Braden Maillet
COMP IV Sec 204: Project Portfolio
Spring 2024

**Contents:**



**8** Credits

Time to complete:15 hours

# 0 PS0: Hello SFML

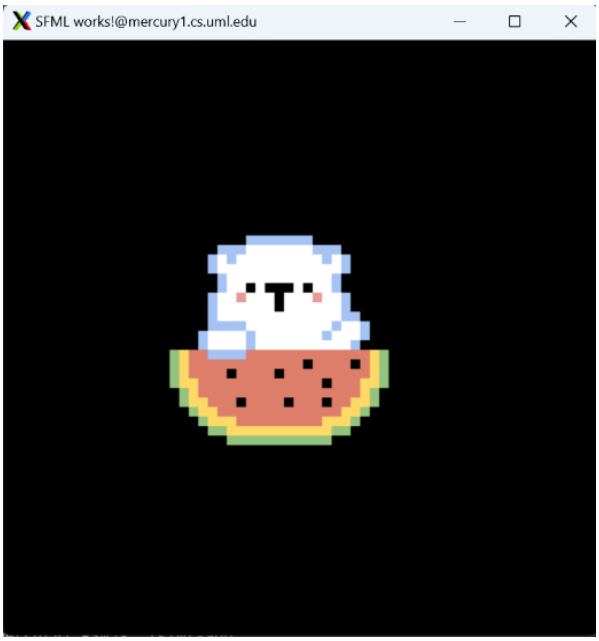
## 0.1 General Discussion

The goal of this project was to set up the SFML library on my machine aswell as start to familiarize myself with SFML. What was given to begin the project was a short program that created a green circle in the display window of SFML. If this ran correctly then SFML must have been correctly set up on my machine. Once this all was done a secondary goal was provided to make the sprite within the given code respond to keystrokes. I decided to give the sprite that was responsible for the image the ability to move. It responded to a series of if statements in what is known as “the game loop” within the main function. The sprite would then move to the keystrokes of W, A, S, D. The next goal within the project was to add some other fun feature to the sprite. I added the ability for the sprite to rotate 90 degrees when the left mouse was pressed.

## 0.2 What Was learned

I learned a significant amount about SFML and some of the classes involved in the library. I also developed some knowledge about the need for a game loop within an SFML program when trying to create an environment that is constantly changing. This was a very simple project, so no advanced algorithms were used nor any examples of OOP.

## 0.3 Output



```
 1: #include <iostream>
 2: #include <SFML/Graphics.hpp>
 3:
 4:
 5: int main() {
 6:     sf::RenderWindow window(sf::VideoMode(500, 500), "SFML works!");
 7:     sf::CircleShape shape(100.f);
 8:     sf::Vector2f vect(100.0f, 100.0f);
 9:     sf::Sprite sprite;
10:     sf::Texture spriteText;
11:     sprite.setPosition(vect);
12:     if (spriteText.loadFromFile("Pixel-Art-PNG-Isolated-File.png") == false)
13:         return EXIT_FAILURE;
14:     sprite.setTexture(spriteText);
15:     while (window.isOpen()) {
16:         sf::Event event;
17:     float angle = 0;
18:         while (window.pollEvent(event)) {
19:         if (event.type == sf::Event::Closed)
20:            window.close();
21:         }
22:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::A)) {
23:           sprite.move(-3.0f, 0.0f);
24:         }
25:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::W)) {
26:           sprite.move(0.0f, -3.0f);
27:         }
28:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::S)) {
29:           sprite.move(0.0f, 3.0f);
30:         }
31:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::D)) {
32:           sprite.move(3.0f, 0.0f);
33:         }
34:         if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
35:           angle += 90;
36:           sprite.rotate(angle);
37:         }
38:     window.clear();
39:     window.draw(sprite);
40:     window.display();
41:     }
42:     return 0;
43: }
44:
```

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

# 1a PS1: Linear Feedback Shift Register

## 1a.1 General Discussion

This project implements a Fibonacci LFSR to encode data. In this case it is encoded or shifted by placing “taps” taps on bits 10, 12, 13 and the largest bit. These taps take the bits in the given location and XOR them. If the XOR returns one, then when the 16 bit piece of data shifts left a one will shift into the 0 bit. In the alternative case if the XOR returns 0 then a zero will shift into the 0 bit(demonstrated in the diagram below). The project required the creation of the class “FibLFSR”. It also required the implementation of a value constructor, a step(), generate() and operator<< member/friend functions. I created the class so that it contained private data members int _size, unsigned int _binary and std::string _seed. The _size data member was so that the class could be expanded to binary strings greater than 16 bits. The _binary held a current representation of the binary string and the _seed held the original binary seed. I chose to make _seed a string because it allowed for easier printing when developing the operator<< for the class. I ended up creating some helper functions that converted between a bit string and a std::string. This was essential for outputting the class correctly. The step() function makes use of the idea that XOR returns 1 in the case that it receives an odd amount of 1 bits. It also was developed to allow for larger than 16-bit strings. Generate() uses a loop to repeatedly run this function and shift the return values into a bit string that it will return itself.

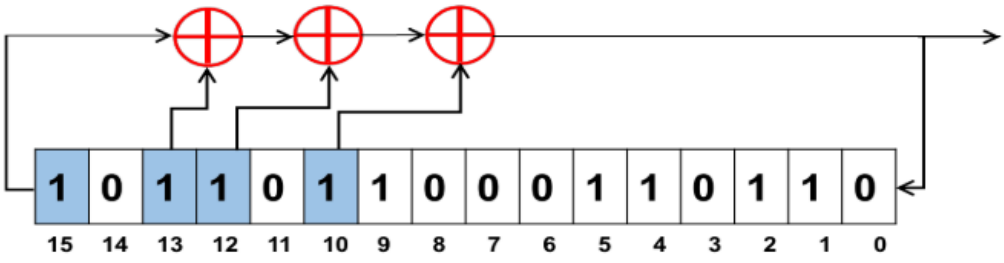



**Figure 2.** A Fibonacci LFSR with taps at positions 10, 12, and 13.

## 1a.2 What Was learned

Within this project I learned a lot about OOP. Creating a class to develop a certain functionality for a larger project(ps1b). The given class was required to be created within its own namespace deemed PhotoMagic. I learned about the importance of namespaces and how they can be beneficial especially when working in larger codebases. I didn't get to demonstrate that specifically in this project as it only included one class. But it's easy to see how it may be useful when working on larger projects. The opportunity to develop an idea without a large amount of guidance on how it should be implemented taught me a lot. Particularly how to trouble shoot and test different ideas.

## 1a.3 Unit Test Output

```
Running 6 test cases...
DEBUG(string base10 newbit) 0110110001101100 27756 0
DEBUG(string base10 newbit) 1101100011011000 55512 0
DEBUG(string base10 newbit) 1011000110110000 45488 0
DEBUG(string base10 newbit) 0110001101100001 25441 1
DEBUG(string base10 newbit) 1100011011000011 50883 1
DEBUG(string base10 newbit) 1000110110000110 36230 0
DEBUG(string base10 newbit) 0001101100001100 6924 0
DEBUG(string base10 newbit) 0011011000011001 13849 1
DEBUG(string base10 newbit) 0110110001101100 27756 0
DEBUG - hold: 0 result: 0
DEBUG(string base10 newbit) 1101100011011000 55512 0
DEBUG - hold: 0 result: 0
DEBUG(string base10 newbit) 1011000110110000 45488 0
DEBUG - hold: 0 result: 0
DEBUG(string base10 newbit) 0110001101100001 25441 1
DEBUG - hold: 1 result: 0
DEBUG - entering if statement in generate
DEBUG(string base10 newbit) 1100011011000011 50883 1
DEBUG - hold: 1 result: 1
DEBUG(string base10 newbit) 1000110110000110 36230 0
DEBUG - hold: 0 result: 3
DEBUG(string base10 newbit) 0001101100001100 6924 0
DEBUG - hold: 0 result: 6
DEBUG(string base10 newbit) 0011011000011001 13849 1
```

# PS1a

DEBUG - hold: 1 result: 12

DEBUG(string base10 newbit) 0110110000110011 27699 1

DEBUG - hold: 1 result: 25

DEBUG - final result: 51

DEBUG(string base10 newbit) 0000000000000000 0 0

DEBUG(string base10 newbit) 1111111111111110 65534 0

DEBUG(string base10 newbit) 0110100000000000 26624 0

DEBUG(string base10 newbit) 1101000000000001 53249 1

DEBUG(string base10 newbit) 1010000000000010 40962 0

DEBUG(string base10 newbit) 0100000000000100 16388 0

*** No errors detected

```
1: // Copyright 2023 <Braden Maillet>
2: #include <iostream>
3:
4: int main() {
5:   std::cout << "main called" << std::endl;
6:   return 1;
7: }
```

```
    1: // Copyright 2024 <Braden Maillet>
    2: #pragma once
    3: #include <stdio.h>
    4: #include <bitset>
    5: #include <string>
    6: #include <iostream>
    7: #include <cmath>
    8: #include <vector>
    9:
   10: #define DEBUG 1
   11: #define LOG_DEBUG if (DEBUG) printf ("DEBUG - "); if (DEBUG)printf
   12:
   13: using std::bitset;
   14: using std::string;
   15: using std::endl;
   16: using std::cout;
   17: using std::ostream;
   18: using std::vector;
   19:
   20: namespace PhotoMagic {
   21: class FibLFSR {
   22:  public:
   23:         explicit FibLFSR(string seed);
   24:         FibLFSR() = delete;
   25:         int step();
   26:         int generate(int k);
   27:         int getsize() { return _size; }
   28:         int getbi() { return _Binary; }
   29:         string& getseed() { return _seed; }
   30:         string getstring() { return this->getbitostring(); }
   31:         int getstringtobi(const FibLFSR& rhs);
   32:         string getbitostring();
   33:
   34:         friend ostream& operator<<(ostream& out, FibLFSR Rhs);
   35:  private:
   36:         int _size;
   37:         unsigned int _Binary;
   38:         string _seed;
   39: };
   40:   FibLFSR::FibLFSR(string seed) {
   41:     this->_seed = seed;
   42:     this->_Binary = getstringtobi(*this);
   43:     this->_size = seed.size();
   44:   }
   45:
   46:   int FibLFSR::step() {  // simulate one step
   47:       int newbit = 0;
   48:       int hold = 0;
   49:       unsigned int largestbit = (static_cast<int>(pow(2, (this->_size - 1)))
);
   50:       vector<unsigned int> taps = { 1024, 4096, 8192, largestbit};
   51:       for (unsigned int i = 0; i < taps.size(); i++) {
   52:           if ((taps.at(i) & this->_Binary) == taps.at(i))
   53:               hold++;
   54:       }
   55:       if ((hold % 2) == 1)
   56:           newbit = 1;
   57:       // end of xor section
   58:       if ((largestbit & this->_Binary) == largestbit) {
   59:           this->_Binary = this->_Binary << 1;
   60:           this->_Binary -= pow(2, this->_size);
```

```
   61:           this->_Binary = this->_Binary | newbit;
   62:       } else {
   63:           this->_Binary = this->_Binary << 1;
   64:           this->_Binary = this->_Binary | newbit;
   65:       }
   66:       cout << "DEBUG(string base10 newbit)" << " " <<
   67:         this->getbitostring() << " " <<
   68:         this->_Binary << " " << newbit << endl;
   69:
   70:       return newbit;
   71:   }
   72:   int FibLFSR::generate(int k) {    // simulate k steps
   73:      int hold;
   74:      int result = 0;
   75:      for (int i = 0; i < k; i++) {
   76:          hold = this->step();
   77:          LOG_DEBUG("hold: %d result: %d\n", hold, result);
   78:          if ((result == 0) && (hold == 1)) {
   79:              LOG_DEBUG("entering if statement in generate\n");
   80:              result = 1;
   81:              continue;
   82:          }
   83:          result = result << 1;
   84:          result = result | hold;
   85:      }
   86:      LOG_DEBUG("final result: %d\n", result);
   87:      return result;
   88:   }
   89:   int FibLFSR::getstringtobi(const FibLFSR& rhs) {
   90:      int power = 1;
   91:      int total = 0;
   92:      for (int i = rhs._seed.size() - 1; i >= 0; i--) {
   93:          if (rhs._seed[i] == '1') {
   94:              total = total + power;
   95:          }
   96:          power *= 2;
   97:      }
   98:      return total;
   99:   }
  100:   string FibLFSR::getbitostring() {   // use mod approach
  101:      string holder;
  102:      int i, j, k;
  103:      int temp = this->_Binary;
  104:      if (temp == 0) {
  105:          for (j = 0; j < this->_size; j++)
  106:              holder.insert(0, "0");
  107:          return holder;
  108:      }
  109:      for (i = 0; temp > 0; i++) {
  110:          if ((temp % 2) == 0)
  111:              holder.insert(0, "0");
  112:          if ((temp % 2) == 1)
  113:              holder.insert(0, "1");
  114:          temp = temp / 2;
  115:      }
  116:      for (k = i; k < this->_size; k++) {
  117:          holder.insert(0, "0");
  118:      }
  119:      return holder;
  120:   }
  121:   ostream& operator<<(ostream& out, FibLFSR rhs) {
```

```
  122:     out << rhs.getbitostring();
  123:     return out;
  124:   }
  125: }  // namespace PhotoMagic
```

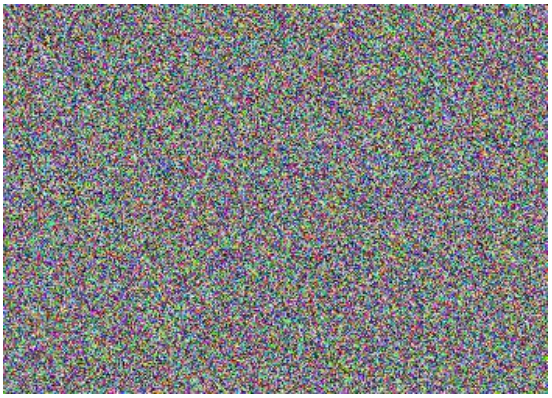
```
 1: // Copyright 2024 <Braden Maillet>
 2: #include "FibLFSR.hpp"
 3: #include <iostream>
 4:
 5: using std::cout;
 6: using std::endl;
 7:
 8: int run(void);
 9:
10: int run(void) {
11:         PhotoMagic::FibLFSR fib("1011011000110110");
12:         int holder = fib.getbi();
13:         cout << holder << endl;
14:
15:         cout << fib.getseed() << " " << fib.getbi() << endl;
16:
17:         cout << fib.getbi() << " " << fib.getbitostring() << endl;
18:
19:         LOG_DEBUG("\n-----1-----\n");
20:         fib.generate(5);
21:         LOG_DEBUG("\n-----2------\n");
22:         fib.generate(5);
23:         LOG_DEBUG("\n-----3------\n");
24:         fib.generate(5);
25:         LOG_DEBUG("\n-----4------\n");
26:         fib.generate(5);
27:         LOG_DEBUG("\n-----5------\n");
28:         fib.generate(5);
29:         LOG_DEBUG("\n-----6------\n");
30:         fib.generate(5);
31:         LOG_DEBUG("\n-----7------\n");
32:         fib.generate(5);
33:         cout << fib << endl;
34:         return 0;
35: }
```

```
    1: CC = g++
    2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
    3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_
test_framework
    4: DEPS = FibLFSR.hpp
    5: OBECTS = FibLFSR.o main.o test.o
    6: PROGRAM = PhotoMagic test PhotoMagic.a
    7:
    8: all:$(PROGRAM)
    9:
   10: %.o:%.cpp $(DEPS)
   11:         $(CC) $(CFLAGS) -c $<
   12: PhotoMagic:FibLFSR.o main.o
   13:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   14: test:test.o
   15:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   16: PhotoMagic.a:FibLFSR.o
   17:         ar rcs PhotoMagic.a FibLFSR.o
   18: clean:
   19:         rm *.o *.a PhotoMagic test
   20: lint:
   21:         cpplint *.cpp *.hpp
```

# 1b PS1b: PhotoMagic

## 1b.1 General Discussion

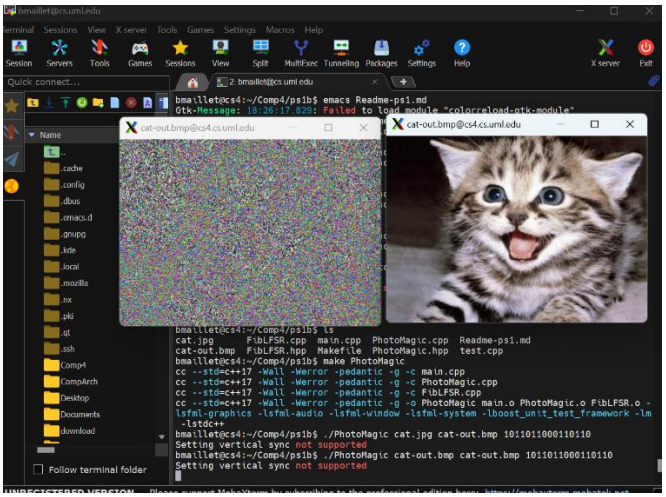
Photomagic is a continuation of the LFSR project also known as ps1a. It utilizes the class that is created to encrypt and decrypt images. Every pixel in most images is encoded in RGB. 3 distinct integers ranging from 0 to 255. We can use an arbitrary seed that is a 16-bit string to encrypt each pixel in the image. The same seed can be used to decrypt the encrypted image. This works because when a number is bitwise ORed twice with a seed it returns to the same number. So, if you don't know the seed it is very hard to figure out what it is because there are 2^16 options for seeds. This makes it very hard to decrypt an image that has been encrypted. This particular project required this to be done through the creation of a function called transform(sf::image&, FibLFSR*). This function simply loops over every pixel in the given image passed in and XORs it with the generate() function using the FibLFSR pointer. Below is an example of what output is expected.

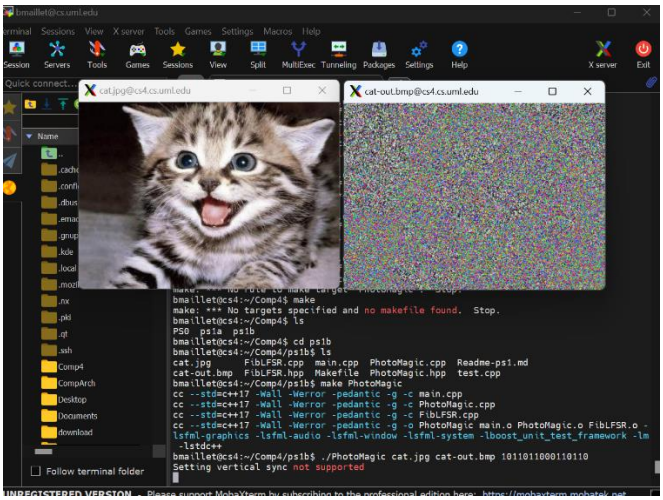


## 1b.2 What Was Learned

This project taught me a lot about encryption and some of the methods that can be used to encrypt data, specifically involving images. These encryption methods can certainly be applied to other topics as well. I also learned a little bit more about the SFML library, like how to insert images as textures and how to create more than one SFML window within the game loop. This project also required Unit tests. Specifically involving the Boost unit test library. I learned more about some of the capabilities of the test library and different methods on testing especially when it came to things outside the console, like images.

# 1b.3 Encryption and Decryption Output



```
 1: // Copyright 2024 Braden Maillet
 2: #include "PhotoMagic.hpp"
 3: #include "FibLFSR.hpp"
 4:
 5: #include <SFML/System.hpp>
 6: #include <SFML/Window.hpp>
 7: #include <SFML/Graphics.hpp>
 8:
 9:
10: int main(int argc, char** argv) {
11:   string in = argv[1];
12:   string out = argv[2];
13:   string seed = argv[3];
14:   sf::Image image1;
15:   if (!image1.loadFromFile(in))  //  save original image for window1
16:     return -1;
17:
18:   sf::Texture texture1;  //  texture for window 1
19:   texture1.loadFromImage(image1);
20:   sf::Sprite sprite1;
21:   sprite1.setTexture(texture1);
22:
23:   PhotoMagic::FibLFSR fib(seed);  //  execution of encryption
24:   PhotoMagic::transform(image1, &fib);
25:
26:   if (!image1.saveToFile(out))  //  save new image to out
27:     return -1;
28:
29:   sf::Image image2;
30:   if (!image2.loadFromFile(out))  //  save out for window2
31:     return -1;
32:
33:   sf::Vector2u size = image1.getSize();
34:   sf::RenderWindow window1(sf::VideoMode(size.x, size.y), in);
35:   sf::RenderWindow window2(sf::VideoMode(size.x, size.y), out);
36:
37:   sf::Texture texture2;   //  texture for window 2 or output
38:   texture2.loadFromImage(image2);
39:   sf::Sprite sprite2;
40:   sprite2.setTexture(texture2);
41:
42:   while (window1.isOpen() && window2.isOpen()) {
43:     sf::Event event;
44:     while (window1.pollEvent(event)) {
45:       if (event.type == sf::Event::Closed)
46:         window1.close();
47:     }
48:     while (window2.pollEvent(event)) {
49:       if (event.type == sf::Event::Closed)
50:         window2.close();
51:     }
52:       window1.clear(sf::Color::White);
53:       window1.draw(sprite1);
54:       window1.display();
55:       window2.clear(sf::Color::White);
56:       window2.draw(sprite2);
57:       window2.display();
58:   }
59:   return 0;
60: }
```

```
 1: // Copyright 2024 <Braden Maillet>
 2: #pragma once
 3: #include <algorithm>
 4: #include "FibLFSR.hpp"
 5: #include <SFML/System.hpp>
 6: #include <SFML/Window.hpp>
 7: #include <SFML/Graphics.hpp>
 8:
 9: namespace PhotoMagic {
10:   void transform(sf::Image& image, FibLFSR* Fib);
11: }  //  namespace PhotoMagic
```

```
 1: // Copyright 2024 <Braden Maillet>
 2: #include "PhotoMagic.hpp"
 3: #include <iostream>
 4: #include <algorithm>
 5: #include <SFML/System.hpp>
 6: #include <SFML/Window.hpp>
 7: #include <SFML/Graphics.hpp>
 8: #include "FibLFSR.hpp"
 9:
10: void PhotoMagic::transform(sf::Image& image, PhotoMagic::FibLFSR* Fib) {
11:   sf::Color p;
12:   sf::Vector2u size = image.getSize();
13:   PhotoMagic::FibLFSR shift = *Fib;
14:     for (unsigned int x = 0; x < size.x; x++) {
15:       for (unsigned int y = 0; y < size.y; y++) {
16:         p = image.getPixel(x, y);
17:         shift.generate(3);
18:         p.r = shift.getbi() ^ p.r;
19:         shift.generate(3);
20:         p.g = shift.getbi() ^ p.g;
21:         shift.generate(3);
22:         p.b = shift.getbi() ^ p.b;
23:         image.setPixel(x, y, p);
24:       }
25:     }
26: }
```

```
 1: // Copyright 2024 <Braden Maillet>
 2: #pragma once
 3: #include <stdio.h>
 4: #include <bitset>
 5: #include <string>
 6: #include <iostream>
 7: #include <cmath>
 8: #include <vector>
 9:
10: #define DEBUG 0
11: #define LOG_DEBUG if (DEBUG) printf ("DEBUG - "); if (DEBUG)printf
12:
13: using std::bitset;
14: using std::string;
15: using std::endl;
16: using std::cout;
17: using std::ostream;
18: using std::vector;
19:
20: namespace PhotoMagic {
21: class FibLFSR {
22:  public:
23:         explicit FibLFSR(string seed);
24:         FibLFSR() = delete;
25:         int step();
26:         int generate(int k);
27:         int getsize() { return _size; }
28:         int getbi() { return _Binary; }
29:         string& getseed() { return _seed; }
30:         string getstring() { return this->getbitostring(); }
31:         int getstringtobi(const FibLFSR& rhs);
32:         string getbitostring();
33:
34:         friend ostream& operator<<(ostream& out, FibLFSR& Rhs);
35:  private:
36:         int _size;
37:         unsigned int _Binary;
38:         string _seed;
39: };
40: }  // namespace PhotoMagic
```

```
 1: // Copyright 2024 <Braden Maillet>
 2: #include "FibLFSR.hpp"
 3: #include <iostream>
 4:
 5: using std::cout;
 6: using std::endl;
 7: using std::string;
 8: using std::vector;
 9:
10: PhotoMagic::FibLFSR::FibLFSR(string seed) {
11:     this->_seed = seed;
12:     this->_Binary = getstringtobi(*this);
13:     this->_size = seed.size();
14: }
15:
16: int PhotoMagic::FibLFSR::step() {  // simulate one step
17:     int newbit = 0;
18:     int hold = 0;
19:     unsigned int largestbit = (static_cast<int>(pow(2, (this->_size - 1))));
20:     vector<unsigned int> taps = { 1024, 4096, 8192, largestbit };
21:     for (unsigned int i = 0; i < taps.size(); i++) {
22:         if ((taps.at(i) & this->_Binary) == taps.at(i))
23:             hold++;
24:     }
25:     if ((hold % 2) == 1)
26:         newbit = 1;
27:     // end of xor section
28:     if ((largestbit & this->_Binary) == largestbit) {
29:         this->_Binary = this->_Binary << 1;
30:         this->_Binary -= pow(2, this->_size);
31:         this->_Binary = this->_Binary | newbit;
32:     } else {
33:         this->_Binary = this->_Binary << 1;
34:         this->_Binary = this->_Binary | newbit;
35:     }
36:     return newbit;
37: }
38: int PhotoMagic::FibLFSR::generate(int k) {    // simulate k steps
39:     int hold;
40:     int result = 0;
41:     for (int i = 0; i < k; i++) {
42:         hold = this->step();
43:         LOG_DEBUG("hold: %d result: %d\n", hold, result);
44:         if ((result == 0) && (hold == 1)) {
45:             LOG_DEBUG("entering if statement in generate\n");
46:             result = 1;
47:             continue;
48:         }
49:         result = result << 1;
50:         result = result | hold;
51:     }
52:     LOG_DEBUG("final result: %d\n", result);
53:     return result;
54: }
55: int PhotoMagic::FibLFSR::getstringtobi(const PhotoMagic::FibLFSR& rhs) {
56:     int power = 1;
57:     int total = 0;
58:     for (int i = rhs._seed.size() - 1; i >= 0; i--) {
59:         if (rhs._seed[i] == '1') {
60:             total = total + power;
61:         }
```

```
62:         power *= 2;
63:     }
64:     return total;
65: }
66: string PhotoMagic::FibLFSR::getbitostring() {   // use mod approach
67:     string holder;
68:     int i, j, k;
69:     int temp = this->_Binary;
70:     if (temp == 0) {
71:         for (j = 0; j < this->_size; j++)
72:             holder.insert(0, "0");
73:         return holder;
74:     }
75:     for (i = 0; temp > 0; i++) {
76:         if ((temp % 2) == 0)
77:             holder.insert(0, "0");
78:         if ((temp % 2) == 1)
79:             holder.insert(0, "1");
80:         temp = temp / 2;
81:     }
82:     for (k = i; k < this->_size; k++) {
83:         holder.insert(0, "0");
84:     }
85:     return holder;
86: }
87: namespace PhotoMagic {
88:   ostream& operator<<(ostream& out, FibLFSR& Rhs) {
89:     out << Rhs.getbitostring();
90:     return out;
91:   }
92: }  //  end of PhotoMagic
93:
```

```
 1: // Copyright 2022
 2: // By Dr. Rykalova
 3: // Editted by Dr. Daly
 4: // test.cpp for PS1a
 5: // updated 1/8/2024
 6:
 7: #include <iostream>
 8: #include <string>
 9:
10: #include "PhotoMagic.hpp"
11: #include "FibLFSR.hpp"
12:
13: using std::endl;
14: using std::string;
15: using PhotoMagic::FibLFSR;
16:
17: #define BOOST_TEST_DYN_LINK
18: #define BOOST_TEST_MODULE Main
19: #include <boost/test/unit_test.hpp>
20:
21: BOOST_AUTO_TEST_CASE(testStepInstr) {
22:          FibLFSR one("1011011000110110");
23:          BOOST_REQUIRE_EQUAL(one.step(), 0);
24:          BOOST_REQUIRE_EQUAL(one.step(), 0);
25:          BOOST_REQUIRE_EQUAL(one.step(), 0);
26:          BOOST_REQUIRE_EQUAL(one.step(), 1);
27:          BOOST_REQUIRE_EQUAL(one.step(), 1);
28:          BOOST_REQUIRE_EQUAL(one.step(), 0);
29:          BOOST_REQUIRE_EQUAL(one.step(), 0);
30:          BOOST_REQUIRE_EQUAL(one.step(), 1);
31: }
32:
33: BOOST_AUTO_TEST_CASE(testGenerateInstr) {
34:   FibLFSR one("1011011000110110");
35:   BOOST_REQUIRE_EQUAL(one.generate(9), 51);
36: }
37: BOOST_AUTO_TEST_CASE(testClassSize) {
38:   FibLFSR one("1011011000110110");
39:   BOOST_REQUIRE_EQUAL(one.getsize(), 16);
40: }
41: BOOST_AUTO_TEST_CASE(teststepzero) {
42:   FibLFSR one("0000000000000000");
43:   BOOST_REQUIRE_EQUAL(one.step(), 0);
44: }
45: BOOST_AUTO_TEST_CASE(teststepone) {
46:   FibLFSR one("1111111111111111");
47:   BOOST_REQUIRE_EQUAL(one.step(), 0);
48: }
49: BOOST_AUTO_TEST_CASE(testseednochange) {
50:   FibLFSR one("1011011000110110");
51:   one.generate(5);
52:   BOOST_REQUIRE_EQUAL(one.getseed(), "1011011000110110");
53: }
54: BOOST_AUTO_TEST_CASE(testNE) {
55:   string stri = "1011011000110110";
56:   FibLFSR one(stri);
57:   BOOST_REQUIRE_NE(one.getbi(), one.step());
58: }
59: BOOST_AUTO_TEST_CASE(testout) {
60:   FibLFSR one("101101100011011");
61:   BOOST_REQUIRE_NO_THROW(cout << one << endl);
```

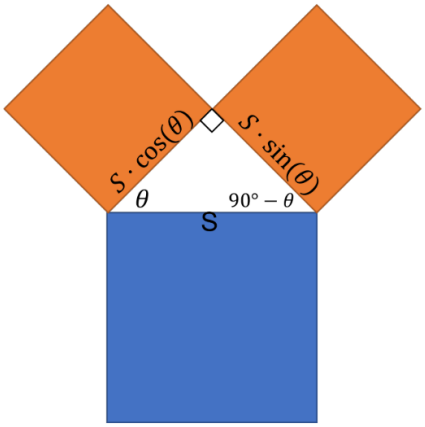
```
    62: }
```

```
    1: ereCC = g++
    2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
    3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_
test_framework
    4: DEPS = PhotoMagic.hpp FibLFSR.hpp
    5: OBECTS = PhotoMagic.o main.o test.o FibLFSR.o
    6: PROGRAM = PhotoMagic test PhotoMagic.a
    7:
    8: all:$(PROGRAM)
    9:
   10: %.o:%.cpp $(DEPS)
   11:         $(CC) $(CFLAGS) -c $<
   12: PhotoMagic:main.o PhotoMagic.o FibLFSR.o
   13:         $(CC) $(CFLAGS) -o $@ $^ $(LIB) -lm -lstdc++
   14: test:test.o PhotoMagic.o FibLFSR.o
   15:         $(CC) $(CFLAGS) -o $@ $^ $(LIB) -lm -lstdc++
   16: PhotoMagic.a:PhotoMagic.o FibLFSR.o
   17:         ar rcs PhotoMagic.a PhotoMagic.o FibLFSR.o
   18: clean:
   19:         rm *.o *.a PhotoMagic test PhotoMagic.a
   20: lint:
   21:         cpplint *.cpp *.hpp
```

# 2 PS2: Pythagoras Tree

## 2.1 General Discussion

Pythagoras tree is a consisting of squares that in which they all touch two other squares to form a right triangle(see output below). The math behind the Pythagoras tree is a class recursive problem. Recursion is the most elegant way of implementing a drawing of a Pythagoras tree. The project required the implementation of a recursive function pTree(), a main() function which calls pTree() and if needed a helper function for pTree(). The math behind the tree can be seen below.




Every left child square should be S * cos(theta) and every right child square should be S * sin(theta). The triangle that is created by the two child squares is inherently isosceles so theta and 90 – theta will always be 45 degrees.

## 2.1 Implementation

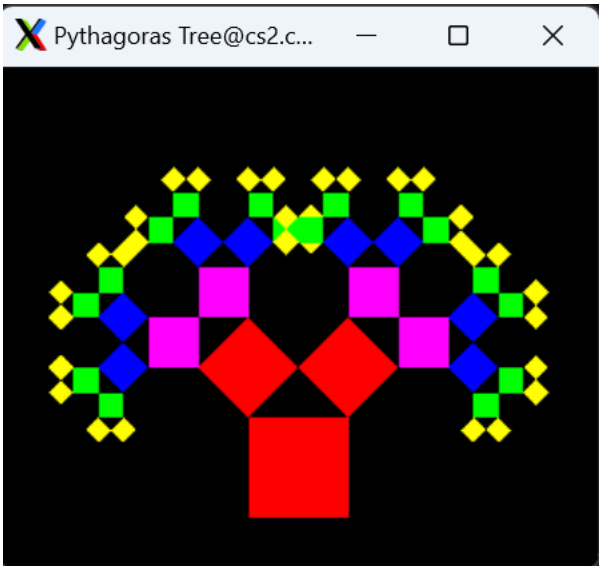
I ran into a considerable number of roadblocks during the implementation with this project. Not because the math was inherently difficult, but the SFML library did not work as I expected. To implement this project it was required to create a class that publicly inherited the SFML base class sf::Drawable. I decided to do this indirectly by inheriting a child of that base class ie sf::rectangleShape. For obvious reasons I was only working with rectangles, so it was beneficial to only have specifically what I needed. I ran into problems when trying to move and rotate the rectangle's relative to their parent rectangles. I ended up discovering a capability of the library within the Transform class. The transform class records all transforms

that have been done on an object. It also records the “local position” of an object and can convert that to global position. The local position in SFML is the top left of the rectangle, being (0, 0). But that same local position has a corresponding global position in the window (x, y). Using the function transformPoint(), which takes local position and outputs global position, I could easily change the position of the rectangle without any complex math. Using this principle, I created the pTree function so that it drew an original rectangle and then drew a left rectangle after changing the origin, size and rotation. Next, I call the pTree recursive helper function that did basically the same thing. I did the same for the right side as well. Within the main function a window is created of size 6 * L by 4 * L. L being the pixel size of the original rectangle. This conveniently creates a window that snuggly fits the whole tree. The pTree class also has a value constructor that sets a side length and number of iterations.

## 2.3 What was learned

I learned a lot more about the drawable library within SFML. Specifically, the RectangleShape child class. I also learned about the convenience of the transform class and how it can work closely with the drawable class to make implementations far simpler. I knew a lot of the math involved in the operation of the assignment already. Although it was not incredibly complex.

## 2.4 Output



```
 1: //  Copyright 2024 Braden Maillet
 2: #include <cmath>
 3: #include <iostream>
 4: #include <string>
 5: #include "PTree.hpp"
 6: #include <SFML/Graphics.hpp>
 7:
 8:
 9: int main(int argc, char** argv) {
10:     //  length of side for base square
11:     double L = static_cast<double>(std::stoi(argv[1]));
12:     int N = std::stoi(argv[2]);  //  depth of the recursion
13:     sf::RenderWindow window(sf::VideoMode((L * 6), L * 4),
14:         "Pythagoras Tree");
15:     myDraw D(L, N);
16:     D.rdraw(D, window);
17:     while (window.isOpen()) {
18:         sf::Event event;
19:         while (window.pollEvent(event)) {
20:             if (event.type == sf::Event::Closed)
21:                 window.close();
22:         }
23:         window.display();
24:     }
25:     return 0;
26: }
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #pragma once
 3: #include <SFML/Graphics.hpp>
 4:
 5: class myDraw : public sf::RectangleShape {
 6:  public:
 7:       myDraw() : sf::RectangleShape() {}
 8:       myDraw(double slength, int reps) : sf::RectangleShape(
 9:         sf::Vector2f(slength, slength)),
10:         _slength(slength), _reps(reps) {}
11:       double getlength() const { return this->_slength;}
12:       int getreps() const { return this->_reps;}
13:       void setlength(double length) {this->_slength = length;}
14:       void setreps(int reps) {this->_reps = reps;}
15:       void rdraw(myDraw& D, sf::RenderWindow& win);
16:       void rdrawhelp(myDraw&, sf::RenderWindow& win);
17:  private:
18:     double _slength;
19:     int _reps;
20: };
```

```
    1: //  Copyright 2024 Braden Maillet
    2: #include "PTree.hpp"
    3: #include <cmath>
    4: #include <iostream>
    5: #include <SFML/Graphics.hpp>
    6:
    7: //  (0,0) is always top left on square even with origin change
    8: void myDraw::rdraw(myDraw& D, sf::RenderWindow& win) {
    9:     sf::Vector2f size(D.getlength(), D.getlength());
   10:     sf::Vector2f temp;
   11:     const float Csize = sin(M_PI/4);
   12:     if (D.getreps() == 0)    //   null case
   13:         return;
   14:     D.setSize(size);        //  base square or first sqaure
   15:     D.setPosition(((6 * D.getlength())/2) - (D.getlength()/2),
   16:       ((4 * D.getlength())/2) + D.getlength());
   17:     D.setFillColor(sf::Color::Red);
   18:     sf::Transform trans = D.getTransform();
   19:     myDraw DL = D;
   20:     myDraw DR = D;
   21:     win.draw(D);
   22:
   23:     DL.setSize(size * Csize);
   24:     DL.setOrigin(0, DL.getSize().y);    //  set EACH time
   25:     temp = trans.transformPoint({0, 0});
   26:     DL.setPosition(temp);       //  moves to location relative to object (0,
 0)
   27:     DL.setRotation(DL.getRotation() - 45);
   28:     rdrawhelp(DL, win);
   29:     DR.setSize(size * Csize);
   30:     DR.setOrigin(DR.getSize().x, DR.getSize().y);
   31:     temp = trans.transformPoint({DR.getSize().x/Csize, 0});
   32:     DR.setPosition(temp);
   33:     DR.setRotation(DR.getRotation() + 45);
   34:     rdrawhelp(DR, win);
   35: }
   36: void myDraw::rdrawhelp(myDraw& D, sf::RenderWindow& win) {
   37:     sf::Transform transform = D.getTransform();
   38:     sf::Vector2f temp;
   39:     sf::Vector2f vectsize = D.getSize();
   40:     sf::Color Orange(255, 69, 0);
   41:     myDraw newDL;
   42:     myDraw newDR;
   43:     const float Csize = sin(M_PI/4);  //  "macro" to help with resize
   44:     if (D.getreps() == 0)    //  null case
   45:         return;
   46:
   47:     win.draw(D);
   48:     newDL = D;
   49:     newDR = D;
   50:     newDL.setSize(vectsize * Csize);  //  Left: HAVE to set size before orig
in
   51:     newDL.setOrigin(0, newDL.getSize().y);  //  set EACH time
   52:     temp = transform.transformPoint({0, 0});
   53:     newDL.setPosition(temp);    //  moves to location relative to object (0,
 0)
   54:     newDL.setRotation(newDL.getRotation() - 45);
   55:     newDL.setreps(D.getreps() - 1);
   56:     switch ((D.getreps() % 6)) {  //  Fun colors
   57:         case 0:
   58:             newDR.setFillColor(sf::Color::Red);
```

```
59:             newDL.setFillColor(sf::Color::Red);
60:             break;
61:         case 1:
62:             newDR.setFillColor(Orange);
63:             newDL.setFillColor(Orange);
64:             break;
65:         case 2:
66:             newDR.setFillColor(sf::Color::Yellow);
67:             newDL.setFillColor(sf::Color::Yellow);
68:             break;
69:         case 3:
70:             newDR.setFillColor(sf::Color::Green);
71:             newDL.setFillColor(sf::Color::Green);
72:             break;
73:         case 4:
74:             newDR.setFillColor(sf::Color::Blue);
75:             newDL.setFillColor(sf::Color::Blue);
76:             break;
77:         case 5:
78:             newDR.setFillColor(sf::Color::Magenta);
79:             newDL.setFillColor(sf::Color::Magenta);
80:             break;
81:         default:
82:             break;
83:     }
84:     rdrawhelp(newDL, win);  //  recursive call
85:     newDR.setSize(vectsize * Csize);  //  Right
86:     //  set origin to bottom right for seemless rotation
87:     newDR.setOrigin(newDR.getSize().x, newDR.getSize().y);
88:     temp = transform.transformPoint({newDR.getSize().x/Csize, 0});
89:     newDR.setPosition(temp);
90:     newDR.setRotation(newDR.getRotation() + 45);  //  r
91:     newDR.setreps(newDR.getreps() - 1);
92:     rdrawhelp(newDR, win);  //  recurseive call
93: }
94:
95:
```

```
 1: CC = g++
 2: CFLAGS = --std=c++17 -Wall -pedantic -g
 3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lstdc++
 4: DEPS = PTree.hpp
 5: OBECTS = main.o PTree.o
 6: PROGRAM = PTree
 7:
 8: all:$(PROGRAM)
 9:
10: %.o:%.cpp $(DEPS)
11:         $(CC) $(CFLAGS) -c $<
12: PTree:main.o PTree.o
13:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
14: clean:
15:         rm *.o *.a $(PROGRAM)
16: lint:
17:         cpplint *.cpp *.hpp
18:
```

# 3 PS3: Sokoban

## 3.1: General Discussion

Sokoban is a Japanese term for a warehouse keeper. This project is a game based on maintaining a warehouse. The project was split up into two different parts. The first part being the creation of the user interface or environment. The second part implemented movement and game features. The environment takes in a .txt file that contains the size of the map as well as a series of characters that indicate different things as seen below.


‘@’ is the initial position of the character Sokoban.

‘.’ is empty space.

‘#’ is an impassible and immovable wall.

‘A’ is a box that can be pushed by Sokoban but not pulled.

‘a’ is a storage space for boxes

‘1’ is a box already in a storage space

## 3.2: Implementation of the Environment

I created a class named Sokoban that publicly derived the SFML drawable library. The class held 2 vector<vector<char>> data members. One contained the current

arena and the other saved the original arena. These data members, which are essentially matrices, held the state of the arena which could then be manipulated when game features were added to the program. The class also contained some textures for drawing the matrix that would be set in the default constructor as well as int width and height. I inherited sf::drawable because of its pure virtual function draw, which can be overridden protected to draw the entire matrix in this case. The plan for the game being to within the main loop constantly check for input and redraw the current matrix. The original arena data member would come in handy later because of the restart functionality that would be required for ps3b. This class was also included in a name space which contained an Enum called Direction which contained constant values for all four cardinal directions.

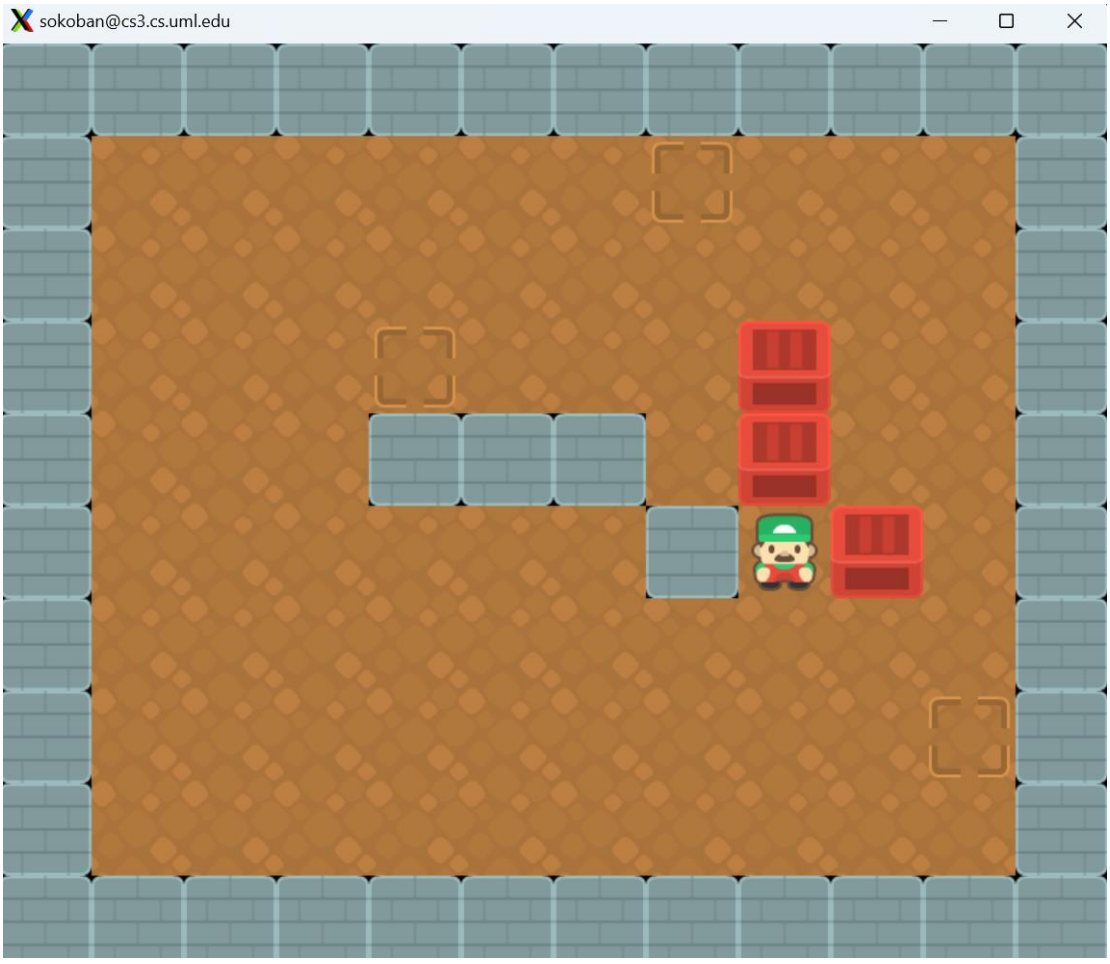
## 3.2: Adding Game Features

Ps3b implements the inner workings of the game. Requiring a few functions like moveplayer(), iswon(), reset(), and some getter member function to the Sokoban class. Within the game loop in the main file character strokes W, A, S, D, and R are constantly being checked. If WASD is detected than the moveplayer() function will be called with the correct Enum Direction. If the R key is detected then the reset() function will be called. Within the same loop the game is constantly being checked for winning conditions using the iswon() function. If winning conditions are met, then a message will be displayed and the game can be reset. The complexity behind the implementation here comes within the moveplayer() function. It must be detected whether or not movement is possible. There are many cases for movement of the player. Not only involving the possibility that player can’t move but also that boxes can't move because they would collide with something. I also had to save data that was temporarily deleted from the matrix. For example, if the player moved over a storage space the matrix would temporarily lose record of that storage space. This Original arena data member came in handy here. Overall checking cases in the moveplayer() function consisted of a switch statement for direction and if statements or movement cases within the switch statement.

## 3.3: What was learned

I learned a lot about making a game and how much it takes to create something that is bug free. Through unit testing I am sure there are few if any bugs left within the program. But if it was a larger project, it would be incredibly difficult to think of all the different ways to break the program. For a larger project it would be beneficial to have a team not only to split up work but also think through the problem and have different minds working together to cover all fronts. I already knew most of what was necessary within SFML to create the environment. I had worked with Sprites and textures as well as the idea of a game loop within the main function. The test functions were fairly simple as well. It involved mostly testing of IO and the functions that were required of the project. Nothing incredibly complex.

## 3.3: Output



```
 1: //  Copyright 2024 Braden Maillet
 2: #include <iostream>
 3: #include <fstream>
 4: #include "Sokoban.hpp"
 5: #include <SFML/Graphics.hpp>
 6:
 7: int main(int argc, char** argv) {
 8:     sf::Vector2i temp;
 9:     std::ifstream myfile;
10:     sf::Text text;
11:     myfile.open(argv[1]);
12:     if (!myfile.is_open()) {
13:         std::cout << "File failed to open" << std::endl;
14:         return -1;
15:     }
16:     SB::Sokoban a;
17:     myfile >> a;
18:     myfile.close();
19:     //  std::cout << a;  //  DEBUG
20:
21:     //  font for "you win" text
22:     //  Font Designed by Isa Ozler
23:     //  Open for use through Google Fonts
24:     sf::Font font;
25:     if (!font.loadFromFile("KodeMono-Regular.ttf")) {
26:         std::cout << "Text failed to load" << std::endl;
27:         return -1;
28:     }
29:     text.setFont(font);
30:     text.setString("You Win! 'r' to continue.");
31:     text.setCharacterSize(24);
32:     text.setFillColor(sf::Color::Blue);
33:     //  width * 64 / 2 ; height * 64 / 2
34:     text.setPosition(a.width() * 32, a.height() * 32);
35:
36:     //  make window and set size
37:     sf::RenderWindow window(sf::VideoMode(64 * a.width()
38:                             , 64 * a.height()), "sokoban");
39:     window.draw(a);
40:     //  temp = a.playerLoc();  //  DEBug
41:     //  std::cout << temp.x << " " << temp.y << std::endl;  //  DEBUG
42:
43:     while (window.isOpen()) {
44:         sf::Event event;
45:         while (window.pollEvent(event)) {
46:             if (event.type == sf::Event::Closed)
47:                 window.close();
48:         }
49:               //  Redraw everytime key is pressed
50:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::W)) {
51:                 a.movePlayer(SB::Direction::Up);
52:             }
53:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::A)) {
54:                 a.movePlayer(SB::Direction::Left);
55:             }
56:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::S)) {
57:                 a.movePlayer(SB::Direction::Down);
58:             }
59:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::D)) {
60:                 a.movePlayer(SB::Direction::Right);
61:             }
```

```
62:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::R)) {
63:                 a.reset();
64:             }
65:             //  test if game is won
66:             if (a.isWon()) {
67:                 window.clear();
68:                 window.draw(a);
69:                 window.draw(text);
70:                 window.display();
71:                 //  Reset game
72:                 while (!sf::Keyboard::isKeyPressed(sf::Keyboard::Key::R)) {
73:                     //  loop holds position in game
74:                     //  at this point the player will
75:                     //  1. click R and the game will reset
76:                     //  or close the window and quit
77:                 }
78:                 a.reset();
79:             }
80:             window.clear();
81:             window.draw(a);
82:             window.display();
83:     }
84:
85:     return 0;
86: }
```

```
    1: //  Copyright 2024 Braden Maillet
    2: #pragma once
    3: #include <vector>
    4: #include <iostream>
    5: #include <SFML/Graphics/Drawable.hpp>
    6: #include <SFML/System/Vector2.hpp>
    7: #include <SFML/Graphics.hpp>
    8:
    9: namespace SB {
   10: enum Direction {Up, Down, Left, Right};
   11: class Sokoban : public sf::Drawable{
   12:  public:
   13:     Sokoban();
   14:     int width() const { return _width; }
   15:     void setwidth(int width) { _width = width; }
   16:     int height() const { return _height; }
   17:     void setheight(int height) { _height = height; }
   18:     char getvectOG(int x, int y) const { return OGarena[x][y]; }
   19:     void setvectOG(const std::vector<char>& v) { OGarena.push_back(v); }
   20:     char getvectcurr(int x, int y) const;
   21:     void setvectcurr(const std::vector<char>& v) { currarena.push_back(v); }
   22:     void setvectcurr(char c, int h, int l) { currarena[h][l] = c; }
   23:     sf::Vector2i playerLoc() const;
   24:     sf::Vector2i DirectionbasedLoc(enum Direction D,
   25:          sf::Vector2i Loc, int count);
   26:     bool checkmove(enum Direction D, sf::Vector2i Loc);
   27:     char newtile_Pos0(sf::Vector2i Loc);
   28:     void movePlayer(enum Direction D);
   29:     bool isWon();
   30:     void reset();
   31:
   32:     friend std::ostream& operator<<(std::ostream& out, SB::Sokoban& a);
   33:     friend std::ifstream& operator>>(std::ifstream& in, SB::Sokoban& a);
   34:
   35:  protected:
   36:     void draw(sf::RenderTarget& target, sf::RenderStates states) const overr
ide;
   37:
   38:  private:
   39:     int _width;
   40:     int _height;
   41:     std::vector<std::vector<char>> OGarena;
   42:     std::vector<std::vector<char>> currarena;
   43:     sf::Texture playerF;
   44:     sf::Texture playerB;
   45:     sf::Texture playerR;
   46:     sf::Texture playerL;
   47:     sf::Texture Wall;
   48:     sf::Texture Box;
   49:     sf::Texture Empty;
   50:     sf::Texture Storage;
   51: };
   52: }  //  namespace SB
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <vector>
 3: #include <iostream>
 4: #include <fstream>
 5: #include <algorithm>
 6: #include "Sokoban.hpp"
 7: #include <SFML/Graphics/Drawable.hpp>
 8: #include <SFML/System/Vector2.hpp>
 9: #include <SFML/Graphics.hpp>
10:
11:     SB::Sokoban::Sokoban() : _width(0), _height(0) {
12:         if (playerF.loadFromFile("sokoban/player_05.png") == false)
13:             std::cout << "failed to load playerF" << std::endl;
14:         if (playerB.loadFromFile("sokoban/player_08.png") == false)
15:             std::cout << "failed to load playerB" << std::endl;
16:         if (playerR.loadFromFile("sokoban/player_17.png") == false)
17:             std::cout << "failed to load playerR" << std::endl;
18:         if (playerL.loadFromFile("sokoban/player_20.png") == false)
19:             std::cout << "failed to load playerL" << std::endl;
20:         if (Wall.loadFromFile("sokoban/block_06.png") == false)
21:             std::cout << "failed to load Wall" << std::endl;
22:         if (Box.loadFromFile("sokoban/crate_03.png") == false)
23:             std::cout << "failed to load Box" << std::endl;
24:         if (Empty.loadFromFile("sokoban/ground_01.png") == false)
25:             std::cout << "failed to load Empty" << std::endl;
26:         if (Storage.loadFromFile("sokoban/ground_04.png") == false)
27:             std::cout << "failed to load Storage" << std::endl;
28:     }
29:     sf::Vector2i SB::Sokoban::playerLoc() const {
30:         sf::Vector2i vect;
31:         for (int x = 0; x < this->height(); x++) {
32:             for (int y = 0; y < this->width(); y++) {
33:                 if (this->getvectcurr(x, y) == '@') {
34:                     vect.x = y;
35:                     vect.y = x;
36:                     return vect;
37:                 }
38:             }
39:         }
40:         return vect;
41:     }
42:     sf::Vector2i SB::Sokoban::DirectionbasedLoc(enum SB::Direction D,
43:                         sf::Vector2i Loc, int count) {
44:         sf::Vector2i temp = Loc;
45:         temp.x++;  //  *****
46:         temp.y++;
47:         switch (D) {
48:             case SB::Up:
49:                 temp.y = temp.y - count;
50:                 break;
51:             case SB::Left:
52:                 temp.x = temp.x - count;
53:                 break;
54:             case SB::Down:
55:                 temp.y = temp.y + count;
56:                 break;
57:             case SB::Right:
58:                 temp.x = temp.x + count;
59:                 break;
60:             default:
61:                 break;
```

```
    62:         }
    63:         return temp;
    64:     }
    65:     char SB::Sokoban::newtile_Pos0(sf::Vector2i Loc) {
    66:         char temp = getvectOG(Loc.y, Loc.x);
    67:         if (temp == 'A')  //  Box
    68:             return '.';
    69:         if (temp == '1')  //  box in storage
    70:             return 'a';
    71:         if (temp == '@')  //  player OG location
    72:             return '.';
    73:
    74:         return temp;
    75:     }
    76:     char SB::Sokoban::getvectcurr(int x, int y) const {
    77:         if (x < 0 || y < 0)  //  check lower bounds
    78:             return 'z';
    79:         if (x >= _height || y >= _width)  //  check upper bounds
    80:             return 'z';
    81:         return this->currarena[x][y];
    82:     }
    83:     //  cases for check move that return false
    84:     //  1. wall sqaure in direction D
    85:     //  2. box sqaure followed by wall
    86:     //  3. box sqaure followed by box square
    87:     //  4. player reaches bounds of game
    88:     //  5. box reaches bounds of game
    89:     //  I'm finding more as i go along
    90:     bool SB::Sokoban::checkmove(enum SB::Direction D, sf::Vector2i Loc) {
    91:         sf::Vector2i hold = Loc;
    92:         //  std::cout << "checkmove debug" <<
    93:         //  hold.x << " " << hold.y << std::endl;
    94:         switch (D) {
    95:             case SB::Direction::Up:
    96:                 if (getvectcurr(hold.y - 1, hold.x) == '#')
    97:                     return false;  // wall works
    98:
    99:                 if (getvectcurr(hold.y - 1, hold.x) == 'A' &&
   100:                 getvectcurr(hold.y - 2, hold.x) == '#')
   101:                     return false;  //  box wall
   102:
   103:                 if (getvectcurr(hold.y - 1, hold.x) == 'A' &&
   104:                 getvectcurr(hold.y - 2, hold.x) == 'A')
   105:                     return false;  //  box box
   106:
   107:                 if (getvectcurr(hold.y - 1, hold.x) == 'A' &&
   108:                 getvectcurr(hold.y - 2, hold.x) == '1')
   109:                     return false;  //  box storage box
   110:
   111:                 if (hold.y == 0)
   112:                     return false;  //  player reaches boundary
   113:
   114:                 if (getvectcurr(hold.y - 1, hold.x) == 'A' &&
   115:                 hold.y - 2 < 0)  //  box reaches boundary
   116:                     return false;
   117:
   118:                 if (getvectcurr(hold.y - 1, hold.x) == '1' &&
   119:                 hold.y - 2 < 0)  //  box in storage reaches boundary
   120:                     return false;
   121:
   122:                 if (getvectcurr(hold.y - 1, hold.x) == '1' &&
```

```
123:                getvectcurr(hold.y - 2, hold.x) == '#')
124:                    return false;  //  box wall second case
125:
126:                if (getvectcurr(hold.y - 1, hold.x) == '1' &&
127:                getvectcurr(hold.y - 2, hold.x) == 'A')
128:                    return false;  //  box in storage and box case
129:                break;
130:
131:            case SB::Direction::Left:
132:                if (getvectcurr(hold.y, hold.x - 1) == '#')
133:                    return false;
134:
135:                if (getvectcurr(hold.y, hold.x - 1) == 'A' &&
136:                getvectcurr(hold.y, hold.x - 2) == '#')
137:                    return false;
138:
139:                if (getvectcurr(hold.y, hold.x - 1) == 'A' &&
140:                getvectcurr(hold.y, hold.x - 2) == 'A')
141:                    return false;
142:
143:                if (getvectcurr(hold.y, hold.x - 1) == 'A' &&
144:                getvectcurr(hold.y, hold.x - 2) == '1')
145:                    return false;
146:
147:                if (hold.x - 1 < 0)
148:                    return false;
149:
150:                if (getvectcurr(hold.y, hold.x - 1) == 'A' &&
151:                hold.x - 2 < 0)
152:                    return false;
153:
154:                if (getvectcurr(hold.y, hold.x - 1) == '1' &&
155:                hold.x - 2 < 0)
156:                    return false;
157:
158:                if (getvectcurr(hold.y, hold.x - 1) == '1' &&
159:                getvectcurr(hold.y, hold.x - 2) == '#')
160:                    return false;
161:
162:                if (getvectcurr(hold.y, hold.x - 1) == '1' &&
163:                getvectcurr(hold.y, hold.x - 2) == 'A')
164:                    return false;
165:                break;
166:
167:            case SB::Direction::Down:
168:                if (getvectcurr(hold.y + 1, hold.x) == '#')
169:                    return false;
170:
171:                if (getvectcurr(hold.y + 1, hold.x) == 'A' &&
172:                getvectcurr(hold.y + 2, hold.x) == '#')
173:                    return false;
174:
175:                if (getvectcurr(hold.y + 1, hold.x) == 'A' &&
176:                getvectcurr(hold.y + 2, hold.x) == 'A')
177:                    return false;
178:
179:                if (getvectcurr(hold.y + 1, hold.x) == 'A' &&
180:                getvectcurr(hold.y + 2, hold.x) == '1')
181:                    return false;
182:
183:                if (hold.y + 1 == this->height())
```

```
184:                     return false;
185:
186:                 if (getvectcurr(hold.y + 1, hold.x) == 'A' &&
187:                 hold.y + 2 == this->height())
188:                     return false;
189:
190:                 if (getvectcurr(hold.y + 1, hold.x) == '1' &&
191:                 hold.y + 2 > this->height())
192:                     return false;
193:
194:                 if (getvectcurr(hold.y + 1, hold.x) == '1' &&
195:                 getvectcurr(hold.y + 2, hold.x) == '#')
196:                     return false;
197:
198:                 if (getvectcurr(hold.y, hold.x - 1) == '1' &&
199:                 getvectcurr(hold.y, hold.x - 2) == 'A')
200:                     return false;
201:                 break;
202:
203:             case SB::Direction::Right:
204:                 if (getvectcurr(hold.y, hold.x + 1) == '#')
205:                     return false;
206:
207:                 if (getvectcurr(hold.y, hold.x + 1) == 'A' &&
208:                 getvectcurr(hold.y, hold.x + 2) == '#')
209:                     return false;
210:
211:                 if (getvectcurr(hold.y, hold.x + 1) == 'A' &&
212:                 getvectcurr(hold.y, hold.x + 2) == 'A')
213:                     return false;
214:
215:                 if (getvectcurr(hold.y, hold.x + 1) == 'A' &&
216:                 getvectcurr(hold.y, hold.x + 2) == '1')
217:                     return false;
218:
219:                 if (hold.x + 1 == this->width())  //  .x  &&  >  *****
220:                     return false;
221:
222:                 if (getvectcurr(hold.y, hold.x + 1) == 'A' &&
223:                 hold.x + 2 == this->width())
224:                     return false;
225:
226:                 if (getvectcurr(hold.y, hold.x + 1) == '1' &&
227:                 hold.x + 2 > this->width())
228:                     return false;
229:
230:                 if (getvectcurr(hold.y, hold.x + 1) == '1' &&
231:                 getvectcurr(hold.y, hold.x + 2) == '#')
232:                     return false;
233:
234:                 if (getvectcurr(hold.y, hold.x - 1) == '1' &&
235:                 getvectcurr(hold.y, hold.x - 2) == 'A')
236:                     return false;
237:                 break;
238:
239:             default:
240:                 break;
241:         }
242:         //  std::cout << "returned true" << std::endl;
243:         return true;
244:     }
```

```
245:     void SB::Sokoban::movePlayer(enum SB::Direction D) {
246:         std::vector<char> temp1;
247:         sf::Vector2i Loc = this->playerLoc();
248:         //  change of tile cases
249:         //  1. current location change will always be
250:         //  '.' or 'a'
251:         //  2. change of one past is always '@'
252:         //  3. change of 2 past: only be effected if pushing box
253:         //  'A' if currently '.'
254:         //  '1' if currently 'a'
255:
256:         //  can seperate these into 2 broader statements
257:         //  1. is a box being pushed
258:         //  2. is a box not being pushed
259:         switch (D) {
260:             case SB::Up:
261:                 if (checkmove(SB::Up, Loc)) {
262:                       //  is box being pushed
263:                     if (getvectcurr(Loc.y - 1, Loc.x) == 'A' &&
264:                     getvectcurr(Loc.y - 2, Loc.x) != 'a') {
265:                         setvectcurr('A', Loc.y - 2, Loc.x);  //  pos + 2
266:                     }
267:                       //  is box being pushed into storage
268:                     if (getvectcurr(Loc.y - 1, Loc.x) == 'A' &&
269:                     getvectcurr(Loc.y - 2, Loc.x) == 'a') {
270:                         setvectcurr('1', Loc.y - 2, Loc.x);  //  pos + 2
271:                     }
272:                       // is box in storage being pushed
273:                     if (getvectcurr(Loc.y - 1, Loc.x) == '1' &&
274:                     getvectcurr(Loc.y - 2, Loc.x) == '.') {
275:                         setvectcurr('A', Loc.y - 2, Loc.x);
276:                     }
277:                     setvectcurr('@', Loc.y - 1, Loc.x);  //  pos + 1
278:                     setvectcurr(newtile_Pos0(Loc), Loc.y, Loc.x);  //  pos
279:                 }
280:                 break;
281:             case SB::Left:
282:                 if (checkmove(SB::Left, Loc)) {
283:                     if (getvectcurr(Loc.y, Loc.x - 1) == 'A' &&
284:                     getvectcurr(Loc.y, Loc.x - 2) != 'a') {
285:                         setvectcurr('A', Loc.y, Loc.x - 2);
286:                     }
287:                       //  is box being pushed into storage
288:                     if (getvectcurr(Loc.y, Loc.x - 1) == 'A' &&
289:                     getvectcurr(Loc.y, Loc.x - 2) == 'a') {
290:                         setvectcurr('1', Loc.y, Loc.x - 2);  //  pos + 2
291:                     }
292:                     if (getvectcurr(Loc.y, Loc.x - 1) == '1' &&
293:                     getvectcurr(Loc.y, Loc.x - 2) == '.') {
294:                         setvectcurr('A', Loc.y, Loc.x - 2);
295:                     }
296:                     setvectcurr('@', Loc.y, Loc.x - 1);
297:                     setvectcurr(newtile_Pos0(Loc), Loc.y, Loc.x);  /////
298:                 }
299:                 break;
300:             case SB::Down:
301:                 if (checkmove(SB::Down, Loc)) {
302:                     if (getvectcurr(Loc.y + 1, Loc.x) == 'A' &&
303:                     getvectcurr(Loc.y + 2, Loc.x) != 'a') {
304:                         setvectcurr('A', Loc.y + 2, Loc.x);
305:                     }
```

```
306:                     if (getvectcurr(Loc.y + 1, Loc.x) == 'A' &&
307:                     getvectcurr(Loc.y + 2, Loc.x) == 'a') {
308:                         setvectcurr('1', Loc.y + 2, Loc.x);
309:                     }
310:                     if (getvectcurr(Loc.y + 1, Loc.x) == '1' &&
311:                     getvectcurr(Loc.y + 2, Loc.x) == '.') {
312:                         setvectcurr('A', Loc.y + 2, Loc.x);
313:                     }
314:                     setvectcurr('@', Loc.y + 1, Loc.x);
315:                     setvectcurr(newtile_Pos0(Loc), Loc.y, Loc.x);
316:                 }
317:                 break;
318:             case SB::Right:
319:                 if (checkmove(SB::Right, Loc)) {
320:                     if (getvectcurr(Loc.y, Loc.x + 1) == 'A' &&
321:                     getvectcurr(Loc.y, Loc.x + 2) != 'a') {
322:                         setvectcurr('A', Loc.y, Loc.x + 2);
323:                     }
324:                     if (getvectcurr(Loc.y, Loc.x + 1) == 'A' &&
325:                     getvectcurr(Loc.y, Loc.x + 2) == 'a') {
326:                         setvectcurr('1', Loc.y, Loc.x + 2);
327:                     }
328:                     if (getvectcurr(Loc.y, Loc.x + 1) == '1' &&
329:                     getvectcurr(Loc.y, Loc.x + 2) == '.') {
330:                         setvectcurr('A', Loc.y, Loc.x + 2);
331:                     }
332:                     setvectcurr('@', Loc.y, Loc.x + 1);
333:                     setvectcurr(newtile_Pos0(Loc), Loc.y, Loc.x);
334:                 }
335:                 break;
336:             default:
337:                 break;
338:         }
339:         //  std::cout << *this << std::endl <<
340:         //  "DEBUG MOVEPLAYER" << std::endl;  //  DEBUG
341:     }
342:     bool SB::Sokoban::isWon() {
343:         //  NYI
344:         //  will be for part b
345:         //  use algorithm c++ library and lambda
346:         //  expression as required by assignment
347:         int countStor = 0;
348:         int countBox = 0;
349:         int countOne = 0;
350:         for (const auto& row : this->currarena) {  //  counts elements of ga
me
351:             countStor += std::count_if(row.begin(), row.end(), [](char a){
352:             return (a == '1' || a == 'a') ? true : false;
353:             });
354:             countBox += std::count_if(row.begin(), row.end(), [](char a){
355:             return (a == '1' || a == 'A') ? true : false;
356:             });
357:             countOne += std::count_if(row.begin(), row.end(), [](char a){
358:             return a == '1' ? true : false;
359:             });
360:         }
361:         //  checks if storage is full or if boxes are all stored
362:         if (countOne == countStor || countOne == countBox) {
363:             return true;
364:         }
365:         //  std::cout << "DEBUG: " << countStor << " " <<
```

```
366:             //  countBox << " " << countOne << std::endl;  //  DEBUG
367:         return false;
368:     }
369: namespace SB {
370:     std::ostream& operator<<(std::ostream& out, SB::Sokoban& a) {
371:         out << a.height() << " " << a.width() << std::endl;
372:         for (int x = 0; x < a.height(); x++) {
373:             for (int y = 0; y < a.width(); y++) {
374:                 out << a.getvectcurr(x, y);
375:             }
376:             out << std::endl;
377:         }
378:         return out;
379:     }
380:     std::ifstream& operator>>(std::ifstream& in, SB::Sokoban& a) {  // NYT
381:         std::vector<char> temp;
382:         char tempchar;
383:         int tempint;
384:         in >> tempint;           //  set height and width
385:         a.setheight(tempint);
386:         in >> tempint;
387:         a.setwidth(tempint);
388:         for (int i = 0; i < a.height(); i++) {
389:             for (int j = 0; j < a.width(); j++) {
390:                 if (!in.eof()) {
391:                 in >> tempchar;
392:                 temp.push_back(tempchar);
393:                 }
394:             }
395:             a.setvectOG(temp);
396:             a.setvectcurr(temp);
397:             temp.clear();
398:         }
399:         return in;
400:     }
401: }  //  namespace SB
402:     void SB::Sokoban::draw(sf::RenderTarget& target,
403:      sf::RenderStates states) const {
404:         //  size of each square is 64 * 64
405:         sf::Vector2f vect(0.0f, 0.0f);
406:         sf::Sprite S;
407:         for (int i = 0; i < this->height(); i++) {   //  draw walls and floo
r
408:             for (int j = 0; j < this->width(); j++) {
409:                 S.setPosition(vect);
410:                 if (this->getvectcurr(i, j) == '#') {
411:                     S.setTexture(Wall);
412:                     target.draw(S);
413:                 } else if (this->getvectcurr(i, j) == '.') {
414:                     S.setTexture(Empty);
415:                     target.draw(S);
416:                 } else if (this->getvectcurr(i, j) == 'a') {
417:                     S.setTexture(Empty);
418:                     target.draw(S);
419:                     S.setTexture(Storage);
420:                     target.draw(S);
421:                 } else if (this->getvectcurr(i, j) == 'A') {
422:                     S.setTexture(Empty);
423:                     target.draw(S);
424:                     S.setTexture(Box);
425:                     target.draw(S);
```

```
426:                 } else if (this->getvectcurr(i, j) == '@') {
427:                     S.setTexture(Empty);
428:                     target.draw(S);
429:                     S.setTexture(playerF);
430:                     target.draw(S);
431:                 } else if (this->getvectcurr(i, j) == '1') {
432:                     S.setTexture(Empty);
433:                     target.draw(S);
434:                     S.setTexture(Storage);
435:                     target.draw(S);
436:                     S.setTexture(Box);
437:                     target.draw(S);
438:                 }
439:                 vect.x = vect.x + 64;
440:             }
441:             vect.x = 0;
442:             vect.y = vect.y + 64;
443:         }
444:     }
445:     void SB::Sokoban::reset() {
446:         this->currarena = this->OGarena;
447:     }
```

```
 1: // Copyright 2024 Braden Maillet
 2: #include <fstream>
 3: #include <iostream>
 4: #include <string>
 5: #include "Sokoban.hpp"
 6: #include <SFML/Graphics.hpp>
 7:
 8:
 9:
10: #define BOOST_TEST_DYN_LINK
11: #define BOOST_TEST_MODULE Main
12: #include <boost/test/unit_test.hpp>
13:
14: //  cantmove test
15: //  Boxwall
16:
17: BOOST_AUTO_TEST_CASE(testplayerLoc) {
18:     std::ifstream myfile;
19:     SB::Sokoban a;
20:     sf::Vector2i tester(8, 5);
21:     sf::Vector2i testee;
22:     myfile.open("sokoban/level2.lvl");  //  player should be at 8 5
23:     if (!myfile.is_open()) {
24:         BOOST_FAIL("File failed to open in testplayerLoc\n");
25:     }
26:     myfile >> a;
27:     testee = static_cast<sf::Vector2<int>>(a.playerLoc());
28:     BOOST_REQUIRE(testee.x == tester.x);
29:     BOOST_REQUIRE(testee.y == tester.y);
30: }
31: BOOST_AUTO_TEST_CASE(testautowin) {
32:     std::ifstream myfile;
33:     myfile.open("sokoban/autowin.lvl");
34:     if (!myfile.is_open()) {
35:         BOOST_FAIL("File failed to open in testautowin\n");
36:     }
37:     SB::Sokoban a;
38:     myfile >> a;
39:     BOOST_REQUIRE_EQUAL(a.isWon(), true);
40: }
41: BOOST_AUTO_TEST_CASE(testautowin2) {
42:     std::ifstream myfile;
43:     myfile.open("sokoban/autowin2.lvl");
44:     if (!myfile.is_open()) {
45:         BOOST_FAIL("File failed to open in testautowin2\n");
46:     }
47:     SB::Sokoban a;
48:     myfile >> a;
49:     BOOST_REQUIRE_EQUAL(a.isWon(), true);
50: }
51: BOOST_AUTO_TEST_CASE(testpushleft) {
52:     std::ifstream myfile;
53:     sf::Vector2i tester(1, 2);
54:     sf::Vector2i testee;
55:     myfile.open("sokoban/pushleft.lvl");
56:     if (!myfile.is_open()) {
57:         BOOST_FAIL("File failed to open in testpushleft\n");
58:     }
59:     SB::Sokoban a;
60:     myfile >> a;
61:     //  player is at 3 3 and should stay there
```

```
    62:     //  after moveplayer(SB::Left);
    63:     a.movePlayer(SB::Direction::Left);
    64:     testee = static_cast<sf::Vector2i>(a.playerLoc());
    65:     BOOST_REQUIRE(testee.x == tester.x);
    66:     BOOST_REQUIRE(testee.y == tester.y);
    67: }
    68: BOOST_AUTO_TEST_CASE(testpushright) {
    69:     std::ifstream myfile;
    70:     sf::Vector2i tester(3, 2);
    71:     sf::Vector2i testee;
    72:     myfile.open("sokoban/pushright.lvl");
    73:     if (!myfile.is_open()) {
    74:         BOOST_FAIL("File failed to open in testpushright\n");
    75:     }
    76:     SB::Sokoban a;
    77:     myfile >> a;
    78:     //  player is at 2 2 and should stay there
    79:     //  after moveplayer(SB::Left);
    80:     a.movePlayer(SB::Direction::Right);
    81:     testee = static_cast<sf::Vector2i>(a.playerLoc());
    82:     BOOST_REQUIRE(testee.x == tester.x);
    83:     BOOST_REQUIRE(testee.y == tester.y);
    84: }
    85: BOOST_AUTO_TEST_CASE(testpushup) {
    86:     std::ifstream myfile;
    87:     sf::Vector2i tester(2, 1);
    88:     sf::Vector2i testee;
    89:     myfile.open("sokoban/pushup.lvl");
    90:     if (!myfile.is_open()) {
    91:         BOOST_FAIL("File failed to open in testpushup\n");
    92:     }
    93:     SB::Sokoban a;
    94:     myfile >> a;
    95:     //  std::cout << a;  //  DEBUG
    96:     //  player is at 2 2 and should stay there
    97:     //  after moveplayer(SB::Left);
    98:     a.movePlayer(SB::Direction::Up);
    99:     testee = static_cast<sf::Vector2i>(a.playerLoc());
   100:     BOOST_REQUIRE(testee.x == tester.x);
   101:     BOOST_REQUIRE(testee.y == tester.y);
   102: }
   103: BOOST_AUTO_TEST_CASE(testpushdown) {
   104:     std::ifstream myfile;
   105:     sf::Vector2i tester(2, 3);
   106:     sf::Vector2i testee;
   107:     myfile.open("sokoban/pushdown.lvl");
   108:     if (!myfile.is_open()) {
   109:         BOOST_FAIL("File failed to open in testpushdown\n");
   110:     }
   111:     SB::Sokoban a;
   112:     myfile >> a;
   113:     //  player is at 2 2 and should stay there
   114:     //  after moveplayer(SB::Left);
   115:     a.movePlayer(SB::Direction::Down);
   116:     testee = static_cast<sf::Vector2i>(a.playerLoc());
   117:     BOOST_REQUIRE(testee.x == tester.x);
   118:     BOOST_REQUIRE(testee.y == tester.y);
   119: }
   120: BOOST_AUTO_TEST_CASE(testboxwal) {
   121:     std::ifstream myfile;
   122:     sf::Vector2i tester(7, 6);
```

```
  123:     sf::Vector2i testee;
  124:     myfile.open("sokoban/level1.lvl");
  125:     if (!myfile.is_open()) {
  126:         BOOST_FAIL("File failed to open in testboxwal\n");
  127:     }
  128:     SB::Sokoban a;
  129:     myfile >> a;
  130:     //  player is at 2 2 and should stay there
  131:     //  after moveplayer(SB::Left);
  132:     a.movePlayer(SB::Direction::Right);
  133:     a.movePlayer(SB::Direction::Right);
  134:     a.movePlayer(SB::Direction::Right);
  135:     a.movePlayer(SB::Direction::Right);
  136:     a.movePlayer(SB::Direction::Right);
  137:     a.movePlayer(SB::Direction::Right);
  138:     testee = static_cast<sf::Vector2i>(a.playerLoc());
  139:     BOOST_REQUIRE(testee.x == tester.x);
  140:     BOOST_REQUIRE(testee.y == tester.y);
  141: }
  142: BOOST_AUTO_TEST_CASE(testwal) {
  143:     std::ifstream myfile;
  144:     sf::Vector2i tester(1, 6);
  145:     sf::Vector2i testee;
  146:     myfile.open("sokoban/level1.lvl");
  147:     if (!myfile.is_open()) {
  148:         BOOST_FAIL("File failed to open in testwal\n");
  149:     }
  150:     SB::Sokoban a;
  151:     myfile >> a;
  152:     //  player is at 2 2 and should stay there
  153:     //  after moveplayer(SB::Left);
  154:     a.movePlayer(SB::Direction::Left);
  155:     a.movePlayer(SB::Direction::Left);
  156:     a.movePlayer(SB::Direction::Left);
  157:     testee = static_cast<sf::Vector2i>(a.playerLoc());
  158:     BOOST_REQUIRE(testee.x == tester.x);
  159:     BOOST_REQUIRE(testee.y == tester.y);
  160: }
  161: BOOST_AUTO_TEST_CASE(testboxbox) {
  162:     std::ifstream myfile;
  163:     sf::Vector2i tester(8, 5);
  164:     sf::Vector2i testee;
  165:     myfile.open("sokoban/level2.lvl");
  166:     if (!myfile.is_open()) {
  167:         BOOST_FAIL("File failed to open in testboxbox\n");
  168:     }
  169:     SB::Sokoban a;
  170:     myfile >> a;
  171:     //  player is at 2 2 and should stay there
  172:     //  after moveplayer(SB::Left);
  173:     a.movePlayer(SB::Direction::Up);
  174:     testee = static_cast<sf::Vector2i>(a.playerLoc());
  175:     BOOST_REQUIRE(testee.x == tester.x);
  176:     BOOST_REQUIRE(testee.y == tester.y);
  177: }
  178: BOOST_AUTO_TEST_CASE(testcanmove) {
  179:     std::ifstream myfile;
  180:     sf::Vector2i tester(1, 2);
  181:     sf::Vector2i testee;
  182:     myfile.open("sokoban/pushup.lvl");
  183:     if (!myfile.is_open()) {
```

```
  184:         BOOST_FAIL("File failed to open in testcanmove\n");
  185:     }
  186:     SB::Sokoban a;
  187:     myfile >> a;
  188:
  189:     a.movePlayer(SB::Direction::Right);
  190:     a.movePlayer(SB::Direction::Down);
  191:     a.movePlayer(SB::Direction::Left);
  192:     a.movePlayer(SB::Direction::Left);
  193:     a.movePlayer(SB::Direction::Up);
  194:
  195:     testee = static_cast<sf::Vector2i>(a.playerLoc());
  196:     BOOST_REQUIRE(testee.x == tester.x);
  197:     BOOST_REQUIRE(testee.y == tester.y);
  198: }
  199: BOOST_AUTO_TEST_CASE(testmoveoffscreenLEFT) {
  200:     std::ifstream myfile;
  201:     sf::Vector2i tester(0, 2);
  202:     sf::Vector2i testee;
  203:     myfile.open("sokoban/pushup.lvl");
  204:     if (!myfile.is_open()) {
  205:         BOOST_FAIL("File failed to open in testmoveoffscreenLEFT\n");
  206:     }
  207:     SB::Sokoban a;
  208:     myfile >> a;
  209:
  210:     a.movePlayer(SB::Direction::Left);
  211:     a.movePlayer(SB::Direction::Left);
  212:     a.movePlayer(SB::Direction::Left);
  213:
  214:     testee = static_cast<sf::Vector2i>(a.playerLoc());
  215:     BOOST_REQUIRE(testee.x == tester.x);
  216:     BOOST_REQUIRE(testee.y == tester.y);
  217: }
  218: BOOST_AUTO_TEST_CASE(testmoveoffscreenRIGHT) {
  219:     std::ifstream myfile;
  220:     sf::Vector2i tester(4, 2);
  221:     sf::Vector2i testee;
  222:     myfile.open("sokoban/pushup.lvl");
  223:     if (!myfile.is_open()) {
  224:         BOOST_FAIL("File failed to open in testmoveoffscreenRight\n");
  225:     }
  226:     SB::Sokoban a;
  227:     myfile >> a;
  228:
  229:     a.movePlayer(SB::Direction::Right);
  230:     a.movePlayer(SB::Direction::Right);
  231:     a.movePlayer(SB::Direction::Right);
  232:
  233:     testee = static_cast<sf::Vector2i>(a.playerLoc());
  234:     BOOST_REQUIRE(testee.x == tester.x);
  235:     BOOST_REQUIRE(testee.y == tester.y);
  236: }
  237: BOOST_AUTO_TEST_CASE(testmoveoffscreenUP) {
  238:     std::ifstream myfile;
  239:     sf::Vector2i tester(3, 0);
  240:     sf::Vector2i testee;
  241:     myfile.open("sokoban/pushup.lvl");
  242:     if (!myfile.is_open()) {
  243:         BOOST_FAIL("File failed to open in testmoveoffscreenUP\n");
  244:     }
```

```
245:     SB::Sokoban a;
246:     myfile >> a;
247:
248:     a.movePlayer(SB::Direction::Right);
249:     a.movePlayer(SB::Direction::Up);
250:     a.movePlayer(SB::Direction::Up);
251:     a.movePlayer(SB::Direction::Up);
252:
253:     testee = static_cast<sf::Vector2i>(a.playerLoc());
254:     BOOST_REQUIRE(testee.x == tester.x);
255:     BOOST_REQUIRE(testee.y == tester.y);
256: }
257: BOOST_AUTO_TEST_CASE(testmoveoffscreenDOWN) {
258:     std::ifstream myfile;
259:     sf::Vector2i tester(2, 4);
260:     sf::Vector2i testee;
261:     myfile.open("sokoban/pushup.lvl");
262:     if (!myfile.is_open()) {
263:         BOOST_FAIL("File failed to open in testmoveoffscreenDOWN\n");
264:     }
265:     SB::Sokoban a;
266:     myfile >> a;
267:
268:     a.movePlayer(SB::Direction::Down);
269:     a.movePlayer(SB::Direction::Down);
270:     a.movePlayer(SB::Direction::Down);
271:
272:     testee = static_cast<sf::Vector2i>(a.playerLoc());
273:     BOOST_REQUIRE(testee.x == tester.x);
274:     BOOST_REQUIRE(testee.y == tester.y);
275: }
276: BOOST_AUTO_TEST_CASE(testpushoffscreenLEFT) {
277:     std::ifstream myfile;
278:     sf::Vector2i tester(1, 1);
279:     sf::Vector2i testee;
280:     myfile.open("sokoban/pushup.lvl");
281:     if (!myfile.is_open()) {
282:         BOOST_FAIL("File failed to open in testpushoffscreenLEFT\n");
283:     }
284:     SB::Sokoban a;
285:     myfile >> a;
286:
287:     a.movePlayer(SB::Direction::Right);
288:     a.movePlayer(SB::Direction::Up);
289:     a.movePlayer(SB::Direction::Left);
290:     a.movePlayer(SB::Direction::Left);
291:     a.movePlayer(SB::Direction::Left);
292:
293:     testee = static_cast<sf::Vector2i>(a.playerLoc());
294:     BOOST_REQUIRE(testee.x == tester.x);
295:     BOOST_REQUIRE(testee.y == tester.y);
296: }
297: BOOST_AUTO_TEST_CASE(testpushoffscreenRIGHT) {
298:     std::ifstream myfile;
299:     sf::Vector2i tester(3, 1);
300:     sf::Vector2i testee;
301:     myfile.open("sokoban/pushup.lvl");
302:     if (!myfile.is_open()) {
303:         BOOST_FAIL("File failed to open in testpushoffscreenRIGHT\n");
304:     }
305:     SB::Sokoban a;
```

```
   306:     myfile >> a;
   307:
   308:     a.movePlayer(SB::Direction::Left);
   309:     a.movePlayer(SB::Direction::Up);
   310:     a.movePlayer(SB::Direction::Right);
   311:     a.movePlayer(SB::Direction::Right);
   312:     a.movePlayer(SB::Direction::Right);
   313:
   314:     testee = static_cast<sf::Vector2i>(a.playerLoc());
   315:     BOOST_REQUIRE(testee.x == tester.x);
   316:     BOOST_REQUIRE(testee.y == tester.y);
   317: }
   318: BOOST_AUTO_TEST_CASE(testpushoffscreenUP) {
   319:     std::ifstream myfile;
   320:     sf::Vector2i tester(3, 1);
   321:     sf::Vector2i testee;
   322:     myfile.open("sokoban/pushup.lvl");
   323:     if (!myfile.is_open()) {
   324:         BOOST_FAIL("File failed to open in testpushoffscreenUP\n");
   325:     }
   326:     SB::Sokoban a;
   327:     myfile >> a;
   328:
   329:     a.movePlayer(SB::Direction::Left);
   330:     a.movePlayer(SB::Direction::Up);
   331:     a.movePlayer(SB::Direction::Right);
   332:     a.movePlayer(SB::Direction::Down);
   333:     a.movePlayer(SB::Direction::Right);
   334:     a.movePlayer(SB::Direction::Up);
   335:     a.movePlayer(SB::Direction::Up);
   336:
   337:     testee = static_cast<sf::Vector2i>(a.playerLoc());
   338:     BOOST_REQUIRE(testee.x == tester.x);
   339:     BOOST_REQUIRE(testee.y == tester.y);
   340: }
   341: BOOST_AUTO_TEST_CASE(testpushoffscreenDOWN) {
   342:     std::ifstream myfile;
   343:     sf::Vector2i tester(4, 3);
   344:     sf::Vector2i testee;
   345:     myfile.open("sokoban/pushup.lvl");
   346:     if (!myfile.is_open()) {
   347:         BOOST_FAIL("File failed to open in testpushoffscreenDOWN\n");
   348:     }
   349:     SB::Sokoban a;
   350:     myfile >> a;
   351:
   352:     a.movePlayer(SB::Direction::Left);
   353:     a.movePlayer(SB::Direction::Up);
   354:     a.movePlayer(SB::Direction::Right);
   355:     a.movePlayer(SB::Direction::Right);
   356:     a.movePlayer(SB::Direction::Up);
   357:     a.movePlayer(SB::Direction::Right);
   358:     a.movePlayer(SB::Direction::Down);
   359:     a.movePlayer(SB::Direction::Down);
   360:     a.movePlayer(SB::Direction::Down);
   361:     a.movePlayer(SB::Direction::Down);
   362:
   363:
   364:     testee = static_cast<sf::Vector2i>(a.playerLoc());
   365:     BOOST_REQUIRE(testee.x == tester.x);
   366:     BOOST_REQUIRE(testee.y == tester.y);
```

```
367: }
368: BOOST_AUTO_TEST_CASE(testWinLotsTargets) {
369:     std::ifstream myfile;
370:     sf::Vector2i tester(4, 3);
371:     sf::Vector2i testee;
372:     myfile.open("sokoban/lotstargets.lvl");
373:     if (!myfile.is_open()) {
374:         BOOST_FAIL("File failed to open in testWinLotsTargets\n");
375:     }
376:     SB::Sokoban a;
377:     myfile >> a;
378:
379:     a.movePlayer(SB::Direction::Up);
380:
381:     BOOST_REQUIRE_EQUAL(a.isWon(), true);
382: }
383: BOOST_AUTO_TEST_CASE(testWinlotsBoxes) {
384:     std::ifstream myfile;
385:     sf::Vector2i tester(4, 3);
386:     sf::Vector2i testee;
387:     myfile.open("sokoban/lotsBoxes.lvl");
388:     if (!myfile.is_open()) {
389:         BOOST_FAIL("File failed to open in testWinlotsBoxes\n");
390:     }
391:     SB::Sokoban a;
392:     myfile >> a;
393:
394:     a.movePlayer(SB::Direction::Up);
395:
396:     BOOST_REQUIRE_EQUAL(a.isWon(), true);
397: }
398: BOOST_AUTO_TEST_CASE(testWinMissingSymbol) {
399:     std::ifstream myfile;
400:     sf::Vector2i tester(2, 2);
401:     sf::Vector2i testee;
402:     myfile.open("sokoban/missingsymbol.lvl");
403:     if (!myfile.is_open()) {
404:         BOOST_FAIL("File failed to open in testWinMissingSymbol\n");
405:     }
406:     SB::Sokoban a;
407:     myfile >> a;
408:
409:     testee = static_cast<sf::Vector2i>(a.playerLoc());
410:     BOOST_REQUIRE(testee.x == tester.x);
411:     BOOST_REQUIRE(testee.y == tester.y);
412:     BOOST_REQUIRE_EQUAL(a.isWon(), false);
413: }
414:
415:
416:
417:
```

```
    1: CC = g++
    2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g -O3
    3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_
test_framework
    4: DEPS = Sokoban.hpp
    5: OBJECTS = Sokoban.o
    6: PROGRAM = Sokoban test Sokoban.a
    7:
    8: .PHONY: all clean lint test
    9:
   10: all:$(PROGRAM)
   11:
   12: %.o: %.cpp $(DEPS)
   13:         $(CC) $(CFLAGS) -c $<
   14: Sokoban:Sokoban.o main.o
   15:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   16: Sokoban.a:Sokoban.o
   17:         ar rcs Sokoban.a Sokoban.o
   18: test:test.o Sokoban.o
   19:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   20: clean:
   21:         rm *.o *.a $(PROGRAM)
   22: lint:
   23:         cpplint *.cpp *.hpp
   24:
   25:
```

# 4 PS4: NBody Simulation

## 4.1 General Discussion

This project was split up into two sections. The first section required that I create an environment through input from stdin with input redirection. This is in the form of a text file that looks like what can be seen below.

```
1   9
2   2.5e11
3    0.0e00    0.000e00   0.00e00   0.00e00   2e32 kevin.gif
4    6.25e10   0.000e00   0.00e00   5.66e05   6e24 mercury.gif
5   -6.25e10   0.000e00   0.00e00  -5.66e05   6e24 earth.gif
6    0          6.250e10  -5.66e05   0.00e00   6e24 mars.gif
7    0         -6.250e10   5.66e05   0.00e00   6e24 venus.gif
8    1.414e11   1.414e11   1.20e05  -1.20e05   2e24 sun.gif
9   -1.414e11   1.414e11   1.20e05   1.20e05   2e24 sun.gif
10   1.414e11  -1.414e11  -1.20e05  -1.20e05   2e24 sun.gif
11  -1.414e11  -1.414e11  -1.20e05   1.20e05   2e24 sun.gif
```

The text file contains information for a universe that will be parsed and then stored within a universe Class. The first digit is the number of celestial bodies. Every line contains position(x,y), velocity(x,y), mass and a string for an image that will be used for displaying each celestial body. The velocity and mass will be important for movement in the second portion of the project. This project is a real-world simulation of physics and how all objects have gravitational force that pulls on all other objects.

## 4.2 Implementation

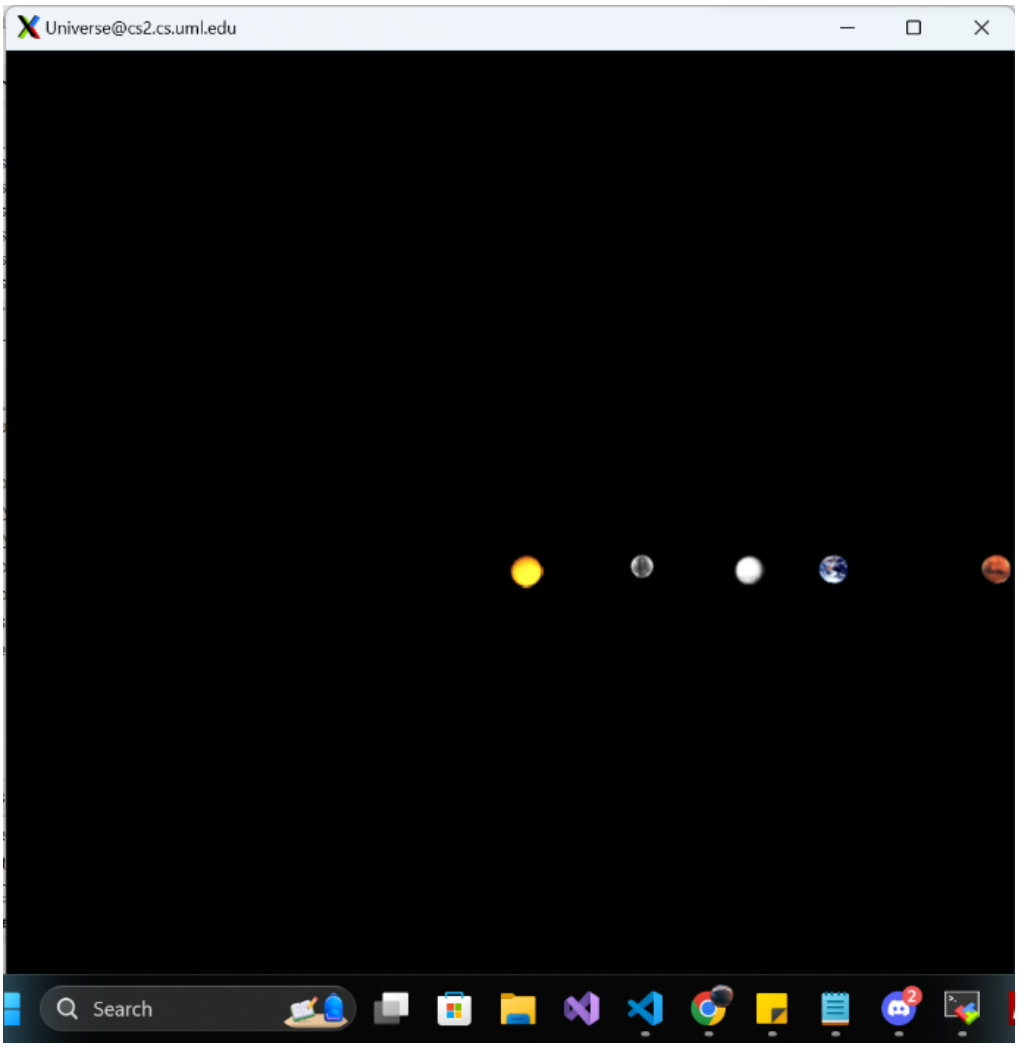
To implement this project, it is required to create both a universe class and a celestial body class. The general idea being that the universe class holds a vector of shared pointers to Celestial bodies. The universe also holds other data members including time and # of bodies. To begin the text file above is taken in from stdin and parsed into the Universe and each Celestial body class. I decided to create the SFML window so that it was always 700 by 700 pixels. The position of the celestial body is then converted to pixels(from meters) and is printed onto the

screen as if the center of the window was (0,0). After this environment is finished setting up the game loop starts, and the movement begins. The program also takes two floating point values from the terminal. The first being a Time in seconds and the second a change in time. Within the Game loop the required step() function takes a change in time and alters the data of each celestial body each iteration until the Time has been reached. The step function is the most important algorithm in all of this. It uses real world physics principles to alter the data of Celestial bodies. All bodies have a gravitational force that is expelled on all other bodies. So, for every step the position and velocity of each body is changed based on all other bodies and their velocity, position, and mass.

## 4.3 What Was Learned

This was my first time using smart pointers on a larger scale. I learned about how they are like raw pointers as well as how they are different. It is a huge benefit that there is no need to free data especially working with classes that contain a lot of different data members. I was already familiar with a lot of the SFML libraries from former assignments. Specifically, the Drawable class which was a requirement that both Universe and CelestialBody class publicly inherit. It was also interesting to create something that implemented some complex topics such as astro-physics. I was already familiar with the importance of double precision numbers when working with such large or small values, but this project reiterated this for me. There was a significant number of tests that failed when using float instead of double precision numbers because they were so much less precise. I also learned some unit testing methods using std::sstream. Specifically testing operators like << and >>.

## 4.4 Output



```
 1: //  Copyright 2024 Braden Maillet
 2: #include <iostream>
 3: #include <fstream>
 4: #include <SFML/Graphics.hpp>
 5: #include "Universe.hpp"
 6:
 7: int main(int argc, char** argv) {
 8:     const int windowsize = 700;
 9:     double timeMax = atof(argv[1]);  //  time for physics
10:     double changeOfTime = atof(argv[2]);
11:     double time = 0;
12:     NB::Universe u;
13:     std::cin >> u;
14:     std::cout << u;
15:     std::cout << "size: " << windowsize << std::endl;
16:     sf::RenderWindow window(sf::VideoMode(700, 700), "Universe");
17:     sf::View view;
18:     //  sf::Vector2u temp = window.getSize();
19:     view.setCenter(windowsize / 2, windowsize / 2);
20:     //  view.zoom(u.getzoom(window));
21:     view.setSize(u.getzoom(window), u.getzoom(window));
22:     window.setView(view);
23:
24:     while (window.isOpen()) {
25:         sf::Event event;
26:         while (window.pollEvent(event)) {
27:             if (event.type == sf::Event::Closed)
28:                 window.close();
29:         }
30:
31:             window.clear();
32:             if (time < timeMax) {
33:               window.draw(u);
34:               window.display();
35:               u.step(changeOfTime);  //  time + changeOfTime
36:               time = time + changeOfTime;
37:                   //  std::cout << "time " << time << std::endl;
38:                   //  std::cout << u;
39:                   //  std::cout << "vel x, y: " << u[0].velocity().x <<
40:                   //    " " << u[0].velocity().y << std::endl;
41:                   //  std::cout << "pos x, y: " << u[0].position().x <<
42:                   //    " " << u[0].position().y << std::endl;
43:             } else if (time >= timeMax) {
44:               std::cout << u;
45:                 //  std::cout << "vel x, y: " << u[0].velocity().x <<
46:                 //    " " << u[0].velocity().y << std::endl;
47:                 //  std::cout << "pos x, y: " << u[0].position().x <<
48:                 //    " " << u[0].position().y << std::endl;
49:               window.close();
50:             }
51:     }
52:     return 0;
53: }
```

```
    1: //  Copyright 2024 Braden Maillet
    2: #pragma once
    3: #include <vector>
    4: #include <memory>
    5: #include <iostream>
    6: #include <string>
    7: #include <SFML/Graphics/Drawable.hpp>
    8: #include <SFML/System/Vector2.hpp>
    9: #include <SFML/Graphics.hpp>
   10: #include "CelestialBody.hpp"
   11:
   12: namespace NB {
   13: class Universe : public sf::Drawable{
   14:  public:
   15:     Universe() : _currTime(0), _numberOfPlanets(), _universeRadius() {}
   16:     explicit Universe(std::string filename);
   17:     CelestialBody& getCelestialBody(int index) const;
   18:     int getNumberOfPlanets() const {return _numberOfPlanets;}
   19:     // numplanets is same as getnumberofplanets
   20:     int numPlanets() const {return _numberOfPlanets;}
   21:     double getUniverseRadius() const {return _universeRadius;}
   22:     //  radius is same as getUniverseRadius
   23:     double radius() const {return _universeRadius;}
   24:     int getPixelCount() const {return _pixelCount;}
   25:     double getzoom(sf::RenderTarget& target);
   26:     double metersPerPixel() const {return _universeRadius/_pixelCount;}
   27:     CelestialBody& operator[](int index) {return *(_p[index]);}
   28:     void step(double seconds);
   29:     void setTime(double time) {_currTime = time;}
   30:     double getTime() {return _currTime;}
   31:
   32:     friend std::ostream& operator<<(std::ostream& out, Universe& a);
   33:     friend std::istream& operator>>(std::istream& in, Universe& a);
   34:
   35:  protected:
   36:     void draw(sf::RenderTarget& target, sf::RenderStates states) const overr
ide;
   37:
   38:  private:
   39:     const int _pixelCount = 700;
   40:     double _currTime;
   41:     int _numberOfPlanets;
   42:     double _universeRadius;
   43:     std::vector<std::shared_ptr<CelestialBody>> _p;
   44: };
   45: }  //  namespace NB
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <memory>
 3: #include <iostream>
 4: #include <cmath>
 5: #include <string>
 6: #include <fstream>
 7: #include <utility>
 8: #include <SFML/Graphics/Drawable.hpp>
 9: #include <SFML/System/Vector2.hpp>
10: #include <SFML/Graphics.hpp>
11: #include "CelestialBody.hpp"
12: #include "Universe.hpp"
13: // changes internal velocity for given time
14: void NB::Universe::step(double seconds) {
15:   const double GRAV = 0.000000000066742;
16:   //  Universe holds current time
17:   //  newTime - Curr time
18:   double changeInTime = seconds;
19:   double F;
20:   double distx = 0;
21:   double disty = 0;
22:   double distR = 0;
23:   double tempForcex = 0;  //  sum of net forces
24:   double tempForcey = 0;
25:   double tempAccelx = 0;
26:   double tempAccely = 0;
27:   double tempVelx = 0;
28:   double tempVely = 0;
29:   double tempPosx = 0;
30:   double tempPosy = 0;
31:   std::pair<double, double> totalF;
32:   //  calculate force for all bodies
33:   //  calculate acceleration for all bodies
34:   for (int i = 0; i < this->_numberOfPlanets; i++) {
35:       for (int j = 0; j < this->_numberOfPlanets; j++) {
36:         if (i != j) {
37:           //  calc distance between  CB's i and j;
38:           distx = (*this)[j].getxpos() - (*this)[i].getxpos();
39:           disty = (*this)[j].getypos() - (*this)[i].getypos();
40:           distR = sqrt((distx * distx) + (disty * disty));
41:           F = (GRAV * ((*this)[i].getmass()) *
42:                   ((*this)[j].getmass()))/ (distR * distR);
43:           tempForcex = (F * distx)/ distR;
44:           tempForcey = (F * disty)/ distR;
45:         } else if (i == j) {
46:           continue;
47:         }
48:         //  calculate net force for specific CB
49:         //  ie: sum individual forces or add Force just found
50:         totalF.first = totalF.first + tempForcex;
51:         totalF.second = totalF.second + tempForcey;
52:         tempForcex = 0;
53:         tempForcey = 0;
54:       }
55:       //  set net force for i and reset net force
56:       //  set accel for i using net force ax = forcex / mass
57:       tempAccelx = totalF.first/(*this)[i].getmass();
58:       tempAccely = totalF.second/(*this)[i].getmass();
59:       tempVelx = (*this)[i].getxvel() + (changeInTime * tempAccelx);
60:       tempVely = (*this)[i].getyvel() + (changeInTime * tempAccely);
61:       (*this)[i].setAccel(tempAccelx, tempAccely);
```

```
   62:       (*this)[i].setForce(totalF.first, totalF.second);
   63:       (*this)[i].setVelocity(tempVelx, tempVely);
   64:       totalF.first = 0;
   65:       totalF.second = 0;
   66:       tempAccelx = 0;
   67:       tempAccelx = 0;
   68:       tempVelx = 0;
   69:       tempVely = 0;
   70:   }
   71:   for (int i = 0; i < this->_numberOfPlanets; i++) {
   72:     //  change pos/vel using old position and new velocity
   73:     //  new velocity has been set already for all CB's
   74:     tempPosx = (*this)[i].getxpos() + (changeInTime * (*this)[i].getxvel());
   75:     tempPosy = (*this)[i].getypos() + (changeInTime * (*this)[i].getyvel());
   76:     (*this)[i].setPos(tempPosx, tempPosy);
   77:     tempPosx = 0;
   78:     tempPosy = 0;
   79:   }
   80:   this->setTime(this->getTime() + seconds);  //  set new time
   81: }
   82:
   83: NB::Universe::Universe(std::string filename) {
   84:   std::ifstream myfile;
   85:   //  std::string file = "nbody-full/";
   86:   //  std::string temp = filename;
   87:   //  file.append(temp);
   88:   myfile.open(filename);
   89:   if (!myfile.is_open()) {
   90:     std::cout << "failed to construct Universe due to faulty file: "
   91:       <<  filename << std::endl;
   92:       return;
   93:   }
   94:   myfile >> (*this);
   95:   myfile.close();
   96: }
   97: //  checks in index is out of bounds but same as []
   98: NB::CelestialBody& NB::Universe::getCelestialBody(int index) const {
   99:     if (index > this->_numberOfPlanets) {
  100:         std::cout <<
  101:         "ERROR: index out of bounds in getter function, returning _p[0]"
  102:         << std::endl;
  103:         return *(this->_p[0]);
  104:     }
  105:     return *(this->_p[index]);
  106: }
  107: //  finds the celestial body with the highest
  108: //  cartesian coordinate. ie furthest to edge.
  109: double NB::Universe::getzoom(sf::RenderTarget& w) {
  110:   double greatest = 0.0;
  111:   sf::Vector2i v(w.getSize());
  112:   double pixels = static_cast<double>(v.x);
  113:   for (int i = 0; i < this->_numberOfPlanets; i++) {
  114:       if ((_p[i])->findPos((_p[i])->getxpos()) >= greatest) {
  115:          greatest = (_p[i])->findPos((_p[i])->getxpos());
  116:       }
  117:       if ((_p[i])->findPos((_p[i])->getypos()) >= greatest) {
  118:          greatest = (_p[i])->findPos((_p[i])->getypos());
  119:       }
  120:   }
  121:   greatest = abs(greatest);
  122:   greatest = greatest + (pixels/2);
```

```
123:     std::cout << "pixels - greatest: "
124:     << greatest << std::endl;
125:   return greatest + 15;
126: }
127:
128: void NB::Universe::draw(sf::RenderTarget& target,
129:     sf::RenderStates states) const {
130:   for (int i = 0; i < this->_numberOfPlanets; i++) {
131:     target.draw(*(this->_p[i]));
132:   }
133: }
134: namespace NB {
135: std::ostream& operator<<(std::ostream& out, Universe& a) {  //  NYT
136:   out << std::scientific;
137:   out << a._numberOfPlanets << std::endl <<
138:      a._universeRadius << std::endl;
139:   for (int i = 0; i < a._numberOfPlanets; i++) {
140:       out << *(a._p[i]);
141:   }
142:   return out;
143: }
144: std::istream& operator>>(std::istream& in, Universe& a) {  //  NYT
145:   NB::CelestialBody temp;
146:   in >> a._numberOfPlanets;
147:   in >> a._universeRadius;
148:   for (int i = 0; i < a._numberOfPlanets; i++) {
149:       a._p.push_back(std::make_shared<CelestialBody>());
150:       in >> *(a._p[i]);
151:       //  sets the pixel count of each body
152:       (*(a._p[i])).setPixel(a.getPixelCount());
153:       //  sets the radius of each body
154:       (*(a._p[i])).setRadius(a.getUniverseRadius());
155:   }
156:   return in;
157: }
158: }  //  namespace NB
```

```
    1: //  Copyright 2024 Braden Maillet
    2: #pragma once
    3: #include <vector>
    4: #include <string>
    5: #include <utility>
    6: #include <memory>
    7: #include <iostream>
    8: #include <SFML/Graphics/Drawable.hpp>
    9: #include <SFML/System/Vector2.hpp>
   10: #include <SFML/Graphics.hpp>
   11: namespace NB {
   12: class CelestialBody : public sf::Drawable{
   13:  public:
   14:     CelestialBody() : _posSF(), _velSF(), _mass(), _pos(), _vel(),
   15:       _force(), _accel(), _filename(), _pixelCount(), _radius() {}
   16:     sf::Vector2f position() {return _posSF;}
   17:     sf::Vector2f velocity() {return _velSF;}
   18:     void setVelocity(double velx, double vely);
   19:     void setPos(double posx, double posy);
   20:     double getxpos() const {return _pos.first;}
   21:     double getypos() const {return _pos.second;}
   22:     double getxvel() const {return _vel.first;}
   23:     double getyvel() const {return _vel.second;}
   24:     double getmass() const {return _mass;}
   25:     double mass() const {return _mass;}
   26:
   27:     void setForce(double Fx, double Fy);
   28:     void setAccel(double Ax, double Ay);
   29:     void setPixel(int size) {_pixelCount = size;}
   30:     void setRadius(double rad) {_radius = rad;}
   31:     double findPos(double pos) const;
   32:
   33:     friend std::ostream& operator<<(std::ostream& out, CelestialBody& a);
   34:     friend std::istream& operator>>(std::istream& in, CelestialBody& a);
   35:
   36:  protected:
   37:     void draw(sf::RenderTarget& target, sf::RenderStates states) const overr
ide;
   38:
   39:  private:
   40:     //  decided to store both a pair
   41:     //  and a vector2f so i could have double
   42:     //  and float values for return and calcs.
   43:     //  _posSF has correct - and + y
   44:     sf::Vector2f _posSF;
   45:     sf::Vector2f _velSF;
   46:     double _mass;
   47:     std::pair<double, double> _pos;  //  (x,y)
   48:     std::pair<double, double> _vel;  //  (x,y)
   49:     std::pair<double, double> _force;
   50:     std::pair<double, double> _accel;
   51:     std::string _filename;
   52:     sf::Texture _texture;
   53:     sf::Sprite _sprite;
   54:     int _pixelCount;
   55:     double _radius;
   56: };
   57: }  //  namespace NB
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <vector>
 3: #include <string>
 4: #include <utility>
 5: #include <memory>
 6: #include <iomanip>
 7: #include <iostream>
 8: #include <fstream>
 9: #include <SFML/Graphics/Drawable.hpp>
10: #include <SFML/System/Vector2.hpp>
11: #include <SFML/Graphics.hpp>
12: #include "CelestialBody.hpp"
13:
14: void NB::CelestialBody::setForce(double Fx, double Fy) {
15:   _force.first = Fx;
16:   _force.second = Fy;
17: }
18: void NB::CelestialBody::setAccel(double Ax, double Ay) {
19:   _accel.first = Ax;
20:   _accel.second = Ay;
21: }
22: void NB::CelestialBody::setVelocity(double velx, double vely) {
23:   _vel.first = velx;
24:   _vel.second = vely;
25:   _velSF.x = velx;
26:   _velSF.y = vely;
27: }
28: void NB::CelestialBody::setPos(double posx, double posy) {
29:   _pos.first = posx;
30:   _pos.second = posy;
31:   _posSF.x = posx;
32:   _posSF.y = posy;
33: }
34: void NB::CelestialBody::draw(sf::RenderTarget& target,
35:          sf::RenderStates states) const {
36:   sf::Sprite S = _sprite;
37:   double temp;
38:   sf::Vector2f vect(findPos(_pos.first) + (_pixelCount/2), 0);  //  x and y
39:     if ((findPos(_pos.second) + (_pixelCount/2)) == 350) {
40:         vect.y = 350;
41:     }
42:     if ((findPos(_pos.second) + (_pixelCount/2)) > 350) {
43:       temp = findPos(_pos.second);
44:       vect.y = (_pixelCount/2) - temp;
45:     }
46:     if ((findPos(_pos.second) + (_pixelCount/2)) < 350) {
47:       temp = findPos(_pos.second);
48:       vect.y = (_pixelCount/2) - temp;
49:     }
50:     //  std::cout << std::fixed << std::setprecision(2)
51:     //  << "Celestial body (x,y) and rad: " <<
52:     //  vect.x << " "
53:     //  << (findPos(_pos.second) + (_pixelCount/2)) << " " << vect.y
54:     //  << " " << std::scientific << this->_radius << " "
55:     //  << this->_filename << std::endl;  //  Debug
56:
57:   S.setPosition(vect);
58:   S.setTexture(_texture);
59:   target.draw(S);
60: }
61: double NB::CelestialBody::findPos(double pos) const {
```

```
   62:   return ((pos/_radius) * (_pixelCount/2));
   63: }
   64: namespace NB {
   65: std::ostream& operator<<(std::ostream& out, CelestialBody& a) {
   66:   out << a._pos.first << "  " << a._pos.second <<
   67:       "  " << a._vel.first << "  " << a._vel.second <<
   68:       "  " << a._mass << "  " << a._filename << std::endl;
   69:   return out;
   70: }
   71: std::istream& operator>>(std::istream& in, CelestialBody& a) {
   72:   std::string file = "nbody/";
   73:   std::string temp;
   74:   in >> a._pos.first;  //  _pos x
   75:   a._posSF.x = a._pos.first;  //  SF
   76:   in >> a._pos.second;  //  _pos y
   77:   a._posSF.y = a._pos.second;  //  SF
   78:   in >> a._vel.first;  //  _vel x
   79:   a._velSF.x = a._vel.first;  //  SF
   80:   in >> a._vel.second;  //  _vel y
   81:   a._velSF.y = a._vel.second;  //  SF
   82:   in >> a._mass;
   83:   in >> a._filename;
   84:   temp = a._filename;
   85:   file.append(temp);
   86:   if (!a._texture.loadFromFile(file)) {
   87:     std::cout << "failed to load " << file << std::endl;
   88:   }
   89:   a._sprite.setTexture(a._texture);
   90:   return in;
   91: }
   92: }  //  namespace NB
```

```
    1: CC = g++
    2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g -O3
    3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_
test_framework
    4: DEPS = Universe.hpp CelestialBody.hpp
    5: OBJECTS = Universe.o CelestialBody.o
    6: PROGRAM = NBody test NBody.a
    7:
    8: .PHONY: all clean lint test
    9:
   10: all:$(PROGRAM)
   11:
   12: %.o: %.cpp $(DEPS)
   13:         $(CC) $(CFLAGS) -c $<
   14: NBody:Universe.o CelestialBody.o main.o
   15:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   16: NBody.a:Universe.o CelestialBody.o
   17:         ar rcs NBody.a Universe.o CelestialBody.o
   18: test:test.o Universe.o CelestialBody.o
   19:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   20: clean:
   21:         rm *.o *.a $(PROGRAM)
   22: lint:
   23:         cpplint *.cpp *.hpp
   24:
   25:
```

```
    1: //  Copyright 2024 Braden Maillet
    2: #include <iostream>
    3: #include <fstream>
    4: #include <SFML/Graphics.hpp>
    5: #include "Universe.hpp"
    6: #include "CelestialBody.hpp"
    7:
    8: #define BOOST_TEST_DYN_LINK
    9: #define BOOST_TEST_MODULE Main
   10: #include <boost/test/unit_test.hpp>
   11:
   12: BOOST_AUTO_TEST_CASE(testgetbodies) {
   13:   NB::Universe u;
   14:   std::stringstream s;
   15:     s << "3\n" << "6.0E8\n" <<
   16:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
   17:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
   18:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
   19:       "1.241117e+02 8.00e+23 earth.gif\n" <<
   20:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
   21:       "1.241117e+02 8.00e+23 earth.gif\n";
   22:   s >> u;
   23:   BOOST_REQUIRE(u.numPlanets() == 3);
   24: }
   25: BOOST_AUTO_TEST_CASE(testgetradius) {
   26:   NB::Universe u;
   27:   std::stringstream s;
   28:     s << "3\n" << "6.0E8\n" <<
   29:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
   30:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
   31:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
   32:       "1.241117e+02 8.00e+23 earth.gif\n" <<
   33:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
   34:       "1.241117e+02 8.00e+23 earth.gif\n";
   35:   s >> u;
   36:   //  std::cout <<  "testgetradius " << u.radius() << std::endl;
   37:   BOOST_REQUIRE(u.radius() == 6.0E8);
   38: }
   39: BOOST_AUTO_TEST_CASE(testCBodyPos) {
   40:   NB::Universe u;
   41:   std::stringstream s;
   42:     s << "3\n" << "6.0E8\n" <<
   43:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
   44:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
   45:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
   46:       "1.241117e+02 8.00e+23 earth.gif\n" <<
   47:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
   48:       "1.241117e+02 8.00e+23 earth.gif\n";
   49:   s >> u;
   50:   sf::Vector2f c(u[0].position());
   51:   //  std::cout << "testCBodyPos debugging: " << c.x << " " << c.y << std::e
ndl;
   52:   BOOST_REQUIRE_CLOSE(c.x, static_cast<float>(5.000000e+08), .01);
   53:   BOOST_REQUIRE_CLOSE(c.y, static_cast<float>(0.000000e+00), .01);
   54: }
   55: BOOST_AUTO_TEST_CASE(testCBodyvel) {
   56:   NB::Universe u;
   57:   std::stringstream s;
   58:     s << "3\n" << "6.0E8\n" <<
   59:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
   60:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
```

```
   61:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
   62:       "1.241117e+02 8.00e+23 earth.gif\n" <<
   63:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
   64:       "1.241117e+02 8.00e+23 earth.gif\n";
   65:   s >> u;
   66:   //  std::cout << "testCBodyvel debugging: " << u[0] << " "  << std::endl;
   67:   sf::Vector2f c(u[0].velocity());
   68:   //  std::cout << "testCBodyvel debugging: " << c.x << " " << c.y << std::e
ndl;
   69:   BOOST_REQUIRE_CLOSE(c.x, static_cast<float>(0.000000e+00), .01);
   70:   BOOST_REQUIRE_CLOSE(c.y, static_cast<float>(-2.482233e+02), .01);
   71: }
   72: BOOST_AUTO_TEST_CASE(testCBodymass) {
   73:   NB::Universe u;
   74:   std::stringstream s;
   75:     s << "3\n" << "6.0E8\n" <<
   76:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
   77:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
   78:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
   79:       "1.241117e+02 8.00e+23 earth.gif\n" <<
   80:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
   81:       "1.241117e+02 8.00e+23 earth.gif\n";
   82:   s >> u;
   83:   //  std::cout << "testCBodyvel debugging: "
   84:   //  << u[0]->mass() << " "  << std::endl;
   85:   BOOST_REQUIRE_CLOSE(u[0].mass(), 8.00e+23, .01);
   86: }
   87: BOOST_AUTO_TEST_CASE(testUout) {
   88:   NB::Universe u;
   89:   std::stringstream s;
   90:     s << "3\n" << "6.0E8\n" <<
   91:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
   92:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
   93:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
   94:       "1.241117e+02 8.00e+23 earth.gif\n" <<
   95:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
   96:       "1.241117e+02 8.00e+23 earth.gif\n";
   97:   s >> u;
   98:   int bodycount;
   99:   double radius;
  100:   double posx;
  101:   double posy;
  102:   double velx;
  103:   double vely;
  104:   double mass;
  105:   std::string str;
  106:   s << u;
  107:   s >> bodycount;
  108:   s >> radius;
  109:   s >> posx;
  110:   s >> posy;
  111:   s >> velx;
  112:   s >> vely;
  113:   s >> mass;
  114:   s >> str;
  115:
  116:   BOOST_REQUIRE(bodycount == 3);
  117:   BOOST_REQUIRE_CLOSE(radius, 6.0E8, .01);
  118:   BOOST_REQUIRE_CLOSE(posx, 5.000000e+08, .01);
  119:   BOOST_REQUIRE_CLOSE(posy, 0.000000e+00, .01);
  120:   BOOST_REQUIRE_CLOSE(velx, 0.000000e+00, .01);
```

```
  121:   BOOST_REQUIRE_CLOSE(vely, -2.482233e+02, .01);
  122:   BOOST_REQUIRE_CLOSE(mass, 8.00e+23, .01);
  123:   BOOST_REQUIRE(str == "earth.gif");
  124: }
  125: BOOST_AUTO_TEST_CASE(testCBout) {
  126:   NB::Universe u;
  127:   std::stringstream s;
  128:     s << "3\n" << "6.0E8\n" <<
  129:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
  130:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
  131:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
  132:       "1.241117e+02 8.00e+23 earth.gif\n" <<
  133:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
  134:       "1.241117e+02 8.00e+23 earth.gif\n";
  135:   s >> u;
  136:   NB::CelestialBody c = u[0];
  137:   double posx;
  138:   double posy;
  139:   double velx;
  140:   double vely;
  141:   double mass;
  142:   std::string str;
  143:   s << c;
  144:   s >> posx;
  145:   s >> posy;
  146:   s >> velx;
  147:   s >> vely;
  148:   s >> mass;
  149:   s >> str;
  150:
  151:   BOOST_REQUIRE_CLOSE(posx, 5.000000e+08, .01);
  152:   BOOST_REQUIRE_CLOSE(posy, 0.000000e+00, .01);
  153:   BOOST_REQUIRE_CLOSE(velx, 0.000000e+00, .01);
  154:   BOOST_REQUIRE_CLOSE(vely, -2.482233e+02, .01);
  155:   BOOST_REQUIRE_CLOSE(mass, 8.00e+23, .01);
  156:   BOOST_REQUIRE(str == "earth.gif");
  157: }
  158: BOOST_AUTO_TEST_CASE(teststepVEL) {
  159:   NB::Universe u;
  160:   std::stringstream s;
  161:   sf::Vector2f vel;
  162:     s << "3\n" << "6.0E8\n" <<
  163:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
  164:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
  165:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
  166:       "1.241117e+02 8.00e+23 earth.gif\n" <<
  167:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
  168:       "1.241117e+02 8.00e+23 earth.gif\n";
  169:   s >> u;
  170:   u.step(25000.0);
  171:   vel = u[0].velocity();
  172:   BOOST_REQUIRE_CLOSE(vel.x, -3.082681e+00, .1);
  173:   BOOST_REQUIRE_CLOSE(vel.y, -2.482233e+02, .1);
  174: }
  175: BOOST_AUTO_TEST_CASE(teststepPOS) {
  176:   NB::Universe u;
  177:   std::stringstream s;
  178:   sf::Vector2f pos;
  179:     s << "3\n" << "6.0E8\n" <<
  180:     " 5.000000e+08  0.000000e+00  0.000000e+00" <<
  181:       "-2.482233e+02 8.00e+23 earth.gif\n" <<
```

```
  182:     "-2.500000e+08  4.330127e+08  2.149677e+02" <<
  183:       "1.241117e+02 8.00e+23 earth.gif\n" <<
  184:     "-2.500000e+08 -4.330127e+08 -2.149677e+02" <<
  185:       "1.241117e+02 8.00e+23 earth.gif\n";
  186:   s >> u;
  187:   u.step(25000.0);
  188:   pos = u[0].position();
  189:
  190:   BOOST_REQUIRE_CLOSE(pos.x, 4.999229e+08, .1);
  191:   BOOST_REQUIRE_CLOSE(pos.y, -6.205582e+06, .1);
  192: }
  193: BOOST_AUTO_TEST_CASE(testprec5) {
  194:   NB::Universe u("nbody-full/uniform3.txt");
  195:   sf::Vector2f vel;
  196:   u.step(25000.0);
  197:   u.step(25000.0);
  198:   u.step(25000.0);
  199:   u.step(25000.0);
  200:   u.step(25000.0);
  201:   //  std::cout << u << std::endl;
  202:   vel = u[0].velocity();
  203:   BOOST_REQUIRE_CLOSE(vel.x, -1.541104e+01, .1);
  204:   BOOST_REQUIRE_CLOSE(vel.y, -2.478405e+02, .1);
  205: }
  206: BOOST_AUTO_TEST_CASE(testprec10) {
  207:   NB::Universe u("nbody-full/uniform3.txt");
  208:   sf::Vector2f vel;
  209:   u.step(25000.0);
  210:   u.step(25000.0);
  211:   u.step(25000.0);
  212:   u.step(25000.0);
  213:   u.step(25000.0);
  214:   u.step(25000.0);
  215:   u.step(25000.0);
  216:   u.step(25000.0);
  217:   u.step(25000.0);
  218:   u.step(25000.0);
  219:   vel = u[0].velocity();
  220:   //  std::cout << u << std::endl;
  221:
  222:   BOOST_REQUIRE_CLOSE(vel.x, -3.078049e+01, .1);
  223:   BOOST_REQUIRE_CLOSE(vel.y, -2.465010e+02, .1);
  224: }
  225: BOOST_AUTO_TEST_CASE(testgrav) {
  226:   NB::Universe u("nbody-full/uniform3.txt");
  227:   sf::Vector2f vel;
  228:   u.step(25000.0);
  229:   u.step(25000.0);
  230:   u.step(25000.0);
  231:   u.step(25000.0);
  232:   u.step(25000.0);
  233:   u.step(25000.0);
  234:   u.step(25000.0);
  235:   u.step(25000.0);
  236:   u.step(25000.0);
  237:   u.step(25000.0);
  238:   vel = u[0].velocity();
  239:   //  std::cout << u << std::endl;
  240:   if (vel.x > 0) {
  241:     BOOST_TEST_FAIL("Gravity inverted");
  242:   }
```

```
  243: }
  244: BOOST_AUTO_TEST_CASE(testdelta) {
  245:   NB::Universe u("nbody-full/uniform3.txt");
  246:   sf::Vector2f vel1;
  247:   sf::Vector2f vel2;
  248:   u.step(25000.0);
  249:   vel1 = u[0].velocity();
  250:   u.step(25000.0);
  251:   vel2 = u[0].velocity();
  252:
  253:   BOOST_REQUIRE_CLOSE(vel1.x, -3.082681e+00, .1);
  254:   BOOST_REQUIRE_CLOSE(vel1.y, -2.482233e+02, .1);
  255:   BOOST_REQUIRE_CLOSE(vel2.x, -6.165600e+00, .1);
  256:   BOOST_REQUIRE_CLOSE(vel2.y, -2.481850e+02, .1);
  257: }
  258: BOOST_AUTO_TEST_CASE(testposchange) {
  259:   NB::Universe u("nbody-full/uniform3.txt");
  260:   sf::Vector2f pos1;
  261:   sf::Vector2f pos2;
  262:   u.step(25000.0);
  263:   pos1 = u[0].position();
  264:   u.step(25000.0);
  265:   pos2 = u[0].position();
  266:
  267:   BOOST_REQUIRE_CLOSE(pos1.x, 4.999229e+08, .1);
  268:   BOOST_REQUIRE_CLOSE(pos1.y, -6.205582e+06, .1);
  269:   BOOST_REQUIRE_CLOSE(pos2.x, 4.997688e+08, .1);
  270:   BOOST_REQUIRE_CLOSE(pos2.y, -1.241021e+07, .1);
  271: }
```

# 5 PS5: DNA Alignment

## 5.1 General Discussion

In the study of Computational Biology there is a need to detect the percentage of similarity between two strings of DNA. For reasons that are outside of my knowledge as a computer scientist. To detect the level of similarity between two strings there needs to be a method of matching those two strings which are not necessarily the same length. Within this project I developed a program that implements the Needleman-Wuncsh algorithm for string alignment. The algorithm creates a matrix using the two strings as guides and calculates all possible different string alignments. Once the matrix is created, the top left corner or (0, 0) will be the optimal "cost" of alignment. Cost of an alignment is arbitrary and is determined by the amount of matches (0 cost) mismatches (1 cost) and gaps (2 cost). This is visualized below.

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

## 5.2 Implementation

To implement this algorithm, I started by creating a class called EDistance. This class held as private data members: a matrix(vector<vector<int>>), both strings being aligned and an Enum which declares the arbitrary cost values of match, mismatch, and gap. The class contains a value constructor which takes two strings

and initialized all data members including the matrix with all 0's. A member function optdistance() takes the class instance and calculates all string alignments and stores them within the matrix row by row. It will then return the cost of the optimal alignment that is guaranteed by the algorithm. A separate member function alignment will trace back through the matrix and return an optimal alignment in the form of a string that looks like the figure in 5.4 Output.

## 5.3 What Was Learned

I learned a lot about optimal practices when it comes to matrices. When you are iterating over a matrix you should always iterate the first subscript operator in the first loop and the second subscript operator in the second loop. It is far more time efficient because while memory is depicted as a matrix it is likely a long string of integers. So matrix[i+1][j] and matrix[i][j] are very far apart while matrix[i][j+1] and matrix[i][j] are right next to each other. I was already familiar with input redirect that was used to take a text file as stdin. In hindsight there are some optimizations that could be made with my program. While doing research I discovered that in industry there are a lot of methods to make this algorithm more space efficient. At the current moment it is O(m*n) space efficiency. Which is problematic for larger strings. One way to cut down on this is to not calculate the whole matrix. Generally, the strings created in the top right and bottom left of the matrix are improbable to be the most efficient. So, these edges can be cut out of the program to save space.

## 5.4 Output

```
bmaillet@cs5:~/Comp4/ps5$ ./EDistance < sequence/example10.txt
Edit distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
Execution time is 0.00518 seconds
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <vector>
 3: #include <string>
 4: #include <iostream>
 5: #include <SFML/System.hpp>
 6: #include "EDistance.hpp"
 7:
 8: int main(int argc, char* argv[]) {
 9:   sf::Clock clock;
10:   std::string input1, input2;
11:   std::cin >> input1;
12:   std::cin >> input2;
13:   EDistance E(input1, input2);
14:   int x = E.optDistance();
15:   std::cout << "Edit distance = " <<
16:     x << std::endl;
17:   std::cout << E.alignment();
18:   //  std::cout << E << std::endl;
19:   sf::Time t = clock.getElapsedTime();
20:   std::cout << "Execution time is " <<
21:      t.asSeconds() << " seconds" << std::endl;
22:   return 0;
23: }
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #pragma once
 3: #include <vector>
 4: #include <string>
 5: #include <iostream>
 6:
 7: class EDistance {
 8:  public:
 9:   EDistance(std::string string1, std::string string2);
10:   static int penalty(char a, char b);
11:   static int min3(int a, int b, int c);
12:   //  optdistance returns the top left of the matrix ie
13:   //  the most efficient arrangement value
14:   int optDistance();
15:   //  aligment will trace back through the matrix
16:   std::string alignment();
17:
18:   friend std::ostream& operator<<(std::ostream& out, EDistance& e);
19:  private:
20:   //  match mismatch gap score
21:   enum MMG {match = 0, mis = 1, gap = 2};
22:   std::vector<std::vector<int>> matrix;
23:   std::string str1;
24:   std::string str2;
25: };
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <vector>
 3: #include <string>
 4: #include "EDistance.hpp"
 5:
 6: EDistance::EDistance(std::string string1, std::string string2) {
 7:   std::vector<int> temp;
 8:   int columLen = string1.size() + 1;
 9:   int rowLen = string2.size() + 1;
10:   temp.assign(rowLen, 0);
11:   this->str1 = string1;
12:   this->str2 = string2;
13:   //  creates matrix of 0's at correct size
14:   //  may just be less than
15:   for (int i = 0; i <= columLen; i++) {
16:     //  this is a row
17:     this->matrix.push_back(temp);
18:   }
19: }
20: int EDistance::penalty(char a, char b) {
21:   return (a == b) ? match : mis;
22: }
23: int EDistance::min3(int a, int b, int c) {
24:   if (a <= b) {
25:     return (a <= c)? a : c;
26:   } else if (b <= c) {
27:     return (b <= c)? b : c;
28:   } else {
29:     return c;
30:   }
31:   std::cout << "min3 " << a << " " << b << " " << c << std::endl;
32:
33:   return -1;
34: }
35: //  optdistance returns the top left of the matrix ie
36: //  the most efficient arrangement value
37: int EDistance::optDistance() {
38:   int columLen = this->str1.size();
39:   int rowLen = this->str2.size();
40:   int count = 0;
41:   int j, k, i, l;
42:   //  implement creation of graph
43:   //  create edges of graph
44:   for (j = rowLen; j >= 0; j--) {
45:     this->matrix[columLen][j] = count;
46:     count += 2;
47:   }
48:   count = 0;
49:   for (k = columLen; k >= 0; k--) {
50:     this->matrix[k][rowLen] = count;
51:     count += 2;
52:   }
53:   for (i = columLen - 1; i >= 0; i--) {
54:     for (l = rowLen - 1; l >= 0; l--) {
55:       //  std::cout << i << " " << l << std::endl;
56:       //  min of three operations
57:       this->matrix[i][l] = min3(
58:         (this->matrix[i+1][l] + 2),
59:         (this->matrix[i][l+1] + 2),
60:         (this->matrix[i+1][l+1] +
61:         (penalty(this->str1[i],
```

```
   62:         this->str2[l])));
   63:     }
   64:   }
   65:   return this->matrix[0][0];
   66: }
   67: //  aligment will trace back through the matrix
   68: std::string EDistance::alignment() {
   69:   int columLen = this->str1.size();
   70:   int rowLen = this->str2.size();
   71:   int i = 0, j = 0;
   72:   std::string final;
   73:   while (1) {
   74:       if (i >= columLen) {
   75:         //  reached edge
   76:         //  push gaps to string 2
   77:         //  return
   78:         for (; j < rowLen; j++) {
   79:           final.push_back('-');
   80:           final.push_back(' ');
   81:           final.push_back(this->str2[j]);
   82:           final = final + " 2\n";
   83:         }
   84:         return final;
   85:       }
   86:       if (j >= rowLen) {
   87:         //  reached edge
   88:         //  push gaps to string 1
   89:         //  return
   90:         if (i == columLen)
   91:           return final;
   92:         for (; i < columLen; i++) {
   93:           final.push_back(this->str1[i]);
   94:           final.push_back(' ');
   95:           final.push_back('-');
   96:           final = final + " 2\n";
   97:         }
   98:         return final;
   99:       }
  100:       //  match case
  101:       if ((this->matrix[i][j] == this->matrix[i + 1][j + 1])
  102:         && (this->str1[i] == this->str2[j])) {
  103:         //  std::cout << "match" << std::endl;
  104:         final.push_back(this->str1[i]);
  105:         final.push_back(' ');
  106:         final.push_back(this->str2[j]);
  107:         final = final + " 0\n";
  108:         i++;
  109:         j++;
  110:       }
  111:       //  mismatch case
  112:       if ((this->matrix[i][j] - 1 == this->matrix[i + 1][j + 1])
  113:         && (this->str1[i] != this->str2[j])) {
  114:         //  std::cout << "mismatch" << std::endl;
  115:         final.push_back(this->str1[i]);
  116:         final.push_back(' ');
  117:         final.push_back(this->str2[j]);
  118:         final = final + " 1\n";
  119:         i++;
  120:         j++;
  121:       }
  122:       //  gap case Right
```

```
123:       if (this->matrix[i][j] - 2 == this->matrix[i][j + 1]) {
124:         //  std::cout << "GR" << std::endl;
125:         final.push_back('-');
126:         final.push_back(' ');
127:         final.push_back(this->str2[j]);
128:         final = final + " 2\n";
129:         j++;
130:       }
131:       //  gap case Right
132:       if (this->matrix[i][j] - 2 == this->matrix[i + 1][j]) {
133:         //  std::cout << "GR" << std::endl;
134:         final.push_back(this->str1[i]);
135:         final.push_back(' ');
136:         final.push_back('-');
137:         final = final + " 2\n";
138:         i++;
139:       }
140:   }
141: }
142: //  output operator for debugging
143: std::ostream& operator<<(std::ostream& out, EDistance& e) {
144:   int i, j;
145:   int columLen = e.str1.size() + 1;
146:   int rowLen = e.str2.size() + 1;
147:   for (i = 0; i < columLen; i++) {
148:     for (j = 0; j < rowLen; j++) {
149:       out << e.matrix[i][j] << " ";
150:     }
151:     out << std::endl;
152:   }
153:   return out;
154: }
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <iostream>
 3: #include <fstream>
 4: #include "EDistance.hpp"
 5:
 6: #define BOOST_TEST_DYN_LINK
 7: #define BOOST_TEST_MODULE Main
 8: #include <boost/test/unit_test.hpp>
 9:
10: BOOST_AUTO_TEST_CASE(testcost1) {
11:   EDistance E("12345", "34567");
12:
13:   BOOST_CHECK_EQUAL(E.optDistance(), 5);
14: }
15: BOOST_AUTO_TEST_CASE(testcost2) {
16:   EDistance E("atattat", "tattata");
17:
18:   BOOST_CHECK_EQUAL(E.optDistance(), 4);
19: }
20: BOOST_AUTO_TEST_CASE(testcost3) {
21:   EDistance E("atattat", "tattata");
22:
23:   BOOST_CHECK_EQUAL(E.optDistance(), 4);
24: }
25: BOOST_AUTO_TEST_CASE(testcost4) {
26:   EDistance E("AACAGTTACC", "TAAGGTCA");
27:
28:   BOOST_CHECK_EQUAL(E.optDistance(), 7);
29: }
30: BOOST_AUTO_TEST_CASE(testendgaps) {
31:   EDistance E("atattat", "tattata");
32:   std::string test = "a - 2\nt t 0\na a 0\nt t 0\n"
33:                      "t t 0\na a 0\nt t 0\n- a 2\n";
34:   E.optDistance();
35:   BOOST_CHECK_EQUAL(E.alignment(), test);
36: }
37: BOOST_AUTO_TEST_CASE(testmin3a) {
38:   EDistance E("atattat", "tattata");
39:   BOOST_CHECK_EQUAL(E.min3(1, 2, 3), 1);
40: }
41: BOOST_AUTO_TEST_CASE(testmin3b) {
42:   EDistance E("atattat", "tattata");
43:   BOOST_CHECK_EQUAL(E.min3(2, 1, 3), 1);
44: }
45: BOOST_AUTO_TEST_CASE(testmin3c) {
46:   EDistance E("atattat", "tattata");
47:   BOOST_CHECK_EQUAL(E.min3(3, 2, 1), 1);
48: }
49: BOOST_AUTO_TEST_CASE(testpenaltymis) {
50:   EDistance E("atattat", "tattata");
51:   BOOST_CHECK_EQUAL(E.penalty('a', 'b'), 1);
52: }
53: BOOST_AUTO_TEST_CASE(testpenaltymatch) {
54:   EDistance E("atattat", "tattata");
55:   BOOST_CHECK_EQUAL(E.penalty('a', 'a'), 0);
56: }
57: BOOST_AUTO_TEST_CASE(testswp) {
58:   EDistance E("atattat", "tattata");
59:   std::string test = "- a 2\n"
60:                      "t t 0\n"
61:                      "a a 0\n"
```

```
   62:                         "t t 0\n"
   63:                         "t t 0\n"
   64:                         "a a 0\n"
   65:                         "t t 0\n"
   66:                         "a - 2\n";
   67:   E.optDistance();
   68:   BOOST_CHECK_NE(E.alignment(), test);
   69: }
```

```
    1: CC = g++
    2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g -O3
    3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_
test_framework
    4: DEPS = EDistance.hpp
    5: OBJECTS = EDistance.o
    6: PROGRAM = EDistance test EDistance.a
    7:
    8: .PHONY: all clean lint test
    9:
   10: all:$(PROGRAM)
   11:
   12: %.o: %.cpp $(DEPS)
   13:         $(CC) $(CFLAGS) -c $<
   14: EDistance:EDistance.o main.o
   15:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   16: EDistance.a:EDistance.o
   17:         ar rcs EDistance.a EDistance.o
   18: test:test.o EDistance.o
   19:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   20: clean:
   21:         rm *.o *.a $(PROGRAM)
   22: lint:
   23:         cpplint *.cpp *.hpp
   24:
   25:
```

# 6 PS6: RandWriter

## 6.1 General Discussion

The Markov Model of Natural Language is a statistical method of producing output that mimics any given input. The input text is broken up into “k-grams” k exists in the Z+ set of positive integers. For example, a 2 gram could be ab, ba, ac, ds, etc. Once the text has been broken up into unique k-grams that exist in the input text, k-gram+1 is calculated for each k-gram. K-gram + 1 is the probability that a specific character will follow a k-gram. This is calculated using brute force which can be time complex. The higher the K the more accurate the text seems to be. This can be seen in the Output section. With this method this program can take in large text files such as books and analyze them to output surprisingly accurate imitation of that text. The example that is seen in the Output section inputs the book Tom sawyer and outputs text that looks strikingly similar. Although when a user takes a closer look, it is obvious to realize there is something off with the text. The program is just printing the next most probable character based on the k-gram at any given moment.

## 6.2 Implementation

This project was a bit more complex when it came to OOP and algorithms. I decided to create 2 classes. A RandWriter class which most importantly held private data members _alpha which is an alphabet including all characters in each input. Secondly it held a map<string, Kgram>. The key being a string that contained a kgram and the kgram class holds most importantly kgram+1’s. The second class I created is the Kgram class which is a part of the map included in RandWriter. Kgram holds a count for how many of that specific Kgram exists in the text. It also holds a separate map with a char key and an integer value. The key is essentially kgram+1 and the integer value is the count for that given kgram+1. To move on to the algorithm behind the program, the RandWriter value constructor is implemented to do a lot of the work. The value constructor takes the entire text to be analyzed and the value for k. Using the text, it reads k characters and creates a new point on the map if the kgram doesn’t already exist. It slides over the entire text reading k-grams until it reaches the wrap around case. Using some modular arithmetic it reads kgrams wrapped from the end of the text to the start until the first character is text[0]. During this process it is also incrementing k-gram+1 for

all kgrams as it reads over the text. It also initializes all possible characters into the alphabet data member.

Other functions like krand(), freq(string&), freq(string&, char) work together with the generate() function to output text. Krand takes a kgram as a string and outputs a random kgram+1. It's important to note that while it does return a random kgram + 1 it is statistically skewed towards the more common values. Both the freq() functions return the number of occurrences of kgrams and kgram+1's in the text. These functions together helped to create the generate function which simply takes a valid starter kgram and outputs however many characters are specified. Creating output text that mimics closely the input text.

## 6.3 What Was Learned

This project taught me a lot about some aspects of the standard template library. It was required that we implement exception handling into the program. Not only did I learn how to throw and catch exceptions but also how they can be used to make debugging and testing far easier. I also learned a little bit about unordered sets. Unordered sets have a unique property which was very useful when creating an alphabet for input text. Unordered sets by nature can not have repetition. So I did not have to check for repetition myself when creating the alphabet for an instance of the RandWriter Class. Lastly, I learned about the separate Random library that is specific to C++. I have some experience with the random.h library included in the C standard library.

## 6.4 Output

1000 characters of Output from the book Tom Sawyer with 10-gram

```
bmaillet@cs3:~/Comp4/ps6$ ./TextWriter 10 1000 < tomsawyer.txt
THE ADVENTURES OF TOM SAWYERBY MARK TWAIN(Samuel Langhorne Clemens)PREFACEMost of the school, for on great dead boughs till these words were not to be appeased
 by such thin plausibilities there. As the great day of the true gush, for the young men knew a portion of it. He pictured himself drawing closer and closed wi
th a shaky voice:"Tom, less take and swear that was what she said:"Kiss me again, Tom, it is too horrid," said Becky."It is horrid, but I better, Becky; they m
ight call down again and leave me alone, can't you! I hate you!"So the boy eagerly drew his sore heart at rest. How she would have had at the hang, go it livel
y!"So they "went it lively!"So they "went it lively," panting and perspiring with each other. Presently she stepped forward abreast the island, and the Sheriff
 came through, and each left behind it a slightly lightened weight of the head."Gracie Miller fall in the beetle fell a couple of blue bottle-glass to look to
it that Tom was uneasy, nevertheless. He moped
```

# 1000 characters of Output from the book Tom Sawyer with 20-gram

```
bmaillet@cs3:~/Comp4/ps6$ ./TextWriter 20 1000 < tomsawyer.txt
THE ADVENTURES OF TOM SAWYERBY MARK TWAIN(Samuel Langhorne Clemens)PREFACEMost of the adventures recorded in this book really occurred; one or two were experie
nces of my own, the rest those of boys who were schoolmates of mine. Huck Finn is drawn from life; Tom Sawyer also, but not from an individual-he is a combinat
ion of the characteristics of three boys whom I knew, and therefore belongs to the composite order of architecture.The odd superstitions touched upon were all
prevalent among children and slaves in the West at the period of this story-that is to say, thirty or forty years ago.Although my book is intended mainly for t
he entertainment of boys and girls, I hope it will not be shunned by men and women on that account, for part of my plan has been to try to pleasantly remind ad
ults of what they once were themselves, and of how they felt and thought and talked, and what queer enterprises they sometimes engaged in.THE AUTHOR.HARTFORD,
1876.CHAPTER I"TOM!"No answer."TOM!"No answer.
```

```
 1: // Copyright 2024 Braden Maillet
 2: #include <string>
 3: #include <map>
 4: #include <algorithm>
 5: #include <iterator>
 6: #include <iostream>
 7: #include "RandWriter.hpp"
 8:
 9: int main(int argc, char* argv[]) {
10:   size_t k = static_cast<size_t>(atoi(argv[1]));
11:   int output = atoi(argv[2]);
12:   string kgr;
13:   string text;
14:   string line;
15:   while (std::getline(std::cin, line)) {
16:         text += line;  //  Append the line and add newline character
17:   }
18:   //  std::cout << text << " " << text.size() << std::endl;
19:   RandWriter r(text, k);
20:   for (int i = 0; i < static_cast<int>(k); ++i) {
21:     kgr.push_back(text.at(i));
22:   }
23:   std::cout << r.generate(kgr, output) << std::endl;
24:   return 0;
25: }
```

```
 1: //  CopyRight 2024 Braden Maillet
 2: #pragma once
 3: #include <string>
 4: #include <map>
 5: #include <iostream>
 6: #include <vector>
 7: #include <utility>
 8:
 9: using std::string;
10: using std::map;
11: using std::ostream;
12: using std::vector;
13: using std::pair;
14:
15: //  kgram class for map
16: class kgram {
17:  public:
18:   kgram(string kgram, int count) :
19:     _gram(kgram), _count(count), _kplus1() {}
20:   void pushkplus1(char c);
21:   void increment() {_count++;}
22:
23:   unsigned int checkmap(char c) const;
24:   string getgram() const {return _gram;}
25:   unsigned int getcount() const {return _count;}
26:   unsigned int getcountkplus1(char c) {return _kplus1.at(c);}
27:   map<char, unsigned int>& getmap() {return _kplus1;}
28:  private:
29:   string _gram;
30:   unsigned int _count;
31:   //  vector<pair<char, unsigned int>> _k1;
32:   map<char, unsigned int> _kplus1;
33: };
34:
35: class RandWriter {
36:  public:
37: // Create a Markov model of order k from given text
38: // Assume that text has length at least k.
39:   RandWriter(const string& text, size_t k);
40:   // Order k of Markov model
41:   size_t orderK() const;
42: // Number of occurences of kgram in text
43: // Throw an exception if kgram is not length k
44:   int freq(const string& kgram) const;
45: // Number of times that character c follows kgram
46: // if order=0, return num of times that char c appears
47: // (throw an exception if kgram is not of length k)
48:   int freq(const string& kgram, char c) const;
49: // Random character following given kgram
50: // (throw an exception if kgram is not of length k)
51: // (throw an exception if no such kgram)
52:   char kRand(const string& kgram);
53: // Generate a string of length L characters by simulating a trajectory
54: // through the corresponding Markov chain. The first k characters of
55: // the newly generated string should be the argument kgram.
56: // Throw an exception if kgram is not of length k.
57: // Assume that L is at least k
58:   string generate(const string& kgram, size_t L);
59:   friend ostream& operator<<(ostream& o, const RandWriter& r);
60:
61:  private:
```

```
   62:   int _K;
   63:   string _alpha;
   64:   //  key value
   65:   map<string, kgram> _Kgrams;
   66: };
   67: // Overload the stream insertion operator << and display the internal state
   68: // of the Markov model. Print out the order, alphabet, and the frequencies
   69: // of the k-grams and k+1-grams
```

```
 1: //  Copyright Braden Maillet 2024
 2: #include <string>
 3: #include <map>
 4: #include <algorithm>
 5: #include <iterator>
 6: #include <exception>
 7: #include <random>
 8: #include <unordered_set>
 9: #include "RandWriter.hpp"
10:
11:
12: using std::string;
13:
14: //  might be a special case if 0
15: RandWriter::RandWriter(const string& text, size_t k) {
16:   //  current kgram;
17:   string temp;
18:   this->_K = k;
19:   if (text.size() < k) {
20:     throw std::logic_error("invalid k is larger than text size");
21:   }
22:
23:   for (int i = 0; i < static_cast<int>(text.size()); ++i) {
24:     if (i + k < text.size()) {
25:         for (int j = 0; j < static_cast<int>(k); ++j) {
26:           temp.push_back(text.at(i + j));
27:         }
28:       //  search map if kgram already exists
29:       //  if it does exist, add one to its count
30:       //  if it doesn't push it onto the map with count 0
31:       //  look at next character aswell for kgram + 1
32:       if (_Kgrams.find(temp) == _Kgrams.end()) {
33:         kgram g(temp, 1);
34:         _Kgrams.insert(std::pair<string, kgram>(temp, g));
35:       } else {
36:         _Kgrams.at(temp).increment();
37:       }
38:       //  increment g + 1
39:       if (i + k < static_cast<size_t>(text.size())) {
40:         _Kgrams.at(temp).pushkplus1(text.at(i + k));
41:       } else {
42:         //  shouldn't need this but just incase
43:         _Kgrams.at(temp).pushkplus1(text.at((i + k) % text.size()));
44:       }
45:     //  wrap around case ***********
46:     } else {
47:       for (int x = 0; x < static_cast<int>(k); ++x) {
48:           temp.push_back(text.at(((i + x) % text.size())));
49:       }
50:       if (_Kgrams.find(temp) == _Kgrams.end()) {
51:         kgram g(temp, 1);
52:         _Kgrams.insert(std::pair<string, kgram>(temp, g));
53:       } else {
54:         _Kgrams.at(temp).increment();  //  idk if this works
55:       }
56:       //  increment g + 1 wrap arround
57:       //  dont have to plus one to i + k
58:       //  because of 0 count system
59:       if (i + k < text.size()) {
60:         _Kgrams.at(temp).pushkplus1(text.at(i + k));
61:       } else {
```

```
   62:         //  std::cout << "DEBUG: index" << ((i + k) % text.size()) << std::e
ndl;
   63:         _Kgrams.at(temp).pushkplus1(text.at(((i + k) % text.size())));
   64:       }
   65:     }
   66:     temp.clear();
   67:   }
   68:   //  create alphabet
   69:   //  std::cout << "DEBUG ALPHA " << _alpha << std::endl;
   70:   std::unordered_set<char> alpha;
   71:   for (int z = 0; z < static_cast<int>(text.size()); ++z) {
   72:     alpha.insert(text.at(z));
   73:   }
   74:   for (char c : alpha) {
   75:         _alpha += c;
   76:   }
   77: }
   78: // Number of occurences of kgram in text
   79: // Throw an exception if kgram is not length k
   80: int RandWriter::freq(const string& kgram) const {
   81:   if (kgram.size() != this->orderK()) {
   82:     throw std::logic_error("freq(): kgram is of invalid length");
   83:   }
   84:   try {
   85:     return _Kgrams.at(kgram).getcount();
   86:   }
   87:   catch(std::exception& e) {
   88:     return 0;
   89:   }
   90: }
   91: // Number of times that character c follows kgram
   92: // if order=0, return num of times that char c appears
   93: // (throw an exception if kgram is not of length k)
   94: int RandWriter::freq(const string& kgram, char c) const {
   95:   if (kgram.size() != this->orderK()) {
   96:     throw std::logic_error("freq(): kgram is of invalid length");
   97:   }
   98:   try {
   99:     return _Kgrams.at(kgram).checkmap(c);
  100:   }
  101:   catch(std::exception& e) {
  102:     return 0;
  103:   }
  104: }
  105: // Random character following given kgram
  106: // (throw an exception if kgram is not of length k)
  107: // (throw an exception if no such kgram)
  108: char RandWriter::kRand(const string& kgram) {
  109:   if (kgram.size() != this->orderK() || freq(kgram) == 0) {
  110:     throw std::logic_error("freq(): kgram is of invalid length");
  111:   }
  112:   std::vector<char> chars;
  113:   std::vector<double> prob;
  114:   std::random_device r;
  115:   int temp;
  116:   char selected_character;
  117:   int index;
  118:   for (int i = 0; i < static_cast<int>(_alpha.size()); ++i) {
  119:     temp = freq(kgram, _alpha[i]);
  120:     if (temp) {
  121:       prob.push_back(temp);
```

```
122:       chars.push_back(_alpha[i]);
123:     }
124:   }
125:   std::discrete_distribution<> dist(prob.begin(), prob.end());
126:   std::mt19937 gen(r());
127:   index = dist(gen);
128:   selected_character = chars[index];
129:   return selected_character;
130: }
131: // Generate a string of length L characters by simulating a trajectory
132: // through the corresponding Markov chain. The first k characters of
133: // the newly generated string should be the argument kgram.
134: // Throw an exception if kgram is not of length k.
135: // Assume that L is at least k
136: string RandWriter::generate(const string& kgram, size_t L) {
137:   if (kgram.size() != this->orderK()) {
138:     throw std::logic_error("freq(): kgram is of invalid length");
139:   }
140:   string kg = kgram;
141:   string output;
142:   char temp;
143:   output = output + kg;
144:   for (int i = 0; i < (static_cast<int>(L - kgram.size())); ++i) {
145:     temp = kRand(kg);
146:     kg.erase(kg.begin());
147:     kg.push_back(temp);
148:     output.push_back(temp);
149:   }
150:   return output;
151: }
152: ostream& operator<<(ostream& o, const RandWriter& r) {
153:   //
154:   o << "order: " << r.orderK() << std::endl
155:     << "alphabet: " << r._alpha << std::endl
156:     << "k grams and their k + 1 grams: " << std::endl;
157:
158:   for (auto it = r._Kgrams.begin(); it !=  r._Kgrams.end(); ++it) {
159:     o << "kgram and count: "
160:       << it->second.getgram()
161:       << " "
162:       << it->second.getcount()
163:       << std::endl;
164:       kgram temp = it->second;
165:     for (auto iter = temp.getmap().begin();
166:      iter != temp.getmap().end(); ++iter) {
167:       o << iter->first
168:         << " "
169:         << iter->second
170:         << std::endl;
171:     }
172:   }
173:   return o;
174: }
175: size_t RandWriter::orderK() const {
176:   if (_K < 0) {
177:     throw std::logic_error("invalid k");
178:   }
179:   return _K;
180: }
181:
182: //  kgram class ***************
```

```
183: void kgram::pushkplus1(char c) {
184:   if (_kplus1.find(c) == _kplus1.end()) {
185:     _kplus1.insert(std::pair<char, int>(c, 1));
186:   } else {
187:     _kplus1.at(c)++;
188:   }
189: }
190:
191: unsigned int kgram::checkmap(char c) const {
192:   try {
193:     return _kplus1.at(c);
194:   }
195:   catch(std::exception& e) {
196:     return 0;
197:   }
198:   return -1;
199: }
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <iostream>
 3: #include <fstream>
 4: #include "RandWriter.hpp"
 5:
 6: #define BOOST_TEST_DYN_LINK
 7: #define BOOST_TEST_MODULE Main
 8: #include <boost/test/unit_test.hpp>
 9:
10: BOOST_AUTO_TEST_CASE(testlength) {
11:   RandWriter r("12345", 2);
12:   std::string test = r.generate("12", 6);
13:   BOOST_REQUIRE(test.size() == 6);
14: }
15: BOOST_AUTO_TEST_CASE(testexceptionfreq) {
16:   RandWriter r("12345", 2);
17:   BOOST_REQUIRE_THROW(r.freq("123"), std::logic_error);
18: }
19: BOOST_AUTO_TEST_CASE(testnoexceptionfreq) {
20:   RandWriter r("12345", 2);
21:   BOOST_REQUIRE_NO_THROW(r.freq("12"));
22: }
23: BOOST_AUTO_TEST_CASE(testexceptionfreqc) {
24:   RandWriter r("12345", 2);
25:   BOOST_REQUIRE_THROW(r.freq("123", 1), std::logic_error);
26: }
27: BOOST_AUTO_TEST_CASE(testnoexceptionfreqc1) {
28:   RandWriter r("12345", 2);
29:   BOOST_REQUIRE_NO_THROW(r.freq("12", 1));
30: }
31: BOOST_AUTO_TEST_CASE(testnoexceptionfreqc2) {
32:   RandWriter r("12345", 2);
33:   BOOST_REQUIRE_NO_THROW(r.freq("51", 1));
34: }
35: BOOST_AUTO_TEST_CASE(testordk) {
36:   RandWriter r("12345", 2);
37:   BOOST_REQUIRE(r.orderK() == 2);
38: }
39: BOOST_AUTO_TEST_CASE(testkrandexcept1) {
40:   RandWriter r("12345", 2);
41:   BOOST_REQUIRE_THROW(r.kRand("123"), std::logic_error);
42: }
43: BOOST_AUTO_TEST_CASE(testkrandexcept2) {
44:   RandWriter r("12345", 2);
45:   BOOST_REQUIRE_THROW(r.kRand("14"), std::logic_error);
46: }
47: BOOST_AUTO_TEST_CASE(testkrandnoexcept) {
48:   RandWriter r("12345", 2);
49:   BOOST_REQUIRE_NO_THROW(r.kRand("51"));
50: }
51: BOOST_AUTO_TEST_CASE(teststart1) {
52:   RandWriter r("12345", 2);
53:   std::string test = r.generate("12", 6);
54:   BOOST_REQUIRE(test.at(0) == '1');
55: }
56:
```

```
    1: CC = g++
    2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g -O3
    3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_
test_framework
    4: DEPS = RandWriter.hpp
    5: OBJECTS = RandWriter.o
    6: PROGRAM = TextWriter test TextWriter.a
    7:
    8: .PHONY: all clean lint test
    9:
   10: all:$(PROGRAM)
   11:
   12: %.o: %.cpp $(DEPS)
   13:         $(CC) $(CFLAGS) -c $<
   14: TextWriter:RandWriter.o TextWriter.o
   15:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   16: TextWriter.a:RandWriter.o TextWriter.o
   17:         ar rcs TextWriter.a TextWriter.o RandWriter.o
   18: test:test.o RandWriter.o
   19:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
   20: clean:
   21:         rm *.o *.a $(PROGRAM)
   22: lint:
   23:         cpplint *.cpp *.hpp
   24:
   25:
```

# 7 PS7: Kronos Log Parsing

## 7.1 General Discussion

The Kronos Intouch Timeclock is a touch screen device that runs on Linux. The device creates Intouch logs for regular processes that can help to detect errors and debug the device. In this program I parse data found in these logs to detect boot start and boot completion of the device. I record the time for both and record the duration of boot to a separate text file. The program also detects if the boot process failed. The text "(log.c.166) server started" indicates that a boot process has begun and the text:

"oejs.AbstractConnector:StartedSelectChannelConnector@0.0.0.0:9080"

determines that a boot process has finished successfully.

## 7.2 Implementation

For the implementation of this problem the regex library was heavily used. A loop is used to search through the thousands of lines of data in the log files. For each line the regex_match() is used to search for the start string and end string. Both of which are held as regexes that can hold any amount of anything before or after them. For example, ".*string.*". If a start string is found twice in a row it is determined that a boot process failed, and that data is recorded in the text file. A line count is recorded and held as data in the text file as well.

## 7.3 What Was Learned

I learned a lot about the regex library and its capabilities when searching for specific strings or patterns in text. I also learned about its downfalls. How it can be time-consuming and should be avoided when possible. This was my first experience with the Boost time library. It was very useful in subtracting very precise points in time/dates from one another. This was crucial in figuring out the boot time for all processes.

## 7.4 Output

One section of an output file from a device log:

```
=== Device boot ===
4(logs/device4_intouch.log): 2013-10-02 18:42:38 Boot Start
112(logs/device4_intouch.log): 2013-10-02 18:45:23 Boot Completed
  Boot Time: 165000 ms

=== Device boot ===
747(logs/device4_intouch.log): 2013-10-03 12:23:21 Boot Start
855(logs/device4_intouch.log): 2013-10-03 12:26:15 Boot Completed
  Boot Time: 174000 ms

=== Device boot ===
1459(logs/device4_intouch.log): 2013-10-04 16:20:03 Boot Start
1568(logs/device4_intouch.log): 2013-10-04 16:23:06 Boot Completed
  Boot Time: 183000 ms

=== Device boot ===
31848(logs/device4_intouch.log): 2013-12-03 16:21:13 Boot Start
31956(logs/device4_intouch.log): 2013-12-03 16:24:08 Boot Completed
  Boot Time: 175000 ms

=== Device boot ===
32789(logs/device4_intouch.log): 2013-12-04 21:50:27 Boot Start
33032(logs/device4_intouch.log): 2013-12-04 21:52:57 Boot Completed
  Boot Time: 150000 ms

=== Device boot ===
33145(logs/device4_intouch.log): 2013-12-04 21:58:45 Boot Start
33390(logs/device4_intouch.log): 2013-12-04 22:01:14 Boot Completed
  Boot Time: 149000 ms
```

```
 1: //  Copyright 2024 Braden Maillet
 2: #include <iostream>
 3: #include <fstream>
 4: #include <regex>
 5: #include <string>
 6: #include "boost/date_time/gregorian/gregorian.hpp"
 7: #include "boost/date_time/posix_time/posix_time.hpp"
 8:
 9: using boost::posix_time::ptime;
10: using boost::posix_time::time_duration;
11: using std::string;
12: using std::regex;
13:
14: int main(int argc, char* argv[]) {
15:   string number = argv[1];
16:   string outfilename = "device";
17:   outfilename += number.at(13);  //  13  11
18:   outfilename += "_intouch.log.rpt";
19:   string t("trials/");
20:   t += outfilename;
21:   std::ofstream out(t);
22:   std::ifstream infile(argv[1]);
23:   regex start(".*\\(log\\.c\\.166\\) server started.*");
24:   string x = ".*oejs\\.AbstractConnector:Started "
25:     "SelectChannelConnector@0\\.0\\.0\\.0:9080.*";
26:   regex end(x);
27:   string s;
28:   std::locale loc(std::cout.getloc(),
29:     new boost::posix_time::time_facet("%Y-%m-%d %H:%M:%S"));
30:   out.imbue(loc);
31:   ptime endT;
32:   ptime startT;
33:   time_duration td;
34:   int lineCount = 0;
35:   int lineStart;
36:   int lineEnd;
37:
38:   //  std::cout << "begining: " << std::endl;
39:   while (getline(infile, s)) {
40:     lineCount++;
41:     if (regex_match(s, start)) {
42:       //  std::cout << "DEBUG: " << s << std::endl;
43:       startT = boost::posix_time::time_from_string(s.substr(0, 20));
44:       lineStart = lineCount;
45:       //  std::cout << "DEBUG: " << startT << std::endl;
46:       while (getline(infile, s)) {
47:         lineCount++;
48:         if (regex_match(s, end)) {
49:           endT = boost::posix_time::time_from_string(s.substr(0, 20));
50:           lineEnd = lineCount;
51:           //  std::cout << "END " << endT << std::endl;
52:           //  find time and print to file
53:           //  case that boot succeeded
54:           td = endT - startT;
55:           if (out.is_open()) {
56:             out << "=== Device boot ===\n"
57:                 << lineStart << "(" << argv[1] << "): "
58:                 << startT << " Boot Start" << std::endl
59:                 << lineEnd << "(" << argv[1] << "): "
60:                 << endT << " Boot Completed" << std::endl
61:                 << "  Boot Time: " << td.total_milliseconds()
```

```
62:                 << " ms" << std::endl
63:                 << std::endl;
64:           }
65:           break;
66:         }
67:         if (regex_match(s, start) || infile.eof()) {
68:           //  std::cout << "start " << startT << std::endl;
69:           //  print to file
70:           //  case that boot failed
71:           if (out.is_open()) {
72:             out << "=== Device boot ===\n"
73:                 << lineStart << "(" << argv[1] << "): "
74:                 << startT << " Boot Start" << std::endl
75:                 << "**** Incomplete boot ****" << std::endl
76:                 << std::endl;
77:           }
78:           startT = boost::posix_time::time_from_string(s.substr(0, 20));
79:           lineStart = lineCount;
80:           continue;
81:         }
82:       }
83:     }
84:   }
85:   out.close();
86:   return 0;
87: }
```

```
 1: CC = g++
 2: CFLAGS = --std=c++17 -Wall -Werror -pedantic -g -O3
 3: LIB = -lboost_date_time
 4: DEPS =
 5: OBJECTS = main.o
 6: PROGRAM = ps7
 7:
 8: .PHONY: all clean lint
 9:
10: all:$(PROGRAM)
11:
12: %.o: %.cpp $(DEPS)
13:         $(CC) $(CFLAGS) -c $<
14: ps7:main.o
15:         $(CC) $(CFLAGS) -o $@ $^ $(LIB)
16: clean:
17:         rm *.o *.a $(PROGRAM)
18: lint:
19:         cpplint *.cpp *.hpp
20:
21:
```

# 8 Credits

[1] Kenney. (n.d). Sokoban Pack (CC0). [Online]. Available: https://kenney.nl/assets/sokoban

[2] I. Ozler, “Kode Mono,” Google Fonts, 2024. [Online]. Available: https://fonts.google.com/specimen/Kode+Mono/about

[3] Y. Rykalova, Diagrams included in 1a.1, 2.1, 5.1 2024. Unpublished.