Computing IV

Project Portfolio

Karan Swamy

Spring 2022

Contents

1 PS0 - Hello World with SFML

1.1 Overview

The purpose of this project is to output a sprite with an image of your choice on an SFML window and have it respond to keystrokes. As an additional feature, instead of the sprite moving out of frame it bounces off the edges of the window.

1.2 Key Concepts

1.2.1 Intro To SFML

SFML is a c++ library which provides a simple interface to the various components of your PC, to ease the development of games and multimedia applications. It is composed of five modules: system, window, graphics, audio and network.

1.2.2 Sprites

A sprite is an image or texture that can be displayed on the screen. In this project the sprite moves and responds to keystrokes.

1.2.3 Texture

A texture stores pixels that can be drawn, with a sprite for example. In this project I used a character from a game called Minecraft.

1.2.4 Keyboard

I used the sf::Keyboard class which made my sprite respond to key inputs from the user. My sprite constantly moves. When you hold the Space key, my sprite stops moving, and when you release the Space key my sprite starts moving again.

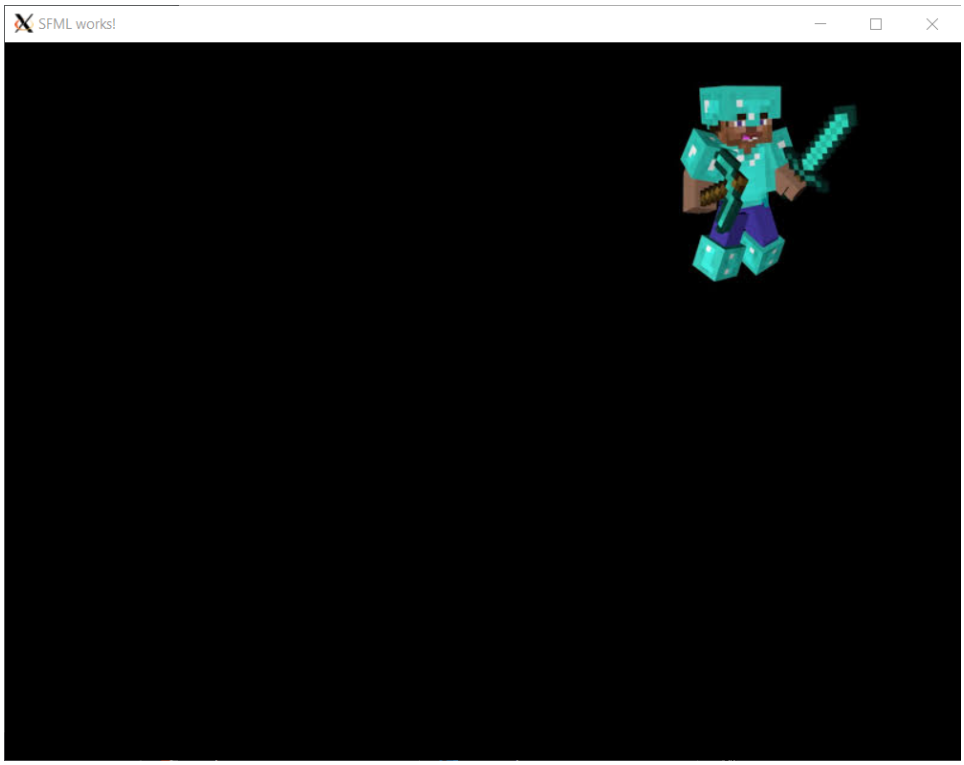
1.3 Learnings

1.3.1 Windows Subsystem for Linux

For this project I downloaded a windows subsystem for linux and the operating system is Ubuntu 20.04.1 which is a linux based operating system. I will be using this operating system for the remainder of the projects in the terminal. Each assignment has its own directory.

1.4 Output

Figure 1: sprite of Minecraft character being displayed and moved in the SFML window.



1.5 Code

1.5.1 main.cpp

```
1 /**
2 *    main.cpp - Implementation of main.cpp
3 *
4 *    Date: 1/24/2022
5 *
6 *    Name: Karan Swamy
7 *
8 *    Course name: Computing 4
```

```
 9 *
10 *   Instructor name: Yelena Rykalova
11 *
12 *   Description: To make a moving sprite which
     responds to a keystroke.
13 *                Pressing Space key will stop the
     animation and releasing Space key will continue
     the animation.
14 *                Additional feature - The sprite
     bounces of the edges of the window rather than
     moving out of the screen.
15 **/
16
17 #include <SFML/Graphics.hpp>
18 #include <iostream>
19 using namespace std;
20
21 int main()
22 {
23     sf::RenderWindow window(sf::VideoMode(800, 600),
          "SFML works!");
24     window.setFramerateLimit(60);
25
26     sf::Texture texture;
27     if(!texture.loadFromFile("sprite.png"))
28     {
29         cout<<"Could not load texture"<<endl;
30         return 0;
31     }
32
33     sf::Sprite sprite;
34     sprite.setTexture(texture);
35
36     sf::Vector2f spritePosition(275,200);
37     sprite.setPosition(spritePosition);
38
39     float xVelocity=5;
40     float yVelocity=5;
41     bool isMoving = true;
42
```

```
43      while (window.isOpen())
44      {
45          sf::Event event;
46          while (window.pollEvent(event))
47          {
48              // check the type of the event...
49              switch (event.type)
50              {
51                  // window closed
52                  case sf::Event::Closed:
53                      window.close();
54                      break;
55
56                  // we don't process other types of
                        events
57                  default:
58                      break;
59              }
60          }
61
62          //toggle the animation if Space key is held
                or released
63          if(sf::Keyboard::isKeyPressed(sf::Keyboard::
                Space))
64          {
65              isMoving = false;
66          }
67          else
68          {
69              isMoving = true;
70          }
71
72          if(isMoving == false)
73              continue;
74
75          //movement of sprite
76          spritePosition.x += xVelocity;
77          spritePosition.y += yVelocity;
78          sprite.setPosition(spritePosition);
79
```

```
80        //bounce off window mechanics
81        if(spritePosition.x<0 || spritePosition.x >
             800-200)
82            xVelocity *= -1;
83
84        if(spritePosition.y<0 || spritePosition.y >
             600-200 )
85            yVelocity *= -1;
86
87         window.clear();
88         window.draw(sprite);
89         window.display();
90     }
91
92     return 0;
93 }
```

2 PS1 - Photo Magic (LFSR Image Encryption)

2.1 Overview

2.1.1 Part A

The first part of this project was to generate pseudo-random bits by simulating a linear feedback shift register that held 16 bits. I used the dynamic bitset boost library to represent the register bits. The dynamic bitset is a library used for bit manipulation. The dynamic bitset class is used to represent a set of bits in either 0(reset) or 1(set) form. The dynamic bitset allocates any required length of bits at runtime. I used this library because this project deals with storing a set of bits and manipulating them(left shifts and xor), which would be more efficient than using a string or a vector. The register has two functions : step() and generate(int k).The step function changes the first bit of the register using the XOR operation on the leftmost bit with tap positions 13,12,10(tap positions are just the indexes of the register). Generate calls the step function k number of times (k is the argument passed to the generate function) and returns the resulting bits.

2.1.2 Part B

Using the LFSR from part A, I took in an input image file and a 16 bit seed, and encrypted the image by XORing the pixels color values with the seed value being passed through the generate function. As a result each pixel value has a different color now which encrypts the image and makes it unrecognizable.

2.2 Key Concepts

2.2.1 Left Fibonacci Shift Register

The register represents a set of 16 bits in either 0(reset) or 1(set) form. The taps are not implemented in the constructor, and they are just evaluated in the step function. The step function XOR's the leftmost bit with the bit at the index 13 and stores that result in a variable called s1. s1 is then XOR'd with the bit at the index 12 and stores that result in s2. s2 is then XOR'd with the bit at the index 10 and stores that result in s3. The bit at the index 0 is then replaced with the value in s3. Generate calls the step function k number of times (k is the argument passed to the generate function) and returns the resulting bits. For the generate function I used the equation

: num = (2 * num) + step(). Once the step() is called k times (k is the argument passed to generate), the resulting num is then returned.

2.2.2 Boost Tests

Boost Tests are a great way to create unit tests for your functions. The first test I added checks whether the LFSR works properly with generate(8) and generate(9). The second test I added checks whether the constructor properly initialises the member variable bitset which is a dynamic bitset. The third test checks whether the data in particular bit postions can be accessed and the value in that bit position is correct according to the string passed to the constructor.

2.2.3 Transform Function

I implemented a function called transform that takes the initial input image(sf::Image) and XOR's all of its pixel color variables(red,green,blue) with the bitstring key passed in. These pixel color variables are then each XOR'd with the bitstring after it uses generate(8). This will assign the colors random values which helps transform the image. I used nested for loops to go through all the x and y values of the pixels of the image. After the XOR operation, the new color is saved to that pixel.

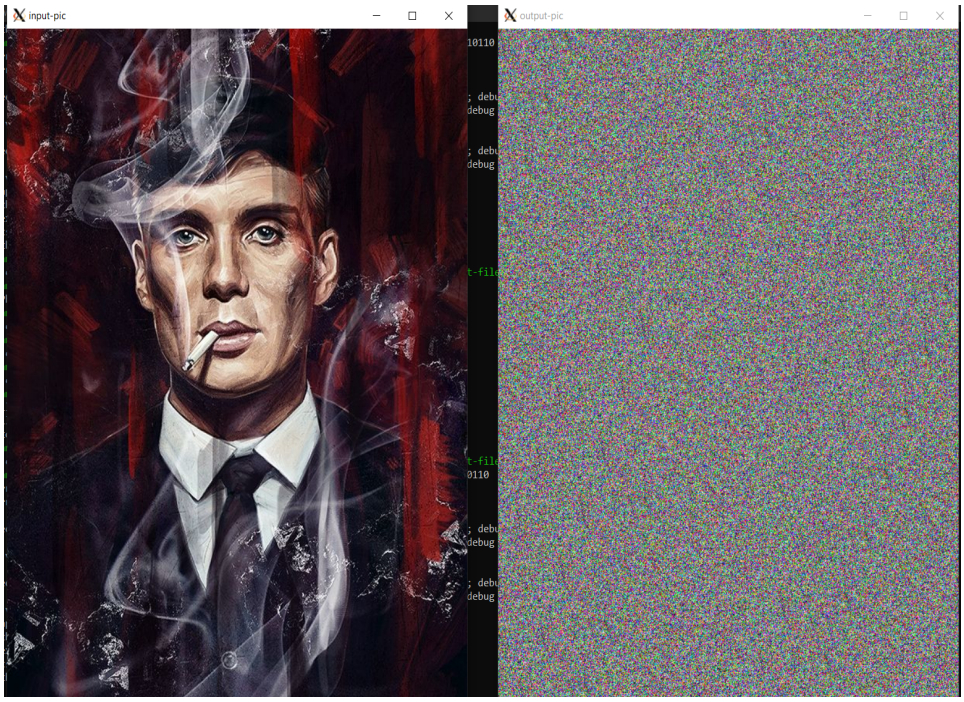
2.3 Learnings

2.3.1 Pixels

Through this project I learned that each image is made up of pixels and for each pixel, the color is determined by mixing red, blue, and green. Each color can be set to a number between 0-255, where 0 means the color is not at all present and 250 means the color is at its maximum intensity.

2.4 Output

Figure 2: Original image on the left and the encrypted image on the right

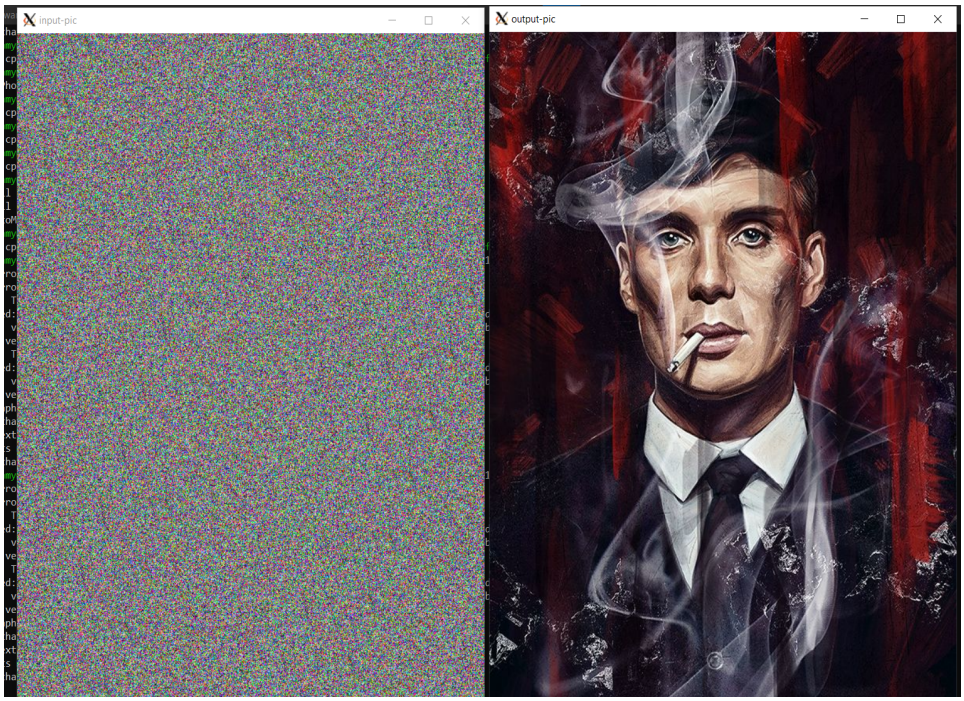


2.5 Code

2.5.1 Makefile

```
1 all: PhotoMagic
2
3 PhotoMagic: FibLFSR.o PhotoMagic.o
4     g++ PhotoMagic.o FibLFSR.o -o PhotoMagic -lsfml-
          graphics -lsfml-window -lsfml-system
5
6 PhotoMagic.o: FibLFSR.cpp FibLFSR.h PhotoMagic.cpp
7     g++ -Wall -Werror -pedantic -c PhotoMagic.cpp
          FibLFSR.cpp
8
9 FibLFSR.o: FibLFSR.cpp FibLFSR.h
10    g++ -Wall -Werror -pedantic -c FibLFSR.cpp
11
12 clean:
```

Figure 3: Encrypted image on the left and the decrypted original image on the right




```
13      rm *.o PhotoMagic
```

2.5.2 PhotoMagic.cpp

```
 1 #include "FibLFSR.h"
 2 #include "SFML/Graphics.hpp"
 3
 4 void transform( sf::Image& image, FibLFSR* code)
 5 {
 6     sf::Color p;
 7     sf::Vector2u size = image.getSize();
          //size of image
 8
 9     for (int x = 0; x< (int)size.x; x++)
10     {
11         for (int y = 0; y< (int)size.y; y++)
12         {
```

```
13              p = image.getPixel(x,y);
14              p.r = p.r ^ code->generate(8);
                   //encoding the red component of the
                   pixel
15              p.g = p.g ^ code->generate(8);
                   //encoding the green component of the
                    pixel
16              p.b = p.b ^ code->generate(8);
                   //encoding the blue component of the
                   pixel
17              image.setPixel(x, y, p);
                   //setting the pixel with the new
                   encoded components
18          }
19      }
20 }
21
22 int main(int argc, char* argv[])
23 {
24
25      sf::Image imageIn;
26      if (!imageIn.loadFromFile(argv[1]))
27          return -1;
28
29      sf::Image imageTemp;                    //stores
            the input image
30      if (!imageTemp.loadFromFile(argv[1]))
31          return -1;
32
33      FibLFSR code(argv[3]);
34      transform(imageIn, &code);              //
           transforms the input image
35
36      sf::Texture textureIn;                  //input
           texture
37      sf::Texture textureOut;                 //output
            texture
38
39      if (!imageIn.saveToFile(argv[2]))       //saves
           transformed image to output file
```

```
40          return -1;
41
42      sf::Image imageOut;
43      if (!imageOut.loadFromFile(argv[2]))
44          return -1;
45
46      sf::Vector2u size = imageIn.getSize();  //size
           of image
47      sf::RenderWindow window1(sf::VideoMode(size.x,
           size.y), "input-pic");   //input window
48      sf::RenderWindow window2(sf::VideoMode(size.x,
           size.y), "output-pic");  //output window
49
50      textureIn.loadFromImage(imageTemp);
51      textureOut.loadFromImage(imageOut);
52
53      sf::Sprite spriteIn;
54      sf::Sprite spriteOut;
55      spriteIn.setTexture(textureIn);
           //sprite of input image
56      spriteOut.setTexture(textureOut);
           //sprite of output image
57
58      while (window1.isOpen() && window2.isOpen())

59      {
60          sf::Event event;
61          while (window1.pollEvent(event))
               //event loop for window1
62          {
63              if (event.type == sf::Event::Closed)
64              window1.close();
65          }
66          while (window2.pollEvent(event))
               //event loop for window2
67          {
68              if (event.type == sf::Event::Closed)
69              window2.close();
70          }
71          window1.clear();
```

```
72          window1.draw(spriteIn);
               //draw input image
73          window1.display();
74          window2.clear();
75          window2.draw(spriteOut);
               //draw output image
76          window2.display();
77      }
78 }
```

2.5.3 FibLFSR.h

```
 1 #ifndef FIBLFSR_H
 2 #define FIBLFSR_H
 3
 4 #include <boost/dynamic_bitset.hpp>
 5 #include <iostream>
 6 #include <string>
 7
 8 using namespace std;
 9 using namespace boost;
10
11 class FibLFSR
12 {
13 public:
14     FibLFSR(string seed); // constructor to create
          LFSR with the given initial seed
15     int step(); // simulate one step and return the
          new bit as 0 or 1
16     int generate(int k); // simulate k steps and
          return k-bit integer
17     dynamic_bitset<> getSeed() const { return bitSet
          ; };    //returns the current register value
18     int getSize() const {return bitSet.size();};
                    //returns the size of the bitSet
19     friend ostream& operator<< (ostream& out, const
          FibLFSR& seed);     //prints current register
           value
20
21 private:
```

```
22      dynamic_bitset<> bitSet;        //register value
            is stored in bitSet
23      const int TAP_1 = 13;           //tap positions
24      const int TAP_2 = 12;
25      const int TAP_3 = 10;
26  };
27
28  #endif
```

2.5.4 FibLFSR.cpp

```
 1  #include "FibLFSR.h"
 2
 3  FibLFSR::FibLFSR(string seed)
 4  {
 5      if(seed.size() == 16)
 6          bitSet = dynamic_bitset<>(seed);
 7      else
 8          throw string("wrong bit length");
 9  }
10
11  int FibLFSR::step()
12  {
13      int leftMostBit;
14
15      leftMostBit = bitSet[bitSet.size()-1];
16      int s1, s2, s3;
17
18      s1 = leftMostBit ^ bitSet[TAP_1];       //
            leftmost bit xor'd with the bit at tap
            position 13 and stored in s1
19      s2 = s1 ^ bitSet[TAP_2];                // s1
            xor'd with the bit at tap position 12 and
            stored in s2
20      s3 = s2 ^ bitSet[TAP_3];                // s2
            xor'd with the bit at tap position 10 and
            stored in s3
21      bitSet = bitSet << 1;                   //left
            shit bitSet
22      bitSet[0] = s3;
```

```
23
24      return bitSet[0];
25  }
26
27  int FibLFSR::generate(int k)
28  {
29      int num = 0;
30
31      for (int i = 0; i < k; i++)
32      {
33          num = (num * 2) + step();
34      }
35      return num;
36  }
37
38  ostream& operator<< (ostream& out, const FibLFSR& in
        )
39  {
40      out << in.getSeed();
41      return out;
42  }
```

2.5.5 test.cpp

```
1  // Dr. Rykalova
2  // test.cpp for PS1a
3  // updated 1/31/2020
4
5  #include <iostream>
6  #include <string>
7
8  #include "FibLFSR.h"
9
10 #define BOOST_TEST_DYN_LINK
11 #define BOOST_TEST_MODULE Main
12 #include <boost/test/unit_test.hpp>
13
14 BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps)
15 {
16      FibLFSR l("1011011000110110");
```

```
17      BOOST_REQUIRE(l.step() == 0);
18      BOOST_REQUIRE(l.step() == 0);
19      BOOST_REQUIRE(l.step() == 0);
20      BOOST_REQUIRE(l.step() == 1);
21      BOOST_REQUIRE(l.step() == 1);
22      BOOST_REQUIRE(l.step() == 0);
23      BOOST_REQUIRE(l.step() == 0);
24      BOOST_REQUIRE(l.step() == 1);
25
26      FibLFSR l2("1011011000110110");
27      BOOST_REQUIRE(l2.generate(9) == 51);
28  }
29
30  //checks whether the contructor properly initialises
        the member variable bitSet.
31  BOOST_AUTO_TEST_CASE(testConstructor)
32  {
33      string buffer;
34      FibLFSR l2("1011011000110110");
35      to_string(l2.getSeed(), buffer);
36      BOOST_CHECK_EQUAL(buffer, "1011011000110110");
37      BOOST_CHECK_EQUAL(l2.getSize(), std::strlen("
            1011011000110110"));
38
39  }
40
41  //checks whether the data in particular bit postions
        can be accessed and the value in that bit
       position is correct according to the string
       passed to the constructor.
42  BOOST_AUTO_TEST_CASE(dataAccessTest)
43  {
44      FibLFSR l1("1011011000110110");
45      BOOST_CHECK_EQUAL(l1.getSeed()[0],0);
46      BOOST_CHECK_EQUAL(l1.getSeed()[l1.getSize()
            -1],1);
47  }
```

3 PS2 - N-Body Simulation

3.1 Overview

3.1.1 Part A

For part A, there are two classes namely "Universe" and "CelestialBody". The "CelestialBody" class creates a celestial body (an instance of the CelestialBody class) with data from a text file. In the text file each line has data from left to right in the following order - xposition, yposition, xvelocity, yvelocity, mass, image-file-name. These are all attributes of the CelestialBody class. The "Universe" class then creates multiple CelestialBody objects and then stores smart pointers to these instances of the CelestialBody objects in a vector. I used a `shared_ptr`(smart pointer) of type CelestialBody. I used overriding which overrides the sf::Drawable virtual function draw so that when you call window.draw(CelestialBody) it draws the sprite of the instance of the celestial body.

3.1.2 Part B

Using the classes implemented in part A, I created a simulation of a universe with a sun and 4 other planets revolving around the sun. Using Newtons laws of motion and gravitation, I implemented a step function that calculates the net force of each planet(celestial body) in relation to others and moved them in orbit accordingly. The speed of revolutions vary with the value of delta t passed through the command line.

3.2 Key Concepts

3.2.1 Abstract base class and Overriding

Sf::drawable is an abstract base class for objects that can be drawn to a render target. It allows objects of derived classes to be drawn to a sf::RenderTarget. All you have to do in your derived class is to override the draw virtual function. In CelestialBody.cpp, the lines 28-31 show the implementation of the overridden virtual function draw.

3.2.2 Overloading the stream input Operator(»)

» reads in values from a given file which contains the universe radius and the attributes of each celestial body. It instantiates a new CelestialBody object with the values read in after each line.

3.2.3 Universe Step Function

With the use of accessor and mutator functions I used the data of each of the planets (celestial body) to calculate the net force,acceleration and velocity of these planets which then calculate the new position of the planet. I go through the vector of pointers to CelestialBody objects and call the updatePosition() function which updates the position of the particular instance of the CelestialBody object. Since the universe needs to fit in the sfml window, I scaled the x and y positions of each planet and then updated the sprite for each planet according to the new scaled coordinates.

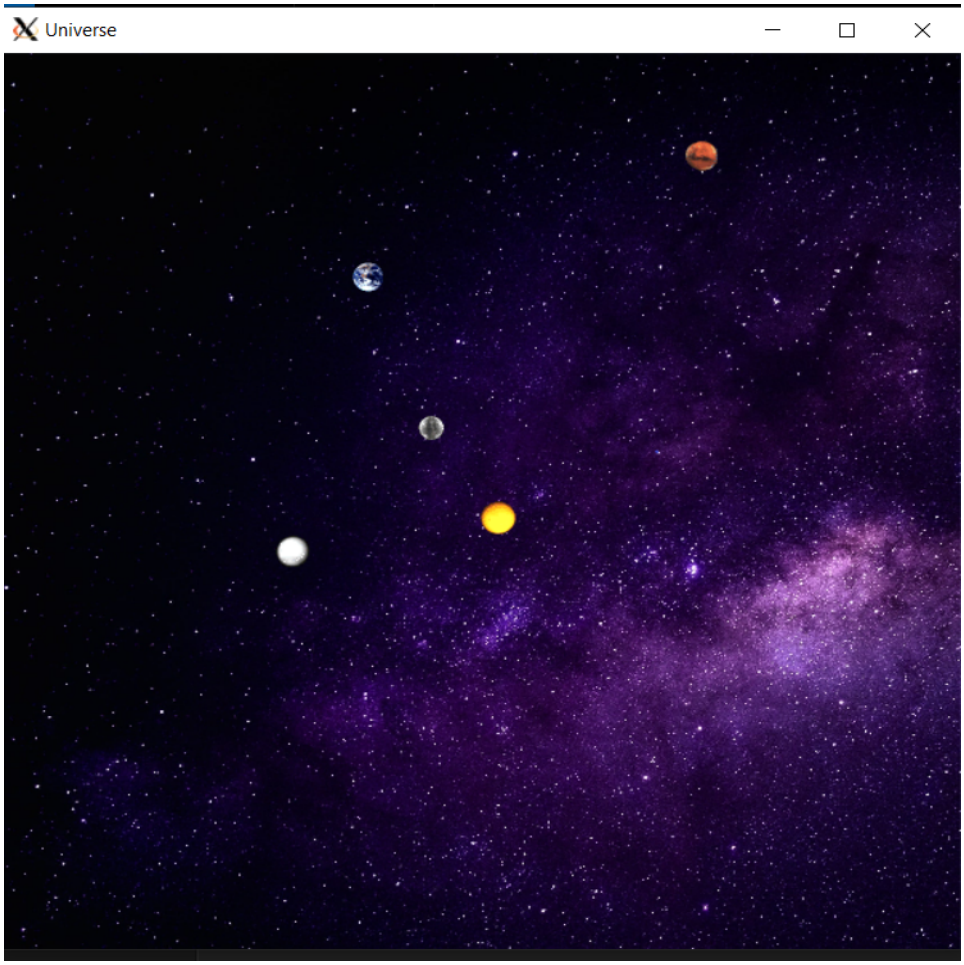
3.3 Learnings

3.3.1 Smart pointers

A Smart Pointer is a wrapper class over a pointer with an operator like * and -> overloaded. The objects of the smart pointer class look like normal pointers. But, unlike Normal Pointers it can deallocate and free destroyed object memory. There are three types of smart pointers namely,`unique_ptr`, `shared_ptr` and `weak_ptr`. For this project I used shared ptr to point to a CelestialBody object in which more than one pointer can point to this one object at a time and it'll maintain a Reference Counter using `use_count()` method.

3.4 Output

Figure 4: Output with planet.txt where 4 planets rotate around the sun.



3.5 Code

3.5.1 planets.txt (Sample Input File)

```
1 5
2 2.50e+11
3  1.4960e+11   0.0000e+00   0.0000e+00   2.9800e+04
      5.9740e+24     earth.gif
```

```
4  2.2790e+11   0.0000e+00   0.0000e+00   2.4100e+04
     6.4190e+23      mars.gif
5  5.7900e+10   0.0000e+00   0.0000e+00   4.7900e+04
     3.3020e+23   mercury.gif
6  0.0000e+00   0.0000e+00   0.0000e+00   0.0000e+00
     1.9890e+30       sun.gif
7  1.0820e+11   0.0000e+00   0.0000e+00   3.5000e+04
     4.8690e+24     venus.gif
8
9 This file contains the sun and the inner 4 planets
    of our Solar System.
```

3.5.2 Makefile

```
1 all: NBody
2
3 NBody: CelestialBody.o Universe.o main.o
4     g++ CelestialBody.o Universe.o main.o -o NBody -
          lsfml-graphics -lsfml-window -lsfml-system
5
6 CelestialBody.o: CelestialBody.cpp CelestialBody.h
    Universe.h
7     g++ -Wall -Werror -pedantic -c CelestialBody.cpp
8
9 Universe.o: CelestialBody.cpp CelestialBody.h
    Universe.h
10     g++ -Wall -Werror -pedantic -c Universe.cpp
11
12 main.o: CelestialBody.cpp CelestialBody.h Universe.h
13     g++ -Wall -Werror -pedantic -c main.cpp
14
15 clean:
16     rm *.o NBody
```

3.5.3 main.cpp

```
1 #include "Universe.h"
2 #include <cmath>
3
4 int main(int argc, char* argv[])
5 {
```

```
6     Universe u;
7     cin >> u;
8     vector<shared_ptr<CelestialBody>> bodyVector;
9     bodyVector = u.getCelestialBodies();
10
11    double dt = strtod(argv[2],NULL);
12    double T = strtod(argv[1], NULL);
13
14    sf::RenderWindow window(sf::VideoMode(
          WINDOW_WIDTH, WINDOW_HEIGHT), "Universe");
15    while (window.isOpen())
16    {
17        sf::Event event;
18        while (window.pollEvent(event))
19        {
20            if (event.type == sf::Event::Closed)
21                window.close();
22        }
23
24        window.clear(sf::Color::Black);
25
26        for (double j = 0.0; j< T; j+= dt)
27        {
28            u.draw_background(window);
29            u.step(dt);
30
31            for (int i = 0; i < (int)bodyVector.size
                  (); i++)
32            {
33                window.draw(*(bodyVector[i]));
34            }
35
36            window.display();
37
38        }
39        break;
40    }
41
42    for (int i = 0; i < (int)bodyVector.size(); i++)
43    {
```

```
44          cout << setw(13)<< bodyVector[i]->
                getXandYPos().x
45                << setw(13)<< bodyVector[i]->
                     getXandYPos().y
46                << setw(13)<< bodyVector[i]->
                     getXandYVel().x << "  "
47                << setw(13)<< bodyVector[i]->
                     getXandYVel().y << "  "
48                << setw(13)<< bodyVector[i]->
                     getPlanetMass() << "  "
49                << setw(13)<< bodyVector[i]->
                     getFileName() << endl;
50      }
51
52      return 0;
53  }
```

3.5.4 CelestialBody.h

```
 1  #ifndef CELESTIALBODY_H
 2  #define CELESTIALBODY_H
 3  #include <iostream>
 4  #include <string>
 5  #include <iomanip>
 6  #include <cmath>
 7  #include "SFML/Graphics.hpp"
 8  #define WINDOW_WIDTH 600
 9  #define WINDOW_HEIGHT 600
10
11  using namespace std;
12
13  class CelestialBody : public sf::Drawable
14  {
15  private:
16      double xPosition;
17      double yPosition;
18      double planetMass;
19      double xVelocity;
20      double yVelocity;
21      string fileName;
```

```
22      sf::Texture texture;
23      sf::Sprite planet;
24
25  public:
26      //default constructor
27      CelestialBody() {};
28      //constructor
29      CelestialBody(double xPos, double yPos, double
            xVel, double yVel, double mass, double
            univRadius, string imageName);
30      //overloaded >> operator to read in universe
            file
31      friend istream& operator>>(istream& input,
            CelestialBody& cBody);
32      void setVelocity(double xPos, double yPos) {
            xVelocity = xPos; yVelocity = yPos; }
33      sf::Vector2<double> getXandYPos() const { return
             sf::Vector2<double>(xPosition, yPosition); }
34      sf::Vector2<double> getXandYVel() const{ return
            sf::Vector2<double>(xVelocity, yVelocity); }
35      double getPlanetMass() { return planetMass; }
36      string getFileName() { return fileName; }
37      void updateVelocity(double deltaT);
38      void setXandYpos(double x, double y) { xPosition
             = x, yPosition = y; }
39      void spriteSetPosition(double xPos, double yPos)
            ;
40
41  private:
42      //draws a CelestialBody object
43      virtual void draw(sf::RenderTarget& target, sf::
            RenderStates states) const;
44
45  };
46  #endif
```

3.5.5 CelestialBody.cpp

```
1  #include "CelestialBody.h"
2
```

```
 3 CelestialBody::CelestialBody(double xPos, double
       yPos, double xVel, double yVel, double mass,
       double univRadius, string imageName)
 4 {
 5     fileName = imageName;
 6     xPosition = xPos;
 7     yPosition = yPos;
 8     xVelocity = xVel;
 9     yVelocity = yVel;
10     planetMass = mass;
11
12     sf::Image image;
13     if (!image.loadFromFile(fileName))
14         cout << "no image" << endl;
15
16     texture.loadFromImage(image);
17     planet.setTexture(texture);
18
19     double scaledXPOS;
20     double scaledYPOS;
21
22     scaledXPOS = (xPos / univRadius) * (WINDOW_WIDTH
           / 2) + (WINDOW_WIDTH / 2);
23     scaledYPOS = (yPos / univRadius) * (
          WINDOW_HEIGHT / 2) + (WINDOW_HEIGHT / 2);
24     planet.setPosition(scaledXPOS, scaledYPOS);
25
26 }
27
28 void CelestialBody::draw(sf::RenderTarget& target,
       sf::RenderStates states) const
29 {
30     target.draw(planet, states);
31 }
32
33 void CelestialBody::spriteSetPosition(double xPos,
       double yPos)
34 {
35     planet.setPosition(double(xPos), double(yPos));
36 }
```

3.5.6 Universe.h

```
1 #ifndef UNIVERSE_H
2 #define UNIVERSE_H
3 #include "CelestialBody.h"
4 #include <vector>
5 #include <memory>
6
7 class Universe
8 {
9 private:
10     vector<shared_ptr<CelestialBody>>
          celestialBodies;
11     int numPlanets;
12     double universeRadius;
13     sf::Vector2<double> calcForceOnPlanet(double
          mass1, double mass2, sf::Vector2<double>
          planetPos1, sf::Vector2<double> planetPos2);
14     sf::Vector2<double> calcNetForce(vector <sf::
          Vector2<double>> forces);
15     sf::Vector2<double> calcAcceleration(sf::Vector2
          <double> netForces, double mass1);
16     sf::Vector2<double> calcNewVelocity(double
          deltaT, sf::Vector2<double> acceleration, sf
          ::Vector2<double> oldVelocity);
17     sf::Vector2<double> calcNewPosition(double
          deltaT, sf::Vector2<double> newVel, sf::
          Vector2<double> oldPos);
18
19 public:
20     //default constructor
21     Universe() {};
22     //Reads input from txt file
23     friend istream& operator>>(istream& input,
          Universe& cBody);
24     //accessor function
25     vector<shared_ptr<CelestialBody>>
          getCelestialBodies() { return celestialBodies
          ; }
26     //draws the background
```

```
27      void draw_background(sf::RenderWindow& win);
28      sf::Vector2<double> updatePosition(double deltaT
           , vector<shared_ptr<CelestialBody>> cBodies,
           int planetNum);
29      //moves a CelestialBody object given its
           internal velocity for the amount of seconds
           passed as an argument
30      void step(double seconds);
31
32 };
33 #endif
```

3.5.7 Universe.cpp

```
 1 #include "Universe.h"
 2
 3 istream& operator>>(istream& input, Universe&
      myUniverse)
 4 {
 5      input >> myUniverse.numPlanets >> myUniverse.
           universeRadius;
 6      double xPos;
 7      double yPos;
 8      double xVel;
 9      double yVel;
10      double mass;
11      string image;
12
13      for (int i = 0; i < myUniverse.numPlanets; i++)
14      {
15          input >> xPos >> yPos >> xVel >> yVel >>
               mass >> image;
16          myUniverse.celestialBodies.push_back(
               make_shared<CelestialBody>(xPos, yPos,
               xVel, yVel, mass, myUniverse.
               universeRadius, image));
17      }
18
19      return input;
20 }
```

```
21
22 void Universe::draw_background(sf::RenderWindow& win
       )
23 {
24     sf::Texture bgTexture;
25     if (!bgTexture.loadFromFile("nebulaBackground.
           png"))
26         cout << "no image" << endl;
27
28     sf::Sprite bSprite;
29     bSprite.setTexture(bgTexture);
30     bSprite.scale(sf::Vector2f(0.35, 0.35));
31
32     win.draw(bSprite);
33 }
34
35 sf::Vector2<double> Universe::calcForceOnPlanet(
       double mass1, double mass2, sf::Vector2<double>
       planetPos1, sf::Vector2<double> planetPos2)
36 {
37     double G = 6.67e-11;
38     double force;
39     double forceX;
40     double forceY;
41     double deltaX;
42     double deltaY;
43     double distBetweenBodies;
44     double deltaSquareSum;
45
46     deltaX = planetPos2.x - planetPos1.x;
47     deltaY = planetPos2.y - planetPos1.y;
48     deltaSquareSum = (deltaX*deltaX) + (deltaY*
           deltaY);
49     distBetweenBodies = sqrt(deltaSquareSum);
50     force = (G * mass1 * mass2) / deltaSquareSum;
51     forceX = (force * deltaX) / distBetweenBodies;
52     forceY = (force * deltaY) / distBetweenBodies;
53
54     return sf::Vector2<double>(forceX, forceY);
55 }
```

```
56
57 sf::Vector2<double> Universe::calcNetForce(vector <
       sf::Vector2<double>> forces)
58 {
59     double netForceX = 0;
60     double netForceY = 0;
61     for (int i = 0; i < int(forces.size()); i++)
62     {
63         netForceX += forces[i].x;
64         netForceY += forces[i].y;
65     }
66
67     return sf::Vector2<double>(netForceX, netForceY)
           ;
68 }
69
70 sf::Vector2<double> Universe::calcAcceleration(sf::
       Vector2<double> netForces, double mass1)
71 {
72     double accX;
73     double accY;
74
75     accX = netForces.x / mass1;
76     accY = netForces.y / mass1;
77
78     return sf::Vector2<double>(accX, accY);
79 }
80
81 sf::Vector2<double> Universe::calcNewVelocity(double
       deltaT, sf::Vector2<double> acceleration, sf::
       Vector2<double> oldVelocity)
82 {
83     double newVelX;
84     double newVelY;
85
86     newVelX = (deltaT * acceleration.x) +
           oldVelocity.x;
87     newVelY = (deltaT * acceleration.y) +
           oldVelocity.y;
88
```

```
89      return sf::Vector2<double>(newVelX, newVelY);
90  }
91
92  sf::Vector2<double> Universe::calcNewPosition(double
        deltaT, sf::Vector2<double> newVel, sf::Vector2<
       double> oldPos)
93  {
94      sf::Vector2<double> newPos;
95
96      newPos.x = oldPos.x + (deltaT * newVel.x);
97      newPos.y = oldPos.y + (deltaT * newVel.y);
98
99      return newPos;
100 }
101
102 sf::Vector2<double> Universe::updatePosition(double
       deltaT, vector<shared_ptr<CelestialBody>> cBodies
       ,int planetNum)
103 {
104     vector <sf::Vector2<double>> pairWiseForces;
105     sf::Vector2<double> NetForce;
106     sf::Vector2<double> acceleration;
107
108      for (int i = 0; i < (int)cBodies.size(); i++)
109      {
110          if (i == planetNum)
111              continue;
112          pairWiseForces.push_back(calcForceOnPlanet(
                cBodies[planetNum]->getPlanetMass(),
                cBodies[i]->getPlanetMass(), cBodies[
                planetNum]->getXandYPos(), cBodies[i]->
                getXandYPos()));
113      }
114
115      sf::Vector2<double> oldVel = cBodies[planetNum
           ]->getXandYVel();
116      sf::Vector2<double> newVel;
117      sf::Vector2<double> oldPos = cBodies[planetNum
           ]->getXandYPos();
118      sf::Vector2<double> newPos;
```

```
119
120     NetForce = calcNetForce(pairWiseForces);
121     acceleration = calcAcceleration(NetForce,
           cBodies[planetNum]->getPlanetMass());
122     newVel = calcNewVelocity(deltaT, acceleration,
           oldVel);
123     celestialBodies[planetNum]->setVelocity(newVel.x
           , newVel.y);
124     newPos = calcNewPosition(deltaT, newVel, oldPos)
           ;
125
126     return newPos;
127 }
128
129 void Universe::step(double seconds)
130 {
131     vector <sf::Vector2<double>> newPositions;
132     for (int i = 0; i < (int)celestialBodies.size();
           i++)
133     {
134         newPositions.push_back(updatePosition(seconds
              , celestialBodies, i));
135     }
136
137     double scaledXPOS;
138     double scaledYPOS;
139
140     for (int i=0; i < (int)newPositions.size(); i++)
141     {
142         celestialBodies[i]->setXandYpos(newPositions
              [i].x, newPositions[i].y);
143         scaledXPOS = ((double)newPositions[i].x /
              universeRadius) * (WINDOW_WIDTH / 2) + (
              WINDOW_WIDTH / 2);
144         scaledYPOS = -((double)newPositions[i].y /
              universeRadius) * (WINDOW_WIDTH / 2) + (
              WINDOW_WIDTH / 2);
145         celestialBodies[i]->spriteSetPosition(
              scaledXPOS, scaledYPOS);
146     }
```

```
147 }
```

4 PS3 - Recursive Graphics (Sierpinski Triangle)

4.1 Overview

In this project I drew a Triangle fractal based on Sierpinski's triangle to the SFML window which used a recursive function to build the triangles. I used sf::VertexArray which is an SFML class to draw the triangles (equilateral triangles) and store them. After each recursive call until the recursive depth passed through the command line is reached, a new set of triangles are created and stored in a vector of vertex arrays. The triangles are then drawn to the SFML window by iterating over the vector of vertex arrays. I did the math to figure out the centre of the new triangles being formed. The length of the sides of the new triangles created are half the length of the triangles created in the previous recursion. For extra credit : I added colors to the triangles by setting each element of the vertex array with sf::color. In addition to color, every time you press the left arrow key, the recursion depth of the fractal triangle is reduced by 1 and is displayed. Left arrow key can be pressed until only the base triangle remains on the screen.

4.2 Key Concepts

4.2.1 Recursion

First the program creates an equilateral base triangle of length L which is passed through the command line. Then the parent triangles of length L/2 are created at the three vertices of the base triangle. The child triangles are then are created by recursive calls on the function ftree(). All these created triangles are then stored in a vector of Vertex arrays.

4.2.2 Math to Calculate Distance

I used trigonometric formulas and general triangle math formulas to find the centres of the new triangles that needed to be created in the next recursion step.

4.3 Learnings

4.3.1 VertexArray

sf::VertexArray is a very simple wrapper around a dynamic array of vertices and a primitives type. The most basic 2D primitives are point, line, triangle and quad (internally the graphics card breaks it into two triangles). It

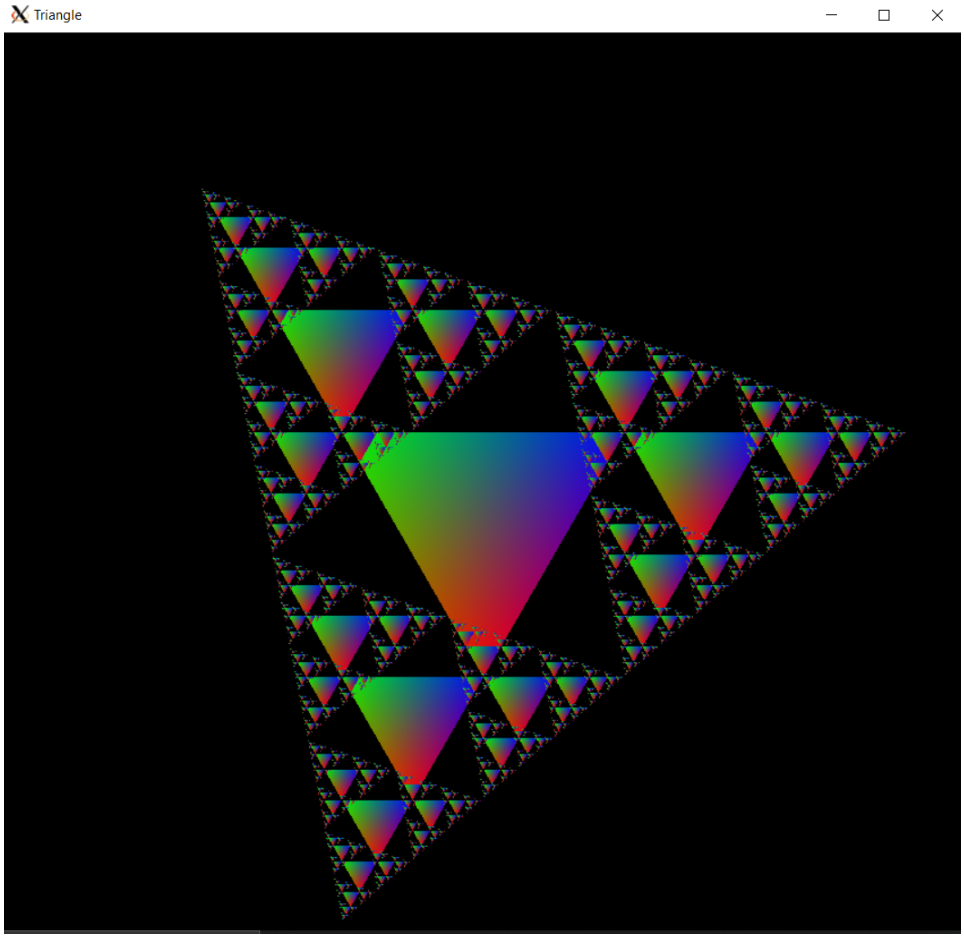
inherits sf::Drawable, but unlike other drawables it is not transformable. In this project I used the vertex array to store the triangles created.

4.3.2 Introduction to Cpplint

Cpplint is a command-line tool to check C/C++ files for style issues following Google's C++ style guide. Cpplint is developed and maintained by Google Inc. Cpplint ensures my code is correctly formatted and professional coding standards are followed. From this project onwards, all the projects pass the Cpplint standard.

4.4 Output

Figure 5: Colored triangles making up the Sierpinski Triangle



4.5 Code

4.5.1 Makefile

```
1 all: Triangle
2
3 Triangle: Triangle.o TFractal.o
```

```
4     g++ Triangle.o TFractal.o -o Triangle -lsfml-
         graphics -lsfml-window -lsfml-system
5
6 Triangle.o: TFractal.cpp Triangle.h
7     g++ -Wall -Werror -pedantic -c Triangle.cpp
8
9 TFractal.o: Triangle.cpp Triangle.h
10    g++ -Wall -Werror -pedantic -c TFractal.cpp
11
12 clean:
13    rm *.o Triangle
```

4.5.2 TFractal.cpp

```
1 // Copyright [2022] <Karan Swamy>
2
3 #include "Triangle.h"
4 #define WINDOW_WIDTH 850
5 #define WINDOW_HEIGHT 850
6
7 void fTree(int N, double L, sf::Vector2f center,
8            std::vector <sf::VertexArray> *v) {
9     double height = (sqrt(3) / 2) * L;
10    sf::VertexArray tri(sf::Triangles, 3);
11
12    // build the base triangle
13    if (N == 0) {
14        tri[0].position = sf::Vector2f(WINDOW_WIDTH
             /2.0,
15                                       WINDOW_HEIGHT
                                            /2 + (L/
                                            sqrt(3)));
16        tri[1].position = sf::Vector2f(WINDOW_WIDTH
             / 2.0 +(L/2.0),
17                                       WINDOW_HEIGHT
                                             / 2 - (L
                                            /(2.0*sqrt
                                            (3))));
18        tri[2].position = sf::Vector2f(WINDOW_WIDTH
             / 2.0 - (L / 2.0),
```

```
19          WINDOW_HEIGHT - (L / (2.0 * sqrt(3))));
20          tri[0].color = sf::Color::Red;
21          tri[1].color = sf::Color::Blue;
22          tri[2].color = sf::Color::Green;
23      }
24
25      // build parent triangle
26      // bottom vertex
27      tri[0].position = sf::Vector2f(center.x, center.
            y + ((2.0 / 3) * height));
28      // top right vertex
29      tri[1].position = sf::Vector2f(center.x + (L/2),
            center.y - (height/3.0));
30      tri[2].position = sf::Vector2f(center.x - (L /
            2),
31                                     center.y - (
                                           height / 3.0))
                                           ;
32      // top left vertex
33      tri[0].color = sf::Color::Red;
34      tri[1].color = sf::Color::Blue;
35      tri[2].color = sf::Color::Green;
36      // pushback tri to vector of vertexArrays
37      v->push_back(tri);
38
39      N--;
40
41      if (N < 0)
42        return;
43
44      // build child triangles
45      fTree(N, (L/2), sf::Vector2f(center.x - (L/4),
46                                   center.y + ((5*
                                         height)/6)), v);
47      fTree(N, (L/2), sf::Vector2f(center.x + ((3*L) /
            4),
48                                   center.y - (height)
                                         / 6), v);
49      fTree(N, (L/2), sf::Vector2f(center.x - (L / 2),
50                                   center.y - (2*
```

```
                                        height) / 3), v)
                                        ;
51 }
52
53 int main(int argc, char* argv[]) {
54     double L = std::strtod(argv[1], NULL);
55     Triangle tri(L);
56     int N = std::stoi(argv[2]);
57     int tempN = N;
58     std::vector<sf::VertexArray> vTriangles;
59
60     fTree(N, tri.getLength(),
61           sf::Vector2f(WINDOW_WIDTH/2, WINDOW_HEIGHT
              /2), &vTriangles);
62
63     sf::RenderWindow window(sf::VideoMode(
          WINDOW_WIDTH, WINDOW_HEIGHT),
64                             "Triangle");
65     while (window.isOpen()) {
66         sf::Event event;
67
68         while (window.pollEvent(event)) {
69             if (event.type == sf::Event::Closed)
70                 window.close();
71
72             if (event.type == sf::Event::KeyReleased
                  ) {
73                 if (event.key.code == sf::Keyboard::
                      Left) {
74                     // window.clear(sf::Color::Black
                          );
75                     vTriangles.clear();
76                     tempN--;
77                     fTree(tempN, tri.getLength(),
78                           sf::Vector2f(WINDOW_WIDTH
                              /2, WINDOW_HEIGHT/2),
79                           &vTriangles);
80                     break;
81                 }
82             }
```

```
83          }
84          window.clear(sf::Color::Black);
85      // draw the Triangles pushed to the vector
86          for (int i = 0; i < static_cast<int>(
                vTriangles.size()); i++) {
87              tri.setVertices(vTriangles[i]);
88              window.draw(tri);
89          }
90    window.display();
91      }
92 }
```

4.5.3 Triangle.h

```
1  // Copyright [2022] <Karan Swamy>
2
3  #ifndef _HOME_KARANSWAMY_PS3_TRIANGLE_H_
4  #define _HOME_KARANSWAMY_PS3_TRIANGLE_H_
5  #include <iostream>
6  #include <string>
7  #include <cmath>
8  #include <vector>
9  #include "SFML/Graphics.hpp"
10
11 class Triangle : public sf::Drawable {
12  public:
13     Triangle() {}
14     explicit Triangle(double L) { length = L; }
15     void setVertices(sf::VertexArray v) { vertices =
           v; }
16     void setLength(double len) { length = len; }
17     double getLength() { return length; }
18
19  private:
20     sf::VertexArray vertices;
21     double length;
22     virtual void draw(sf::RenderTarget& target, sf::
          RenderStates states) const;
23 };
24
```

```
25 #endif  // _HOME_KARANSWAMY_PS3_TRIANGLE_H_
```

4.5.4 Triangle.cpp

```
1 // Copyright [2022] <Karan Swamy>
2
3 #include "Triangle.h"
4
5 void Triangle::draw(sf::RenderTarget& target, sf::
    RenderStates states) const {
6     target.draw(vertices, states);
7 }
```

5 PS4 - Synthesizing a Plucked String Sound

5.1 Overview

5.1.1 Part A

For part A of this project I implemented a Circular Buffer. I created a class for this data structure with methods, enqueue(`int16_t` x), dequeue(), peek(), size(), isEmpty() and isFull(). Exceptions are thrown when the circular buffer is not being used properly.

Constructor - Creates an empty circular buffer with the argument passed to the constructor as size of the circular buffer. If the argument passed is less than 1, it throws an `std::invalid_argument` exception.

size - Returns the number of items currently in the circular buffer .

isEmpty - Returns true if the buffer is empty (size <= 0).

isFull - Returns true if the buffer is full (size == capacity)

Enqueue - Adds an item x(argument passed to the method) to the end

Dequeue - Deletes and returns the item from the front. If circular buffer is empty then it throws a run time error, "can't dequeue an empty ring."

Peek - Returns (but does not delete) the item from the front of the buffer.

5.1.2 Part B

Part B of this assignment is to simulate a guitar being plucked by using the circular buffer implemented in part A to store the frequencies of each musical note. In addition to the CircularBuffer class, I implemented a StringSound class. Each time a string is plucked, the plucked function is called which fills the buffer with random values between -32768 and 32767. These random values simulate white noise.

5.2 Key Concepts

5.2.1 Circular Buffer

A Circular Buffer is a queue like data structure that uses a single, fixed-size buffer as if it were circular. The next index after the last index is the first index. So it is connected end-to-end like a circle.

5.2.2 Lambda Function

In CircularBuffer.cpp, I used lambda functions in both the isEmpty(lines 26-28) function and the isFull(lines 30-32) function. The lambda for the

isEmpty function takes bufferSize as a parameter and the lambda for the isFull function takes bufferSize and bufferCapacity as parameters. I just return the lambda functions which check whether the buffer is empty and whether the buffer is full.

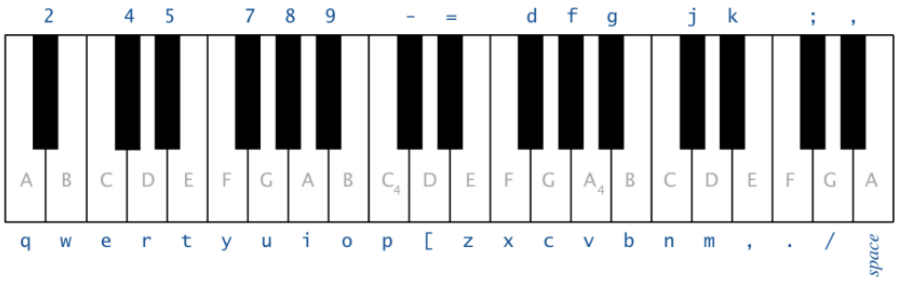
5.3 Learnings

5.3.1 Random Library

This library allows you to produce random numbers using combinations of generators and distributions. It is much more efficient than srand() and rand(). In lines 31-33 of StringSound.cpp an example of this library used to generate random numbers can be seen. The code here generates a random value between -32768 and 32767 which would represent the white noise.

5.4 Output

Figure 6: This Project has no visual component but this image shows the keys with their keyboard characters.



5.5 Code

5.5.1 Makefile

```
1 all: KSGuitarSim
2
3 KSGuitarSim: CircularBuffer.o StringSound.o
    KSGuitarSim.o
4     g++ CircularBuffer.o StringSound.o KSGuitarSim.o
          -o KSGuitarSim -lsfml-graphics -lsfml-window
          -lsfml-system -lsfml-audio
5
6 CircularBuffer.o: CircularBuffer.cpp CircularBuffer.
    h
7     g++ -Wall -Werror -pedantic -c CircularBuffer.
        cpp
8
9 StringSound.o: StringSound.cpp StringSound.h
    CircularBuffer.cpp CircularBuffer.h
10    g++ -Wall -Werror -pedantic -c StringSound.cpp
11
12 KSGuitarSim.o: StringSound.cpp StringSound.h
    CircularBuffer.cpp CircularBuffer.h
13    g++ -Wall -Werror -pedantic -c KSGuitarSim.cpp
14
```

```
15 clean:
16     rm *.o KSGuitarSim
```

5.5.2 KSGuitarSim.cpp

```
 1 /*
 2   Copyright 2015 Fred Martin,
 3   Y. Rykalova, 2020
 4 */
 5 #include "CircularBuffer.h"
 6 #include "StringSound.h"
 7
 8 #include <math.h>
 9 #include <limits.h>
10
11 #include <iostream>
12 #include <string>
13 #include <exception>
14 #include <stdexcept>
15 #include <vector>
16
17 #include <SFML/Graphics.hpp>
18 #include <SFML/System.hpp>
19 #include <SFML/Audio.hpp>
20 #include <SFML/Window.hpp>
21
22 #define CONCERT_A 220.0
23 #define SAMPLES_PER_SEC 44100
24
25 std::vector<sf::Int16> makeSamples(StringSound& gs)
     {    //NOLINT
26     std::vector<sf::Int16> samples;
27
28     gs.pluck();
29     int duration = 8;  // seconds
30     int i;
31     for (i = 0; i < SAMPLES_PER_SEC * duration; i++)
           {
32         gs.tic();
33         samples.push_back(gs.sample());
```

```
34      }
35
36      return samples;
37  }
38
39  int main() {
40      sf::RenderWindow window(sf::VideoMode(300, 200),
41                              "SFML Plucked String
                                   Sound Lite");
42      sf::Event event;
43      std::string keyboard = "q2we4r5ty7u8i9op-[=
           zxdcfvgbnjmk,.;/' ";
44      std::vector <sf::Sound> sounds;
45      sf::Sound sound;
46      sf::SoundBuffer* sBuffer;
47      std::string error = "sf::SoundBuffer:failed to
           load from samples.";
48
49      for (int i = 0; i < 37; i++) {
50          sBuffer = new sf::SoundBuffer;
51          std::vector<sf::Int16> samples;
52          double freq = 440 * pow(2, ((i - 24) / 12.0)
               );
53          StringSound strSound(freq);
54          samples = makeSamples(strSound);
55          if (!sBuffer->loadFromSamples(&samples[0],
               samples.size(),
56                                        2,
                                             SAMPLES_PER_SEC
                                             ))
57              throw std::runtime_error(error);
58          sound.setBuffer(*sBuffer);
59          sounds.push_back(sound);
60      }
61
62      while (window.isOpen()) {
63          while (window.pollEvent(event)) {
64              if (event.type == sf::Event::Closed) {
65                  window.close();
66              } else if (event.type == sf::Event::
```

```
                TextEntered) {
67                int index = keyboard.find(event.text
                      .unicode);
68                if (index >= 0 && index <= 36)
69                    sounds[index].play();
70            }
71
72            window.clear();
73            window.display();
74        }
75    }
76    return 0;
77 }
```

5.5.3 StringSound.h

```
1 //  Copyright [2022] <Karan Swamy>
2 #ifndef _HOME_KARANSWAMY_PS4B_STRINGSOUND_H_
3 #define _HOME_KARANSWAMY_PS4B_STRINGSOUND_H_
4 #include "CircularBuffer.h"
5 #include <vector>
6 #include "SFML/Graphics.hpp"
7
8 class StringSound {
9  public:
10   explicit StringSound(double frequency);
11   explicit StringSound(std::vector <sf::Int16> init)
       ;
12   StringSound(const StringSound& obj) = delete;  //
       no copy const
13   ~StringSound();
14   void pluck();
15   void tic();
16   sf::Int16 sample();
17   int time();
18  private:
19   CircularBuffer* _cb;
20   int _time;
21 };
22 #endif  // _HOME_KARANSWAMY_PS4B_STRINGSOUND_H_
```

5.5.4 StringSound.cpp

```
1 //  Copyright [2022] <Karan Swamy>
2 #include "StringSound.h"
3 #define SAMPLING_RATE 44100
4 #include <cmath>
5 #include <random>
6
7 StringSound::StringSound(double frequency) {
8     int N;
9     _time = 0;
10    N = ceil(SAMPLING_RATE / frequency);
11    _cb = new CircularBuffer(N);
12 }
13
14 StringSound::StringSound(std::vector <sf::Int16>
      init) {
15    if (init.size() == 0)
16        throw(std::invalid_argument("Constructor (
              vector): init is empty"));
17    int N = init.size();
18    _time = 0;
19    _cb = new CircularBuffer(N);
20
21    for (int i = 0; i < N; i++) {
22        _cb->enqueue(init[i]);
23    }
24 }
25
26 void StringSound::pluck() {
27    while (!_cb->isEmpty()) {
28        _cb->dequeue();
29    }
30
31    std::random_device rd;
32    std::mt19937 mt(rd());
33    std::uniform_int_distribution<int16_t> dist
          (-32768, 32767);
34
35    while (!_cb->isFull()) {
```

```
36          _cb->enqueue(dist(mt));
37      }
38 }
39
40 void StringSound::tic() {
41      int16_t newVal = _cb->peek();
42      int16_t deletedVal = _cb->dequeue();
43      int16_t kStrongVal = 0.996 * 0.5 * (newVal +
            deletedVal);
44
45      _cb->enqueue(kStrongVal);
46      ++_time;
47 }
48
49 sf::Int16 StringSound::sample() {
50      if (_cb->isEmpty())
51          throw std::runtime_error("Sample : No
                samples found");
52
53      return _cb->peek();
54 }
55 int StringSound::time() {
56      return _time;
57 }
58
59 StringSound::~StringSound() {
60      delete _cb;
61      _cb = nullptr;
62      _time = 0;
63 }
```

5.5.5 CircularBuffer.h

```
1 //  Copyright [2022] <Karan Swamy>
2 #ifndef  _HOME_KARANSWAMY_PS4B_CIRCULARBUFFER_H_
3 #define  _HOME_KARANSWAMY_PS4B_CIRCULARBUFFER_H_
4 #include <stdint.h>
5 #include <iostream>
6 #include <memory>
7
```

```
 8 class CircularBuffer {
 9  public:
10   // create an empty ring buffer, with given max
11   explicit CircularBuffer(int capacity);
12       // capacity
13   int size();  // return number of items currently
        in the buffer
14   bool isEmpty();  // is the buffer empty (size
        equals zero)?
15   bool isFull();  // is the buffer full (size equals
         capacity)?
16   void enqueue(int16_t x);  // add item x to the end
17   int16_t dequeue();  // delete and return item from
         the front
18   int16_t peek();  // return (but do not delete)
        item from the front
19
20  private:
21   std::unique_ptr<int16_t[]> buffer;
22   int bufferSize;
23   int bufferCapacity;
24   int front;
25   int back;
26 };
27 #endif  // _HOME_KARANSWAMY_PS4B_CIRCULARBUFFER_H_"
```

5.5.6 CircularBuffer.cpp

```
 1 //  Copyright [2022] <Karan Swamy>
 2 #include "CircularBuffer.h"
 3 #include <vector>
 4
 5 CircularBuffer::CircularBuffer(int capacity) {
 6     bufferCapacity = capacity;
 7     bufferSize = 0;
 8     front = 0;
 9     back = 0;
10
11     if (capacity < 1)
12         throw(std::invalid_argument(
```

```
13              "CircularBufer constructor: capacity
                   must be greater than zero."));
14      buffer = std::unique_ptr<int16_t[]>(new int16_t[
            capacity]);
15
16      for (int i = 0; i < capacity; i++) {
17          buffer[i] = 0;
18      }
19  }
20
21  int CircularBuffer::size() {
22      return bufferSize;
23  }
24
25  bool CircularBuffer::isEmpty() {
26      return [](int size) { return size <= 0 ? true :
            false; }(bufferSize);
27  }
28
29  bool CircularBuffer::isFull() {
30      auto myLambda = [](int size, int capacity)
31      { return size == capacity ? true : false; };
32      return myLambda(bufferSize, bufferCapacity);
33  }
34
35  void CircularBuffer::enqueue(int16_t x) {
36      if (isFull())
37          throw(std::runtime_error("enqueue: can't
                enqueue to a full ring."));
38
39      buffer[back % bufferCapacity] = x;
40      bufferSize++;
41      back++;
42  }
43
44  int16_t CircularBuffer::dequeue() {
45      if (isEmpty())
46          throw(std::runtime_error("dequeue: can't
                dequeue an empty ring."));
47
```

```
48      int16_t deletedData = buffer[front %
           bufferCapacity];
49      front++;
50      bufferSize--;
51
52      return deletedData;
53  }
54
55  int16_t CircularBuffer::peek() {
56      if (isEmpty())
57          throw(std::runtime_error("peak: can't peak
               an empty ring."));
58
59      return buffer[front % bufferCapacity];
60  }
```

5.5.7 test.cpp

```
1  //  Copyright [2022] <Karan Swamy>
2  #include "CircularBuffer.h"
3  #define BOOST_TEST_DYN_LINK
4  #define BOOST_TEST_MODULE Main
5  #include <boost/test/unit_test.hpp>
6
7  BOOST_AUTO_TEST_CASE(capacityCheck) {
8      BOOST_REQUIRE_THROW(CircularBuffer cb(0), std::
           invalid_argument);
9      BOOST_REQUIRE_NO_THROW(CircularBuffer cb(1));
10 }
11
12 BOOST_AUTO_TEST_CASE(enqueueFullRingException) {
13     CircularBuffer cb(1);
14     cb.enqueue(45);
15     BOOST_REQUIRE_THROW(cb.enqueue(56), std::
           runtime_error);
16     cb.dequeue();
17     BOOST_REQUIRE_NO_THROW(cb.enqueue(56));
18 }
19
20 BOOST_AUTO_TEST_CASE(dequeueEmptyRingException) {
```

```
21      CircularBuffer cb(1);
22      BOOST_REQUIRE_THROW(cb.dequeue(), std::
           runtime_error);
23      cb.enqueue(56);
24      BOOST_REQUIRE_NO_THROW(cb.dequeue());
25  }
26
27  BOOST_AUTO_TEST_CASE(checkSize) {
28      CircularBuffer cb(2);
29      cb.enqueue(45);
30      cb.enqueue(56);
31      BOOST_REQUIRE(cb.size() == 2);
32  }
33
34  BOOST_AUTO_TEST_CASE(checkIsEmpty) {
35      CircularBuffer cb(2);
36      BOOST_REQUIRE(cb.isEmpty() == true);
37      cb.enqueue(56);
38      BOOST_REQUIRE(cb.isEmpty() == false);
39  }
40
41  BOOST_AUTO_TEST_CASE(checkIsFull) {
42      CircularBuffer cb(2);
43      cb.enqueue(56);
44      cb.enqueue(45);
45      BOOST_REQUIRE(cb.isFull() == true);
46      cb.dequeue();
47      BOOST_REQUIRE(cb.isFull() == false);
48  }
49
50  BOOST_AUTO_TEST_CASE(checkEnqueue) {
51      CircularBuffer cb(1);
52      cb.enqueue(56);
53      BOOST_REQUIRE(cb.peek() == 56);
54  }
55
56  BOOST_AUTO_TEST_CASE(checkDequeue) {
57      CircularBuffer cb(1);
58      cb.enqueue(56);
59      cb.dequeue();
```

```
60      BOOST_REQUIRE(cb.size() == 0);
61  }
62
63  BOOST_AUTO_TEST_CASE(checkPeek) {
64      CircularBuffer cb(1);
65      BOOST_REQUIRE_THROW(cb.peek(), std::
            runtime_error);
66      cb.enqueue(56);
67      BOOST_REQUIRE_NO_THROW(cb.peek());
68  }
```

6 PS5 - DNA Sequence Alignment

6.1 Overview

6.1.1 Partner

This Project was done in collaboration with Samuel Vilt.

6.1.2 Objective

The goal of this project is to calculate the optimal sequence alignment of two DNA strings. What this means is that we align two strings in the most optimal pattern using dashes for a gap or space to find the edit distance(shortest possible distance) when comparing them both. For example, lets look at the two strings a = "AACAGTTACC" and b = "TAAGGTCA". The optimal sequence for this would be to add a space after the first A in b and a space after the second T in b. After doing this, the resulting edit distance is 7.

6.2 Key Concepts

6.2.1 Dynamic Programming (Needleman and Wunsch method)

The solution we implemented for this project used the Needleman and Wunsch method. The algorithm splits up a large problem into a series of smaller problems, and it uses the solutions to the smaller problems to find an optimal solution to the larger problem. This solution is faster than using a recursive approach but it does use more memory. The way this algorithm works: there are three possible moves(diagonal,left,up). It finds the move that costs the least and uses that. Moving left/up(signifies a gap) costs 2 points, moving diagonally(signifies a mismatch or match of characters) costs 1 if a mismatch or 0 if a match.

6.2.2 Lambda Function

In lines 47-48 of EDistance.cpp in the penalty() function, we used a lambda function which returns a 0 if the characters match and a 1 if the characters don't match.

6.3 Learnings

6.3.1 Memory

To look at the heap memory usage when dynamically allocating data in our project, we used valgrind massif which is a heap profiler. This helped us check and make our program more efficient in terms of heap usage.

6.4 Output

Figure 7: Output showing the optimal distance calculation

6.5 Code

6.5.1 Makefile

```
1 CC = g++
2 CFLAGS = -g -Wall -Werror -pedantic
3 DEPS = EDistance.h
4 OBJECTS = EDistance.o main.o
5 LIBS = -lsfml-graphics -lsfml-window -lsfml-system -
     lsfml-audio
6 EXE = EDistance
7
8 all: $(OBJECTS)
9     $(CC) $(OBJECTS) -o $(EXE) $(LIBS)
10 %.0: %.cpp $(DEPS)
11     $(CC) $(CFLAGS) -o $@ $<
12 clean:
13     rm $(OBJECTS) $(EXE)
```

6.5.2 main.cpp

```
1 //  Copyright [2022] Samuel Vilt, Karan Swamy
2 #include <iostream>
3 #include "EDistance.h"
4 #include <SFML/System.hpp>
5
6 int main(int argc, char* argv[]) {
7     sf::Clock clock;
8     sf::Time t;
9     std::string s1, s2;
10    std::cin >> s1 >> s2;
11    EDistance e(s1, s2);
12    int result = e.optDistance();
13
14    std::cout << e.alignment() <<std::endl;
15    std::cout << result << std::endl;
16    t = clock.getElapsedTime();
17    std::cout << "Execution time is " << t.asSeconds
          () << " seconds \n";
18
19    return 0;
20 }
```

6.5.3 EDistance.h

```
1 //  Copyright[2022] Karan Swamy, Samuel Vilt
2 #ifndef _HOME_SAMUEL_COMP4_PS5_EDISTANCE_H_
3 #define _HOME_SAMUEL_COMP4_PS5_EDISTANCE_H_
4
5 #include <iostream>
6 #include <string>
7 #include <vector>
8 #include <algorithm>
9
10 class EDistance{
11  public:
12      EDistance(std::string s1, std::string s2);
13      static int penalty(char a, char b);
14      static int min(int a, int b, int c);
15      int optDistance();
```

```
16      std::string getDna3() { return dna3; }
17      std::string alignment();
18
19  private:
20      std::vector <std::vector <int>> matrix;
21      int sSize1;
22      int sSize2;
23      std::string dna1;
24      std::string dna2;
25      std::string dna3;
26 };
27 #endif  //  _HOME_SAMUEL_COMP4_PS5_EDISTANCE_H_
```

6.5.4 EDistance.cpp

```
 1 //  Copyright[2022] Karan Swamy, Samuel Vilt
 2 #include "EDistance.h"
 3 #include <string>
 4 #include <algorithm>
 5 #include <iomanip>
 6
 7 EDistance::EDistance(std::string s1, std::string s2)
       {
 8     sSize1 = s1.size() + 1;
 9     sSize2 = s2.size() + 1;
10     if (sSize1 > sSize2) {
11         int temp = sSize1;
12         sSize1 = sSize2;
13         sSize2 = temp;
14         dna1 = s2;
15         dna2 = s1;
16     } else {
17         dna1 = s1;
18         dna2 = s2;
19     }
20     for (int r = 0; r < sSize1 + 1; r++) {
21         matrix.push_back(std::vector <int >(sSize2 +
               1, 0));
22     }
23
```

```
24      for (int i = 0; i < sSize1; i++) {
25          matrix[i][sSize2] = 2 *(sSize1-i);
26      }
27      for (int i = 0; i < sSize2; i++) {
28          matrix[sSize1][i] = 2 * (sSize2-i);
29      }
30  }
31
32  int EDistance::min(int a, int b, int c) {
33      int minNum = 0;
34
35      if (a <= b && a <= c)
36          minNum = a;
37      else if (b <= a && b <= c )
38          minNum = b;
39      else if (c <= a && c <= b)
40          minNum = c;
41
42      return minNum;
43  }
44
45  int EDistance::penalty(char a, char b) {
46      int p = 1;
47      return [](int pen, int a, int b)
48      { return a == b ? pen = 0 : pen; }(p, a, b);
49  }
50
51  int EDistance::optDistance() {
52      int gap1 = 0, gap2 = 0, comp = 0, optimal = 0;
53      for (int x = sSize1 - 1; x >= 0; x--) {
54          for (int y = sSize2 - 2; y >= 0; y--) {
55              comp = matrix[x + 1][y + 1] + penalty(
                    dna1[x], dna2[y]);
56              gap1 = matrix[x][y + 1] + 2;
57              gap2 = matrix[x + 1][y] + 2;
58              matrix[x][y] = min(comp, gap1, gap2);
59          }
60      }
61      optimal = matrix[0][0];
62      return optimal;
```

```
63 }
64
65 std::string EDistance::alignment() {
66     std::string output;
67     int x = 0, y = 0;
68
69     while (x < sSize1 && y < sSize2) {
70         if (dna1[x] == dna2[y] || matrix[x][y] ==
               matrix[x + 1][y + 1] + 1) {
71             output += dna1[x];
72             output += " ";
73             output += dna2[y];
74             output += " " + std::to_string(matrix[x
                  ][y] -
75             matrix[x + 1][y + 1]) + '\n';
76
77     x++;
78     y++;
79         } else if (matrix[x][y] == matrix[x + 1][y]
               + 2) {
80             output += dna1[x];
81             output += " - 2\n";
82             x++;
83         } else {
84             output += "- ";
85             output += dna2[y];
86             output += " 2\n";
87             y++;
88         }
89     }
90     while (x < sSize1 - 1) {
91         output += dna1[x];
92         output += " - 2\n";
93         x++;
94     }
95     while (y < sSize2 - 1) {
96         output += "- ";
97         output += dna2[y];
98         output += " 2\n";
99         y++;
```

```
100     }
101
102     output.pop_back();
103     output.pop_back();
104
105     return output;
106 }
```

7 PS6 - Random Writer

7.1 Overview

7.1.1 Partner

This Project was done in collaboration with Samuel Vilt.

7.1.2 Objective

In this project, we implemented a pseudo-random text generator using Markov chains. The algorithm has the following steps : 1.Get the input text as a string 2.Find the order-k(An order-k Markov model uses strings of k-letters to predict text, these are called k-grams) 3. Parse the input string according to the order k accessing the string as it were circular(when finding the sub string of length k at the end, wrap around to the front of the string). 4. Call generate function which takes a string k-gram(stored in Markov Chain) and the Length L of the characters to be generated.

7.2 Key Concepts

7.2.1 Markov Model

The Markov model gets a k-gram of user input length k. With this k-gram we find the frequency of every character that follows it and then randomly generate one of them in the new text. At only a moderate order of k (orders 5-7), the randomly generated text begins to take on many of the characteristics of the source text.

7.2.2 Map

Maps are associative containers that store elements in a mapped fashion. We used nested maps to store the data for our Markov model for this project. This was the declaration of the nested map. `std::unordered_map`<string,std::pair<int, `std::unordered_map`<char, int»>. The string(key of outer map) of the outer map stores the k-gram of length k. The first member of the pair(value of outer map) stores the frequency of the particular k-gram and the second member stores another map. This inner map stores the characters that follow the particular k-gram(key of inner map) and the frequency of these characters(value of outer map). Since we were dealing with so many values that were associated with each other we decided to use maps.

7.2.3 Lambda Function

In lines 87-91 we used a lambda and it was in our function to create a random k-gram. It generates a random number to be used to find a random k-gram in the map. It returns this string to be returned by the function.

7.2.4 Overloading the stream insertion operator «

We overloaded the stream insertion operator which displayed the internal state of the Markov Model. It prints out the order, the alphabet, and the frequencies of the k-grams and k+1-grams. This was very helpful with debugging as we could see if the Markov chain was being constructed properly.

7.3 Learnings

7.3.1 Nested Maps and std::pair

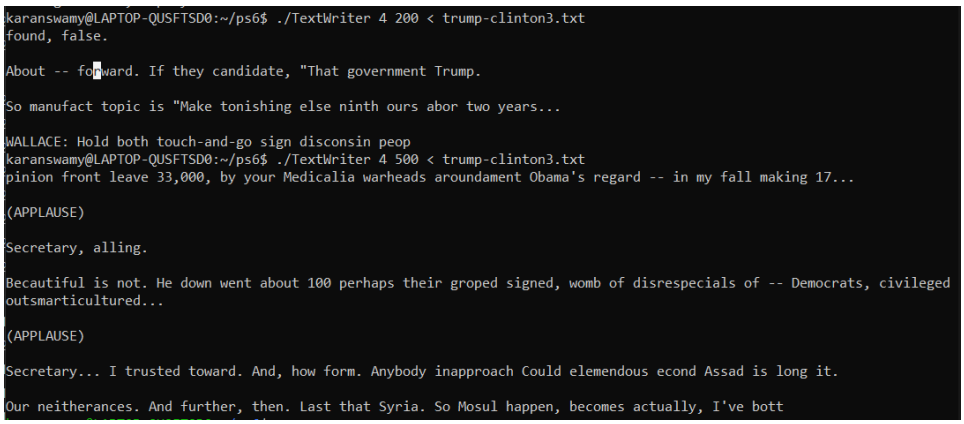
Prior to this project I had only used a single map to store data. However, through this project I learned how to use nested maps along with std::pair.

7.4 Exception Handling

We throw exceptions in lines 22,26,33,44, and 48 in RandWriter.cpp. These exceptions are significant for unit testing which ensures all the features of our program are working as they should.

7.5 Output

Figure 8: kgrams of size 10 generating 100,000 characters of trump text. Remember larger kgrams will always create more readable outputs.



7.6 Code

7.6.1 Makefile

```
CC = g++
CFLAGS = -g -Wall -Werror -pedantic
DEPS = RandWriter.h
OBJECTS = RandWriter.o TextWriter.o
LIBS = -lboost_unit_test_framework
EXE = TextWriter

all: TextWriter test

TextWriter: $(OBJECTS)
    $(CC) $(OBJECTS) -o $(EXE) $(LIBS)
test: test.o RandWriter.o
    $(CC) test.o RandWriter.o -o test $(LIBS)
%.0: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -o $@$<
clean:
    rm $(OBJECTS) $(EXE) test.o test
```

7.6.2 TextWriter.cpp

```
1 //  CopyRight [2022] Samuel Vilt, Karan Swamy
2 #include "RandWriter.h"
3
4 int main(int argc, char* argv[]) {
5         int K = std::atoi(argv[1]);
6         int L = std::atoi(argv[2]);
7
8         string str1 , str2;
9         while (std::getline(std::cin, str2, '\0')) {
10                str1 += str2;
11        }
12        RandWriter text(str1, K);
13        std::cout << text.generate(text.k_gramRand()
             , L) << std::endl;
14        return 0;
15 }
```

7.6.3 RandWriter.h

```
1 //  CopyRight [2022] Samuel Vilt, Karan Swamy
2 #ifndef _HOME_SAMUEL_COMP4_PS6_RANDWRITER_H_
3 #define _HOME_SAMUEL_COMP4_PS6_RANDWRITER_H_
4
5 #include <random>
6 #include <utility>
7 #include <algorithm>
8 #include <iterator>
9 #include <sstream>
10 #include <stdexcept>
11 #include <iostream>
12 #include <unordered_map>
13 #include <string>
14
15 using std::string;
16
17 class RandWriter{
18  public:
19     RandWriter(string text, int k);
20     int orderK() const { return order; }
```

```
21      int freq(string kgram) const;
22      int freq(string kgram, char c) const;
23      char kRand(string kgram);
24      string generate(string kgram, int L);
25      friend std::ostream& operator
26          <<(std::ostream& out, const RandWriter&
               rndWriter);
27      string k_gramRand();
28
29   private:
30      int order;
31      std::unordered_map <string,
32           std::pair<int, std::unordered_map<char, int
                >>> k_grams;
33      int randNumGen(int num);
34 };
35 #endif  //  _HOME_SAMUEL_COMP4_PS6_RANDWRITER_H_
```

7.6.4 RandWriter.cpp

```
1  //  CopyRight [2022] Samuel Vilt, Karan Swamy
2  #include "RandWriter.h"
3
4  RandWriter::RandWriter(string text, int k) {
5      order = k;
6      size_t i = 0, j = 0;
7      while (i < text.length()) {
8              std::string kstring;
9              while (j < k) {
10             kstring += text[(i+j)%text.length()];
11             j++;
12             }
13             j = 0;
14             ++k_grams[kstring].first;
15             ++k_grams[kstring].second[text[(i+k)%
                  text.length()]];
16             i++;
17     }
18 }
19
```

```
20 int RandWriter::freq(string kgram) const {
21     if (static_cast<int>(kgram.length()) != order) {
22         throw std::invalid_argument
23         ("freq(string) : kgram lenght not equal to
              order");
24     }
25     if (k_grams.find(kgram) == k_grams.end()) {
26         throw std:: invalid_argument("kgram does not
               exsist");
27     }
28     return k_grams.at(kgram).first;
29 }
30
31 int RandWriter::freq(std::string kgram, char c)
      const {
32         if (static_cast<int>(kgram.length()) !=
              order) {
33         throw std::invalid_argument
34         ("freq(string, char) : kgram lenght not
              equal to order");
35     }
36     if (k_grams.at(kgram).second.find(c) != k_grams.
          at(kgram).second.end()) {
37         return k_grams.at(kgram).second.at(c);
38     }
39     return 0;
40 }
41
42 char RandWriter::kRand(string kgram) {
43     if (static_cast<int>(kgram.length()) != order) {
44         throw std::invalid_argument
45         ("freq(string) : kgram length not equal to
              order");
46     }
47     if (k_grams.find(kgram) == k_grams.end()) {
48         throw std :: invalid_argument("kgram does
              not exist ");
49     }
50     auto val = k_grams[kgram].second.begin();
51     int rand = randNumGen(k_grams[kgram].second.size
```

```
        () - 1);
52      std::advance(val, rand);
53      return val->first;
54  }
55
56  int RandWriter::randNumGen(int num) {
57      std::random_device rd;
58      std::mt19937 gen(rd());
59      std::uniform_int_distribution <> distrib(0, num)
            ;
60      return static_cast<int>(distrib(gen));
61  }
62
63  string RandWriter::generate(string kgram, int L) {
64      string kStr(kgram);
65      kStr += kRand(kStr);
66      for (int i = 1; static_cast<int>(kStr.length())
            < L; ++i) {
67      kStr += kRand(kStr.substr(i, order));
68      }
69      return kStr;
70  }
71
72  std::ostream& operator <<(std::ostream& out, const
        RandWriter& rndWriter) {
73          for (auto i = rndWriter.k_grams.begin();
74              i != rndWriter.k_grams.end(); i++) {
75          out << i->first << " = " << (i->second).
               first << ", ";
76          for (auto j = i->second.second.begin();
77              j != i->second.second.end(); j++) {
78              out << j->first << "=" << j->second << "
                    ";
79              }
80              out << std::endl;
81          }
82      return out;
83  }
84
85  string RandWriter::k_gramRand() {
```

```
86      int num = randNumGen(k_grams.size() - 1);
87      string lambda = [this](int n) {
88          auto ptr = k_grams.begin();
89          std::advance(ptr, n);
90          return ptr->first;
91      }(num);
92      return lambda;
93  }
```

7.6.5 test.cpp

```
1  //  CopyRight [2022] Samuel Vilt, Karan Swamy
2  # include "RandWriter.h"
3  # define BOOST_TEST_DYN_LINK
4  # define BOOST_TEST_MODULE Main
5  # include <boost/test/unit_test.hpp>
6
7  BOOST_AUTO_TEST_CASE(check_kgram_length) {
8      RandWriter text("gagggagaggcgagaaa", 3);
9      BOOST_CHECK_THROW(text.freq("gg"), std::
           invalid_argument);
10     BOOST_CHECK_NO_THROW(text.freq("gga"));
11 }
12
13 BOOST_AUTO_TEST_CASE(checkValid_kgram) {
14     RandWriter text("gagggagaggcgagaaa", 3);
15     BOOST_CHECK_THROW(text.freq("gac") , std::
           invalid_argument);
16 }
17
18 BOOST_AUTO_TEST_CASE(checkValid_order) {
19     RandWriter text("gagggagaggcgagaaa", 3);
20     BOOST_REQUIRE(text.orderK() == 3);
21 }
22
23 BOOST_AUTO_TEST_CASE(check_two_input_freq_return ) {
24     RandWriter text("gagggagaggcgagaaa", 3);
25     BOOST_REQUIRE(text.freq("gag", 'g') == 2);
26 }
27
```

```
28 BOOST_AUTO_TEST_CASE(check_kRand_return) {
29     RandWriter text("gagggagaggcgagaaa", 3);
30     BOOST_REQUIRE(text.kRand("gag") != 'c');
31     char temp = text.kRand("gag");
32     BOOST_REQUIRE(temp == 'g' || temp == 'a');
33 }
34
35 BOOST_AUTO_TEST_CASE(check_string_generated_length)
       {
36     RandWriter text("gagggagaggcgagaaa", 3);
37     string temp = text.generate(text.k_gramRand(),
           4);
38     BOOST_REQUIRE(temp.size() == 4);
39 }
40
41 BOOST_AUTO_TEST_CASE(check_kRand_Exceptions) {
42     RandWriter text("gagggagaggcgagaaa", 3);
43     BOOST_CHECK_THROW(text.kRand("gg"), std::
           invalid_argument);
44     BOOST_CHECK_NO_THROW(text.freq("gga"));
45 }
```

8 PS7 - Kronos Regex Parsing

8.1 Overview

The goal of this project is to parse a large log file of a Kronos InTouch device which is provided by a company called Kronos. We need to look for messages that indicate the device has had a successful boot and how long it took to boot. This can be accomplished by using regular expressions and Gregorian features.

8.2 Key Concepts

8.2.1 Regular Expressions

A regular expression(regex) is an expression which contains a sequence of characters that define a search pattern.The first regex looks for the start message and boot date and time. The second regex looks for the success message and date. Then to store the matches at particular locations and to correctly output the details of the device boot to the output file I used smatch from the boost library.

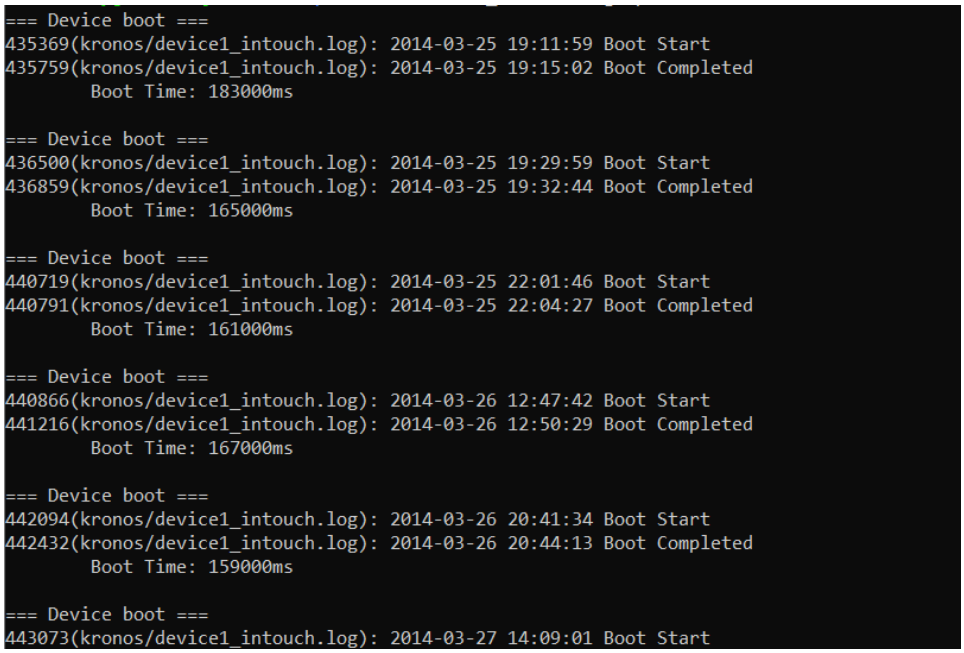
8.3 Learnings

8.3.1 Gregorian Features - Boost Library

Through this project I learned how to use Gregorian features to figure out the dates of the boot times and also find the duration of these boots.

8.4 Output

Figure 9: What an output file of a particular device log would look like after running the program.



8.5 Code

8.5.1 Makefile

```
1  all: ps7
2
3  ps7: main.o
4      g++ main.o -o ps7 -lboost_regex
5
6  main.o: main.cpp
7      g++ -Wall -Werror -pedantic -c main.cpp
8
9  clean:
10     rm *.o ps7
```

8.5.2 main.cpp

```
1 // Copyright [2022] Karan Swamy
2 #include <exception>
3 #include <iostream>
4 #include <fstream>
5 #include <string>
6 #include <boost/regex.hpp>
7 #include <boost/date_time/gregorian/gregorian.hpp>
8 #include <boost/date_time/posix_time/posix_time.hpp>
9
10 using std::cout;
11 using std::cin;
12 using std::endl;
13 using std::string;
14 using boost::gregorian::date;
15 using boost::gregorian::from_simple_string;
16 using boost::gregorian::date_period;
17 using boost::gregorian::date_duration;
18 using boost::posix_time::ptime;
19 using boost::posix_time::time_duration;
20
21 int main(int argc , char* argv[]) {
22     std::ifstream inFile(argv[1]);
23     string output = argv[1];
24     output += ".rpt";
25
26     if (!inFile.is_open()) {
27         cout << "Cannot open file" <<endl;
28         exit(1);
29     }
30
31     std::ofstream outFile(output);
32
33     boost::regex bootStart(
34     "([0-9]+)-([0-9]+)-([0-9]+) "
35     "([0-9]+):([0-9]+):([0-9]+): "
36     "\\(log.c.166\\) server started.*");
37
38     boost::regex bootSuccess(
39     "([0-9]+)-([0-9]+)-([0-9]+) "
40     "([0-9]+):([0-9]+):([0-9]+).([0-9]+):INFO:"
```

```
41      "oejs.AbstractConnector:Started
           SelectChannelConnector@0.0.0.0:9080.*");
42
43      string lineRead;
44      boost::smatch m;
45      bool boot = false;
46      int lineNum = 1;
47      ptime timeBegin;
48      date dt;
49
50      while (getline(inFile, lineRead)) {
51          if (regex_match(lineRead, m, bootStart)) {
52              date tempDate(stoi(m[1]), stoi(m[2]),
                   stoi(m[3]));
53              dt = tempDate;
54              ptime tempTime(dt, time_duration(stoi(m
                   [4]),
55                                  stoi(m[5]), stoi(m[6])));
56              timeBegin = tempTime;
57
58              if (boot) {
59                  outFile << " **** Incomplete Boot
                       **** " <<
60                  endl << endl;
61                  boot = false;
62              }
63
64              outFile << "=== Device boot ==="
65                      << endl
66                      << lineNum << "(" << argv[1] <<
                           "): "
67                      << m[1] << "-" << m[2] << "-" <<
                           m[3]
68                      << " "
69                      << m[4] << ":" << m[5] << ":" <<
                           m[6]
70                      << " " << "Boot Start" << endl;
71              boot = true;
72          } else if (regex_match(lineRead, m,
               bootSuccess)) {
```

```
73              date tempDateEnd(stoi(m[1]), stoi(m[2]),
                    stoi(m[3]));
74              ptime tempEnd(tempDateEnd,
75                            time_duration(stoi(m[4])
                                ,
76                            stoi(m[5]), stoi(m[6])))
                                ;
77              outFile << lineNum << "(" << argv[1] <<
                   "): "
78                    << m[1] << "-" << m[2] << "-" << m
                         [3]
79                    << " "
80                    << m[4] << ":" << m[5] << ":" << m
                         [6]
81                    << " Boot Completed" << endl;
82
83              time_duration td = tempEnd - timeBegin;
84              int bootTime = td.total_milliseconds();
85              outFile << "\t" << "Boot Time: "
86                      << bootTime << "ms" << endl <<
                           endl;
87              boot = false;
88          }
89          lineNum++;
90      }
91  }
```