l;
100: if ( !p_l.loadFromFile("Textures/left.png") ) {
101: cout << "Error loading texture" << std::endl;
102: }
103: sf::Texture p_r;
104: if ( !p_r.loadFromFile("Textures/right.png") ) {
105: cout << "Error loading texture" << std::endl;
106: }
107: player_textures.push_back(p_u);
```

```
108: player_textures.push_back(p_d);
109: player_textures.push_back(p_l);
110: player_textures.push_back(p_r);
111: }
112: void Sokoban::Load_sprites() {
113: // cout << "Loading Sprites..." << std::endl;
114: // cout << "akjwl" << std::endl;
115: block_data blocks;
116: char c = 'v';
117: int i = 0;
118: for ( int w = 0; w < (x * y); w++ ) {
119: game_map.push_back(sf::Sprite());
120: // cout << "akjwl" << std::endl;
121: }
122: for ( int w = 0; w < (x * y); w++ ) {
123: c = game_data.at(i);
124: if ( !isspace(c) ) {
125: switch (c) {
126: case '#':
127: game_map.at(w).setTexture(wall);
128: // cout << "DIFFERENT" << std::endl;
129: break;
130: case '.':
131: game_map.at(w).setTexture(ground);
132: break;
133: case 'A':
134: blocks.place = w;
135: blocks.block.setTexture(block);
136: vblocks.push_back(blocks);
137: game_map.at(w).setTexture(ground);
138: break;
139: case 'a':
140: game_map.at(w).setTexture(goal);
141: goals.push_back(w);
142: break;
143: case '1':
144: game_map.at(w).setTexture(block); // might need to change this
145: break;
146: case '@': // player
147: game_map.at(w).setTexture(ground);
148: player_pos = w;
149: player.setTexture(player_textures.at(1));
150: break;
```

```
151: default:
152: game_map.at(w).setTexture(wall);
153: }
154: }
155: i++;
156: }
157: }
158: void Sokoban::Calculate_points() {
159: // cout << "Calculating Points..." << std::endl;
160: for ( int w_y = 0; w_y < y; w_y++ ) {
161: for ( int i_x = 0; i_x < x; i_x++ ) {
162: // cout << i_x << std::endl;
163: draw_points.push_back(sf::Vector2f((64.f * i_x) + 32.f, (64.f * w_y)
+ 32.f));
164: }
165: }
166: }
167: void Sokoban::move_sprites() {
168: // cout << "Moving sprites " << std::endl;
169: for ( int i = 0; i < (x * y); i++ ) {
170: game_map.at(i).setOrigin(32.f, 32.f);
171: game_map.at(i).setPosition(draw_points.at(i));
172: if ( i == player_pos ) {
173: cout << "Placing character" << std::endl;
174: player.setOrigin(32.f, 32.f);
175: player.setPosition(draw_points.at(i));
176: }
177: }
178: for ( unsigned int i = 0; i < vblocks.size(); i++ ) {
179: vblocks.at(i).block.setOrigin(32.f, 32.f);
180: vblocks.at(i).block.setPosition(draw_points.at(vblocks.at(i).place));
181: }
182: }
183: ifstream& operator>>(ifstream& file, Sokoban& game) {
184: game.Load_level(file);
185: return file;
186: }
187:
188: void Sokoban::movePlayer(Direction dir) {
189: // cout << player_pos << std::endl;
190: // cout << game_data << std::endl;
191: switch (dir) {
192: case Up:
```

```
193: player.setTexture(player_textures.at(0));
194: if ( player_pos - width() <= 0 || game_data.at(player_pos - width()) =
= '#' ||
195: game_data.at(player_pos - width()) == '1' ) {
196: break;
197: } else if ( game_data.at(player_pos - width()) == 'A' ) {
198: if ( !moveBlocks(Up, player_pos - width()) ) {
199: break;
200: } else {
201: game_data.at(player_pos - width()) = '.';
202: if ( game_data.at(player_pos - (1 * width())) == 'a' ) {
203: // cout << player_pos << std::endl;
204: game_data.at(player_pos - (1 * width())) = '1';
205: game_data.at(player_pos) = '.';
206: } else if ( !(game_data.at(player_pos - (2 * width())) == '1') ) {
207: // cout << player_pos << std::endl;
208: game_data.at(player_pos - (2 * width())) = 'A';
209: }
210: }
211: }
212: player_pos -= width();
213: // player.setTexture(player_textures.at(0));
214: player.setPosition(draw_points.at(player_pos));
215: break;
216: case Down:
217: player.setTexture(player_textures.at(1));
218: if (player_pos + width() >= (width() * width()) ||
219: game_data.at(player_pos + width()) == '#' || game_data.at(player_po
s + width()) == '1') {
220: break;
221: } else if ( game_data.at(player_pos + width()) == 'A' ) {
222: if (!moveBlocks(Down, player_pos + width())) {
223: break;
224: } else {
225: game_data.at(player_pos + width()) = '.';
226: if ( game_data.at(player_pos + (1 * width())) == 'a' ) {
227: game_data.at(player_pos + (1 * width())) = '1';
228: game_data.at(player_pos) = '.';
229: } else if ( !(game_data.at(player_pos + (2 * width())) == '1') ) {
230: // cout << player_pos << std::endl;
231: game_data.at(player_pos + (2 * width())) = 'A';
232: }
233: }
```

```
234: }
235: player_pos += width();
236: // player.setTexture(player_textures.at(1));
237: player.setPosition(draw_points.at(player_pos));
238: break;
239: case Left:
240: player.setTexture(player_textures.at(2));
241: if ( player_pos - 1 <= 0 || game_data.at(player_pos - 1) == '#' ||
242: game_data.at(player_pos - 1) == '1' ) {
243: break;
244: } else if ( game_data.at(player_pos - 1) == 'A' ) {
245: if ( !moveBlocks(Left, player_pos - 1) ) {
246: break;
247: } else {
248: game_data.at(player_pos - 1) = '.';
249: if ( game_data.at(player_pos - 1) == 'a' ) {
250: game_data.at(player_pos - 1) = '1';
251: game_data.at(player_pos) = '.';
252: } else if ( !(game_data.at(player_pos - 2) == '1') ) {
253: // cout << player_pos << std::endl;
254: game_data.at(player_pos - 2) = 'A';
255: }
256: }
257: }
258: player_pos -= 1;
259: // player.setTexture(player_textures.at(2));
260: player.setPosition(draw_points.at(player_pos));
261: break;
262: case Right:
263: player.setTexture(player_textures.at(3));
264: if ( player_pos + 1 >= (width() * width()) || game_data.at(player_pos
+ 1) == '#' ||
265: game_data.at(player_pos + 1) == '1' ) {
266: break;
267: } else if ( game_data.at(player_pos + 1) == 'A' ) {
268: if ( !moveBlocks(Right, player_pos + 1) ) {
269: break;
270: } else {
271: game_data.at(player_pos + 1) = '.';
272: if ( game_data.at(player_pos + 1) == 'a' ) {
273: game_data.at(player_pos + 1) = '1';
274: game_data.at(player_pos) = '.';
275: } else if ( !(game_data.at(player_pos + 2) == '1') ) {
```

```
276: // cout << player_pos << std::endl;
277: game_data.at(player_pos + 2) = 'A';
278: }
279: }
280: }
281: player_pos += 1;
282: // player.setTexture(player_textures.at(3));
283: player.setPosition(draw_points.at(player_pos));
284: break;
285: }
286: }
287: int Sokoban::moveBlocks(Direction dir, int place) {
288: // cout << "in mve bblocks " << std::endl;
289: for ( unsigned int i = 0; i < vblocks.size(); i++ ) {
290: if ( place == vblocks.at(i).place ) {
291: switch (dir) {
292: case Up:
293: // cout << " up " << std::endl;
294: // cout << game_data.at(place - (2 * width())) << std::endl;
295: if ( place - width() <= 0 || game_data.at(place - width()) == '#'
||
296: game_data.at(place - width()) == 'A' ) {
297: return 0;
298: break;
299: } else if ( game_data.at(place - width()) == 'a' ) {
300: // cout << "VHJHGFVHJUG" << std::endl;
301: game_data.at(place - width()) = '1';
302: }
303: vblocks.at(i).block.setPosition(draw_points.at(place - width()));
304: vblocks.at(i).place = place - width();
305: return 1;
306: case Down:
307: // cout << "down" << std::endl;
308: // cout << game_data.at(place + (2 * width())) << std::endl;
309: if ( place + width() >= (width() * width()) || game_data.at(place
+ width()) == '#' ||
310: game_data.at(place + width()) == 'A' ) {
311: return 0;
312: break;
313: } else if ( game_data.at(place + width()) == 'a' ) {
314: // cout << "VHJHGFVHJUG" << std::endl;
315: game_data.at(place + width()) = '1';
316: }
```

```
317: vblocks.at(i).block.setPosition(draw_points.at(place + width()));
318: vblocks.at(i).place = place + width();
319: return 1;
320: case Left:
321: // cout << "left" << std::endl;
322: // cout << game_data.at(place - 2) << std::endl;
323: if ( place - 1 <= 0 || game_data.at(place - 1) == '#' ||
324: game_data.at(place - 1) == 'A' ) {
325: return 0;
326: break;
327: } else if (game_data.at(place - 1) == 'a') {
328: // cout << "VHJHGFVHJUG" << std::endl;
329: game_data.at(place - 1) = '1';
330: }
331: vblocks.at(i).block.setPosition(draw_points.at(place - 1));
332: vblocks.at(i).place = place - 1;
333: return 1;
334: case Right:
335: if ( place + 1 >= (width() * width()) || game_data.at(place + 1) =
= '#' ||
336: game_data.at(place + 1) == 'A' ) {
337: return 0;
338: break;
339: } else if (game_data.at(place + 1) == 'a') {
340: game_data.at(place + 1) = '1';
341: }
342: vblocks.at(i).block.setPosition(draw_points.at(place + 1));
343: vblocks.at(i).place = place + 1;
344: return 1;
345: }
346: }
347: }
348: return 0;
349: }
350: void Sokoban::reset(ifstream& level) {
351: game_data.resize(0);
352: char b = 'v';
353: char c = 'c';
354: int i = 0;
355: char array_c[10000];
356: level.readsome(array_c, 10000);
357: // cout << array_c << std::endl;
358: while ( b != '#' ) {
```

```
359: b = array_c[i];
360: // cout << array_c[w];
361: i++;
362: }
363: i--;
364: for ( int p = 0; p < (x * y); p++ ) {
365: if ( !isspace(array_c[i]) ) {
366: game_data.push_back(array_c[i]);
367: // array_c[w];
368: } else {
369: p--;
370: }
371: i++;
372: }
373: i = 0;
374: // cout << "WE FO TO ANK:DJAS" << std::endl;
375: for ( int w = 0; w < (x * y); w++ ) {
376: c = game_data.at(i);
377: if ( c == '@' ) {
378: player_pos = w;
379: }
380: }
381: vblocks.resize(0);
382: this->Load_sprites();
383: this->move_sprites();
384: }
385: int Sokoban::isWon() const {
386: for ( int i = 0; i < static_cast<int>(goals.size()); i++ ) {
387: if ( !(game_data.at(goals.at(i)) == '1') ) {
388: return 0;
389: }
390: }
391: return 1;
392: }
393: sf::Vector2i Sokoban::playerLoc() const {
394: int y = player_position() / height();
395: int x = player_position() - (y * height());
396: return sf::Vector2i(x, y);
397: }
398: } // namespace SB
399:
```

test.cpp

```cpp
// Copyright 2024
// By Steven Mellett
// test.cpp for ps3b

#include <iostream>
#include <string>
#include <fstream>
#include "Sokoban.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

using SB::Sokoban;

BOOST_AUTO_TEST_CASE(test_level_size_square) {
std::string FILE_NAME;
FILE_NAME = "level1.txt";
Sokoban game;
std::ifstream level_file(FILE_NAME);
level_file >> game;
level_file.close();
BOOST_REQUIRE_EQUAL(game.width(), 10);
BOOST_REQUIRE_EQUAL(game.height(), 10);
}

BOOST_AUTO_TEST_CASE(test_move_through_boxes_and_walls) {
std::string FILE_NAME;
FILE_NAME = "level1.txt";
Sokoban game;
std::ifstream level_file(FILE_NAME);
level_file >> game;
level_file.close();
game.movePlayer(SB::Direction::Right);
game.movePlayer(SB::Direction::Right);
game.movePlayer(SB::Direction::Right);
game.movePlayer(SB::Direction::Right);
game.movePlayer(SB::Direction::Right);
BOOST_REQUIRE_EQUAL(game.playerLoc().x, 7);
BOOST_REQUIRE_EQUAL(game.playerLoc().y, 6);
}
BOOST_AUTO_TEST_CASE(test_move_box_box) {
```

```
43: std::string FILE_NAME;
44: FILE_NAME = "level1.txt";
45: Sokoban game;
46: std::ifstream level_file(FILE_NAME);
47: level_file >> game;
48: level_file.close();
49: game.movePlayer(SB::Direction::Up);
50: game.movePlayer(SB::Direction::Up);
51: game.movePlayer(SB::Direction::Up);
52: game.movePlayer(SB::Direction::Up);
53: game.movePlayer(SB::Direction::Right);
54: game.movePlayer(SB::Direction::Right);
55: game.movePlayer(SB::Direction::Up);
56: game.movePlayer(SB::Direction::Right);
57: game.movePlayer(SB::Direction::Down);
58: game.movePlayer(SB::Direction::Down);
59: game.movePlayer(SB::Direction::Down);
60: game.movePlayer(SB::Direction::Down);
61: BOOST_REQUIRE_EQUAL(game.playerLoc().x, 6);
62: BOOST_REQUIRE_EQUAL(game.playerLoc().y, 4);
63: }
64: BOOST_AUTO_TEST_CASE(test_correct_player_start_pos) {
65: std::string FILE_NAME;
66: FILE_NAME = "level1.txt";
67: Sokoban game;
68: std::ifstream level_file(FILE_NAME);
69: level_file >> game;
70: level_file.close();
71: BOOST_REQUIRE_EQUAL(game.playerLoc().x, 3);
72: BOOST_REQUIRE_EQUAL(game.playerLoc().y, 6);
73: // game data store in vector - player should be located at 64 (63)
place
in vector
74: }
75:
76: BOOST_AUTO_TEST_CASE(test_player_movement) {
77: std::string FILE_NAME;
78: FILE_NAME = "level1.txt";
79: Sokoban game;
80: std::ifstream level_file(FILE_NAME);
81: level_file >> game;
82: level_file.close();
83: game.movePlayer(SB::Direction::Up);
```

```
84: BOOST_REQUIRE_EQUAL(game.playerLoc().y, 5);
85: game.movePlayer(SB::Direction::Left);
86: BOOST_REQUIRE_EQUAL(game.playerLoc().x, 2);
87: game.movePlayer(SB::Direction::Down);
88: BOOST_REQUIRE_EQUAL(game.playerLoc().y, 6);
89: game.movePlayer(SB::Direction::Right);
90: BOOST_REQUIRE_EQUAL(game.playerLoc().x, 3);
91: }
```

# 5 PS4: Static/Dynamic N-Body Simulation

PS4a: Static N-Body Simulation

## 5.1 Discussion

This is another command line app that takes two doubles and an .txt input file. The first double specifies the amount of time to simulate and the second specifies delta time between steps of simulation. This part of the project is more about reading in the file and setting up the simulation than any math stuff.

The input .txt is in this format:

```
...xpos...  ...ypos...  ...xvel...  ...yvel...  ...mass...  filename
1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24  earth.gif
2.2790e+11  0.0000e+00  0.0000e+00  2.4100e+04  6.4190e+23  mars.gif
5.7900e+10  0.0000e+00  0.0000e+00  4.7900e+04  3.3020e+23  mercury.gif
0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00  1.9890e+30  sun.gif
1.0820e+11  0.0000e+00  0.0000e+00  3.5000e+04  4.8690e+24  venus.gif
```

Each planet has an assigned position, velocities, mass and image file. The .txt also says how many planets there are and the size of the universe (which will be important later).

The solution involves two different classes specified as a Universe class and a CelestialBody class. I chose to handle the physics in the Universe class and use getters and setters from the CelestialBody class to access and change the values stored in the CelestialBody class. The CelestialBody class handles reading the

input from the file and also outputting to stdout. This separation was something I stumbled into but found it was easy enough to work with.

I could initialize the correct amount of CelestialBody's immediately and have them stored in a vector inside of the Universe class. This system allowed the Universe object to easily iterate through each planet and add forces to them later on in the project.

Reading in the data wasn't that hard after I sorted out how classes were going to be organized with data being stored in the CelestialBodies. Then outputting to stdout was a little more difficult because I had to get it exactly in the format it came from. This involved some cool standard library functions that I will talk about in the What I Learned part 1 section.

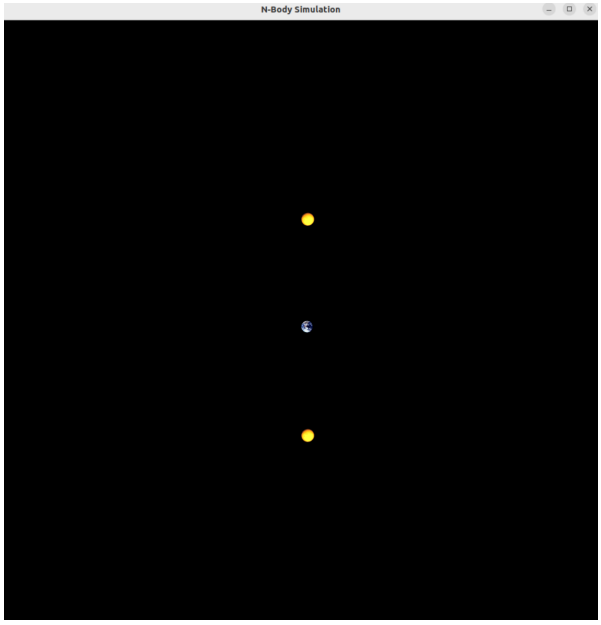
One thing to note about the numbers given in the .txt is that they are doubles in exponential form. They are way larger than you can represent pixels on your screen. Maybe you would be able to do it if each planet is 1 pixel wide but that defeats the purpose of the simulation. In order to accomplish this some simple math needs to alter the values of the position of the planets to scale it to the window size.

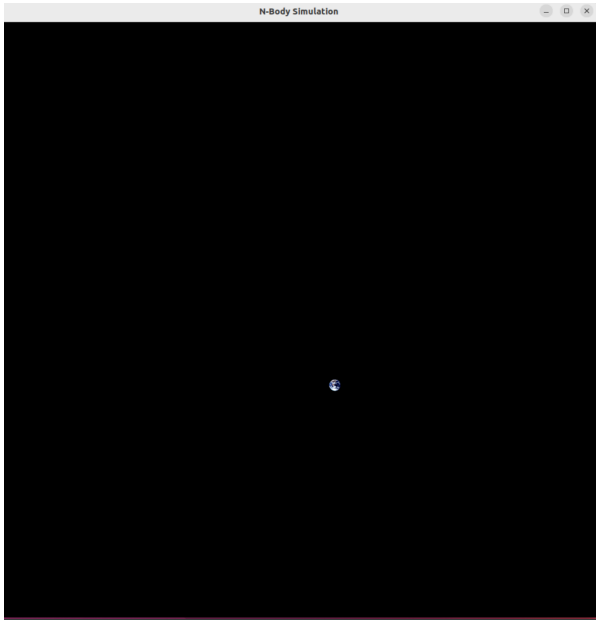
To do this I chose an easy 1000x1000 pixel wide window to display the planets. By using the ratio of the planet's position to the universe size in one x or y direction. This would give some value from 0 to 1. Then multiplying that number by 1000 gives you the position along that respective axis. Adding 500 to each axis then centers the entire universe. This is because (0, 0) origin is in the top left according to sf::Window object.

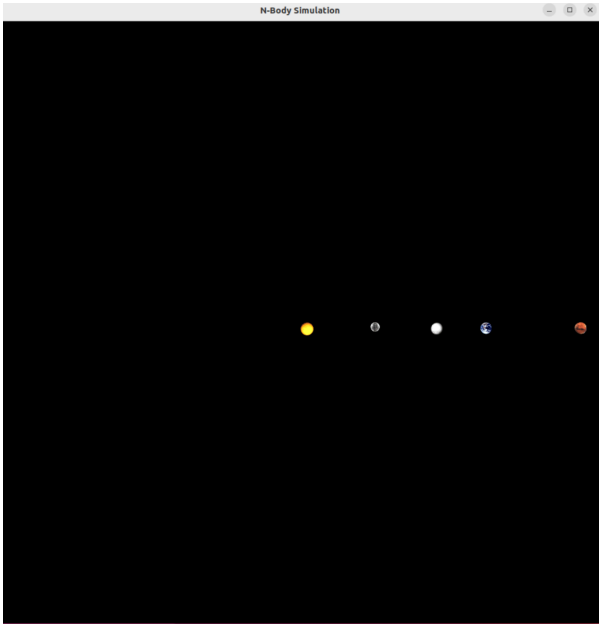
At this point I had everything working. I just had to draw them to the window. This is where I ran into most of my troubles at first. The Universe class inherits from sf::Drawable meaning I can overload the virtual draw function exactly like last project. When I did this however, the planets were only drawn to the screen as white boxes. The sprite objects were not able to keep the texture object when I assigned them to the vector in the Universe class. One possible and best solution to this would be to make the vector hold pointers to the CelestialBody objects so they

aren’t moved when assigning them to the vector. I could not get this implementation to work so I opted for another less ideal solution. I assigned the textures to each CelestialBody sf::Sprite object after they had been added to the vector inside the Universe class. This fixed the issue but kinda defeated the purpose of the CelestialBody’s being able to operate completely independent from one another.

When all was working here is what a few different Universes looked like:



### 5.2 What I learned part 1

Specifically in this assignment it was mentioned that managing the lifetime of the CelestialBody object was important and I ran into that exact problem with this program. As I said before, the sf::Sprite object would not hold onto the sf::Texture it was assigned before adding it to the vector stored inside the Universe class. The reason behind this has something to do with the sprite objects only having a reference to the texture rather than carrying around a whole texture object with them. If I were to do this from the beginning I would put a lot more time into changing the Universe bodies vector to hold pointers to the CelestialBody objects rather than the objects themselves.

Formatting the output using the normal stdout functions was also something I had not done in C++. In C, in order to use stdout you would use printf or something similar with different kinds of parameters to specify the format. In C++ it's much easier, using std::setw() and other functions from the <iomanip> library you can do the same things without the hassle of using printf.

### 5.3 Codebase part 1

```
Makefile
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
test_framework
4: OBJ = main.o Universe.o CelestialBody.o
5:
6: .PHONY: all clean lint
7:
8: all: NBody test NBody.a
9:
10: NBody : main.o Universe.o CelestialBody.o
11: $(CC) $(CFLAGS) -o NBody main.o Universe.o CelestialBody.o $(LIB)
12:
13: test : test.o Universe.o CelestialBody.o
14: $(CC) $(CFLAGS) -o test test.o Universe.o CelestialBody.o $(LIB)
15:
16: main.o : main.cpp Universe.hpp CelestialBody.hpp
17: $(CC) $(CFLAGS) -c main.cpp
```

```
18:
19: Universe.o : Universe.cpp Universe.hpp
20: $(CC) $(CFLAGS) -c Universe.cpp
21:
22: CelestialBody.o : CelestialBody.cpp CelestialBody.hpp
23: $(CC) $(CFLAGS) -c CelestialBody.cpp
24:
25: test.o : test.cpp Universe.hpp CelestialBody.hpp
26: $(CC) $(CFLAGS) -c test.cpp
27:
28: clean:
29: rm *.o NBody test
30:
31: lint:
32: cpplint *.cpp *.hpp
33:
34: pdf:
35: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
36: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
37: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
38:
39: NBody.a:
40: ar rcs NBody.a Universe.o CelestialBody.o
41:
42:
```

main.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // main.cpp for ps4a
4: // updated 3/18/2024
5:
6:
7: #include <iostream>
8: #include <istream>
9: #include <fstream>
10: #include "Universe.hpp"
11:
12:
13: using NB::Universe;
14: using NB::CelestialBody;
15:
```

```
16: int main(int argc, char* argv[]) {
17: sf::RenderWindow window(sf::VideoMode(1000, 1000), "N-Body
Simulation");
18: sf::Event event;
19: Universe uni;
20: CelestialBody planet;
21: std::cin >> uni;
22: while ( window.isOpen() ) {
23: while ( window.pollEvent(event) ) {
24: if ( event.type == sf::Event::Closed ) {
25: window.close();
26: }
27: }
28: window.clear();
29: for ( int i = 0; i < uni.planets(); i++ ) {
30: window.draw(uni.get_planet(i));
31: // std::cout << uni.get_planet(i);
32: }
33: window.display();
34: }
35: return 0;
36: }
```

```
Universe.hpp Tue Mar 19 17:45:57 2024 1
1: // Copyright 2024
2: // By Steven Mellett
3: // Universe.hp for ps4a
4: // updated 3/18/2024
5: #ifndef UNIVERSE_HPP
6: #define UNIVERSE_HPP
7: #include <vector>
8: #include <istream>
9: #include <iostream>
10: #include "CelestialBody.hpp"
11: #include <SFML/Graphics.hpp>
12: namespace NB {
13:
14: class Universe : public sf::Drawable {
15: public:
16: Universe();
17: int planets() const { return num_planets; }
18: CelestialBody get_planet(int i) const { return bodies.at(i); }
```

```
19: double get_size() const { return size; }
20: int find_farthest_planet();
21: void place_planets();
22: CelestialBody get_body(int i) const { return bodies.at(i); }
23:
24: private:
25: int num_planets = 0;
26: double size;
27: std::vector<CelestialBody> bodies;
28:
29: friend std::istream& operator>>(std::istream& in, Universe& uni);
30: friend std::ostream& operator<<(std::ostream& out, Universe& uni);
31: protected:
32: virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const
{
33: for ( int i = 0; i < num_planets; i++ ) {
34: target.draw(bodies.at(i).get_sprite());
35: }
36: }
37: };
38: } // namespace NB
39:
40: #endif
```

```
Universe.cpp
1: // Copyright 2024
2: // By Steven Mellett
3: // Universe.cpp for ps4a
4: // updated 3/18/2024
5:
6: #include <iostream>
7: #include <istream>
8: #include <string>
9: #include "Universe.hpp"
10: #include "CelestialBody.hpp"
11:
12:
13:
14: namespace NB {
15:
16: Universe::Universe() {
```

```
17: num_planets = 0;
18: }
19: int Universe::find_farthest_planet() {
20: int index = 0;
21: for ( int i = 0; i < num_planets - 1; i++ ) {
22: if ( abs(bodies.at(index).get_xpos()) < abs(bodies.at(i + 1).get_xpos(
)) ) {
23: index = i + 1;
24: }
25: }
26: return index;
27: }
28: void Universe::place_planets() {
29: float x;
30: float y;
31: for ( int i = 0; i < num_planets; i++ ) {
32: bodies.at(i).load_texture();
33: x = static_cast<float>((1000 * (bodies.at(i).get_xpos() / (size*2))) +
500);
34: y = static_cast<float>((1000 * (bodies.at(i).get_ypos() / (size*2))) +
500);
35: bodies.at(i).move_sprite(x, y);
36: }
37: }
38: std::istream& operator>>(std::istream& in, Universe& uni) {
39: char line[256];
40: in.getline(line, 10);
41: int num = std::stoi(line, nullptr, 10);
42: uni.num_planets = num;
43: in >> uni.size;
44: for ( int i = 0; i < uni.num_planets; i++ ) {
45: CelestialBody planet;
46: in >> planet;
47: uni.bodies.push_back(planet);
48: }
49: uni.place_planets();
50: return in;
51: }
52: std::ostream& operator<<(std::ostream& out, Universe& uni) {
53: out << uni.num_planets << std::endl;
54: out << uni.size << std::endl;
55: for ( int i = 0; i < uni.num_planets; i++ ) {
56: out << uni.bodies.at(i);
```

```
57: }
58: return out;
59: }
60: } // namespace NB
```

```
CelestialBody.hpp Tue Mar 19 17:32:57 2024 1
1: // Copyright 2024
2: // By Steven Mellett
3: // CelestialBody.hpp for ps4a
4: // updated 3/18/2024
5: #ifndef CELESTIALBODY_HPP
6: #define CELESTIALBODY_HPP
7: #include <string>
8: #include <iostream>
9: #include <SFML/Graphics.hpp>
10: #include <SFML/Graphics/Drawable.hpp>
11: #include <SFML/Graphics/Texture.hpp>
12: #include <SFML/Graphics/Sprite.hpp>
13:
14: namespace NB {
15:
16: class CelestialBody : public sf::Drawable {
17: public:
18: CelestialBody();
19: double get_xpos() const { return xpos; }
20: double get_ypos() const { return ypos; }
21: double get_xvel() const { return xvel; }
22: double get_yvel() const { return yvel; }
23: double get_mass() const { return mass; }
24: std::string get_name() const { return FILE_NAME; }
25: void move_sprite(float x, float y) { sprite.setPosition(x, y); }
26: void load_texture() { texture.loadFromFile(FILE_NAME);
sprite.setTexture(t
exture); }
27: sf::Sprite get_sprite() const { return sprite; }
28:
29: private:
30: double xpos;
31: double ypos;
32: double xvel;
33: double yvel;
34: double mass;
```

```
35: std::string FILE_NAME;
36: sf::Sprite sprite;
37: sf::Texture texture;
38: friend std::istream& operator>>(std::istream& in, CelestialBody&
planet);
39: friend std::ostream& operator<<(std::ostream& out, CelestialBody&
planet);
40:
41: protected:
42: virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const {
43: target.draw(sprite, states);
44: }
45: };
46: } // namespace NB
47:
48: #endif
```

CelestialBody.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // CelestialBody.cpp for ps4a
4: // updated 3/18/2024
5: #include <iostream>
6: #include <iomanip>
7: #include "CelestialBody.hpp"
8: #include "Universe.hpp"
9:
10: namespace NB {
11: CelestialBody::CelestialBody() {}
12: std::istream& operator>>(std::istream& in, CelestialBody& planet) {
13: in >> planet.xpos;
14: in >> planet.ypos;
15: in >> planet.xvel;
16: in >> planet.yvel;
17: in >> planet.mass;
18: in >> planet.FILE_NAME;
19: return in;
20: }
21: std::ostream& operator<<(std::ostream& out, CelestialBody& planet) {
22: out << " " << std::scientific << planet.get_xpos() << " ";
23: out << planet.get_ypos() << " " << planet.get_xvel() << " ";
```

```
24: out << planet.get_yvel() << " " << planet.get_mass() << " ";
25: out << std::setw(14) << std::right << planet.get_name() << std::endl;
26: return out;
27: }
28: } // namespace NB
```

test.cpp

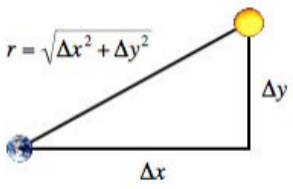
```
1: // Copyright 2024
2: // By Steven Mellett
3: // test.cpp for ps4a
4: #include <iostream>
5: #include <string>
6: #include <fstream>
7: #include "Universe.hpp"
8: #include "CelestialBody.hpp"
9: #define BOOST_TEST_DYN_LINK
10: #define BOOST_TEST_MODULE Main
11: #include <boost/test/unit_test.hpp>
12: using NB::Universe;
13:
14: BOOST_AUTO_TEST_CASE(test_correct_size) {
15: std::string FILE_NAME;
16: FILE_NAME = "planets.txt";
17: Universe uni;
18: std::ifstream level_file(FILE_NAME);
19: level_file >> uni;
20: level_file.close();
21: BOOST_REQUIRE_EQUAL(uni.get_size(), static_cast<double>(2.5e+11));
22: }
23:
24: BOOST_AUTO_TEST_CASE(test_number_planets) {
25: std::string FILE_NAME;
26: FILE_NAME = "planets.txt";
27: Universe uni;
28: std::ifstream level_file(FILE_NAME);
29: level_file >> uni;
30: level_file.close();
31: BOOST_REQUIRE_EQUAL(uni.planets(), 5);
32: }
33: BOOST_AUTO_TEST_CASE(test_planet_xvel_zero) {
34: std::string FILE_NAME;
35: FILE_NAME = "planets.txt";
```

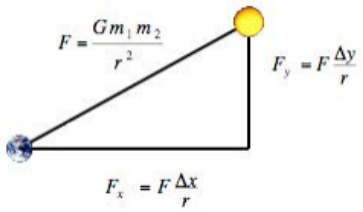
```
36: Universe uni;
37: std::ifstream level_file(FILE_NAME);
38: level_file >> uni;
39: level_file.close();
40: for ( int i = 0; i < uni.planets(); i++ ) {
41: BOOST_REQUIRE_EQUAL(uni.get_body(i).get_xvel(), 0);
42: }
43: }
44: BOOST_AUTO_TEST_CASE(test_planet_mass_non_zero) {
45: std::string FILE_NAME;
46: FILE_NAME = "planets.txt";
47: Universe uni;
48: std::ifstream level_file(FILE_NAME);
49: level_file >> uni;
50: level_file.close();
51: for ( int i = 0; i < uni.planets(); i++ ) {
52: BOOST_REQUIRE(uni.get_body(i).get_mass() > 0);
53: }
54: }
55:
```

PS4b: Dynamic N-Body Simulation

### 5.4 Discussion part 2

This section of the assignment is adding the actual moving part of the simulation to the static universe I had before. For me, converting the formulas into something that had the correct behavior was very difficult. I found that I had gotten movement to work pretty easily but it was very much the wrong movement. Planets would shoot toward the center sun and fly away off screen within less than 1000 steps of the simulation.



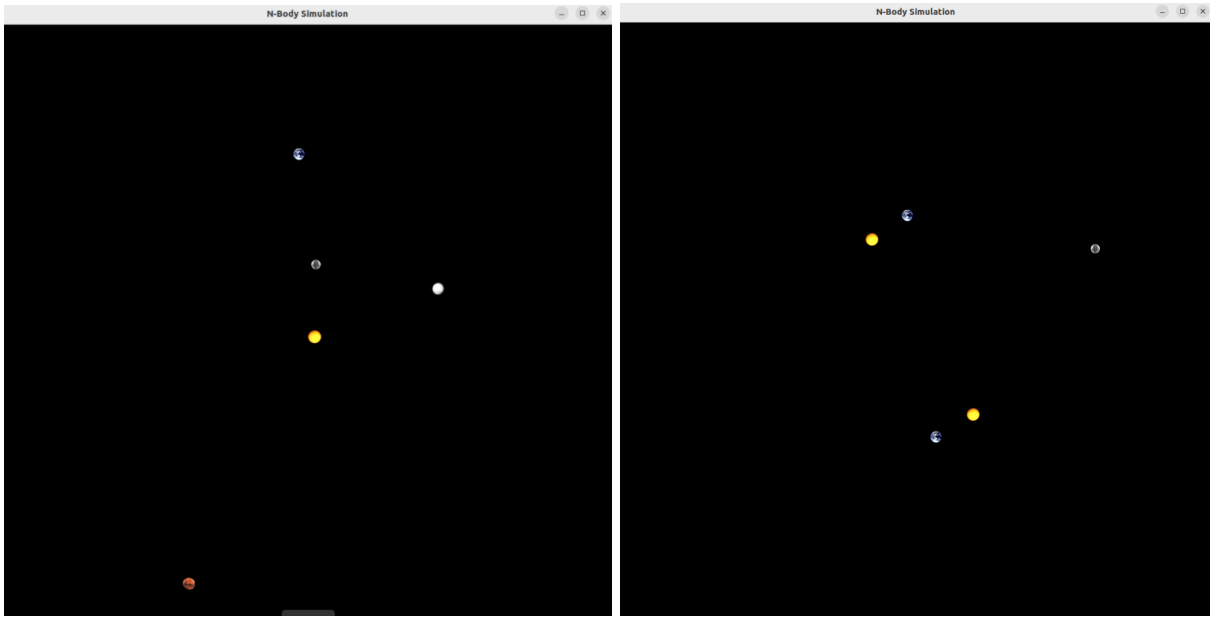
I spent a really long time reviewing the formulas and rewriting them thinking they were the problem. Initially I had found some issues but the same behavior of planets flying towards the center was still occurring. The problem actually was a scoping issue. I had one variable that would add up the net forces acting on one body, that variable was defined outside of the loop that iterates through the rest of the planets. So the variable would calculate the correct position for the first step of the first planet but the next planet would also add onto the net force of the last planet which was the cause of the behavior in my program. Once I added the variable inside the loop, so that it would be redefined each iteration, the simulation worked.

This might have been avoidable if I had chosen to have the calculation done in the CelestialBody class. This would've avoided any circumstance that planets shared calculations.

The tests for this were also difficult to fulfill because of the precision and some more varying factors. I wasn't sure how much time to simulate or delta time between steps there was supposed to be in order to compare with a correct implementation. Either way, my implementation would get the calculations for a year correct, but not the first step calculation. It's strange because I would've figured that simulation a year of time would mean that my calculation would be spot on for other time periods, but for shorter time periods my solution seems to struggle with accuracy.

This means that the UI for my Universe would be imperceptibly different from a correct implementation before the first few steps; however, without looking at the numbers you could never tell the difference.

Here are some frames from 2 different Universes:




## 5.5 What I Learned part 2

I definitely will be double checking where I have variables defined if issues come up in the future. It was not a thing I checked for before this project but it single handedly cost hours of time trying to find the problem. Even proactively looking at variables and deciding where what scope it should be defined in would've been better than not at all. My problem was I knew all the variables I needed to calculate net forces but put the outside of the scope that was going to be using them so each iteration for each CelestialBody took the calculations from the last one and added onto it.

For the future as well, I will have to invest more time into getting smart pointers to work within my program because using them from the start would've helped when it came to object lifetime management.

## 5.6 Codebase part 2

```
Makefile
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
```

```
test_framework
4: OBJ = main.o Universe.o CelestialBody.o
5:
6: .PHONY: all clean pdf lint
7:
8: all: NBody test NBody.a
9:
10: NBody : main.o Universe.o CelestialBody.o
11: $(CC) $(CFLAGS) -o NBody main.o Universe.o CelestialBody.o $(LIB)
12:
13: test : test.o Universe.o CelestialBody.o
14: $(CC) $(CFLAGS) -o test test.o Universe.o CelestialBody.o $(LIB)
15:
16: main.o : main.cpp Universe.hpp CelestialBody.hpp
17: $(CC) $(CFLAGS) -c main.cpp
18:
19: Universe.o : Universe.cpp Universe.hpp
20: $(CC) $(CFLAGS) -c Universe.cpp
21:
22: CelestialBody.o : CelestialBody.cpp CelestialBody.hpp
23: $(CC) $(CFLAGS) -c CelestialBody.cpp
24:
25: test.o : test.cpp Universe.hpp CelestialBody.hpp
26: $(CC) $(CFLAGS) -c test.cpp
27:
28: clean:
29: rm *.o NBody test
30:
31: lint:
32: cpplint *.cpp *.hpp
33:
34: pdf:
35: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
36: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
37: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
38:
39: NBody.a:
40: ar rcs NBody.a Universe.o CelestialBody.o
41:
42:
```

```
main.cpp
```

```cpp
// Copyright 2024
// By Steven Mellett
// main.cpp for ps4a
// updated 3/18/2024


#include <iostream>
#include <istream>
#include <fstream>
#include <memory>
#include <string>
#include "Universe.hpp"


using NB::Universe;
using NB::CelestialBody;

int main(int argc, char* argv[]) {
sf::RenderWindow window(sf::VideoMode(1000, 1000), "N-Body
Simulation");
sf::Event event;
Universe uni;
CelestialBody planet;
if ( argc <= 1 ) {
// std::cout << "arg less than or equal to 1" << std::endl;
uni.set_time(100000000, 10000);
} else {
uni.set_time(std::stod(argv[1]), std::stod(argv[2]));
std::cout << argv[1] << " " << argv[2] << std::endl;
}
std::cin >> uni;
std::cout << uni << std::endl;
while ( window.isOpen() ) {
while ( window.pollEvent(event) ) {
if ( event.type == sf::Event::Closed ) {
window.close();
}
}
window.clear();
for ( int i = 0; i < uni.planets(); i++ ) {
window.draw(uni.get_planet(i));
// std::cout << uni.get_planet(i);
}
```

```
43: window.display();
44: // std::cout << uni.get_time() << std::endl;
45: if ( uni.get_time() > 0 ) {
46: uni.step(uni.get_dtime());
47: }
48: }
49: std::cout << uni << std::endl;
50: return 0;
51: }
```

Universe.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // Universe.hp for ps4a
4: // updated 3/18/2024
5: #ifndef UNIVERSE_HPP
6: #define UNIVERSE_HPP
7: #include <vector>
8: #include <istream>
9: #include <iostream>
10: #include <memory>
11: #include "CelestialBody.hpp"
12: #include <SFML/Graphics.hpp>
13: namespace NB {
14:
15: class Universe : public sf::Drawable {
16: public:
17: Universe();
18: int planets() const { return num_planets; }
19: CelestialBody get_planet(int i) const { return bodies.at(i); }
20: double get_size() const { return size; }
21: int find_farthest_planet();
22: void place_planets();
23: CelestialBody get_body(int i) const { return bodies.at(i); }
24: void step(double seconds);
25: void print_vel();
26: void set_time(double Time, double det) { T = Time; dt = det; }
27: void calculate_forces();
28: double get_time() { return T; }
29: double get_dtime() { return dt; }
30:
31: private:
```

```
32: double T;
33: double dt;
34: int num_planets = 0;
35: double size;
36: std::vector<std::shared_ptr<sf::Vector2<double>>> forces;
37: std::vector<CelestialBody> bodies;
38:
39: friend std::istream& operator>>(std::istream& in, Universe& uni);
40: friend std::ostream& operator<<(std::ostream& out, Universe& uni);
41:
42: protected:
43: virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const
{
44: for ( int i = 0; i < num_planets; i++ ) {
45: target.draw(bodies.at(i).get_sprite());
46: }
47: }
48: };
49: } // namespace NB
50:
51: #endif
```

```
Universe.cpp
1: // Copyright 2024
2: // By Steven Mellett
3: // Universe.cpp for ps4a
4: // updated 3/18/2024
5:
6: #include <iostream>
7: #include <istream>
8: #include <string>
9: #include <memory>
10: #include <cmath>
11: #include "Universe.hpp"
12: #include "CelestialBody.hpp"
13:
14:
15:
16: namespace NB {
17:
18: Universe::Universe() {
```

```
19: num_planets = 0;
20: }
21: int Universe::find_farthest_planet() {
22: int index = 0;
23: for ( int i = 0; i < num_planets - 1; i++ ) {
24: if ( abs(bodies.at(index).get_xpos()) < abs(bodies.at(i + 1).get_xpos(
)) ) {
25: index = i + 1;
26: }
27: }
28: return index;
29: }
30: void Universe::place_planets() {
31: float x;
32: float y;
33: for ( int i = 0; i < num_planets; i++ ) {
34: bodies.at(i).load_texture();
35: x = static_cast<float>((1000 * (bodies.at(i).get_xpos() / (size*2))) +
500);
36: y = static_cast<float>((1000 * (bodies.at(i).get_ypos() / (size*2))) +
500);
37: bodies.at(i).move_sprite(x, y);
38: }
39: }
40: std::istream& operator>>(std::istream& in, Universe& uni) {
41: char line[256];
42: in.getline(line, 10);
43: int num = std::stoi(line, nullptr, 10);
44: uni.num_planets = num;
45: in >> uni.size;
46: for ( int i = 0; i < uni.num_planets; i++ ) {
47: CelestialBody planet;
48: in >> planet;
49: uni.bodies.push_back(planet);
50: uni.forces.push_back(std::make_shared<sf::Vector2<double>>());
51: uni.bodies.at(i).set_force(uni.forces.at(i));
52: }
53: uni.place_planets();
54: return in;
55: }
56: std::ostream& operator<<(std::ostream& out, Universe& uni) {
57: out << uni.num_planets << std::endl;
58: out << uni.size << std::endl;
```

```
59: for ( int i = 0; i < uni.num_planets; i++ ) {
60: out << uni.bodies.at(i);
61: }
62: return out;
63: }
64: void Universe::print_vel() {
65: for ( int i = 0; i < num_planets; i++ ) {
66: std::cout << i << std::endl;
67: std::cout << forces.at(i)->x << std::endl;
68: std::cout << forces.at(i)->y << std::endl;
69: }
70: }
71: void Universe::step(double seconds) {
72: float x;
73: float y;
74: float rx;
75: float ry;
76: // std::cout << "STEP" << std::endl;
77: calculate_forces();
78: for ( int i = 0; i < num_planets; i++ ) {
79: x = static_cast<float>((bodies.at(i).get_xpos() + (seconds * bodies.at
(i).get_xvel())));
80: y = static_cast<float>((bodies.at(i).get_ypos() + (seconds * -bodies.a
t(i).get_yvel())));
81: rx = static_cast<float>(1000 * (x / (size * 2)));
82: ry = static_cast<float>(1000 * (y / (size * 2)));
83: rx += 500;
84: ry += 500;
85: bodies.at(i).move_sprite(rx , ry);
86: bodies.at(i).change_pos(x, y);
87: }
88: T -= dt;
89: }
90: void Universe::calculate_forces() {
91: double grav_const = 6.67e-11;
92: for ( int i = 0; i < num_planets; i++ ) {
93: double force_x = 0;
94: double force_y = 0;
95: for ( int w = 0; w < num_planets; w++ ) {
96: if ( w != i ) {
97: double dx = bodies.at(w).get_xpos() - bodies.at(i).get_xpos();
98: double dy = bodies.at(w).get_ypos() - bodies.at(i).get_ypos();
99: double r = sqrt(dx*dx + dy*dy);
```

```
100: double m1 = bodies.at(i).get_mass();
101: double m2 = bodies.at(w).get_mass();
102: double force = (grav_const * m1 * m2) /(r * r);
103: // std::cout << "Force y == " << (force * dy)/r << std::endl;
104: force_x += (force * (dx / r));
105: force_y += (force * (dy / r));
106: // forces.at(i)->x += (force * (dx / r));
107: // forces.at(i)->y += (force * (dy / r));
108: }
109: }
110: // now acceleration
111: double ax = force_x/(bodies.at(i).get_mass());
112: double ay = force_y/(bodies.at(i).get_mass());
113: bodies.at(i).change_xvel(bodies.at(i).get_xvel() + (dt * ax));
114: bodies.at(i).change_yvel(bodies.at(i).get_yvel() - (dt * ay));
115: }
116: }
117: } // namespace NB
```

CelestialBody.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // CelestialBody.hpp for ps4a
4: // updated 3/18/2024
5: #ifndef CELESTIALBODY_HPP
6: #define CELESTIALBODY_HPP
7: #include <string>
8: #include <memory>
9: #include <iostream>
10: #include <SFML/Graphics.hpp>
11: #include <SFML/Graphics/Drawable.hpp>
12: #include <SFML/Graphics/Texture.hpp>
13: #include <SFML/Graphics/Sprite.hpp>
14:
15: namespace NB {
16:
17: class CelestialBody : public sf::Drawable {
18: public:
19: CelestialBody();
20: double get_xpos() const { return xpos; }
21: double get_ypos() const { return ypos; }
22: double get_xvel() const { return xvel; }
```

```
23: double get_yvel() const { return yvel; }
24: double get_mass() const { return mass; }
25: void change_xvel(double x) { xvel = x; }
26: void change_yvel(double y) { yvel = y; }
27: std::string get_name() const { return FILE_NAME; }
28: void move_sprite(float x, float y) { sprite.setPosition(x, y); }
29: void load_texture() { texture.loadFromFile(FILE_NAME);
sprite.setTexture(t
exture); }
30: sf::Sprite get_sprite() const { return sprite; }
31: void set_force(std::shared_ptr<sf::Vector2<double>> v);
32: void change_speed(double x, double y) { force->x = x; force->y = y;
xvel =
x; yvel = y; }
33: void change_pos(double x, double y) { xpos = x; ypos = y; }
34:
35: private:
36: std::shared_ptr<sf::Vector2<double>> force;
37: double xpos;
38: double ypos;
39: double xvel;
40: double yvel;
41: double mass;
42: std::string FILE_NAME;
43: sf::Sprite sprite;
44: sf::Texture texture;
45: friend std::istream& operator>>(std::istream& in, CelestialBody&
planet);
46: friend std::ostream& operator<<(std::ostream& out, CelestialBody&
planet);
47:
48: protected:
49: virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const
{
50: target.draw(sprite, states);
51: }
52: };
53: } // namespace NB
54:
55: #endif
```

CelestialBody.cpp

```cpp
// Copyright 2024
// By Steven Mellett
// CelestialBody.cpp for ps4a
// updated 3/18/2024
#include <iostream>
#include <iomanip>
#include <memory>
#include "CelestialBody.hpp"
#include "Universe.hpp"

namespace NB {
CelestialBody::CelestialBody() {}
std::istream& operator>>(std::istream& in, CelestialBody& planet) {
in >> planet.xpos;
in >> planet.ypos;
in >> planet.xvel;
in >> planet.yvel;
in >> planet.mass;
in >> planet.FILE_NAME;
return in;
}
std::ostream& operator<<(std::ostream& out, CelestialBody& planet) {
out << " " << std::scientific << planet.get_xpos() << " ";
out << planet.get_ypos() << " " << planet.get_xvel() << " ";
out << planet.get_yvel() << " " << planet.get_mass() << " ";
out << std::setw(14) << std::right << planet.get_name() << std::endl;
return out;
}
void CelestialBody::set_force(std::shared_ptr<sf::Vector2<double>> v) {
force = v;
force->y = yvel;
force->x = xvel;
}
} // namespace NB
```

test.cpp

```cpp
// Copyright 2024
// By Steven Mellett
// test.cpp for ps4a
#include <iostream>
#include <string>
```

```
6: #include <fstream>
7: #include "Universe.hpp"
8: #include "CelestialBody.hpp"
9: #define BOOST_TEST_DYN_LINK
10: #define BOOST_TEST_MODULE Main
11: #include <boost/test/unit_test.hpp>
12: using NB::Universe;
13:
14: BOOST_AUTO_TEST_CASE(test_correct_size) {
15: std::string FILE_NAME;
16: FILE_NAME = "planets.txt";
17: Universe uni;
18: std::ifstream level_file(FILE_NAME);
19: level_file >> uni;
20: level_file.close();
21: BOOST_REQUIRE_EQUAL(uni.get_size(), static_cast<double>(2.5e+11));
22: }
23:
24: BOOST_AUTO_TEST_CASE(test_number_planets) {
25: std::string FILE_NAME;
26: FILE_NAME = "planets.txt";
27: Universe uni;
28: std::ifstream level_file(FILE_NAME);
29: level_file >> uni;
30: level_file.close();
31: BOOST_REQUIRE_EQUAL(uni.planets(), 5);
32: }
33: BOOST_AUTO_TEST_CASE(test_planet_xvel_zero) {
34: std::string FILE_NAME;
35: FILE_NAME = "planets.txt";
36: Universe uni;
37: std::ifstream level_file(FILE_NAME);
38: level_file >> uni;
39: level_file.close();
40: for ( int i = 0; i < uni.planets(); i++ ) {
41: BOOST_REQUIRE_EQUAL(uni.get_body(i).get_xvel(), 0);
42: }
43: }
44: BOOST_AUTO_TEST_CASE(test_planet_mass_non_zero) {
45: std::string FILE_NAME;
46: FILE_NAME = "planets.txt";
47: Universe uni;
48: std::ifstream level_file(FILE_NAME);
```

```
49: level_file >> uni;
50: level_file.close();
51: for ( int i = 0; i < uni.planets(); i++ ) {
52: BOOST_REQUIRE(uni.get_body(i).get_mass() > 0);
53: }
54: }
```

# 6 PS5: DNA Sequence Alignment

## 6.1 Discussion

The concepts behind this project were very familiar to me because I had taken an AP biology course in high school and the teacher had access to similar software from previous work she did. So going into this I was very confident in the concepts; however, it turns out this was my least successful project yet.

The idea behind it is pretty easy to do out on paper but very hard to get exactly correct in code because of the recursive solution I chose to pursue. In order to get a comparison that had the least cost I had to assemble a table of values that represented 1 of either 3 options:

There was a match - costs 0.
There wasn't a match - costs 1.
There's a gap inserted - costs 2.

Each point in the table has coordinates of (x, y) where x is a character in one DNA string and y is another. I chose to represent this table as two vectors because the syntax .at().at() I am very familiar with this.

The table of values cumulatively holds the data to find the lowest cost alignment, also known as edit distance.

Once the table is made through the recursive function there is a clever way to iterate back down and find the path of lowest edit distance. Looking at one point on the table and comparing it to points around it as well as whether or not the

coordinates x and y are the same allows you to infer whether or not there's a match, mismatch, or gap in the lowest possible solution.

Here's an example of a table that's created and the alignment you can get from it:

```
smellett@Ubuntu:~/CompIV/DNA/ps5$ ./EDistance
AACAGTTACC
TAAGGTCA
Edit distance = 7
First String: AACAGTTACC
Second String: TAAGGTCA
Matrix:
  7   8  10  12  13  15  16  18  20
  6   6   8  10  11  13  14  16  18
  6   5   6   8   9  11  12  14  16
  7   5   4   6   7   9  11  12  14
  9   7   5   4   5   7   9  10  12
  8   8   6   4   4   5   7   8  10
  9   8   7   5   3   3   5   6   8
 11   9   7   6   4   2   3   4   6
 13  11   9   7   5   3   1   3   4
 14  12  10   8   6   4   2   1   2
 16  14  12  10   8   6   4   2   0

A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
```

From the PDF for the assignment the same two strings produce:

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | T | A | A | G | G | T | C | A | - |
| 0 | A | 7 | 8 | 10 | 12 | 13 | 15 | 16 | 18 | 20 |
| 1 | A | 6 | 6 | 8 | 10 | 11 | 13 | 14 | 16 | 18 |
| 2 | C | 6 | 5 | 6 | 8 | 9 | 11 | 12 | 14 | 16 |
| 3 | A | 7 | 5 | 4 | 6 | 7 | 9 | 11 | 12 | 14 |
| 4 | G | 9 | 7 | 5 | 4 | 5 | 7 | 9 | 10 | 12 |
| 5 | T | 8 | 8 | 6 | 4 | 4 | 5 | 7 | 8 | 10 |
| 6 | T | 9 | 8 | 7 | 5 | 3 | 3 | 5 | 6 | 8 |
| 7 | A | 11 | 9 | 7 | 6 | 4 | 2 | 3 | 4 | 6 |
| 8 | C | 13 | 11 | 9 | 7 | 5 | 3 | 1 | 3 | 4 |
| 9 | C | 14 | 12 | 10 | 8 | 6 | 4 | 2 | 1 | 2 |
| 10 | - | 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 | 0 |

The red outlines the behavior of how the retrieve alignment function is supposed to work. The top left-most box at (0,0) is the edit distance of the best case alignment.

There are a few problems with my own solution but in general using a recursive function would not work for massive DNA sequences. The solution to this is to

implement a dynamic approach to creating the table where values are saved after being calculated once by the function. This is not required but was still outside of my capabilities when implementing my solution.

Other than the general issue with recursion being slow for large data sets, my program specifically does not work with a lot of edge cases that come up in larger and larger data sets. From my understanding my program can't correctly compute edit distance alignment with more than one gap in a row. I have narrowed the problem specifically to the alignment retrieving function because my program does get the edit distance correctly in most cases, even for larger data sets. One of the tests this program passes is correctly identifying edit distances for a 10,000 character long DNA sequence of two E. coli bacteria. Yet, the alignment function fails most alignments.

### 6.2 What I Learned

From this project it reinforced to me the importance of the order you chose to recurse in. I had an issue where I was getting a 2 in the top left of the matrix, this was caused because I called the recursive function for an aligned character before the two cases where a gap or mismatch occurs. I spent a while trying to figure out what the problem with my function was but it was just the order it was being called in from the beginning.

Another thing I will avoid doing in the future is only testing my program on the easy example data sets provided. I ended up only testing my program with a few short DNA strings and from that I concluded my program was completely done, but I realized when trying to run the speed assessment my implementation had some major flaws.

### 6.3 Codebase

```
Makefile
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
test_framework
```

```
4: OBJ = main.o EDistance.o
5:
6: .PHONY: all clean pdf lint
7:
8: all: EDistance test EDistance.a
9:
10: EDistance : main.o EDistance.o
11: $(CC) $(CFLAGS) -o EDistance main.o EDistance.o $(LIB)
12:
13: test : test.o EDistance.o
14: $(CC) $(CFLAGS) -o test test.o EDistance.o $(LIB)
15:
16: main.o : main.cpp EDistance.hpp
17: $(CC) $(CFLAGS) -c main.cpp
18:
19: EDistance.o : EDistance.cpp EDistance.hpp
20: $(CC) $(CFLAGS) -c EDistance.cpp
21:
22: test.o : test.cpp EDistance.hpp
23: $(CC) $(CFLAGS) -c test.cpp
24:
25: clean:
26: rm *.o EDistance test
27:
28: lint:
29: cpplint *.cpp *.hpp
30:
31: pdf:
32: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
33: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
34: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
35:
36: EDistance.a:
37: ar rcs EDistance.a EDistance.o
38:
39:
```

```
main.cpp
1: // Copyright 2024
2: // By Steven Mellett
3: // main.cpp for ps5
4: // updated 4/1/2024
```

```
5:
6: #include <string>
7: #include <iostream>
8: #include "EDistance.hpp"
9:
10: int main(int argc, char* argv[]) {
11: std::string string1;
12: std::string string2;
13: std::cin >> string1;
14: std::cin >> string2;
15: EDistance e(string1, string2);
16: // e.Print_Strings(std::cout);
17: std::cout << "Edit distance = " << e.optDistance() << std::endl;
18: // e.Print_Strings(std::cout);
19: // std::cout << "Alignment: " << std::endl;
20: std::cout << e.alignment() << std::endl;
21: return 0;
22: }
```

EDistance.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // EDistance.hpp for ps5
4: // updated 4/1/2024
5: #ifndef EDISTANCE_HPP
6: #define EDISTANCE_HPP
7:
8: #include <vector>
9: #include <iostream>
10: #include <string>
11:
12: class EDistance {
13: public:
14: EDistance();
15: EDistance(std::string string1, std::string string2);
16:
17: void Print_Strings(std::ostream& o);
18: int optDistance();
19: std::string alignment();
20: static int penalty(char a, char b) { if ( a == b ) { return 0; } else {
re
turn 1; } }
```

```
21: static int min3(int a, int b, int c);
22: private:
23: std::vector<char> x;
24: std::vector<char> y;
25: std::vector<std::vector<int>> matrix;
26: int edit_distance;
27: int optDistance2(int i, int j);
28: };
29: #endif
```

EDistance.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // EDistance.cpp for ps5
4: // updated 4/1/2024
5:
6: #include "EDistance.hpp"
7: #include <vector>
8: #include <string>
9: #include <iostream>
10: #include <iomanip>
11:
12: EDistance::EDistance() {
13: std::string _x = "AACAGTTACC";
14: std::string _y = "TAAGGTCA";
15: for ( unsigned int i = 0; i < _x.size(); i++ ) {
16: x.push_back(_x.at(i));
17: }
18: for ( unsigned int i = 0; i < _y.size(); i++ ) {
19: y.push_back(_y.at(i));
20: }
21: for ( unsigned int i = 0; i <= x.size(); i++ ) {
22: matrix.push_back(std::vector<int>());
23: for ( unsigned int w = 0; w <= y.size(); w++ ) {
24: matrix.at(i).push_back(-1);
25: }
26: }
27: }
28: EDistance::EDistance(std::string string1, std::string string2) {
29: for ( unsigned int i = 0; i < string1.size(); i++ ) {
30: x.push_back(string1.at(i));
31: }
```

```
32: for ( unsigned int i = 0; i < string2.size(); i++ ) {
33: y.push_back(string2.at(i));
34: }
35: for ( unsigned int i = 0; i <= x.size(); i++ ) {
36: matrix.push_back(std::vector<int>());
37: for ( unsigned int w = 0; w <= y.size(); w++ ) {
38: matrix.at(i).push_back(-1);
39: }
40: }
41: }
42: void EDistance::Print_Strings(std::ostream& o) {
43: o << "First String: ";
44: for ( char c : x ) {
45: o << c;
46: }
47: o << std::endl;
48: o << "Second String: ";
49: for ( char w : y ) {
50: o << w;
51: }
52: o << std::endl;
53: o << "Matrix: " << std::endl;
54: for ( std::vector<int> v : matrix ) {
55: for ( unsigned int i = 0; i < v.size(); i++ ) {
56: o << std::setw(3)<< v.at(i) << " ";
57: }
58: o << std::endl;
59: }
60: o << std::endl;
61: }
62: int EDistance::min3(int a, int b, int c) {
63: int smallest = a;
64:
65: if ( b < smallest ) {
66: smallest = b;
67: }
68: if ( c < smallest ) {
69: smallest = c;
70: }
71: return smallest;
72: }
73: int EDistance::optDistance() {
74: return optDistance2(0, 0);
```

```
75: }
76: int EDistance::optDistance2(int i, int j) {
77: // std::cout << "recursion time" << std::endl;
78: if ( matrix.at(i).at(j) != -1 ) {
79: return matrix.at(i).at(j);
80: }
81: if ( i >= static_cast<int>(x.size()) && j >= static_cast<int>(y.size())
)
{
82: matrix.at(i).at(j) = 0;
83: return 0;
84: } else if ( i >= static_cast<int>(x.size()) ) {
85: matrix.at(i).at(j) = matrix.at(i).at(j + 1) + 2;
86: return matrix.at(i).at(j);
87: } else if ( j >= static_cast<int>(y.size()) ) {
88: // std::cout << i << " " << j << std::endl;
89: matrix.at(i).at(j) = matrix.at(i + 1).at(j) + 2;
90: return matrix.at(i).at(j);
91: }
92: if ( i == 0 && j == 0 ) {
93: optDistance2(static_cast<int>(x.size()), static_cast<int>(y.size()));
94: int i = static_cast<int>(x.size());
95: for ( std::vector<int> v : matrix ) {
96: optDistance2(i, static_cast<int>(y.size()));
97: i--;
98: }
99: }
100: matrix.at(i).at(j) = min3(optDistance2(i, j + 1) + 2, optDistance2(i +
1,
j) + 2,
101: optDistance2(i + 1 , j + 1) + penalty(x.at(i), y.at(j)));
102: return matrix.at(i).at(j);
103: }
104: std::string EDistance::alignment() {
105: std::string ment;
106: int i = 0;
107: int j = 0;
108: while ( matrix.at(i).at(j) != 0 ) {
109: if ( x.at(i) == y.at(j) ) {
110: ment.push_back(x.at(i));
111: ment.push_back(' ');
112: ment.push_back(y.at(j));
113: ment.push_back(' ');
```

```
114: ment.push_back('0');
115: ment.push_back('\n');
116: i++;
117: j++;
118: } else if ( matrix.at(i).at(j) == matrix.at(i + 1).at(j + 1) + 1 ) {
119: ment.push_back(x.at(i));
120: ment.push_back(' ');
121: ment.push_back(y.at(j));
122: ment.push_back(' ');
123: ment.push_back('1');
124: ment.push_back('\n');
125: i++;
126: j++;
127: } else if ( matrix.at(i).at(j) == matrix.at(i + 1).at(j) + 2 ) {
128: ment.push_back(x.at(i));
129: ment.push_back(' ');
130: ment.push_back('-');
131: ment.push_back(' ');
132: ment.push_back('2');
133: ment.push_back('\n');
134: i++;
135: } else if ( matrix.at(i).at(j) == matrix.at(i).at(j + 1) + 2 ) {
136: ment.push_back('-');
137: ment.push_back(' ');
138: ment.push_back(y.at(j));
139: ment.push_back(' ');
140: ment.push_back('2');
141: ment.push_back('\n');
142: j++;
143: }
144: }
145: return ment;
146: }
```

test.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // test.cpp for ps4a
4: #include <iostream>
5: #include <string>
6: #include <fstream>
7: #include "EDistance.hpp"
```

```
8: #define BOOST_TEST_DYN_LINK
9: #define BOOST_TEST_MODULE Main
10: #include <boost/test/unit_test.hpp>
11: BOOST_AUTO_TEST_CASE(test_correct_ed) {
12: EDistance e("AACAGTTACC", "TAAGGTCA");
13: BOOST_REQUIRE_EQUAL(e.optDistance(), 7);
14: }
15: BOOST_AUTO_TEST_CASE(test_correct_align) {
16: EDistance e("AACAGTTACC", "TAAGGTCA");
17: e.optDistance();
18: BOOST_REQUIRE_EQUAL(e.alignment(), "A T 1\nA A 0\nC - 2\nA A 0\nG G
0\nT G
1\nT T 0\nA - 2\nC C 0\nC A 1\n");
19: }
20: BOOST_AUTO_TEST_CASE(test_min) {
21: EDistance e("AA", "BB");
22: BOOST_REQUIRE_EQUAL(e.min3(1, 2, 3), 1);
23: }
24: BOOST_AUTO_TEST_CASE(test_penalty) {
25: EDistance e("AA", "BB");
26: BOOST_REQUIRE_EQUAL(e.penalty('A', 'G'), 1);
27: }
28: BOOST_AUTO_TEST_CASE(test_no_penalty) {
29: EDistance e("AA", "BB");
30: BOOST_REQUIRE_EQUAL(e.penalty('A', 'A'), 0);
31: }
```

# 7 PS6: Random Writer

## 7.1 Discussion

This was my favorite project so far. The implementation was required to follow this class implementation:

```
class RandWriter {
 public:
    // Create a Markov model of order k from given text
    // Assume that text has length at least k.
    RandWriter(const string& text, size_t k);

    size_t  orderK() const; // Order k of Markov model

    // Number of occurences of kgram in text
    // Throw an exception if kgram is not length k
    int freq(const string& kgram) const;
    // Number of times that character c follows kgram
    // if order=0, return num of times that char c appears
    // (throw an exception if kgram is not of length k)
    int freq(const string& kgram, char c) const;

    // Random character following given kgram
    // (throw an exception if kgram is not of length k)
    // (throw an exception if no such kgram)
    char kRand(const string& kgram);

    // Generate a string of length L characters by simulating a trajectory
    // through the corresponding Markov chain.  The first k characters of
    // the newly generated string should be the argument kgram.
    // Throw an exception if kgram is not of length k.
    // Assume that L is at least k
    string generate(const string& kgram, size_t L);
};
// Overload the stream insertion operator << and display the internal state
// of the Markov model.  Print out the order, alphabet, and the frequencies
// of the k-grams and k+1-grams
```

To my understanding this is similar to how chatGPT operates on a much simplified and smaller scale.

This project is about taking in a large body of text, such as a book or novel, then storing every occurrence of a letter after each “kgram” in order to later randomly generate text based on the ratios of all the letters that have occurred after a starting string of size k.

A “kgram” is just a section/string of text that is k characters long.

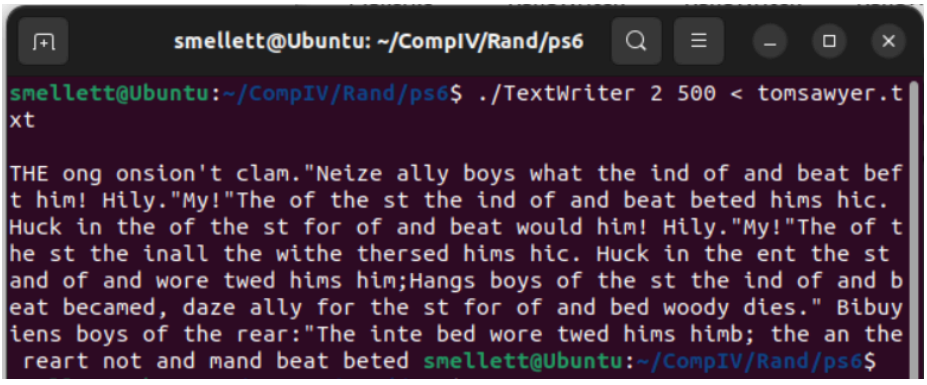
To store all of this data it wouldn't be fast enough to use a vector or similar 1-dimensional array because the time complexity of looking up a certain kgram in would surpass $O(n^2)$ because you also have to find the occurrences of every letter that has happened before it. Maybe a binary search tree would suffice however there is a better structure.

I chose to use the std::map structure because of many factors. It is faster than a vector because of the O(log n) look up time because it's implemented as a binary search tree but on top of that you can use the kgram as a key into the map. The only problem left is how to store what letter has what occurrence.

I thought about storing them in alphabetical order so every new kgram had 26 (or more for special characters) values associated with it, but this seemed more difficult than the solution I decided on. I defined a pair struct that held a char and an integer. All in all, the map took in a kgram as a key and returned a vector of pair structs. This made it super easy to iterate through an entire text character by character in k-sized chunks checking inside the map to see if the next character in the text had already occurred and if not to make another pair struct.

Once I had that done I used some random number generator functions provided by <random> as well as a cool std::uniform_int_distribution function in order to pick from the vector of pair structs based on the amount they occurred in the input text. The value picked would be sent to stdout and that would also be used for the new kgram to generate the next character.

Using the entire "The Adventures of Tom Sawyer" book with kgrams of size 2 to generate 500 characters looks like:



There are a few issues that come up when the input text is not big enough to have every possible combination that can come up for words. This becomes more apparent when using kgrams that are more than 3 or 4. My solution to this was to just pick a random letter if this occurs, but a better solution would be to pick based on the ratio of letters in the actual text.

### 7.2 What I Learned

This was the first program I used a map as an integral part for. With some help I have implemented a hashmap in C before so with that in the back of my mind using the map was very easy. Defining and using a struct isn't something I do all the time so it was nice to refresh how that works and how handy it is.

Using the <random> library was also very different from how C uses <random.h>, the functions were harder to get working with 0 knowledge but reading the documentation helped me figure it out.

Here is the random class with the seed based on system clock I used:

```
84   auto seed = std::chrono::system_clock::now().time_since_epoch().count();
85   std::linear_congruential_engine<unsigned int, 16807, 0, 2147483647> e(seed);
86   std::uniform_int_distribution<int> dis(0, 99);
```

### 7.3 Codebase

Makefile

```
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
-lboost_unit_
test_framework
4: OBJ = main.o RandWriter.o
5:
6: .PHONY: all clean pdf lint
7:
8: all: TextWriter test TextWriter.a
9:
10: TextWriter : TextWriter.o RandWriter.o
11: $(CC) $(CFLAGS) -o TextWriter TextWriter.o RandWriter.o $(LIB)
12:
13: test : test.o RandWriter.o
```

```
14: $(CC) $(CFLAGS) -o test test.o RandWriter.o $(LIB)
15:
16: TextWriter.o : TextWriter.cpp RandWriter.hpp
17: $(CC) $(CFLAGS) -c TextWriter.cpp
18:
19: RandWriter.o : RandWriter.cpp RandWriter.hpp
20: $(CC) $(CFLAGS) -c RandWriter.cpp
21:
22: test.o : test.cpp RandWriter.hpp
23: $(CC) $(CFLAGS) -c test.cpp
24:
25: clean:
26: rm *.o TextWriter test
27:
28: lint:
29: cpplint *.cpp *.hpp
30:
31: pdf:
32: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
33: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
34: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
35:
36: TextWriter.a:
37: ar rcs TextWriter.a RandWriter.o
38:
39:
```

TextWriter.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // main.cpp for ps6
4: // updated 4/8/2024
5: #include <string>
6: #include <iostream>
7: #include "RandWriter.hpp"
8: using std::string;
9:
10: int main(int argc, char* argv[]) {
11: int k = 2;
12: int L = 100;
13: string s;
14: string line;
```

```
15: string s2;
16: char temp;
17: if ( argc <= 1 ) {
18: k = 2;
19: L = 100;
20: } else {
21: k = std::stoi(argv[1]);
22: L = std::stoi(argv[2]);
23: }
24: for ( int i = 0; i < k; i++ ) {
25: std::cin >> temp;
26: s2.push_back(temp);
27: }
28: while ( std::getline(std::cin, line) ) {
29: s += line;
30:
31: if ( std::cin.eof() ) {
32: break;
33: }
34: }
35: RandWriter r(s, k);
36: std::cout << r.generate(s2, L);
37: return 0;
38: }
```

RandWriter.hpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // RandWriter.hpp for ps6
4: // updated 4/8/2024
5:
6: #include <string>
7: #include <map>
8: #include <vector>
9:
10: using std::vector;
11: using std::string;
12: using std::map;
13:
14: struct pair {
15: char c;
16: int count = 0;
```

```
17: };
18:
19: class RandWriter {
20: public:
21: RandWriter(const string& text, size_t k);
22:
23: size_t orderK() const { return _orderK; }
24:
25:
26: int freq(const string& kgram) const;
27:
28: int freq(const string& kgram, char c) const;
29:
30: char kRand(const string& gram);
31:
32: string generate(const string& kgram, size_t L);
33:
34: private:
35: map<string, vector<pair>> m;
36: size_t _orderK;
37: size_t string_size;
38: };
```

RandWriter.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // RandWriter.cpp for ps6
4: // updated 4/8/2024
5:
6: #include "RandWriter.hpp"
7: #include <string>
8: #include <random>
9: #include <chrono>
10: #include <iostream>
11: #include <stdexcept>
12:
13: using std::string;
14:
15: RandWriter::RandWriter(const string& text, size_t k) {
16: _orderK = k;
17: string_size = text.size();
18: // std::cout << text.size() << std::endl;
```

```
19: for (unsigned int i = 0; i < text.size(); i++) {
20: pair p;
21: string c;
22: unsigned int startIndex = i % text.size();
23: for (unsigned int j = 0; j < k; j++) {
24: unsigned int index = (startIndex + j) % text.size();
25: c.push_back(text.at(index));
26: }
27: // std::cout << c << std::endl;
28: if (m[c].size() != 0) {
29: int found = 0;
30: for (unsigned int t = 0; t < m[c].size(); t++) {
31: if (m[c].at(t).c == text.at((startIndex + k) % text.size()))
{
32: m[c].at(t).count++;
33: found = 1;
34: }
35: }
36: if (found == 0) {
37: p.c = text.at((startIndex + k) % text.size());
38: p.count++;
39: m[c].push_back(p);
40: }
41: } else {
42: p.c = text.at((startIndex + k) % text.size());
43: p.count++;
44: m[c].push_back(p);
45: }
46: }
47: }
48: int RandWriter::freq(const string& kgram) const {
49: if ( kgram.size() != _orderK ) {
50: throw std::invalid_argument("Wrong Size kGram");
51: }
52: int total = 0;
53: auto it = m.find(kgram);
54: if ( it == m.end() || it->second.size() == 0 ) {
55: return 0;
56: } else {
57: for ( unsigned int i = 0; i < it->second.size(); i++ ) {
58: total += it->second.at(i).count;
59: }
60: }
```

```
61: return total;
62: }
63:
64: int RandWriter::freq(const string& kgram, char c) const {
65: if ( kgram.size() != _orderK ) {
66: throw std::invalid_argument("Wrong Size kGram");
67: }
68: int total = 0;
69: auto it = m.find(kgram);
70: if ( it->second.size() == 0 ) {
71: return 0;
72: } else {
73: for ( unsigned int i = 0; i < it->second.size(); i++ ) {
74: if ( it->second.at(i).c == c ) {
75: total = it->second.at(i).count;
76: }
77: }
78: }
79: return total;
80: }
81:
82: char RandWriter::kRand(const string& kgram) {
83: // int max = m[kgram].size() - 1;
84: auto seed =
std::chrono::system_clock::now().time_since_epoch().count();
85: std::linear_congruential_engine<unsigned int, 16807, 0, 2147483647>
e(seed
);
86: std::uniform_int_distribution<int> dis(0, 99);
87: std::uniform_int_distribution<int> dis2(97, 122);
88: int random = dis(e);
89: int not_random_fill = dis2(e);
90: try {
91: double total_times = freq(kgram);
92: // std::cout << "Freq of : " << kgram << " is " << freq(kgram) << std::
endl;
93: double percent = 0;
94: for ( unsigned int i = 0; i < m[kgram].size(); i++ ) {
95: percent += (static_cast<double>(m[kgram].at(i).count) / total_times) *
100;
96: // std::cout << (m[kgram].at(i).count / total_times) * 100 << std::en
dl;
97: // std::cout << percent << std::endl;
```

```
98: if ( percent > static_cast<double>(random) ) {
99: return m[kgram].at(i).c;
100: }
101: }
102: } catch (const std::invalid_argument& e) {
103: std::cerr << "Invalid Argument Error: " << e.what() << std::endl;
104: }
105: return not_random_fill; // error letter : when the starter kgram
106: // is not it in the input text return random letter
107: }
108:
109: string RandWriter::generate(const string& kgram, size_t L) {
110: string random_string;
111: unsigned int i;
112: for ( i = 0; i < _orderK; i++ ) {
113: random_string.push_back(kgram.at(i));
114: }
115: for ( unsigned int j = i; j < L; j++ ) {
116: string new_kgram;
117: for ( unsigned int w = j; w < _orderK + j; w++ ) {
118: new_kgram.push_back(random_string.at(w - _orderK));
119: }
120: random_string.push_back(kRand(new_kgram));
121: }
122: return random_string;
123: }
```

test.cpp

```
1: // Copyright 2024
2: // By Steven Mellett
3: // test.cpp for ps6
4: #include <iostream>
5: #include <string>
6: #include <fstream>
7: #include <stdexcept>
8: #include "RandWriter.hpp"
9: #define BOOST_TEST_DYN_LINK
10: #define BOOST_TEST_MODULE Main
11: #include <boost/test/unit_test.hpp>
12: BOOST_AUTO_TEST_CASE(test_freq_throw) {
13: RandWriter r("test the test case", 3);
14: std::string badkgram = "morethanthree";
```

```
15: BOOST_REQUIRE_THROW(r.freq(badkgram), std::invalid_argument);
16: }
17:
18: BOOST_AUTO_TEST_CASE(test_freq_no_throw) {
19: RandWriter r("test the good case", 3);
20: std::string goodkgram = "the";
21: BOOST_REQUIRE_NO_THROW(r.freq(goodkgram));
22: }
23: BOOST_AUTO_TEST_CASE(test_freq2_throw) {
24: RandWriter r("test differnet freq2", 3);
25: std::string badkgram = "morethanthree";
26: char c = 'c';
27: BOOST_REQUIRE_THROW(r.freq(badkgram, c), std::invalid_argument);
28: }
29: BOOST_AUTO_TEST_CASE(test_planet_mass_non_zero) {
30: RandWriter r("test the good freq2", 3);
31: std::string goodkgram = "the";
32: char c = 'c';
33: BOOST_REQUIRE_NO_THROW(r.freq(goodkgram, c));
34: }
35: BOOST_AUTO_TEST_CASE(test_orderk_returns_correct_value) {
36: RandWriter r("test returns", 3);
37: BOOST_CHECK_EQUAL(r.orderK(), static_cast<size_t>(3));
38: }
39:
```

# 8 PS7: Kronos Log Parsing

## 8.1 Discussion

This was a shorter project for me even though I knew nothing about regular expressions and how they worked before starting this.

The goal was to read through a log file for a Kronos InTouch time clock and retrieve when it booted up and how long it took to boot up.

I looked up and watched a quick youtube video that helped me immensely in this project, here is the link: https://www.youtube.com/watch?v=sXQxhojSdZM

In the video it explains one part that is vital to coming up with the correct regex that fulfills that correct behavior. The “d+” is what I used in these two expressions:

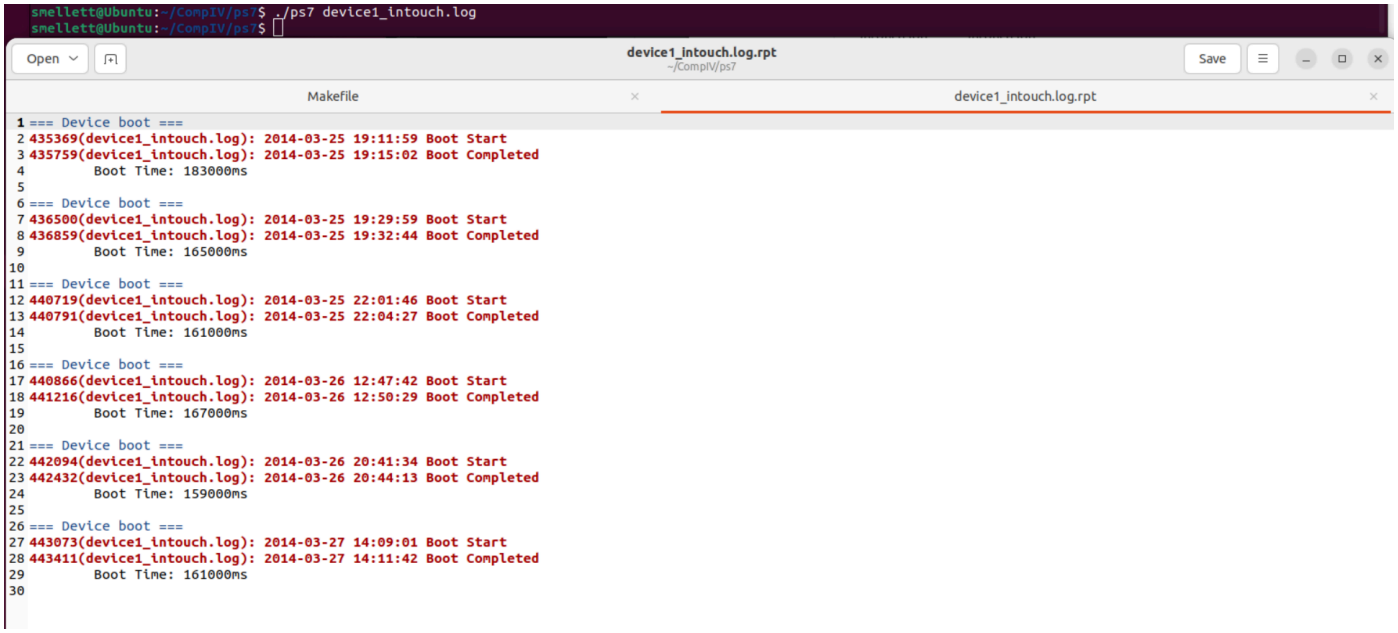
```
73      regex beginBootRegex(
74          R"((\d+)-(\d+)-(\d+) (\d+):(\d+):(\d+): \(log\.c\.166\) server started.*)");
75      regex endBootRegex(
76          R"((\d+)-(\d+)-(\d+) (\d+):(\d+):(\d+)\.(\d+):INFO:oejs\.AbstractConnector:Started SelectChannelConnector@0\.0\.0\.0:9080.*)");
```

It stands for digit and the plus just means to take in the entire number rather than a single character of a number. Once I figured that's how you were supposed to get the date from the logs in the file the rest was simple. This is because we were given the log for boot up and log for boot up confirmation and you could use that to assemble the rest of the regex based off of a few more syntax rules.

Using the Boost library for the regex and smatch object were essential to saving the regex matches you get. The best I can describe what the smatch object is is an array that saves all the important data from the regex_match function. I used it to output the time and calculate how long it took the InTouch device to boot.

Writing to an output file was very familiar, we did a lot of that in Comp II as well as previous projects.

Here it reads through log file named device1_intouch.log and outputs to device1_intouch.log.rpt:



## 7.2 What I Learned

Regex was completely new to me from this project, I had to use a lot of outside resources to get an understanding of what the syntax actually means. Creating the actual expressions was tedious but in the end this is one of my more robust programs.

I also had a problem compiling at first but I realized it was that I did not have the correct libraries in my Makefile.

## 7.3 Codebase

Makefile

```
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3: LIB = -lboost_regex -lboost_date_time
4: OBJ = ps7.o
5:
6: .PHONY: all clean pdf lint
7:
8: all: ps7 ps7.a
9:
10: ps7 : ps7.o
11: $(CC) $(CFLAGS) -o ps7 ps7.o $(LIB)
12:
13: ps7.o : ps7.cpp
14: $(CC) $(CFLAGS) -c ps7.cpp $(LIB)
15:
16: clean:
17: rm *.o ps7
18:
19: lint:
20: cpplint *.cpp *.hpp
21:
22: pdf:
23: enscript -C --margins=50:50:50:50 *.cpp -o *.cpp.ps
24: enscript -C --margins=50:50:50:50 *.hpp -o *.hpp.ps
25: enscript -C --margins=50:50:50:50 Makefile -o Makefile.ps
26:
27: ps7.a:
28: ar rcs ps7.a ps7.o
29:
```

```
30:
31:
```

ps7.cpp

```
1: // Copyright 2024 Steven Mellett
2:
3: #include <exception>
4: #include <iostream>
5: #include <fstream>
6: #include <string>
7: #include <boost/regex.hpp>
8: #include <boost/date_time/gregorian/gregorian.hpp>
9: #include <boost/date_time/posix_time/posix_time.hpp>
10:
11:
12: using boost::regex;
13: using boost::smatch;
14: using std::cout;
15: using std::endl;
16: using std::cin;
17: using std::string;
18: using boost::gregorian::date;
19: using boost::gregorian::years;
20: using boost::gregorian::months;
21: using boost::gregorian::days;
22: using boost::gregorian::from_simple_string;
23: using boost::gregorian::date_duration;
24: using boost::gregorian::date_period;
25: using boost::posix_time::ptime;
26: using boost::posix_time::hours;
27: using boost::posix_time::minutes;
28: using boost::posix_time::seconds;
29: using boost::posix_time::time_duration;
30: using std::cerr;
31: using std::exception;
32: using std::ifstream;
33: using std::runtime_error;
34: using std::ofstream;
35:
36:
37:
38: void processLogFile(const string& filename);
```

```
39:
40: int main(int argc, char* argv[]) {
41: if (argc != 2) {
42: cerr << "Please include one log file name." << endl;
43: return 1;
44: }
45:
46: try {
47: processLogFile(argv[1]);
48: } catch (const exception& e) {
49: cerr << "Error: " << e.what() << endl;
50: return 1;
51: }
52:
53: return 0;
54: }
55:
56: void processLogFile(const string& filename) {
57: ifstream logFile(filename);
58: if (!logFile.is_open()) {
59: throw runtime_error("Cannot open file: " + filename);
60: }
61:

ps7.cpp Tue Apr 16 18:20:30 2024 2
62: ofstream rptFile(filename + ".rpt");
63: if (!rptFile.is_open()) {
64: throw runtime_error("Cannot create report file: " + filename + ".rpt
");
65: }
66:
67: string line;
68: date bootDate;
69: ptime bootStartTime;
70: bool bootInProgress = false;
71: int lines = 1;
72:
73: regex beginBootRegex(
74: R"((\d+)-(\d+)-(\d+) (\d+):(\d+):(\d+): \(log\.c\.166\) server start
ed.*)");
75: regex endBootRegex(
76: R"((\d+)-(\d+)-(\d+) (\d+):(\d+):(\d+)\.(\d+):INFO:oejs\.AbstractCon
nector:Started SelectChannelConnector@0\.0\.0\.0:9080.*)");
```

```
77:
78: while (getline(logFile, line)) {
79: smatch sm;
80: if (regex_match(line, sm, beginBootRegex)) {
81: bootDate = date(stoi(sm[1]), stoi(sm[2]), stoi(sm[3]));
82: bootStartTime = ptime(bootDate, time_duration(stoi(sm[4]), stoi(
sm[5]), stoi(sm[6])));
83: if (bootInProgress) {
84: rptFile << "**** Incomplete boot ****\n\n";
85: }
86: rptFile << "=== Device boot ===\n"
87: << lines << "(" << filename << "): "
88: << sm[1] << "-" << sm[2] << "-" << sm[3] << " "
89: << sm[4] << ":" << sm[5] << ":" << sm[6]
90: << " Boot Start\n";
91: bootInProgress = true;
92: } else if (regex_match(line, sm, endBootRegex)) {
93: date bootEndDate(stoi(sm[1]), stoi(sm[2]), stoi(sm[3]));
94: ptime bootEndTime(bootEndDate, time_duration(stoi(sm[4]), stoi(s
m[5]), stoi(sm[6])));
95: rptFile << lines << "(" << filename << "): "
96: << sm[1] << "-" << sm[2] << "-" << sm[3] << " "
97: << sm[4] << ":" << sm[5] << ":" << sm[6]
98: << " Boot Completed\n";
99: time_duration bootDuration = bootEndTime - bootStartTime;
100: int bootTimeMillis = bootDuration.total_milliseconds();
101: rptFile << "\tBoot Time: " << bootTimeMillis << "ms\n\n";
102: bootInProgress = false;
103: }
104: lines++;
105: }
106:
107: logFile.close();
108: rptFile.close();
109: }
```