Computing IV Sec 110: Project Portfolio

Leonard Nguyen

Summer 2022

Contents

Time to Complete Portfolio: 24 hours

1 PS0: Hello SFML

1.1 Discussion

This assignment was a prerequisite for the rest of the projects that're documented here. As this is titled "ps0", the whole idea of the assignment was to get us all up to speed with what softwares we'll be using before we move to "ps1" and onward. More specifically, we needed to install **SFML**, Simple and Fast Multimedia Library. We were to then test the features of SFML using some of the supplied code from their website found here: `https://www.sfml-dev.org/tutorials/2.4/start-linux.php`

1.2 What I accomplished

We started by finding a way to install **Ubuntu**, the Linux distro that runs all of our assigned projects. I personally went the WSL route that was discussed during class, which works wonders in terms of running the basic commands needed for the rest of the projects. The default code copied off the SFML website worked as expected, shown in Figure 1.1. After this, we were tasked to extend the given code so that we could move a new sprite around the window using the arrow keys. I opted to add one more sprite for the new sprite to move to, revealing a "secret" message shown in Figure 1.2.

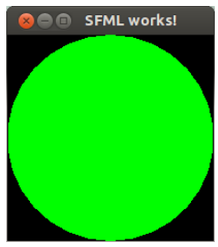


Figure 1.1: Window produced from running the tutorial code

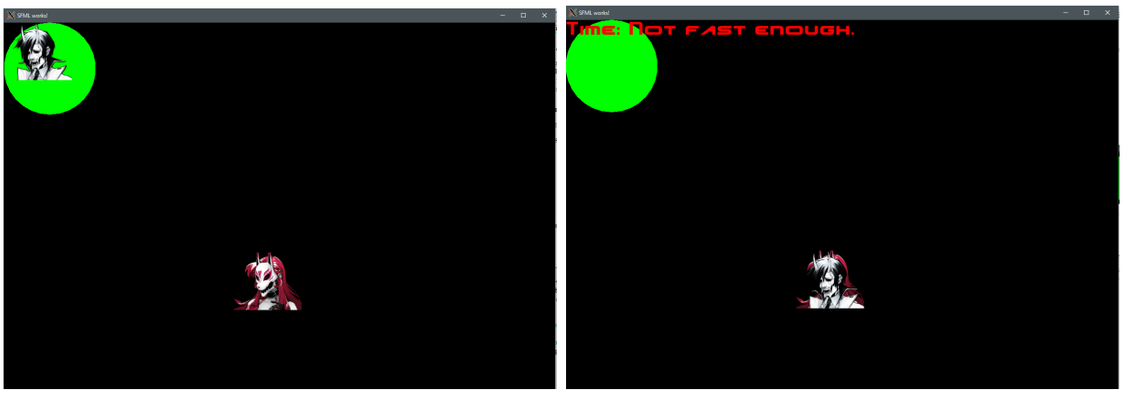


Figure 1.2: Extended program before and after sprite move

1.3 What I already knew

I've worked on this assignment before for the same course with the same instructions of "install SFML, pick up the starter code to open an SFML window, and extend it so a new sprite can accept keystrokes". I've also coded in C and C++ from previous classes. With this in mind, I also already knew of a plan to approach the problem. However, I also knew that implementing music to play in SFML would be a bit of personal learning curve that I still can't quite figure out as of the time of writing this document.

1.4 What I learned

Despite having coded something similar, there was still plenty to be learned, especially during the setup process. As mentioned earlier, I enabled the Windows Subsystem for Linux (WSL), a feature I didn't even know existed until being told about it during lecture. Having learned of this, I spent the majority of development time in the setup stage, figuring out how to

install it along with the graphics and audio servers that help the wsl to load SFML related media. This was also when I learned of linting, which I didn't quite grasp the concept of until later projects.

1.5 Challenges

Most of my problems were during the setup stage. Each step had tons of loading times to get through as well as instructions to read. The process was done as cautiously as possible, keeping my own storage space in mind. However, setting up the audio server, PulseAudio was the most difficult, as I had to install a new browser to pick up the archive, only for it to not work. Although, this could be due to me setting it up improperly.

1.6 Codebase

Makefile

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
# Your .hpp files
DEPS =
# Your compiled .o files
OBJECTS =
# The name of your program
PROGRAM = sfml-app

.PHONY: all clean lint

all: $(PROGRAM)

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

main.cpp

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(1200, 800), "SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

    //building up a sprite with a texture (neon sprite)
    sf::Texture neonTexture;
    if (!neonTexture.loadFromFile("sprite.png"))
        return -1;
    sf::Sprite neonSprite;
```

```
    neonSprite.setTexture(neonTexture);

    //building up a sprite with a texture (neon red)
    sf::Texture neonRedTexture;
    if (!neonRedTexture.loadFromFile("neonRed.png"))
        return -1;
    sf::Sprite neonRedSprite;
    neonRedSprite.setTexture(neonRedTexture);

    // reveal a very secret text (with font)
    sf::Font font;
    if (!font.loadFromFile("NiseJSRF.ttf"));
    sf::Text winMessage;
    winMessage.setFont(font);
    winMessage.setString("Time: Not fast enough.");
    winMessage.setCharacterSize(32);
    winMessage.setFillColor(sf::Color::Red);
    winMessage.setStyle(sf::Text::Bold);

    /*
    //play music
    sf::Music music;
    if (!music.openFromFile("totallynotcopyrighted.wav"))
        return -1; // error
    music.play();
    */

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        neonRedSprite.setPosition(500, 500);

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
            // move left...
            neonSprite.move(-20, 0);
        }
        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
            // move right...
            neonSprite.move(20, 0);
        }
        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
            // move up...
            neonSprite.move(0, -20);
        }
        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
            // move down...
            neonSprite.move(0, 20);
        }

        window.clear();
        window.draw(shape);
        window.draw(neonRedSprite);
        window.draw(neonSprite);
```

```
74
75          if (neonSprite.getPosition() == neonRedSprite.getPosition()) {
76              window.draw(winMessage);
77          }
78
79          window.display();
80      }
81
82      return 0;
83  }
```

2 PS1: LFSR and Image Encoding

2.1 Discussion

The assignment was to implement an image encoder that uses a *Linear Feedback Shift Register* (LFSR) that pseudorandomly alters each individual pixel of an inputed image file. This is one of the larger programs we had to develop, hence it requires two parts to complete to be completed by separate dates. PS1a is the stage when we implement the LFSR structure that performs its bitwise operations. For this, we use a Fibonacci LFSR feed it a seed string of 1's and 0's to return the desired result for each pixel. PS1b is the stage when we utilize our LFSR to build a function that actually encodes and decodes the inputed images.

2.2 What I accomplished

The general accomplishment made with developing this project is that it does in fact encode and decode images to some capacity. Of course, to achieve this, we'd need look into the more detailed milestones between the LFSR part, and the image encoding part.

2.2.1 Part A: Fibonacci Linear Feedback Shift Register (FibLFSR)

Fortunately, the code to start us off was provided to us in the form of a Fibonacci LFSR (FibLFSR) class. This class includes declarations of all the functions to be implemented, giving us a proper blueprint on where to start writing our code.

The functions are as follows:

- `step()`: shifts the a bit once through the LFSR, returning the resulting bit
- `generate(int k)`: takes an integer value and repeats the `step()` function 'k' times, returning its final bit
- `ostream overrider`: lets us print the FibLFSR object as a string of 1's an 0's with cout

However, the fields for the class were something we ourselves had to write into the class file. The instructions do give the values necessary for a FibLFSR to have. Note that since there is a dynamic array call, it had to get deleted through a manually written destructor.

The fields added are as follows:

- The number of bits `n`
- The seed in the form of a string
- A dynamic array of tap positions size `n`

Once getting all the functions and fields set for the structure, we implement the given functions. For the `step()` function, I first isolated a character from the seed and store it into the tap positions. From here, I set conditions for each tap position that represent an "XOR", an "or" operation that returns false when both conditions are true. Since there wasn't a decimal XOR operator in C++, I wrote out an alternative that expands an "XOR" function to just "ands" and "ors". Essentially it says the condition is true if both taps are 1 or 0.

2.2.2 Part B: PhotoMagic

Here, we're given a declaration to a transform function and an example `pixels.cpp` file to work off of. To start, the instructions asked for us to run the pixel file just to see what would happen. We were also given an image of a cat to use for this operation. The results are shown in Figure 2.1. This gives us a basic idea of how our main routine should look for when we start using our LFSR. From here, I implemented the transform function in a separate PhotoMagic object as instructed, which takes an LFSR object and uses it to shift through the entire image's pixels using LFSR's built in `generate()` function. The results for this are shown for the default cat image in Figure 2.2 and the results for a separate test is shown in Figure 2.3.

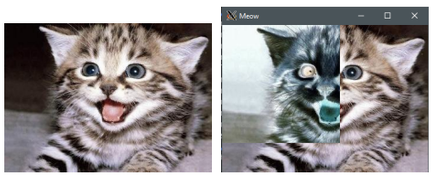


Figure 2.1: First encryption with `pixel.cpp`

2.3 What I already knew

Although I've seen this assignment before, I've never truly gotten a good grasp on how to implement an LFSR. The most I knew or could remember about an LFSR is that it has tap positions and you perform XOR operations in some way or another. In summary, what I knew about the assignment before going into it is limited.

2.4 What I learned

I mainly learned more about LFSRs from this assignment. Although, now my knowledge of them is limited to how to implement specifically a Fibonacci LFSR. However, it was as this assignment where I finally understood how to run boost tests. However, running them is something I still need to work on.

2.5 Challenges

Even though I've learned a bit more on LFSRs, they were and still the biggest challenge for this assignment. More specifically, I had some trouble implementing a `step()` function for it to shift once. This was mainly due to my lack of understanding of them during that time.

2.6 Codebase

Makefile

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
# Your .hpp files
DEPS = FibLFSR.hpp
# Your compiled .o files
OBJECTS = FibLFSR.o PhotoMagic.o
# The name of your program
PROGRAM = PhotoMagic

.PHONY: all clean lint

all: $(PROGRAM) test pixels

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

pixels: pixels.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

test: test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) test pixels
```

```
29
30 lint:
31     cpplint *.cpp *.hpp
```

main.cpp

```
1  #include <iostream>
2  #include <SFML/System.hpp>
3  #include <SFML/Window.hpp>
4  #include <SFML/Graphics.hpp>
5
6  #include "PhotoMagic.hpp"
7
8  int main() {
9      // read three arguments from command line: source image file name,
       output image file name, and FibLFSR seed
10     std::string sourceImageFileName;
11     std::string outputImageFileName;
12     std::string inputSeed;
13     std::cout << "% PhotoMagic ";
14     std::cin >> sourceImageFileName >> outputImageFileName >> inputSeed;
15
16     /*****************************************************************
17     std::cout << "Enter the name of your image file: " << std::endl;
18     std::cin >> sourceImageFileName;
19     std::cout << "Enter a name for your new image file:  " << std::endl;
20     std::cin >> outputImageFileName;
21     std::cout << "Enter a seed (in bits): " << std::endl;
22     std::cin >> inputSeed;
23     *****************************************************************/
24
25     // set up the lfsr, sprite to display the image, and render the windows
26     FibLFSR inputLFSR(inputSeed);
27     sf::Image inputImage;
28     if (!inputImage.loadFromFile(sourceImageFileName))
29         return -1;
30     sf::Image decryptedImage;
31     if (!decryptedImage.loadFromFile(sourceImageFileName))
32         return -1;
33
34     sf::Vector2u size = inputImage.getSize();
35     sf::RenderWindow afterWindow(sf::VideoMode(size.x, size.y), "After");
36     sf::RenderWindow beforeWindow(sf::VideoMode(size.x, size.y), "Before");
37
38     sf::Texture decryptedTexture;
39     decryptedTexture.loadFromImage(decryptedImage);
40
41     sf::Sprite decryptedSprite;
42     decryptedSprite.setTexture(decryptedTexture);
43
44     // encrypt the image with the FibLFSR transform function
45     transform(inputImage, &inputLFSR);
46
47     sf::Texture inputTexture;
48     inputTexture.loadFromImage(inputImage);
49
50     sf::Sprite inputSprite;
51     inputSprite.setTexture(inputTexture);
52
```

```
    while (afterWindow.isOpen() && beforeWindow.isOpen()) {
        sf::Event event;
        while (afterWindow.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                afterWindow.close();
        }

        while (beforeWindow.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                beforeWindow.close();
        }

        afterWindow.clear(sf::Color::White);
        afterWindow.draw(inputSprite);
        afterWindow.display();

        beforeWindow.clear(sf::Color::White);
        beforeWindow.draw(decryptedSprite);
        beforeWindow.display();
    }

    // fredm: saving a PNG segfaults for me, though it does properly
    //   write the file
    if (!inputImage.saveToFile(outputImageFileName))
        return -1;

    return 0;
}
```

FibLFSR.hpp

```
#pragma once

#include <iostream>
#include <string>
#include <vector>

class FibLFSR {
public:
    // Constructor to create LFSR with the given initial seed
    FibLFSR(std::string seed): _seed(seed) {}
    // Simulate one step and return the new bit as 0 or 1
    int step();
    // Simulate k steps and return a k-bit integer
    int generate(int k);
    // get seed
    std::string getSeed() const { return _seed; }
    // destructor
    ~FibLFSR() { delete [] _tap; _tap = NULL; }

private:
    // Any fields that you need
    int _n;
    std::string _seed;
    int *_tap = new int(_n); // taps between pos 15, 13, 12, and 10
};
std::ostream &operator<<(std::ostream &out, const FibLFSR &lfsr);
```

FibLFSR.cpp

```cpp
#include "FibLFSR.hpp"

int FibLFSR::step() {
    // initialize output and first 2 taps
    int remainingBit = 0;
    _tap[0] = _seed.at(0) - '0'; // pos 15
    _tap[1] = _seed.at(2) - '0'; // pos 13

    /*********************************************************
    // read and perform XORs
    for (size_t i = 0; i < _seed.size(); i++) {
        if ((_tap[0] == 1 && _tap[1] == 1) || (_tap[0] == 0 && _tap[1] == 0)) {
            remainingBit = 0;
        } else remainingBit = 1;
    }
    *********************************************************/

    // 15 xor 13
    if ((_tap[0] == 1 && _tap[1] == 1) || (_tap[0] == 0 && _tap[1] == 0)) {
        remainingBit = 0;
    } else remainingBit = 1;

    // set tap position and previous result xor 12
    _tap[2] = _seed.at(3) - '0'; // pos 12
    if ((remainingBit == 1 && _tap[2] == 1) || (remainingBit == 0 && _tap[2] == 0)) {
        remainingBit = 0;
    } else remainingBit = 1;

    // set tap position to 10 and previous result xor 10
    _tap[3] = _seed.at(5) - '0'; // pos 10
    if ((remainingBit == 1 && _tap[3] == 1) || (remainingBit == 0 && _tap[3] == 0)) {
        remainingBit = 0;
    } else remainingBit = 1;

    _seed.erase(0, 1);
    _seed.push_back(remainingBit + '0');

    return remainingBit;
}

int FibLFSR::generate(int k) {
    int result = 0;
    for (int i = 0; i < k; i++) {
        result = result * 2 + step();
    }
    return result;
}

std::ostream &operator<<(std::ostream &out, const FibLFSR &lfsr) {
    out << lfsr.getSeed();
    return out;
}
```

PhotoMagic.hpp

```
#pragma once

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

#include "FibLFSR.hpp"

// Transforms image using FibLFSR
void transform(sf::Image &image, FibLFSR *lfsr);
// Display an encrypted copy of the picture, using the LFSR to do the
   encryption
```

PhotoMagic.cpp

```
#include "PhotoMagic.hpp"

void transform(sf::Image &image, FibLFSR *lfsr) {
    int width = image.getSize().x;
    int height = image.getSize().y;
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            // p is a pixelimage.getPixel(x, y);
            sf::Color p = image.getPixel(x, y);
            p.r = p.r ^ lfsr->generate(70); // red
            p.g = p.g ^ lfsr->generate(50); // green
            p.b = p.b ^ lfsr->generate(30); // blue
            image.setPixel(x, y, p);
        }
    }
}
```

test.cpp

```
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly
// test.cpp for PS1a
// updated 5/12/2022

#include <iostream>
#include <string>

#include "FibLFSR.hpp"

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

BOOST_AUTO_TEST_CASE(testStepFunc1) {
    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);
```

```
23      BOOST_REQUIRE_EQUAL(l.step(), 0);
24      BOOST_REQUIRE_EQUAL(l.step(), 0);
25      BOOST_REQUIRE_EQUAL(l.step(), 1);
26  }
27
28  BOOST_AUTO_TEST_CASE(testStepFunc2) {
29      FibLFSR l("1111111111111111");
30      BOOST_REQUIRE_EQUAL(l.step(), 0);
31      BOOST_REQUIRE_EQUAL(l.step(), 0);
32      BOOST_REQUIRE_EQUAL(l.step(), 0);
33      BOOST_REQUIRE_EQUAL(l.step(), 0);
34      BOOST_REQUIRE_EQUAL(l.step(), 0);
35      BOOST_REQUIRE_EQUAL(l.step(), 0);
36      BOOST_REQUIRE_EQUAL(l.step(), 0);
37      BOOST_REQUIRE_EQUAL(l.step(), 0);
38  }
39
40  BOOST_AUTO_TEST_CASE(testStepFunc3) {
41      FibLFSR l("0000000000000000");
42      BOOST_REQUIRE_EQUAL(l.step(), 0);
43      BOOST_REQUIRE_EQUAL(l.step(), 0);
44  }
45
46  BOOST_AUTO_TEST_CASE(testStepFunc4) {
47      FibLFSR l("1010101010101010");
48      BOOST_REQUIRE_EQUAL(l.step(), 0);
49      BOOST_REQUIRE_EQUAL(l.step(), 0);
50      BOOST_REQUIRE_EQUAL(l.step(), 0);
51      BOOST_REQUIRE_EQUAL(l.step(), 0);
52      BOOST_REQUIRE_EQUAL(l.step(), 0);
53      BOOST_REQUIRE_EQUAL(l.step(), 0);
54      BOOST_REQUIRE_EQUAL(l.step(), 0);
55      BOOST_REQUIRE_EQUAL(l.step(), 0);
56      BOOST_REQUIRE_EQUAL(l.step(), 0);
57      BOOST_REQUIRE_EQUAL(l.step(), 0);
58      BOOST_REQUIRE_EQUAL(l.step(), 0);
59      BOOST_REQUIRE_EQUAL(l.step(), 1);
60  }
61
62  BOOST_AUTO_TEST_CASE(testStepFunc5) {
63      FibLFSR l("101101");
64      BOOST_REQUIRE_EQUAL(l.step(), 0);
65      BOOST_REQUIRE_EQUAL(l.step(), 1);
66      BOOST_REQUIRE_EQUAL(l.step(), 1);
67      BOOST_REQUIRE_EQUAL(l.step(), 1);
68      BOOST_REQUIRE_EQUAL(l.step(), 0);
69      BOOST_REQUIRE_EQUAL(l.step(), 1);
70      BOOST_REQUIRE_EQUAL(l.step(), 1);
71      BOOST_REQUIRE_EQUAL(l.step(), 1);
72  }
73
74  BOOST_AUTO_TEST_CASE(testStepFunc6) {
75      FibLFSR l("10110110001101101011011000110110");
76      BOOST_REQUIRE_EQUAL(l.step(), 0);
77      BOOST_REQUIRE_EQUAL(l.step(), 0);
78      BOOST_REQUIRE_EQUAL(l.step(), 0);
79      BOOST_REQUIRE_EQUAL(l.step(), 1);
80      BOOST_REQUIRE_EQUAL(l.step(), 1);
81      BOOST_REQUIRE_EQUAL(l.step(), 0);
```

```
82      BOOST_REQUIRE_EQUAL(l.step(), 0);
83      BOOST_REQUIRE_EQUAL(l.step(), 1);
84  }
85
86  BOOST_AUTO_TEST_CASE(testGenerateFunc1) {
87      FibLFSR l2("1011011000110110");
88      BOOST_REQUIRE_EQUAL(l2.generate(9), 51);
89  }
90
91  BOOST_AUTO_TEST_CASE(testGenerateFunc2) {
92      FibLFSR l2("1011011000110110");
93      BOOST_REQUIRE_EQUAL(l2.generate(5), 3);
94      BOOST_REQUIRE_EQUAL(l2.generate(5), 6);
95      BOOST_REQUIRE_EQUAL(l2.generate(5), 14);
96      BOOST_REQUIRE_EQUAL(l2.generate(5), 24);
97      BOOST_REQUIRE_EQUAL(l2.generate(5), 1);
98      BOOST_REQUIRE_EQUAL(l2.generate(5), 13);
99      BOOST_REQUIRE_EQUAL(l2.generate(5), 28);
100 }
101
102 BOOST_AUTO_TEST_CASE(testGenerateFunc3) {
103     FibLFSR l2("1111111111111111");
104     BOOST_REQUIRE_EQUAL(l2.generate(7), 0);
105     BOOST_REQUIRE_EQUAL(l2.generate(7), 6);
106     BOOST_REQUIRE_EQUAL(l2.generate(7), 96);
107     BOOST_REQUIRE_EQUAL(l2.generate(7), 60);
108     BOOST_REQUIRE_EQUAL(l2.generate(7), 123);
109     BOOST_REQUIRE_EQUAL(l2.generate(7), 45);
110     BOOST_REQUIRE_EQUAL(l2.generate(7), 73);
111 }
```

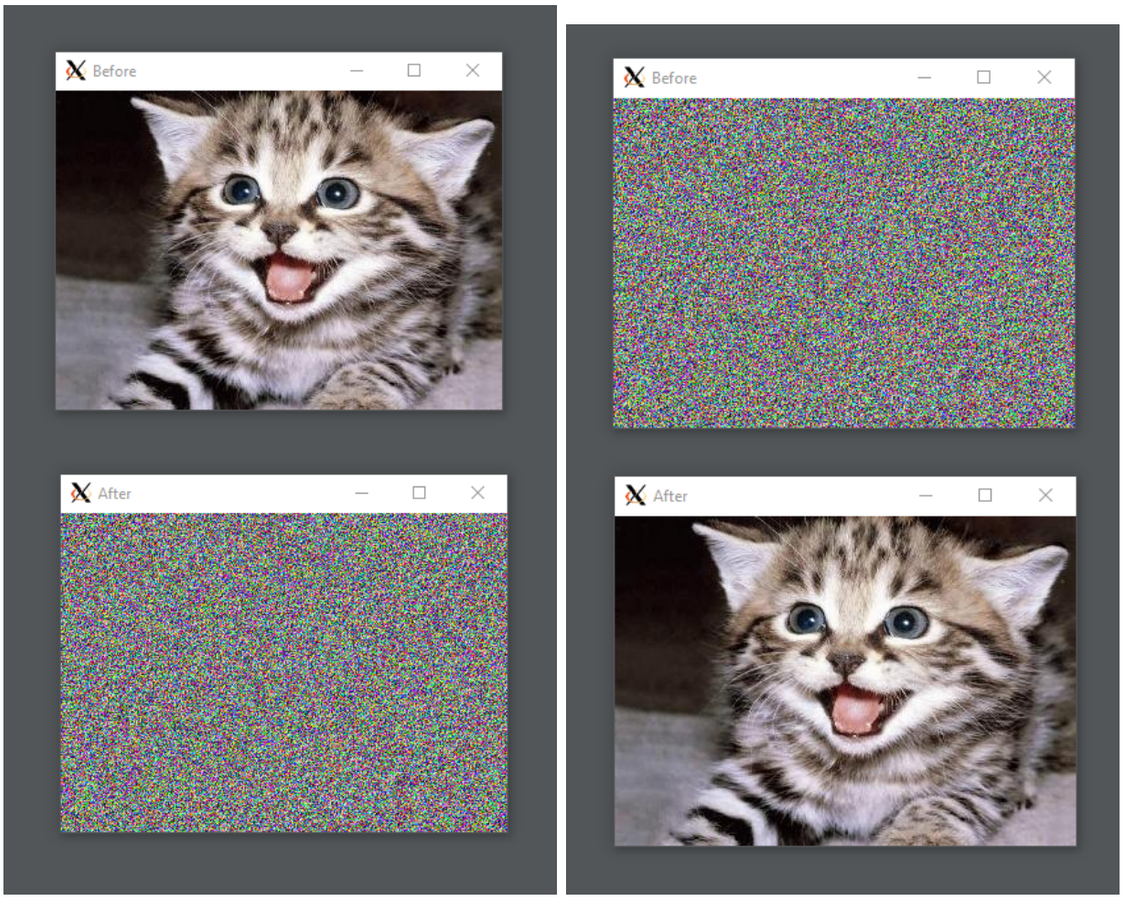


Figure 2.2: Encryption and decryption using PhotoMagic

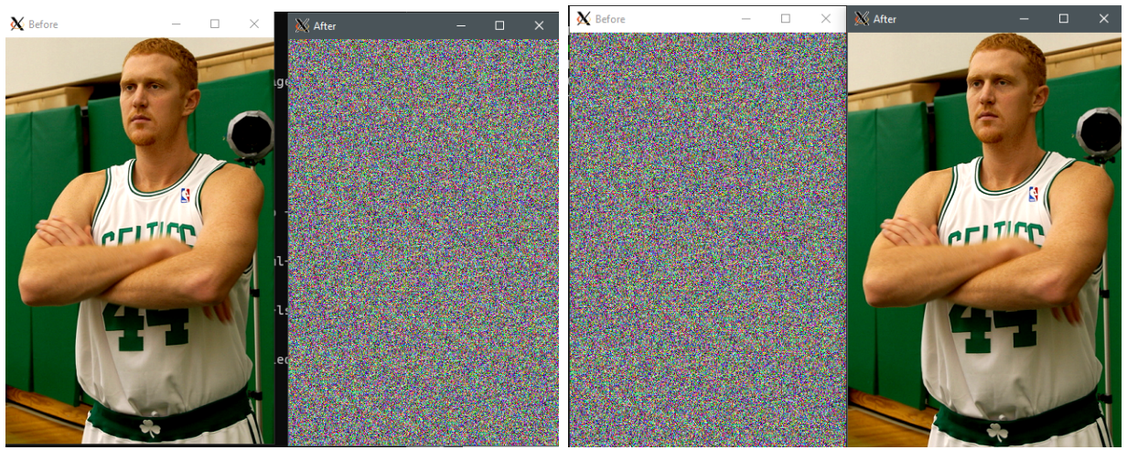


Figure 2.3: Second example using different sized png files

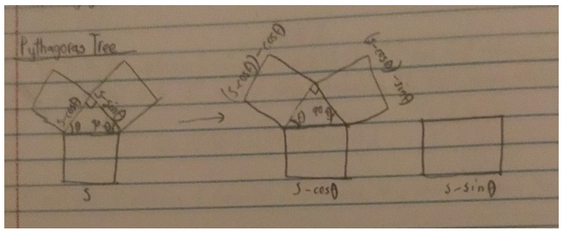
3 PS2: Pythagorean Tree

3.1 Discussion

This project is made to visually represent a fractal known as the Pythagoras Tree. To make the fractal, the biggest point made was to be able to write out the key math portions that fuel its generation. This being said, Pythagoras's theorem and some trigonometry were necessary in completing this assignment. As of now, this program only represents the Pythagoras Tree for when the triangles within them are 90 45 45 triangles. However, this assignment is also an introduction to an SFML class known as "Drawable". This class comes with a `draw()` function that our tree class can inherit, allowing for us to "draw" the tree onto an SFML window.

3.2 What I accomplished

For this assignment, we needed to represent the tree as an object. This meant that rather than implementing the tree directly into the main routine, we needed to make the header and source file for the tree. These are our `PTree.hpp` and `PTree.cpp` files. In our usual top down approach to the problem, I wrote out the header file's functions and fields first and foremost. What's interesting about this particular structure is how it really only represents a single square on its own. Since that's the case, the structure requires a separate function that recursively draws the rest of the tree. However, for this to happen, it was recommended that we write out some math in order to prevent us eyeballing the positions and angles of each square. A bit of the math is shown in Figure 3.1. Here, I determine $sin\theta$ and $cos\theta$ with the use of SOHCAHTOA. I also kept in mind the numerical calculations that go with this way of calculating side lengths. With the math in mind, the tree utilizes these calculations interchangably between the structure's draw function and the separate `pTree()` function. They each call each other recursively to generate each section of the tree on both sides. The final result is shown in Figure 3.2.



$$s - (1/\sqrt{2}), s - (1/\sqrt{2}) - (1/\sqrt{2}), ...$$

Figure 3.1: Some drawings and math indicating the general pattern for each branch

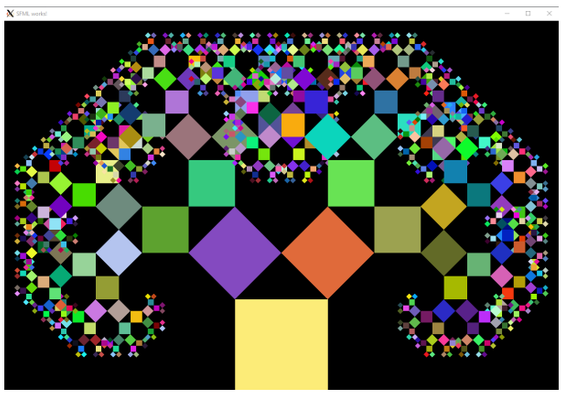


Figure 3.2: Final Tree with flashing colors using rand()

3.3 What I already knew

One of the earliest exposures I've had to computer science growing up was how fractals are drawn in computer science, recursion. I've written some recursive graphics at some point

in high school using Java, but also drew Sierpinski Triangles during a previous run of this course as well as in a computer graphics class. This being the case, I already had a general strategy for how to approach a problem such as this.

3.4 What I learned

Despite Java being an object oriented language, I've never actually written any recursive graphics in between source files. I also now have a better understanding of the Drawable class in SFML and how it'd be of good use in the later projects going forward. I've also learned just how much easier drawing graphics like these can get if I were to write out some basic calculations before writing them in code. This was demonstrated during one of our demonstrations during class on drawing sierpinski triangles and snowflakes.

3.5 Challenges

The main challenge was making sure I understand the Drawable class and how it works. This was something that I had the most trouble with when I first took this class. I also didn't understand right away that the `pTree()` function was a nonmember function, so the concept of calling both `pTree()` and `draw()` between each other was something I didn't get until later.

3.6 Codebase

Makefile

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = PTree.hpp
# Your compiled .o files
OBJECTS = PTree.o
# The name of your program
PROGRAM = PTree

.PHONY: all clean lint

all: $(PROGRAM)

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

main.cpp

```
#include <iostream>

#include "PTree.hpp"

int main() {
```

```
    double length;
    int depth;
    std::cout << "Enter a length for the base square and a tree depth: ";
    std::cin >> length >> depth;
    const unsigned int windowSizeWidth = 6*length;
    const unsigned int windowSizeHeight = 4*length;
    PTree tree(length, depth);

    sf::RenderWindow window(sf::VideoMode(windowSizeWidth, windowSizeHeight)
    , "SFML works!");

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();
        // Draw Tree
        window.draw(tree);
        window.display();
    }

    return 0;
}
```

PTree.hpp

```
#pragma once

#include <cmath>
#include <SFML/Graphics.hpp>

class PTree: public sf::Drawable {
public:
    PTree(double l, int n): _baseSquare(sf::Vector2f(0, 0), sf::Vector2f(l,
    l)), _n(n) {}

private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
    const;
    sf::FloatRect _baseSquare;
    int _n;
};

void pTree(sf::RenderTarget &target, const int depth, const sf::
    RectangleShape &root);
```

PTree.cpp

```
#include "PTree.hpp"

void PTree::draw(sf::RenderTarget &target, sf::RenderStates states) const {
    sf::RectangleShape square{{_baseSquare.width, _baseSquare.width}};
    square.setPosition(target.getSize().x / 2.f, target.getSize().y -
    _baseSquare.width / 2.f);
    square.setOrigin(_baseSquare.width/2, _baseSquare.height/2);
```

```
7     square.setFillColor(sf::Color(rand(), rand(), rand()));
8     pTree(target, _n, square);
9 }
10
11 void pTree(sf::RenderTarget &target, const int depth, const sf::
      RectangleShape &root) {
12     static const float soh = sqrt(2)/2;
13     static const float cah = sqrt(2)/2;
14     if (depth <= 0)
15         return;
16
17     target.draw(root);
18     sf::Vector2f parentSize = root.getSize();
19     sf::Transform parentTransform = root.getTransform();
20
21     // left side
22     sf::RectangleShape leftBranch = root;
23     leftBranch.setFillColor(sf::Color(rand(), rand(), rand()));
24     leftBranch.setSize(parentSize*cah);
25     leftBranch.setOrigin(0, leftBranch.getSize().y);
26     leftBranch.setPosition(parentTransform.transformPoint({0,0}));
27     leftBranch.rotate(-45);
28     pTree(target, depth-1, leftBranch);
29
30     // right side
31     sf::RectangleShape rightBranch = root;
32     rightBranch.setFillColor(sf::Color(rand(), rand(), rand()));
33     rightBranch.setSize(parentSize*soh);
34     rightBranch.setOrigin(rightBranch.getSize());
35     rightBranch.setPosition(parentTransform.transformPoint({parentSize.x,
       0}));
36     rightBranch.rotate(45);
37     pTree(target, depth-1, rightBranch);
38 }
```

4 PS3: NBody Simulation

4.1 Discussion

The assignment is to implement a simulation that demonstrates the motion of particles according to gravitational forces. For this project, we call these particles NBodies or celestial bodies. The point of this assignment is to introduce us to more applications of the Drawable class, providing our drawable objects some physics calculations that help it behave in accordance to Isaac Newton's gravitational principles within his *Principia Mathematica*. For our project specifically, we were tasked to mainly simulate our solar system's orbit from the sun to mars.

4.2 What I accomplished

Generally, the simulation is drawn into an SFML window and does show some semblence of planetary orbit. The program can take a time limit, a time increment value, and a sample file of what we'd want to simulate. The simulation also displays an elapsed time that increments by the time increment value we feed it from the command line.

4.2.1 Part A: Static Drawing

For this part, I implemented two drawable objects, a CelestialBody and a Universe. The CelestialBody object is able to take in a single line of celestial body data through the use of an istream overrider, which includes a celestial body's position, velocity, mass, and the associated image representing the body. Using this data, I write out the `draw()` function for the celestial body. The `draw()` function scales the massive data from the sample to a size that an SFML window can realistically display. The CelestialBody object can also print out its data onto the console. The Universe object does everything a CelestialBody object can, just with different fields for the radius and how many bodies there are in the universe.

Memory

However, since multiple celestial bodies are to be drawn to the universe, the Universe object must have a structure that holds said celestial bodies. For this I use a vector of smart pointers that set a counter for the amount of CelestialBody objects being fed into the universe. This results in a static drawing of the planets in their initial positions in Figure 4.1. I've also performed some boost test as well as tested out some other given samples. The other samples are shown in Figure 4.3

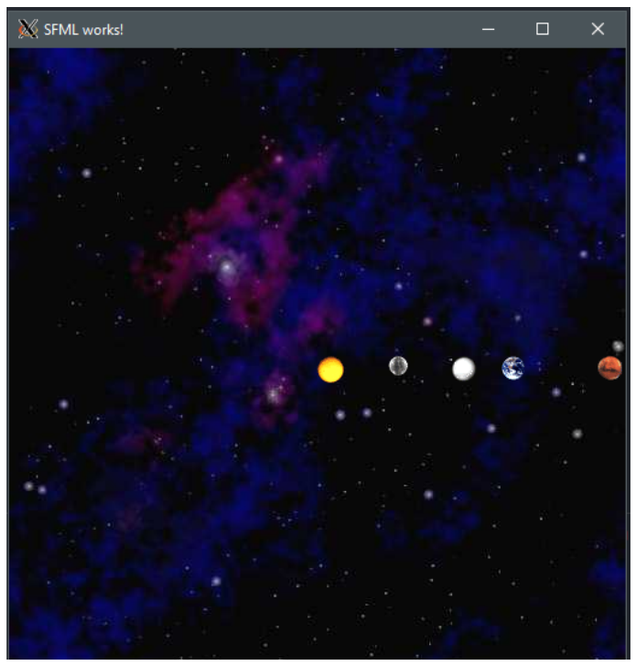


Figure 4.1: Static planets in initial positions

4.2.2 Part B: Physics Simulation

This is the stage where we take our static bodies and make them move. For this, we were given some equations for gravitational force, acceleration, and displacement. We were also tasked to implement these calculations using the *leapfrog finite difference approximation scheme*. Using this helps us make calculations at each time increment between particles. However, I haven't figured out how I could calculate the force specifically at a given time nor have I made any calculations with this method. Instead, I opted for a method where I compare two particles through a nested for each loop in order to calculate displacement and net force. We were also given test data to record our results according to the amount of times our `step()` function is called. A screenshot of the planets orbiting is shown in Figure 4.2. Some more examples of the simulation running are shown in Figure 4.4

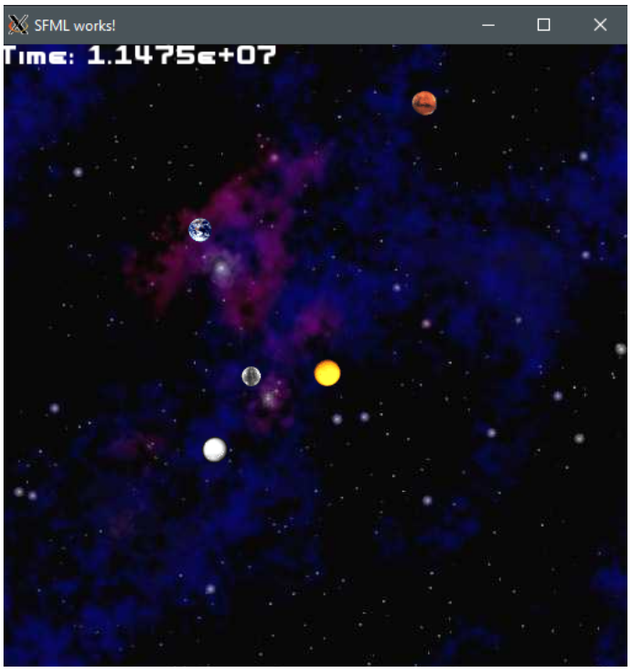


Figure 4.2: Screenshot at an arbitrary point during the simulation's runtime

4.3 What I already knew

By now, I've gotten a decent grasp on how to implement `draw()` functions thanks to the Pythagorean Tree project in Section 3. I also had a basic idea on how to implement the istream and ostream operators, given that I had previous knowledge of this program needing smart pointers. I also gained a decent understanding for how to utilize boost tests from the LFSR project in Section 2. Overall, I had a decent foundation for developing this program thanks to the previous projects.

4.4 What I learned

Here, I've learned how to call and use smart pointers for objects such as celestial bodies. By learning this, I've gained a new way to manage objects and data in a way that helps me and the program keep track of ownership between memory. More specifically, I've learned what library the smart pointers belong to and how to write the syntax for shared pointers.

4.5 Challenges

Two of the main challenges I faced in this project are smart pointers and leapfrogging. Before going into this assignment, I needed to take some development time off to research documentation for smart pointers, what they're used for, and how to call them into my program. The same problem persisted when leapfrogging. Due to the time constraints I had with this project, I never got to fix my code so that the program actually does "leapfrog".

4.6 Codebase

Makefile

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = Universe.hpp CelestialBody.hpp
# Your compiled .o files
OBJECTS = Universe.o CelestialBody.o
# The name of your program
PROGRAM = NBody

.PHONY: all clean lint

all: $(PROGRAM) test

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

test: test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) test

lint:
    cpplint *.cpp *.hpp
```

main.cpp

```
// Copyright 2023 Leonard Nguyen
#include <fstream>
#include <sstream>
#include <SFML/Audio.hpp>

#include "Universe.hpp"

int main(int argc, char *argv[]) {
    double timeLimit = atof(argv[1]);
    double deltaT = atof(argv[2]);
    double time = 0;

    sf::Font font;
    if (!font.loadFromFile("./src/NiseJSRF.ttf")) {} else {}
    sf::Text timeMessage;
    timeMessage.setFont(font);
    timeMessage.setCharacterSize(16);
    timeMessage.setFillColor(sf::Color::White);
    timeMessage.setStyle(sf::Text::Bold);

    /*
    sf::Music music;
    if (!music.openFromFile("./src/2001.wav")) {} else {}
```

```
    */

    Universe bigUniverse;
    std::cin >> bigUniverse;
    bigUniverse.setTimeLimit(timeLimit);

    const unsigned int windowSize = 512;
    sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "SFML works!");

    while (window.isOpen() && time < timeLimit) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        // music.play();
        bigUniverse.step(deltaT);
        std::stringstream timeString;
        timeString << "Time: " << time << " secs";
        timeMessage.setString(timeString.str());
        window.clear();
        window.draw(bigUniverse);
        window.draw(timeMessage);
        window.display();
        time += deltaT;
    }

    std::cout << bigUniverse << std::endl;

    return 0;
}
```

CelestialBody.hpp

```
// Copyright 2023 Leonard Nguyen
#pragma once

#include <iostream>
#include <string>
#include <SFML/Graphics.hpp>

class CelestialBody: public sf::Drawable {
 public:
    CelestialBody() {}
    explicit CelestialBody(double r): _universeRadius(r) {}
    friend std::istream &operator>>(std::istream &in, CelestialBody &celestialBody);

    double getPosX() const {return _pos.x;}
    double getPosY() const {return _pos.y;}
    double getVelX() const {return _vel.x;}
    double getVelY() const {return _vel.y;}
    double getMass() const {return _mass;}
    std::string getBodyFileName() const {return _bodyFileName;}

    void setPosX(double newX) {_pos.x = newX;}
    void setPosY(double newY) {_pos.y = newY;}
```

```
23      void setVelX(double newX) {_vel.x = newX;}
24      void setVelY(double newY) {_vel.y = newY;}
25
26      void calculateVel(double forceX, double forceY, double seconds);
27      void calculatePos(double seconds);
28
29  private:
30      virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
        const;
31      sf::Vector2f _pos;
32      sf::Vector2f _vel;
33      double _mass;
34      std::string _bodyFileName;
35      double _universeRadius;
36      // sf::Vector2f _netForce;
37  };
38
39  std::ostream &operator<<(std::ostream &out, CelestialBody &celestialBody);
```

CelestialBody.cpp

```
1   // Copyright 2023 Leonard Nguyen
2   #include "CelestialBody.hpp"
3
4   void CelestialBody::draw(sf::RenderTarget &target, sf::RenderStates states)
        const {
5       sf::Texture celestialTexture;
6       if (!celestialTexture.loadFromFile(_bodyFileName)) {
7           return;
8       } else {
9           sf::Sprite celestialSprite;
10          celestialSprite.setTexture(celestialTexture);
11          double conversionFactor = (target.getSize().x / 2) / _universeRadius
        ;
12          float x = (_pos.x * conversionFactor);
13          float y = -(_pos.y * conversionFactor);
14          celestialSprite.setPosition(x + (target.getSize().x / 2), y + (
        target.getSize().y / 2));
15          target.draw(celestialSprite);
16      }
17  }
18
19  void CelestialBody::calculateVel(double forceX, double forceY, double
        seconds) {
20      // calculate accel
21      double accelX = forceX / _mass;
22      double accelY = forceY / _mass;
23      _vel.x = _vel.x + seconds * accelX;
24      _vel.y = _vel.y + seconds * accelY;
25  }
26
27  void CelestialBody::calculatePos(double seconds) {
28      _pos.x = _pos.x + seconds * _vel.x;
29      _pos.y = _pos.y + seconds * _vel.y;
30  }
31
32  std::istream &operator>>(std::istream &in, CelestialBody &celestialBody) {
33      in >> celestialBody._pos.x >> celestialBody._pos.y;
34      in >> celestialBody._vel.x >> celestialBody._vel.y;
```

```
    in >> celestialBody._mass;
    in >> celestialBody._bodyFileName;
    celestialBody._bodyFileName = "./src/" + celestialBody._bodyFileName;
    return in;
}
std::ostream &operator<<(std::ostream &out, CelestialBody &celestialBody) {
    out << celestialBody.getPosX() << "  ";
    out << celestialBody.getPosY() << "  ";
    out << celestialBody.getVelX() << "  ";
    out << celestialBody.getVelY() << "  ";
    out << celestialBody.getMass() << "  ";
    out << celestialBody.getBodyFileName();
    return out;
}
```

Universe.hpp

```
// Copyright 2023 Leonard Nguyen
#pragma once

#include <memory>
#include <vector>
#include <cmath>
// #include <algorithm>

#include "CelestialBody.hpp"

class Universe: public sf::Drawable {
 public:
    const double G = 6.67e-11;

    Universe() {}
    friend std::istream &operator>>(std::istream &in, Universe &universe);
    friend std::ostream &operator<<(std::ostream &out, Universe &universe);

    int getN() {return _n;}         // getter for particle count(planets)
    double getR() {return _r;}      // getter for radius
    std::shared_ptr<CelestialBody> getBody(int index) { return
   _celestialVector.at(index); }

    void setTimeLimit(double t) { _t = t; }
    void step(double seconds);

 private:
    int _n;       // number of particles
    double _r;    // radius of the universe
    double _t;    // time limit of steps
    virtual void draw(sf::RenderTarget &target, sf::RenderStates states)
   const;

    std::vector<std::shared_ptr <CelestialBody>> _celestialVector;
};
```

Universe.cpp

```
// Copyright 2023 Leonard Nguyen
#include "Universe.hpp"

```

```
4  void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const
       {
5      sf::Texture backgroundTexture;
6      if (!backgroundTexture.loadFromFile("./src/starfield.jpg")) {
7          return;
8      } else {
9          sf::Sprite backgroundSprite;
10         backgroundSprite.setTexture(backgroundTexture);
11         backgroundSprite.setScale(target.getSize().x / backgroundTexture.
       getSize().x,
12             target.getSize().y / backgroundTexture.getSize().y);
13         target.draw(backgroundSprite);
14
15         for (auto i : _celestialVector) {
16             // 1 celestial body knows the calculation for the positions*
17             // 2 separate universe and screen positions
18             target.draw(*i);
19         }
20     }
21 }
22
23 void Universe::step(double seconds) {
24     if (seconds > 0 && seconds <= _t) {      // 0 < seconds <= timeLimit
25         for (auto &particle : _celestialVector) {
26             for (auto &otherParticle : _celestialVector) {  // to compare
       two bodies
27                 if (particle != otherParticle) {
28                     // calculate displacement
29                     double deltaX = otherParticle->getPosX() - particle->
       getPosX();
30                     double deltaY = otherParticle->getPosY() - particle->
       getPosY();
31                     double r = sqrt((deltaX * deltaX) + (deltaY * deltaY));
32
33                     // calculate force
34                     double gForce = (G * otherParticle->getMass() * particle
       ->getMass()) / (r * r);
35                     double forceX = gForce * deltaX/r;
36                     double forceY = gForce * deltaY/r;
37
38                     // calculate new vel
39                     particle->calculateVel(forceX, forceY, seconds);
40                 }
41             }
42             // calculate new pos
43             particle->calculatePos(seconds);
44         }
45     } else { return; }
46 }
47
48 std::istream &operator>>(std::istream &in, Universe &universe) {
49     in >> universe._n;
50     in >> universe._r;
51     for (int i = 0; i < universe._n; i++) {
52         std::shared_ptr<CelestialBody> inputBody = std::make_shared<
       CelestialBody>(universe._r);
53         in >> *inputBody;
54         universe._celestialVector.push_back(inputBody);
55     }
```

```
56      return in;
57  }
58
59  std::ostream &operator<<(std::ostream &out, Universe &universe) {
60      out << universe._n << std::endl;
61      out << universe._r << std::endl;
62      for (auto & i : universe._celestialVector) {
63          out << *i << std::endl;
64      }
65      return out;
66  }
```

test.cpp

```
1   // Copyright 2023 Leonard Nguyen
2   #include <iostream>
3   #include <string>
4   #include <fstream>
5
6   #include "Universe.hpp"
7
8   #define BOOST_TEST_DYN_LINK
9   #define BOOST_TEST_MODULE Main
10  #include <boost/test/unit_test.hpp>
11
12
13  BOOST_AUTO_TEST_CASE(universe1BodyTest) {
14      std::ifstream fin("./src/1body.txt");
15      Universe oneBodyUniverse;
16      fin >> oneBodyUniverse;
17      BOOST_CHECK_EQUAL(oneBodyUniverse.getN(), 1);
18      BOOST_CHECK_EQUAL(oneBodyUniverse.getR(), 100);
19      BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getPosX(), 10);
20      BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getPosY(), 20);
21      BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getVelX(), 2);
22      BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getVelY(), 1);
23      BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getMass(), 1e20);
24      BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getBodyFileName(), "./src/
        earth.gif");
25  }
26
27  BOOST_AUTO_TEST_CASE(universe3BodyTest) {
28      std::ifstream fin("./src/3body.txt");
29      Universe threeBodyUniverse;
30      fin >> threeBodyUniverse;
31      BOOST_CHECK_EQUAL(threeBodyUniverse.getN(), 3);
32      BOOST_CHECK_EQUAL(threeBodyUniverse.getR(), 1.25e11);
33      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getPosX(), 0);
34      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getPosY(), 0);
35      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getVelX(), 0.05e04);
36      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getVelY(), 0);
37      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getMass(), 5.974e24);
38      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getBodyFileName(), "./
        src/earth.gif");
39
40      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getPosX(), 0);
41      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getPosY(), 44999999488);
42      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getVelX(), 3.00e04);
43      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getVelY(), 0);
```

```
44      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getMass(), 1.989e30);
45      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getBodyFileName(), "./
        src/sun.gif");
46
47      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getPosX(), 0);
48      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getPosY(), -44999999488)
        ;
49      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getVelX(), -3.00e04);
50      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getVelY(), 0);
51      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getMass(), 1.989e30);
52      BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getBodyFileName(), "./
        src/sun.gif");
53  }
54
55  BOOST_AUTO_TEST_CASE(universeSoapOperaTest) {
56      std::ifstream fin("./src/soap-opera.txt");
57      Universe soapOperaUniverse;
58      fin >> soapOperaUniverse;
59      BOOST_CHECK_EQUAL(soapOperaUniverse.getN(), 4);
60      BOOST_CHECK_EQUAL(soapOperaUniverse.getR(), 3.00e11);
61      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(0)->getPosX(), 0);
62      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(0)->getPosY(), 115470000128)
        ;
63      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(0)->getVelX(), -3.6524e4);
64      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(0)->getVelY(), 0);
65      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(0)->getMass(), 3.3e30);
66      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(0)->getBodyFileName(), "./
        src/sun.gif");
67
68      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(1)->getPosX(), 99999997952);
69      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(1)->getPosY(), -57735000064)
        ;
70      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(1)->getVelX(), 1.8262e4);
71      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(1)->getVelY(), 3.1631e4);
72      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(1)->getMass(), 3.3e30);
73      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(1)->getBodyFileName(), "./
        src/mars.gif");
74
75      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(2)->getPosX(), -99999997952)
        ;
76      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(2)->getPosY(), -57735000064)
        ;
77      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(2)->getVelX(), 1.8262e4);
78      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(2)->getVelY(), -3.1631e4);
79      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(2)->getMass(), 3.3e30);
80      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(2)->getBodyFileName(), "./
        src/venus.gif");
81
82      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(3)->getPosX(),
        -249999998976);
83      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(3)->getPosY(), 0);
84      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(3)->getVelX(), 0);
85      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(3)->getVelY(), 6e4);
86      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(3)->getMass(), 3.302e20);
87      BOOST_CHECK_EQUAL(soapOperaUniverse.getBody(3)->getBodyFileName(), "./
        src/mercury.gif");
88  }
89
90  BOOST_AUTO_TEST_CASE(universe1BodyPhysicsTest) {
```

```
91      std::ifstream fin("./src/1body.txt");
92      Universe oneBodyUniverse;
93      double timeLimit = 60;
94      double deltaT = 1;
95      double time = 0;
96      fin >> oneBodyUniverse;
97      oneBodyUniverse.setTimeLimit(timeLimit);
98
99      while (time < timeLimit) {
100         oneBodyUniverse.step(deltaT);
101         time += deltaT;
102     }
103
104     BOOST_CHECK_EQUAL(oneBodyUniverse.getN(), 1);
105     BOOST_CHECK_EQUAL(oneBodyUniverse.getR(), 100);
106     BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getPosX(), 130);
107     BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getPosY(), 80);
108     BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getVelX(), 2);
109     BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getVelY(), 1);
110     BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getMass(), 1e20);
111     BOOST_CHECK_EQUAL(oneBodyUniverse.getBody(0)->getBodyFileName(), "./src/
    earth.gif");
112 }
113
114 BOOST_AUTO_TEST_CASE(universe3BodyPhysicsTest) {
115     std::ifstream fin("./src/3body.txt");
116     Universe threeBodyUniverse;
117     double timeLimit = 157788000;
118     double deltaT = 25000;
119     double time = 0;
120     fin >> threeBodyUniverse;
121     threeBodyUniverse.setTimeLimit(timeLimit);
122
123     while (time < timeLimit) {
124         threeBodyUniverse.step(deltaT);
125         time += deltaT;
126     }
127     BOOST_CHECK_EQUAL(threeBodyUniverse.getN(), 3);
128     BOOST_CHECK_EQUAL(threeBodyUniverse.getR(), 1.25e11);
129     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(0)->getPosX(), -1.28106e13,
    1);    // -1.28106e+13
130     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(0)->getPosY(), -2.78618e13,
    1);    // -2.78618e+13
131     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(0)->getVelX(), -167856, 1);
            // -167856
132     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(0)->getVelY(), -364999, 1);
         // -364999
133     BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getMass(), 5.974e24);
134     BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(0)->getBodyFileName(), "./
    src/earth.gif");
135
136     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(1)->getPosX(), 2.70375e12,
    1);      // 2.70375e+12
137     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(1)->getPosY(), 4.94576e12,
    1);      // 4.94576e+12
138     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(1)->getVelX(), 154792, 1);
        // 154792
139     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(1)->getVelY(), 284047.625,
    1);         // 284048
```

```
140     BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getMass(), 1.989e30);
141     BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(1)->getBodyFileName(), "./
        src/sun.gif");
142 
143     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(2)->getPosX(), -1.42527e13,
        1);    // -1.42527e+13
144     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(2)->getPosY(), 1.54206e13,
        1);     // 1.54206e+13
145     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(2)->getVelX(), -819958, 1);
            // -819958
146     BOOST_CHECK_CLOSE(threeBodyUniverse.getBody(2)->getVelY(), 885668, 1);
          // 885668
147     BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getMass(), 1.989e30);
148     BOOST_CHECK_EQUAL(threeBodyUniverse.getBody(2)->getBodyFileName(), "./
        src/sun.gif");
149 }
150 
151 BOOST_AUTO_TEST_CASE(planets0Steps) {
152     std::ifstream fin("./src/planets.txt");
153     Universe planetsUniverse;
154     fin >> planetsUniverse;
155     BOOST_CHECK_EQUAL(planetsUniverse.getN(), 5);
156     BOOST_CHECK_EQUAL(planetsUniverse.getR(), 2.50e11);
157     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosX(), 1.4960e+11, 1);
158     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosY(), 0, 1);
159     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelX(), 0, 1);
160     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelY(), 2.9800e+4, 1);
161     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getMass(), 5.974e+24);
162     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getBodyFileName(), "./src/
        earth.gif");
163 
164     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosX(), 2.2790e+11, 1);
165     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosY(), 0, 1);
166     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelX(), 0, 1);
167     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelY(), 2.4100e+4, 1);
168     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getMass(), 6.4190e+23);
169     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getBodyFileName(), "./src/
        mars.gif");
170 
171     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosX(), 5.7900e+10, 1);
172     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosY(), 0, 1);
173     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelX(), 0, 1);
174     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelY(), 4.7900e+4, 1);
175     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getMass(), 3.3020e+23);
176     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getBodyFileName(), "./src/
        mercury.gif");
177 
178     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosX(), 0, 1);
179     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosY(), 0, 1);
180     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelX(), 0, 1);
181     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelY(), 0, 1);
182     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getMass(), 1.9890e+30);
183     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getBodyFileName(), "./src/
        sun.gif");
184 
185     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosX(), 1.0820e+11, 1);
186     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosY(), 0, 1);
187     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelX(), 0, 1);
188     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelY(), 3.5000e+4, 1);
```

```
189     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getMass(), 4.8690e+24);
190     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getBodyFileName(), "./src/
        venus.gif");
191 }
192
193 BOOST_AUTO_TEST_CASE(planets1Step) {
194     std::ifstream fin("./src/planets.txt");
195     Universe planetsUniverse;
196     fin >> planetsUniverse;
197     planetsUniverse.step(25000);
198     BOOST_CHECK_EQUAL(planetsUniverse.getN(), 5);
199     BOOST_CHECK_EQUAL(planetsUniverse.getR(), 2.50e11);
200     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosX(), 1.4960e+11, 1);
201     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosY(), 7.4500e+08, 1);
202     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelX(), -1.4820e+02, 1)
        ;
203     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelY(), 2.9800e+4, 1);
204     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getMass(), 5.974e+24);
205     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getBodyFileName(), "./src/
        earth.gif");
206
207     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosX(), 2.2790e+11, 1);
208     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosY(), 6.0250e+08, 1);
209     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelX(), -6.3860e+01, 1)
        ;
210     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelY(), 2.4100e+4, 1);
211     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getMass(), 6.4190e+23);
212     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getBodyFileName(), "./src/
        mars.gif");
213
214     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosX(), 5.7875e+10, 1);
215     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosY(), 1.1975e+09, 1);
216     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelX(), -9.8933e+02, 1)
        ;
217     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelY(), 4.7900e+04, 1);
218     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getMass(), 3.3020e23);
219     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getBodyFileName(), "./src/
        mercury.gif");
220
221     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosX(), 3.3087e+01, 1);
222     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosY(), 0, 1);
223     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelX(), 1.3235e-03, 1);
224     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelY(), 0, 1);
225     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getMass(), 1.9890e30);
226     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getBodyFileName(), "./src/
        sun.gif");
227
228     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosX(), 1.0819e+11, 1);
229     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosY(), 8.7500e+08, 1);
230     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelX(), -2.8329e+02, 1)
        ;
231     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelY(), 3.5000e+04, 1);
232     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getMass(), 4.8690e24);
233     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getBodyFileName(), "./src/
        venus.gif");
234 }
235
236 BOOST_AUTO_TEST_CASE(planets2Step) {
237     std::ifstream fin("./src/planets.txt");
```

```
238     Universe planetsUniverse;
239     fin >> planetsUniverse;
240     planetsUniverse.step(25000);
241     planetsUniverse.step(25000);
242     BOOST_CHECK_EQUAL(planetsUniverse.getN(), 5);
243     BOOST_CHECK_EQUAL(planetsUniverse.getR(), 2.50e11);
244     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosX(), 1.4959e+11, 1);
245     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosY(), 1.4900e+09, 1);
246     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelX(), -2.9640e+02, 1)
        ;
247     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelY(), 2.9799e+04, 1);
248     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getMass(), 5.974e+24);
249     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getBodyFileName(), "./src/
        earth.gif");
250
251     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosX(), 2.2790e+11, 1);
252     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosY(), 1.2050e+09, 1);
253     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelX(), -1.2772e+02, 1)
        ;
254     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelY(), 2.4100e+04, 1);
255     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getMass(), 6.4190e+23);
256     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getBodyFileName(), "./src/
        mars.gif");
257
258     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosX(), 5.7826e+10, 1);
259     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosY(), 2.3945e+09, 1);
260     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelX(), -1.9789e+03, 1)
        ;
261     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelY(), 4.7880e+04, 1);
262     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getMass(), 3.3020e23);
263     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getBodyFileName(), "./src/
        mercury.gif");
264
265     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosX(), 9.9262e+01, 1);
266     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosY(), 2.8198e-01, 1);
267     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelX(), 2.6470e-03, 1);
268     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelY(), 1.1279e-05, 1);
269     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getMass(), 1.9890e30);
270     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getBodyFileName(), "./src/
        sun.gif");
271
272     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosX(), 1.0818e+11, 1);
273     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosY(), 1.7499e+09, 1);
274     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelX(), -5.6660e+02, 1)
        ;
275     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelY(), 3.4998e+04, 1);
276     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getMass(), 4.8690e24);
277     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getBodyFileName(), "./src/
        venus.gif");
278 }
279
280 BOOST_AUTO_TEST_CASE(planets3Step) {
281     std::ifstream fin("./src/planets.txt");
282     Universe planetsUniverse;
283     fin >> planetsUniverse;
284     planetsUniverse.step(25000);
285     planetsUniverse.step(25000);
286     planetsUniverse.step(25000);
287     BOOST_CHECK_EQUAL(planetsUniverse.getN(), 5);
```

```
288     BOOST_CHECK_EQUAL(planetsUniverse.getR(), 2.50e11);
289     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosX(), 1.4958e+11, 1);
290     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosY(), 2.2349e+09, 1);
291     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelX(), -4.4460e+02, 1)
    ;
292     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelY(), 2.9798e+04, 1);
293     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getMass(), 5.974e+24);
294     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getBodyFileName(), "./src/
    earth.gif");
295
296     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosX(), 2.2789e+11, 1);
297     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosY(), 1.8075e+09, 1);
298     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelX(), -1.9158e+02, 1)
    ;
299     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelY(), 2.4099e+04, 1);
300     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getMass(), 6.4190e+23);
301     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getBodyFileName(), "./src/
    mars.gif");
302
303     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosX(), 5.7752e+10, 1);
304     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosY(), 3.5905e+09, 1);
305     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelX(), -2.9682e+03, 1)
    ;
306     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelY(), 4.7839e+04, 1);
307     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getMass(), 3.3020e23);
308     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getBodyFileName(), "./src/
    mercury.gif");
309
310     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosX(), 1.9852e+02, 1);
311     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosY(), 1.1280e+00, 1);
312     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelX(), 3.9705e-03, 1);
313     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelY(), 3.3841e-05, 1);
314     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getMass(), 1.9890e30);
315     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getBodyFileName(), "./src/
    sun.gif");
316
317     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosX(), 1.0816e+11, 1);
318     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosY(), 2.6248e+09, 1);
319     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelX(), -8.4989e+02, 1)
    ;
320     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelY(), 3.4993e+04, 1);
321     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getMass(), 4.8690e24);
322     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getBodyFileName(), "./src/
    venus.gif");
323 }
324 /*
325 BOOST_AUTO_TEST_CASE(planets1Year) {
326     std::ifstream fin("./src/planets.txt");
327     Universe planetsUniverse;
328     fin >> planetsUniverse;
329     planetsUniverse.step(3.154e+7);
330     BOOST_CHECK_EQUAL(planetsUniverse.getN(), 5);
331     BOOST_CHECK_EQUAL(planetsUniverse.getR(), 2.50e11);
332     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosX(), 1.4959e+11, 1);
333     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getPosY(), -1.6531e+09, 1)
    ;
334     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelX(), 3.2949e+02, 1);
335     BOOST_CHECK_CLOSE(planetsUniverse.getBody(0)->getVelY(), 2.9798e+04, 1);
336     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getMass(), 5.974e+24);
```

```
337     BOOST_CHECK_EQUAL(planetsUniverse.getBody(0)->getBodyFileName(), "./src/
        earth.gif");
338
339     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosX(), -2.2153e+11, 1)
        ;
340     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getPosY(), -4.9263e+10, 1)
        ;
341     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelX(), 5.1805e+03, 1);
342     BOOST_CHECK_CLOSE(planetsUniverse.getBody(1)->getVelY(), -2.3640e+04, 1)
        ;
343     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getMass(), 6.4190e+23);
344     BOOST_CHECK_EQUAL(planetsUniverse.getBody(1)->getBodyFileName(), "./src/
        mars.gif");
345
346     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosX(), 3.4771e+10, 1);
347     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getPosY(), 4.5752e+10, 1);
348     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelX(), -3.8269e+04, 1)
        ;
349     BOOST_CHECK_CLOSE(planetsUniverse.getBody(2)->getVelY(), 2.9415e+04, 1);
350     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getMass(), 3.3020e23);
351     BOOST_CHECK_EQUAL(planetsUniverse.getBody(2)->getBodyFileName(), "./src/
        mercury.gif");
352
353     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosX(), 5.9426e+05, 1);
354     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getPosY(), 6.2357e+06, 1);
355     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelX(), -5.8569e-02, 1)
        ;
356     BOOST_CHECK_CLOSE(planetsUniverse.getBody(3)->getVelY(), 1.6285e-01, 1);
357     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getMass(), 1.9890e30);
358     BOOST_CHECK_EQUAL(planetsUniverse.getBody(3)->getBodyFileName(), "./src/
        sun.gif");
359
360     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosX(), -7.3731e+10, 1)
        ;
361     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getPosY(), -7.9391e+10, 1)
        ;
362     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelX(), 2.5433e+04, 1);
363     BOOST_CHECK_CLOSE(planetsUniverse.getBody(4)->getVelY(), -2.3973e+04, 1)
        ;
364     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getMass(), 4.8690e24);
365     BOOST_CHECK_EQUAL(planetsUniverse.getBody(4)->getBodyFileName(), "./src/
        venus.gif");
366 }
367 */
```

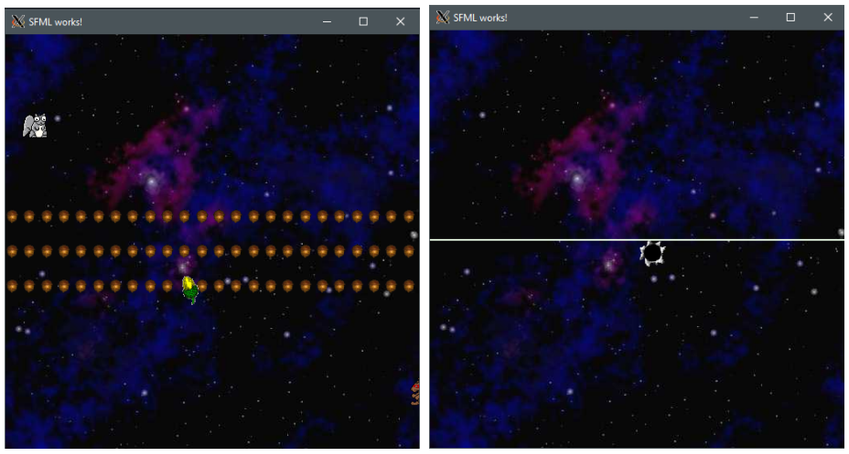


Figure 4.3: Other static examples

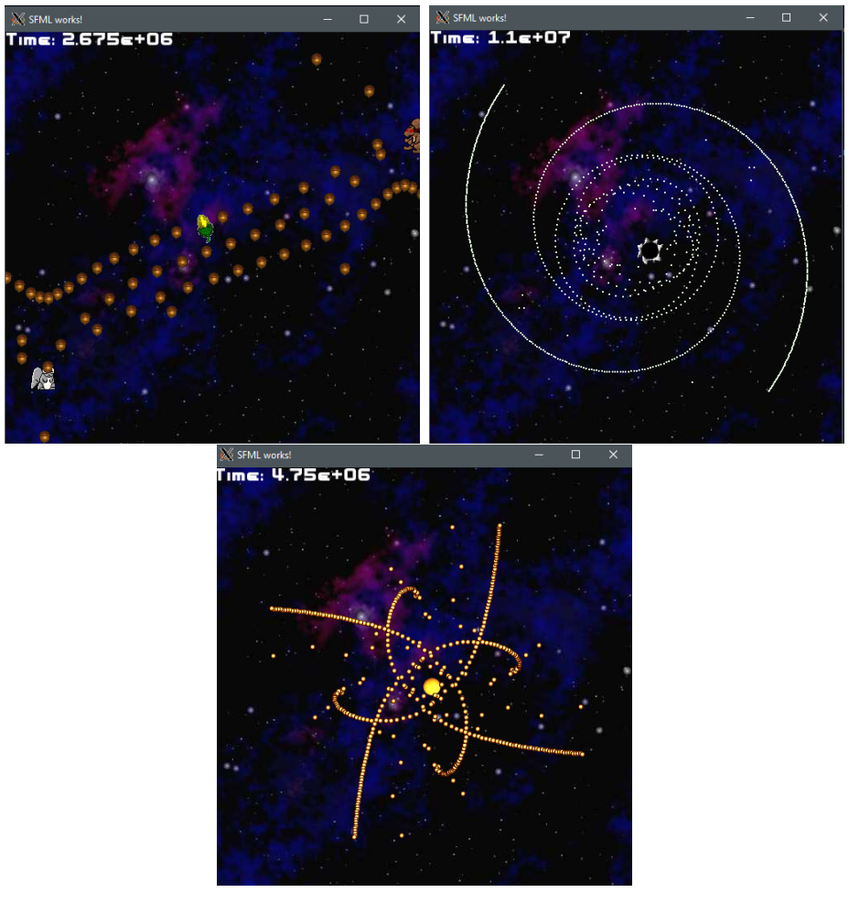


Figure 4.4: Other static examples

5 PS4: Sokoban

5.1 Discussion

For this project, we were tasked to implement Sokoban, a Japanese tile game where we play as a warehouse keeper that pushes boxes into their designated locations. The player wins when all the boxes are pushed into their locations. There're plenty of implementations of this game found, especially nowadays as mini puzzle games within larger ones. For this project specifically, we once again used the SFML library to implement our version of Sokoban in C++. As with the previous few assignments, this project was also split into parts A and B, one for the UI and one for the actual gameplay.

5.2 What I accomplished

In summary, I was able to get the game in a playable state. Although, there are a few major box glitches that occur in regards to the collision rules.

5.2.1 Part A: Static UI

For this part, I first created an object for the entire game on its own, which are the `Sokoban.hpp` and `Sokoban.cpp` files. Since the game state is represented by a bunch of text characters, the game object is implemented to store these characters in a singular long vector.

Data Structure

The interesting concept with this vector is how it's supposed to represent a 2D matrix, despite it being a single dimension structure. The key idea of making the long vector a 2D matrix is to draw each tile by the height h and width w values given by the level files, two dimensions of a map. However, since the vector is only one dimension, indexing needed to be calculated based on the formula $i = x + y * w$. Getting the x and y values of the matrix back are through $x = i\%w$ and $y = i/w$. With this way of converting 2D to 1D indexes, I was able to yield the results in Figure 5.1.

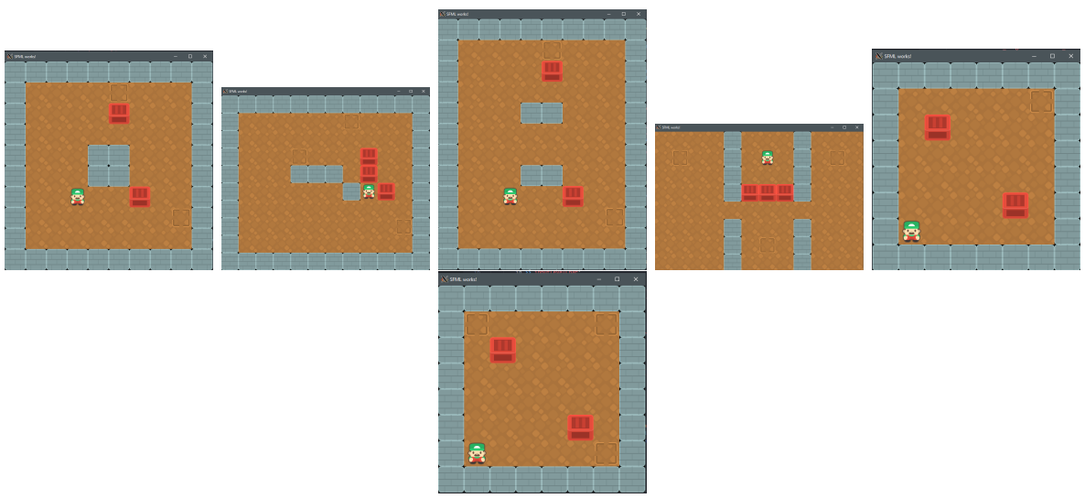


Figure 5.1: Levels Started

5.2.2 Part B: Gameplay

This part was when I needed to add collisions and player movement. Fortunately we were already given the functions we need, which we stubbed in the first part. The two key functions we needed for the game to be playable are the `movePlayer()` and `isWon()` functions. The `movePlayer()` function is interesting in that it handles not just the player's movements, but the box's movements as well. Logically, this makes sense as the boxes move dependently with how the player pushes them.

Algorithms

With this being the case, player movement and box movement are split by a single algorithm from the C++ algorithm library, the `find_if()` function. The player movement is coded with enumerations for the 4 cardinal directions, each key mapping to each direction. However, the box's movement is coded with a target position, and the position beyond the target position. This means that for every box the `find()` function finds, it must go through the calculations for deciding whether something is or isn't blocking its path. The win conditions are done in a nested for each loop, covering for the amount of boxes and locations to be filled in order to win. Some examples of the player winning the game are shown in the Figure 5.2.

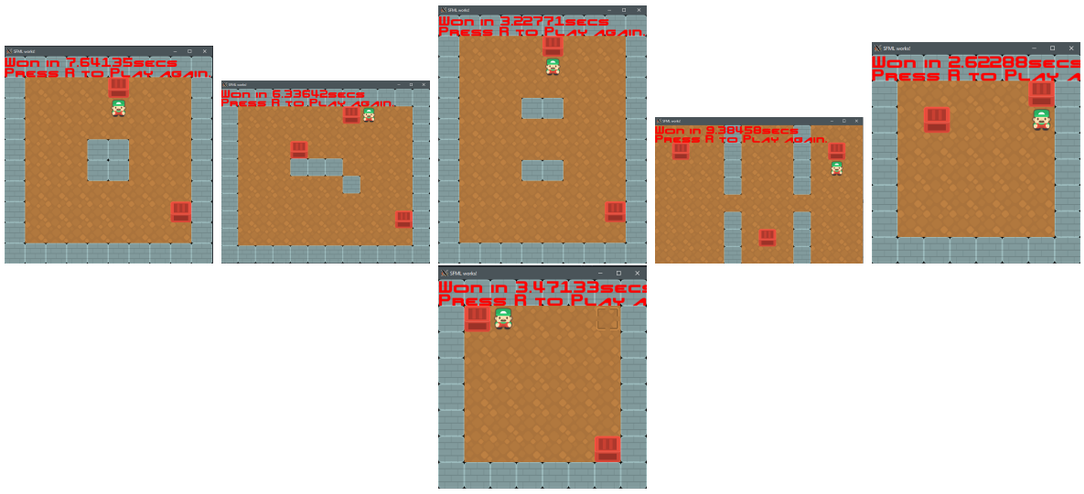


Figure 5.2: Levels Finished

5.3 What I already knew

I've played games that have some form of Sokoban built in, but have never seen any implementations of it until having done this assignment. Despite this, I still came in with some knowledge on drawable objects and moving around sprites with arrow keys. Of course, this is all due to having done PS0, which required us to use keystrokes to encode sprite movement. This is found in Section 1.

5.4 What I learned

This project along with some in-class activities were my introduction to the `<algorithms>` library for C++. With this, I've also gotten to brush up on C++'s lambda syntax. In this project's case, I learned how to write the syntax for the `find_if()` function as well as a lamda expression of which it accepts. I also brushed up on how to use SFML's Vector types in order to make my code more readable in regards to my calculations. It was also at this project where I figured out how to use lint. Overall, this project is the one that I've learned the most from.

5.5 Challenges

For the first half of part B, the challenge was to organize my calculations so that I could better visualize my conditions for the player and box movement. The rest was figuring out how to handle collisions for more than 2 boxes. I couldn't get a condition that compares each box individually. This results in box locations that're not adjacent to each other to ignore collision. For example, box 1 and 2, 2 and 3, but not 1 and 3.

5.6 Codebase

Makefile

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

```
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = Sokoban.hpp
# Your compiled .o files
OBJECTS = Sokoban.o
# The name of your program
PROGRAM = Sokoban

.PHONY: all clean lint

all: $(PROGRAM)

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

main.cpp

```
// Copyright 2023 Leonard Nguyen
#include <iostream>
#include <fstream>
#include <sstream>

#include "Sokoban.hpp"

int main() {
    sf::Clock clock;
    sf::Time t, bestTime;

    sf::Font font;
    if (!font.loadFromFile("./src/NiseJSRF.ttf")) {} else {}
    sf::Text winMessage;
    winMessage.setFont(font);
    winMessage.setCharacterSize(32);
    winMessage.setFillColor(sf::Color::Red);
    winMessage.setStyle(sf::Text::Bold);

    Sokoban game;
    std::cin >> game;
    std::cout << game << std::endl;

    const unsigned int windowSizeX = game.width();
    const unsigned int windowSizeY = game.height();
    sf::RenderWindow window(sf::VideoMode(windowSizeX*Sokoban::TILE_SIZE,
    windowSizeY*Sokoban::TILE_SIZE), "SFML works!");

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
```

```
33                  window.close();
34              } else if (event.type == sf::Event::KeyPressed) {
35                  // restart key
36                  switch (event.key.code) {
37                      case sf::Keyboard::R:   // restart
38                          window.clear();
39                          game.restartGame();
40                          clock.restart();
41                          break;
42
43                      default: break;
44                  }
45
46                  // control player
47                  if (!game.isWon()) {
48                      t = clock.getElapsedTime();
49                      switch (event.key.code) {
50                          case sf::Keyboard::W:   // Up
51                              game.movePlayer(Up);
52                              break;
53                          case sf::Keyboard::Up:  // Up
54                              game.movePlayer(Up);
55                              break;
56
57                          case sf::Keyboard::S:   // Down
58                              game.movePlayer(Down);
59                              break;
60                          case sf::Keyboard::Down:    // Down
61                              game.movePlayer(Down);
62                              break;
63
64                          case sf::Keyboard::A:   // Left
65                              game.movePlayer(Left);
66                              break;
67                          case sf::Keyboard::Left:    // Left
68                              game.movePlayer(Left);
69                              break;
70
71                          case sf::Keyboard::D:   // Right
72                              game.movePlayer(Right);
73                              break;
74
75                          case sf::Keyboard::Right:   // Right
76                              game.movePlayer(Right);
77                              break;
78
79                          default: break;
80                      }
81                  }
82              }
83          }
84
85          window.clear();
86          window.draw(game);
87          if (game.isWon()) {
88              std::stringstream winString;
89              winString << "Won in " << t.asSeconds() << "secs" << std::endl
90              << "Press R to Play again.";
91              winMessage.setString(winString.str());
```

```
            window.draw(winMessage);
        }
        window.display();
    }

    return 0;
}
```

Sokoban.hpp

```
// Copyright 2023 Leonard Nguyen
#pragma once

#include <iostream>
#include <vector>
#include <memory>
#include <algorithm>
#include <SFML/Graphics.hpp>

enum Direction {Up, Down, Left, Right};

class Sokoban: public sf::Drawable {
 public:
    static const int TILE_SIZE = 64;

    // constructor(s)
    Sokoban();
    Sokoban(int height, int width);

    friend std::istream &operator>>(std::istream &in, Sokoban &sokoban);
    friend std::ostream &operator<<(std::ostream &out, Sokoban &sokoban);

    void movePlayer(Direction dir);
    bool isWon() const;
    void restartGame();

    int width() {return _width;}
    int height() {return _height;}

 private:
    int _height, _width;
    std::vector<char> _tiles;

    sf::Vector2i _playerLocation;
    std::vector<sf::Vector2i> _boxLocations;
    std::vector<sf::Vector2i> _boxDropOffLocations;

    Direction _facing;
    // std::vector<sf::Texture> _playerTextures;
    sf::Texture _playerTexture;
    sf::Texture _emptySpaceTexture;
    sf::Texture _wallTexture;
    sf::Texture _boxTexture;
    sf::Texture _boxDropOffTexture;
    sf::Texture _inLocationTexture;
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
   const;
};
```

Sokoban.cpp

```cpp
// Copyright 2023 Leonard Nguyen
#include "Sokoban.hpp"

Sokoban::Sokoban() {
    _emptySpaceTexture.loadFromFile("./src/ground_01.png");
    _wallTexture.loadFromFile("./src/block_06.png");

    /*
    _playerTextures.resize(4);
    _playerTextures[Up].loadFromFile("./src/player_08.png");
    _playerTextures[Down].loadFromFile("./src/player_05.png");
    _playerTextures[Left].loadFromFile("./src/player_20.png");
    _playerTextures[Right].loadFromFile("./src/player_17.png");
    */

    _playerTexture.loadFromFile("./src/player_05.png");
    _boxTexture.loadFromFile("./src/crate_03.png");
    _boxDropOffTexture.loadFromFile("./src/ground_04.png");
}

std::istream &operator>>(std::istream &in, Sokoban &sokoban) {
    // std::vector<char> inputLevelLine;
    in >> sokoban._height >> sokoban._width;
    for (int i = 0; i < sokoban._width*sokoban._height; i++) {
        char inputLevelChar;
        in >> inputLevelChar;
        if (inputLevelChar == '@') {
            sokoban._playerLocation.x = i%sokoban._width;
            sokoban._playerLocation.y = i/sokoban._width;
        } else if (inputLevelChar == 'A') {
            sf::Vector2i boxCoords;
            boxCoords.x = i%sokoban._width;
            boxCoords.y = i/sokoban._width;
            sokoban._boxLocations.push_back(boxCoords);
        } else if (inputLevelChar == 'a') {
            sf::Vector2i boxDropOffCoords;
            boxDropOffCoords.x = i%sokoban._width;
            boxDropOffCoords.y = i/sokoban._width;
            sokoban._boxDropOffLocations.push_back(boxDropOffCoords);
        }
        sokoban._tiles.push_back(inputLevelChar);
    }
    return in;
}

std::ostream &operator<<(std::ostream &out, Sokoban &sokoban) {
    out << "Width: " << sokoban._width << std::endl;
    out << "Height: " << sokoban._height << std::endl;
    out << "Level Stream: ";
    for (auto i : sokoban._tiles) {
        out << i;
    }
    out << std::endl<< "Player Location: ("
    << sokoban._playerLocation.x << ", " << sokoban._playerLocation.y
    << ")";
    return out;
}

```

```
59 void Sokoban::movePlayer(Direction dir) {
60     sf::Vector2i moveDir;
61     switch (dir) {
62         case Up:
63             _facing = Up;
64             moveDir = {0, -1};
65             break;
66         case Down:
67             _facing = Down;
68             moveDir = {0, 1};
69             break;
70         case Left:
71             _facing = Left;
72             moveDir = {-1, 0};
73             break;
74         case Right:
75             _facing = Right;
76             moveDir = {1, 0};
77             break;
78         default:
79             throw std::invalid_argument("Unknown enum");
80     }
81     sf::Vector2i targetPos = _playerLocation + moveDir;
82     if (targetPos.y < 0 || targetPos.x < 0 || targetPos.y >= _height ||
    targetPos.x >= _width) {
83         return;
84     }
85     char targetSymbol = _tiles.at(targetPos.y * _width + targetPos.x);
86     if (targetSymbol == '#') {
87         return;
88     }
89     // box movement
90     auto box = std::find_if(_boxLocations.begin(), _boxLocations.end(),
91     [targetPos](const sf::Vector2i &pos){return pos == targetPos;});
92     if (box != _boxLocations.end()) {
93         sf::Vector2i beyondBoxPos = targetPos + moveDir;
94         if (beyondBoxPos.y < 0 || beyondBoxPos.x < 0 ||
95         beyondBoxPos.y >= _height || beyondBoxPos.x >= _width) {
96             return;
97         }
98         if (_tiles.at(beyondBoxPos.y * _width + beyondBoxPos.x) == '#') {
99             return;
100        }
101        if (beyondBoxPos == *(box-1) || beyondBoxPos == *(box+1)) {     //
    fix this
102            return;
103        }
104        *box = beyondBoxPos;
105    }
106    _playerLocation = targetPos;
107 }
108
109 void Sokoban::restartGame() {
110     _boxLocations.clear();
111     for (int i = 0; i < _width*_height; i++) {
112         if (_tiles[i] == '@') {
113             _playerLocation.x = i%_width;
114             _playerLocation.y = i/_width;
115         } else if (_tiles[i] == 'A') {
```

```
116             sf::Vector2i boxCoords;
117             boxCoords.x = i%_width;
118             boxCoords.y = i/_width;
119             _boxLocations.push_back(boxCoords);
120         }
121     }
122 }
123
124 bool Sokoban::isWon() const {
125     size_t count = 0;
126     // auto box = std::find_if(_boxLocations.begin(), _boxLocations.end(),
    []());
127
128     for (auto box : _boxLocations) {
129         for (auto dropOff : _boxDropOffLocations) {
130             if (box == dropOff) {
131                 count++;
132                 if (count >= _boxLocations.size()) {
133                     return true;
134                 } else if (count >= _boxDropOffLocations.size()) {
135                     return true;
136                 }
137             }
138         }
139     }
140     return _boxDropOffLocations.empty();    // returns false unless no
    dropoff locs
141 }
142
143 void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
144     sf::Sprite overworldSprite;
145     for (int i = 0; i < _height; i++) {
146         for (int j = 0; j < _width; j++) {
147             if (_tiles.at(i* _width +j) == '#') {   // wall
148                 overworldSprite.setTexture(_wallTexture);
149             } else if ((_tiles.at(i* _width +j) == '.') ||
150             (_tiles.at(i* _width +j) == '@') ||
151             (_tiles.at(i* _width +j) == 'A')) {      // ground
152                 overworldSprite.setTexture(_emptySpaceTexture);
153             } else if (_tiles.at(i * _width + j) == 'a') {      // box drop-
    off
154                 overworldSprite.setTexture(_boxDropOffTexture);
155             }
156             overworldSprite.setPosition(j * TILE_SIZE,  i * TILE_SIZE);
157             target.draw(overworldSprite);
158         }
159     }
160
161     sf::Sprite boxSprite;
162     boxSprite.setTexture(_boxTexture);
163     for (auto & boxCoord : _boxLocations) {
164         boxSprite.setPosition(boxCoord.x*TILE_SIZE, boxCoord.y*TILE_SIZE);
165         target.draw(boxSprite);
166     }
167
168     sf::Sprite playerSprite;
169     playerSprite.setTexture(_playerTexture);
```

```
170     playerSprite.setPosition(_playerLocation.x * TILE_SIZE, _playerLocation.
        y * TILE_SIZE);
171     target.draw(playerSprite);
172 }
```

6 PS5: DNA Sequence Alignment

6.1 Discussion

This assignment was to develop a DNA Sequence Aligner program. Basically, it's a program that takes two sequences of characters and then edits the distances in a way that allows for the most optimal "alignment" between the two sequences. Despite it being named a "DNA Sequence Aligner", the program doesn't necessarily have to exclusively use the proteins *A, C, G, T* but could also take any sequence of strings provided that each string is divided by a white space. Of course, the main intention of such a program is to compare sequences of DNA strands. The actual instructions for this program includes a better explaination of the biological problem as well as an example of how such a program can be used. For our sake, the purpose of this project is to give us a way to view processing time and data management, especially for big "real life" data such as those from the ecoli files. We used valgrind to keep track of the memory usage.

6.2 What I accomplished

For this assignment, we were tasked to implement the EDistance object. The functions and fields for it are as follows:

- `penalty()`: static function returning a cost value 0 or 1
- `min3()`: custom static function returning the smallest of 3 numbers
- `optDistance()`: return the cost of the most optimal alignment
- `alignment()`: return the alignment in the form of string columns

Algorithm and Memory

The main structure used is the vector of column vectors, as suggested by the project's instructions. This acts as our vector that allocates memory according to string length. This is different from how I handled the 2D matrix in Chapter 5, since the sequences in this assignment were split by a whitespace per sample. This is also important for how we read through the vector in our `optDistance()` function. As far as algorithms are concerned, I performed my `optDistance()` and `alignment()` functions using nested loops, which is likely not optimal for the program's runtime. However, the logic behind the `optDistance()` function does return the cost of the most optimal sequence alignment.

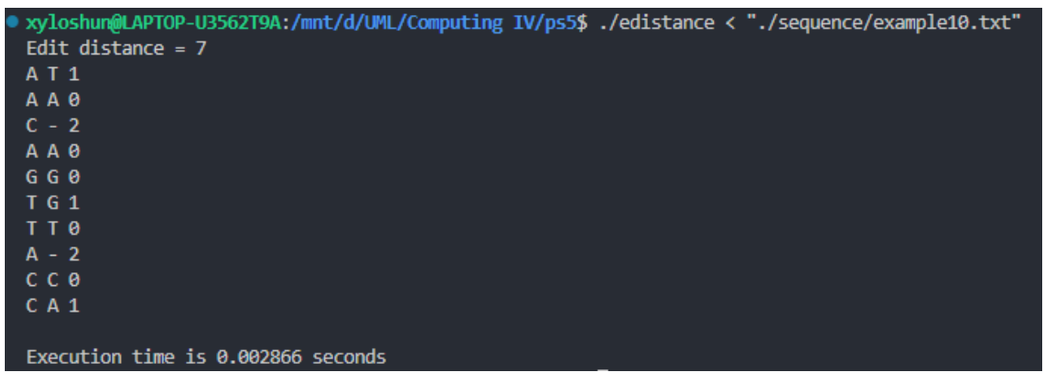


Figure 6.1: Results from the example.txt sample

6.3 What I already knew

The most I know about implementing this program comes from what I've done in Sokoban in Section 5. More specifically, I had an idea for how to iterate through the 2D vector and how to split the conditions to fit dynamic programming. I also knew how to implement a min3 function, taught to me back in Computing II. Calculating runtimes and memory was something I did for Analysis of Algorithms, although for this particular assignment, I found it to be a bit of a challenge.

6.4 What I learned

Dynamic programming was a big topic for this assignment, for it's a coding practice that very much reduces the time it takes for the DNA alignment to run as a program. Despite having done some form of dynamic programming in the past, I also wasn't aware that what I've done was called "dynamic programming". I also learned of how this assignment could've been done with recursion, but also how it'd be terrible for bigger samples such as the ecoli. Overall, the lesson to be learned here was to figure out how to see, analyze, and calculate runtimes for programs such as this.

6.5 Challenges

My personal challenge was figuring out DNA alignments as a whole. A lot of my work came down to inputting sequences into existing DNA alignment programs and seeing what gets outputted. I also needed to read the instructions more thoroughly to be sure that the conditions I put for my `optDistance()` were what was asked. I also had a bit of a challenge calculating the memory usage of my code.

6.6 Codebase

Makefile

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
# Your .hpp files
DEPS = EDistance.hpp
# Your compiled .o files
OBJECTS = EDistance.o
# The name of your program
PROGRAM = EDistance

.PHONY: all clean lint

all: $(PROGRAM) test

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

test: test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) test

lint:
    cpplint *.cpp *.hpp
```

main.cpp

```
// Copyright 2023 Leonard Nguyen
#include <SFML/System.hpp>

#include "EDistance.hpp"

int main() {
```

```
    sf::Clock clock;
    std::string sequence1, sequence2;
    std::cin >> sequence1 >> sequence2;

    EDistance editDistance(sequence1, sequence2);

    std::cout << "Edit distance = " << editDistance.optDistance() << std::
   endl;
    std::cout << editDistance.alignment() << std::endl;

    sf::Time t = clock.getElapsedTime();
    std::cout << "Execution time is " << t.asSeconds() << " seconds " << std
   ::endl;
}
```

EDistance.hpp

```
// Copyright 2023 Leonard Nguyen
#pragma once

#include <iostream>
#include <string>
#include <vector>

class EDistance {
 public:
    // constructor
    EDistance(std::string input1, std::string input2);

    static int penalty(char a, char b);
    static int min3(int a, int b, int c);

    int optDistance();          // return optimal distance
    std::string alignment();    // return alignment

 private:
    std::vector<std::vector<int>> matrix;
    std::string _rowString;
    std::string _columnString;
};
```

EDistance.cpp

```
// Copyright 2023 Leonard Nguyen
#include <sstream>
#include "EDistance.hpp"

EDistance::EDistance(std::string input1, std::string input2) {
    _rowString = input1;
    _columnString = input2;
    std::vector<int> columnInput;

    // column allocation
    for (int i = 0; i < static_cast<int>(_columnString.length())+1; i++) {
        columnInput.push_back(0);
    }
    for (int j = 0; j < static_cast<int>(_rowString.length())+1; j++) {
        matrix.push_back(columnInput);
```

```
16      }
17  }
18
19  int EDistance::penalty(char a, char b) {
20      if (a == b) {
21          return 0;
22      } else {
23          return 1;
24      }
25  }
26
27  // basic minimum of 3 ints function
28  int EDistance::min3(int a, int b, int c) {
29      int min;
30      if (a < b) {
31          min = a;
32          if (c < min) {
33              min = c;
34              return min;
35          } else {
36              return min;
37          }
38      } else {
39          min = b;
40          if (c < min) {
41              min = c;
42              return min;
43          } else {
44              return min;
45          }
46      }
47  }
48
49  int EDistance::optDistance() {
50      for (int i = matrix.size()-1; i >= 0; i--) {
51          for (int j = matrix[i].size()-1; j >= 0; j--) {
52              if ((i == static_cast<int>(matrix.size())-1) &&
53              (j == static_cast<int>(matrix[i].size())-1)) {
54                  matrix[i][j] = 0;
55              } else if (i == static_cast<int>(matrix.size())-1) {
56                  matrix[i][j] = matrix[i][j+1] + 2;
57              } else if (j == static_cast<int>(matrix[i].size())-1) {
58                  matrix[i][j] = matrix[i+1][j] + 2;
59              } else {
60                  matrix[i][j] = min3(matrix[i+1][j+1] + penalty(_rowString[i
    ], _columnString[j]),
61                  matrix[i+1][j] + 2, matrix[i][j+1] + 2);
62              }
63          }
64      }
65      return matrix[0][0];
66  }
67
68  std::string EDistance::alignment() {
69      int i = 0;
70      int j = 0;
71      std::stringstream alignedSequence;
72
```

```
    while ((i < static_cast<int>(matrix.size())-1) || (j < static_cast<int>(
    matrix[0].size())-1)) {
        if ((i < static_cast<int>(matrix.size())-1) &&
        (j < static_cast<int>(matrix[0].size())-1) &&
        (matrix[i+1][j+1] <= matrix[i+1][j]+1) &&
        (matrix[i+1][j+1]+1 <= matrix[i][j+1]+1)) {
            alignedSequence << _rowString[i] << " "
            << _columnString[j] << " "
            << matrix[i][j] - matrix[i+1][j+1]
            << std::endl;
            i++;
            j++;
        } else if (((i < static_cast<int>(matrix.size())-1) &&
        (matrix[i+1][j] <= matrix[i][j+1])) ||
        (j == static_cast<int>(matrix.size())-1)) {
            alignedSequence << _rowString[i] << " "<< "-" << " "
            << matrix[i][j] - matrix[i+1][j]
            << std::endl;
            i++;
        } else {
            alignedSequence << "-" << " "
            << _columnString << " "
            << matrix[i][j] - matrix[i][j+1]
            << std::endl;
            j++;
        }
    }

    return alignedSequence.str();
}
```

test.cpp

```
// Copyright 2023 Leonard Nguyen
#include <iostream>
#include <string>
#include <fstream>

#include "EDistance.hpp"

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

BOOST_AUTO_TEST_CASE(penaltyTest1) {
    BOOST_CHECK_EQUAL(EDistance::penalty('A', 'B'), 1);
    BOOST_CHECK_EQUAL(EDistance::penalty('B', 'A'), 1);
    BOOST_CHECK_EQUAL(EDistance::penalty('A', 'A'), 0);
}

BOOST_AUTO_TEST_CASE(min3Test1) {
    BOOST_CHECK_EQUAL(EDistance::min3(1, 2, 3), 1);
    BOOST_CHECK_EQUAL(EDistance::min3(0, 1, 2), 0);
    BOOST_CHECK_EQUAL(EDistance::min3(0, 0, 1), 0);
    BOOST_CHECK_EQUAL(EDistance::min3(1, 0, 0), 0);
    BOOST_CHECK_EQUAL(EDistance::min3(3, 2, 1), 1);
}

BOOST_AUTO_TEST_CASE(editDistanceTest1) {
```

```
27      EDistance example10("AACAGTTACC", "TAAGGTCA");
28      BOOST_CHECK_EQUAL(example10.optDistance(), 7);
29
30      EDistance endgaps7("atattat", "tattata");
31      BOOST_CHECK_EQUAL(endgaps7.optDistance(), 4);
32
33      EDistance fli8("CAGCTACAA", "CAGACAA");
34      BOOST_CHECK_EQUAL(fli8.optDistance(), 4);
35  }
36
37  BOOST_AUTO_TEST_CASE(editDistanceTest2) {       // tests with letters other
        than proteins
38      EDistance amongus("amongus", "amongus");
39      BOOST_CHECK_EQUAL(amongus.optDistance(), 0);
40
41      EDistance amongusWithGap("amongus", "amongsus");
42      BOOST_CHECK_EQUAL(amongusWithGap.optDistance(), 2);
43
44      EDistance johnnyTest("Johnny", "Test");
45      BOOST_CHECK_EQUAL(johnnyTest.optDistance(), 8);
46  }
47
48  /*
49  BOOST_AUTO_TEST_CASE(fileInputEditDistanceTest1) {
50      std::string sequence1, sequence2;
51      std::cin >> sequence1 >> sequence2;
52
53      // 28284 took a while
54      EDistance ecoli28284(sequence1, sequence2);
55      BOOST_CHECK_EQUAL(ecoli28284.optDistance(), 118);
56  }
57  */
```

7 PS7: Kronos Log Parsing

7.1 Discussion

This assignment was to use regular expressions, or "regex" to parse through log files. More specifically, the log files we're tasked to parse were for the Kronos InTouch device. Essentially, we're taking the logs found in the device and reviewing some of the information the project instructions ask for. For this project, we were asked to output the device's boot up timing by parsing through some log files, and storing the information in a report file, ending in the extension `.rpt`. Since we're tracking the boot up timing, the instructions asked us to read the first logging message: `(log.c.166)`. This is when the booting starts. The log entry that shows a boot up has been complete (successful) is shown here: `oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080` Boot ups can also fail, which don't have any log entries. When this is the case, we just write into the report saying the boot up is `incomplete`.

7.2 What I accomplished

The main feature of this assignment is the regex search loop. The loop takes a getline condition that stores each line of a log file until the end. However, there are two checks in the loop, one to check for the start message, and the other for the complete message. The interesting thing about these checks is the usage of `regex_search()` functions. The basic idea is if the regex_search finds (log.c.166) server started, it outputs the information related to the start of the boot up. If regex_search finds `oejs.AbstractConnector:Started SelectChannelConnector`, then the boot up is detected as "complete". The information on this is also outputted. The output is stored in a stringstream, all to eventually be printed as a separate group from the header created for extra credit.

Regex

There're two regular expressions used in this project. One is the start, the other is the complete message:

$$\backslash\backslash(log.c.166\backslash\backslash)server started$$

$$(oejs.AbstractConnector : StartedSelectChannelConnector)$$

The first regex uses "\\" for both ends of the parentheses to indicate they're part of the string we're looking for. Setting the regex "(log.c.166) server started" without "\\" groups the log.c.166 into one string, completely ignoring the required parentheses. This leads to how parentheses are handled in endRegex. Here, the parentheses are used to group the entire string `oejs.AbstractConnector:Started SelectChannelConnector` as something to be looked for, as opposed to it searching for parts of the string in the log file. With this, I was able to produce some reports that look identical to some of the examples given for us to replicate, shown here:

```
1  Device Boot Report
2
3  InTouch log file: device5_intouch.log
4  Lines Scanned: 41855
5
6  Device boot count: initiated = 25, completed: 10
7
8
9  === Device boot ===
10 31063(device5_intouch.log): 2014-01-26 09:55:07 Boot Start
11 31176(device5_intouch.log): 2014-01-26 09:58:04 Boot Complete
12     Boot Time: 177000ms
13
14 === Device boot ===
15 31274(device5_intouch.log): 2014-01-26 12:15:18 Boot Start
16 **** Incomplete boot ****
17
```

```
=== Device boot ===
31293(device5_intouch.log): 2014-01-26 14:02:39 Boot Start
31401(device5_intouch.log): 2014-01-26 14:05:24 Boot Complete
    Boot Time: 165000ms

=== Device boot ===
32623(device5_intouch.log): 2014-01-27 12:27:55 Boot Start
**** Incomplete boot ****

=== Device boot ===
32641(device5_intouch.log): 2014-01-27 12:30:23 Boot Start
**** Incomplete boot ****

=== Device boot ===
32656(device5_intouch.log): 2014-01-27 12:32:51 Boot Start
**** Incomplete boot ****

=== Device boot ===
32674(device5_intouch.log): 2014-01-27 12:35:19 Boot Start
**** Incomplete boot ****

=== Device boot ===
32693(device5_intouch.log): 2014-01-27 14:02:38 Boot Start
32801(device5_intouch.log): 2014-01-27 14:05:21 Boot Complete
    Boot Time: 163000ms

=== Device boot ===
33709(device5_intouch.log): 2014-01-28 12:44:17 Boot Start
**** Incomplete boot ****

=== Device boot ===
33725(device5_intouch.log): 2014-01-28 14:02:33 Boot Start
33833(device5_intouch.log): 2014-01-28 14:05:15 Boot Complete
    Boot Time: 162000ms

=== Device boot ===
34594(device5_intouch.log): 2014-01-29 12:43:07 Boot Start
**** Incomplete boot ****

=== Device boot ===
34613(device5_intouch.log): 2014-01-29 14:02:35 Boot Start
34721(device5_intouch.log): 2014-01-29 14:05:19 Boot Complete
    Boot Time: 164000ms

=== Device boot ===
37428(device5_intouch.log): 2014-01-30 12:43:05 Boot Start
**** Incomplete boot ****

=== Device boot ===
37447(device5_intouch.log): 2014-01-30 14:02:40 Boot Start
37555(device5_intouch.log): 2014-01-30 14:05:22 Boot Complete
    Boot Time: 162000ms

=== Device boot ===
38258(device5_intouch.log): 2014-01-31 14:02:33 Boot Start
38366(device5_intouch.log): 2014-01-31 14:05:16 Boot Complete
    Boot Time: 163000ms

=== Device boot ===
```

```
39150(device5_intouch.log): 2014-02-01 12:39:38 Boot Start
**** Incomplete boot ****

=== Device boot ===
39166(device5_intouch.log): 2014-02-01 12:42:07 Boot Start
**** Incomplete boot ****

=== Device boot ===
39182(device5_intouch.log): 2014-02-01 14:02:32 Boot Start
39290(device5_intouch.log): 2014-02-01 14:05:16 Boot Complete
    Boot Time: 164000ms

=== Device boot ===
40288(device5_intouch.log): 2014-02-02 14:02:39 Boot Start
40397(device5_intouch.log): 2014-02-02 14:05:31 Boot Complete
    Boot Time: 172000ms

=== Device boot ===
41615(device5_intouch.log): 2014-02-03 12:35:55 Boot Start
**** Incomplete boot ****

=== Device boot ===
41633(device5_intouch.log): 2014-02-03 12:38:22 Boot Start
**** Incomplete boot ****

=== Device boot ===
41648(device5_intouch.log): 2014-02-03 12:40:48 Boot Start
**** Incomplete boot ****

=== Device boot ===
41666(device5_intouch.log): 2014-02-03 12:43:17 Boot Start
**** Incomplete boot ****

=== Device boot ===
41684(device5_intouch.log): 2014-02-03 12:45:46 Boot Start
**** Incomplete boot ****

=== Device boot ===
41694(device5_intouch.log): 2014-02-03 14:02:34 Boot Start
41802(device5_intouch.log): 2014-02-03 14:05:18 Boot Complete
    Boot Time: 164000ms
```

7.3 What I already knew

I've worked with regular expressions in a Foundations of Computer Science class. However, it took a while for me to relearn them for this project. I also previously learned how to write output into files during the time I've searched how to read file input for the previous projects. Other than these, there wasn't much prequisite knowledge I had going into this project.

7.4 What I learned

This project was my chance to brush up on regular expressions. However, I also learned of boost's time and date libraries and picked up some ways to use them for this project. To start, I picked up the `datetime.cpp` and `stdin_boost.cpp` files from the project's instructions. From here, I spent the majority of development time messing around with different inputs to give to the `stdin_boost.cpp` file. Using this is my new tool for reviewing regular expressions, which does help in helping me learn and review regular expressions. I also used the `datetime.cpp` file to see how boost's time formatting works.

7.5 Challenges

Review regular expressions and using the date and time libraries from boost are the two challenges I needed to get through in order to complete the assignment. Although I have learned a decent amount from the files, I still needed to take most of development time to be sure that my program both returns the proper time formats, and for my regular expressions to be as they're intended to be written.

7.6 Codebase

Makefile

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lboost_regex -lboost_date_time
# Your .hpp files
DEPS =
# Your compiled .o files
OBJECTS =
# The name of your program
PROGRAM = ps7

.PHONY: all clean lint

all: $(PROGRAM)

%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp
```

main.cpp

```
// Copyright 2023 Leonard Nguyen

// http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/gregorian.html
// http://www.boost.org/doc/libs/1_58_0/doc/html/date_time/posix_time.html

#include <iostream>
#include <string>
#include <fstream>
#include <regex>
#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

// using boost::gregorian::date;
using boost::gregorian::date_duration;
// using boost::gregorian::date_period;
// using boost::gregorian::from_simple_string;

using boost::posix_time::ptime;
using boost::posix_time::time_facet;
```

```
20 using boost::posix_time::time_input_facet;
21 using boost::posix_time::time_duration;
22 using boost::posix_time::time_from_string;
23
24 int main(int argc, char *argv[]) {
25     std::string inputFileName = argv[1];
26     std::string outputFileName = inputFileName + ".rpt";
27
28     std::ifstream fin(inputFileName);
29     if (!fin.is_open()) {
30         std::cout << "error: " << "Please enter a log file name in the local
    directory."
31         << std::endl;
32         return -1;
33     } else {}
34     std::ofstream fout(outputFileName);
35
36     ptime startTimeStamp, endTimeStamp;
37
38     std::regex startRegex("\\(log.c.166\\) server started");
39     std::regex endRegex("(oejs.AbstractConnector:Started
   SelectChannelConnector)");
40     std::string logLineContainer;
41
42     int initiated = 0;
43     int completed = 0;
44     int linesScanned = 0;
45     bool isBooting = false;
46     std::stringstream finalOutput;
47
48     // parse through file
49     while (getline(fin, logLineContainer)) {
50         linesScanned++;
51
52         // find (log.c.166)
53         if (regex_search(logLineContainer, startRegex)) {
54             initiated++;
55             std::stringstream dateStream(logLineContainer.substr(0, 19));
56             // dateStream.imbue(std::locale(dateStream.getloc(),
57             // new time_input_facet("%Y-%m-%d %H:%M:%S")));
58             startTimeStamp = time_from_string(dateStream.str());
59
60             if (isBooting == true) {    // check if boot is complete
61                 finalOutput << "**** Incomplete boot **** " << std::endl <<
   std::endl;
62             } else {
63                 isBooting = true;
64             }
65
66             finalOutput << "=== Device boot ===" << std::endl
67             << linesScanned << "("<< inputFileName << "): "
68             << dateStream.str() << " Boot Start" << std::endl;
69         }
70
71         // find oejs.AbstractConnector:Started SelectChannelConnector
72         if (regex_search(logLineContainer, endRegex)) {
73             completed++;
74             std::stringstream dateStream(logLineContainer.substr(0, 19));
75             // dateStream.imbue(std::locale(dateStream.getloc(),
```

```
76              // new time_input_facet("%Y-%m-%d %H:%M:%S")));
77              endTimeStamp = time_from_string(dateStream.str());
78              isBooting = false;  // default false
79
80              finalOutput << linesScanned << "(" << inputFileName << "): "
81              << dateStream.str() << " Boot Complete" << std::endl;
82
83              time_duration duration = endTimeStamp - startTimeStamp;
84              finalOutput << "\tBoot Time: " << duration.total_milliseconds()
    << "ms"
85              << std::endl << std::endl;
86          }
87      }
88
89      // print EVERYTHING
90      fout << "Device Boot Report" << std::endl << std::endl;
91      fout << "InTouch log file: " << inputFileName << std::endl;
92      fout << "Lines Scanned: " << linesScanned << std::endl << std::endl;
93      fout << "Device boot count: initiated = " <<  initiated << ", completed:
     " << completed
94      << std::endl << std::endl << std::endl;
95      fout << finalOutput.str();
96
97      return 0;
98  }
```