Elick Coval
COMP IV: Project Portfolio
Spring 2019

Contents:

PS0 Hello World with SFML

PS1 Linear Feedback Shift Register and Image Encoding

PS2 Recursive Graphics (Pythagoras tree)

PS3 N-Body Simulation

PS4 Airport

PS5 Ring Buffer and Guitar Hero

PS0 Hello World with SFML:

The main purpose of this assignment was to get us up and running with our environments and the SFML library. We were tasked with displaying a window with a green circle and then asked to display an image and have it respond to keystrokes. The image I chose was one of my cat Mia:

I was able to fully complete this assignment. I also implemented an additional feature which would rotate the image when the space bar was depressed on line 43.

This assignment was pretty straight forward, I had to review the SFML documentation and figure out how to open a window with the dimensions I needed. I learned how SFML handles images it calls “sprites” and how to manipulate them. It was also interesting how the motion was shown by redrawing the image over and over again in a loop until the window is closed.

```
 1: CC = g++
 2: CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -ansi -pedantic
 3: OBJ = main.o
 4: DEPS =
 5: LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 6: EXE = Ps0
 7:
 8: all: $(OBJ)
 9:         $(CC) $(OBJ) -o $(EXE) $(LIBS)
10:
11: main.o: main.cpp
12:         $(CC) $(CFLAGS) -c $<
13:
14: clean:
15:         rm $(OBJ) $(EXE)
```

```
 1: /*
 2: Elick Coval
 3: 1/28/2019
 4: SFML Hello-world
 5: */
 6:
 7: #include <SFML/Graphics.hpp>
 8:
 9: int main()
10: {
11:     sf::RenderWindow window(sf::VideoMode(400, 400), "Voila!");
12:     sf::CircleShape shape(200.f);
13:     shape.setFillColor(sf::Color::Green);
14:
15:     sf::Texture texture;
16:     if (!texture.loadFromFile("sprite.png"))
17:         return EXIT_FAILURE;
18:     sf::Sprite sprite(texture);
19:     int spriteX = -75, spriteY = -100;
20:     sprite.setOrigin(spriteX, spriteY);
21:
22:     while (window.isOpen())
23:     {
24:         sf::Event event;
25:
26:         while (window.pollEvent(event))
27:         {
28:             if (event.type == sf::Event::Closed)
29:                 window.close();
30:         }
31:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
32:                 sprite.move(0,-.25);
33:
34:                 if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
35:                 sprite.move(0, .25);
36:
37:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
38:                 sprite.move(-0.25, 0);
39:
40:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
41:                 sprite.move(0.25, 0);
42:
43:         if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space))
44:                 sprite.rotate(5);
45:
46:
47:         window.clear();
48:         window.draw(shape);
49:         window.draw(sprite);
50:         window.display();
51:     }
52:
53:     return 0;
54: }
```

PS1 Linear Feedback Shift Register and Image Encoding:

This assignment was split into two parts. First we had to write a program that produced pseudo-random bits by simulating a linear feedback shift register, and then use it to implement a form of encryption for digital pictures. A linear feedback shift register, or LFSR, is a register that takes a linear function of a previous state as input. Using a Boolean exclusive OR, LFSR performs discrete step operations that:
1) Shifts the bits one position to the left
2) Replaces the vacated bit by the exclusive or of the bit shifted off and the bit previously at a given tap position in the register.

A LFSR has three parameters that characterize the sequence of bits it produces: the number of bits N, the initial seed (the sequence of bits that initializes the register), and the tap position tap. The following illustrates one step of an 11-bit LFSR with initial seed 01101000010 and tap position 8.

It was fun learning how rudimentary encryption works and how to perform unit tests using the boost test framework for this assignment. I was able to add two additional tests which focused on different bit and tap position scenarios.

The image encoding/decoding was a little tricky for me. I was able to encode to static and decode back, showing the LFSR works. But I wasn't able to negate a portion of the image like the assignment called for. I am fairly certain the problem lies between lines 13 and 22 of PhotoMagic.cpp.

```
    1: CC = g++
    2: OFLAGS = -c -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_frame
work
    3: CFLAGS = -Wall -ansi -pedantic -Werror -std=c++0x -lboost_unit_test_framewor
k
    4: LFLAGS = -lsfml-window -lsfml-graphics -lsfml-system
    5:
    6: all: ps1b
    7:
    8: ps2b: LFSR.o PhotoMagic.o
    9:         $(CC) PhotoMagic.o LFSR.o $(CFLAGS) $(LFLAGS)  -o PhotoMagic
   10:
   11: PhotoMagic.o: PhotoMagic.cpp
   12:         $(CC) $(OFLAGS) $(LFLAGS) PhotoMagic.cpp
   13:
   14: LFSR.o: LFSR.cpp LFSR.hpp
   15:         $(CC) $(OFLAGS) LFSR.cpp
   16:
   17:
   18: clean:
   19:         \rm -f *.o  *~ PhotoMagic ps1b
```

```
 1: #include <SFML/System.hpp>
 2: #include <SFML/Window.hpp>
 3: #include <SFML/Graphics.hpp>
 4: #include <iostream>
 5: #include <string>
 6: #include <algorithm>
 7: #include "LFSR.hpp"
 8:
 9: sf::Image transform(sf::Image picture, LFSR lfsr)
10: {
11:   sf::Vector2u size = picture.getSize();
12:   sf::Color p2;
13:   for (size_t x = 0; x < size.x; x++) {
14:     for (size_t y = 0; y < size.y; y++) {
15:       p2 = picture.getPixel(x, y);
16:       p2.r = p2.r ^ lfsr.generate(8);
17:       p2.g = p2.g ^ lfsr.generate(8);
18:       p2.b = p2.b ^ lfsr.generate(8);
19:       picture.setPixel(x, y, p2);
20:     }
21:   }
22:   return picture;
23: }
24:
25: int main(int argc, char *argv[])
26: {
27:   std::string fi, out;
28:   if (argc < 3) {
29:     std::cout << "Usage: " << argv[0] << " inputfile LFSR bit" << std::endl;
30:     exit(0);
31:   } else {
32:     fi = argv[1];
33:     out = argv[2];
34:   }
35:   sf::Image image;
36:   if (!image.loadFromFile(fi))
37:     return -1;
38:   sf::Image imageOut;
39:   if (!imageOut.loadFromFile(fi))
40:     return -1;
41:   LFSR lfsr(argv[3], atoi(argv[4]));
42:   imageOut = transform(imageOut, lfsr);
43:   sf::Vector2u size = image.getSize();
44:   sf::RenderWindow window(sf::VideoMode(size.x * 2, size.y), "Meow");
45:   if (!imageOut.saveToFile(fi))
46:     return -1;
47:   sf::Texture texture;
48:   texture.loadFromImage(image);
49:   sf::Sprite sprite;
50:   sprite.setTexture(texture);
51:   sf::Texture textureOut;
52:   textureOut.loadFromImage(imageOut);
53:   sf::Sprite spriteOut;
54:   spriteOut.setTexture(textureOut);
55:   spriteOut.setPosition(size.x, 0);
56:   while (window.isOpen()) {
57:     sf::Event event;
58:     while (window.pollEvent(event)) {
59:       if (event.type == sf::Event::Closed)
60:         window.close();
61:     }
```

```
   62:     window.clear(sf::Color::White);
   63:     window.draw(sprite);
   64:     window.draw(spriteOut);
   65:     window.display();
   66:   }
   67:
   68:
   69:
   70:   if (!imageOut.saveToFile(out))
   71:     return -1;
   72:   return 0;
   73: }
```

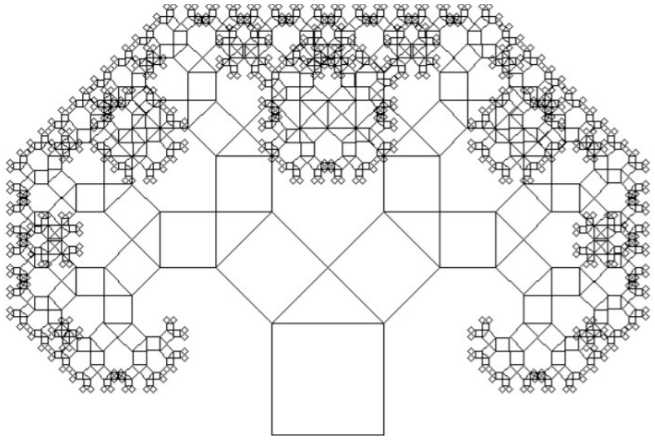
```
 1: #ifndef LFSR_HPP_
 2: #define LFSR_HPP_
 3:
 4: #include <iostream>
 5: #include <string>
 6:
 7: using namespace std;
 8:
 9: class LFSR{
10:     private:
11:         int length;
12:         int tap;
13:         string _data;
14:
15: public:
16:
17:     LFSR(std::string seed, int t);
18:     int step();
19:     int generate(int k);
20:     friend ostream& operator << (ostream& out, LFSR& lfsr);
21: };
22:
23: #endif
```

```
 1: #include <iostream>
 2: #include <cmath>
 3: #include "LFSR.hpp"
 4:
 5: LFSR::LFSR(std::string seed, int t)
 6: {
 7:     _data = seed;
 8:     length = _data.size();
 9:     tap = t;
10: }
11:
12: int LFSR::step()
13: {
14:     int bit;
15:     int length = _data.size();
16:     std::string s_bit;
17:     bit = _data.front() ^ _data[length - tap - 1];
18:     s_bit = std::to_string(bit);
19:     _data.erase(0,1);
20:     _data = _data + s_bit;
21:     return bit;
22: }
23:
24: int LFSR::generate(int k)
25: {
26:     int count = 0;
27:     for (int i = k - 1 ;i >= 0; i--)
28:     {
29:         if(step() == 1)
30:             count+= pow(2, i);
31:     }
32:     return count;
33: }
34:
35: std::ostream& operator << (std::ostream& out, LFSR& lfsr)
36: {
37:     out << lfsr._data;
38:     return out;
39: }
```

```
 1: #include <iostream>
 2: #include <string>
 3:
 4: #include "LFSR.hpp"
 5:
 6: #define BOOST_TEST_DYN_LINK
 7: #define BOOST_TEST_MODULE Main
 8: #include <boost/test/included/unit_test.hpp>
 9:
10: BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo){
11:
12:     LFSR l("00111", 2);
13:     BOOST_REQUIRE(l.step() == 1);
14:     BOOST_REQUIRE(l.step() == 1);
15:     BOOST_REQUIRE(l.step() == 0);
16:     BOOST_REQUIRE(l.step() == 0);
17:     BOOST_REQUIRE(l.step() == 0);
18:     BOOST_REQUIRE(l.step() == 1);
19:     BOOST_REQUIRE(l.step() == 1);
20:     BOOST_REQUIRE(l.step() == 0);
21:
22:     LFSR l2("00111", 2);
23:     BOOST_REQUIRE(l2.generate(8) == 198);
24: }
25:
26: BOOST_AUTO_TEST_CASE(thirtyTwoBitsTapAtOne){
27:
28:     LFSR l("01101101101101101101101101101101", 1);
29:     BOOST_REQUIRE(l.step() == 0);
30:     BOOST_REQUIRE(l.step() == 0);
31:     BOOST_REQUIRE(l.step() == 1);
32:     BOOST_REQUIRE(l.step() == 0);
33:     BOOST_REQUIRE(l.step() == 0);
34:     BOOST_REQUIRE(l.step() == 1);
35:     BOOST_REQUIRE(l.step() == 0);
36:     BOOST_REQUIRE(l.step() == 0);
37: }
38:
39: BOOST_AUTO_TEST_CASE(threeBitTapAtZero){
40:
41:     LFSR l("100",0);
42:     BOOST_REQUIRE(l.step() == 1);
43:     BOOST_REQUIRE(l.step() == 1);
44:     BOOST_REQUIRE(l.step() == 1);
45:     BOOST_REQUIRE(l.step() == 0);
46:     BOOST_REQUIRE(l.step() == 1);
47:     BOOST_REQUIRE(l.step() == 0);
48:     BOOST_REQUIRE(l.step() == 0);
49:     BOOST_REQUIRE(l.step() == 1);
50:
51:     LFSR l2("100", 0);
52:     BOOST_REQUIRE(l2.generate(8) == 233);
53: }
```

PS2 Recursive Graphics (Pythagoras tree):

For this assignment we were tasked with writing a program that plots a Pythagoras tree using a square as a base like the one below:



Our program was supposed to take in two integer command-line arguments L (size of the base square) and N (depth of the recursion).

This assignment was little more challenging than the preceding assignments. The recursion and the geometry took a little while to wrap my brain around but I started by calculating the left and right vertices and center point (Lines 32-42 of PTree.cpp). Then, I use one of the new vertices and the center point to generate a new child square which becomes the parent of the next pair of child squares (Lines 44-61 of PTree.cpp). This continues recursively until it reaches the target depth.

This assignment forced me to get a little more familiar with recursion and see how it can be used with graphics to create some neat effects. It doesn't show up in a static image but I was able to randomize the colors assigned to each parent on the way back up. The effect is the colors are continuously changing, creating a disco ball-like effect. This is due to the fact that the window is always in a loop continuously redrawing the squares with a new color each time through.

```
 1: CC = g++
 2: CFLAGS = -std=c++11 -c -g -Og -Wall -pedantic
 3: OBJ = PTree.o main.o
 4: DEPS =
 5: LIBS = -lsfml-graphics -lsfml-window -lsfml-system
 6: EXE = PTree
 7:
 8: all: $(OBJ)
 9:         $(CC) $(OBJ) -o $(EXE) $(LIBS)
10:
11: Main.o: main.cpp
12:         $(CC) $(CFLAGS) -c $<
13:
14: PTree.o: PTree.cpp PTree.hpp
15:         $(CC) $(CFLAGS) -c $<
16:
17: PTree: $(OBJ)
18:         $(cc) $(OBJ) -o ps2
19:
20: clean:
21:         rm $(OBJ) $(EXE)
```

```
 1: #include <SFML/Graphics.hpp>
 2: #include <SFML/Window.hpp>
 3: #include "PTree.hpp"
 4:
 5: using namespace sf;
 6:
 7: int main(int argc, char** argv) {
 8:
 9:         int size = std::stoi(argv[1]);
10:         int depth = std::stoi(argv[2]);
11:
12:         sf::RenderWindow window(sf::VideoMode(6 * size, 4 * size), "Pythagor
as Tree");
13:
14:         sf::ConvexShape square(4);
15:         square.setPoint(0, sf::Vector2f(0, 0));
16:         square.setPoint(1, sf::Vector2f(size, 0));
17:         square.setPoint(2, sf::Vector2f(size, size));
18:         square.setPoint(3, sf::Vector2f(0, size));
19:         square.setPosition(2.5 * size, 3 * size);
20:
21:         sf::Vector2f origin = square.getOrigin();
22:         sf::Vector2f newOrigin = sf::Vector2f(size / 2, size / 2);
23:         sf::Vector2f offset = newOrigin - origin;
24:         square.setOrigin(newOrigin);
25:         square.move(offset);
26:
27:         PTree* pTree = new PTree(square);
28:         pTree->btmLeft = square.getPoint(3);
29:         pTree->btmRight = square.getPoint(2);
30:
31:         while (window.isOpen()) {
32:
33:                 pTree->depth = depth;
34:                 pTree->parentSquare.setFillColor(sf::Color::Green);
35:
36:                 // check all the triggered window events
37:                 sf::Event event;
38:                 while (window.pollEvent(event)) {
39:                         // close the window
40:                         if (event.type == sf::Event::Closed)
41:                                 window.close();
42:                 }
43:                 pTree->pTreeRec(window);
44:                 window.display();
45:         }
46:
47:         return 0;
48: }
```

```
 1: #ifndef PTREE_HPP_
 2: #define PTREE_HPP_
 3: #include <SFML/Graphics.hpp>
 4: #include <SFML/Window.hpp>
 5: #include <iostream>
 6:
 7: using namespace sf;
 8:
 9: class PTree : public sf::Drawable, sf::Transformable {
10: public:
11:         PTree(const ConvexShape& baseSquare);
12:
13:         void draw(sf::RenderTarget& target, sf::RenderStates states) const;
14:         void pTreeRec(sf::RenderWindow &window);
15:
16:         int depth;
17:         sf::ConvexShape parentSquare;
18:         sf::Vector2f btmLeft;
19:         sf::Vector2f btmRight;
20: };
21:
22: #endif
```

```
 1: #define _USE_MATH_DEFINES
 2: #include <iostream>
 3: #include <cmath>
 4: #include "PTree.hpp"
 5:
 6: using namespace sf;
 7:
 8: PTree::PTree(const ConvexShape& baseSquare) {
 9:         parentSquare = baseSquare;
10: }
11:
12: void PTree::pTreeRec(sf::RenderWindow &window) {
13:         depth--;
14:
15:         if (depth == -1){
16:                 return;
17:         }
18:
19:         sf::Color newColor = sf::Color();
20:         newColor = parentSquare.getFillColor();
21:
22:         newColor.r = rand();
23:         newColor.b = rand();
24:         newColor.g = rand();
25:
26:         parentSquare.setFillColor(newColor);
27:         sf::RenderStates* renderStates = new RenderStates();
28:         draw(window, *renderStates);
29:
30:         sf::Vector2f topLeft, topRight, coordDiff, midPoint;
31:
32:         coordDiff.x = btmRight.x - btmLeft.x;
33:         coordDiff.y = btmLeft.y - btmRight.y;
34:
35:         topLeft.x = btmLeft.x - coordDiff.y;
36:         topLeft.y = btmLeft.y - coordDiff.x;
37:
38:         topRight.x = btmRight.x - coordDiff.y;
39:         topRight.y = btmRight.y - coordDiff.x;
40:
41:         midPoint.x = topLeft.x + (coordDiff.x - coordDiff.y) / 2;
42:         midPoint.y = topLeft.y - (coordDiff.x + coordDiff.y) / 2;
43:
44:         parentSquare.setPoint(0, btmLeft);
45:         parentSquare.setPoint(1, btmRight);
46:         parentSquare.setPoint(2, topRight);
47:         parentSquare.setPoint(3, topLeft);
48:
49:         auto leftChild = parentSquare;
50:         PTree* leftNode = new PTree(leftChild);
51:         leftNode->depth = depth;
52:         leftNode->btmLeft = midPoint;
53:         leftNode->btmRight = topRight;
54:         leftNode->pTreeRec(window);
55:
56:         auto rChildSquare = parentSquare;
57:         PTree* rightNode = new PTree(rChildSquare);
58:         rightNode->depth = depth;
59:         rightNode->btmLeft = topLeft;
60:         rightNode->btmRight = midPoint;
61:         rightNode->pTreeRec(window);
```

```
62:
63:         return;
64: }
65:
66: void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const
67: {
68:         target.draw(parentSquare, states);
69:         return;
70: }
```

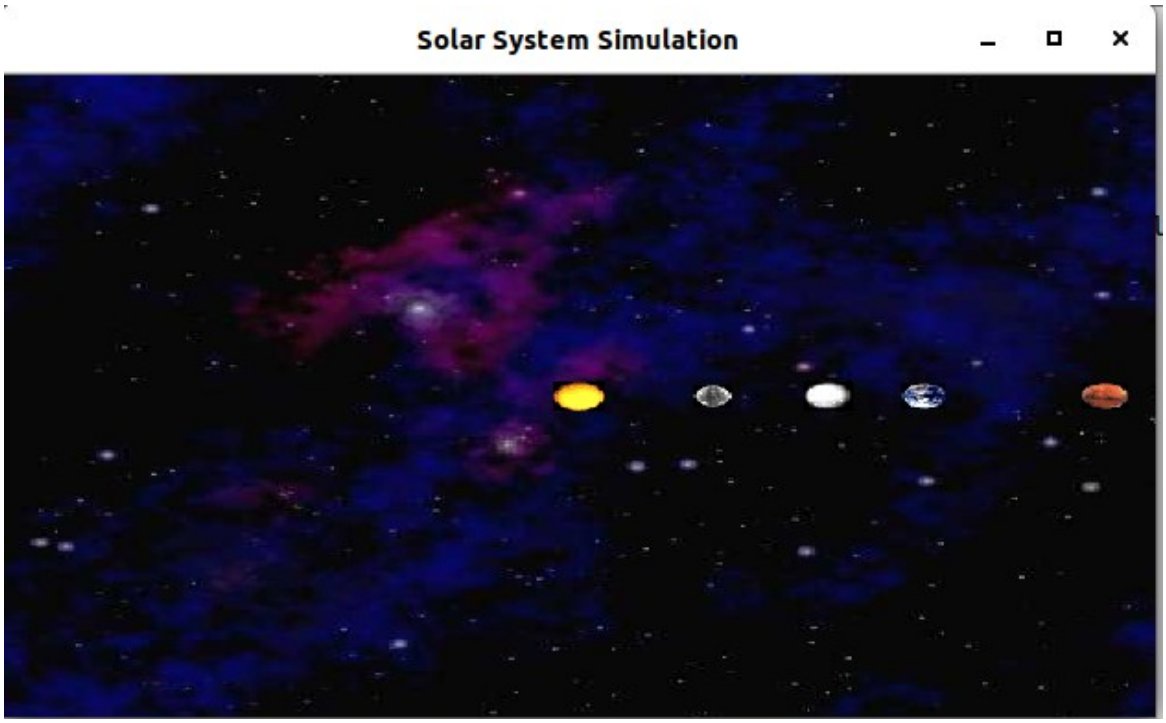
PS3 N-Body Simulation

This assignment was another that was broken up into two parts; the first asked us to read data from a text file and use that data to simulate our solar system graphically, the second introduced physics and had us put the celestial bodies into motion using Newton's law of universal gravitation.

We were tasked with implementing a construct known as an "input redirect." The planets.txt file contained the Sun and first four planets, with the Sun at the center, and the four planets in order toward the right. This involved overloading the input stream operator >> and use it to load parameter data into an object.

Even though it took me a while to come up with the solution, this was one of my favorite assignments. It seems extremely useful to be able to read information in from a file and eventually getting the planets to spin around the sun was very rewarding. It was also rewarding experiencing the value of smart pointers, made life much easier not having to worry about memory leaks.

After overloading the >> operator to allow for the collection of data from the text file, the two main parts of the code happen on lines 30 to 41 of main.cpp where each celestial body's data is read into a vector of shared pointers to a <Body> data type. Later on lines 106 to 110 of main.cpp, those Body objects are drawn and displayed. I was also able to get the background changed.



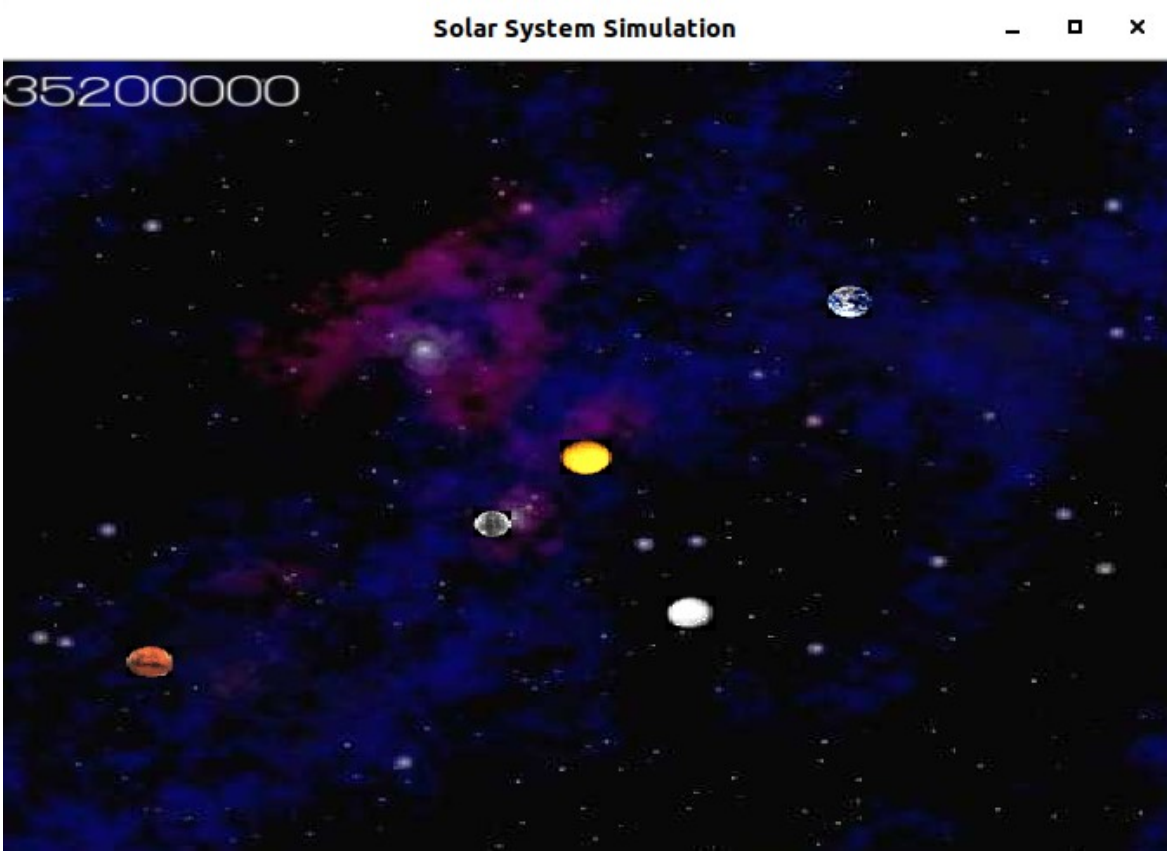
The second part was quite difficult as it required a solid understanding of some physics principles, mainly Newton's law of universal gravitation. It states that every

particle attracts every other particle in the universe with a force which is directly proportional to the product of their masses and inversely proportional to the square of the distance between their centers. Luckily, planets.txt contained most of this information. We were also given two doubles as input representing the number of particles and the radius of the universe (solar system) so we were able to determine the scaling of the window.

I was able to calculate the acceleration of each particle in the step function of Body.cpp on lines 34 to 42. Which allowed for the correct movement of the Body objects given a passed amount of time.

Also, I needed to find the radius and the force for each body at a specific (current) time, this was done via a nested for loop on lines 77 to 88 of main.cpp.

Finally, I was able to play music and display the elapsed time in a font that I found on the internet named Walkway, which made for a very entertaining finale.



```
 1: CC = g++
 2: CFLAGS = -std=c++11 -c -g -Og -Wall
 3: OBJ = main.o Body.o
 4: DEPS =
 5: LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 6: EXE = NBody
 7:
 8: all: $(OBJ)
 9:         $(CC) $(OBJ) -o $(EXE) $(LIBS)
10:
11: main.o: main.cpp
12:         $(CC) $(CFLAGS) -c $<
13:
14: Body.o: Body.cpp Body.hpp
15:         $(CC) $(CFLAGS) -c $<
16:
17: NBody: $(OBJ)
18:         $(cc) $(OBJ) -o ps3b
19:
20: clean:
21:         rm $(OBJ) $(EXE)
```

```
    1: #include <SFML/Graphics.hpp>
    2: #include <SFML/Window.hpp>
    3: #include <SFML/Audio.hpp>
    4: #include <memory>
    5: #include <cmath>
    6: #include <fstream>
    7: #include "Body.hpp"
    8:
    9: using namespace std;
   10: using namespace sf;
   11:
   12: const double gravity = 6.67 * 1e-11;
   13:
   14: float findRadius(float dX, float dY);
   15: float calculateForce(float massA, float massB, float radius);
   16:
   17: int main(int argc, char** argv) {
   18:
   19:         float T = atoi(argv[1]);
   20:         float dT = atoi(argv[2]);
   21:         int currentT = 0;
   22:         int numBodies;
   23:         float radiusUniverse;
   24:
   25:         cin >> numBodies >> radiusUniverse;
   26:
   27:         sf::Vector2f windowSize(500, 500);
   28:         sf::RenderWindow window(sf::VideoMode(windowSize.x, windowSize.y), "
Solar System Simulation");
   29:
   30:         vector<shared_ptr<Body>> bodies(numBodies);
   31:
   32:         for (int i = 0; i < numBodies; i++) {
   33:                 bodies[i] = make_shared<Body>(Body(radiusUniverse, windowSiz
e));
   34:                 cin >> *bodies[i];
   35:         }
   36:         for (int i = 0; i < numBodies; i++) {
   37:                 bodies[i]->setSprite();
   38:         }
   39:         for (int i = 0; i < numBodies; i++) {
   40:                 bodies[i]->setPosition();
   41:         }
   42:
   43:         sf::Texture textureBackground;
   44:         textureBackground.loadFromFile("starfield.jpg");
   45:         sf::Sprite spriteBackground(textureBackground);
   46:
   47:         sf::Font font;
   48:         if (!font.loadFromFile("Walkway_Bold.ttf")) {
   49:                 cerr << "Failed to load Font File" << endl;
   50:         }
   51:
   52:         sf::Text time;
   53:         time.setFont(font);
   54:         time.setString(to_string(currentT));
   55:         time.setCharacterSize(28);
   56:         time.setFillColor(sf::Color::White);
   57:
   58:         sf::Music music;
   59:         if (music.openFromFile("2001.wav")) {
```

```
   60:                 music.play();
   61:         } music.play();
   62:
   63:         while (window.isOpen()) {
   64:                 sf::Event event;
   65:                 while (window.pollEvent(event)) {
   66:                         if (event.type == sf::Event::Closed)
   67:                                 window.close();
   68:                 }
   69:
   70:                 double radius = 0;
   71:                 double force = 0;
   72:                 double netXForce = 0;
   73:                 double netYForce = 0;
   74:                 double dX = 0;
   75:                 double dY = 0;
   76:
   77:                 if (currentT < T) {
   78:                         for (int i = 0; i < numBodies; i++) {
   79:                                 for (int j = 0; j < numBodies; j++) {
   80:                                         if (i != j) {
   81:                                                 dX = bodies[j]->xpos - bodie
s[i]->xpos;
   82:                                                 dY = bodies[j]->ypos - bodie
s[i]->ypos;
   83:                                                 radius = findRadius(dX, dY);
   84:                                                 force = calculateForce(bodie
s[i]->mass, bodies[j]->mass, radius);
   85:                                                 netXForce += force * (dX / r
adius);
   86:                                                 netYForce += force * (dY / r
adius);
   87:                                                 bodies[i]->xForce = netXForc
e;
   88:                                                 bodies[i]->yForce = netYForce;
   89:                                         }
   90:                                 }
   91:                                 netXForce = 0;
   92:                                 netYForce = 0;
   93:                         }
   94:
   95:                         for (int i = 0; i < numBodies; i++) {
   96:                                 bodies[i]->step(dT);
   97:                         }
   98:
   99:                         for (int i = 0; i < numBodies; i++) {
  100:                                 bodies[i]->setPosition();
  101:                         }
  102:
  103:                         window.clear();
  104:                         window.draw(spriteBackground);
  105:
  106:                         for (int i = 0; i < numBodies; i++) {
  107:                                 window.draw(*bodies[i]);
  108:                         }
  109:                         window.draw(time);
  110:                         window.display();
  111:                         currentT = currentT + dT;
  112:                         time.setString(to_string(currentT));
  113:                 }
  114:         }
```

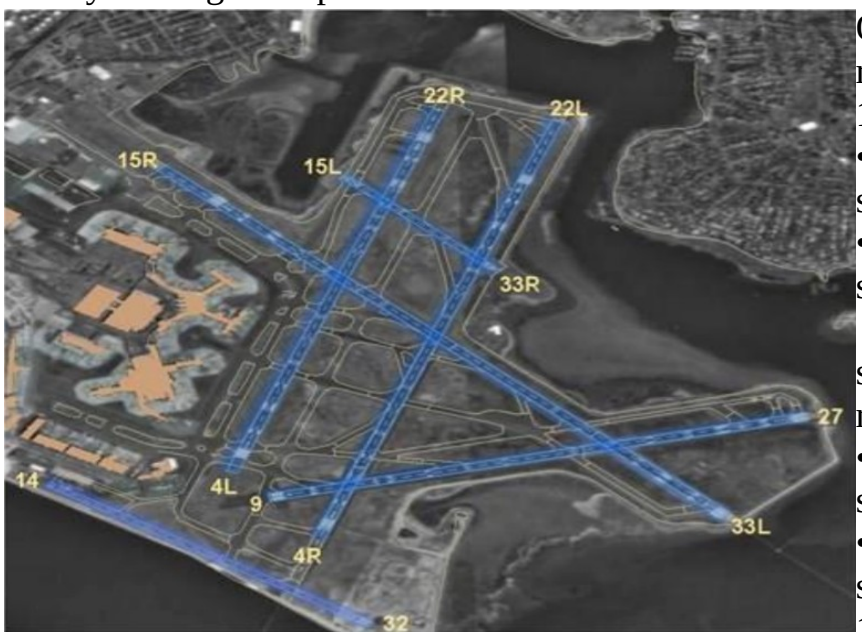
```
  115:
  116:          std::ofstream out("output.txt");
  117:          out.precision(4);
  118:          out << numBodies << endl;
  119:          out << radiusUniverse << endl;
  120:          for (int i = 0; i < numBodies; i++) {
  121:                  out << std::scientific << " " << bodies[i]->xpos << "  " <<
bodies[i]->ypos << "  " << bodies[i]->xvel <<
  122:                  "  " << bodies[i]->yvel << "  " << bodies[i]->mass << "  " <
< bodies[i]->filename << endl;
  123:          }
  124:          return 0;
  125: }
  126:
  127: float findRadius(float dX, float dY) {
  128:          return sqrt(pow(dX, 2) + pow(dY, 2));
  129: }
  130:
  131: float calculateForce(float massA, float massB, float radius) {
  132:          return (gravity * massA * massB) / pow(radius, 2);
  133: }
```

```
    1: #ifndef BODY_HPP_
    2: #define BODY_HPP_
    3:
    4: #include <SFML/Graphics.hpp>
    5: #include <SFML/Window.hpp>
    6: #include <iostream>
    7:
    8: using namespace sf;
    9: using namespace std;
   10:
   11: class Body : public sf::Drawable, sf::Transformable {
   12: public:
   13:         Body(float radiusUniverse, const sf::Vector2f& windowSize);
   14:         friend istream& operator>>(istream& cin, Body& Body);
   15:
   16:         const float radiusUniverse;
   17:         const sf::Vector2f windowSize;
   18:
   19:         void setSprite();
   20:         void setPosition();
   21:         void step(double dT);
   22:
   23:         float xpos;
   24:         float ypos;
   25:         float mass;
   26:         float xvel;
   27:         float yvel;
   28:         float xForce;
   29:         float yForce;
   30:
   31:         std::string filename;
   32:         sf::Texture texture;
   33:         sf::Sprite sprite;
   34:
   35: private:
   36:         void virtual draw(sf::RenderTarget& target, sf::RenderStates states)
 const;
   37: };
   38:
   39: #endif
```

```
    1: #define _USE_MATH_DEFINES
    2: #include <cmath>
    3: #include "Body.hpp"
    4:
    5: using namespace sf;
    6: using namespace std;
    7:
    8: Body::Body(float radiusUniverse, const sf::Vector2f& windowSize) : radiusUni
verse(radiusUniverse), windowSize(windowSize) {}
    9:
   10: istream& operator>>(istream& cin, Body& Body) {
   11:         cin >> Body.xpos >> Body.ypos >> Body.xvel >> Body.yvel >> Body.mass
 >> Body.filename;
   12:         return cin;
   13: }
   14:
   15: void Body::setSprite() {
   16:         if (!(texture.loadFromFile(filename)))
   17:                 cerr << "Failed to load file: " << filename << " ." << endl;
   18:         sprite.setTexture(texture);
   19: }
   20:
   21: void Body::setPosition() {
   22:         float xScale = xpos / radiusUniverse;
   23:         float yScale = ypos / radiusUniverse;
   24:
   25:         sf::Vector2f position(xScale * (windowSize.x / 2) + windowSize.x / 2
, -yScale * (windowSize.y / 2) + windowSize.y / 2);
   26:         sf::Vector2f offset(sprite.getGlobalBounds().width / 2, sprite.getGl
obalBounds().height / 2);
   27:         sprite.setPosition(position - offset);
   28: }
   29:
   30: void Body::draw(sf::RenderTarget& target, sf::RenderStates states) const {
   31:         target.draw(sprite);
   32: }
   33:
   34: void Body::step(double dT) {
   35:         double xAccel = xForce / mass;
   36:         double yAccel = yForce / mass;
   37:
   38:         xvel += (xAccel * dT);
   39:         xpos += (xvel * dT);
   40:
   41:         yvel += (yAccel * dT);
   42:         ypos += (yvel * dT);
   43: }
```

PS4 Airport

I enjoyed this assignment as well because I have been taking operating systems along side this course and we've been learning a lot about threads and locks. We were tasked with creating a program that simulates landings of multiple airplanes on multiple runways at Logan Airport seen below:



Our airport simulation uses six runways labeled 4L, 4R, 9, 14, 15L, & 15R*

• Runway 4L may be used simultaneously with 4R

• Runway 9 may be used simultaneously with 15L

• Runway 14 may be used simultaneously with any other runway

• Runway 15L may be used simultaneously with 15R

• Runway 9 may not be used simultaneously with 4R or 15R

• Runways 15L or 15R may not be used simultaneously with 4L or 4R

• Only one airplane at a time may occupy a given runway

• Due to restrictions of Air Traffic Control, only six requests for landing may be active simultaneously

For our purposes "airplanes" are really threads and we had to figure out a way to restrict the threads from executing in such a way where they wouldn't "crash" given the conditions above. There was also the stipulation that there were 7 airplanes total, but only 6 could attempt to land at any time.

Most of the work was done in AirportServer.cpp where a mutex "lock" and a condition variable "landingRequest" were used to ensure only 6 planes were trying to land at any one time on lines 27 to 30 and 147 to 150. To ensure the runway rules were followed I used mutex locks for each runway and locked or unlocked them in the same order ensuring a deadlock wouldn't be caused (lines 35 to 68 and 112 to 145).

I ran into some trouble debugging this on the given VM, seemed to be running into deadlocks when I shouldn't have been. I tried running the same code on my windows machine and was able to finish up quickly. I believe I completed this assignment 100% it ran for well over 30 minutes without a crash.

I was able to learn how mutex and condition variables are used in c/c++ in a fun and interesting way. I would definitely urge the next class to not even attempt to debug this on a VM.

```
 1: CC = g++
 2: CFLAGS = -c -g -Og -std=c++11
 3: OBJ = Airplane.o Airport.o AirportRunways.o AirportServer.o
 4: DEPS =
 5: LIBS = -pthread
 6: EXE = Airport
 7:
 8: all: $(OBJ)
 9:         $(CC) $(OBJ) -o $(EXE) $(LIBS)
10:
11: %.o: %.cpp $(DEPS)
12:         $(CC) $(CFLAGS) -o $@ $<
13:
14: clean:
15:         rm -f $(OBJ) $(EXE)
```

```
 1: /**
 2: *  Airport driver program
 3: */
 4:
 5: #include <iostream>
 6: #include <thread>
 7: #include <vector>
 8:
 9: #include "AirportServer.h"
10: #include "AirportRunways.h"
11: #include "Airplane.h"
12:
13: using namespace std;
14:
15:
16: int main(void)
17: {
18:         AirportServer as;
19:
20:         vector<thread> apths; // Airplane threads
21:
22:         // Create and launch the individual Airplane threads
23:         for (int i = 1; i <= AirportRunways::NUM_AIRPLANES; i++)
24:         {
25:                 Airplane* ap = new Airplane(i, &as);
26:
27:                 apths.push_back(thread([&] () {
28:                         ap->land();
29:                 }));
30:         }
31:
32:         // Wait for all Airplane threads to terminate (shouldn't happen!)
33:         for (auto& th : apths)
34:         {
35:                 th.join();
36:         }
37:
38:         return 0;
39:
40: } // end main
```

```
    1: /**
    2: *  AirportServer.h
    3: *  This class defines the methods called by the Airplanes
    4: */
    5:
    6: #ifndef AIRPORT_SERVER_H
    7: #define AIRPORT_SERVER_H
    8:
    9: #include <mutex>
   10: #include <random>
   11: #include <condition_variable>
   12: #include "AirportRunways.h"
   13:
   14:
   15:
   16: class AirportServer
   17: {
   18: public:
   19:
   20:         /**
   21:         *  Default constructor for AirportServer class
   22:         */
   23:         AirportServer()
   24:         {
   25:                 // ***** Initialize any Locks and/or Condition Variables her
e as necessary *****
   26:         safeToLand = true;
   27:         } // end AirportServer default constructor
   28:
   29:
   30:           /**
   31:           *  Called by an Airplane when it wishes to land on a runway
   32:           */
   33:         void reserveRunway(int airplaneNum, AirportRunways::RunwayNumber run
way);
   34:
   35:         /**
   36:         *  Called by an Airplane when it is finished landing
   37:         */
   38:         void releaseRunway(int airplaneNum, AirportRunways::RunwayNumber run
way);
   39:
   40:
   41: private:
   42:
   43:         // Constants and Random number generator for use in Thread sleep cal
ls
   44:         static const int MAX_TAXI_TIME = 10; // Maximum time the airplane wi
ll occupy the requested runway after landing, in milliseconds
   45:         static const int MAX_WAIT_TIME = 100; // Maximum time between landin
gs, in milliseconds
   46:
   47:         int numLandingRequests = 0;
   48:         bool safeToLand;
   49:
   50:         /**
   51:         *  Declarations of mutexes and condition variables
   52:         */
   53:         mutex runwaysMutex; // Used to enforce mutual exclusion for acquirin
g & releasing runways
   54:
```

```
   55:         /**
   56:          *  ***** Add declarations of your own Locks and Condition Variables
 here *****
   57:          */
   58:
   59:         mutex runway4R, runway4L, runway9, runway14, runway15R, runway15L;
   60:         condition_variable landingRequest;
   61:
   62: }; // end class AirportServer
   63:
   64: #endif
```

```
    1: #include <iostream>
    2: #include <thread>
    3: #include <condition_variable>
    4: #include <mutex>
    5: #include <chrono>
    6: #include "AirportServer.h"
    7:
    8:
    9: /**
   10: *  Called by an Airplane when it wishes to land on a runway
   11: */
   12: void AirportServer::reserveRunway(int airplaneNum, AirportRunways::RunwayNum
ber runway)
   13: {
   14:         // Acquire runway(s)
   15:
   16:         {
   17:                 lock_guard<mutex> lk(AirportRunways::checkMutex);
   18:
   19:                 cout << "Airplane #" << airplaneNum << " is acquiring any ne
eded runway(s) for landing on Runway "
   20:                         << AirportRunways::runwayName(runway) << endl;
   21:         }
   22:
   23:         /**
   24:          *  ***** Add your synchronization here! *****
   25:          */
   26:
   27:         numLandingRequests++;
   28:         unique_lock<mutex> lock(runwaysMutex);
   29:         while(!safeToLand){
   30:                 landingRequest.wait(lock);
   31:         }
   32:
   33:         lock.unlock();
   34:
   35:         switch (runway){
   36:         case 0:                                                 //4L
   37:                 runway4L.lock();
   38:                 runway15L.lock();
   39:                 runway15R.lock();
   40:                 break;
   41:         case 1:                                                 //4R
   42:                 runway4R.lock();
   43:                 runway9.lock();
   44:                 runway15L.lock();
   45:                 runway15R.lock();
   46:                 break;
   47:         case 2:                                                 //9
   48:                 runway4R.lock();
   49:                 runway9.lock();
   50:                 runway15R.lock();
   51:                 break;
   52:         case 3:                                                 //14
   53:                 runway14.lock();
   54:                 break;
   55:         case 4:                                                 //15L
   56:                 runway4L.lock();
   57:                 runway4R.lock();
   58:                 runway15L.lock();
   59:                 break;
```

```
   60:         case 5:                                                 //15R
   61:                 runway4L.lock();
   62:                 runway4R.lock();
   63:                 runway9.lock();
   64:                 runway15R.lock();
   65:                 break;
   66:         default:
   67:                 break;
   68:         }
   69:
   70:         // Check status of the airport for any rule violations
   71:         AirportRunways::checkAirportStatus(runway);
   72:
   73:         // obtain a seed from the system clock:
   74:         unsigned seed = std::chrono::system_clock::now().time_since_epoch().
count();
   75:         std::default_random_engine generator(seed);
   76:
   77:         // Taxi for a random number of milliseconds
   78:         std::uniform_int_distribution<int> taxiTimeDistribution(1, MAX_TAXI_
TIME);
   79:         int taxiTime = taxiTimeDistribution(generator);
   80:
   81:         {
   82:                 lock_guard<mutex> lk(AirportRunways::checkMutex);
   83:
   84:                 cout << "Airplane #" << airplaneNum << " is taxiing on Runwa
y " << AirportRunways::runwayName(runway)
   85:                         << " for " << taxiTime << " milliseconds\n";
   86:         }
   87:
   88:         std::this_thread::sleep_for(std::chrono::milliseconds(taxiTime));
   89:
   90: } // end AirportServer::reserveRunway()
   91:
   92:
   93:  /**
   94:   *  Called by an Airplane when it is finished landing
   95:   */
   96: void AirportServer::releaseRunway(int airplaneNum, AirportRunways::RunwayNum
ber runway)
   97: {
   98:         // Release the landing runway and any other needed runways
   99:
  100:         {
  101:                 lock_guard<mutex> lk(AirportRunways::checkMutex);
  102:
  103:                 cout << "Airplane #" << airplaneNum << " is releasing any ne
eded runway(s) after landing on Runway "
  104:                         << AirportRunways::runwayName(runway) << endl;
  105:         }
  106:
  107:         /**
  108:          *  ***** Add your synchronization here! *****
  109:          */
  110:
  111:
  112:         switch (runway){
  113:         case 0:                                                 //4L
  114:                 runway4L.unlock();
  115:                 runway15L.unlock();
```

```
  116:                    runway15R.unlock();
  117:                    break;
  118:            case 1:                                                 //4R
  119:                    runway4R.unlock();
  120:                    runway9.unlock();
  121:                    runway15L.unlock();
  122:                    runway15R.unlock();
  123:                    break;
  124:            case 2:                                                 //9
  125:                    runway4R.unlock();
  126:                    runway9.unlock();
  127:                    runway15R.unlock();
  128:                    break;
  129:            case 3:                                                 //14
  130:                    runway14.unlock();
  131:                    break;
  132:            case 4:                                                 //15L
  133:                    runway4L.unlock();
  134:                    runway4R.unlock();
  135:                    runway15L.unlock();
  136:                    break;
  137:            case 5:                                                 //15R
  138:                    runway4L.unlock();
  139:                    runway4R.unlock();
  140:                    runway9.unlock();
  141:                    runway15R.unlock();
  142:                    break;
  143:            default:
  144:                    break;
  145:            }
  146:
  147:            numLandingRequests--;
  148:            if(numLandingRequests < 7){
  149:                    safeToLand = true;
  150:                    landingRequest.notify_one();
  151:            }
  152:
  153:            // Update the status of the airport to indicate that the landing is
complete
  154:            AirportRunways::finishedWithRunway(runway);
  155:
  156:            // obtain a seed from the system clock:
  157:            unsigned seed = std::chrono::system_clock::now().time_since_epoch().
count();
  158:            std::default_random_engine generator(seed);
  159:
  160:            // Wait for a random number of milliseconds before requesting the ne
xt landing for this Airplane
  161:            std::uniform_int_distribution<int> waitTimeDistribution(1, MAX_WAIT_
TIME);
  162:            int waitTime = waitTimeDistribution(generator);
  163:
  164:            {
  165:                    lock_guard<mutex> lk(AirportRunways::checkMutex);
  166:
  167:                    cout << "Airplane #" << airplaneNum << " is waiting for " <<
 waitTime << " milliseconds before landing again\n";
  168:            }
  169:
  170:            std::this_thread::sleep_for(std::chrono::milliseconds(waitTime));
  171:
```

```
172: } // end AirportServer::releaseRunway()
```

PS5 Ring Buffer and Guitar Hero

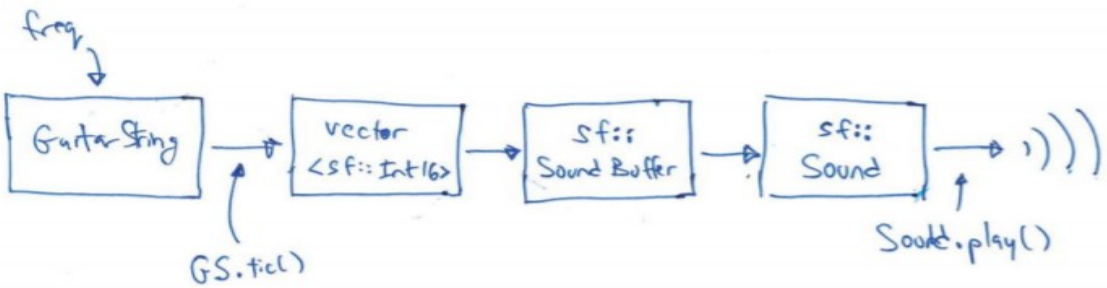
This assignment was reminiscent of PS1 where we implemented an underlying structure and then built some cool functionality on top of it. PS5 tasked us first with implementing a ring buffer, which we then used to play frequencies simulating guitar plucks creating an electronic keyboard using the Karplus-Strong algorithm.

The two primary components that make the Karplus-Strong algorithm work are the ring buffer feedback mechanism and the averaging operation.

The ring buffer feedback mechanism. The ring buffer models the medium (a string tied down at both ends) in which the energy travels back and forth. The length of the ring buffer determines the fundamental frequency of the resulting sound. Sonically, the feedback mechanism reinforces only the fundamental frequency and its harmonics (frequencies at integer multiples of the fundamental). The energy decay factor (.996 in this case) models the slight dissipation in energy as the wave makes a round trip through the string.

The averaging operation serves as a gentle low-pass filter (which removes higher frequencies while allowing lower frequencies to pass, hence the name). Because it is in the path of the feedback, this has the effect of gradually attenuating the higher harmonics while keeping the lower ones, which corresponds closely with how a plucked guitar string sounds. It repeatedly deletes the first sample from the buffer and adds to the end of the buffer the average of the deleted sample and the first sample, scaled by an energy decay factor of 0.996. This was accounted for on line 45 of GuitarString.cpp

The i th character of the string keyboard corresponds to a frequency of 440 × 2(i - 24) / 12 (Line 69 of GuitarHero.cpp), so that the character 'q' is 110Hz, 'i' is 220Hz, 'v' is 440Hz, and ' ' is 880Hz. I was able to use a couple of vectors to shepard the samples (frequencies) from the GuitarString class to SFML’s sound buffer and sound object so that it would play on a key press. Once I was able to get one to play, it was fairly straight forward getting the rest.



When using SFML, we have to have an existing sf::SoundBuffer that's created with a vector of sound samples. This SoundBuffer is created from a vector of sf::Int16s. Then we create a sf::Sound object from the sf::SoundBuffer. The sf::Sound object can then be played. This happens in the for loop on lines 69 to 81 of GuitarHero.cpp

Enabling a cpplint extension on my code editor made things easier at the end for part A and didn’t come across any errors when compiling the given Gstest.cpp file. I also added unit tests for part A which tested the functionality of peek(), isEmpty(), and isFull().

I believe I completed this assignment 100% and it gave an interesting look into how a sound is actually produced electronically, no small feat. I enjoyed this assignment the most as my cat was intrigued by the plethora of the sounds coming out of the speakers during testing.

```
    1: CC = g++
    2: CFLAGS = -std=c++11 -c -g -Og -Wall -pedantic -lz
    3: OBJ = RingBuffer.o GuitarString.o GuitarHero.o GStest.o
    4: DEPS =
    5: LIBS = -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system -lboost_unit
_test_framework
    6: EXE = GuitarHero GStest
    7:
    8: %.o: %.cpp $(DEPS)
    9:         $(CC) $(CFLAGS) -o $@ $<
   10:
   11: GuitarHero: $(OBJ)
   12:         $(CC) RingBuffer.o GuitarString.o GuitarHero.o -o GuitarHero $(LIBS)
   13:
   14: GStest: $(OBJ)
   15:         $(CC) RingBuffer.o GuitarString.o GStest.o -o GStest $(LIBS)
   16:
   17: all: $(OBJ)
   18:         $(CC) $(OBJ) -o $(EXE) $(LIBS)
   19:
   20: clean:
   21:         rm $(OBJ) $(EXE)
   22:
```

```
 1: /*
 2:   Copyright 2015 Fred Martin, fredm@cs.uml.edu
 3:   Mon Mar 30 08:58:49 2015
 4:
 5:   based on Princeton's GuitarHeroLite.java
 6:   www.cs.princeton.edu/courses/archive/fall13/cos126/assignments/guitar.html
 7:
 8:   build with
 9:   g++ -Wall -c GuitarHeroLite.cpp -lsfml-system \
10:      -lsfml-audio -lsfml-graphics -lsfml-window
11:   g++ -Wall GuitarHeroLite.o RingBuffer.o GuitarString.o \
12:    -o GuitarHeroLite -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-windo
w
13: */
14:
15: #include <SFML/Graphics.hpp>
16: #include <SFML/System.hpp>
17: #include <SFML/Audio.hpp>
18: #include <SFML/Window.hpp>
19:
20: #include <math.h>
21: #include <limits.h>
22:
23: #include <iostream>
24: #include <string>
25: #include <exception>
26: #include <stdexcept>
27: #include <vector>
28:
29: #include "RingBuffer.hpp"
30: #include "GuitarString.hpp"
31:
32: #define CONCERT_A 440.0
33: #define SAMPLES_PER_SEC 44100
34:
35: vector<sf::Int16> makeSamplesFromString(GuitarString *gs) {
36:   std::vector<sf::Int16> samples;
37:
38:   gs->pluck();
39:   int duration = 8;
40:   for (int i = 0; i < SAMPLES_PER_SEC * duration; i++) {
41:     gs->tic();
42:     samples.push_back(gs->sample());
43:   }
44:
45:   return samples;
46: }
47:
48: int main() {
49:   sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero");
50:   sf::Event event;
51:   double freq;
52:   vector<sf::Int16> samples;
53:
54:   // we're reusing the freq and samples vars, but
55:   // there are separate copies of GuitarString, SoundBuffer, and Sound
56:   //   for each note
57:   //
58:   // GuitarString is based on freq
59:   // samples are generated from GuitarString
60:   // SoundBuffer is loaded from samples
```

```
   61:   // Sound is set to SoundBuffer
   62:
   63: vector<std::vector <sf::Int16> > sampleVector;
   64:         vector<sf::SoundBuffer> bufferVector;
   65:         vector<sf::Sound> soundVector;
   66:
   67:         for (int i = 0; i < 37; i++)
   68:         {
   69:                 freq = CONCERT_A * pow(2, (i - 24.0) / 12.0);
   70:
   71:                 GuitarString gs = GuitarString(freq);
   72:                 sf::SoundBuffer& buffer = *new sf::SoundBuffer();
   73:                 sf::Sound& sound = *new sf::Sound();
   74:                 samples = makeSamplesFromString(&gs);
   75:                 sampleVector.push_back(samples);
   76:
   77:                 if (!buffer.loadFromSamples(&sampleVector[i][0], sampleVecto
r[i].size(), 2, SAMPLES_PER_SEC))
   78:                         throw std::runtime_error("sf::SoundBuffer: load fail
ure");
   79:
   80:                 sound.setBuffer(buffer);
   81:                 bufferVector.push_back(buffer);
   82:                 soundVector.push_back(sound);
   83:         }
   84:
   85:
   86:   while (window.isOpen()) {
   87:                 while (window.pollEvent(event)) {
   88:                         switch (event.type) {
   89:                         case sf::Event::Closed:
   90:                                 window.close();
   91:                                 break;
   92:                         case sf::Event::KeyPressed:
   93:                                 switch (event.key.code) {
   94:                                 case sf::Keyboard::Q:
   95:                                         soundVector[0].play();
   96:                                         break;
   97:                                 case sf::Keyboard::Num2:
   98:                                         soundVector[1].play();
   99:                                         break;
  100:                                 case sf::Keyboard::W:
  101:                                         soundVector[2].play();
  102:                                         break;
  103:                                 case sf::Keyboard::E:
  104:                                         soundVector[3].play();
  105:                                         break;
  106:                                 case sf::Keyboard::Num4:
  107:                                         soundVector[4].play();
  108:                                         break;
  109:                                 case sf::Keyboard::R:
  110:                                         soundVector[5].play();
  111:                                         break;
  112:                                 case sf::Keyboard::Num5:
  113:                                         soundVector[6].play();
  114:                                         break;
  115:                                 case sf::Keyboard::T:
  116:                                         soundVector[7].play();
  117:                                         break;
  118:                                 case sf::Keyboard::Y:
  119:                                         soundVector[8].play();
```

```
120:                                        break;
121:                                    case sf::Keyboard::Num7:
122:                                        soundVector[9].play();
123:                                        break;
124:                                    case sf::Keyboard::U:
125:                                        soundVector[10].play();
126:                                        break;
127:                                    case sf::Keyboard::Num8:
128:                                        soundVector[11].play();
129:                                        break;
130:                                    case sf::Keyboard::I:
131:                                        soundVector[12].play();
132:                                        break;
133:                                    case sf::Keyboard::Num9:
134:                                        soundVector[13].play();
135:                                        break;
136:                                    case sf::Keyboard::O:
137:                                        soundVector[14].play();
138:                                        break;
139:                                    case sf::Keyboard::P:
140:                                        soundVector[15].play();
141:                                        break;
142:                                    case sf::Keyboard::Dash:
143:                                        soundVector[16].play();
144:                                        break;
145:                                    case sf::Keyboard::LBracket:
146:                                        soundVector[17].play();
147:                                        break;
148:                                    case sf::Keyboard::Equal:
149:                                        soundVector[18].play();
150:                                        break;
151:                                    case sf::Keyboard::Z:
152:                                        soundVector[19].play();
153:                                        break;
154:                                    case sf::Keyboard::X:
155:                                        soundVector[20].play();
156:                                        break;
157:                                    case sf::Keyboard::D:
158:                                        soundVector[21].play();
159:                                        break;
160:                                    case sf::Keyboard::C:
161:                                        soundVector[22].play();
162:                                        break;
163:                                    case sf::Keyboard::F:
164:                                        soundVector[23].play();
165:                                        break;
166:                                    case sf::Keyboard::V:
167:                                        soundVector[24].play();
168:                                        break;
169:                                    case sf::Keyboard::G:
170:                                        soundVector[25].play();
171:                                        break;
172:                                    case sf::Keyboard::B:
173:                                        soundVector[26].play();
174:                                        break;
175:                                    case sf::Keyboard::N:
176:                                        soundVector[27].play();
177:                                        break;
178:                                    case sf::Keyboard::J:
179:                                        soundVector[28].play();
180:                                        break;
```

```
181:                                    case sf::Keyboard::M:
182:                                            soundVector[29].play();
183:                                            break;
184:                                    case sf::Keyboard::K:
185:                                            soundVector[30].play();
186:                                            break;
187:                                    case sf::Keyboard::Comma:
188:                                            soundVector[31].play();
189:                                            break;
190:                                    case sf::Keyboard::Period:
191:                                            soundVector[32].play();
192:                                            break;
193:                                    case sf::Keyboard::SemiColon:
194:                                            soundVector[33].play();
195:                                            break;
196:                                    case sf::Keyboard::Slash:
197:                                            soundVector[34].play();
198:                                            break;
199:                                    case sf::Keyboard::Quote:
200:                                            soundVector[35].play();
201:                                            break;
202:                                    case sf::Keyboard::Space:
203:                                            soundVector[36].play();
204:                                            break;
205:                                    default:
206:                                            break;
207:                                    }
208:                            default:
209:                                    break;
210:                            }
211:                            window.clear();
212:                            window.display();
213:                    }
214:            }
215:
216:            return 0;
217: }
```

```
 1: #ifndef GS_HPP_
 2: #define GS_HPP_
 3:
 4: #include <vector>
 5: #include <SFML/Audio.hpp>
 6: #include <stdint.h>
 7: #include "RingBuffer.hpp"
 8:
 9: class GuitarString{
10:     private:
11:     Ringbuffer *ptrRB;
12:     int N;
13:
14:     public:
15:     explicit GuitarString(double frequency);
16:     explicit GuitarString(std::vector<sf::Int16> init);
17:     ~GuitarString();
18:     void pluck();
19:     void tic();
20:     sf::Int16 sample();
21:     int time();
22:     int count;
23:
24: };
25: #endif
```

```
 1: // Copyright 2019 <Elick Coval>
 2: #ifndef RB_HPP_
 3: #define RB_HPP_
 4:
 5: #include <stdint.h>
 6: #include <iostream>
 7: #include <vector>
 8: #include <stdexcept>
 9:
10: using std::vector;
11: using std::invalid_argument;
12: using std::runtime_error;
13:
14:
15: class Ringbuffer {
16:  private:
17:     std::vector<int16_t> ringbuffer;
18:     int head;
19:     int tail;
20:     int rbCap;
21:     int rbSize;
22:
23:  public:
24:      // Creates an empty ring buffer, with given max capacity
25:      explicit Ringbuffer(int capacity);
26:      // returns number of items currently in the buffer
27:      int size();
28:      // Is the buffer empty(size = zero?)
29:      bool isEmpty();
30:      // Is the buffer full(size = capacity)
31:      bool isFull();
32:      // Add item x to the end
33:      void enqueue(int16_t x);
34:      // Delete and return item from the front
35:      int16_t dequeue();
36:      // Return(doesn't delete) item from the front
37:      int16_t peek();
38: };
39: #endif  // RB_HPP_
```

```
 1: #include "GuitarString.hpp"
 2: #include <stdint.h>
 3: #include <math.h>
 4: #include <SFML/Audio.hpp>
 5: #include <vector>
 6: #include <iostream>
 7: #include "RingBuffer.hpp"
 8:
 9: GuitarString::GuitarString(double frequency)
10: {
11:         count = 0;
12:         N = ceil(44100 /frequency);
13:         ptrRB = new Ringbuffer(N);
14:         while ((*ptrRB).isEmpty())
15:         (*ptrRB).enqueue(0);
16: }
17:
18: GuitarString::GuitarString(std::vector<sf::Int16> init)
19: {
20:         count = 0;
21:         N = init.size();
22:         ptrRB = new Ringbuffer(N);
23:         for (std::vector<sf::Int16>::
24:         iterator it = init.begin(); it != init.end(); ++it)
25:         (*ptrRB).enqueue(*it);
26: }
27:
28: GuitarString::~GuitarString()
29: {
30:         delete ptrRB;
31: }
32:
33: void GuitarString::pluck()
34: {
35:         while (!(*ptrRB).isEmpty())
36:         (*ptrRB).dequeue();
37:                 while (!(*ptrRB).isFull())
38:                 (*ptrRB).enqueue((sf::Int16)(rand() & 0xffff));
39: }
40:
41: void GuitarString::tic()
42: {
43:         int16_t front = (*ptrRB).dequeue();
44:         int16_t frontNext = (*ptrRB).peek();
45:         float result = ((front + frontNext)/2) * 0.996;
46:         (*ptrRB).enqueue(result);
47: }
48:
49: sf::Int16 GuitarString::sample()
50: {
51:         return (*ptrRB).peek();
52: }
53:
54: int GuitarString::time()
55: {
56:         return count++;
57: }
```

```
 1: // Copyright 2019 Elick Coval
 2:
 3: #include <stdint.h>
 4: #include <iostream>
 5: #include <vector>
 6: #include <stdexcept>
 7: #include "RingBuffer.hpp"
 8:
 9: // Default constructor
10: Ringbuffer::Ringbuffer(int capacity) {
11:     if (capacity < 1) {
12:         throw invalid_argument("Error*: capacity must be greater than zero."
);
13:         }
14:         ringbuffer.resize(capacity);
15:         rbCap = capacity;
16:         rbSize = 0;
17:         head = 0;
18:         tail = 0;
19: }
20:
21: // Returns the size of the vector
22: int Ringbuffer::size() {
23:     return rbSize;
24: }
25:
26: // Function to check to see if the vector is empty
27: bool Ringbuffer::isEmpty() {
28:     if (rbSize == 0)
29:         return 1;
30:     else
31:         return 0;
32: }
33: // Function to check to see if the vector is full
34: bool Ringbuffer::isFull() {
35:     int current;
36:     current = rbCap;
37:     if (size() == current)
38:         return 1;
39:     else
40:         return 0;
41: }
42:
43: // Function which allows us to enqueue a type "double" onto the vector
44: void Ringbuffer::enqueue(int16_t x) {
45:     if (!isFull()) {
46:          ringbuffer[tail] = x;
47:          tail = (tail + 1) % rbCap;
48:          rbSize++;
49:     } else {
50:         throw runtime_error("Enqueue error*: Can’t enqueue to a full ring");
51: }}
52:
53: // Function which allows us to dequeue a type from the vector
54: int16_t Ringbuffer::dequeue() {
55:      if (rbSize == 0) {
56:         throw std::runtime_error("The ring buffer is empty!");
57:     }
58:
59:     int temp = ringbuffer[head];
60:     head = (head + 1) % rbCap;
```

```
    rbSize--;
    return temp;
}

int16_t Ringbuffer::peek() {
    if (rbSize == 0) {
        throw std::runtime_error("The ring buffer is empty!");
    }
    return ringbuffer[head];
}
```

```
 1: #define BOOST_TEST_DYN_LINK
 2: #define BOOST_TEST_MODULE Main
 3: #include <boost/test/unit_test.hpp>
 4:
 5: #include <stdint.h>
 6: #include <iostream>
 7: #include <string>
 8: #include <exception>
 9: #include <stdexcept>
10:
11: #include "RingBuffer.hpp"
12:
13: BOOST_AUTO_TEST_CASE(RBconstructor) {
14:         // normal constructor
15:         Ringbuffer RB(100);
16:         BOOST_REQUIRE_NO_THROW(RB);
17:
18:         // this should fail
19: //      BOOST_REQUIRE_THROW(RB(0), std::exception);
20: //      BOOST_REQUIRE_THROW(RB(0), std::invalid_argument);
21: }
22:
23: BOOST_AUTO_TEST_CASE(RBenque_dequeue) {
24:         Ringbuffer rb(100);
25:
26:         rb.enqueue(2);
27:         rb.enqueue(1);
28:         rb.enqueue(0);
29:
30:         BOOST_REQUIRE(rb.dequeue() == 2);
31:         BOOST_REQUIRE(rb.dequeue() == 1);
32:         BOOST_REQUIRE(rb.dequeue() == 0);
33:
34:         BOOST_REQUIRE_THROW(rb.dequeue(), std::runtime_error);
35: }
36:
37: BOOST_AUTO_TEST_CASE(RBpeek){
38:         Ringbuffer rb(100);
39:         BOOST_REQUIRE_THROW(rb.peek(), std::runtime_error);
40:         rb.enqueue(0);
41:
42:         //should fail
43: //      BOOST_REQUIRE(rb.peek() == 1);
44: }
45:
46: BOOST_AUTO_TEST_CASE(RBfull_empty){
47:         Ringbuffer rb(3);
48:         BOOST_REQUIRE(rb.isEmpty() == true);
49:         BOOST_REQUIRE(rb.size() == 0);
50:
51:         rb.enqueue(2);
52:         BOOST_REQUIRE(rb.size() == 1);
53:         rb.enqueue(1);
54:         BOOST_REQUIRE(rb.size() == 2);
55:         rb.enqueue(0);
56:         BOOST_REQUIRE(rb.size() == 3);
57:         BOOST_REQUIRE(rb.isFull() == true);
58:         BOOST_REQUIRE_THROW(rb.enqueue(-1), std::runtime_error);
59:
60:         BOOST_REQUIRE(rb.dequeue() == 2);
61:         BOOST_REQUIRE(rb.size() == 2);
```

```
   62:         BOOST_REQUIRE(rb.dequeue() == 1);
   63:         BOOST_REQUIRE(rb.size() == 1);
   64:         BOOST_REQUIRE(rb.dequeue() == 0);
   65:         BOOST_REQUIRE(rb.size() == 0);
   66: }
```

```
    1: /*
    2:   Copyright 2015 Fred Martin, fredm@cs.uml.edu
    3:   Wed Apr  1 09:43:12 2015
    4:   test file for GuitarString class
    5:
    6:   compile with
    7:   g++ -c GStest.cpp -lboost_unit_test_framework
    8:   g++ GStest.o GuitarString.o RingBuffer.o -o GStest -lboost_unit_test_frame
work
    9: */
   10:
   11: #define BOOST_TEST_DYN_LINK
   12: #define BOOST_TEST_MODULE Main
   13: #include <boost/test/unit_test.hpp>
   14:
   15: #include <vector>
   16: #include <exception>
   17: #include <stdexcept>
   18:
   19: #include "GuitarString.hpp"
   20:
   21: BOOST_AUTO_TEST_CASE(GS) {
   22:   vector<sf::Int16> v;
   23:
   24:   v.push_back(0);
   25:   v.push_back(2000);
   26:   v.push_back(4000);
   27:   v.push_back(-10000);
   28:
   29:   BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));
   30:
   31:   GuitarString gs = GuitarString(v);
   32:
   33:   // GS is 0 2000 4000 -10000
   34:   BOOST_REQUIRE(gs.sample() == 0);
   35:
   36:   gs.tic();
   37:   // it's now 2000 4000 -10000 996
   38:   BOOST_REQUIRE(gs.sample() == 2000);
   39:
   40:   gs.tic();
   41:   // it's now 4000 -10000 996 2988
   42:   BOOST_REQUIRE(gs.sample() == 4000);
   43:
   44:   gs.tic();
   45:   // it's now -10000 996 2988 -2988
   46:   BOOST_REQUIRE(gs.sample() == -10000);
   47:
   48:   gs.tic();
   49:   // it's now 996 2988 -2988 -4483
   50:   BOOST_REQUIRE(gs.sample() == 996);
   51:
   52:   gs.tic();
   53:   // it's now 2988 -2988 -4483 1984
   54:   BOOST_REQUIRE(gs.sample() == 2988);
   55:
   56:   gs.tic();
   57:   // it's now -2988 -4483 1984 0
   58:   BOOST_REQUIRE(gs.sample() == -2988);
   59:
   60:   // a few more times
```

```
   61:   gs.tic();
   62:   BOOST_REQUIRE(gs.sample() == -4483);
   63:   gs.tic();
   64:   BOOST_REQUIRE(gs.sample() == 1984);
   65:   gs.tic();
   66:   BOOST_REQUIRE(gs.sample() == 0);
   67: }
```