Peyton Somerville
Computing IV: Project Portfolio
Spring 2020

Contents:

PS0 Hello World with SFML

PS1 Linear Feedback Shift Register and Image Encoding

PS2 Recursive Graphics (Pythagoras Tree)

PS3 N-Body Simulation

PS4 Ring Buffer and Guitar Hero

PS5 DNA Sequence Alignment

PS6 Markov Model of Natural Language

PS0 Hello World with SFML

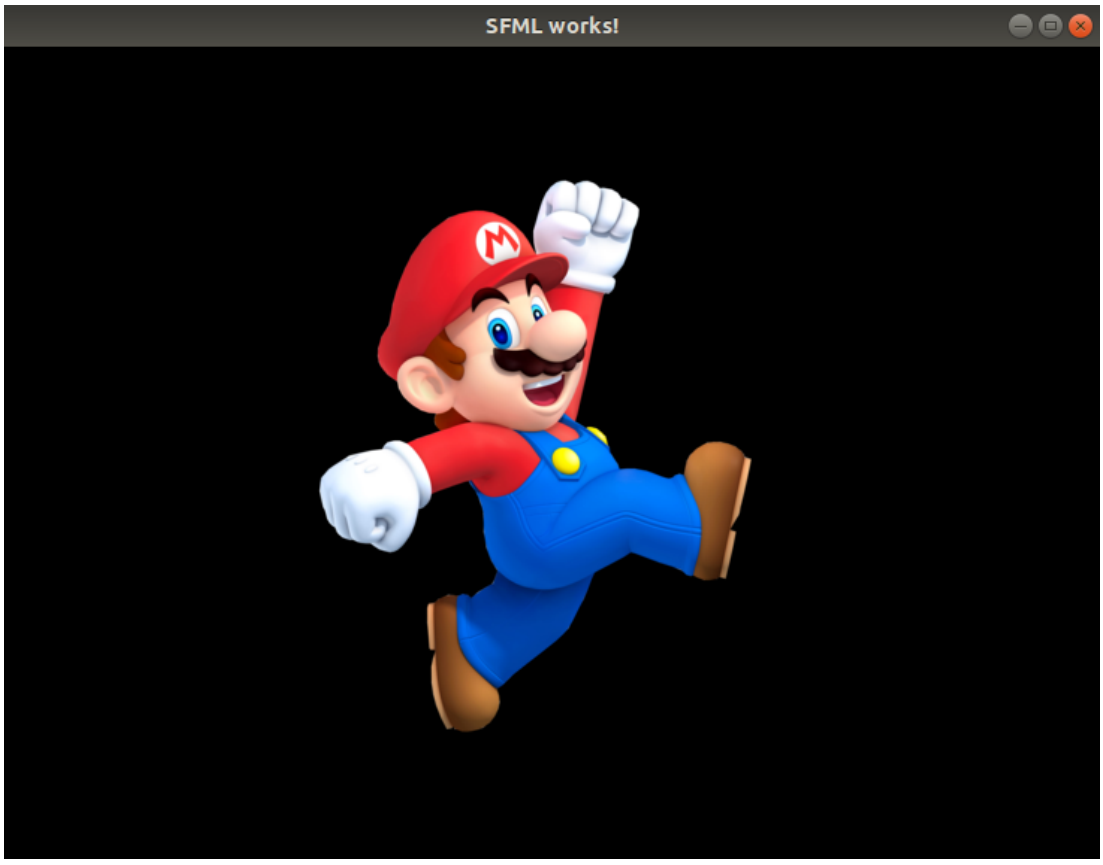
Overview:

The objective of this assignment was to get the virtual box setup with SFML library, and to start getting into the features of SFML, like creating an image sprite and drawing it to the window, as well as making the sprite move around.

What I Learned:

- The basics of SFML.
- Process of loading in an image and making a texture from it, then a sprite.
- Events like WindowEvent and KeyEvent.

Output and Code:



```
 1: /*
 2:   Peyton Somerville
 3: */
 4: #include <SFML/Graphics.hpp>
 5:
 6: using namespace sf;
 7:
 8: int main()
 9: {
10:     sf::RenderWindow window(sf::VideoMode(1000, 1000), "SFML works!");
11:     // sf::CircleShape shape(100.f);
12:     // shape.setFillColor(sf::Color::Green);
13:
14:     sf::Texture mario;
15:     if(!mario.loadFromFile("sprite.png"))
16:       return EXIT_FAILURE;
17:     sf::Sprite sprite(mario);
18:
19:     int x, y;
20:
21:     sprite.setPosition(150, 100);
22:
23:     while (window.isOpen())
24:     {
25:         sf::Event event;
26:         while (window.pollEvent(event))
27:         {
28:             if (event.type == sf::Event::Closed)
29:                 window.close();
30:         }
31:
32:         window.clear();
33:         // window.draw(shape);
34:
35:         if(Keyboard::isKeyPressed(Keyboard::A))
36:         {
37:           x = -2;
38:         }
39:         else if(Keyboard::isKeyPressed(Keyboard::D))
40:         {
41:           x = 2;
42:         }
43:         else
44:         {
45:           x = 0;
46:         }
47:         if(Keyboard::isKeyPressed(Keyboard::W))
48:         {
49:           y = -2;
50:         }
51:         else if(Keyboard::isKeyPressed(Keyboard::S))
52:         {
53:           y = 2;
54:         }
55:         else
56:         {
57:           y = 0;
58:         }
59:         //sprite.move(0.25, 0);  //2. make sprite move
60:         sprite.move(x, y);       //3. make sprite respong to key
61:         //sprite.rotate(0.5);      //4. do something else
```

```
   62:         window.draw(sprite);
   63:         window.display();
   64:     }
   65:
   66:     return 0;
   67: }
```

PS1 Linear Feedback Shift Register and Image Encoding

Overview:

This program produces pseudo-random bits by simulating a linear feedback shift register, and uses them to encode and decode images.

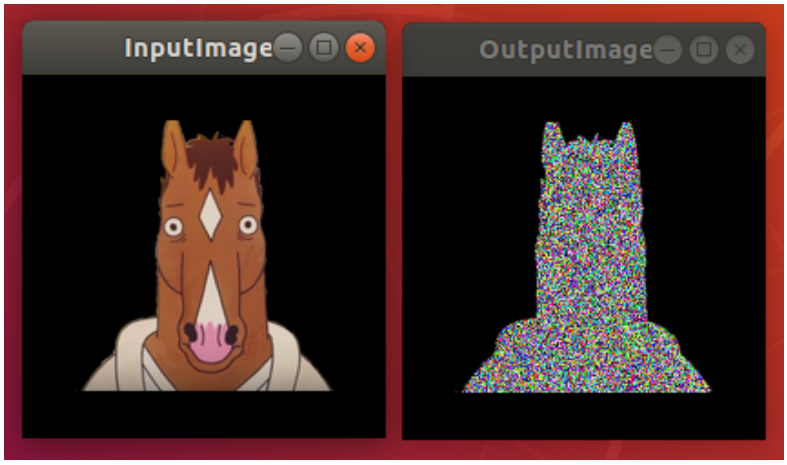
Implementation:

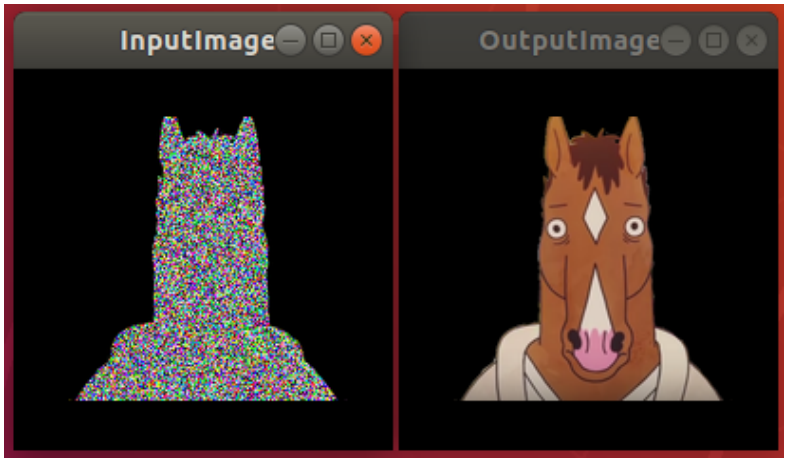
I used a string to implement the linear feedback shift register, because it is easy to manipulate specific elements of the string.

What I Learned:

- How to find the pseudo-random number by stepping through the binary number.
- How to run two SFML windows at the same time.
- How to manipulate specific pixels of an image, and their rgb values.

Output and Code:





Makefile Sun Feb 09 14:53:59 2020 1

```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -ansi -pedantic
 3: LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
 4: BOOST = -lboost_unit_test_framework
 5: OBJECTS = FibLFSR.o PhotoMagic.o
 6:
 7: all: PhotoMagic
 8:
 9: PhotoMagic: $(OBJECTS)
10:         $(CC) $(OBJECTS) -o PhotoMagic $(LFLAGS)
11:
12: FibLFSR.o:FibLFSR.cpp FibLFSR.hpp
13:         $(CC) -c FibLFSR.cpp $(CFLAGS)
14:
15: PhotoMagic.o:PhotoMagic.cpp FibLFSR.hpp
16:         $(CC) -c PhotoMagic.cpp FibLFSR.hpp $(LFLAGS)
17:
18: clean:
19:         rm $(OBJECTS) PhotoMagic *\~
```

```
 1: //Peyton Somerville
 2: //PS1b
 3:
 4: #include <iostream>
 5: #include <string>
 6: #include <SFML/Graphics.hpp>
 7: #include "FibLFSR.hpp"
 8:
 9: using namespace std;
10:
11: void transform(sf::Image& pic, FibLFSR* l);
12:
13: int main(int argc, char* argv[])
14: {
15:   string inputFileName(argv[1]);
16:   string outputFileName(argv[2]);
17:   string seed(argv[3]);
18:
19:   FibLFSR test(seed);
20:
21:   sf::Image inputPicture1;
22:   if(!inputPicture1.loadFromFile(inputFileName))
23:     return EXIT_FAILURE;
24:
25:   transform(inputPicture1, &test);
26:
27:   inputPicture1.saveToFile(outputFileName); //copy from input file to output
28:
29:   sf::Image inputPicture2;
30:   if(!inputPicture2.loadFromFile(inputFileName)) //load unchanged input file
31:     return EXIT_FAILURE;
32:   sf::Image outputPicture;
33:   if(!outputPicture.loadFromFile(outputFileName))
34:     return EXIT_FAILURE;
35:
36:   sf::Vector2u size = inputPicture2.getSize();
37:
38:   sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "InputImage");
39:   sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "OutputImage");
40:
41:   sf::Texture inputTexture, outputTexture;
42:   inputTexture.loadFromImage(inputPicture2);
43:   outputTexture.loadFromImage(outputPicture);
44:
45:   sf::Sprite inputSprite, outputSprite;
46:   inputSprite.setTexture(inputTexture);
47:   outputSprite.setTexture(outputTexture);
48:
49:   while (window1.isOpen() && window2.isOpen())
50:   {
51:     sf::Event event;
52:     while (window1.pollEvent(event))
53:     {
54:       if (event.type == sf::Event::Closed)
55:         window1.close();
56:     }
57:     while (window2.pollEvent(event))
58:     {
59:       if (event.type == sf::Event::Closed)
60:         window2.close();
61:     }
```

```
   62:     window1.clear();
   63:     window1.draw(inputSprite);
   64:     window1.display();
   65:     window2.clear();
   66:     window2.draw(outputSprite);
   67:     window2.display();
   68:    }
   69:
   70:   return 0;
   71: }
   72:
   73: void transform(sf::Image& pic, FibLFSR* l)
   74: {
   75:   uint i, j;
   76:   sf::Vector2u size = pic.getSize();
   77:   sf::Color col;
   78:   for(i = 0; i < size.x; i++)
   79:   {
   80:     for(j = 0; j < size.y; j++)
   81:     {
   82:       col = pic.getPixel(i, j);
   83:
   84:       col.r = col.r ^ l->generate(8);
   85:       col.g = col.g ^ l->generate(8);
   86:       col.b = col.b ^ l->generate(8);
   87:
   88:       pic.setPixel(i, j, col);
   89:     }
   90:   }
   91: }
```

```
 1: #ifndef FIBLFSR_HPP
 2: #define FIBLFSR_HPP
 3: #include <iostream>
 4: #include <string>
 5: #include <SFML/Graphics.hpp>
 6:
 7: using namespace std;
 8:
 9: class FibLFSR
10: {
11: public:
12:   FibLFSR(string seed);
13:
14:   int step(void);
15:
16:   int generate(int k);
17:
18:   friend ostream& operator<< (ostream &os, FibLFSR &lfsr);
19:
20: private:
21:   string bits;
22: };
23:
24: #endif
```

```
 1: #include "FibLFSR.hpp"
 2: #include <string>
 3: #include <iostream>
 4:
 5: using namespace std;
 6:
 7: FibLFSR::FibLFSR(string seed)
 8: {
 9:   bits = seed;
10: }
11:
12: int FibLFSR::step(void)
13: {
14:   int newBit;
15:   if(bits.size() != 16)
16:   {
17:     cout << "There needs to be 16 bits.\n";
18:     newBit = -1;
19:   }
20:   else
21:   {
22:     newBit = ((bits[0] ^ bits[2]) ^ bits[3]) ^ bits[5]; //XOR on bits
23:     char strNewBit;
24:     int i;
25:
26:     if(newBit == 1)
27:       strNewBit = '1';
28:     else
29:       strNewBit = '0';
30:
31:     for(i = 0; i < 16; i++)  //shift bits to the left
32:     {
33:         if(i == 15)
34:       {
35:         bits[i] = strNewBit;
36:       }
37:         else
38:       {
39:         bits[i] = bits[i + 1];
40:       }
41:     }
42:   }
43:
44:   return newBit;
45: }
46:
47: int FibLFSR::generate(int k)
48: {
49:   int result = 0, bit = 0;
50:   int i;
51:
52:   for(i = 0; i < k; i++)
53:   {
54:     bit = step();
55:     result = (result * 2) + bit;
56:   }
57:
58:   return result;
59: }
60:
61: ostream& operator<< (ostream &os, FibLFSR &lfsr)
```

```
62: {
63:   os << lfsr.bits;
64:   return os;
65: }
```

```
 1: // Peyton Somerville
 2:
 3: #include <iostream>
 4: #include <string>
 5:
 6: #include "FibLFSR.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(sixteenBitsThreeTaps) {
13:
14:   FibLFSR l("1011011000110110");
15:
16:   BOOST_REQUIRE(l.step() == 0);
17:   BOOST_REQUIRE(l.step() == 0);
18:   BOOST_REQUIRE(l.step() == 0);
19:   BOOST_REQUIRE(l.step() == 1);
20:   BOOST_REQUIRE(l.step() == 1);
21:   BOOST_REQUIRE(l.step() == 0);
22:   BOOST_REQUIRE(l.step() == 0);
23:   BOOST_REQUIRE(l.step() == 1);
24:
25:   FibLFSR l2("1011011000110110");
26:   BOOST_REQUIRE(l2.generate(9) == 51);
27: }
28:
29: BOOST_AUTO_TEST_CASE(tooManyOrFewBits)
30: {
31:   FibLFSR l3("101101100011011"); //only 15 bits
32:   BOOST_REQUIRE(l3.step() == -1); //step returns -1 if bits too short/long
33:
34:   FibLFSR l4("10110110001101101"); //17 bits
35:   BOOST_REQUIRE(l4.step() == -1);
36:
37:   FibLFSR l5(""); //no bits
38:   BOOST_REQUIRE(l5.step() == -1);
39: }
40:
41: BOOST_AUTO_TEST_CASE(allBitsAreOne)
42: {
43:   FibLFSR l6("1111111111111111");
44:
45:   BOOST_REQUIRE(l6.step() == 0);
46:   BOOST_REQUIRE(l6.step() == 0);
47:   BOOST_REQUIRE(l6.step() == 0);
48:   BOOST_REQUIRE(l6.step() == 0);
49:   BOOST_REQUIRE(l6.step() == 0);
50:   BOOST_REQUIRE(l6.step() == 0);
51:   BOOST_REQUIRE(l6.step() == 0);
52:   BOOST_REQUIRE(l6.step() == 0);
53:   BOOST_REQUIRE(l6.step() == 0);
54:   BOOST_REQUIRE(l6.step() == 0);
55:   BOOST_REQUIRE(l6.step() == 0);
56:   BOOST_REQUIRE(l6.step() == 1);
57: }
58:
```

PS2 Recursive Graphics (Pythagoras Tree)

Overview:

This program uses recursion to draw a Pythagoras tree.

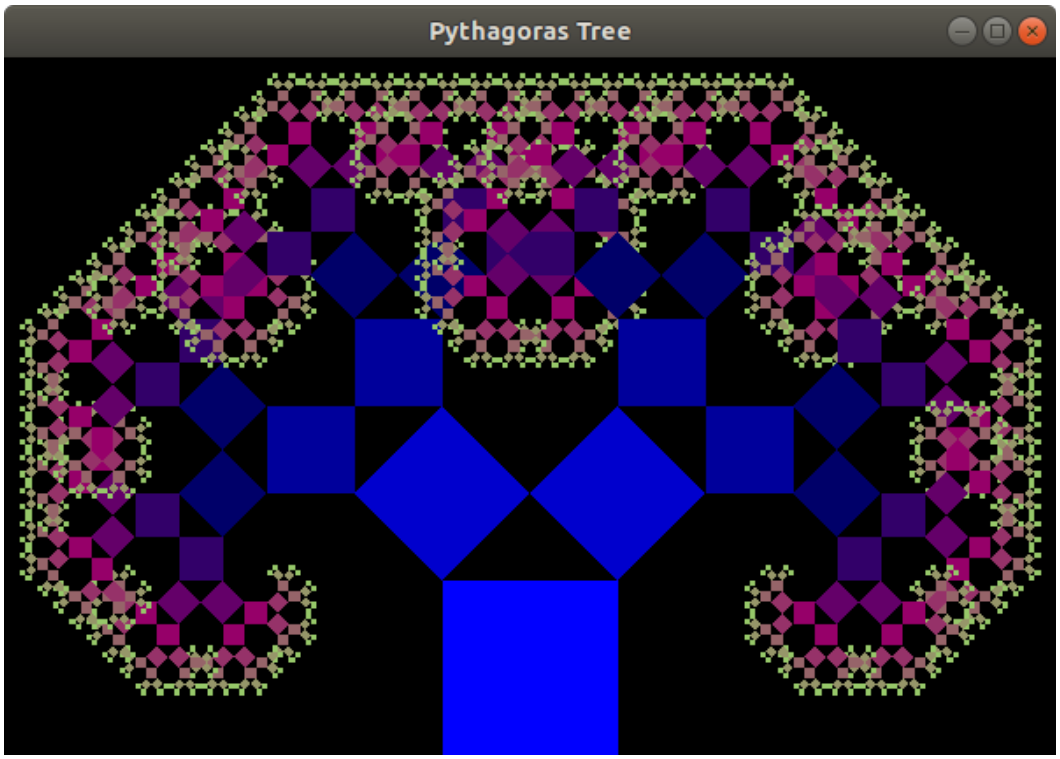
Implementation:

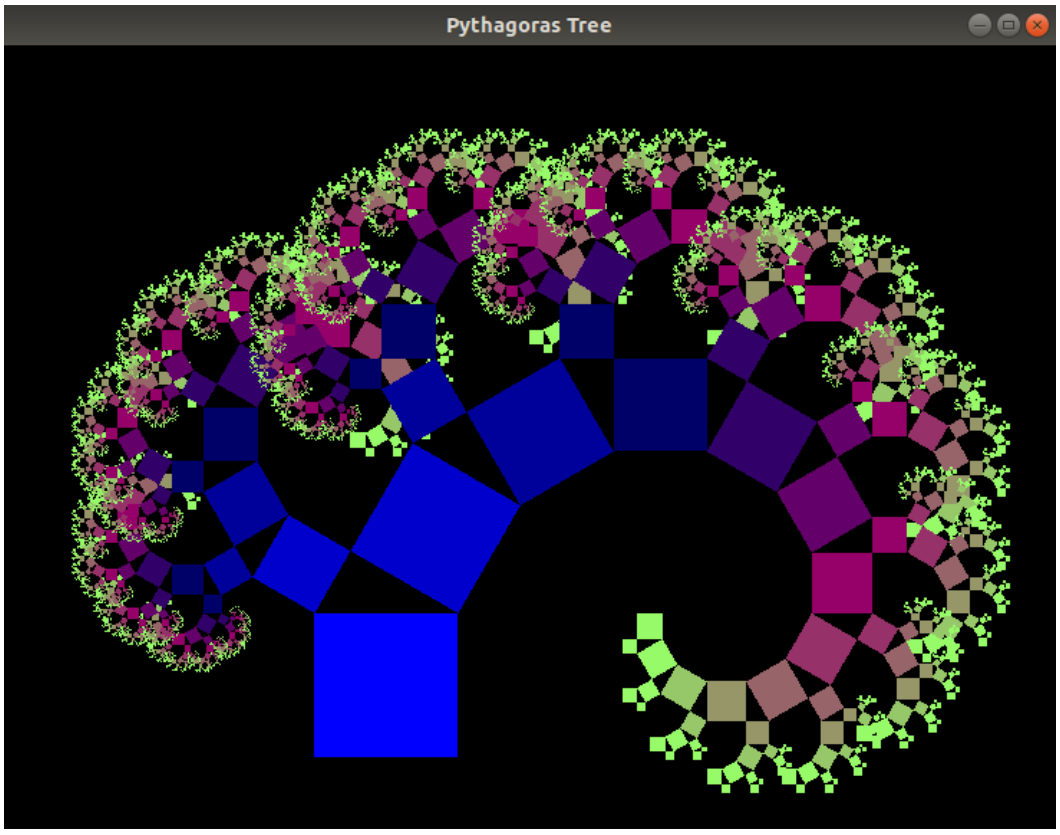
A base square is drawn, and it has a left and a right child that are scaled down and rotated above the base square. The left and right child are now treated as bases, and they each have a left and right child, creating a tree-like image when several iterations have been done.

What I Learned:

- Had already learned about recursion, but this made my understanding of it very solid.
- Inheritance from sf::Drawable to make drawing an object to the window more simple.
- Drawing shapes and adding color to them.

Output and Code:





Makefile Thu Feb 13 21:10:55 2020 1

```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -ansi -pedantic
 3: LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
 4: BOOST = -lboost_unit_test_framework
 5: OBJECTS = PTree.o main.o
 6:
 7: all: tree
 8:
 9: tree: $(OBJECTS)
10:         $(CC) $(OBJECTS) -o tree $(LFLAGS)
11:
12: PTree.o:PTree.cpp PTree.hpp
13:         $(CC) -c PTree.cpp $(CFLAGS)
14:
15: main.o:main.cpp PTree.hpp
16:         $(CC) -c main.cpp PTree.hpp $(LFLAGS)
17:
18: clean:
19:         rm $(OBJECTS) tree *\~
```

```
 1: /*
 2:   Peyton Somerville
 3: */
 4:
 5: #include <iostream>
 6: #include <string>
 7: #include "PTree.hpp"
 8:
 9: using std::cout;
10: using std::endl;
11: using std::stoi;
12: using std::stof;
13: using std::cin;
14:
15: int main(int argc, char* argv[])
16: {
17:   float baseSquareSize = stof(argv[1]);
18:   int depth = stoi(argv[2]);
19:
20:   if(baseSquareSize <= 0)
21:   {
22:     cout << "Please enter a base square size larger than 0.\n";
23:     return -1;
24:   }
25:   else if(depth < 0)
26:   {
27:     cout << "Please enter a depth of at least 0.\n";
28:     return -1;
29:   }
30:
31:   int LWindowX = baseSquareSize * 6;
32:   int LWindowY = baseSquareSize * 4;
33:
34:   bool type;
35:   cout << "Choose one of the following:\n";
36:   cout << "0: 45 45 90 Tree\n";
37:   cout << "1: 30 60 90 Tree\n>> ";
38:   cin >> type;
39:
40:   sf::RenderWindow window;
41:   if(type == false)
42:     window.create(sf::VideoMode(LWindowX, LWindowY), "Pythagoras Tree");
43:   else
44:     window.create(sf::VideoMode(650, 650), "Pythagoras Tree");
45:
46:   PTree object(baseSquareSize, depth, type);
47:
48:   while (window.isOpen())
49:   {
50:     sf::Event event;
51:     while (window.pollEvent(event))
52:     {
53:       if (event.type == sf::Event::Closed)
54:         window.close();
55:     }
56:
57:     window.clear();
58:     window.draw(object);
59:     window.display();
60:   }
61:
```

main.cpp Mon Feb 24 22:45:51 2020 2

```
   62:   return 0;
   63: }
```

PTree.hpp Mon Feb 24 22:33:48 2020 1

```
    1: #ifndef PTREE_HPP
    2: #define PTREE_HPP
    3: #include <iostream>
    4: #include <SFML/Graphics.hpp>
    5:
    6: class PTree : public sf::Drawable
    7: {
    8: public:
    9:   PTree(float baseSquareSize, int depth, bool type);
   10:
   11: private:
   12:   float L;
   13:   int N;
   14:   bool specialTree;
   15:   virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
;
   16: };
   17:
   18: void pTree(sf::RenderTarget& target, const int N,
   19:            const sf::RectangleShape& base); //45 45 90 triangle
   20:
   21: void pTree2(sf::RenderTarget& target, const int N,
   22:             const sf::RectangleShape& base); //30 60 90 triangle
   23:
   24: #endif
```

```
 1: #include <iostream>
 2: #include <cmath>
 3: #include <SFML/Graphics.hpp>
 4: #include "PTree.hpp"
 5:
 6: using std::cout;
 7: using std::endl;
 8:
 9: PTree::PTree(float baseSquareSize, int depth, bool type)
10: {
11:   L = baseSquareSize;
12:   N = depth;
13:   specialTree = type;
14: }
15:
16: void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const
17: {
18:   sf::RectangleShape baseSquare;
19:   baseSquare.setSize(sf::Vector2f(L, L));
20:   baseSquare.setOrigin(baseSquare.getSize() / 2.f);
21:   if(specialTree == true)
22:   {
23:     baseSquare.setPosition((target.getSize().x / 2.f) - L,
24:                            (target.getSize().y - L / 2.f) - L / 2);
25:   }
26:   else
27:   {
28:     baseSquare.setPosition(target.getSize().x / 2.f,
29:                            target.getSize().y - L / 2.f);
30:   }
31:   baseSquare.setFillColor(sf::Color::Blue);
32:
33:   if(specialTree == true)
34:     pTree2(target, N, baseSquare);
35:
36:   else
37:     pTree(target, N, baseSquare);
38: }
39:
40: void pTree(sf::RenderTarget& target, const int N,
41:            const sf::RectangleShape& base)
42: {
43:   const float scaleFactor = sqrt(2.f) / 2;
44:
45:   if(N < 0)
46:     return;
47:
48:   target.draw(base);
49:
50:   const sf::Vector2f& size = base.getSize();
51:   const sf::Transform& trans = base.getTransform();
52:
53:   sf::Color col = base.getFillColor();
54:   if(col.b > 150)
55:     col.b -= 50;
56:   else if(col.r < 150)
57:     col.r += 50;
58:   else if(col.g < 205)
59:     col.g += 50;
60:
61:   sf::RectangleShape left = base;
```

```
  62:   left.setSize(size * scaleFactor);
  63:   left.setOrigin(0, left.getSize().y);
  64:   left.setPosition(trans.transformPoint(0, 0));
  65:   left.rotate(-45);
  66:   left.setFillColor(col);
  67:   pTree(target, N - 1, left);
  68:
  69:   sf::RectangleShape right = base;
  70:   right.setSize(size * scaleFactor);
  71:   right.setOrigin(right.getSize());
  72:   right.setPosition(trans.transformPoint(size.x, 0));
  73:   right.rotate(45);
  74:   right.setFillColor(col);
  75:   pTree(target, N - 1, right);
  76: }
  77:
  78: void pTree2(sf::RenderTarget& target, const int N,
  79:             const sf::RectangleShape& base)
  80: {
  81:   const float scaleFactor1 = 1.f / 2;
  82:   const float scaleFactor2 = (1.f / 2) * sqrt(3.f);
  83:
  84:   if(N < 0)
  85:     return;
  86:   target.draw(base);
  87:
  88:   const sf::Vector2f& size = base.getSize();
  89:   const sf::Transform& trans = base.getTransform();
  90:
  91:   sf::Color col = base.getFillColor();
  92:   if(col.b > 150)
  93:     col.b -= 50;
  94:   else if(col.r < 150)
  95:     col.r += 50;
  96:   else if(col.g < 205)
  97:     col.g += 50;
  98:
  99:   sf::RectangleShape left = base;
 100:   left.setSize(size * scaleFactor1);
 101:   left.setOrigin(0, left.getSize().y);
 102:   left.setPosition(trans.transformPoint(0, 0));
 103:   left.rotate(-60);
 104:   left.setFillColor(col);
 105:   pTree2(target, N - 1, left);
 106:
 107:   sf::RectangleShape right = base;
 108:   right.setSize(size * scaleFactor2);
 109:   right.setOrigin(right.getSize());
 110:   right.setPosition(trans.transformPoint(size.x, 0));
 111:   right.rotate(30);
 112:   right.setFillColor(col);
 113:   pTree2(target, N - 1, right);
 114: }
 115:
 116:
 117:
```

PS3 N-Body Simulation

Overview:

This program takes in a text file as input that contains x position, y position, x velocity, y velocity, mass, and filename for a celestial object, and simulates planetary motion based on the input.

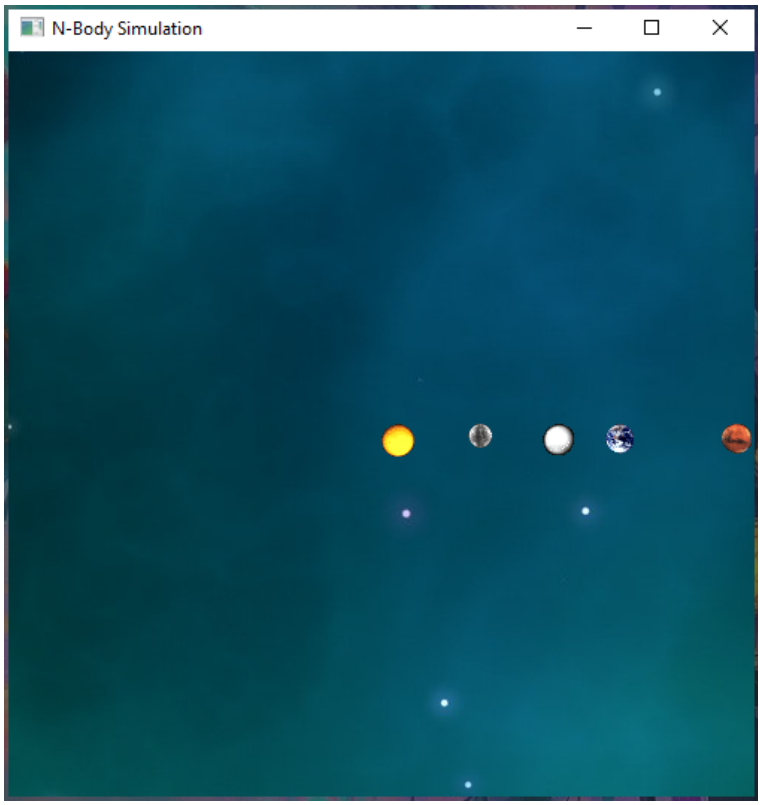
Implementation:

A class called CelestialBody was implemented, that sets up each celestial body in its correct position, and gets its associated image. Another class class Universe held CelestialBody objects, and implemented the movement for the bodies. It accomplishes this by using Newton's laws of motion and gravity, as well as the leapfrog finite difference approximation scheme.

What I Learned:

- How to animate with SFML.
- How to calculate forces acting on objects.
- How to use a unique_ptr.

Output and Code:



Makefile Mon Mar 09 15:09:04 2020 1

```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -ansi -pedantic -std=c++11
 3: LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 4: BOOST = -lboost_unit_test_framework
 5: OBJECTS = NBody.o main.o
 6:
 7: all: NBody
 8:
 9: NBody: $(OBJECTS)
10:         $(CC) $(OBJECTS) -o NBody $(LFLAGS)
11:
12: NBody.o:NBody.cpp NBody.hpp
13:         $(CC) -c NBody.cpp $(CFLAGS)
14:
15: main.o:main.cpp NBody.hpp
16:         $(CC) -c main.cpp NBody.hpp $(LFLAGS)
17:
18: clean:
19:         rm $(OBJECTS) NBody *\~
```

```
    1: /*
    2:         Peyton Somerville
    3: */
    4:
    5: #include <SFML/Graphics.hpp>
    6: #include <SFML/Audio.hpp>
    7: #include <iostream>
    8: #include <string>
    9: #include <sstream>
   10: #include <math.h>
   11: #include "NBody.hpp"
   12:
   13: using std::cin;
   14: using std::cout;
   15: using std::stod;
   16:
   17: int main(int argc, char* argv[])
   18: {
   19:         int numBodies;
   20:         double universeRadius, scaledUniverseSize;
   21:         int counter = 0;
   22:         double totalTime = stod(argv[1]);
   23:         double changeInTime = stod(argv[2]);
   24:         double simTime = 0;
   25:
   26:         cin >> numBodies;
   27:         cin >> universeRadius;
   28:
   29:         //calculating scale factor
   30:         double tempR = universeRadius;
   31:         while (tempR > 1000)
   32:         {
   33:                 tempR /= 10;
   34:                 counter++;
   35:         }
   36:         double scaleFactor = pow(10.0, counter);
   37:
   38:         if (tempR > 500)                //window has to be under 1000 x 1000
   39:         {
   40:                 scaledUniverseSize = universeRadius / scaleFactor;
   41:         }
   42:         else
   43:         {
   44:                 scaledUniverseSize = (universeRadius * 2) / scaleFactor;
   45:         }
   46:
   47:         Universe solarSystem(numBodies, scaledUniverseSize, scaleFactor);
   48:
   49:
   50:         sf::RenderWindow window(sf::VideoMode(scaledUniverseSize, scaledUniv
erseSize), "N-Body Simulation");
   51:
   52:         sf::Image backgroundPicture;
   53:         if (!backgroundPicture.loadFromFile("space.jpg"))
   54:         {
   55:                 cout << "Failed to open image file.\n";
   56:                 exit(1);
   57:         }
   58:
   59:         sf::Texture backgroundTexture;
   60:         backgroundTexture.loadFromImage(backgroundPicture);
```

```
   61:
   62:         sf::Sprite background;
   63:         background.setTexture(backgroundTexture);
   64:
   65:         sf::Font timeFont;
   66:         timeFont.loadFromFile("arial.ttf");
   67:
   68:         sf::Text timeText;
   69:         timeText.setFont(timeFont);
   70:         timeText.setCharacterSize(12);
   71:
   72:         sf::Music music;
   73:         if (!music.openFromFile("2001.wav"))
   74:         {
   75:                 return -1;
   76:         }
   77:         music.play();
   78:
   79:         window.setFramerateLimit(60);
   80:         while (window.isOpen())
   81:         {
   82:                 sf::Event event;
   83:                 while (window.pollEvent(event))
   84:                 {
   85:                         if (event.type == sf::Event::Closed)
   86:                                 window.close();
   87:                         else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Es
cape))
   88:                                 window.close();
   89:                 }
   90:
   91:                 window.clear();
   92:
   93:                 window.draw(background);
   94:
   95:                 timeText.setString("Time elapsed: " + std::to_string(simTime
));
   96:                 window.draw(timeText);
   97:
   98:                 window.draw(solarSystem);
   99:                 //cout << solarSystem << "\n";
  100:                 solarSystem.step(changeInTime);
  101:
  102:                 window.display();
  103:
  104:                 simTime += changeInTime;
  105:
  106:                 if (simTime >= totalTime)
  107:                 {
  108:                         break;
  109:                 }
  110:         }
  111:
  112:         cout << "// Final Results:\n" << numBodies << "\n" << universeRadius
 << "\n" << solarSystem << "\n";
  113:
  114:         return 0;
  115: }
```

```
    1: #ifndef NBODY_HPP
    2: #define NBODY_HPP
    3: #include <iostream>
    4: #include <SFML/Graphics.hpp>
    5:
    6: using std::string;
    7: using std::vector;
    8: using std::unique_ptr;
    9: using std::istream;
   10: using std::ostream;
   11:
   12: class CelestialBody : public sf::Drawable
   13: {
   14: public:
   15:         CelestialBody();                //default constructor
   16:         CelestialBody(double r, double s);              //value constructor
when called from Universe class
   17:         CelestialBody(double r, double x, double y, double xV, double yV, do
uble m, double s, string file);         //default value constuctor
   18:
   19:         double getXPos() const;
   20:         double getYPos() const;
   21:         double getMass() const;
   22:         double getXVel() const;
   23:         double getYVel() const;
   24:
   25:         void setXPos(double newPos);
   26:         void setYPos(double newPos);
   27:         void setXVel(double newVel);
   28:         void setYVel(double newVel);
   29:         void setSpritePos(double newXPos, double newYPos);
   30:
   31:         double findXForce(CelestialBody &cb1, CelestialBody &cb2);
   32:         double findYForce(CelestialBody &cb1, CelestialBody &cb2);
   33:         void setForces(double xForce, double yForce);
   34:         double getXFor() const;
   35:         double getYFor() const;
   36:
   37:
   38:         friend istream &operator >> (istream &input, CelestialBody& cb);
   39:         friend ostream &operator << (ostream &output, const CelestialBody& c
b);
   40:
   41: private:
   42:         double xPos, yPos;
   43:         double xVel, yVel;
   44:         double xFor, yFor;
   45:         double xAcc, yAcc;
   46:         double mass;
   47:
   48:         double radius;
   49:         double scale;
   50:
   51:         string fileName;
   52:         sf::Texture texture;
   53:         sf::Sprite sprite;
   54:
   55:         virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
 const;
   56: };
   57:
```

```
   58: class Universe : public sf::Drawable
   59: {
   60: public:
   61:         Universe(int N, double r, double s);
   62:
   63:         void step(double seconds);
   64:
   65:         friend ostream &operator << (ostream &output, const Universe& u);
   66:
   67: private:
   68:         vector<unique_ptr<CelestialBody>> bodies;
   69:
   70:         virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
 const;
   71: };
   72:
   73: #endif
```

```
    1: #include "NBody.hpp"
    2: #include <math.h>
    3:
    4: using std::cout;
    5: using std::endl;
    6: using std::cin;
    7: using std::move;
    8:
    9: /*********************Celestial Body Class*******************************/
   10: CelestialBody::CelestialBody()
   11: {
   12:         return;
   13: }
   14:
   15: CelestialBody::CelestialBody(double r, double s)
   16: {
   17:         radius = r / 2;
   18:         scale = s;
   19: }
   20:
   21: CelestialBody::CelestialBody(double r, double x, double y, double xV, double
 yV, double m, double s, string file)
   22: {
   23:         scale = s;
   24:         r /= scale;
   25:         xPos = x / scale;
   26:         yPos = y / scale;
   27:         xVel = xV;
   28:         yVel = yV;
   29:         mass = m;
   30:
   31:         if (!texture.loadFromFile(file))
   32:         {
   33:                 cout << "Error loading " << file << endl;
   34:                 exit(1);
   35:         }
   36:         sprite.setTexture(texture);
   37:
   38:         sprite.setPosition(r + xPos, r + yPos);
   39: }
   40:
   41: double CelestialBody::getXPos() const
   42: {
   43:         return xPos;
   44: }
   45: double CelestialBody::getYPos() const
   46: {
   47:         return yPos;
   48: }
   49: double CelestialBody::getMass() const
   50: {
   51:         return mass;
   52: }
   53: double CelestialBody::getXVel() const
   54: {
   55:         return xVel;
   56: }
   57: double CelestialBody::getYVel() const
   58: {
   59:         return yVel;
   60: }
```

```
   61:
   62: void CelestialBody::setXPos(double newPos)
   63: {
   64:         xPos = newPos;
   65: }
   66: void CelestialBody::setYPos(double newPos)
   67: {
   68:         yPos = newPos;
   69: }
   70: void CelestialBody::setXVel(double newVel)
   71: {
   72:         xVel = newVel;
   73: }
   74: void CelestialBody::setYVel(double newVel)
   75: {
   76:         yVel = newVel;
   77: }
   78: void CelestialBody::setSpritePos(double newXPos, double newYPos)
   79: {
   80:         sprite.setPosition(radius + newXPos / scale, radius - newYPos / scal
e);
   81: }
   82:
   83: double CelestialBody::findXForce(CelestialBody &cb1, CelestialBody &cb2)
   84: {
   85:         double x = cb2.xPos - cb1.xPos;
   86:         double y = cb2.yPos - cb1.yPos;
   87:         double r = sqrt((x * x) + (y * y));
   88:
   89:         const double G = 0.0000000000667;
   90:
   91:         double F = (G * cb1.mass * cb2.mass) / (r * r);
   92:         double Fx = F * (x / r);
   93:
   94:         return Fx;
   95: }
   96: double CelestialBody::findYForce(CelestialBody &cb1, CelestialBody &cb2)
   97: {
   98:         double x = cb2.xPos - cb1.xPos;
   99:         double y = cb2.yPos - cb1.yPos;
  100:         double r = sqrt((x * x) + (y * y));
  101:
  102:         const double G = 0.0000000000667;
  103:
  104:         double F = (G * cb1.mass * cb2.mass) / (r * r);
  105:         double Fy = F * (y / r);
  106:
  107:         return Fy;
  108: }
  109: void CelestialBody::setForces(double xForce, double yForce)
  110: {
  111:         xFor = xForce;
  112:         yFor = yForce;
  113: }
  114: double CelestialBody::getXFor() const
  115: {
  116:         return xFor;
  117: }
  118: double CelestialBody::getYFor() const
  119: {
  120:         return yFor;
```

```
  121: }
  122:
  123: istream &operator >> (istream &input, CelestialBody& cb)
  124: {
  125:         input >> cb.xPos >> cb.yPos;
  126:         input >> cb.xVel >> cb.yVel;
  127:         input >> cb.mass;
  128:         input >> cb.fileName;
  129:
  130:         if (!cb.texture.loadFromFile(cb.fileName))
  131:         {
  132:                 cout << "Error loading " << cb.fileName << endl;
  133:                 exit(1);
  134:         }
  135:         cb.sprite.setTexture(cb.texture);
  136:
  137:         cb.sprite.setPosition(cb.radius + cb.xPos / cb.scale, cb.radius - cb
.yPos / cb.scale);
  138:
  139:         return input;
  140: }
  141:
  142: ostream &operator << (ostream &output, const CelestialBody& cb)
  143: {
  144:         output.precision(4);
  145:         output << std::scientific;
  146:         output << cb.xPos << "  " << cb.yPos << "  ";
  147:         output << cb.xVel << "  " << cb.yVel << "  ";
  148:         output << cb.mass << "  ";
  149:         output << cb.fileName;
  150:
  151:         return output;
  152: }
  153:
  154: void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states)
const
  155: {
  156:         target.draw(sprite);
  157: }
  158:
  159: /*************************Universe Class*********************************
*/
  160:
  161: Universe::Universe(int N, double r, double s)
  162: {
  163:         int i;
  164:         for (i = 0; i < N; i++)
  165:         {
  166:                 unique_ptr<CelestialBody> temp(new CelestialBody(r, s));
  167:                 cin >> *temp;
  168:                 bodies.push_back(move(temp));
  169:                 temp.reset();
  170:         }
  171: }
  172:
  173: void Universe::step(double seconds)
  174: {
  175:         uint i, j, k;
  176:         double forceX, forceY;
  177:
  178:         for (j = 0; j < bodies.size(); j++)
```

```
  179:         {
  180:                 forceX = 0;
  181:                 forceY = 0;
  182:
  183:                 for (k = 0; k < bodies.size(); k++)
  184:                 {
  185:                         if (j != k)
  186:                         {
  187:                                 forceX += (*bodies.at(j)).findXForce(*bodies
.at(j), *bodies.at(k));
  188:                                 forceY += (*bodies.at(j)).findYForce(*bodies
.at(j), *bodies.at(k));
  189:                         }
  190:                 }
  191:
  192:                 (*bodies.at(j)).setForces(forceX, forceY);
  193:         }
  194:
  195:         for (i = 0; i < bodies.size(); i++)
  196:         {
  197:                 double Ax = (*bodies.at(i)).getXFor() / (*bodies.at(i)).getM
ass();
  198:                 double Ay = (*bodies.at(i)).getYFor() / (*bodies.at(i)).getM
ass();
  199:
  200:                 double Vx = (*bodies.at(i)).getXVel() + (seconds * Ax);
  201:                 double Vy = (*bodies.at(i)).getYVel() + (seconds * Ay);
  202:                 (*bodies.at(i)).setXVel(Vx);
  203:                 (*bodies.at(i)).setYVel(Vy);
  204:
  205:                 double Px = (*bodies.at(i)).getXPos() + (seconds * Vx);
  206:                 double Py = (*bodies.at(i)).getYPos() + (seconds * Vy);
  207:                 (*bodies.at(i)).setXPos(Px);
  208:                 (*bodies.at(i)).setYPos(Py);
  209:
  210:                 (*bodies.at(i)).setSpritePos(Px, Py);
  211:         }
  212: }
  213:
  214: ostream &operator << (ostream &output, const Universe& u)
  215: {
  216:         uint i;
  217:         for (i = 0; i < u.bodies.size(); i++)
  218:         {
  219:                 output << *(u.bodies).at(i) << endl;
  220:         }
  221:
  222:         return output;
  223: }
  224:
  225: void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const
  226: {
  227:         uint i;
  228:         for(i = 0; i < bodies.size(); i++)
  229:         {
  230:           target.draw(*bodies.at(i));
  231:         }
  232: }
```

PS4 Ring Buffer and Guitar Hero

Overview:

This program simulates a guitar with 37 notes ranging from 110Hz to 880Hz using the Karplus-Strong algorithm.

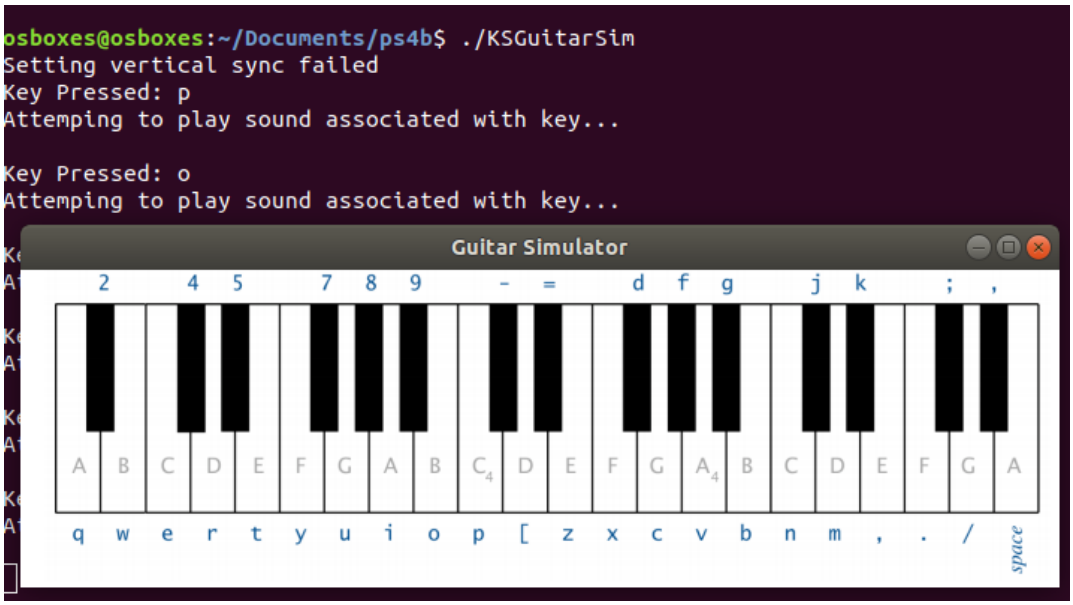
Implementation:

A class RingBuffer was implemented with a vector representing the buffer. Another class StringSound included a pointer to a RingBuffer object. This class includes a pluck() function that fills the ring buffer with random values that represent white noise. As well as a function tic() that advances the Karplus-Strong algorithm one step.

What I Learned:

- How to use sf::Sound and sf::SoundBuffer to create a sampling of sounds that produce a note.
- SFML audio output.
- Using c++ random number generator instead of using rand().

Output and Code:



Makefile Sun Apr 26 21:13:23 2020 1

```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -ansi -pedantic -std=c++11 -O2
 3: LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 4: BOOST = -lboost_unit_test_framework
 5: OBJECTS = RingBuffer.o StringSound.o KSGuitarSim.o
 6:
 7: all: KSGuitarSim
 8:
 9: KSGuitarSim: $(OBJECTS)
10:         $(CC) $(OBJECTS) -o KSGuitarSim $(LFLAGS)
11:
12: RingBuffer.o:RingBuffer.cpp RingBuffer.h
13:         $(CC) -c RingBuffer.cpp $(CFLAGS)
14:
15: StringSound.o:StringSound.cpp StringSound.h
16:         $(CC) -c StringSound.cpp $(CFLAGS)
17:
18: KSGuitarSim.o:KSGuitarSim.cpp RingBuffer.h StringSound.h
19:         $(CC) -c KSGuitarSim.cpp RingBuffer.h StringSound.h $(CFLAGS)
20:
21: clean:
22:         rm $(OBJECTS) KSGuitarSim *\~
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include <SFML/Graphics.hpp>
 6: #include <SFML/System.hpp>
 7: #include <SFML/Audio.hpp>
 8: #include <SFML/Window.hpp>
 9:
10: #include <math.h>
11: #include <limits.h>
12:
13: #include <iostream>
14: #include <string>
15: #include <exception>
16: #include <stdexcept>
17: #include <vector>
18:
19: #include "RingBuffer.h"
20: #include "StringSound.h"
21:
22: #define SAMPLES_PER_SEC 44100
23: #define SAMPLES_SIZE 44100 * 8  // SAMPLES_PER_SEC * duration (seconds)
24: #define KEYBOARD_SIZE 37
25:
26: std::vector<sf::Int16> makeSamples(StringSound gs) {
27:   std::vector<sf::Int16> samples;
28:
29:   gs.pluck();
30:   int i;
31:   for (i = 0; i < SAMPLES_SIZE; i++) {
32:     gs.tic();
33:     samples.push_back(gs.sample());
34:   }
35:
36:   return samples;
37: }
38:
39: int main() {
40:   try {
41:     // weird size to fit note reference picture perfectly
42:     sf::RenderWindow window(sf::VideoMode(695, 211), "Guitar Simulator");
43:     sf::Event event;
44:
45:     std::vector<sf::Int16> samples;
46:     std::vector<std::vector<sf::Int16>> vectSamples(KEYBOARD_SIZE);
47:     std::vector<sf::SoundBuffer> buffers(KEYBOARD_SIZE);
48:     std::vector<sf::Sound> sounds(KEYBOARD_SIZE);
49:
50:     int i;
51:     double freq;
52:     for (i = 0; i < KEYBOARD_SIZE; i++) {
53:       freq = 440 * pow(2, (i - 24) / 12.0);
54:       StringSound gs = StringSound(freq);
55:
56:       samples = makeSamples(gs);
57:       vectSamples[i] = samples;
58:
59:       if (!buffers[i].loadFromSamples(&vectSamples[i][0],
60:                                       SAMPLES_SIZE, 1, SAMPLES_PER_SEC))
61:         throw std::runtime_error("sf::SoundBuffer: failed to load "
```

```
   62:                                  "from samples.");
   63:
   64:       sounds[i].setBuffer(buffers[i]);
   65:     }
   66:     std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
   67:     int j;
   68:
   69:     // Note reference picture
   70:     sf::Image noteRefImage;
   71:     if (!noteRefImage.loadFromFile("GuitarSimKeyReference.PNG"))
   72:       return EXIT_FAILURE;
   73:     sf::Texture noteRefTexture;
   74:     noteRefTexture.loadFromImage(noteRefImage);
   75:     sf::Sprite noteRef;
   76:     noteRef.setTexture(noteRefTexture);
   77:
   78:     while (window.isOpen()) {
   79:       while (window.pollEvent(event)) {
   80:         switch (event.type) {
   81:         case sf::Event::Closed:
   82:           window.close();
   83:           break;
   84:
   85:         case sf::Event::TextEntered:
   86:           // need to convert from unicode to ASCII
   87:           if (event.text.unicode < 128) {
   88:             char key = (char)(event.text.unicode);
   89:
   90:             for (j = 0; j < KEYBOARD_SIZE; j++) {
   91:               if (keyboard[j] == key) {
   92:                 std::cout << "Key Pressed: " << keyboard[j] << "\n";
   93:                 std::cout << "Attemping to play sound associated "
   94:                   "with key...\n\n";
   95:                 sounds[j].play();
   96:                 break;
   97:               }
   98:             }
   99:           }
  100:
  101:         default:
  102:           break;
  103:         }
  104:
  105:         window.clear();
  106:         window.draw(noteRef);
  107:         window.display();
  108:       }
  109:     }
  110:   }
  111:   catch (std::invalid_argument& ia) {
  112:     std::cerr << "Invalid Argument - " << ia.what() << "\n";
  113:   }
  114:   catch (std::runtime_error& re) {
  115:     std::cerr << "Runtime Error - " << re.what() << "\n";
  116:   }
  117:   return 0;
  118: }
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #ifndef _HOME_OSBOXES_DOCUMENTS_PS4B_RINGBUFFER_H_
 6: #define _HOME_OSBOXES_DOCUMENTS_PS4B_RINGBUFFER_H_
 7: #include <stdint.h>
 8: #include <iostream>
 9: #include <vector>
10:
11: using std::ostream;
12:
13: class RingBuffer {
14: public:
15:         // create an empty ring buffer, with given max capacity
16:         explicit RingBuffer(int capacity);
17:
18:         int getCapacity();
19:
20:         int size();  // return number of items currently in the buffer
21:
22:         bool isEmpty();  // is the buffer empty (size equals zero)?
23:         bool isFull();  // is the buffer full (size equals capacity)?
24:
25:         void enqueue(int16_t x);  // add item x to the end
26:         int16_t dequeue();  // delete and return item from the front
27:
28:         int16_t peek();  // return (but do not delete) item from the front
29:
30:         // return (but do not delete) item from the location
31:         int16_t peekAt(int location);
32:
33:         friend ostream& operator<<(ostream& os, RingBuffer& rb);
34:
35:         void printBuffer();
36:
37: private:
38:         std::vector<int16_t> buffer;
39: };
40:
41: #endif  // _HOME_OSBOXES_DOCUMENTS_PS4B_RINGBUFFER_H_#endif
```

```
    1: /*
    2:   Copyright 2020 Peyton Somerville
    3: */
    4:
    5: #include <bits/stdc++.h>
    6: #include "RingBuffer.h"
    7: // Google wants it this way:
    8: // #include "/home/osboxes/Documents/ps4b/RingBuffer.h"
    9: // But then it would not work on every machine.
   10:
   11: RingBuffer::RingBuffer(int capacity) {
   12:         if (capacity < 1)
   13:                 throw std::invalid_argument("RingBuffer constructor: capacit
y "
   14:                         "must be greater than zero.");
   15:
   16:         buffer.reserve(capacity);
   17: }
   18:
   19: int RingBuffer::getCapacity()
   20: {
   21:         return buffer.capacity();
   22: }
   23:
   24: int RingBuffer::size() {
   25:         return buffer.size();
   26: }
   27:
   28: bool RingBuffer::isEmpty() {
   29:         if (buffer.empty() == true)
   30:                 return true;
   31:         return false;
   32: }
   33: bool RingBuffer::isFull() {
   34:         if (buffer.size() == buffer.capacity())
   35:                 return true;
   36:         return false;
   37: }
   38:
   39: void RingBuffer::enqueue(int16_t x) {
   40:         if (isFull())
   41:                 throw std::runtime_error("enqueue: can't enqueue to a full r
ing buffer.");
   42:
   43:         buffer.push_back(x);
   44: }
   45: int16_t RingBuffer::dequeue() {
   46:         if (isEmpty())
   47:                 throw std::runtime_error("dequeue: can't dequeue from"
   48:                         "an empty ring buffer.");
   49:
   50:         int16_t item;
   51:         item = buffer.at(0);
   52:         buffer.erase(buffer.begin());
   53:
   54:         return item;
   55: }
   56:
   57: int16_t RingBuffer::peek() {
   58:         if (isEmpty())
   59:                 throw std::runtime_error("peek: can't peek into an empty rin
```

```
g buffer.");
   60:
   61:         return buffer.front();
   62: }
   63:
   64: int16_t RingBuffer::peekAt(int location) {
   65:         if (isEmpty())
   66:                 throw std::runtime_error("peekAt: can't peek into an empty r
ing buffer.");
   67:
   68:         return buffer.at(location);
   69: }
   70:
   71: ostream& operator<<(ostream& os, RingBuffer& rb) {
   72:         if (rb.isEmpty())
   73:                 throw std::runtime_error("operator<<: can't print an empty r
ing buffer.");
   74:
   75:         int i;
   76:         for (i = 0; i < rb.size(); i++) {
   77:                 os << rb.peekAt(i) << "\n";
   78:         }
   79:
   80:         return os;
   81: }
   82:
   83: void RingBuffer::printBuffer() {
   84:         if (isEmpty())
   85:                 throw std::runtime_error("operator<<: can't print an empty r
ing buffer.");
   86:
   87:         // lambda expression to print vector
   88:         std::for_each(buffer.begin(), buffer.end(), [](int i) {
   89:                 std::cout << i << "  ";
   90:         });
   91:
   92:         std::cout << "\n";
   93: }
```

```
    1: /*
    2:   Copyright 2020 Peyton Somerville
    3: */
    4:
    5: #ifndef _HOME_OSBOXES_DOCUMENTS_PS4B_STRINGSOUND_H_
    6: #define _HOME_OSBOXES_DOCUMENTS_PS4B_STRINGSOUND_H_
    7:
    8: #include <iostream>
    9: #include <vector>
   10: #include <SFML/Audio.hpp>
   11: #include "RingBuffer.h"
   12:
   13: class StringSound{
   14: public:
   15:         // create a guitar string sound of the given frequency using a sampl
ing rate of 44,100
   16:         explicit StringSound(double frequency);
   17:
   18:         // create a guitar string with size and initial values are given by
the vector
   19:         explicit StringSound(std::vector<sf::Int16> init);
   20:
   21:         // pluck the guitar string by replacing the buffer with random value
s, representing white noise
   22:         void pluck();
   23:
   24:         // advance the simulation one time step
   25:         void tic();
   26:
   27:         // return the current sample
   28:         sf::Int16 sample();
   29:
   30:         // return number of times tic was called so far
   31:         int time();
   32:
   33:         friend ostream& operator<<(ostream& os, StringSound& ss);
   34:
   35: private:
   36:         RingBuffer* rb;
   37:         int ticCounter;
   38: };
   39: #endif  // _HOME_OSBOXES_DOCUMENTS_PS4B_STRINGSOUND_H_#endif
```

```
    1: /*
    2:   Copyright 2020 Peyton Somerville
    3: */
    4:
    5: #include <math.h>
    6: #include <random>
    7: #include "StringSound.h"
    8:
    9: #define SAMPLES_PER_SECOND 44100
   10:
   11: // create a guitar string sound of the given frequency using a sampling rate
 of 44,100
   12: StringSound::StringSound(double frequency){
   13:         if (frequency < 1)
   14:                 throw std::invalid_argument("StringSound constructor: freque
ncy "
   15:                         "must be greater than zero.");
   16:
   17:         int ringBufferCap = ceil(SAMPLES_PER_SECOND / frequency);
   18:
   19:         rb = new RingBuffer(ringBufferCap);
   20:
   21:         ticCounter = 0;
   22: }
   23:
   24: // create a guitar string with size and initial values are given by the vect
or
   25: StringSound::StringSound(std::vector<sf::Int16> init){
   26:         if (init.size() < 1)
   27:                 throw std::invalid_argument("StringSound constructor: init.s
ize() "
   28:                         "must be greater than zero.");
   29:
   30:         rb = new RingBuffer(init.size());
   31:
   32:         unsigned int i;
   33:         for (i = 0; i < init.size(); i++) {
   34:                 rb->enqueue(init.at(i));
   35:         }
   36:
   37:         ticCounter = 0;
   38: }
   39:
   40: // pluck the guitar string by replacing the buffer with random values, repre
senting white noise
   41: void StringSound::pluck(){
   42:         int i;
   43:         int size = rb->size();
   44:         if (rb->isEmpty() == false) {  // clear buffer if not empty
   45:                 for (i = 0; i < size; i++) {
   46:                         if (rb->isEmpty() == true)
   47:                                 break;
   48:                         rb->dequeue();
   49:                 }
   50:         }
   51:
   52:         int j;
   53:
   54:         // Seed with a real random value, if available
   55:         std::random_device r;
   56:
```

```
   57:         for (j = 0; j < rb->getCapacity(); j++) {
   58:                 // Choose a random mean between -32768 and 32767
   59:                 std::default_random_engine e1(r());
   60:                 std::uniform_int_distribution<int> uniform_dist(-32768, 3276
7);
   61:                 int randNum = uniform_dist(e1);
   62:
   63:                 rb->enqueue(randNum);
   64:         }
   65: }
   66:
   67: // advance the simulation one time step
   68: void StringSound::tic(){
   69:         if (rb->isEmpty())
   70:                 throw std::runtime_error("tic: can't tic with an empty ring
buffer.");
   71:
   72:         int16_t num1 = rb->dequeue();
   73:         int16_t num2 = rb->peek();
   74:         int16_t newNum = 0.996 * ((num1 + num2) / 2.0);
   75:
   76:         rb->enqueue(newNum);
   77:
   78:         ticCounter++;
   79: }
   80:
   81: // return the current sample
   82: sf::Int16 StringSound::sample(){
   83:         if (rb->isEmpty())
   84:                 throw std::runtime_error("sample: can't return a sample from
 an empty ring buffer.");
   85:         return (sf::Int16)rb->peek();
   86: }
   87:
   88: // return number of times tic was called so far
   89: int StringSound::time(){
   90:         return ticCounter;
   91: }
   92:
   93: ostream& operator<<(ostream& os, StringSound& ss)
   94: {
   95:         os << *(ss.rb);
   96:
   97:         return os;
   98: }
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include "RingBuffer.h"
 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(function_testing) {
11:   BOOST_CHECK_THROW(RingBuffer b1(0), std::invalid_argument);
12:   BOOST_CHECK_THROW(RingBuffer b2(-4), std::invalid_argument);
13:   BOOST_CHECK_NO_THROW(RingBuffer b3(10));
14:   BOOST_CHECK_NO_THROW(RingBuffer b4(3));
15:
16:   RingBuffer b(3);
17:   BOOST_REQUIRE(b.isEmpty() == true);
18:   BOOST_REQUIRE(b.isFull() == false);
19:
20:   BOOST_CHECK_THROW(b.dequeue(), std::runtime_error);
21:   BOOST_CHECK_THROW(b.peek(), std::runtime_error);
22:   BOOST_CHECK_THROW(b.peekAt(0), std::runtime_error);
23:   BOOST_CHECK_THROW(std::cout << b, std::runtime_error);
24:   BOOST_CHECK_THROW(b.printBuffer(), std::runtime_error);
25:   BOOST_CHECK_THROW(b.pluckString(), std::runtime_error);
26:
27:   b.enqueue(1);
28:   b.enqueue(2);
29:   b.enqueue(3);
30:   BOOST_CHECK_THROW(b.enqueue(4), std::runtime_error);
31:
32:   BOOST_CHECK_NO_THROW(b.dequeue());
33:   BOOST_CHECK_NO_THROW(b.peek());
34:   BOOST_CHECK_NO_THROW(b.peekAt(0));
35:   BOOST_CHECK_NO_THROW(b.printBuffer());
36:   BOOST_CHECK_NO_THROW(std::cout << b);
37:
38:   BOOST_REQUIRE(b.size() == 2);
39:   b.enqueue(4);
40:   BOOST_REQUIRE(b.size() == 3);
41:   BOOST_REQUIRE(b.isFull() == true);
42:   BOOST_REQUIRE(b.isEmpty() == false);
43:
44:   BOOST_REQUIRE(b.peek() == 2);
45:   BOOST_REQUIRE(b.dequeue() == 2);
46:   BOOST_REQUIRE(b.peekAt(1) == 4);
47:   BOOST_REQUIRE(b.dequeue() == 3);
48:   BOOST_REQUIRE(b.dequeue() == 4);
49:   BOOST_REQUIRE(b.size() == 0);
50:   BOOST_REQUIRE(b.isEmpty() == true);
51:
52:   b.enqueue(3);
53:   b.enqueue(7);
54:   b.enqueue(11);
55:   BOOST_REQUIRE(b.pluckString() == (int16_t)(0.996 * ((3 + 7) / 2.0)));
56:   BOOST_REQUIRE(b.pluckString() == (int16_t)(0.996 * ((7 + 11) / 2.0)));
57:   BOOST_CHECK_NO_THROW(b.pluckString());
58: }
```

PS5 DNA Sequence Alignment

Overview:

This program calculates the optimal sequence alignment of two DNA strings. A major part of this assignment was analyzing the time and space usage with Valgrind.

Implementation:

This program is implemented with a dynamic programming approach that creates a N by M matrix, N and M being the sizes of the DNA strings. This matrix is used to find the edit distance and the optimal alignment.

What I Learned:

- How to use dynamic programming.
- How to display my program execution time.
- How to analyze the results of running valgrind to find memory usage of the program.
- How to calculate the largest input my program could theoretically handle with 8GB of RAM, or if it had to 24 hours to execute.

Output and Code:

```
osboxes@osboxes:~/Documents/ps5$ ./ED < 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 4.4e-05 seconds.
```

Makefile Sun Apr 19 14:36:06 2020 1

```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -ansi -pedantic -std=c++11 -g -O2
 3: LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
 4: BOOST = -lboost_unit_test_framework
 5: OBJECTS = ED.o main.o
 6:
 7: all: ED
 8:
 9: ED: $(OBJECTS)
10:         $(CC) $(OBJECTS) -o ED $(LFLAGS)
11:
12: ED.o:ED.cpp ED.hpp
13:         $(CC) -c ED.cpp $(CFLAGS)
14:
15: main.o:main.cpp ED.hpp
16:         $(CC) -c main.cpp ED.hpp $(LFLAGS)
17:
18: clean:
19:         rm $(OBJECTS) ED *\~
```

main.cpp Sat Apr 18 18:57:36 2020 1

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include <SFML/System.hpp>
 6: #include <string>
 7: #include "ED.hpp"
 8:
 9: int main(int argc, char* argv[]) {
10:   sf::Clock clock;
11:   sf::Time t;
12:
13:   string input1, input2;
14:   std::cin >> input1 >> input2;
15:
16:   ED test(input1, input2);
17:   std::cout <<"Edit distance = " << test.OptDistance() << "\n";
18:   // test.printMatrix();
19:   std::cout << test.Alignment();
20:
21:   t = clock.getElapsedTime();
22:   std::cout << "Execution time is " << t.asSeconds() << " seconds.\n";
23:
24:   return 0;
25: }
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include <iostream>
 6: #include <string>
 7: #include <vector>
 8:
 9: using std::string;
10: using std::vector;
11:
12: class ED {
13: public:
14:
15:   ED(string s1, string s2);  // Constructor
16:
17:   // Returns penalty for aligning chars a and b (will be 0 or 1)
18:   static int penalty(char a, char b);
19:
20:   // Returns the minimum of the three agruments
21:   static int minimum(int a, int b, int c);
22:
23:   int OptDistance();
24:
25:   string Alignment();
26:
27:   void printMatrix();
28:
29: private:
30:   string str1;
31:   string str2;
32:   int rowSize;
33:   int colSize;
34:
35:   vector<vector<int>> opt;
36: };
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include <algorithm>
 6: #include <string>
 7: #include "ED.hpp"
 8:
 9: ED::ED(string s1, string s2) {
10:   str1 = s1;
11:   str2 = s2;
12:
13:   rowSize = str1.size() + 1;
14:   colSize = str2.size() + 1;
15:   opt.resize(rowSize);
16:
17:   int i;
18:   for (i = 0; i < rowSize; i++) {
19:     opt[i].resize(colSize);
20:   }
21:
22:   unsigned int x, y;
23:   for (x = 0; x < opt.capacity(); x++) {
24:     for (y = 0; y < opt[x].capacity(); y++) {
25:       opt[x][y] = 0;  // fill whole matrix with 0s
26:     }
27:   }
28: }
29:
30: int ED::penalty(char a, char b) {
31:   if (a == b)
32:     return 0;
33:   else
34:     return 1;
35: }
36:
37: int ED::minimum(int a, int b, int c) {
38:   return std::min({ a, b, c });
39: }
40:
41: int ED::OptDistance() {
42:   int x, y;
43:   for (x = 0; x < rowSize; x++) {
44:     opt[x][colSize - 1] = (rowSize - 1 - x) * 2;
45:   }
46:   for (y = 0; y < colSize; y++) {
47:     opt[rowSize - 1][y] = (colSize - 1 - y) * 2;
48:   }
49:
50:   int counter = 2;
51:   for (x = rowSize - 2; x >= 0; x--) {
52:     opt[x][colSize - 1] = counter;  // fill right column
53:     counter += 2;
54:   }
55:
56:   counter = 2;
57:   for (y = colSize - 2; y >= 0; y--) {
58:     opt[rowSize - 1][y] = counter;  // fill bottom row
59:     counter += 2;
60:   }
61:
```

```
   62:   int num1, num2, num3;
   63:   for (x = opt.size() - 2; x >= 0 ; x--) {
   64:     for (y = opt[x].size() - 2; y >= 0; y--) {
   65:       num1 = opt[x + 1][y + 1] + penalty(str1.at(x), str2.at(y));
   66:       num2 = opt[x + 1][y] + 2;
   67:       num3 = opt[x][y + 1] + 2;
   68:
   69:       opt[x][y] = minimum(num1, num2, num3);
   70:     }
   71:   }
   72:
   73:   return opt[0][0];
   74: }
   75:
   76: string ED::Alignment() {
   77:   string output = "\0";
   78:   int i = 0;
   79:   int j = 0;
   80:   int p = 0;
   81:   int case1, case2, case3;
   82:   while (i < rowSize - 1 || j < colSize - 1) {
   83:     try {
   84:       p = penalty(str1.at(i), str2.at(j));
   85:       case1 = opt.at(i + 1).at(j + 1) + p;
   86:     }
   87:     catch(const std::out_of_range& err) {
   88:       case1 = -1;  // Out of range
   89:     }
   90:     try {
   91:       case2 = opt.at(i + 1).at(j) + 2;
   92:     }
   93:     catch(const std::out_of_range& err) {
   94:       case2 = -1;
   95:     }
   96:     try {
   97:       case3 = opt.at(i).at(j + 1) + 2;
   98:     }
   99:     catch(const std::out_of_range& err) {
  100:       case3 = -1;
  101:     }
  102:
  103:     string pen;
  104:     if (p == 0) {
  105:       pen = '0';
  106:     } else {
  107:       pen = '1';
  108:     }
  109:
  110:     if (opt[i][j] == case1) {
  111:       output += str1.at(i) + string("  ") + str2.at(j) + string("  ") + pen;
  112:       i++;
  113:       j++;
  114:     } else if (opt[i][j] == case2) {
  115:       output += str1.at(i) + string("  -  2");
  116:       i++;
  117:     } else if (opt[i][j] == case3) {
  118:       output += string("-  ") + str2.at(j) + string("  2");
  119:       j++;
  120:     }
  121:
  122:     output += "\n";
```

```
123:   }
124:
125:   return output;
126: }
127:
128: void ED::printMatrix() {
129:   std::cout << "\t";
130:   unsigned int w, x, y;
131:   for (w = 0; w < str2.size(); w++) {
132:     std::cout << str2.at(w) << "\t";
133:   }
134:   std::cout << "-\n";
135:
136:   for (x = 0; x < opt.size(); x++) {
137:     if (x == opt.size() - 1) {
138:       std::cout << "-\t";
139:     } else {
140:       std::cout << str1.at(x) << "\t";
141:     }
142:     for (y = 0; y < opt[x].size(); y++) {
143:       std::cout << opt[x][y] << "\t";
144:     }
145:     std::cout << "\n\n";
146:   }
147:   std::cout << "\n";
148: }
```

PS6 Markov Model of Natural Language

Overview:

This program creates a Markov Model to produce a probabilistic model of any given text. The model is used to generate random text that is surprisingly reasonable.

Implementation:

This program was implemented with a map, since it has a key and a value. The key was a string that represented a kgram, and the value was an integer representing the frequency of the associated kgram in the given text. With this information, the probability of letters following each kgram can be calculated.

What I Learned:

- How to go through a map to find specific elements.
- How to create a symbol table.
- How to make a string circular (wraps around).

Output and Code:

```
osboxes@osboxes:~/Documents/ps6$ ./TextGenerator 12 100 < amendments.txt
Input:
 Article I. Congress shall make no law respecting an establishment of religion,
 or prohibiting the f

Output:
 Article I. Congress shall have died, the Vice President, to be electors appoin
ted; and if no person
```

Makefile Thu Apr 23 21:05:53 2020 1

```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -ansi -pedantic -std=c++11
 3: LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 4: BOOST = -lboost_unit_test_framework
 5: OBJECTS = MModel.o TextGenerator.o
 6:
 7: all: TextGenerator Test
 8:
 9: TextGenerator: $(OBJECTS)
10:         $(CC) $(OBJECTS) -o TextGenerator $(CFLAGS)
11:
12: Test: MModel.o test.o
13:         $(CC) MModel.o test.o -o Test $(BOOST)
14:
15: MModel.o:MModel.cpp MModel.h
16:         $(CC) -c MModel.cpp $(CFLAGS)
17:
18: TextGenerator.o:TextGenerator.cpp MModel.h
19:         $(CC) -c TextGenerator.cpp MModel.h $(CFLAGS)
20:
21: test.o:test.cpp MModel.h
22:         $(CC) -c test.cpp MModel.h $(CFLAGS)
23:
24: clean:
25:         rm $(OBJECTS) test.o TextGenerator Test *\~ *.gch
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include "MModel.h"
 6: #include <string>
 7:
 8: using std::stoi;
 9:
10: int main(int argc, char* argv[]) {
11:   int k = stoi(argv[1]);
12:   int L = stoi(argv[2]);
13:   string input = "";
14:   string text;
15:   while (std::cin >> text) {
16:     input += " " + text;
17:   }
18:
19:   MModel mm(input, k);
20:
21:   string output = mm.generate(input.substr(0, k), L);
22:
23:   std::cout << "Input:\n";
24:   int i;
25:   for (i = 0; i < L; i++) {
26:     std::cout << input[i];
27:   }
28:
29:   std::cout << "\n\nOutput:\n";
30:   std::cout << output << "\n";
31:
32:   return 0;
33: }
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #ifndef MMODEL_H
 6: #define MMODEL_H
 7:
 8: #include <iostream>
 9: #include <string>
10: #include <map>
11:
12: using std::string;
13: using std::ostream;
14: using std::map;
15:
16: class MModel{
17:  public:
18:   // create a Markov model of order k from given text
19:   // Assume that text has length at least k.
20:   MModel(string text, int k);
21:
22:   int kOrder();  // order k of Markov model
23:
24:   // number of occurrences of kgram in text
25:   // (throw an exception if kgram is not of length k)
26:   int freq(string kgram);
27:
28:   // number of times that character c follows kgram
29:   // if order=0, return num of times char c appears
30:   // (throw an exception if kgram is not of length k)
31:   int freq(string kgram, char c);
32:
33:   // random character following given kgram
34:   // (Throw an exception if kgram is not of length k.
35:   // Throw an exception if no such kgram.)
36:   char kRand(string kgram);
37:
38:   // generate a string of length L characters
39:   // by simulating a trajectory through the corresponding
40:   // Markov chain.  The first k characters of the newly
41:   // generated string should be the argument kgram.
42:   // Throw an exception if kgram is not of length k.
43:   // Assume that L is at least k.
44:   string generate(string kgram, int L);
45:
46:   // overload the stream insertion operator and display
47:   // the internal state of the Markov Model. Print out
48:   // the order, the alphabet, and the frequencies of
49:   // the k-grams and k+1-gram.
50:   friend ostream& operator<<(ostream& out, MModel& mm);
51:
52:  private:
53:   string wrappedText;
54:   string letters;
55:   unsigned int order;
56:   map<string, int> kgrams;
57: };
58: #endif
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include "MModel.h"
 6: #include <algorithm>
 7: #include <random>
 8: #include <string>
 9: #include <map>
10: #include <utility>
11:
12: MModel::MModel(string text, int k) {
13:   order = k;
14:
15:   wrappedText = text;
16:   unsigned int i;
17:   for (i = 0; i < order - 1; i++) {
18:     wrappedText.push_back(text.at(i));  // makes text circular
19:   }
20:
21:   string tempStr;
22:   // finds kgrams and puts them in the kgrams map
23:   for (i = 0; i < text.size(); i++) {
24:     tempStr = wrappedText.substr(i, order);
25:     kgrams.insert(std::pair<string, int>(tempStr, 0));  // 0 freq for now
26:     tempStr.clear();
27:   }
28:
29:   map<string, int>::iterator it;
30:   int counter = 0;
31:   // finds corresponding kgram frequencies
32:   for (i = 0; i < text.size(); i++) {
33:     tempStr = wrappedText.substr(i, order);
34:
35:     it = kgrams.find(tempStr);
36:     counter = it->second;
37:     counter++;
38:
39:     kgrams[tempStr] = counter;
40:
41:     tempStr.clear();
42:   }
43:
44:   unsigned int j;
45:   char tempChar;
46:   bool repeat;
47:   // finds letters used in text
48:   for (i = 0; i < text.size(); i++) {
49:     repeat = false;
50:     tempChar = text.at(i);
51:
52:     for (j = 0; j < letters.size(); j++) {
53:       if (tempChar == letters.at(j)) {
54:         repeat = true;
55:         break;
56:       }
57:     }
58:
59:     if (repeat == false) {
60:       letters.push_back(tempChar);
61:     }
```

```
   62:   }
   63:
   64:   // put letters in order
   65:   std::sort(letters.begin(), letters.end());
   66: }
   67:
   68: int MModel::kOrder() {
   69:   return order;
   70: }
   71:
   72: int MModel::freq(string kgram) {
   73:   if (kgram.size() != order)
   74:     throw std::runtime_error("freq: kgram is not of length k");
   75:
   76:   map<string, int>::iterator it;
   77:   it = kgrams.find(kgram);
   78:
   79:   if (it == kgrams.end())  // did not find kgram
   80:     return 0;
   81:
   82:   return it->second;
   83: }
   84:
   85: int MModel::freq(string kgram, char c) {
   86:   if (kgram.size() != order)
   87:     throw std::runtime_error("freq: kgram is not of length k");
   88:
   89:   unsigned int x;
   90:   string temp;
   91:   int freqCounter = 0;
   92:   for (x = 0; x < wrappedText.size() - order + 1; x++) {
   93:     temp = wrappedText.substr(x, order);
   94:
   95:     if (temp == kgram) {
   96:       if (x == wrappedText.size() - order) {
   97:         if (wrappedText.at(order - 1) == c) {
   98:           freqCounter++;
   99:         }
  100:       } else {
  101:         if (wrappedText.at(x + order) == c) {
  102:           freqCounter++;
  103:         }
  104:       }
  105:     }
  106:
  107:     temp.clear();
  108:   }
  109:
  110:   return freqCounter;
  111: }
  112:
  113: char MModel::kRand(string kgram) {
  114:   if (kgram.size() != order)
  115:     throw std::runtime_error("kRand: kgram is not of length k");
  116:
  117:   map<string, int>::iterator it;
  118:   it = kgrams.find(kgram);
  119:
  120:   if (it == kgrams.end())  // did not find kgram
  121:     throw std::runtime_error("kRand: could not find kgram");
  122:
```

```
123:   int freqKgram = freq(kgram);
124:
125:   string tempStr;
126:   unsigned int x;
127:   int y;
128:   for (x = 0; x < letters.size(); x++) {
129:     for (y = 0; y < freq(kgram, letters.at(x)); y++) {
130:       tempStr.push_back(letters.at(x));
131:     }
132:   }
133:
134:   // Seed with a real random value, if available
135:   std::random_device r;
136:
137:   // Choose a random mean between 0 and kgram freq - 1
138:   std::default_random_engine e1(r());
139:   std::uniform_int_distribution<int> uniform_dist(0, freqKgram - 1);
140:   int randNum = uniform_dist(e1);
141:
142:   return tempStr.at(randNum);
143: }
144:
145: string MModel::generate(string kgram, int L) {
146:   if (kgram.size() != order)
147:     throw std::runtime_error("generate: kgram is not of length k");
148:
149:   string retStr = kgram;
150:
151:   char randChar;
152:
153:   unsigned int i;
154:   for (i = 0; i < L - order; i++) {
155:     randChar = kRand(retStr.substr(i, order));
156:
157:     retStr.push_back(randChar);
158:   }
159:
160:   return retStr;
161: }
162:
163: ostream& operator<<(ostream& out, MModel& mm) {
164:   out << "Text: ";
165:   out << mm.wrappedText.substr(0, mm.wrappedText.size() - mm.order + 1);
166:   out << "\n";
167:   out << "Order: " << mm.order << "\n";
168:   out << "Letters: ";
169:   unsigned int i;
170:   for (i = 0; i < mm.letters.size(); i++) {
171:     out << mm.letters.at(i) << "  ";
172:   }
173:   out << "\n\n";
174:
175:   out << "kgram\tfreq\tfreq of next char\tprob next char is\n";
176:   out << "\t\t";
177:   for (i = 0; i < mm.letters.size(); i++) {
178:     out << mm.letters.at(i) << "  ";
179:   }
180:   out << "\t\t";
181:   for (i = 0; i < mm.letters.size(); i++) {
182:     out << mm.letters.at(i) << "    ";
183:   }
```

```
184:   out << "\n";
185:
186:   map<string, int>::iterator it;
187:   unsigned int j;
188:   for (it = mm.kgrams.begin(); it != mm.kgrams.end(); ++it) {
189:     out << it->first << "\t" << it->second << "\t";
190:
191:     for (j = 0; j < mm.letters.size(); j++) {
192:       out << mm.freq(it->first, mm.letters.at(j)) << "  ";
193:     }
194:
195:     out << "\t\t";
196:     for (j = 0; j < mm.letters.size(); j++) {
197:       if (mm.freq(it->first, mm.letters.at(j)) == 0) {
198:         out << "0    ";
199:       } else if (mm.freq(it->first, mm.letters.at(j)) == it->second) {
200:         out << "1    ";
201:       } else {
202:         out << mm.freq(it->first, mm.letters.at(j));
203:         out << "/" << it->second << "  ";
204:       }
205:     }
206:
207:     out << "\n";
208:   }
209:
210:   return out;
211: }
```

```
 1: /*
 2:   Copyright 2020 Peyton Somerville
 3: */
 4:
 5: #include "MModel.h"
 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(function_testing) {
11:   MModel test("bbbabbabbbbaba", 3);
12:   std::cout << test << "\n";
13:
14:   BOOST_CHECK_THROW(test.freq("ab"), std::runtime_error);
15:   BOOST_CHECK_THROW(test.freq("ab", 'b'), std::runtime_error);
16:   BOOST_CHECK_THROW(test.kRand("ab"), std::runtime_error);
17:   BOOST_CHECK_THROW(test.kRand("abc"), std::runtime_error);
18:   BOOST_CHECK_NO_THROW(test.freq("abb"));
19:   BOOST_CHECK_NO_THROW(test.freq("abb", 'b'));
20:   BOOST_CHECK_NO_THROW(test.kRand("abb"));
21:
22:   BOOST_REQUIRE(test.freq("abb") == 3);
23:   BOOST_REQUIRE(test.freq("bab") == 4);
24:   BOOST_REQUIRE(test.freq("aba") == 1);
25:   BOOST_REQUIRE(test.freq("abb", 'a') == 1);
26:   BOOST_REQUIRE(test.freq("abb", 'b') == 2);
27:   BOOST_REQUIRE(test.freq("bab", 'a') == 1);
28:   BOOST_REQUIRE(test.freq("bab", 'b') == 3);
29:
30:   MModel test2("gagggagaggcgagaaa", 2);
31:   std::cout << test2 << "\n";
32:   BOOST_REQUIRE(test2.freq("ag") == 5);
33:   BOOST_REQUIRE(test2.freq("gg") == 3);
34:   BOOST_REQUIRE(test2.freq("gc") == 1);
35:   BOOST_REQUIRE(test2.freq("ag", 'a') == 3);
36:   BOOST_REQUIRE(test2.freq("ag", 'c') == 0);
37:   BOOST_REQUIRE(test2.freq("ag", 'g') == 2);
38:   BOOST_REQUIRE(test2.freq("gg", 'a') == 1);
39:   BOOST_REQUIRE(test2.freq("gg", 'c') == 1);
40:   BOOST_REQUIRE(test2.freq("gg", 'g') == 1);
41: }
```