

Khim Sok  
COMP IV: Project Portfolio  
Spring 2020



*Learning with Purpose*

**Contents:**

<b>PS0</b> Hello World with SFML .....	page 2
<b>PS1</b> Linear Feedback Shift Register and Image Encoding.....	page 6
<b>PS2</b> Recursive Graphics (Pythagoras tree).....	page 16
<b>PS3</b> N-Body Simulation.....	page 23
<b>PS4</b> DNA Sequence Alignment.....	page 38
<b>PS5</b> Ring Buffer and Guitar Hero.....	page 51
<b>PS6</b> Markov Model of Natural Language.....	page 60

## **PS0: Hello World with SFML**

### **Assignment:**

This assignment will help student get familiar with Linux system environment and get to know a very useful C++ Library; Simple Fast Media Library known as SFML. Student can find tutorials about using SFML Library on <https://www.sfml-dev.org/tutorials/2.5/>.

SFML is C++ Library that allow programmer to build graphics, video animate, and audio sound tools. Students will find this very interesting and fun to learn.

Getting start on this assignment, students are required to install Linux Ubuntu Operating System that professor provided using Visual Machine. A secondary option is installing X2go Client on your computer and use your UML CS account to connect with UMass Lowell server.

After student get everything set up, we will start working on a demo code that professor provided in PS0, and try to extend the code by adding some features such as (1).drawing an image sprite, (2).make the image sprite move, (3).make the image sprite respond to keystroke, and (4).make it do something else, be creative.

### **Key algorithms:**

The key algorithms that I used in this assignment is some built-in function in SFML Library by implementing texture class, sprite class, and shape class to create image and loading image, and also RenderWindow to create a window display on screen as video mode.

### **What I learn from this assignment:**

From this assignment, I have learned using SFML library to load images, create texture and sprite on a window, create some shapes, and make it move to keystroke when keyboard key is pressed. More importantly, learned to create Makefile which is necessary for compiling the code and run the program.

## Makefile

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

## main.cpp

```
1: //Name:Khim Sok
2:
3: #include <SFML/Graphics.hpp>
4:
5: int main(){
6:
7: sf::RenderWindow window(sf::VideoMode(600, 600), "SFML works!");
8: sf::CircleShape shape(100.f);
9: shape.setFillColor(sf::Color::Yellow);
10:
11: // Load a sprite to display a Rocket
12: sf::Texture texture;
13: if (!texture.loadFromFile("sprite.png"))
14: return EXIT_FAILURE;
15: sf::Sprite sprite(texture);
16: sprite.setPosition(225.f, 485.f);
17:
18: // Load a sprite to display Moon
19: sf::Texture textureMoon;
20: if (!textureMoon.loadFromFile("moon.png"))
21: return EXIT_FAILURE;
22: sf::Sprite spriteMoon(textureMoon);
23: spriteMoon.setPosition(170.f, -55.f);
24:
25: // Load a sprite to display background
26: sf::Texture textureSpace;
27: if (!textureSpace.loadFromFile("space.png"))
28: return EXIT_FAILURE;
29: sf::Sprite spriteSpace(textureSpace);
30:
31: // Load a sprite to display an astronaut
32: sf::Texture textureAst;
33: if (!textureAst.loadFromFile("astronaut.png"))
34: return EXIT_FAILURE;
35: sf::Sprite spriteAst(textureAst);
36: spriteAst.setPosition(90, 30);
37:
38: //load a font and set font
```

```

39: sf::Font font;
40: if(!font.loadFromFile("aller.ttf"))
41: return EXIT_FAILURE;
42:
43: //set a text and font style
44: sf::Text text;
45: text.setFont(font);
46: text.setString("Hello Moon!!!\nMission Complete!!!");
47: text.setCharacterSize(16);
48: text.setFillColor(sf::Color::Yellow);
49: text.setPosition(20.f, 5.f);
50:
51: //set a text and set the font
52: sf::Text textGuide;
53: textGuide.setFont(font);
54: textGuide.setCharacterSize(16);
55: textGuide.setString("Your mission is to send the Rocket \nto the Moon
    and land on the moon \nsuccuessfully!!!");
56: textGuide.setPosition(10.f, 430.f);
57:
58:
59:
60:
61: while (window.isOpen()){
62: sf::Event event;
63: while (window.pollEvent(event)){
64: if (event.type == sf::Event::Closed)
65: window.close();
66: }
67:
68: //Keyboard
69: if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
70: {
71: // left key is pressed: move our character
72: sprite.move(-0.5f, 0.0f);
73: }
74: if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
75: {
76: // Right key is pressed: move our character
77: sprite.move(0.5f, 0.0f);
78: }
79: if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
80: {
81: // Up key is pressed: move our character
82: sprite.move(0.0f, -0.5f);
83: }
84: if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
85: {
86: // Down key is pressed: move our character
87: sprite.move(0.0f, 0.5f);
88: }
89: //keep udate the display
90: window.clear();
91: window.draw(spriteSpace);
92: window.draw(spriteMoon);
93: window.draw(sprite);
94: window.draw(textGuide);

```

```

95: textGuide.setFillColor(sf::Color::Red);
96:
97: //set condition when the rocket is on the moon
98: //will display a picter and text.
99: if(sprite.getPosition().x <= 300 &&
100: sprite.getPosition().x >= 150 &&
101: sprite.getPosition().y <= 35 &&
102: sprite.getPosition().y >=5)
103: { window.draw(spriteAst);
104: window.draw(text);
105: textGuide.setFillColor(sf::Color::Green);
106: }
107:
108: window.display();
109:
110: }
111:
112: return 0;
113: }

```

**Running SFML app that the image responds to keystroke:**

```

• Terminal
File Edit View Search Terminal Help
ksok@cs4:~/Desktop/PS_Folder/PS0/PS0$ make
g++ -c -g -Og -Wall -Werror -ansi -pedantic -o main.o main.cpp
g++ main.o -o sfml-app -lsfml-graphics -lsfml-window -lsfml-system
ksok@cs4:~/Desktop/PS_Folder/PS0/PS0$ ./sfml-app

```

**Press key arrow up to move up**

**Press key arrow down to move down**

**Press key arrow left to move left**

**Press key arrow right to move right**



## **PS1: Linear Feedback Shift Register and Image Encoding**

### **Assignment:**

In this assignment, student will need to write a program that produces pseudo-random bits by simulating and implementing a conception of LFSR; Linear Feedback Shift Register.

There are two part in this assignment. Part A, we will build the LFSR simulation that has `step()` function and `generate()` function. In Part B, we will use Part A as a tool to decode and encode an image using Transform function as Linear Feedback Shift Register simulation.

I begin this assignment by learning and try to understand what LFSR is and started to implement the concept and apply the algorithm that move and shift the register bits one position to the left and replaces a new generated bit. The goal is to apply the LFSF concept and XOR method on FibLFSR class that provided by professor and do several times of testing on Boost test framework that provided.

### **Key algorithms:**

The key algorithm that I used in this assignment is the concept of Linear Feedback Shift Register and method of XOR.

I applied the integer as a string to a constructor of FibLFSR that I have build and using `step()` function to simulate one step of LFSR that return a right bit as an integer(0 or 1). Then using `generated()` function to take an integer k as an argument and return a k-bit integer by simulating k steps of the LFSR. To be able to know if my program is run correctly, and has no error, I used `boost_unit_test_framework` library to test my program and check if all the function is working properly and has no error occurs.

### **What I learn from this assignment:**

This assignment taught me on using the concept of LFSR and XOR gate method. More importantly, the Boost Test Framework which is very important tool for programmer to test our code and program to make sure it works perfectly, efficient, and minimize the potential of error occurs by testing serval time with difference scenario.

**Makefile**

```
1: CC = g++
2: CFLAGS = -g -Wall -Werror -std=c++11 -pedantic
3: LIBS = -lboost_unit_test_framework -lsfml-graphics -lsfml-window -lsfml
  system
4:
5: all: PhotoMagic test FibLFSR.o
6:
7: PhotoMagic: PhotoMagic.o FibLFSR.o
8: $(CC) $(CFLAGS) -o PhotoMagic PhotoMagic.o FibLFSR.o $(LIBS)
9:
10: test: test.o
11: $(CC) $(CFLAGS) -o test test.o FibLFSR.o $(LIBS)
12:
13: PhotoMagic.o: PhotoMagic.cpp FibLFSR.hpp
14: $(CC) $(CFLAGS) -c PhotoMagic.cpp -o PhotoMagic.o
15:
16: FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
17: $(CC) $(CFLAGS) -c FibLFSR.cpp -o FibLFSR.o
18:
19: test.o: test.cpp FibLFSR.hpp
20: $(CC) $(CFLAGS) -c test.cpp -o test.o
21:
22: clean:
23: rm -f *.o all
```

### PhotoMagic.cpp

```
1: #include<SFML/Graphics.hpp>
2: #include<iostream>
3: #include<cstdlib>
4: #include<string>
5: #include"FibLFSR.hpp"
6: using namespace std;
7:
8: //Fuction phtotype
9: void transform( sf::Image&, FibLFSR*);
10:
11: int main(int argc, char* argv[])
12: {
13:     //initial the value from command into seed
14:     string seed = argv[argc - 1];
15:     FibLFSR flfsr(seed);
16:
17:     //=====
18:     //Input Image
19:     //=====
20:     sf::Image in_img;
21:     if (!in_img.loadFromFile(argv[1]))
22:         return -1;
23:
24:     //create window for input image
25:     sf::RenderWindow window1(sf::VideoMode(in_img.getSize().x,
in_img.getSize().y), "Input Image");
26:
27:     //texture for input image
28:     sf::Texture in_texture;
29:     in_texture.loadFromImage(in_img);
30:
31:     //sprite for input image
32:     sf::Sprite in_sprite(in_texture);
33:     //*****
34:
35:
36:     //=====
37:     //Output Image
38:     //=====
39:     sf::Image out_img = in_img;
40:
41:     sf::RenderWindow window2(sf::VideoMode(out_img.getSize().x,
out_img.getSize().y), "Output Image");
42:
43:     transform(out_img, &flfsr);
44:
45:     sf::Texture out_texture;
46:     out_texture.loadFromImage(out_img);
47:
48:     sf::Sprite out_sprite(out_texture);
49:
50:     // Save the image to a file
51:     if (!out_img.saveToFile(argv[2]))
52:         return -1;
53:     //*****
54:
```



```

55:  //=====
56:  //While loop to display windows
57:  //=====
58:  while (window1.isOpen() && window2.isOpen())
59:  {
60:      sf::Event event;
61:      while (window1.pollEvent(event))
62:      {
63:          if (event.type == sf::Event::Closed)
64:              window1.close();
65:      }
66:
67:      while (window2.pollEvent(event))
68:      {
69:          if (event.type == sf::Event::Closed)
70:              window2.close();
71:      }
72:
73:      window1.clear();
74:      window1.draw(in_sprite);
75:      window1.display();
76:      window2.clear();
77:      window2.draw(out_sprite);
78:      window2.display();
79:
80:  } //end while
81:
82: } //end main
83:
84: //Transform function to call generate function from
85: //class FibLFSR in order to encode and decode image
86: void transform( sf::Image& image, FibLFSR* flfsr)
87: {
88:     sf::Color color;
89:     sf::Vector2u img = (image).getSize();
90:     for(unsigned int i = 0; i<img.x; i++)
91:     {
92:         for(unsigned int j = 0; j<img.y; j++)
93:         {
94:             color = (image).getPixel(i, j);
95:             color.r ^= (*flfsr).generate(8);
96:             color.g ^= (*flfsr).generate(8);
97:             color.b ^= (*flfsr).generate(8);
98:             (image).setPixel(i, j, color);
99:         }
100:     }
101: }
102:
103: } //end transform fuction

```

**FibLFSR.hpp Thu Feb 06 18:36:30 2020 1**

```
1: #ifndef FIBLFSR_HPP
2: #define FIBLFSR_HPP
3: #include<iostream>
4: #include<string>
5: #include<vector>
6: using namespace std;
7:
8: //Class FibLFSR declaration
9: class FibLFSR
10: {
11: public:
12:     FibLFSR(string seed); // constructor to create LFSR with the given
        initial seed and tap
13:
14:     //deconstructor
15:     ~FibLFSR();
16:
17:     //simulate one step and return the new bit as 0 or 1
18:     int step();
19:
20:     //simulate k steps and return k-bit integer
21:     int generate(int k);
22:
23:     //return seed's length
24:     int getLength();
25:
26:     // transforms image using FibLFSR
27:     // void transform( sf::Image&, FibLFSR*);
28:
29:     //print out the string when user call an object of this class
30:     friend ostream& operator<< (ostream &print, const FibLFSR &fiblfsr);
31:
32:     //variable declaration
33: private:
34:
35:     int newBit; //store result of step() function
36:     int multiBit; //store result of generate() function
37:     string seed; //store the user input seet as a string
38:     vector<int> vect; //store the seed in vector as integer number
39:
40: };
41:
42: #endif /* FIBLFSR_HPP */
43:
```

**FibLFSR.cpp**

```
1: #include "FibLFSR.hpp"
2: #include <iostream>
3: #include <cstdlib>
4: #include <string>
5: #include <vector>
6: #include <cmath>
7: using namespace std;
8:
9: //Constructor
10: FibLFSR::FibLFSR(string seed)
11: {
12:     for(unsigned int i=0; i<seed.size(); i++)
13:     {
14:         vect.push_back(((static_cast<int>(seed.at(i))-48)));
15:     }
16:
17:     try{
18:         if(getLength() > 16 || getLength() < 16)
19:         {
20:             throw "The Seed Lenght must be 16 bits!!!";
21:         }
22:     }catch(const char* ex)
23:     {
24:         cout << "Error detected: " << ex << endl;
25:     }
26: }
27:
28: //Deconstructor
29: FibLFSR::~FibLFSR(){}
30:
31: //return seed's length
32: int FibLFSR::getLength()
33: {
34:     return vect.size();
35: }
36:
37: //Function step to shift register one step to the left and return a
    newbit
38: int FibLFSR::step()
39: {
40:     newBit = vect.front()^vect.at(2)^vect.at(3)^vect.at(5);
41:
42:     vect.erase(vect.begin());
43:     vect.push_back(newBit);
44:
45:     return newBit;
46: }
47:
48: //Function generate to recieve integer k and run k step
49: //then add up the first 5 bit of binary number and convert to
50: //decimal number and return the result
51: int FibLFSR::generate(int k)
52: {
53:     multiBit = 0;
54:     int bit = 0;
55:
```

```

56:   for(int i=k; i>0; i--)
57:   {
58:       bit = step();
59:       multiBit += pow(2, i-1)*bit;
60:
61:   }
62:
63:   return multiBit;
64: }
65:
66:
67: /*this operator will display a string of 16 bit and
68: a result of generate function*/
69: ostream& operator<< (ostream &print, const FibLFSR &fiblfsr)
70: {
71:   for(unsigned int i=0; i<fiblfsr.vect.size(); i++)
72:       print<<fiblfsr.vect.at(i);
73:
74:   return print;
75: }

```

```

test.cpp
1: // Dr. Rykalova
2: // test.cpp for PS1a
3: // updated 1/31/2020
4:
5: #include <iostream>
6: #include <string>
7:
8: #include "FibLFSR.hpp"
9:
10: #define BOOST_TEST_DYN_LINK
11: #define BOOST_TEST_MODULE Main
12: #include <boost/test/unit_test.hpp>
13:
14: BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {
15:
16:     FibLFSR l("1011011000110110");
17:     BOOST_REQUIRE(l.step() == 0);
18:     BOOST_REQUIRE(l.step() == 0);
19:     BOOST_REQUIRE(l.step() == 0);
20:     BOOST_REQUIRE(l.step() == 1);
21:     BOOST_REQUIRE(l.step() == 1);
22:     BOOST_REQUIRE(l.step() == 0);
23:     BOOST_REQUIRE(l.step() == 0);
24:     BOOST_REQUIRE(l.step() == 1);
25:
26:     FibLFSR l2("1011011000110110");
27:     BOOST_REQUIRE(l2.generate(9) == 51);
28:
29:     //test1
30:     cout << "test1" << endl;
31:     FibLFSR additionalTest1("1011011000110110");
32:     BOOST_REQUIRE(additionalTest1.getLength() == 16);
33:
34:     //test2
35:     cout << "test2" << endl;
36:     FibLFSR additionalTest2("1011011000110110");
37:
38:     //this seed is greater than 16 bits, so it will show error message.
39:     //FibLFSR additionalTest2("1011011000110110110");
40:     BOOST_REQUIRE(additionalTest2.getLength() == 16);
41:
42:     //test3
43:     cout << "test3" << endl;
44:     FibLFSR additionalTest3("1011011000110110");
45:
46:     //this seed is less than 16 bits, so it will show error message.
47:     //FibLFSR additionalTest3("1000110110");
48:     BOOST_REQUIRE(additionalTest3.getLength() == 16);
49: }

```

## Testing PartA:

```
• Terminal
File Edit View Search Terminal Help
ksok@cs4:~/Desktop/PS_Folder/PS1a$ make
g++ -g -Wall -Werror -std=c++17 -pedantic -c main.cpp
g++ -g -Wall -Werror -std=c++17 -pedantic -c FibLFSR.cpp
g++ -g -Wall -Werror -std=c++17 -pedantic -c test.cpp -o test.o
g++ -g -Wall -Werror -std=c++17 -pedantic -o test test.o FibLFSR.o -lboost_unit_test_framework
g++ -g -Wall -Werror -std=c++17 -pedantic main.o FibLFSR.o -o ps1a
ksok@cs4:~/Desktop/PS_Folder/PS1a$ ./ps1a
0110110001101100 0
1101100011011000 0
1011000110110000 0
0110001101100001 1
1100011011000011 1
1000110110000110 0
0001101100001100 0
0011011000011001 1
0110110000110011 1
1101100001100110 0

1100011011000011 3
1101100001100110 6
0000110011001110 14
1001100111011000 24
0011101100000001 1
0110000000101101 13

0000010110111100 28
1011011110010001 17
1111001000100100 4
0100010010011101 29
1001001110111100 28
0111011110000110 6
```

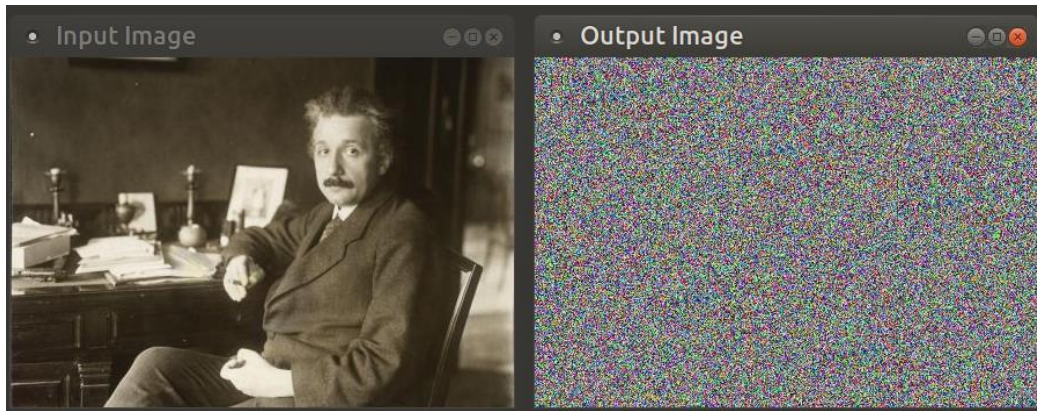
```
ksok@cs4:~/Desktop/PS_Folder/PS1a$ ./test
Running 1 test case...
test1
test2
test3

*** No errors detected
```

## Running PS1 PartB:

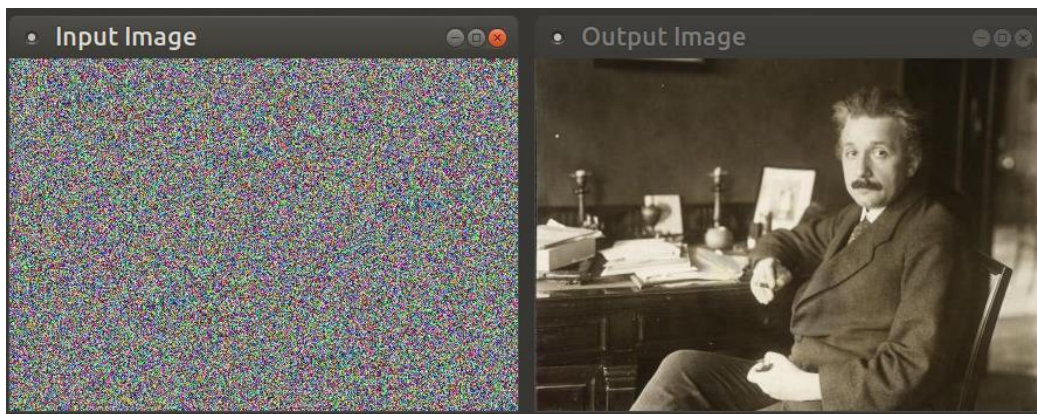
### First run:

```
ksok@cs4:~/Desktop/PS_Folder/PS1b$ ./PhotoMagic input-file.png output-file.png 1011011000110110
```



### Second run:

```
ksok@cs4:~/Desktop/PS_Folder/PS1b$ ./PhotoMagic input-file.png output-file.png 1011011000110110  
ksok@cs4:~/Desktop/PS_Folder/PS1b$ ./PhotoMagic output-file.png input-file.png 1011011000110110
```



\*Notice that the image will return after the second run. This is the concept of encode and decode.

## PS2 Recursive Graphics (Pythagoras tree)

### Assignment:

For this assignment, we will write a program that create a Pythagoras tree using recursion algorithm that take an integer  $N$  to generate  $N$  recursion. To begin, we need to build a class `PTree` that has a constructor that take two command-line arguments  $L$  and  $N$ , size of the base square, and the depth of the recursion, respectively. Then I implemented `sf::Drawable` to have it draw itself to main window created by SFML window; `sf::RenderWindow`, inside main function that generate an exactly as big as  $6L \times 4L$ .

### Key algorithms:

The main algorithms in this assignment is **Recursion**. The recursive method will us generate a  $N$  depth of recursive, then apply the method on `sf::Drawable` to help it draw a Pythagoras tree base  $6L \times 4L$  on a window.

### What I learn from this assignment:

I have learned that recursive method is very useful yet has its own complexity that we need to understand and draw a plan when using it. This assignment was one of the most difficult projects in this course yet spend the most time on it. The recursive is the most challenging and confusing that take the difficulty to the next level.



## Makefile

```
1: CC = g++
2: CFLAGS = -g -Wall -Werror -std=c++11 -pedantic
3: LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4:
5: all: PTree.o main.o tree
6:
7: tree: main.o PTree.o
8: $(CC) $(CFLAGS) -o tree main.o PTree.o $(LIBS)
9:
10: main.o: main.cpp PTree.hpp
11: $(CC) $(CFLAGS) -c main.cpp -o main.o
12:
13: PTree.o: PTree.cpp PTree.hpp
14: $(CC) $(CFLAGS) -c PTree.cpp -o PTree.o
15:
16: clean:
17: rm -f *.o all tree
```

**main.cpp**

```
1: #include "PTree.hpp"
2: #include <SFML/Graphics.hpp>
3: #include <SFML/Window.hpp>
4: #include <iostream>
5: #include <cstdlib>
6: #include <string>
7: #include <cmath>
8: #include <vector>
9: using namespace std;
10:
11: void pTree(sf::RenderTarget& target, const int N, const
    sf::RectangleShape&parentSquare);
12:
13: int main(int argc, char *argv[])
14: {
15:     if(argc != 3)
16:     {
17:         cout<<"This program only accept 3 arguments!" << endl;
18:         cout<<"Please try again later.\n" << endl;
19:         return 0;
20:     }
21:
22:     int L = atoi(argv[1]);
23:     int N = atoi(argv[2]);
24:
25:     const unsigned window_width = static_cast<unsigned>(6*L);
26:     const unsigned window_height = static_cast<unsigned>(4*L);
27:
28:     //create window for input image
29:     sf::RenderWindow window(sf::VideoMode(window_width, window_height),
        "Pythagoras Tree");
30:
31:     PTree pT(L, N, window);
32:
33:     while (window.isOpen())
34:     {
35:         sf::Event event;
36:         while (window.pollEvent(event))
37:         {
38:             if (event.type == sf::Event::Closed)
39:                 window.close();
40:         }
41:
42:         window.clear();
43:         pTree(window, N, pT.rect);
44:         window.display();
45:
46:     } //end while
47:
48: } //end main
49:
50: //recursive function
51: void pTree(sf::RenderTarget& target, const int N, const
    sf::RectangleShape&parentSquare)
52: {
53:
```

```

54:     if(N == 1)
55:         return;
56:     else{
57:
58:         static const float halfsqrt = sqrt(2.f) / 2;
59:
60:         target.draw(parentSquare);
61:
62:         auto const& sizeSquare = parentSquare.getSize();
63:         auto const& transformSquare = parentSquare.getTransform();
64:
65:         //Left Square
66:         auto leftChild = parentSquare;
67:         leftChild.setSize(sizeSquare * halfsqrt);
68:         leftChild.setOrigin(0, leftChild.getSize().y);
69:         leftChild.setPosition(transformSquare.transformPoint(0, 0));
70:         leftChild.rotate(-45);
71:         leftChild.setFillColor(sf::Color::Blue);
72:
73:         //Right Square
74:         auto rightChild = parentSquare;
75:         rightChild.setSize(sizeSquare * halfsqrt);
76:         rightChild.setOrigin(rightChild.getSize());
77:         rightChild.setPosition(transformSquare.transformPoint({sizeSquare.x,
78:         0}));
79:         rightChild.rotate(45);
80:         rightChild.setFillColor(sf::Color::Cyan);
81:
82:         //recursive until N = 1
83:         pTree(target, N-1, leftChild);
84:         pTree(target, N-1, rightChild);
85:     } //end else
86: } //end main

```

**Ptree.hpp**

```
1: #ifndef PTREE_HPP
2: #define PTREE_HPP
3: #include<SFML/Graphics.hpp>
4: #include<SFML/Window.hpp>
5: #include<iostream>
6: #include<string>
7:
8: using namespace std;
9:
10: class PTree : public sf::Drawable, sf::Transformable
11: { private:
12:     int L; //length
13:     int N; //Number of Recursion
14:
15: public:
16:     //constructor
17:     PTree(int L, int N, sf::RenderTarget& target);
18:
19:     //deconstrutor
20:     ~PTree();
21:
22:     sf::RectangleShape rect;
23:
24:     void draw(sf::RenderTarget& target, sf::RenderStates states) const
        override;
25:
26:     void setNum_Recursion(int Num_Recursive);
27:     void setRotate(int angle);
28:     void setColor(sf::Color color);
29:
30:     int getNum_Recursion();
31:     int getRotate();
32:     sf::Color getColor();
33:
34:     void setOrigin(const sf::Vector2f& origin);
35:     void setOrigin(float origin);
36:
37:     void setPosition(float x, float y);
38:     void setPosition(const sf::Vector2f& position);
39:
40:     const sf::Vector2f& getPosition() const;
41:     const sf::Vector2f& getTarget() const;
42:
43: };
44: #endif
```

**PTree.cpp**

```
1: #include "PTree.hpp"
2: #include <SFML/Graphics.hpp>
3: #include <SFML/Window.hpp>
4: #include <iostream>
5: #include <cstdlib>
6: #include <string>
7: #include <cmath>
8: #include <vector>
9: using namespace std;
10:
11: PTree::PTree(int length, int Num_Recursive, sf::RenderTarget& target)
12: {
13:     L = length;
14:     N = Num_Recursive;
15:
16:     rect = sf::RectangleShape(sf::Vector2f(L, L));
17:
18:     rect.setOrigin(rect.getSize() / 2.f);
19:     rect.setPosition(target.getSize().x / 2.f, target.getSize().y - L /
20:         2.f);
21:     rect.setFillColor(sf::Color::Blue);
22: }
23: PTree::~PTree() {}
24:
25: void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const
26: {
27:     states.transform *= getTransform();
28:     target.draw(rect);
29: }
30:
31: void PTree::setNum_Recursion(int Num_Recursive)
32: {
33:     N = Num_Recursive;
34: }
35:
36: void PTree::setRotate(int angle)
37: {
38:     rect.setRotation(angle);
39: }
40:
41: void PTree::setColor(sf::Color color)
42: {
43:     rect.setFillColor(color);
44: }
45:
46: int PTree::getNum_Recursion()
47: {
48:     return N;
49: }
50:
51: int PTree::getRotate()
52: {
53:     return rect.getRotation();
54: }
55:
```

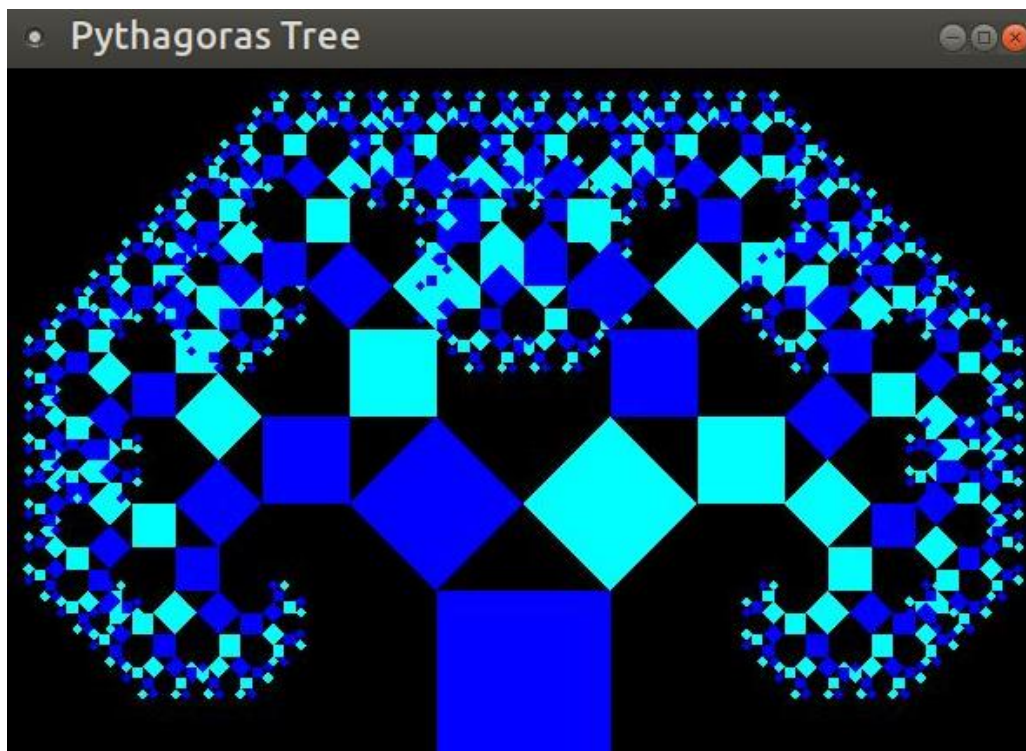
```

56: sf::Color PTree::getColor()
57: {
58:     return rect.getFillColor();
59: }
60:
61: void PTree::setOrigin(const sf::Vector2f& origin)
62: {
63:     rect.setOrigin(origin.x, origin.y);
64: }
65:
66: void PTree::setPosition(float x, float y)
67: {
68:     rect.setPosition(x, y);
69: }
70:
71: const sf::Vector2f& PTree::getPosition() const
72: {
73:     return rect.getPosition();
74: }

```

### Running Pythagoras Tree

```
ksok@cs4:~/Desktop/PS_Folder/PS2$ ./tree 100 11
```



## **PS3: N-Body Simulation**

### **Assignment:**

This assignment is about creating a N-body simulation. There are two parts in this assignment. In Part A, I began with modeling the planets by applying Law of gravity of Newton Law, and then extract the information such as coordinate, velocity, and mass of each planet from a given text file “planets.txt”, and then loads and displays a static universe on a SFML window display. The important thing in this part is using standard I/O to load and apply the data from the *planets.txt* to populate the sprite and texture and locate the image of each planet correctly in the SFML window. In Part B, I worked on making rotation motion and apply law of physics into simulation to create a motion of planets rotating around the sun. I also built a constructor that take two lines of arguments on command line which will be used as time phase and simulation time.

### **Key algorithms:**

On this assignment, SFML Library is the main part of simulation by setting up the coordinates system of solar system to display on the SFML window correctly. The key is location of each planet on the window, I began with identify the SFML window system coordinate with initial location at the top left corner of the screen. Then, convert the universe coordinate system to properly match with SFML window system. After that, I used Standard I/O to read data from command line that will take the filename of the text file “planets.txt” as an argument. Then extract the information from the text file and simulate the rotation and motion on the window.

### **What I learn from this assignment:**

I have learned to add motion on sprite and texture in SFML simulation, especially apply the law of physics into the program and some mathematic calculation were applied to set up the coordination, and rotation of each planet and putting together in a simulation.

**Makefile**

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



**main.cpp Tue Mar 10 20:37:01 2020 1**

```
1: #include <cstdlib>
2: #include <iostream>
3: #include <string>
4: #include <iomanip>
5: #include <sstream>
6: #include <cmath>
7: #include <time.h>
8: #include <fstream>
9: #include <memory>
10: #include <SFML/Audio.hpp>
11: #include <SFML/Graphics.hpp>
12: #include <SFML/Window.hpp>
13: #include "CelestialBody.hpp"
14: #include "Universe.hpp"
15:
16: int showcase();
17:
18: int main(int argc, char* argv[])
19: {
20:     if(argc != 3)
21:     {
22:         std::cerr<<"\nPlease input 2 arguments.\n";
23:     }
24:
25:     double timeExprie;
26:     double timeTotal;
27:     double time = 0;
28:
29:     //check first argument
30:     std::istringstream arg1(argv[1]);
31:     if(!(arg1>>timeExprie))
32:     {
33:         std::cerr << "\n" << argv[1] <<"IS NOT VALID" << std::endl;
34:         return showcase();
35:     }
36:
37:     //check first argument
38:     std::istringstream argv2(argv[2]);
39:     if(!(argv2>>timeTotal))
40:     {
41:         std::cerr << "\n" << argv[2] << "IS NOT VALID" << std::endl;
42:         return showcase();
43:     }
44:
45:     CelestialBody celestialbody;
46:     Universe uni;
47:     uni.setUnitPointer(celestialbody);
48:
49:     unsigned int window = 510;
50:     unsigned int numberOfParticle;
51:     double radUni;
52:
53:     std::cin >> numberOfParticle;
54:     std::cin >> radUni;
55:
56:     std::vector<std::shared_ptr<CelestialBody>> celbody;
```

```

57:     for(unsigned int i=0; i<numberOfParticle; i++){
58:         std::shared_ptr<CelestialBody> pointer(new CelestialBody());
59:         uni.smartPtr.push_back(pointer);
60:     }
61:
62:
63:     for(unsigned int i=0; i<numberOfParticle; i++){
64:         std::cin >>(*uni.smartPtr[i]);(*uni.smartPtr[i]).setWindowSize(window);
        (*uni.smartPtr[i]).setUniverseRadius(radUni);
65:     }
66:
67:
68:     //Window display
69:     sf::RenderWindow showWindow({window, window}, "UniverseSimulation",
        sf::Style::Close | sf::Style::Resize);
70:     showWindow.setFramerateLimit(35);
71:
72:     sf::Texture background;
73:     if(!background.loadFromFile("starfield.jpg"))
74:         return EXIT_FAILURE;
75:     sf::Sprite backgroundImage(background);
76:
77:     sf::Music music;
78:     music.openFromFile("Cradle.ogg");
79:     music.play();
80:
81:     sf::Vector2u backgroundDimension = background.getSize();
82:     double scaleWin = ((double>window/backgroundDimension.x);
83:     backgroundImage.setScale(scaleWin, scaleWin);
84:     backgroundImage.setTexture(background);
85:
86:     sf::Font font;
87:     if(!font.loadFromFile("Aller.ttf"))
88:         std::cout<<"Cannot load font." << std::endl;
89:
90:     std::ostringstream strings;
91:     sf::Text txt;
92:     txt.setFont(font);
93:     txt.setCharacterSize(21);
94:     txt.setFillColor(sf::Color::Yellow);
95:
96:     //field
97:     double gravity = 6.67e-11;
98:     double totalX, totalY;
99:     double radius, radiusSquared;
100:    double forceRadius;
101:    double forceX, forceY;
102:    double massA, massB;
103:
104:    //Display mode
105:    sf::Event event;
106:    while(showWindow.isOpen())
107:    {
108:        while(showWindow.pollEvent(event)) {
109:            if(event.type == sf::Event::Closed)
110:            {
111:                showWindow.close();

```

```

112:         break;
113:     }
114: }
115:
116: if(time <= timeExprie)
117: {
118:     showWindow.clear(sf::Color::Black);
119:     showWindow.draw(backgroundImage);
120:
121: for(unsigned int i=0; i<numberOfParticle; i++)
122: {
123:     forceRadius = 0;
124:     forceX = 0;
125:     forceY = 0;
126:     massA = (*uni.smartPtr[i]).getMass();
127:
128:     for(unsigned int j=0; j<numberOfParticle; j++)
129:     {
130:         if(j==i)
131:             continue;
132:
133:         totalX =
134:             (*uni.smartPtr[j]).getPosX() (*uni.smartPtr[i]).getPosX();
135:         totalY = (*uni.smartPtr[j]).getPosY() -
136:             (*uni.smartPtr[i]).getPosY();
137:         radiusSquared = (totalX * totalX) + (totalY * totalY);
138:         radius = sqrt(radiusSquared);
139:         massB = (*uni.smartPtr[j]).getMass();
140:
141:         forceRadius = (gravity * (massA * massB))/radiusSquared;
142:         forceX += forceRadius * (totalX/radius);
143:         forceY += forceRadius * (totalY/radius);
144:     }
145:
146:     (*uni.smartPtr[i]).setAccelerationX(forceX);
147:     (*uni.smartPtr[i]).setAccelerationY(forceY);
148:     (*uni.smartPtr[i]).step(timeTotal);
149: }
150:
151: for(unsigned int i=0; i<numberOfParticle; i++)
152: {
153:     showWindow.draw((*uni.smartPtr[i]));
154: }
155: //draw the text
156: strings.str("");
157: strings.clear();
158: strings<<time;
159: txt.setString(strings.str());
160: showWindow.draw(txt);
161: time += timeTotal;
162: showWindow.display();
163:
164: }
165: else
166: {

```

```

167: strings.str("");
168: strings.clear();
169: strings << time;
170: txt.setString(strings.str());
171: showWindow.draw(txt);
172: }
173:
174: }
175:
176: std::cout << "Time Limit: " << timeExprie << std::endl;
177: std::cout << "Total Time: " << timeTotal << std::endl;
178:
179: for(unsigned int i=0; i<numberOfParticle; i++)
180: std::cerr << (*uni.smartPtr[i]);
181:
182: music.stop();
183:
184: return 0;
185: }
186:
187: int showcase(){
188:
189: std::cout<<"Usage:\n\tNBody<time expire> <time total>"
    "<universe_file>\n"<<std::endl;
190: std::cout<<"Value:\n\tNBody 15778800.0 25000.0""< planets.txt\n>" <<
    std::endl;
191: return 0;
192: }

```

### **CelestialBody.hpp**

```
1: #ifndef CELESTIALBODY_HPP
2: #define CELESTIALBODY_HPP
3: #include <SFML/Graphics.hpp>
4: #include <SFML/Window.hpp>
5: #include <iostream>
6:
7: using namespace std;
8:
9: class CelestialBody : public sf::Drawable, sf::Transformable {
10: public:
11: //Constructor
12: CelestialBody();
13:
14: CelestialBody(double pX,double pY,double vX, double vY,double m,
    std::string& fileName);
15:
16: //Deconstructor
17: ~CelestialBody();
18:
19: void draw(sf::RenderTarget& target, sf::RenderStates states) const
    override;
20:
21: friend istream &operator >> (istream &input, CelestialBody &cel);
22: friend ostream &operator << (ostream &output,CelestialBody &cel);
23:
24: void setUniverseRadius(double radius);
25: void setImage(std::string imageName);
26: void setWindowSize(int windowScale);
27:
28: sf::Vector2f drawPixel() const;
29:
30: //mutator
31: void setPosX(double px);
32: void setPosY(double py);
33: void setVelX(double vx);
34: void setVelY(double vy);
35: void setMass(double m);
36:
37: double getRadius();
38: double getPosX();
39: double getPosY();
40: double getVelX();
41: double getVelY();
42: double getMass();
43:
44: void setAccelerationX(double force);
45: void setAccelerationY(double force);
46:
47: void step(double seconds);
48:
49:
50: public:
51:
52: sf::Texture imageTexture;
53: sf::Sprite imageSprite;
54: std::string imageName;
```

```
55:     sf::Image image;
56:
57:     int windowScale;
58:     double radius;
59:     double PosX;
60:     double PosY;
61:     double VelX;
62:     double VelY;
63:     double mass;
64:     double aX;
65:     double aY;
66:
67: };
68:
69: #endif
```

**CelestialBody.cpp Tue Mar 10 20:29:34 2020 1**

```
1: #include <iostream>
2: #include <cmath>
3: #include <SFML/Graphics.hpp>
4: #include <SFML/Window.hpp>
5: #include <iostream>
6: #include "CelestialBody.hpp"
7:
8: //Constructor
9: CelestialBody::CelestialBody()
10: {
11:     this->PosX = 0;
12:     this->PosY = 0;
13:     this->VelX = 0;
14:     this->VelY = 0;
15:     this->mass = 0;
16:     this->imageName = std::string();
17: }
18:
19: //Constructor
20: CelestialBody::CelestialBody(double px, double py, double vx,
    double vy, double m, std::string& fileName)
21: {
22:     setPosX(px);
23:     setPosY(py);
24:     setVelX(vx);
25:     setVelY(vy);
26:     setMass(m);
27:     setImage(fileName);
28: }
29:
30: //Destructor
31: CelestialBody::~CelestialBody(){}
32:
33: void CelestialBody::setWindowSize(int winScale)
34: {
35:     this->windowScale = winScale;
36: }
37:
38: void CelestialBody::setUniverseRadius(double rad)
39: {
40:     this->radius = rad;
41: }
42:
43: void CelestialBody::setImage(std::string imgName)
44: {
45:     imageName = imgName;
46:     image.loadFromFile(imageName);
47:     if(!imageTexture.loadFromImage(image))
48:     {exit(1);}
49:     imageSprite.setTexture(imageTexture);
50: }
51:
52: sf::Vector2f CelestialBody::drawPixel() const
53: {
54:
55:     double MeterPerPixel = radius/(windowScale/2.0);
```

```

56: double pixelX;
57: double pixelY;
58:
59: pixelX = (windowScale/2.0) + (PosX/MeterPerPixel);
60:
61: pixelY = (windowScale/2.0) - (PosY/MeterPerPixel);
62:
63: return sf::Vector2f(pixelX, pixelY);
64:
65: }
66:
67: //mutator
68: void CelestialBody::setPosX(double px)
69: {
70:     this->PosX = px;
71: }
72:
73: void CelestialBody::setPosY(double py)
74: {
75:     this->PosY = py;
76: }
77:
78: void CelestialBody::setVelX(double vx)
79: {
80:     this->VelX = vx;
81: }
82:
83: void CelestialBody::setVelY(double vy)
84: {
85:     this->VelY = vy;
86: }
87:
88: void CelestialBody::setMass(double m)
89: {
90:     this->mass = m;
91: }
92:
93: double CelestialBody::getRadius()
94: {
95:     return radius;
96: }
97: double CelestialBody::getPosX()
98: {
99:     return PosX;
100: }
101:
102: double CelestialBody::getPosY()
103: {
104:     return PosY;
105: }
106:
107: double CelestialBody::getVelX()
108: {
109:     return VelX;
110: }
111:
112: double CelestialBody::getVelY()

```



```

113: {
114:   return VelY;
115: }
116:
117: double CelestialBody::getMass()
118: {
119:   return mass;
120: }
121:
122: void CelestialBody::setAccelerationX(double force)
123: {
124:   this->aX = force / mass;
125: }
126:
127: void CelestialBody::setAccelerationY(double force)
128: {
129:   this->aY = force / mass;
130: }
131:
132: void CelestialBody::draw(sf::RenderTarget& target,
    sf::RenderStates states) const
133: {
134:   sf::Sprite storeAllImage = imageSprite;
135:   storeAllImage.setPosition(drawPixel());
136:   states.transform *= getTransform();
137:   target.draw(storeAllImage, states);
138: }
139: }
140:
141: void CelestialBody::step(double seconds)
142: {
143:   double vx = VelX + aX * seconds;
144:   double vy = VelY + aY * seconds;
145:   setVelX(vx);
146:   setVelY(vy);
147:
148:   double px = PosX + VelX * seconds;
149:   double py = PosY + VelY * seconds;
150:   setPosX(px);
151:   setPosY(py);
152: }
153: }
154:
155: istream &operator >> (istream &input, CelestialBody &cel)
156: {
157:
158:   input >> cel.PosX >> cel.PosY >> cel.VelX >> cel.VelY >> cel.mass
    >> cel.imageName;
159:   cel.setImage(cel.imageName);
160:
161:   return input;
162: }
163:
164: ostream &operator << (ostream &output, CelestialBody &cel)
165: {
166:   output << cel.PosX << " ";
167:   output << cel.PosY << " ";

```

```
168: output << cel.VelX << " ";
169: output << cel.VelY << " ";
170: output << cel.mass << " ";
171: output << cel.imageName << endl;
172:
173: return output;
174: }
```

**Universe.hpp**

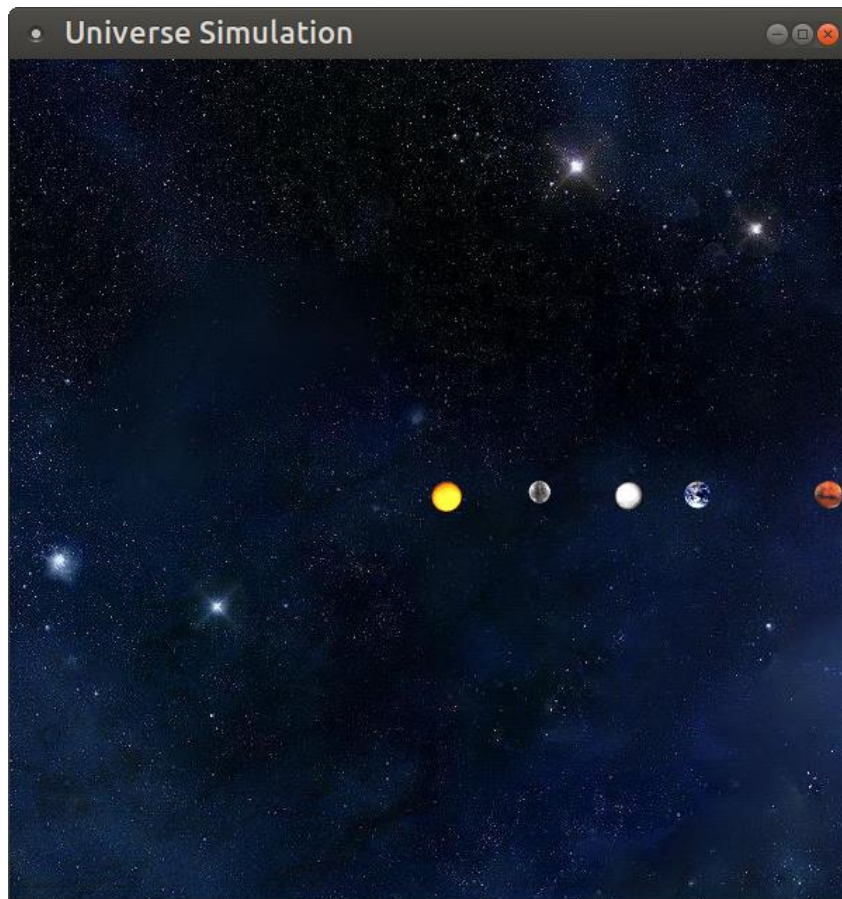
```
1: #ifndef UNIVERSE_HPP
2: #define UNIVERSE_HPP
3:
4: #include <SFML/Graphics.hpp>
5: #include <SFML/Window.hpp>
6: #include <memory>
7: #include <iostream>
8: #include "CelestialBody.hpp"
9:
10: class Universe: public CelestialBody
11: {
12: public:
13:     //Constructor
14:     Universe();
15:
16:     //Destructor
17:     ~Universe();
18:
19:     void setUnitPointer(const CelestialBody& unitptr);
20:
21:     public:
22:     //Smart Pointer
23:     std::vector<std::shared_ptr<CelestialBody>> smartPtr;
24:
25: };
26: #endif /* UNIVERSE_HPP */
```

### Universe.cpp

```
1: #include <iostream>
2: #include <cmath>
3: #include <SFML/Graphics.hpp>
4: #include <SFML/Window.hpp>
5: #include <iostream>
6: #include <memory>
7: #include "Universe.hpp"
8: #include "CelestialBody.hpp"
9:
10: //Constructor
11: Universe::Universe() {}
12:
13: //Destructor
14: Universe::~Universe() {}
15:
16: void Universe::setUnitPointer(const CelestialBody& unitptr)
17: {
18:     this->smartPtr.emplace_back(std::make_shared<CelestialBody>(unitptr));
19: }
```

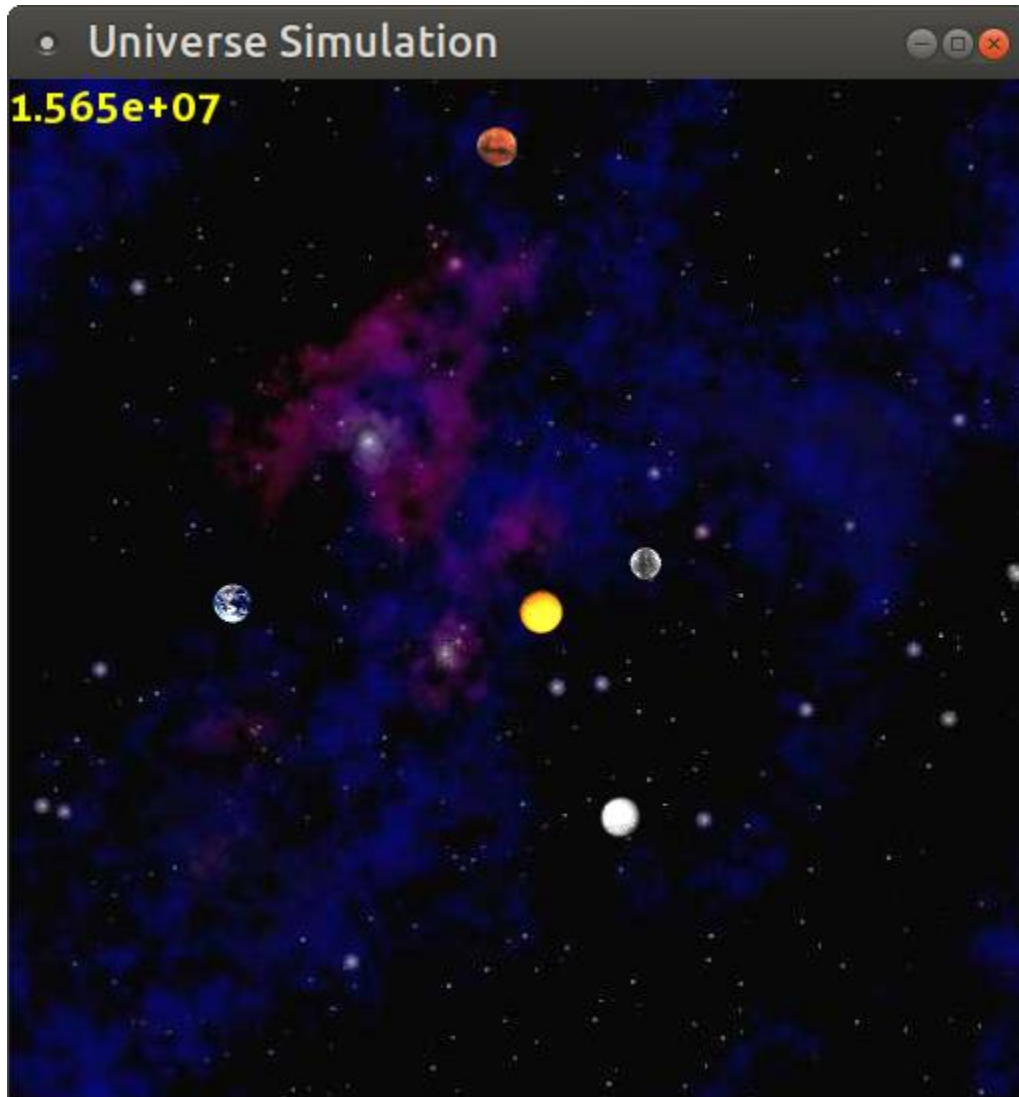
### PartA: Running a NBody Solar System Static Simulation

```
ksok@cs4:~/Desktop/PS_Folder/PS3a$ ./NBody < planets.txt
```



## PartB: Running Solar System Simulation

```
ksok@cs4:~/Desktop/PS_Folder/PS3b$ ./NBody 157788000.0 25000.0 < planets.txt
Setting vertical sync not supported
Time Limit: 1.57788e+08
Total Time: 25000
8.47084e+10 1.23044e+11 -24543.9 16977.5 5.974e+24 earth.gif
1.9834e+11 1.1181e+11 -11866.1 21002.6 6.419e+23 mars.gif
-3.6819e+10 -4.54701e+10 36372.6 -30406 3.302e+23 mercury.gif
485655 265590 0.157151 0.138789 1.989e+30 sun.gif
-7.7487e+08 1.07664e+11 -35172.7 -140.958 4.869e+24 venus.gif
```



## **PS4: Ring Buffer and Guitar Hero**

### **Assignment:**

This assignment is about to create a simulation of plucking a guitar string using Karplus-Strong algorithm. In Part A, I began with implementing RingBuffer and testing on Unit Test with multiple exceptions. I used vector of **sharepointer** to store the value of buffer. Then I used the unit testing to verify the RingBuffer class whether there is any error exception occur. I had also implemented some useful exception such as invalid argument and run time error exception. In Part B, I began with modeling Guitar String sound by using the concept of Karplu-Strong algorithms to generate plucking of guitar sounds. Furthermore, I implemented SFML library to set up a keyboard to produce sound simulation when particular key is pressed for which key is represented the different notes of guitar sound.

### **Key algorithms:**

The key algorithm is Karplus-Strong algorithm which can be used to emergence of physically modeled sound synthesis of a guitar sound simulation of vibration. The key is to simulate the vibration by maintaining a ring buffer of the N samples, the algorithms repeatedly removes the first sample from the buffer and adds to the end of the buffer the average of the deleted sample and the first same, and scaled by an energy decay factor of 0.996. As a result, it produces a sound like plucking guitar.

### **What I learn from this assignment:**

I have learned the Karplus-Strong algorithm by using it to simulate the plucking guitar sound. I also learn how to check my codes format using Google's Cpplink script. It is a very helpful tool to check the format of my code and help it organize and clean.

## Makefile

```
1: CC=g++
2: CFLAGS= -g -Wall -Werror -std=c++11 -pedantic
3: Boost=-lboost_unit_test_framework
4: LIBS1 = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
5:
6: all: KSGuitarSim test
7:
8: KSGuitarSim: KSGuitarSim.o StringSound.o RingBuffer.o
9: $(CC) $(CFLAGS) -o KSGuitarSim KSGuitarSim.o StringSound.o RingBuffer.o
   $(LIBS1)
10:
11: KSGuitarSim.o: KSGuitarSim.cpp StringSound.hpp RingBuffer.hpp
12: $(CC) $(CFLAGS) -c KSGuitarSim.cpp -o KSGuitarSim.o
13:
14: StringSound.o: StringSound.cpp StringSound.hpp RingBuffer.hpp
15: $(CC) $(CFLAGS) -c StringSound.cpp -o StringSound.o
16:
17: test: test.o RingBuffer.o
18: $(CC) $(CFLAGS) -o test test.o RingBuffer.o $(Boost)
19:
20: RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
21: $(CC) $(CFLAGS) -c RingBuffer.cpp -o RingBuffer.o
22:
23: test.o: test.cpp RingBuffer.hpp
24: $(CC) $(CFLAGS) -c test.cpp -o test.o
25:
26: clean:
27: rm *.o KSGuitarSim test
```

**KSGuitarSim.cpp**

```
1: #include <SFML/Graphics.hpp>
2: #include <SFML/System.hpp>
3: #include <SFML/Audio.hpp>
4: #include <SFML/Window.hpp>
5:
6: #include <math.h>
7: #include <limits.h>
8:
9: #include <iostream>
10: #include <string>
11: #include <exception>
12: #include <stdexcept>
13: #include <vector>
14:
15: #include "RingBuffer.hpp"
16: #include "StringSound.hpp"
17:
18: #define CONCERT_A 220.0
19: #define SAMPLES_PER_SEC 44100
20: const int keyboard_size = 37;
21:
22: std::vector<sf::Int16> makeSamples(StringSound gs) {
23:     std::vector<sf::Int16> samples;
24:
25:     gs.pluck();
26:     int duration = 8; // seconds
27:     int i;
28:     for (i= 0; i < SAMPLES_PER_SEC * duration; i++)
29:     {
30:         gs.tic();
31:         samples.push_back(gs.sample());
32:     }
33:
34:     return samples;
35: }
36:
37: int main(){
38:     sf::RenderWindow window(sf::VideoMode(500, 300), "SFML Plucked
        StringSound Lite");
39:     sf::Event event;
40:
41:     double freq;
42:     std::vector<sf::Int16> sample;
43:
44:     std::vector<sf::Sound> sounds(keyboard_size);
45:     std::vector<sf::SoundBuffer> ringbuffer(keyboard_size);
46:     std::vector<std::vector<sf::Int16>> samples(keyboard_size);
47:
48:     std::string keySounds = "q2we4r5ty7u8i9op-=[zxdcfvgbnjmk,.;/' ";
49:
50:     for (int i=0; i<(signed)keySounds.size(); i++){
51:         freq = CONCERT_A *pow(2, ((i-24.0)/12.0));
52:         StringSound gs1 = StringSound(freq);
53:
54:         sample = makeSamples(gs1);
55:         samples[i] = sample;
```



```

56:
57:     if(!ringbuffer[i].loadFromSamples(&samples[i][0],samples[i].size(),2,
SAMPLES_PER_SEC)){
58:         throw std::runtime_error("Error Notice:: cannot load sample.");
59:     }
60:
61:     sounds[i].setBuffer(ringbuffer[i]);
62: }
63:
64: while(window.isOpen()){
65:     while(window.pollEvent(event)){
66:
67:         if (event.type == sf::Event::TextEntered){
68:
69:             if(event.text.unicode < 128){
70:                 char sound = static_cast<char>(event.text.unicode);
71:
72:                 for(int i=0; i<(signed)keySounds.size();i++){
73:                     if(sound == keySounds[i]){
74:                         std::cout << "Key sound \' " << keySounds[i] <<"\' is
pressed." <<"\n";
75:                         sounds[i].play();
76:                         break;
77:                     }
78:                 }
79:             }
80:         }
81:     }
82:     window.clear();
83:     window.display();
84: }
85: }
86: return 0;
87: }

```

**RingBuffer.hpp**

```
1: #ifndef RINGBUFFER_H
2: #define RINGBUFFER_H
3: #include <stdint.h>
4: #include <iostream>
5: #include <string>
6: #include <sstream>
7: #include <exception>
8: #include <stdexcept>
9: #include <vector>
10:
11: using namespace std;
12:
13: class RingBuffer{
14:
15: private:
16:     int first_node;
17:     int last_node;
18:     int cap;
19:     int length;
20:     std::vector<int16_t> ringbuffer;
21:
22: public:
23:     //constructor- create empty ring buffer
24:     explicit RingBuffer(int capacity);
25:
26:     //destructor
27:     ~ RingBuffer();
28:
29:     //return num of items in buffer
30:     int size();
31:
32:     //check if buffer is empty
33:     bool isEmpty();
34:
35:     //check if buffer is full
36:     bool isFull();
37:
38:     //add item x to the end
39:     void enqueue(int16_t x);
40:
41:     //delect and return item from the front
42:     int16_t dequeue();
43:
44:     //return item from the front
45:     int16_t peek();
46:
47:     //function
48:     void display();
49:     void loop(const std::vector<int16_t>& values, void(*func)(int16_t));
50:
51: };
52:
53: #endif /* RINGBUFFER_H */
```

**RingBuffer.cpp**

```
1: #include <iostream>
2: #include <string>
3: #include <vector>
4: #include <cmath>
5:
6: #include "RingBuffer.hpp"
7:
8: using namespace std;
9:
10: //constructor
11: RingBuffer::RingBuffer(int fix_length){
12:
13:     if (fix_length < 1)
14:         throw std::invalid_argument("RingBuffer's fix_length have to be
            greater than zero.");
15:
16:     this->first_node = 0;
17:     this->last_node = 0;
18:     this->length = 0;
19:     this->cap = fix_length;
20:     ringbuffer.resize(cap);
21:
22:     return;
23: }
24:
25: //Destructor
26: RingBuffer::~RingBuffer(){}
27:
28: //return buffer size
29: int RingBuffer::size(){
30:     return length;
31: }
32:
33: //return true if buffer is full
34: bool RingBuffer::isFull(){
35:     if(size() == cap)
36:         return true;
37:     else
38:         return false;
39: }
40:
41: //return true if buffer is empty
42: bool RingBuffer::isEmpty(){
43:     if(size() == 0)
44:         return true;
45:     else
46:         return false;
47: }
48:
49: //Enqueue method
50: void RingBuffer::enqueue(int16_t x){
51:
52:     if(isFull())
53:     {
54:         throw
```

```

55:         std::runtime_error("Error Notice >> enqueue() Fail!!! >>
           BufferRing is FULL\n");
56:     }
57:
58:     if(last_node >= cap){
59:         last_node=0;
60:     }
61:
62:     //add to the end of vector
63:     ringbuffer.at(last_node) = x;
64:
65:     //increment the size and last elements
66:     last_node++;
67:     length++;
68: }
69:
70: int16_t RingBuffer::dequeue() {
71:
72:     if(isEmpty()){
73:         throw std::runtime_error("Error Notice >> dequeue() Fail!!! >>
           BufferRing is EMPTY\n");
74:     }
75:
76:     //remove from front
77:     int16_t begin = ringbuffer.at(first_node);
78:     ringbuffer.at(first_node) = 0;
79:
80:     //increment first
81:     first_node++;
82:     //decrement the size
83:     length--;
84:
85:     //varify if first need to be change back to zero
86:     if(first_node >= cap){
87:         first_node = 0;
88:     }
89:
90:     return begin;
91: }
92:
93: int16_t RingBuffer::peek() {
94:
95:     if(isEmpty())
96:         throw
97:             std::runtime_error("Error Notice >> peek() Fail!!! >> Buffer Ring
           is EMPTY\n");
98:
99:     return ringbuffer.at(first_node);
100: }
101:
102: //lamda expression
103: void RingBuffer::loop(const std::vector<int16_t>& buff,
           void(*function_lamda)(int16_t)) {
104:
105:     for(int16_t b: buff)
106:         function_lamda(b);
107: }

```

```

108:
109: void RingBuffer::display() {
110:
111:     std::cout << "\nShow all values in buffer \n";
112:
113:     //lamda expression loop in the vector
114:     loop(ringbuffer, [](int16_t r){ std::cout<< r << " "; });
115:
116:     std::cout<< "\n"<<std::endl;
117:
118:     std::cout << " First_Node....." <<first_node<< std::endl;
119:     std::cout << " Last_Node....." <<last_node<< std::endl;
120:     std::cout << " Fix Length....." <<cap<< std::endl;
121:     std::cout << " Length....." <<length<< std::endl;
122:     std::cout << " Vector Size....." <<ringbuffer.size()<<
        std::endl;
123:     std::cout << " Vector capacity....." <<ringbuffer.capacity()<<
        std::endl;
124:
125: }

```

**StringSound.hpp**

```
1: #ifndef STRINGSOUND_H
2: #define STRINGSOUND_H
3:
4: #include <SFML/Audio.hpp>
5: #include <SFML/Graphics.hpp>
6: #include <SFML/System.hpp>
7: #include <SFML/Window.hpp>
8:
9: #include <cmath>
10: #include <iostream>
11: #include <string>
12: #include <vector>
13: #include <random>
14: #include <stdio.h>
15:
16: #include "RingBuffer.hpp"
17:
18: const int SAMPLING_RATE = 44100;
19: const double ENERGY_DECAY_FACTOR = 0.996;
20:
21: class StringSound{
22:
23: private:
24:     int note;
25:     int timer;
26:     RingBuffer* ringB;
27:
28: public:
29:     //constructor -- create a guitar string sound of
30:     // a given frequency 44,100
31:     explicit StringSound(double frequency);
32:
33:     //create a guitar string with size and initial values
34:     //are given by the vector
35:     explicit StringSound(std::vector<sf::Int16> init);
36:
37:     //deconstruct
38:     ~ StringSound();
39:
40:     //pluck the guitar string by replacing the buffer with
41:     //random value.
42:     void pluck();
43:
44:     //advance the simulation one time step
45:     void tic();
46:
47:     //return the current sample
48:     sf::Int16 sample();
49:
50:     //return number of times tic was called
51:     int time();
52:
53: };
54:
55: #endif /* STRINGSOUND_H */
```

**StringSound.cpp**

```
1: #include "StringSound.hpp"
2: #include <vector>
3:
4:
5: StringSound::StringSound(double frequency):ringB(new RingBuffer(
    ceil(SAMPLING_RATE/frequency))) {
6:
7: //try the frequency, if frequency < 1, threw exception
8: if(frequency < 1)
9: {
10: throw
11: std::invalid_argument("Error Notice:: Frequency < 1. It must greater
    than 1");
12: }
13:
14: //create a soundnote
15: note = ceil(SAMPLING_RATE/frequency);
16:
17: //enqueue note
18: for(int i = 0; i<note; i++){
19:     ringB->enqueue((int16_t)0);
20: }
21:
22:     timer = 0;
23: }
24:
25: StringSound::StringSound(std::vector<sf::Int16> init):ringB(new
    RingBuffer(init.size())) {
26:
27: note = init.size();
28:
29: //Iterator to keep track of the vector
30: std::vector<sf::Int16>::iterator iIterator;
31:
32: //add all item to the vector
33: for (iIterator=init.begin(); iIterator < init.end(); iIterator++){
34:     ringB->enqueue((int16_t)*iIterator);
35: }
36:
37: //set _tic to 0 for the tic/time method
38: timer = 0;
39: }
40:
41: //Destructor
42: StringSound::~StringSound() {
43:
44: delete ringB;
45: }
46:
47: //pluck the guitar string by replacing the buffer with random values
48: void StringSound::pluck() {
49:
50: //remove note items
51: for(int i=0; i<note ; i++)
```

```

52: {
53:     ringB->dequeue();
54: }
55:
56: //Random number
57: std::random_device random;
58: for(int i=0; i<note ; i++){
59:     std::default_random_engine randomEngine(random());
60:     std::uniform_int_distribution<int> x(-32768,32767);
61:
62:     int number = x(randomEngine);
63:     ringB->enqueue((int16_t) (number));
64: }
65:     return;
66: }
67:
68: //advance the simulation one time step
69: void StringSound::tic(){
70:
71: //Get the first value, and the second value
72: int16_t first_note = ringB->dequeue();
73: int16_t second_note = ringB->peek();
74:
75: //Karplus_strong algoritm
76: int16_t karplus = ((first_note + second_note) / 2) * ENERGY_DECAY_FACTOR;
77:
78: //Now enqueue the Karplus-strong
79: ringB-> enqueue((sf::Int16) karplus);
80:
81: timer++;
82:
83: return;
84: }
85:
86: int StringSound::time(){
87:     return timer;
88: }
89:
90: sf::Int16 StringSound::sample(){
91:
92: //return the current sample using peek()
93: sf::Int16 currentSample = (sf::Int16) ringB->peek();
94:     return currentSample;
95: }
96:

```



**test.cpp**

```
1: #include <iostream>
2: #include <string>
3:
4: #include "RingBuffer.hpp"
5:
6: #define BOOST_TEST_DYN_LINK
7: #define BOOST_TEST_MODULE Main
8: #include <boost/test/unit_test.hpp>
9:
10: BOOST_AUTO_TEST_CASE(ringbuffer_constructor) {
11:
12:     BOOST_REQUIRE_NO_THROW(RingBuffer(100));
13:
14:     BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
15:     BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
16:     BOOST_REQUIRE_THROW(RingBuffer(-1), std::invalid_argument);
17:
18: }
19:
20: BOOST_AUTO_TEST_CASE(Size)
21: {
22:     RingBuffer test_ringbuffer(1);
23:
24:     BOOST_REQUIRE(test_ringbuffer.size()==0);
25:
26:     test_ringbuffer.enqueue(7);
27:
28:     BOOST_REQUIRE(test_ringbuffer.size()==1);
29:
30:     test_ringbuffer.dequeue();
31:
32:     BOOST_REQUIRE(test_ringbuffer.size()==0);
33: }
34:
35: BOOST_AUTO_TEST_CASE(isEmpty)
36: {
37:     RingBuffer test_ringbuffer_1(10);
38:     BOOST_REQUIRE(test_ringbuffer_1.isEmpty() == true);
39:
40:     RingBuffer test_ringbuffer_2(10);
41:     test_ringbuffer_2.enqueue(4);
42:     BOOST_REQUIRE(test_ringbuffer_2.isEmpty() == false);
43: }
44:
45: BOOST_AUTO_TEST_CASE(isFull)
46: {
47:     RingBuffer test_ringbuffer_1(8);
48:     BOOST_REQUIRE(test_ringbuffer_1.isFull() == false);
49:
50:     RingBuffer test_ringbuffer_2(1);
51:     test_ringbuffer_2.enqueue(8);
52:     BOOST_REQUIRE(test_ringbuffer_2.isFull() == true);
53: }
54:
55:
56: BOOST_AUTO_TEST_CASE(Enqueue)
```

```

57: {
58:   RingBuffer test_ringbuffer(5);
59:   BOOST_REQUIRE_NO_THROW(test_ringbuffer.enqueue(1));
60:   BOOST_REQUIRE_NO_THROW(test_ringbuffer.enqueue(2));
61:   BOOST_REQUIRE_NO_THROW(test_ringbuffer.enqueue(3));
62:   BOOST_REQUIRE_NO_THROW(test_ringbuffer.enqueue(4));
63:   BOOST_REQUIRE_NO_THROW(test_ringbuffer.enqueue(5));
64:
65:   BOOST_REQUIRE_THROW(test_ringbuffer.enqueue(6), std::runtime_error);
66: }
67:
68: BOOST_AUTO_TEST_CASE(Dequeue)
69: {
70:   RingBuffer test_ringbuffer(5);
71:
72:   test_ringbuffer.enqueue(10);
73:   test_ringbuffer.enqueue(20);
74:   test_ringbuffer.enqueue(30);
75:
76:   BOOST_REQUIRE(test_ringbuffer.dequeue() == 10);
77:   BOOST_REQUIRE(test_ringbuffer.dequeue() == 20);
78:   BOOST_REQUIRE(test_ringbuffer.dequeue() == 30);
79:   BOOST_REQUIRE_THROW(test_ringbuffer.dequeue(), std::runtime_error);
80:
81: }

```

#### Running Boost Test:

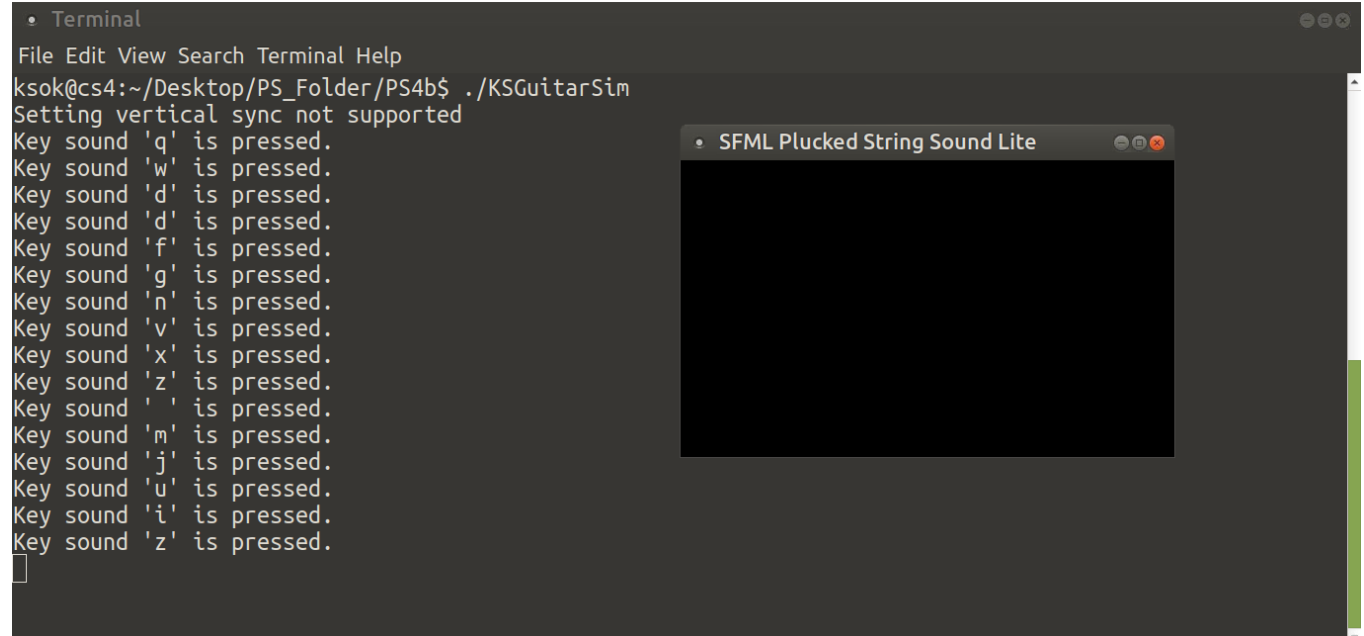
```

ksok@cs4:~/Desktop/PS_Folder/PS4b$ ./test
Running 6 test cases...

*** No errors detected

```

#### Running Guitar String Simulation:



The screenshot shows a terminal window with the following output:

```

• Terminal
File Edit View Search Terminal Help
ksok@cs4:~/Desktop/PS_Folder/PS4b$ ./KSGuitarSim
Setting vertical sync not supported
Key sound 'q' is pressed.
Key sound 'w' is pressed.
Key sound 'd' is pressed.
Key sound 'd' is pressed.
Key sound 'f' is pressed.
Key sound 'g' is pressed.
Key sound 'n' is pressed.
Key sound 'v' is pressed.
Key sound 'x' is pressed.
Key sound 'z' is pressed.
Key sound ' ' is pressed.
Key sound 'm' is pressed.
Key sound 'j' is pressed.
Key sound 'u' is pressed.
Key sound 'i' is pressed.
Key sound 'z' is pressed.

```

Overlaid on the terminal is a window titled "SFML Plucked String Sound Lite". The window is currently black, indicating it is not yet displaying a visual representation of the plucked string.

## **PS5 DNA Sequence Alignment**

### **Assignment:**

This assignment is about to compute the optimal sequence alignment of two DNA strings using dynamic programming by optimize the code, reduces time complexities. This problem focusses on comparing the DNA sequence alignment, and apply edited distance concept into the program.

### **Key algorithms:**

The key algorithms of this assignment are Dynamic Programming and Edit Distance concept. We optimize the code from Recursion Exponential to Linear to reduces time complexities. We also need to calculate the edit-distance between the two original string  $x$  and  $y$  by solving many edit-distance problems on smaller suffixes of the two strings. To compute the edit distance between two string, we use the notation  $x[i]$  to refer to character  $i$  of the string. The first pair of character in an optimal alignment of  $x[i..M]$  with  $y[i..N]$ . In order to compute  $opt[i][j]$  by taking the minimum of three quantities, we need to apply an equation:

$$opt[i][j] = \min \{ opt[i+1][j+1] + 0/1, opt[i+1][j] + 2, opt[i][j+1] + 2 \}$$

In this equation, we will assume  $i < M$  and  $j < N$ . This equation will help us aligning an empty string with another string of length  $k$  requires inserting  $k$  gaps, for a total cost of  $2k$ .

### **What I learn from this assignment:**

I have learned to align the DNA sequence using Dynamic programming approach which will help to reduce time complexity that very important for programmer to minimize the running of the programs. I also learned how to compute optimal sequence alignment of two DNA string.

**Makefile**

```
1: CC=g++
2: CFLAGS=-g -Wall -Werror -std=c++11
3: LIBS=-lsfml-system
4:
5: all: ED
6:
7: ED: main.o ED.o
8: $(CC) $(CFLAGS) -o ED main.o ED.o $(LIBS)
9:
10: main.o: main.cpp ED.hpp
11: $(CC) $(CFLAGS) -c main.cpp -o main.o
12:
13: ED.o: ED.cpp ED.hpp
14: $(CC) $(CFLAGS) -c ED.cpp -o ED.o
15:
16: clean:
17: rm *.o massif.out.* ED
```

**main.cpp**

```
1: #include <cstdlib>
2: #include <iostream>
3: #include <string>
4: #include <SFML/System.hpp>
5:
6: #include "ED.hpp"
7:
8: int main(int argc, char* argv[]) {
9:
10:     sf::Clock clock;
11:     sf::Time t;
12:
13:     std::string input_x;
14:     std::string input_y;
15:     std::string output;
16:     int distance;
17:
18:     std::cin >> input_x;
19:     std::cin >> input_y;
20:
21:     ED editDistance(input_x, input_y);
22:
23:     distance = editDistance.optDistance();
24:     output = editDistance.Alignment();
25:     std::cout << output;
26:
27:     t = clock.getElapsedTime();
28:
29:     std::cout << "Execution time is " << t.asSeconds() << " seconds. \n";
30:     std::cout << "Edit_distance is " << distance << "\n";
31:
32:     return 0;
33: }
34:
```

**ED.hpp**

```
1: #ifndef ED_H
2: #define ED_H
3:
4: #include <iostream>
5: #include <string>
6: #include <vector>
7:
8: class ED {
9:
10: private:
11:     std::string row;
12:     std::string cln;
13:     std::vector< std::vector<int> > cost;
14:
15: public:
16:
17:     ED(std::string str1, std::string str2);
18:     ~ED();
19:     int penalty(char a, char b);
20:     int min(int a , int b, int c);
21:     int optDistance();
22:     std::string Alignment();
23:
24:     //Extra points
25:     void output();
26:     void loop(const std::vector<std::vector<int> >& val, void(*d)(int));
27: };
28:
29: #endif /* ED_H */
```

**ED.cpp**

```
1: #include <iostream>
2: #include <string>
3: #include <vector>
4: #include <cmath>
5: #include "ED.hpp"
6:
7:
8: ED::ED(std::string str1, std::string str2){
9:
10:     row = str1;
11:     row.append("-");
12:     cln = str2;
13:     cln.append("-");
14:     std::vector< std::vector<int> >vect(str2.size()+1, std::vector<int>
        (str1.size()+1,-1));
15:     cost = vect;
16:
17: }
18:
19: ED::~~ED(){}
20:
21: std::string ED::Alignment(){
22:
23:     std::string cost_str;
24:     unsigned int cln_x = 0;
25:     unsigned int cln_y = 0;
26:
27:     while(cln_x < cln.size()-1 && cln_y < row.size()-1){
28:         if(cost.at(cln_x).at(cln_y) == cost.at(cln_x+1).at(cln_y+1)+
            penalty(row.at(cln_y),cln.at(cln_x)))
29:         {
30:             cost_str.push_back(row.at(cln_y));
31:             cost_str.push_back(' ');
32:             cost_str.push_back(cln.at(cln_x));
33:             cost_str.push_back(' ');
34:             cost_str.push_back(penalty(row.at(cln_y),cln.at(cln_x)) + '0');
35:             cost_str.push_back('\n');
36:             cln_x++;
37:             cln_y++;
38:         }
39:         else if(cost.at(cln_x).at(cln_y) == cost.at(cln_x).at(cln_y+1) + 2)
40:         {
41:             cost_str.push_back(row.at(cln_y));
42:             cost_str.push_back(' ');
43:             cost_str.push_back('-');
44:             cost_str.push_back(' ');
45:             cost_str.push_back('2');
46:             cost_str.push_back('\n');
47:             cln_y++;
48:         }
49:         else if(cost.at(cln_x).at(cln_y) == cost.at(cln_x+1).at(cln_y) + 2)
50:         {
51:             cost_str.push_back('-');
52:             cost_str.push_back(' ');
53:             cost_str.push_back(cln.at(cln_x));
54:             cost_str.push_back(' ');
```

```

55:         cost_str.push_back('2');
56:         cost_str.push_back('\n');
57:         cln_x++;
58:     }
59: }
60:
61: if(cln_y < row.size())
62: {
63:     cost_str.push_back(row.at(cln_y));
64:     cost_str.push_back(' ');
65:     cost_str.push_back('-');
66:     cost_str.push_back(' ');
67:     cost_str.push_back('2');
68:     cost_str.push_back('\n');
69:     cln_y++;
70: }
71: else if(cln_x < cln.size())
72: {
73:     cost_str.push_back('-');
74:     cost_str.push_back(' ');
75:     cost_str.push_back(cln.at(cln_x));
76:     cost_str.push_back('2');
77:     cost_str.push_back('\n');
78:     cln_x++;
79: }
80: return cost_str;
81: }
82:
83: int ED::min(int a , int b, int c){
84:     if( a<=b ){
85:         if(a <= c){
86:             return a;
87:         }else{
88:             return c;
89:         }
90:     }else{
91:         if( b <=c ){
92:             return b;
93:         }else{return c;}
94:     }
95: }
96:
97: int ED::optDistance(){
98:
99:     int dist = 0;
100:
101:     for(int i=row.size()-1; i>=0; i--)
102:     {
103:         cost.at(cln.size()-1).at(i)=dist;
104:         dist += 2;
105:     }
106:
107:     dist = 0;
108:
109:     for(int i=cln.size()-1; i>=0; i--)
110:     {
111:         cost.at(i).at(row.size()-1) = dist;

```



```

112:     dist += 2;
113: }
114: for(int i=row.size()-2;i>=0 ; i--)
115: {
116:     for(int j=cln.size()-2 ;j >= 0; j--)
117:     {
118:         cost.at(j).at(i)=min(cost.at(j+1).at(i+1)+penalty(row.at(i),
119:             cln.at(j)),
120:             cost.at(j).at(i+1)+2,cost.at(j+1).at(i)+2);
121:     }
122: return cost.at(0).at(0);
123: }
124:
125: int ED::penalty(char a, char b)
126: {
127:     if(a == b){return 0;}
128:     else if((a != b) && (a == '-' || b == '-'))
129:     {
130:         return 2;
131:     }else{return 1;}
132: }
133:
134: //Lambda Expression- Extra credit
135: void ED::loop(const std::vector<std::vector<int> >& v2d,
136:     void(*print)(int)){
137:     for(const std::vector<int>& info: v2d )
138:     {
139:         for(int r : info)
140:             print(r);
141:         std::cout <<"\n";
142:     }
143:
144: //Extra point
145: void ED::output() {
146:     std :: cout <<"Print all element from vector: "<< std::endl;
147:
148:     //lambda expression
149:     loop(cost,[](int epr ){ std::cout<< epr << std::endl; });
150: }

```

Using valgrind command to see running time and distance:

```
• Terminal
File Edit View Search Terminal Help
ksok@cs4:~/Desktop/PS_Folder/PS5$ valgrind --tool=massif ./ED < ecoli2500.txt
==13800== Massif, a heap profiler
==13800== Copyright (C) 2003-2017, and GNU GPL'd, by Nicholas Nethercote
==13800== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==13800== Command: ./ED
==13800==
```

Result of running on text file ecoli2500.txt:

Execution time is 3.07452 seconds

Edit distance is 118

```
C C 0
C T 1
G G 0
A A 0
G G 0
G G 0
G G 0
T T 0
G G 0
A A 0
T T 0
G G 0
T T 0
T T 0
G G 0
C C 0
C - 2
Execution time is 3.07452 seconds.
Edit distance is 118
==13800==
ksok@cs4:~/Desktop/PS_Folder/PS5$
```

Use the code that generated by valgrind command on ecoli2500.txt, which was ==13800==, to run `ms_print massif.out."code" | less`

```
ksok@cs4:~/Desktop/PS_Folder/PS5$ ms_print massif.out.13800 | less
```

```
Command: ./ED
Massif arguments: (none)
ms_print arguments: massif.out.13800
-----
MB
48.03^ #####
| #
| @#
| @#
| @#
| @#
| @#
| @#
| @#
| @#
| @#
| @#
:
:
```

## **PS6 Markov Model of Natural Language**

### **Assignment:**

This final assignment is about Markov model of natural language. This assignment is completed by analyzing an input text for transitions between k-grams that has a fixed number of characters, and the following letter. The concept of probabilistic model was used to predict a following letter in term of probabilities from a given particular k-gram series of letters. Lastly, this program used the model of natural language to generate a nonsense text that will surprisingly reasonable.

### **Key algorithms:**

The key algorithms are Markov Chain algorithm and Map concept in data structure. Then Markov algorithm was applied on current state of the k-gram text and then the next character will be selected at random stage by Markov modeling. All the letter that was generated is being appended by Markov Chain to produce a nonsense text. Map structure was used to hold all the data in k-gram table and then display the final out which was generated by Markov chain algorithm.

### **What I learn from this assignment:**

I have learned Markov chain algorithm which is an incredible algorithm that used in artificial intelligence field such as handwritten recognition and speech recognition which is very helpful for variety of application in AI and Machine Learning.

**Makefile**

```
1: CC=g++
2: CFLAGS= -g -Wall -Werror -std=c++17 -pedantic
3: Boost=-lboost_unit_test_framework
4: LIBS1 = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
5:
6: all: TextGenerator test
7:
8: TextGenerator: TextGenerator.o MModel.o
9: $(CC) $(CFLAGS) -o TextGenerator TextGenerator.o MModel.o $(LIBS1)
10:
11: TextGenerator.o: TextGenerator.cpp MModel.hpp
12: $(CC) $(CFLAGS) -c TextGenerator.cpp -o TextGenerator.o
13:
14: test: test.o MModel.o
15: $(CC) $(CFLAGS) -o test test.o MModel.o $(Boost)
16:
17: MModel.o: MModel.cpp MModel.hpp
18: $(CC) $(CFLAGS) -c MModel.cpp -o MModel.o
19:
20: test.o: test.cpp MModel.hpp
21: $(CC) $(CFLAGS) -c test.cpp -o test.o
22:
23: clean:
24: rm *.o TextGenerator test
```

# TextGenerator.cpp

```
1: /*
2: Copyright 2015 Fred Martin,
3: Y. Rykalova, 2020
4: */
5:
6: #include <iostream>
7: #include <string>
8: #include <exception>
9: #include <stdexcept>
10:
11: #include "MModel.hpp"
12:
13: int main(int argc, const char* argv[]) {
14:
15:     std::string sA(argv[1]);
16:     std::string sB(argv[2]);
17:
18:     int iNumOne = std::stoi(sA);
19:     int iNumTwo = std::stoi(sB);
20:
21:     std::string nInputString = "";
22:     std::string result = "";
23:     while (std::cin >> result) {
24:         nInputString += " " + result;
25:         result = "";
26:     }
27:     std::cout << "Input String is: \n";
28:
29:     std::cout << nInputString;
30:
31:     std::string resultT = "";
32:
33:     MModel _M(nInputString, iNumOne);
34:     resultT += _M.generate(nInputString.substr(0, iNumOne), iNumTwo);
35:     std::cout << "\n\nResult is : \n";
36:
37:     for (int s = 0; s < iNumTwo; s++) {
38:         std::cout << resultT[s];
39:     }
40:     std::cout << "\n";
41:
42:     return 0;
43: }
```

**MModel.hpp**

```
1: #ifndef MMODEL_H
2: #define MMODEL_H
3: #include <stdint.h>
4: #include <string>
5: #include <map>
6: #include <vector>
7:
8: class MModel {
9:
10: public:
11:     std::string new_Text;
12:     std::string input_Text;
13:     int order;
14:     std::map<std::string, int> kMap;
15:
16: public:
17:     MModel(std::string text,int k);
18:     ~MModel();
19:     int kOrder();
20:     int freq(std::string kgram);
21:     int freq(std::string kgram, char c);
22:     char kRand(std::string kgram);
23:     std::string generate(std::string kgram, int L);
24:     friend std::ostream& operator<< (std::ostream &out, MModel &M);
25:
26: };
27: #endif
28:
```

**MModel.cpp**

```
1: #include <string>
2: #include <stdexcept>
3: #include <vector>
4: #include <utility>
5: #include <iostream>
6: #include <time.h>
7: #include <stdio.h>
8: #include <stdlib.h>
9: #include <map>
10: #include <algorithm>
11:
12: #include "MModel.hpp"
13:
14: //constructor
15: MModel::MModel(std::string text,int k)
16: {
17:     srand(time(NULL));
18:     input_Text =text;
19:     order = k;
20:     char tChar;
21:     std::string sTemp;
22:     int loopCount = 0;
23:     bool boolCheck = false;
24:
25:     for (int i = 0; i < order;i++)
26:     {
27:         input_Text.push_back(text[i]);
28:     }
29:
30:     for (unsigned int i = 0; i < input_Text.length(); i++)
31:     {
32:         tChar = input_Text.at(i);
33:         for (unsigned int j = 0; j < new_Text.length(); j++)
34:         {
35:             if (new_Text.at(j) == tChar)
36:                 boolCheck = true;
37:         }
38:         if (!boolCheck)
39:             new_Text.push_back(tChar);
40:     }
41:
42:     std::sort(new_Text.begin(), new_Text.end());
43:     for (int i = order; i < order; i++)
44:     {
45:         for (unsigned int j = 0; j < input_Text.length(); j++)
46:         {
47:             sTemp.clear();
48:             sTemp = input_Text.substr(j, i);
49:             kMap.insert(std::pair<std::string, int>(sTemp, 0));
50:         }
51:     }
52:
53:     std::map<std::string, int>::iterator loopMap;
54:
55:     for (int i = order; i <= order + 1; i++) {
56:         for (unsigned int j = 0; j < text.length(); j++)
```



```

57:         {
58:             sTemp.clear();
59:             sTemp = input_Text.substr(j, i);
60:             loopMap = kMap.find(sTemp);
61:             loopCount = loopMap->second;
62:             loopCount++;
63:             kMap[sTemp] = loopCount;
64:         }
65:     }
66: }
67:
68: MModel::~MModel(){}
69:
70: int MModel::freq(std::string kgram)
71: {
72:     if (kgram.length() != static_cast<unsigned>(order))
73:         throw std::runtime_error("Error !! Wrong kgram length");
74:     std::map<std::string, int>::iterator loopMap;
75:     loopMap = kMap.find(kgram);
76:     if (loopMap == kMap.end())
77:         return 0;
78:     return loopMap->second;
79: }
80:
81: int MModel::freq(std::string kgram, char loopCount)
82: {
83:     if (kgram.length() != static_cast<unsigned>(order))
84:         throw std::runtime_error("Error !! Wrong kgram length");
85:     std::map<std::string, int>::iterator loopMap;
86:     kgram.push_back(loopCount);
87:     loopMap = kMap.find(kgram);
88:     if (loopMap == kMap.end())
89:         return 0;
90:     return loopMap->second;
91: }
92:
93: char MModel::kRand(std::string kgram) {
94:     if (kgram.length() != static_cast<unsigned>(order))
95:         throw std::runtime_error("Error !! Wrong kgram length");
96:
97:     std::map<std::string, int>::iterator loopMap = kMap.find(kgram);
98:     if (loopMap == kMap.end())
99:         throw std::runtime_error("Error");
100:
101:     double TF = 0;
102:     double FV = 0;
103:     int TempF = freq(kgram);
104:     int randI = rand() % TempF;
105:     double randDouble = static_cast<double>(randI)/TempF;
106:
107:     for (unsigned int i = 0; i < new_Text.length(); i++)
108:     {
109:         TF = static_cast<double>(freq(kgram, new_Text[i])) / TempF;
110:         if (randDouble < TF + FV && TF != 0)
111:         {
112:             return new_Text[i];
113:         }

```

```

114:     FV += TF;
115: }
116: throw std::runtime_error("Error!!");
117: }
118:
119:
120: std::string MModel::generate(std::string kgram, int L) {
121:     char randomChar;
122:     std::string randomString("");
123:
124:     randomString += "" + kgram;
125:     for (unsigned int i = 0; i < (L - (unsigned)order); i++)
126:     {
127:         randomChar = kRand(randomString.substr(i, order));
128:         randomString.push_back(randomChar);
129:     }
130:     return randomString;
131: }
132:
133: int MModel::kOrder()
134: {
135:     return order;
136: }
137:
138: std::ostream& operator<< (std::ostream &out, MModel &M)
139: {
140:     out << "\nString : "<< M.new_Text << " \n";
141:     out << "kOrder : " << M.order << " \n";
142:     out << "Map : \n\n";
143:
144:     std::map<std::string, int>::iterator loopMap;
145:     loopMap = M.kMap.begin();
146:
147:     while (loopMap != M.kMap.end()) {
148:         out << loopMap->first << "\t" << loopMap->second << "\n ";
149:         loopMap++;
150:     }
151:     return out;
152: }
153:

```

**test.cpp**

```
1: #include <iostream>
2: #include <string>
3: #include <exception>
4: #include <stdexcept>
5: #include "MModel.hpp"
6:
7: #define BOOST_TEST_DYN_LINK
8: #define BOOST_TEST_MODULE Main
9: #include <boost/test/unit_test.hpp>
10:
11: using namespace std;
12:
13: BOOST_AUTO_TEST_CASE(testing1) {
14:     BOOST_REQUIRE_NO_THROW(MModel("gagaaagagggagaggc", 2));
15:     MModel mTesting1("gagaaagagggagaggc", 2);
16:     BOOST_REQUIRE(mTesting1.kOrder() == 2);
17:     BOOST_REQUIRE_THROW(mTesting1.freq("", 'g'), std::runtime_error);
18:     BOOST_REQUIRE_THROW(mTesting1.freq("x", 'g'), std::runtime_error);
19:     BOOST_REQUIRE_THROW(mTesting1.freq(""), std::runtime_error);
20:     BOOST_REQUIRE_THROW(mTesting1.freq("x"), std::runtime_error);
21:
22: }
23:
24: BOOST_AUTO_TEST_CASE(testing2) {
25:     BOOST_REQUIRE_NO_THROW(MModel("cgagaaagagggagagg", 1));
26:
27:     MModel mTesting2("cgagaaagagggagagg", 1);
28:
29:     BOOST_REQUIRE(mTesting2.kOrder() == 1);
30:     BOOST_REQUIRE_THROW(mTesting2.freq(""), std::runtime_error);
31:     BOOST_REQUIRE_THROW(mTesting2.kRand("x"), std::runtime_error);
32:     BOOST_REQUIRE_THROW(mTesting2.kRand("xx"), std::runtime_error);
33:     BOOST_REQUIRE_THROW(mTesting2.freq("xx"), std::runtime_error);
34:
35: }
```

**Running test.cpp file:**

```
ksok@cs4:~/Desktop/PS_Folder/PS6$ ./test
Running 2 test cases...

*** No errors detected
ksok@cs4:~/Desktop/PS_Folder/PS6$
```

**Running TextGenerator on a text file input17.txt:**

```
ksok@cs4:~/Desktop/PS_Folder/PS6$ ./TextGenerator 2 11 < input17.txt
Input String is:
gagggagagggcgagaaa

Result is :
gaa gaa ga
ksok@cs4:~/Desktop/PS_Folder/PS6$
```