Vibishan Wigneswaran

COMP IV: Project Portfolio

Spring 2020

Table of Contents:

PS0: Hello World

The Assignment:

PS0 is an intro assignment in which the programmer works to output a basic image and allows us to accustom ourselves with the basics of SFML libraries as well as the use of Linux and compiling a basic program using a Makefile. The assignment gives us the opportunity to experiment with documentation and basic SFML functionality. During this program we also run Linux via virtual box which gives us the opportunity to get used to using command lines in terminal and the pros of Linux in programming.

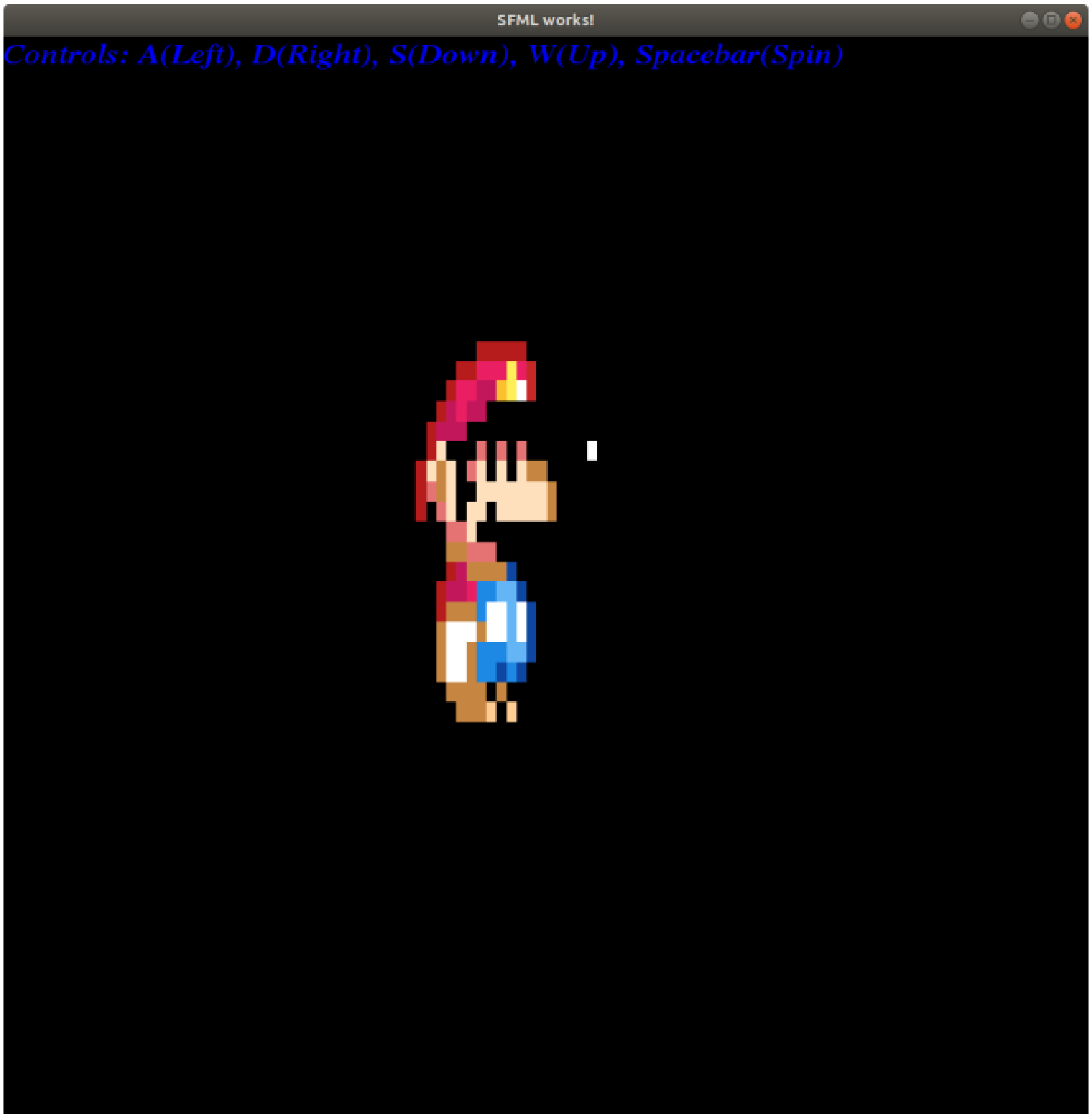
Central Designs:

The key concept of this program was the use of libraries through 3rd part libraries. By including SFML we can easily output images, sounds, and keyboard outputs. We also gain a basic understanding of how to pass images into textures and textures into sprites. In my attempt at the assignment I used a sprite of Mario from the classic game Super Mario Brothers and gave him the ability to move around using arrow key and a background.

What I learned:

This assignment although basic taught me many core abilities in programming using Linux. I learned to use the command line to install packages and create Makefiles to compile programs. I also learned to read and utilize documentation to get the most out of SFML library and compress programs using tar.gz.

Output:



Controls: A(Left), D(Right), S(Down), W(Up), Spacebar(Spin)

```
CC = g++
CFLAGS = -c -g -Og -Wall -Werror -ansi -pedantic
OBJ = main.o
DEPS =
LIBS = -lsfml-graphics -lsfml-window -lsfml-system
EXE = SFML-app

all: $(OBJ)
	$(CC) $(OBJ) -o $(EXE) $(LIBS)

%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -o $@ $<

clean:
	rm $(OBJ) $(EXE)
```

```cpp
#include <SFML/Graphics.hpp>

int main()
{
    //instruction texture
    sf::Font font;
    if(!font.loadFromFile("font.ttf")){
      return EXIT_FAILURE;
    }
    sf::Text text("Controls: A(Left), D(Right), S(Down), W(Up), Spacebar(Spin)",font, 25);
    text.setFillColor(sf::Color::Blue);

    sf::RenderWindow window(sf::VideoMode(1000, 1000), "SFML works!");
    sf::Texture texture;
    if(!texture.loadFromFile("sprite.png"))
      return EXIT_FAILURE;
    sf::Sprite sprite(texture);

    //Sprite initial settings
    sprite.scale(.25f, .50f);
    sprite.setPosition(10.f, 50.f);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed){
                window.close();
              }

            //keypress events for sprite
      //A to go Left
            if(sf::Keyboard::isKeyPressed(sf::Keyboard::A)){
        sprite.move(-10.f,0.f);
      }
      //D to go Right
      if(sf::Keyboard::isKeyPressed(sf::Keyboard::D)){
        sprite.move(10.f,0.f);
      }
      //W to go Up
      if(sf::Keyboard::isKeyPressed(sf::Keyboard::W)){
        sprite.move(0.f,-10.f);
      }
      //S to go Down
      if(sf::Keyboard::isKeyPressed(sf::Keyboard::S)){
        sprite.move(0.f,10.f);
      }
      //Space to Flip(rotate)
      if(sf::Keyboard::isKeyPressed(sf::Keyboard::Space)){
        sprite.rotate(90.f);
      }
        window.clear();
        window.draw(text);
        window.draw(sprite);
        window.display();
    }
  }

    return 0;
}
```

PS1a: Linear Feedback Shift Register

The Assignment:

In this assignment we learn to develop a Linear Feedback Shift Register where we XOR the left most digit of a seed to generate a new number to the right most digit. We complete this task using the implementation of classes to organize tasks and test the codes capabilities using Boost libraries.

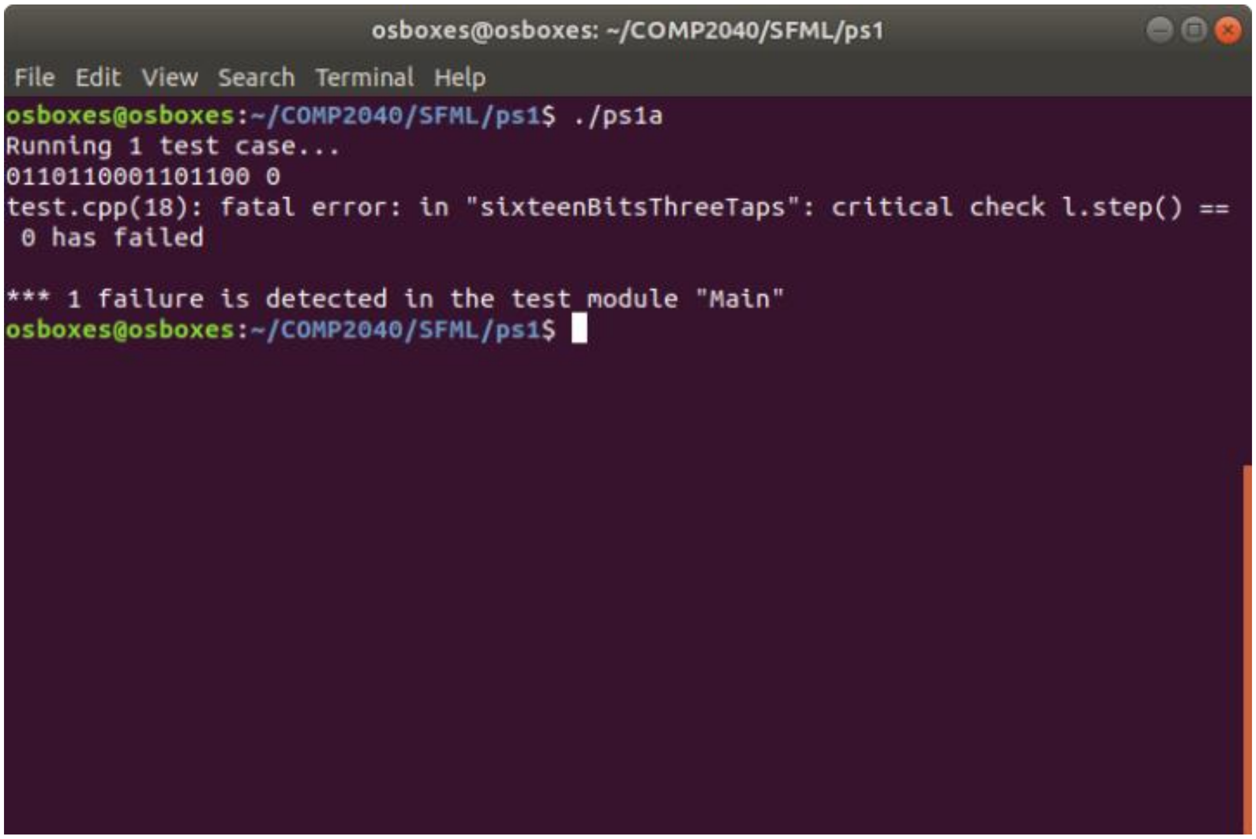
Central Designs:

Key concepts include the knowledge of how to XOR digits and implementing the functions of LFSR via a class. We use constructors, destructors, and class-based functions to generate new seeds and tap integer positions as well as XORing digits. By feeding the command line a string where the program taps the left most digit and generates a new digit for the right side of the string. The process is emulated through key functions including a step() function to XOR a new digit and a generate() function to create the new string. We also utilize Boost library tests to prove the capabilities of the program by developing cases of odd scenarios that the program can circumvent.

What I learned:

I learned what XORing through this assignment and how to implement logic-based designs into code. This program also taught me how to extensively debug a program to pinpoint and fix issues. I also learned how to use Boost libraries to develop tests to prove the capabilities of my contrasted program.

Output:


osboxes@osboxes: ~/COMP2040/SFML/ps1
File Edit View Search Terminal Help

```
osboxes@osboxes:~/COMP2040/SFML/ps1$ ./ps1a
Running 1 test case...
0110110001101100 0
test.cpp(18): fatal error: in "sixteenBitsThreeTaps": critical check l.step() ==
 0 has failed

*** 1 failure is detected in the test module "Main"
osboxes@osboxes:~/COMP2040/SFML/ps1$
```

```
CC= g++
#CFLAGS= -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system


all: PhotoMagic
	make clean

PhotoMagic: PhotoMagic.o FibLFSR.o
	$(CC) PhotoMagic.o FibLFSR.o -o PhotoMagic $(SFMLFLAGS)

PhotoMagic.o:	PhotoMagic.cpp FibLFSR.hpp
	$(CC) -c PhotoMagic.cpp FibLFSR.hpp $(CFLAGS)

FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
	$(CC) -c FibLFSR.cpp $(CFLAGS)

clean:
			rm *.o *.gch *~
```

```cpp
/*
Vibishan Wigneswaran
February 3rd, 2020
*/
#include <iostream>
#include <string>
#include <sstream>
#include "FibLFSR.hpp"

using namespace std;

int main(){
  cout << endl;
  cout << "Instructions" << endl;
  cout << "First Line is the original seed and right most number is the right most bit" << endl;
  cout << "Right most number after indentation on the last seed is generated binary to decimal conversion" << endl << endl;

  FibLFSR test1("1011011000110110");

//tests seed from manual
  cout << "Test 1" << endl;

  test1.generate(5);

  cout << endl;
  cout << test1 << endl << endl;

//tests smaller seed with smaller amount of binary digits
  cout << "Test 2" << endl;

  FibLFSR test2("101001110");
  test2.generate(3);
  cout << endl << test2 << endl << endl;

  //PS1b test


  return 0;
}
```

```
#ifndef FibLFSR_HPP
#define FibLFSR_HPP

#include <iostream>

using namespace std;

class FibLFSR {
public:
  FibLFSR(string Seed):seed(Seed), RMB(0), gen(0){}
  int step();
  int generate(int k);
  int printi();
  //overload << operator
  friend ostream& operator<< (ostream &out, const FibLFSR &FibLFSR);

private:
  string seed;
  int RMB;
  int gen;
};

#endif
```

```cpp
#include <iostream>
#include <cstdlib>
#include <string>
#include <cmath>
#include <sstream>
#include "FibLFSR.hpp"

using namespace std;

int FibLFSR::printi(){
  return stoi(seed, NULL, 2);
}

int FibLFSR::step(){
  int LMB=stoi(seed.substr(0,1));
  int tab1=stoi(seed.substr(2,1));
  int tab2=stoi(seed.substr (3,1));
  int tab3=stoi(seed.substr(5,1));
  //Find RMB
  int x = LMB;
  int y = tab1;
  if(x==y){
    x=0;
  }
  else{
    x=1;
  }
y = tab2;
if(x==y){
  x=0;
}
else{
  x=1;
}
y = tab3;
if(x==y){
  x=0;
}
else{
  x=1;
}
  RMB=x;
  //new seed
  int len=seed.length();
  for(int i = 0; i < len; i++)
  {
    seed.substr(i,1)=seed.substr(++i,1);
  }
  seed.erase(0,1);
  seed = seed + to_string(RMB);
}
int FibLFSR::generate(int k){
    int num = 0;
    for(int j = 0; j<k; j++)
    {
      step();
    }
    int l = seed.length();
    int temp = l - k;
    int t = k;
    string bitstr=seed.substr(temp,k);

    for(int i = 0; i!=k; i++){
      num += pow(2,i)*(stoi(bitstr.substr(--t,1)));
    }
  gen = num;
  }

ostream& operator<< (ostream &out, const FibLFSR &FibLFSR){
    out << FibLFSR.seed << " " << FibLFSR.gen;
    return out;
}
```

```
// Dr. Rykalova
// test.cpp for PS1a
// updated 1/31/2020

#include <iostream>
#include <string>
#include <sstream>

#include "FibLFSR.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {

  FibLFSR l("1011011000110110");
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);

  FibLFSR l2("1011011000110110");
  BOOST_REQUIRE(l2.generate(9) == 51);
}
```

PS1b: Linear Feedback Shift Register

The Assignment:

This part 2 of PS1 extends the capabilities of the Linear Feedback Shift Register into a program that can encode and decode images. By utilizing LFSR as a core computational ability we can use it to accurately encrypt and decrypt images that we feed into the program. Furthermore the program outputs both the encrypted and decrypted images via SFML.

Central Designs:

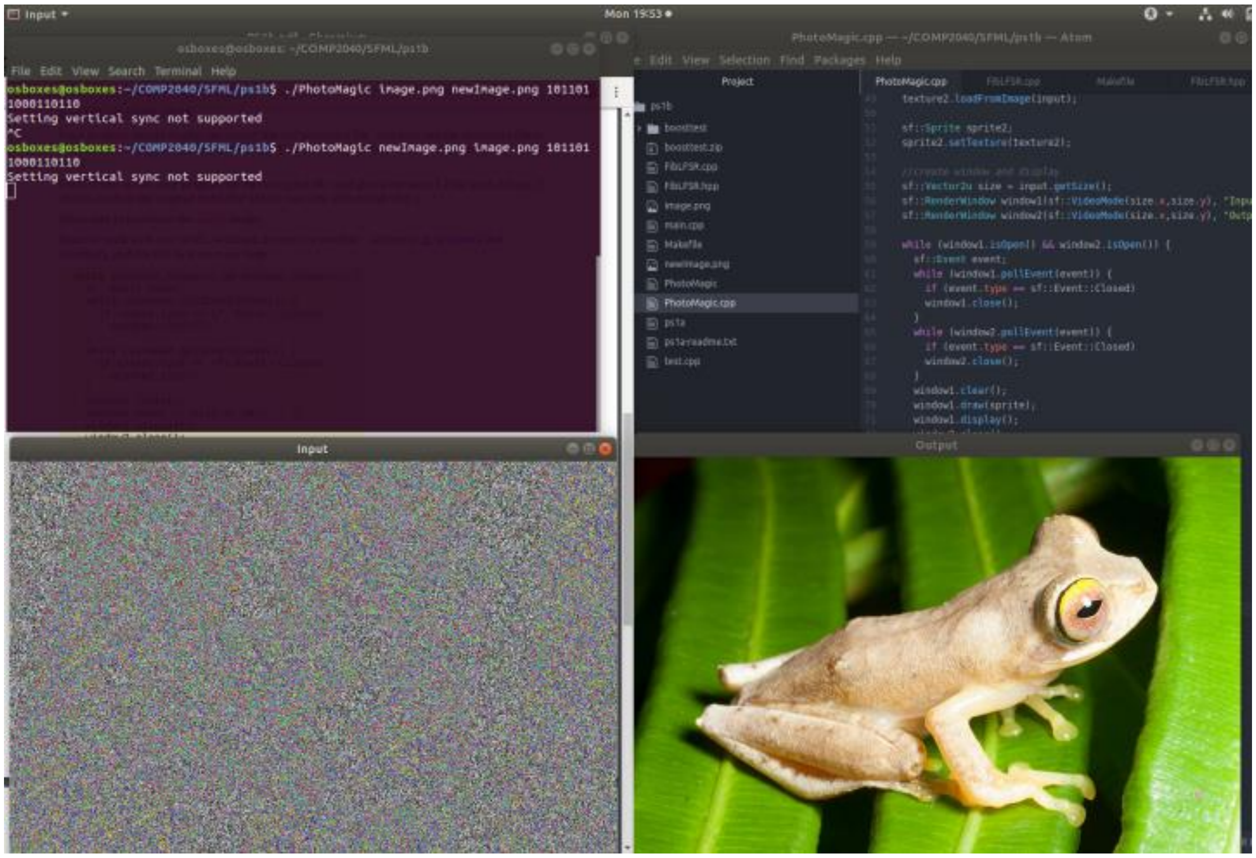
This assignment builds off the core of LFSR developed in PS1a and focuses on the utilization of the previously programmed LFSR class. We convert images into encrypted images using the LFSR class and then proceed to pass the image into a texture and into a sprite which we output into a window using SFML. We utilize seeds created via LFSR and SFML's pixel scanning and editing capabilities to change the coloration of the images or vice versa.

What I learned:

Through this program I learned to utilize complex functions and classes to create colorful results. By separately implementing the functions of the encoding and the LFSR, I can keep further develop the program while keeping the LFSR untouched. I also learned to use more advanced function from SFML library such as getPixel() and utilizing the red blue and green parameters of SFML's image library. I also learned to create Makefiles with more that 2 cpp and hpp files. This was a great program to learn how to build off of existing programs from.

Output:

Decode:



```
osboxes@osboxes:~/COMP2040/SFML/ps1b$ ./PhotoMagic image.png newImage.png 101101
1000110110
Setting vertical sync not supported
^C
osboxes@osboxes:~/COMP2040/SFML/ps1b$ ./PhotoMagic newImage.png image.png 101101
1000110110
Setting vertical sync not supported
```

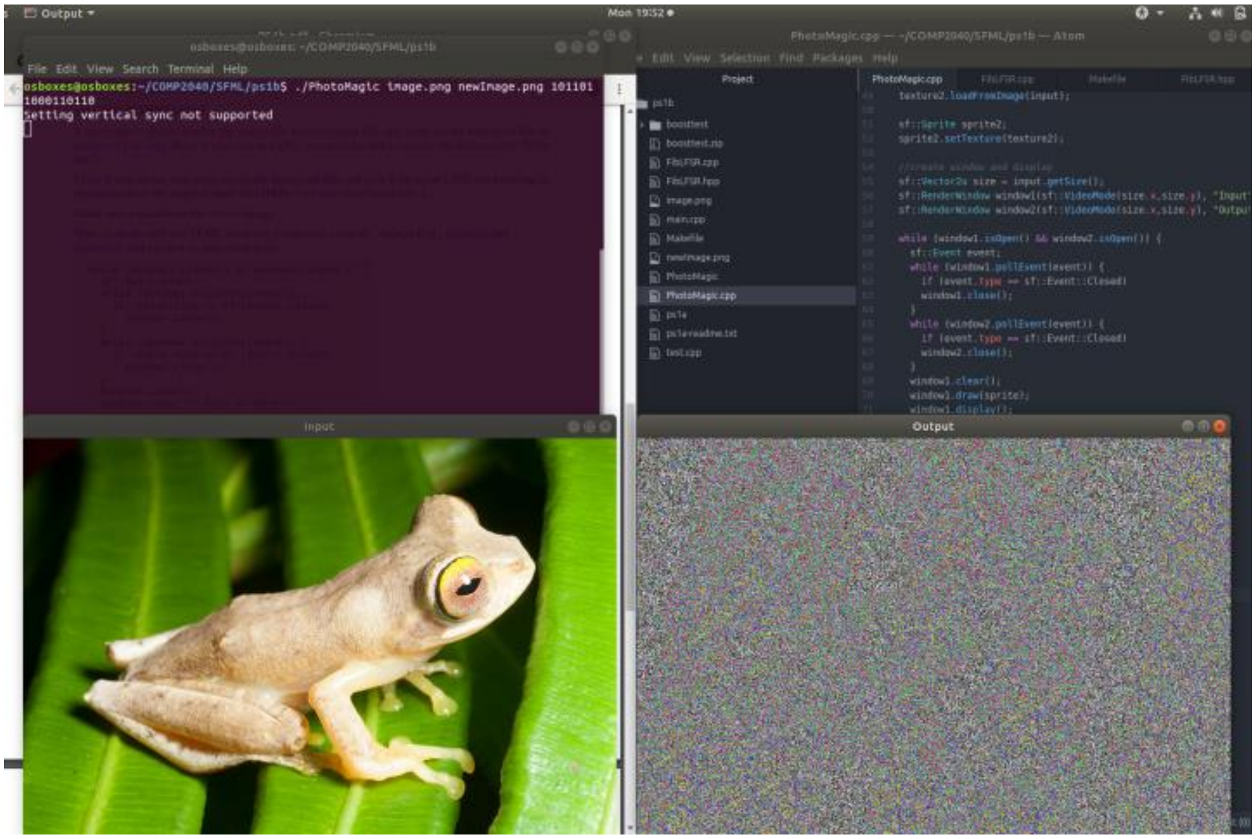
```
  texture2.loadFromImage(input);

  sf::Sprite sprite2;
  sprite2.setTexture(texture2);

  //create window and display
  sf::Vector2u size = input.getSize();
  sf::RenderWindow window1(sf::VideoMode(size.x,size.y), "Inpu
  sf::RenderWindow window2(sf::VideoMode(size.x,size.y), "Outp

  while (window1.isOpen() && window2.isOpen()) {
    sf::Event event;
    while (window1.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
      window1.close();
    }
    while (window2.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
      window2.close();
    }
    window1.clear();
    window1.draw(sprite);
    window1.display();
```

Encode:



```
osboxes@osboxes:~/COMP2040/SFML/ps1b$ ./PhotoMagic image.png newImage.png 101101
1000110110
Setting vertical sync not supported
```

```
    texture2.loadFromImage(input);

    sf::Sprite sprite2;
    sprite2.setTexture(texture2);

    //create window and display
    sf::Vector2u size = input.getSize();
    sf::RenderWindow window1(sf::VideoMode(size.x,size.y), "Input"
    sf::RenderWindow window2(sf::VideoMode(size.x,size.y), "Outpu

    while (window1.isOpen() && window2.isOpen()) {
      sf::Event event;
      while (window1.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
        window1.close();
      }
      while (window2.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
        window2.close();
      }
      window1.clear();
      window1.draw(sprite);
      window1.display();
```

```
CC= g++
#CFLAGS= -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system


all: PhotoMagic
	make clean

PhotoMagic: PhotoMagic.o FibLFSR.o
	$(CC) PhotoMagic.o FibLFSR.o -o PhotoMagic $(SFMLFLAGS)

PhotoMagic.o:   PhotoMagic.cpp FibLFSR.hpp
	$(CC) -c PhotoMagic.cpp FibLFSR.hpp $(CFLAGS)

FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
	$(CC) -c FibLFSR.cpp $(CFLAGS)

clean:
		rm *.o *.gch *~
```

```cpp

#include <iostream>
#include <string>
#include <sstream>
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include "FibLFSR.hpp"

using namespace std;

void transform(sf::Image&, FibLFSR*);

int main(int argc, char* argv[]){
  //checks for correct arguements in commang line
  if(argc != 4)
  {
    cout << "Failed to provide correct number of arguements\n";
    return -1;
  }

  // assign command line arguements to variables
  string infile(argv[1]);
  string outfile(argv[2]);
  string seed(argv[3]);

  FibLFSR seedgen(seed);

  //load image
  sf::Image input;

  if(!input.loadFromFile(infile))
  {
    return EXIT_FAILURE;
  }

  sf::Texture texture;
  texture.loadFromImage(input);

  transform(input,&seedgen);

  //convert image to sprite for display 1

  sf::Sprite sprite;
  sprite.setTexture(texture);

  //convert image to sprite for display 2
  sf::Texture texture2;
  texture2.loadFromImage(input);

  sf::Sprite sprite2;
  sprite2.setTexture(texture2);

  //create window and display
  sf::Vector2u size = input.getSize();
  sf::RenderWindow window1(sf::VideoMode(size.x,size.y), "Input");
  sf::RenderWindow window2(sf::VideoMode(size.x,size.y), "Output");

  while (window1.isOpen() && window2.isOpen()) {
    sf::Event event;
    while (window1.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
      window1.close();
    }
    while (window2.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
      window2.close();
    }
    window1.clear();
    window1.draw(sprite);
    window1.display();
    window2.clear();
    window2.draw(sprite2);
    window2.display();
}
  if(!input.saveToFile(outfile))
  {
    return EXIT_FAILURE;
  }
}

void transform(sf::Image& input, FibLFSR* LFSR){
  sf::Vector2u size = input.getSize();
  sf::Color color;

  //get size of image
  for(int countx = 0; countx < size.x; countx++){
    for(int county = 0; county < size.y; county++){
      color = input.getPixel(countx,county);
      color.r=color.r ^ LFSR->printi(); LFSR->step();
      color.b=color.b ^ LFSR->printi(); LFSR->step();
      color.g=color.g ^ LFSR->printi(); LFSR->step();
      input.setPixel(countx,county,color);
    }
  }

}
```

```
#ifndef FibLFSR_HPP
#define FibLFSR_HPP

#include <iostream>

using namespace std;

class FibLFSR {
public:
  FibLFSR(string Seed):seed(Seed), RMB(0), gen(0){}
  int step();
  int generate(int k);
  int printi();
  //overload << operator
  friend ostream& operator<< (ostream &out, const FibLFSR &FibLFSR);

private:
  string seed;
  int RMB;
  int gen;
};

#endif
```

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <cmath>
#include <sstream>
#include "FibLFSR.hpp"

using namespace std;

int FibLFSR::printi(){
  return stoi(seed, NULL, 2);
}

int FibLFSR::step(){
  int LMB=stoi(seed.substr(0,1));
  int tab1=stoi(seed.substr(2,1));
  int tab2=stoi(seed.substr (3,1));
  int tab3=stoi(seed.substr(5,1));
  //Find RMB
  int x = LMB;
  int y = tab1;
  if(x==y){
    x=0;
  }
  else{
    x=1;
  }
y = tab2;
if(x==y){
  x=0;
}
else{
  x=1;
}
y = tab3;
if(x==y){
  x=0;
}
else{
  x=1;
}
  RMB=x;
  //new seed
  int len=seed.length();
  for(int i = 0; i < len; i++)
  {
    seed.substr(i,1)=seed.substr(++i,1);
  }
  seed.erase(0,1);
  seed = seed + to_string(RMB);
}
int FibLFSR::generate(int k){
    int num = 0;
    for(int j = 0; j<k; j++)
    {
      step();
    }
    int l = seed.length();
    int temp = l - k;
    int t = k;
    string bitstr=seed.substr(temp,k);

    for(int i = 0; i!=k; i++){
      num += pow(2,i)*(stoi(bitstr.substr(--t,1)));
    }
  gen = num;
  }

ostream& operator<< (ostream &out, const FibLFSR &FibLFSR){
    out << FibLFSR.seed << " " << FibLFSR.gen;
    return out;
}
```

PS2: Recursive Graphics (Pythagoras Tree)

The Assignment:

In this assignment, we implement the theories of the Greek mathematician Pythagoras by creating a Pythagoras tree using SFML libraries. By using the theory that if the largest square has a size of L x L an entire Pythagoras tree will fit in a 6L * 4L box, we must find a way to generate a Pythagoras tree and output it in a window.

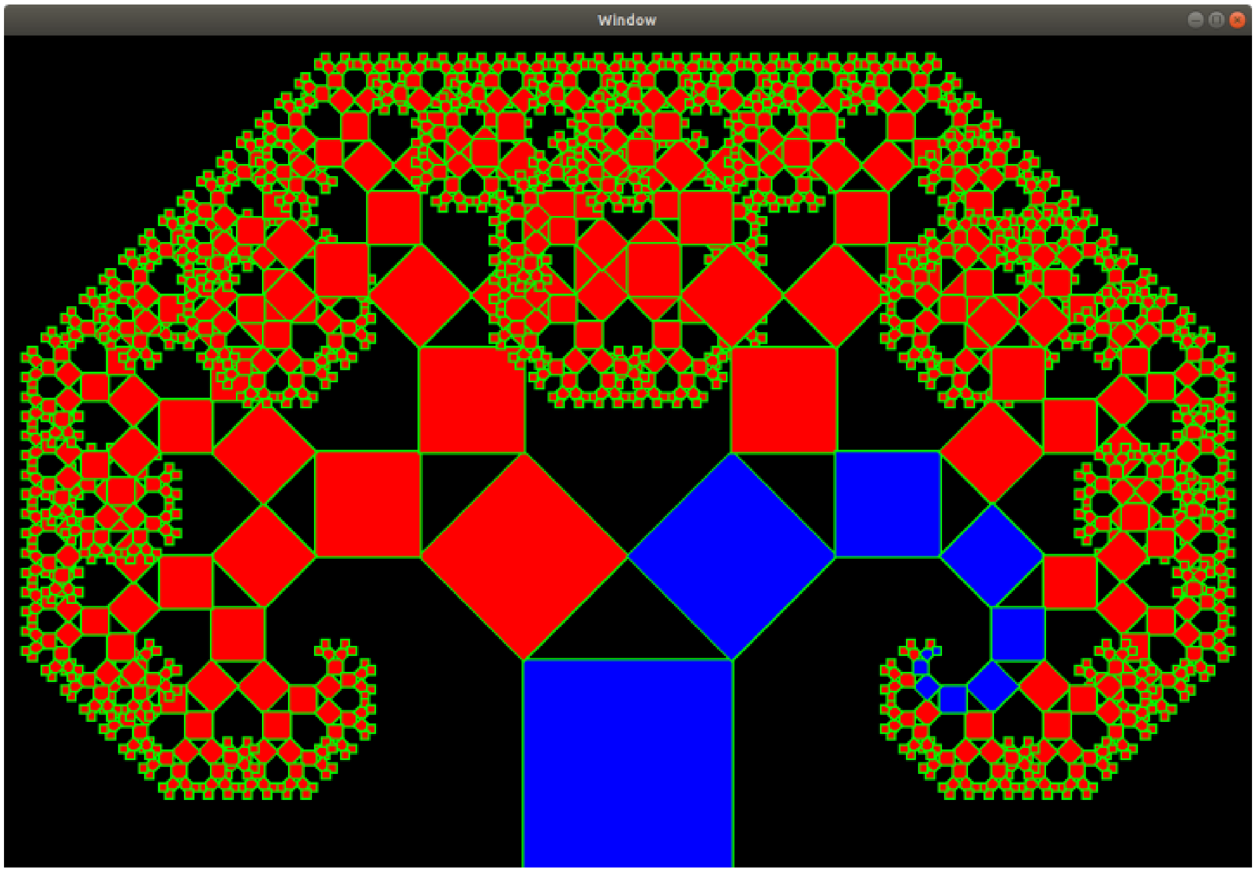
Central Designs:

Key concepts to this program heavily revolve around recursion. By creating a recursive class we can save hundreds of lines of code to since the basis of the creating each shape for this program is the same by utilizing parameters to loop the recursive function the correct amount of times we can develop a Pythagoras tree with a limited amount of code. We also touch upon virtual functions and how to create classes with abstract classes from 3rd party libraries.

What I learned:

Through this program I learned a lot about recursion and debugging for logic errors which resulted in excess or limited amounts of recursion. I also learned how to utilize abstract classes such as making a class sf::drawable so that I can use outputting and image malleability functions seamlessly. Although this program was extensively difficult it was good experience in learning to develop full fledge programs with eye catching results.

Output:



```
CC= g++
#CFLAGS= -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system


all: Tree
	make clean

Tree: PTree.o
	$(CC) PTree.o -o PTree $(SFMLFLAGS)

PTree.o: PTree.cpp PTree.hpp
	$(CC) -c PTree.cpp $(SFMLFLAGS)

clean:
	rm *.o
```

```
#ifndef PTree_HPP
#define PTree_HPP

//include libraries
#include <iostream>
#include <SFML/Graphics.hpp>
#include "PTree.hpp"

using namespace std;

//derived class from drawable
class PTree : public sf::Drawable{
public:
  //constructor
PTree(int b, int d, int a);
//destructor
~PTree(){}
//recursive function
void Ptree(const sf::RectangleShape pRect, int depth, int angle, sf::RenderTarget& point) const;

private:
  virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
  int base;
  int depth;
  int angle;
  sf::RectangleShape baseR;
};

#endif
```

```cpp
//including library
#include <cmath>
#include <cassert>
#include <iostream>
#include <fstream>
#include <vector>
#include <array>
#include <algorithm>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <sstream>
#include "PTree.hpp"

//namespace (not for sfml)
using namespace std;

//Define pi
const float pi = 3.14;

//class constructor
PTree::PTree(int length, int depth, int angle)
{
        //get depth
        this->depth = depth;
        this->angle = angle;
        baseR.setSize(sf::Vector2f(length, length));
        baseR.setPosition(2.5*length, 3 * length);
        baseR.setFillColor(sf::Color::Blue);
        baseR.setOutlineColor(sf::Color::Green);
        baseR.setOutlineThickness(2);

}
//recursive function
void PTree::Ptree(const sf::RectangleShape pRect, int depth, int angle, sf::Render
Target& point) const
{
        if (depth <= 0)
          {
                return;
          }

        auto& tf = pRect.getTransform();
        sf::RectangleShape lrectangle(pRect);

        lrectangle.setFillColor(sf::Color::Red);
        sf::Vector2f size = lrectangle.getSize();

        lrectangle.setSize(size * static_cast <float> (cos(angle * pi/180)));

        lrectangle.setOrigin(0, lrectangle.getSize().y);
        lrectangle.setPosition(tf.transformPoint({0, 0}));
        lrectangle.rotate(-angle);
        point.draw(lrectangle);
        Ptree(lrectangle, depth - 1, angle, point);

        sf::RectangleShape rrectangle(pRect);
        rrectangle.setSize(size * static_cast <float> (cos(angle * pi/180)));

        rrectangle.setOrigin(rrectangle.getSize());
        rrectangle.setPosition(tf.transformPoint({size.x, 0}));
        rrectangle.rotate(90 - angle);
        point.draw(rrectangle);
        Ptree(rrectangle, depth - 1, angle, point);
}

//virtual draw
void PTree::draw(sf::RenderTarget& point, sf::RenderStates states) const
{
        //draw tree
        point.draw(baseR, states);
        //start recursion
        Ptree(baseR, depth, angle, point);
}


int main(int argc, char* argv[])
{
  //checks for correct arguements in command line
  if(argc != 3)
  {
    cout << "Failed to provide correct number of arguements\n";
    return -1;
  }

  //assign command line arguements to variables
  string strbase(argv[1]); // L size of the base square
  string strdepth(argv[2]); // N the depth of the recursion
  double base = stoi(strbase);
  int depth = stoi(strdepth);

  //set window
  int Length = 6*base;
  int Width = 4*base;

  //create window and display
        sf::RenderWindow window(sf::VideoMode(Length, Width), "Window");

        int angle = 45;
        //create tree
        PTree newPTree(base, depth, angle);

        sf::Event event;
        while (window.isOpen())
        {
            sf::Event event;
            while (window.pollEvent(event))
            {
                if (event.type == sf::Event::Closed){
                    window.close();
                  }
          }
        //  window.clear();
          window.draw(newPTree);
          window.display();
        }
        return 0;
      }
```

PS3a: N-Body Simulation

The Assignment:

The purpose of this assignment is to simulate the universe in program. For this portion of the program we create a static universe. We utilize command arguments to input a simulation time and time step and create a Universe and Celestial Body class to simulate the Universe and planets respectively. The program must read txt files and store all the data revolving around each planet into its own object and output the solar system at its initial static position.

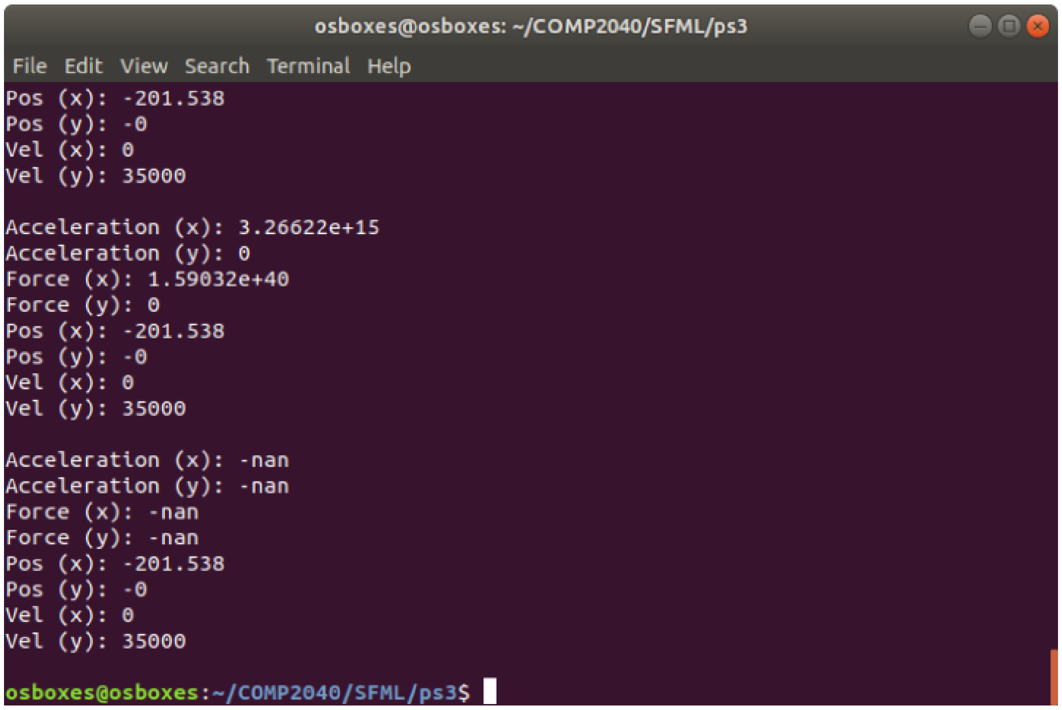
Central Designs:

Key concepts of this program include the storing command line arguments, creating separate classes that build off each other and using smart pointers to store class objects (celestial bodies in this case) into other classes. We also further delve into the usage of virtual functions to create drawable classes.

What I learned:

Through this assignment I learned to store objects I build in a class into another class. By storing celestial body objects which contain the information pertaining a planet into a universe class using a smart pointer I can easily change values while minding the ability of creating and destroying objects with little code. I also learned to properly set windows and utilize audio files for the output of my simulation.

Output:

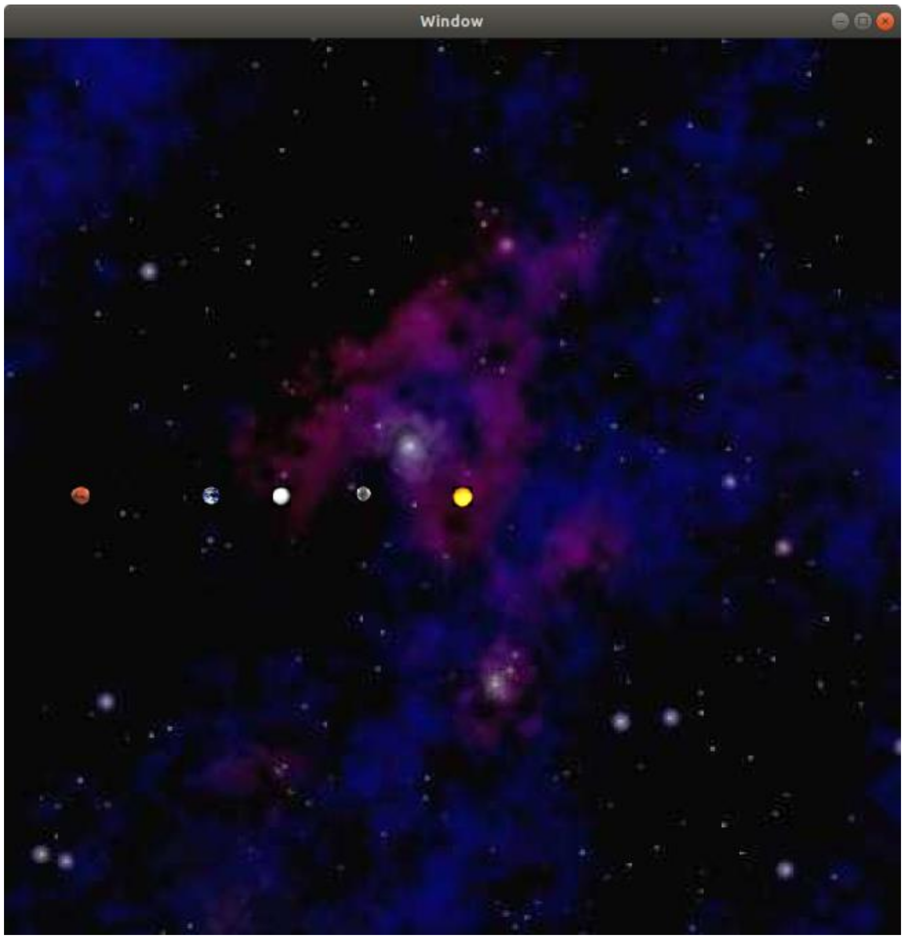

osboxes@osboxes: ~/COMP2040/SFML/ps3
File Edit View Search Terminal Help

```
Pos (x): -201.538
Pos (y): -0
Vel (x): 0
Vel (y): 35000

Acceleration (x): 3.26622e+15
Acceleration (y): 0
Force (x): 1.59032e+40
Force (y): 0
Pos (x): -201.538
Pos (y): -0
Vel (x): 0
Vel (y): 35000

Acceleration (x): -nan
Acceleration (y): -nan
Force (x): -nan
Force (y): -nan
Pos (x): -201.538
Pos (y): -0
Vel (x): 0
Vel (y): 35000

osboxes@osboxes:~/COMP2040/SFML/ps3$
```



```
CC= g++
#CFLAGS= -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system


all: NBody
	make clean

NBody: ps3.o
	$(CC) ps3.o -o NBody $(SFMLFLAGS)

NBody.o: ps3.cpp ps3.hpp
	$(CC) -c ps3.cpp $(SFMLFLAGS)

clean:
	rm *.o
```

```cpp
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <cstring>
#include "CelestialBody.hpp"

#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>

using namespace std;

int main(int argc, char* argv[]) {

  int windowSize = 500;
  double G = 6.67e-11;   //gravity constant
  double univRad;
  string input;
  int n;

  if(argc < 3) {
    cout << "nott enought arguements" << endl;
    return 1;
  } else if(argc > 3) {
    cout << "Too many arguements" << endl;
    return 1;
  }

  // Prepare time variables
  double timeTotal = atof(argv[1]);
  double timeStep = atof(argv[2]);
  double timeElapsed = 0.0;

  // Prepare the time text
  sf::Font courier_new;
  courier_new.loadFromFile("cour.ttf");

  sf::Text timeText("0.0", courier_new);

  // Prepare the image background
  sf::Image backgroundImage;
  backgroundImage.loadFromFile("starfield.jpg");

  sf::Texture backgroundTex;
  backgroundTex.loadFromImage(backgroundImage);

  sf::Sprite sprite(backgroundTex);

  // Get number of planets
  getline(cin, input);
  n = (atoi(input.c_str()));

  // Get size
  getline(cin, input);
  univRad = (atof(input.c_str()));

  CelestialBody temp(windowSize, univRad);

  vector<CelestialBody> universe;
  for(int i = 0; i < n; i++)
    universe.push_back(temp);

  for(int i = 0; i < n; i++) {
    getline(cin, input);
    istringstream iss(input);
    iss >> universe[i];
  }

  sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "Universe");

  while(window.isOpen()) {
    sf::Event event;
    while(window.pollEvent(event)) {
      if(event.type == sf::Event::Closed)
        window.close();
    }

    // Vector of Force, initialized to 0
    vector<double> forceX;
    vector<double> forceY;

    for(int i = 0; i < n; i++) {
      forceX.push_back(0.0);
      forceY.push_back(0.0);
    }

    // Calculate Acceleration
    for(unsigned i = 0; i < universe.size(); i++)
      for(unsigned j = 0; j < universe.size(); j++) {
        if(i == j) {
          forceX[i] += 0;
          forceY[i] += 0;
        }
        else {
          double deltaX = universe[j].getxPos() - universe[i].getxPos();
          double deltaY = universe[j].getyPos() - universe[i].getyPos();
          double r = sqrt(deltaX * deltaX + deltaY * deltaY);
          double F = (G * universe[i].getMass() * universe[j].getMass()) / (r * r)
;
          double Fx = F * (deltaX / r);
          double Fy = F * (deltaY / r);
          forceX[i] += Fx;
          forceY[i] += Fy;
        }
      }

    // Calculate New Velocity
    for(unsigned i = 0; i < universe.size(); i++) {
      double Ax = forceX[i] / universe[i].getMass();
      double Ay = forceY[i] / universe[i].getMass();

      universe[i].updateXVel(timeStep * Ax);
      universe[i].updateYVel(timeStep * Ay);

      // Update position
      universe[i].step(timeStep);
    }

    window.clear();

    window.draw(sprite);

    for(unsigned i = 0; i < universe.size(); i++)
      window.draw(universe[i]);

    timeElapsed += timeStep;
    if(timeElapsed > timeTotal)
      window.close();

    stringstream ss;
    ss << timeElapsed;
    timeText.setString(ss.str());

    // Draw Text
    window.draw(timeText);

    window.display();
  }

  for(unsigned i = 0; i < universe.size(); i++)
    cout << universe[i] << endl;
}
```

```
#ifndef PS3_HPP
#define PS3_HPP

#include <iostream>
#include <cstring>
#include <vector>
#include <memory>
#include <string>
#include <SFML/Graphics.hpp>

class CelestialBody : public sf::Drawable {
private:
  double xpos, ypos;
  double xvel, yvel;
  double mass;
  double universeSize;
  int windSize;
  std::string fname;
  sf::Sprite sprite;
  sf::Texture texture;

public:
  CelestialBody();
  CelestialBody(int _windowSize, double _universeSize);
  ~CelestialBody();

  void setWindowSize(int _size);
  void setUniverseSize(double _size);

  double getxPos();
  double getyPos();
  double getMass();

  void xVel(double newVel);
  void yVel(double newVel);

  void step(double time);
  void draw(sf::RenderTarget& target, sf::RenderStates states) const;

  friend std::istream &operator>>(std::istream &input, CelestialBody &arg);
  friend std::ostream &operator<<(std::ostream &output, CelestialBody &arg);
};

class Universe{
public:
  Universe(int r):radius(r){}
  ~Universe(){}


private:
  //virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
  int radius=0;
  //vector<shared_ptr<CelestialBody>> vPlanets;
};

#endif
```

```
//Vibishan Wigneswaran
#include <iostream>
#include <sstream>
#include <iomanip>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <cstring>

#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>

#include "ps3.hpp"

using namespace std;

int main(int argc, char* argv[]) {

  int windSize = 500;

  //grav const
  double G = 6.67e-11;
  double universeSize;

  string input;
  int n;

  if(argc < 3) {
    cout << "Missing command line arguements" << endl;
    return 1;
  } else if(argc > 3) {
    cout << "Too many command line arguements" << endl;
    return 1;
  }

  // variable declaration
  double timeTotal = atof(argv[1]);
  double timeStep = atof(argv[2]);
  double timeElapsed = 0.0;

  // Text for outpu
  sf::Font courier_new;
  courier_new.loadFromFile("arial.ttf");

  sf::Text timeText("0.0", courier_new);

  // background
  sf::Image backgroundI;
  backgroundI.loadFromFile("starfield.jpg");

  sf::Texture backgroundT;
  backgroundT.loadFromImage(backgroundI);

  sf::Sprite sprite(backgroundT);

  getline(cin, input);
  n = (atoi(input.c_str()));

  getline(cin, input);
  universeSize = (atof(input.c_str()));

  CelestialBody temp(windSize, universeSize);

  vector<CelestialBody> Univ;
  for(int i = 0; i < n; i++)
    Univ.push_back(temp);

  for(int i = 0; i < n; i++) {
    getline(cin, input);
    istringstream iss(input);
    iss >> Univ[i];
  }

  // play background music
  sf::Music music;
    if (!music.openFromFile("music.wav")){
        return EXIT_FAILURE;
      }
    music.setLoop(1);
    music.play();

  sf::RenderWindow window(sf::VideoMode(windSize, windSize), "Univ");

  while(window.isOpen()) {
    sf::Event event;
    while(window.pollEvent(event)) {
      if(event.type == sf::Event::Closed)

        window.close();

    }

    // Vector of Force, initialized to 0
    vector<double> Xforce;
    vector<double> Yforce;

    for(int i = 0; i < n; i++) {
      Xforce.push_back(0.0);
      Yforce.push_back(0.0);
    }

    // Calculate Acceleration
    for(unsigned i = 0; i < Univ.size(); i++)
      for(unsigned j = 0; j < Univ.size(); j++) {
        if(i == j) {
          Xforce[i] += 0;
          Yforce[i] += 0;
        }
        else {
          double dX = Univ[j].getxPos() - Univ[i].getxPos();
          double dY = Univ[j].getyPos() - Univ[i].getyPos();
          double r = sqrt(dX * dX + dY * dY);
          double F = (G * Univ[i].getMass() * Univ[j].getMass()) / (r * r);
          double Fx = F * (dX / r);
          double Fy = F * (dY / r);
          Xforce[i] += Fx;
          Yforce[i] += Fy;
        }
      }

    // Velocity
    for(unsigned i = 0; i < Univ.size(); i++) {
      double Ax = Xforce[i] / Univ[i].getMass();
      double Ay = Yforce[i] / Univ[i].getMass();

      Univ[i].xVel(timeStep * Ax);
      Univ[i].yVel(timeStep * Ay);

      // position
      Univ[i].step(timeStep);
    }

    window.clear();

    window.draw(sprite);

    //draw bodies
    for(unsigned i = 0; i < Univ.size(); i++){
      window.draw(Univ[i]);
    }

    timeElapsed += timeStep;
    if(timeElapsed > timeTotal)
      window.close();

    stringstream ss;
    ss << timeElapsed;
    timeText.setString(ss.str());

    window.draw(timeText);

    window.display();
  }

  for(unsigned i = 0; i < Univ.size(); i++)
    cout << Univ[i] << endl;
}


CelestialBody::CelestialBody() {
  xpos = 0;
  ypos = 0;
  xvel = 0;
  yvel = 0;
  mass = 0;
}

CelestialBody::CelestialBody(int _windowSize, double _universeSize) {
  xpos = 0;
  ypos = 0;
  xvel = 0;
  yvel = 0;
  mass = 0;
  windSize = _windowSize;
  universeSize = _universeSize;
}

CelestialBody::~CelestialBody() { }

void CelestialBody::setWindowSize(int _size) {
  windSize = _size;
}

void CelestialBody::setUniverseSize(double _size) {
  universeSize = _size;
}

double CelestialBody::getxPos() {
  return xpos;
}

double CelestialBody::getyPos() {
  return ypos;
}

double CelestialBody::getMass() {
  return mass;
}

void CelestialBody::xVel(double newVel) {
  xvel -= newVel;
}

void CelestialBody::yVel(double newVel) {
  yvel -= newVel;
}

void CelestialBody::step(double time) {
  xpos = xpos - time * xvel;
  ypos = ypos - time * yvel;

}

void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
  sf::Sprite sprite_temp = sprite;
  double ratio = (windSize / 2) / universeSize;
  double rxpos = xpos * ratio + (windSize / 2);
  double rypos = ypos * ratio + (windSize / 2);

  sprite_temp.setPosition(rxpos, rypos);

  target.draw(sprite_temp);
}

istream &operator>>(istream &input, CelestialBody &arg) {
  input >> arg.xpos;
  input >> arg.ypos;
  input >> arg.xvel;
  input >> arg.yvel;
  input >> arg.mass;
  input >> arg.fname;

  arg.texture.loadFromFile(arg.fname);
  arg.sprite.setTexture(arg.texture);

  return input;
}

ostream &operator<<(ostream &output, CelestialBody &arg) {
  output << setw(14) << arg.xpos;
  output << setw(14) << arg.ypos;
  output << setw(14) << arg.xvel;
  output << setw(14) << arg.yvel;
  output << setw(14) << arg.mass;
  output << setw(14) << arg.fname;

  return output;
}
```

PS3b: N-Body Simulation

The Assignment:

This assignment further builds on part a of the N-Body simulation. Having organized the data pertaining to the planets into CelestialBodies objects and these CelestialBodies Objects into a vector of points in a Universe object we must now simulate the planets movements using math and create an animation over a set period of time given to the command line.

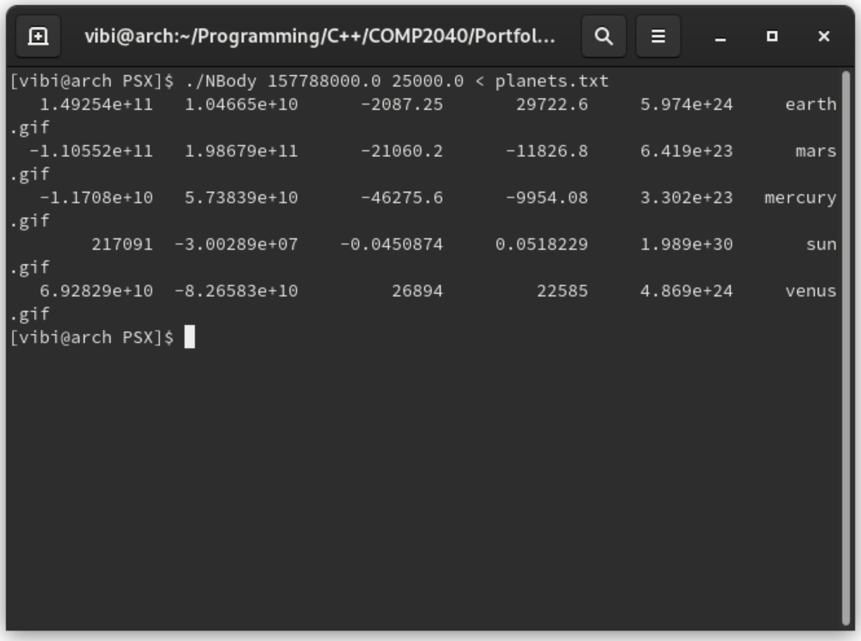
Central Designs:

Key concepts to this program include physics concepts including: Newton's Gravitational constant and newtons laws of motion as well as formulas that simulate acceleration and calculate for new positions of the X and Y axis of each planet.

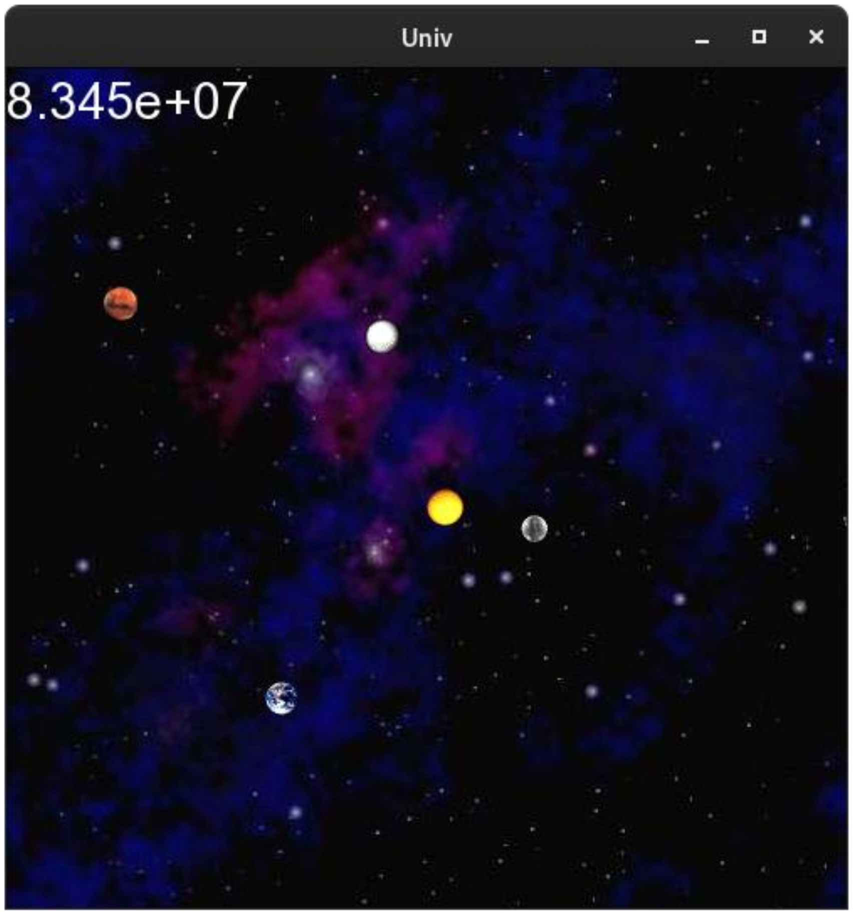
What I learned:

Through this assignment I learned how to extensively debug(as with the other assignments) for virtual errors such as errors with passing numbers and mathematical errors when utilizing formalas. I also learned the physics behind a solar system's simulation in more depth, since I needed to reflect the simulation into this program. Furthermore, I learned to utilize SFML's functions pertaining to vectors and audio for storing objects and playing music over my simulation for extra credit.

Output:


vibi@arch:~/Programming/C++/COMP2040/Portfol...

```
[vibi@arch PSX]$ ./NBody 157788000.0 25000.0 < planets.txt
   1.49254e+11   1.04665e+10      -2087.25       29722.6     5.974e+24     earth
.gif
  -1.10552e+11   1.98679e+11      -21060.2      -11826.8     6.419e+23      mars
.gif
   -1.1708e+10   5.73839e+10      -46275.6      -9954.08     3.302e+23   mercury
.gif
        217091  -3.00289e+07    -0.0450874     0.0518229     1.989e+30       sun
.gif
   6.92829e+10  -8.26583e+10         26894         22585     4.869e+24     venus
.gif
[vibi@arch PSX]$ 
```



```
CC= g++
#CFLAGS= -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system


all: NBody
	make clean

NBody: ps3.o
	$(CC) ps3.o -o NBody $(SFMLFLAGS)

NBody.o: ps3.cpp ps3.hpp
	$(CC) -c ps3.cpp $(SFMLFLAGS)

clean:
	rm *.o
```

```cpp
#include <iostream>
#include <sstream>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <cstring>
#include "CelestialBody.hpp"

#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>

using namespace std;

int main(int argc, char* argv[]) {

  int windowSize = 500;
  double G = 6.67e-11;   //gravity constant
  double univRad;
  string input;
  int n;

  if(argc < 3) {
    cout << "nott enought arguements" << endl;
    return 1;
  } else if(argc > 3) {
    cout << "Too many arguements" << endl;
    return 1;
  }

  // Prepare time variables
  double timeTotal = atof(argv[1]);
  double timeStep = atof(argv[2]);
  double timeElapsed = 0.0;

  // Prepare the time text
  sf::Font courier_new;
  courier_new.loadFromFile("cour.ttf");

  sf::Text timeText("0.0", courier_new);

  // Prepare the image background
  sf::Image backgroundImage;
  backgroundImage.loadFromFile("starfield.jpg");

  sf::Texture backgroundTex;
  backgroundTex.loadFromImage(backgroundImage);

  sf::Sprite sprite(backgroundTex);

  // Get number of planets
  getline(cin, input);
  n = (atoi(input.c_str()));

  // Get size
  getline(cin, input);
  univRad = (atof(input.c_str()));

  CelestialBody temp(windowSize, univRad);

  vector<CelestialBody> universe;
  for(int i = 0; i < n; i++)
    universe.push_back(temp);

  for(int i = 0; i < n; i++) {
    getline(cin, input);
    istringstream iss(input);
    iss >> universe[i];
  }

  sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "Universe");

  while(window.isOpen()) {
    sf::Event event;
    while(window.pollEvent(event)) {
      if(event.type == sf::Event::Closed)
        window.close();
    }

    // Vector of Force, initialized to 0
    vector<double> forceX;
    vector<double> forceY;

    for(int i = 0; i < n; i++) {
      forceX.push_back(0.0);
      forceY.push_back(0.0);
    }

    // Calculate Acceleration
    for(unsigned i = 0; i < universe.size(); i++)
      for(unsigned j = 0; j < universe.size(); j++) {
        if(i == j) {
          forceX[i] += 0;
          forceY[i] += 0;
        }
        else {
          double deltaX = universe[j].getxPos() - universe[i].getxPos();
          double deltaY = universe[j].getyPos() - universe[i].getyPos();
          double r = sqrt(deltaX * deltaX + deltaY * deltaY);
          double F = (G * universe[i].getMass() * universe[j].getMass()) / (r * r)
;
          double Fx = F * (deltaX / r);
          double Fy = F * (deltaY / r);
          forceX[i] += Fx;
          forceY[i] += Fy;
        }
      }

    // Calculate New Velocity
    for(unsigned i = 0; i < universe.size(); i++) {
      double Ax = forceX[i] / universe[i].getMass();
      double Ay = forceY[i] / universe[i].getMass();

      universe[i].updateXVel(timeStep * Ax);
      universe[i].updateYVel(timeStep * Ay);

      // Update position
      universe[i].step(timeStep);
    }

    window.clear();

    window.draw(sprite);

    for(unsigned i = 0; i < universe.size(); i++)
      window.draw(universe[i]);

    timeElapsed += timeStep;
    if(timeElapsed > timeTotal)
      window.close();

    stringstream ss;
    ss << timeElapsed;
    timeText.setString(ss.str());

    // Draw Text
    window.draw(timeText);

    window.display();
  }

  for(unsigned i = 0; i < universe.size(); i++)
    cout << universe[i] << endl;
}
```

```cpp
#ifndef PS3_HPP
#define PS3_HPP

#include <iostream>
#include <cstring>
#include <vector>
#include <memory>
#include <string>
#include <SFML/Graphics.hpp>

class CelestialBody : public sf::Drawable {
private:
  double xpos, ypos;
  double xvel, yvel;
  double mass;
  double universeSize;
  int windSize;
  std::string fname;
  sf::Sprite sprite;
  sf::Texture texture;

public:
  CelestialBody();
  CelestialBody(int _windowSize, double _universeSize);
  ~CelestialBody();

  void setWindowSize(int _size);
  void setUniverseSize(double _size);

  double getxPos();
  double getyPos();
  double getMass();

  void xVel(double newVel);
  void yVel(double newVel);

  void step(double time);
  void draw(sf::RenderTarget& target, sf::RenderStates states) const;

  friend std::istream &operator>>(std::istream &input, CelestialBody &arg);
  friend std::ostream &operator<<(std::ostream &output, CelestialBody &arg);
};

class Universe{
public:
  Universe(int r):radius(r){}
  ~Universe(){}


private:
  //virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
  int radius=0;
  //vector<shared_ptr<CelestialBody>> vPlanets;
};

#endif
```

```
//Vibishan Wigneswaran
#include <iostream>
#include <sstream>
#include <iomanip>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <cstring>

#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>

#include "ps3.hpp"

using namespace std;

int main(int argc, char* argv[]) {

  int windSize = 500;

  //grav const
  double G = 6.67e-11;
  double universeSize;

  string input;
  int n;

  if(argc < 3) {
    cout << "Missing command line arguements" << endl;
    return 1;
  } else if(argc > 3) {
    cout << "Too many command line arguements" << endl;
    return 1;
  }

  // variable declaration
  double timeTotal = atof(argv[1]);
  double timeStep = atof(argv[2]);
  double timeElapsed = 0.0;

  // Text for outpu
  sf::Font courier_new;
  courier_new.loadFromFile("arial.ttf");

  sf::Text timeText("0.0", courier_new);

  // background
  sf::Image backgroundI;
  backgroundI.loadFromFile("starfield.jpg");

  sf::Texture backgroundT;
  backgroundT.loadFromImage(backgroundI);

  sf::Sprite sprite(backgroundT);

  getline(cin, input);
  n = (atoi(input.c_str()));

  getline(cin, input);
  universeSize = (atof(input.c_str()));

  CelestialBody temp(windSize, universeSize);

  vector<CelestialBody> Univ;
  for(int i = 0; i < n; i++)
    Univ.push_back(temp);

  for(int i = 0; i < n; i++) {
    getline(cin, input);
    istringstream iss(input);
    iss >> Univ[i];
  }

  // play background music
  sf::Music music;
    if (!music.openFromFile("music.wav")){
        return EXIT_FAILURE;
      }
    music.setLoop(1);
    music.play();

  sf::RenderWindow window(sf::VideoMode(windSize, windSize), "Univ");

  while(window.isOpen()) {
    sf::Event event;
    while(window.pollEvent(event)) {
      if(event.type == sf::Event::Closed)

        window.close();

    }

    // Vector of Force, initialized to 0
    vector<double> Xforce;
    vector<double> Yforce;

    for(int i = 0; i < n; i++) {
      Xforce.push_back(0.0);
      Yforce.push_back(0.0);
    }

    // Calculate Acceleration
    for(unsigned i = 0; i < Univ.size(); i++)
      for(unsigned j = 0; j < Univ.size(); j++) {
        if(i == j) {
          Xforce[i] += 0;
          Yforce[i] += 0;
        }
        else {
          double dX = Univ[j].getxPos() - Univ[i].getxPos();
          double dY = Univ[j].getyPos() - Univ[i].getyPos();
          double r = sqrt(dX * dX + dY * dY);
          double F = (G * Univ[i].getMass() * Univ[j].getMass()) / (r * r);
          double Fx = F * (dX / r);
          double Fy = F * (dY / r);
          Xforce[i] += Fx;
          Yforce[i] += Fy;
        }
      }

    // Velocity
    for(unsigned i = 0; i < Univ.size(); i++) {
      double Ax = Xforce[i] / Univ[i].getMass();
      double Ay = Yforce[i] / Univ[i].getMass();

      Univ[i].xVel(timeStep * Ax);
      Univ[i].yVel(timeStep * Ay);

      // position
      Univ[i].step(timeStep);
    }

    window.clear();

    window.draw(sprite);

    //draw bodies
    for(unsigned i = 0; i < Univ.size(); i++){
```

```
      window.draw(Univ[i]);
    }

    timeElapsed += timeStep;
    if(timeElapsed > timeTotal)
      window.close();

    stringstream ss;
    ss << timeElapsed;
    timeText.setString(ss.str());

    window.draw(timeText);

    window.display();
  }

  for(unsigned i = 0; i < Univ.size(); i++)
    cout << Univ[i] << endl;
}


CelestialBody::CelestialBody() {
  xpos = 0;
  ypos = 0;
  xvel = 0;
  yvel = 0;
  mass = 0;
}

CelestialBody::CelestialBody(int _windowSize, double _universeSize) {
  xpos = 0;
  ypos = 0;
  xvel = 0;
  yvel = 0;
  mass = 0;
  windSize = _windowSize;
  universeSize = _universeSize;
}

CelestialBody::~CelestialBody() { }

void CelestialBody::setWindowSize(int _size) {
  windSize = _size;
}

void CelestialBody::setUniverseSize(double _size) {
  universeSize = _size;
}

double CelestialBody::getxPos() {
  return xpos;
}

double CelestialBody::getyPos() {
  return ypos;
}

double CelestialBody::getMass() {
  return mass;
}

void CelestialBody::xVel(double newVel) {
  xvel -= newVel;
}

void CelestialBody::yVel(double newVel) {
  yvel -= newVel;
}

void CelestialBody::step(double time) {
  xpos = xpos - time * xvel;
  ypos = ypos - time * yvel;

}

void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
  sf::Sprite sprite_temp = sprite;
  double ratio = (windSize / 2) / universeSize;
  double rxpos = xpos * ratio + (windSize / 2);
  double rypos = ypos * ratio + (windSize / 2);

  sprite_temp.setPosition(rxpos, rypos);

  target.draw(sprite_temp);
}

istream &operator>>(istream &input, CelestialBody &arg) {
  input >> arg.xpos;
  input >> arg.ypos;
  input >> arg.xvel;
  input >> arg.yvel;
  input >> arg.mass;
  input >> arg.fname;

  arg.texture.loadFromFile(arg.fname);
  arg.sprite.setTexture(arg.texture);

  return input;
}

ostream &operator<<(ostream &output, CelestialBody &arg) {
  output << setw(14) << arg.xpos;
  output << setw(14) << arg.ypos;
  output << setw(14) << arg.xvel;
  output << setw(14) << arg.yvel;
  output << setw(14) << arg.mass;
  output << setw(14) << arg.fname;

  return output;
}
```

PS4a: Synthesizing a Plucked String Sound

The Assignment:

The purpose of this assignment is to create a RingBuffer class to store 16 bit values. The program should have error exception abilities to avoid common errors and must prove its capabilities through a boost test. The RingBuffer will utilize the stack based programming to queue and dequeue elements and check if the buffer is full or empty.

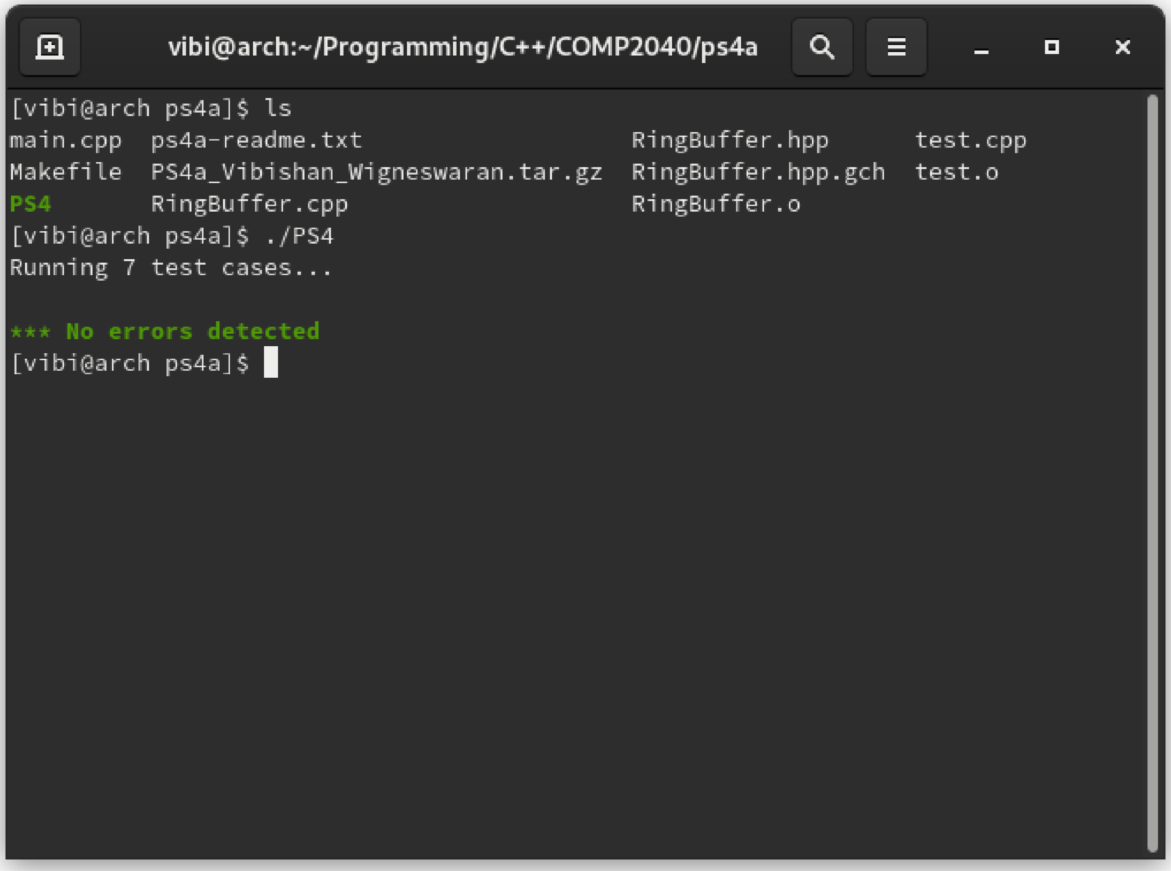
Central Designs:

Key concepts of this program include stack programming in which we utilize the concept of expanding an array to queue new objects into and as well as pull from the stack. This will allow us to simplify the process of adding and removing objects into the buffer. We also utilize exception handling to easily catch errors and debug with ease. Another key capability for the program is its ability to pass specific tests to ensure that the program can handle any case given to it.

What I learned:

This assignment taught me how to implement stack base programming structures in the form of a class. It also further enhanced my experience creating test cases using boost libraries and reinforced my usage of exception handling to avoid user induced faults in my programs.

Output:



```
[vibi@arch ps4a]$ ls
main.cpp  ps4a-readme.txt                 RingBuffer.hpp      test.cpp
Makefile  PS4a_Vibishan_Wigneswaran.tar.gz  RingBuffer.hpp.gch  test.o
PS4       RingBuffer.cpp                  RingBuffer.o
[vibi@arch ps4a]$ ./PS4
Running 7 test cases...

*** No errors detected
[vibi@arch ps4a]$ 
```

```
CC=g++
CFLAGS= -std=c++14 -Wall -Werror -pedantic
OBJ=KSGuitarSim.o RingBuffer.o StringSound.o
LIBS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
DEPS=
EXE=KSGuitarSim

all: $(EXE)
	@echo Make complete.
$(EXE): $(OBJ)
	$(CC) $(CFLAGS) $(OBJ) -o $(EXE) $(LIBS)
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
%.o: $.cpp $.hpp
	$(CC) $(CFLAGS) -c $^
clean:
	rm $(EXE) $(OBJ)
```

```cpp
#include <iostream>
#include "RingBuffer.hpp"

using namespace std;

//main and tests functions
int main(){
  cout << "test" << endl;

  RingBuffer check(10);

  check.enqueue(1);
  check.enqueue(2);
  check.enqueue(3);
  check.dequeue();
  cout << check.peek() << endl;
  check.printBuffer();

}
```

```
#ifndef RINGBUFFER_HPP
#define RINGBUFFER_HPP
#include <stdint.h>
#include <iostream>

class RingBuffer {
 public:
    RingBuffer(int capacity);
    int size();  // return number of items currently in the buffer
    bool isEmpty();  // is the buffer empty (size equals zero)?
    bool isFull();  // is the buffer full  (size equals capacity)?
    void enqueue(int16_t x);  // add item x to the end
    int16_t dequeue();  // delete and return item from the front
    int16_t peek();  // return (but do not delete) item from the front
 private:
    int s;
    int cap;
    int16_t top;
    int16_t *Buffer;
};
#endif
```

```cpp
#include "RingBuffer.hpp"
#include <stdint.h>
#include <iostream>
#include <exception>
#include <stdexcept>

using namespace std;

RingBuffer::RingBuffer(int capacity) {
  if (capacity < 1){
    throw invalid_argument("Cannot Instantiate with capacity < 1)\n");
  }
  s = 0, top = 0, cap = capacity;
  Buffer = new int16_t[cap];
}

int RingBuffer::size() {
  return s;
}

bool RingBuffer::isEmpty() {
  return s == 0;
}

bool RingBuffer::isFull() {
  return s == cap;
}
void RingBuffer::enqueue(int16_t x) {
  if (isFull()){
    throw runtime_error("RingBuffer Full\n");
  }

  Buffer[(top + s) % cap] = x;
  s++;
}

int16_t RingBuffer::dequeue() {
  if (isEmpty()){
    throw runtime_error("RingBuffer Empty\n");
  }

  s--;
  int dequeued = top;
  top = (top + 1) % cap;

  return Buffer[dequeued];
}

int16_t RingBuffer::peek() {
  if (isEmpty()){
    throw runtime_error("RingBuffer Empty\n");
  }

  return Buffer[top];
}
```

```
#include "RingBuffer.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(RingBufferConstructor) {
  BOOST_REQUIRE_NO_THROW(RingBuffer(100));

  // requires a thrown exception
  BOOST_REQUIRE_THROW(RingBuffer(0), std::exception);
  BOOST_REQUIRE_THROW(RingBuffer(0), std::invalid_argument);
  BOOST_REQUIRE_THROW(RingBuffer(-1), std::invalid_argument);
}

//test size
BOOST_AUTO_TEST_CASE(size) {
  RingBuffer test(10);

  BOOST_REQUIRE(test.size() == 0);
  //add 1 to the buffer and check if buffer total equals 1
  test.enqueue(1);
  BOOST_REQUIRE(test.size() == 1);
}

//test isEmpty
BOOST_AUTO_TEST_CASE(isEmpty) {
  RingBuffer test(1);
  BOOST_REQUIRE(test.isEmpty()==1);
  test.enqueue(1);
  BOOST_REQUIRE(test.isFull()==1);
}

//test isFull
BOOST_AUTO_TEST_CASE(isFull) {
  RingBuffer test(1);
  test.enqueue(1);
  BOOST_REQUIRE(test.isFull()==1);
}

//test enqueue
BOOST_AUTO_TEST_CASE(enqueue) {
  RingBuffer test(3);
  BOOST_REQUIRE_NO_THROW(test.enqueue(0));
  BOOST_REQUIRE_NO_THROW(test.enqueue(3));
  BOOST_REQUIRE_NO_THROW(test.enqueue(6));
}

//test dequeue
BOOST_AUTO_TEST_CASE(dequeue) {
  RingBuffer test(3);
  test.enqueue(0);
  test.enqueue(2);
  test.enqueue(4);
  BOOST_REQUIRE_NO_THROW(test.dequeue());
  BOOST_REQUIRE_NO_THROW(test.dequeue());
  BOOST_REQUIRE_NO_THROW(test.dequeue());
}

//test peek
BOOST_AUTO_TEST_CASE(peek) {
  RingBuffer test(3);
  test.enqueue(0);
  test.enqueue(2);
  test.enqueue(4);
  BOOST_REQUIRE_NO_THROW(test.peek());
  BOOST_REQUIRE_NO_THROW(test.peek());
  BOOST_REQUIRE_NO_THROW(test.peek());

}
```

PS4b: Synthesizing a Plucked String Sound

The Assignment:

This assignment builds off the previous part of PS4 and utilizes the RingBuffer class as a base. We must extend the program to simulate the sounds of a guitar using the Karplus-Strong algorithm. To do this we create a new class to emulate the sound samples and model designated frequencies. By completing this program, we will be able to sound designated guitar notes via keyboard input.

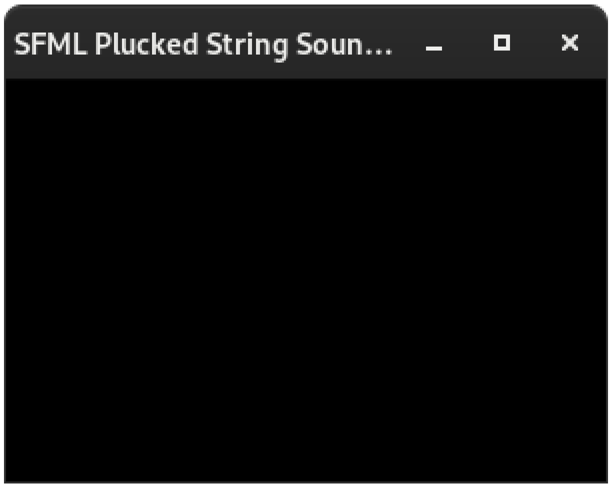
Central Designs:

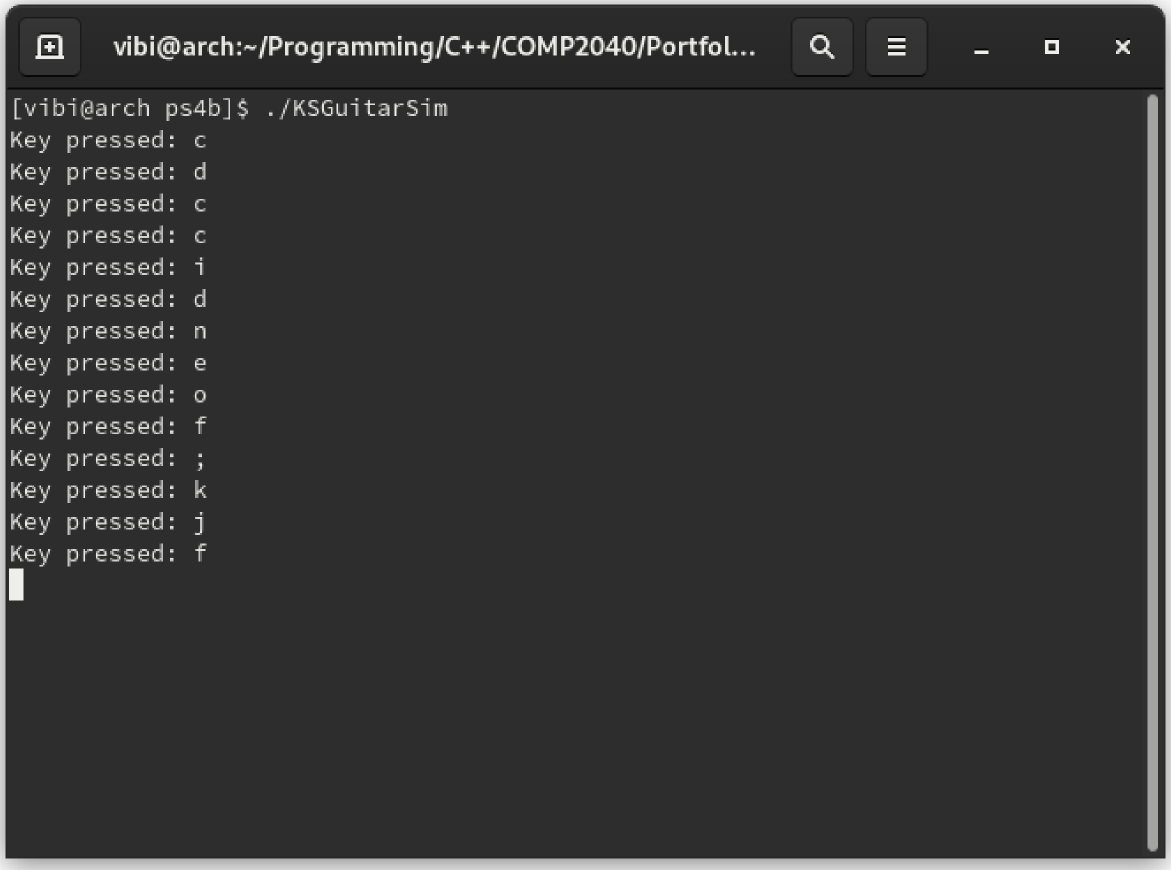
Key concepts of this program revolve around the Karplus-Strong algorithm. In order to simulate the sound of a guitar we average 2 values and multiply it by an energy decay factor. By using the Karplus-Strong algorithm alongside the RingBuffer we previously developed we can emulate the sounds of a guitar.

What I learned:

This program taught me the theory behind Karplus-Strong's algorithm and taught me how to implement theories into code. I also got accustom to SFML's audio capabilities through its documentation as well as SFML's keyboard input capabilities which I used alongside a keyboard parser to link sounds on the guitar to keys on the keyboard in the correct order.

Output:





```
[vibi@arch ps4b]$ ./KSGuitarSim
Key pressed: c
Key pressed: d
Key pressed: c
Key pressed: c
Key pressed: i
Key pressed: d
Key pressed: n
Key pressed: e
Key pressed: o
Key pressed: f
Key pressed: ;
Key pressed: k
Key pressed: j
Key pressed: f
```

```
CC=g++
CFLAGS= -std=c++14 -Wall -Werror -pedantic
OBJ=KSGuitarSim.o RingBuffer.o StringSound.o
LIBS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
DEPS=
EXE=KSGuitarSim

all: $(EXE)
	@echo Make complete.
$(EXE): $(OBJ)
	$(CC) $(CFLAGS) $(OBJ) -o $(EXE) $(LIBS)
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
%.o: $.cpp $.hpp
	$(CC) $(CFLAGS) -c $^
clean:
	rm $(EXE) $(OBJ)
```

```
/*
Vibishan Wigneswaran
PS4b
  Copyright 2015 Fred Martin,
  Y. Rykalova, 2020
*/
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

#include <math.h>
#include <limits.h>

#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>

#include "RingBuffer.hpp"
#include "StringSound.hpp"

#define CONCERT_A 220.0
#define SAMPLES_PER_SEC 44100
const int keyInput = 37;

using namespace std;

vector<sf::Int16> makeSample(StringSound gs) {
  vector<sf::Int16> samples;

  gs.pluck();
  int duration = 8;
  int i;
  for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
    gs.tic();
    samples.push_back(gs.sample());
  }

  return samples;
}

int main() {
  sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Plucked String Sound Lite
");
  sf::Event event;

  // define freq and a vector for sound samples
  double freq;
  vector<sf::Int16> sample;
  vector<vector<sf::Int16>> samples(keyInput);
  vector<sf::SoundBuffer> buffers(keyInput);
  vector<sf::Sound> sounds(keyInput);

  // Keys used for sound
  string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";

  // Loop transitions through the keyboard for user input
  for (int i = 0; i < (signed)keyboard.size(); i++) {
    freq = CONCERT_A * pow(2, ( (i - 24)/12.0));
    StringSound tmp = StringSound(freq);
    sample = makeSample(tmp);
    samples[i] = sample;

    if (!buffers[i].loadFromSamples(&samples[i][0],
        samples[i].size(), 2, SAMPLES_PER_SEC)) {
        throw runtime_error("Failed to load Samples");
    }
    sounds[i].setBuffer(buffers[i]);
  }

  while (window.isOpen()) {
    while (window.pollEvent(event)) {
        if (event.type == sf::Event::TextEntered) {
          if (event.text.unicode < 128) {
            char key = static_cast<char>(event.text.unicode);
            for (int i = 0; i < (signed)keyboard.size(); i++) {
              if (keyboard[i] == key) {
                cout << "Key pressed: " << keyboard[i] << "\n";
                sounds[i].play();
                break;
              }
            }
          }
        }
      }
    window.clear();
    window.display();
  }
  return 0;
}
```

```
#ifndef StringSound_HPP
#define StringSound_HPP

#include <stdint.h>
#include <vector>
#include <exception>
#include "RingBuffer.hpp"

using namespace std;

class StringSound{
public:
  StringSound(double frequency);
  StringSound(vector<int16_t> init);
  void pluck();
  void tic();
  int16_t sample();
  int time();
private:
  RingBuffer* pBuffer;
  int step;
};
#endif
```

```cpp
#include "StringSound.hpp"
#include <vector>
#include <cmath>


StringSound::StringSound(double frequency){
  int x = ceil(44100 / frequency);
  pBuffer = new RingBuffer(x);
  while(!pBuffer->isFull()){
    pBuffer->enqueue(0);
  }
  step = 0;
}

StringSound::StringSound(vector<int16_t> init){
  pBuffer = new RingBuffer(init.size());
  int i = 0;
  while(!pBuffer->isFull()){
    pBuffer->enqueue(init[i]);
    i++;
  }
}

void StringSound::pluck(){
  if(pBuffer->isFull()){
    for(int i = 0; i < pBuffer->size(); i++){
      pBuffer->dequeue();
    }
  }
  while(!pBuffer->isFull()){
    pBuffer->enqueue((int16_t)(rand()*0xffff));
  }
}

void StringSound::tic(){
  int16_t x = .5 * .996 * (pBuffer->dequeue() + pBuffer->peek());
  pBuffer->enqueue(x);

  ++step;
}

int16_t StringSound::sample(){
  int16_t Sample = pBuffer->peek();
  return Sample;
}

int StringSound::time(){
  return step;
}
```

```
#ifndef RINGBUFFER_HPP
#define RINGBUFFER_HPP
#include <stdint.h>
#include <iostream>

class RingBuffer {
 public:
    RingBuffer(int capacity);
    int size();  // return number of items currently in the buffer
    bool isEmpty();  // is the buffer empty (size equals zero)?
    bool isFull();  // is the buffer full  (size equals capacity)?
    void enqueue(int16_t x);  // add item x to the end
    int16_t dequeue();  // delete and return item from the front
    int16_t peek();  // return (but do not delete) item from the front
 private:
    int s;
    int cap;
    int16_t top;
    int16_t *Buffer;
};
#endif
```

```
#include "RingBuffer.hpp"
#include <stdint.h>
#include <iostream>
#include <exception>
#include <stdexcept>

using namespace std;

RingBuffer::RingBuffer(int capacity) {
  if (capacity < 1){
    throw invalid_argument("Cannot Instantiate with capacity < 1)\n");
  }
  s = 0, top = 0, cap = capacity;
  Buffer = new int16_t[cap];
}

int RingBuffer::size() {
  return s;
}

bool RingBuffer::isEmpty() {
  return s == 0;
}

bool RingBuffer::isFull() {
  return s == cap;
}
void RingBuffer::enqueue(int16_t x) {
  if (isFull()){
    throw runtime_error("RingBuffer Full\n");
  }

  Buffer[(top + s) % cap] = x;
  s++;
}

int16_t RingBuffer::dequeue() {
  if (isEmpty()){
    throw runtime_error("RingBuffer Empty\n");
  }

  s--;
  int dequeued = top;
  top = (top + 1) % cap;

  return Buffer[dequeued];
}

int16_t RingBuffer::peek() {
  if (isEmpty()){
    throw runtime_error("RingBuffer Empty\n");
  }

  return Buffer[top];
}
```

PS5: DNA Sequence Alignment

The Assignment:

The goal of this assignment is to write a program to compute optimal sequence alignment of 2 DNA string. We must also utilize Valgrind a memory analysis tool to check for memory leaks and utilize dynamic programming. We must be sure to output edit distances as well as the matrix and execution time.

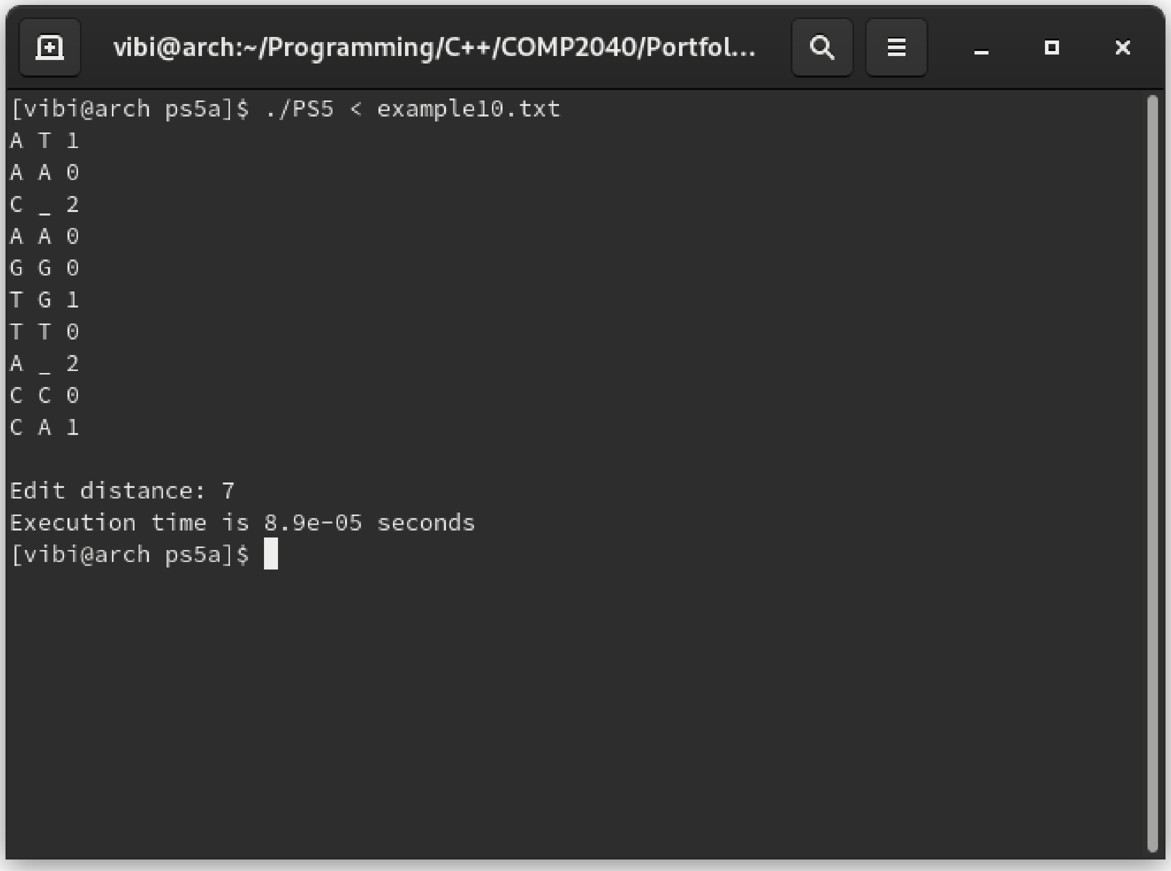
Central Designs:

This program revolves around the concept of Needleman-Wunsch, a form of dynamic programming, in which we calculate subproblems and use the subproblems to solve the original problem. We manage to implement Needleman-Wunsch by traversing a N x M matrix.

What I learned:

This assignment taught me how to implement dynamic programming and the importance of finding simple solutions for complex problems. Although this assignment is quite complex there are many different options to simplify the coding process to make code run with no memory leaks and low complexity. I also learned to use valgrind to test for memory leaks.

Output:



vibi@arch:~/Programming/C++/COMP2040/Portfol...

```
[vibi@arch ps5a]$ ./PS5 < example10.txt
A T 1
A A 0
C _ 2
A A 0
G G 0
T G 1
T T 0
A _ 2
C C 0
C A 1

Edit distance: 7
Execution time is 8.9e-05 seconds
[vibi@arch ps5a]$ 
```

```
#MakeFile Vibishan Wigneswaran PS5
CC= g++
CFLAGS= -g -Wall -Werror -std=c++0x -pedantic
SFLAGS= -lsfml-system

all: PS5

PS5: ps5.o main.o
	$(CC) ps5.o main.o -o PS5 $(SFLAGS)

ps5.o: ps5.cpp ps5.hpp
	$(CC) -c ps5.cpp ps5.hpp $(CFLAGS)

main.o: main.cpp ps5.hpp
	$(CC) -c main.cpp ps5.hpp $(CFLAGS)

clean:
	rm *.o
	rm PS5
	rm .gch
```

```
#include <iostream>
#include <sstream>
#include <SFML/System.hpp>

#include "ps5.hpp"

using namespace std;

int main(){
    //SFML clock
    sf::Clock clock;
    sf::Time t;

    string Xstr, Ystr;
    cin >> Xstr;
    cin >> Ystr;

    //instantiate new ED
        ED newED(Xstr, Ystr);

    //output in terminal
    cout << newED.getES() << endl;
    cout << "Edit distance: " << newED.getED() << endl;
        t = clock.getElapsedTime();
        cout << "Execution time is " << t.asSeconds() << " seconds \n";

    return 0;
}
```

```
#ifndef PS5_HPP
#define PS5_HPP

#include <iostream>
#include <vector>

using namespace std;

class ED {
public:
  ED(string n, string m);
  int penalty(char a, char b);
  int min(int a, int b, int c);
  int OptDistance();
  string Alignment();
  void print();
  int getED() const { return editDistance;}
  string getES() const{ return editStr;}

private:
  vector< vector< int > > opt;  //private vector holds opt data
  string N; //X axis string
  string M; //Y axis string
  string editStr; //edit string
  int editDistance; //edit distance
};

#endif
```

```
//Vibishan Wigneswaran
//Rykalova
//PS5
#include <iostream>
#include <vector>
#include <sstream>

#include "ps5.hpp"

using namespace std;

//constructor
ED::ED(string n, string m):N(n),M(m){
      vector<int> newVect;

      for(int i = 0; i < static_cast<int>(M.length()) + 1; i++){
          newVect.push_back(0);
        }

      for(int i = 0; i < static_cast<int>(N.length()) + 1; i++){
          opt.push_back(newVect);
        }

         editDistance = OptDistance();
         editStr = Alignment();
}

//penalties for same char or mismatch char
int ED::penalty(char a, char b){
  if(a==b){
    return 0;
  }
  else{
    return 1;
  }
}

//find min between 3 int values
int ED::min(int a, int b, int c){
  int min;
  if(a<b){
    min = a;
    if(min>c){
      min = c;
      return min;
    } else{
      return min;
    }
  }
  else{
    min = b;
    if(min>c){
      min = c;
      return min;
    } else{
      return min;
    }
  }
}

//finds optdistance and returns
int ED::OptDistance(){
  for(int i = opt.size() - 1; i >= 0; i--)
        {
                for(int j = opt[i].size() - 1; j >= 0; j--)
                {
                        if((i == static_cast<int>(opt.size() - 1)) && (j == static
_cast<int>(opt[i].size() - 1))){
                                opt[i][j] = 0;
      }
```

```cpp
                        else if(i == static_cast<int>(opt.size() - 1)){
                                opt[i][j] = opt[i][j + 1] + 2;
      }
                        else if (j == static_cast<int>(opt[i].size() - 1)){
                                opt[i][j] = opt[i + 1][j] + 2;
      }
                        else{
                                opt[i][j] = min(static_cast<int>(opt[i+1][j+1] + penalty(N[i], M[j])), static_cast<int>(opt[i + 1][j] + 2), static_cast<int>(opt[i][j + 1] + 2));
      }
                }
        }
        return opt[0][0];
}

//Alighnment function
string ED::Alignment(){
  int i = 0;
  int j = 0;

        stringstream sstream;

        while(i < static_cast<int>(opt.size() - 1) || j < static_cast<int>(opt[0].size() - 1))
        {
                if((i < static_cast<int>(opt.size() - 1)) && (j < static_cast<int>(opt[0].size() - 1)) && (opt[i+1][j+1] <= opt[i+1][j] + 1) && (opt[i+1][j+1] <= opt[i][j+1] + 1))
                {
                        sstream << N[i] << " " << M[j] << " " << opt[i][j] - opt[i+1][j+1] << '\n';
                        i++;
                        j++;
                }
                else if(((i < static_cast<int>(opt.size() - 1)) && (opt[i+1][j] <= opt[i][j+1])) || (j == static_cast<int>(opt[0].size() - 1)))
                {
                        sstream << N[i] << " " << "_" << " " << opt[i][j] - opt[i+1][j] << '\n';
                        i++;
                }
                else
                {
                        sstream << "_" << " " << M[j] << " " << opt[i][j] - opt[i][j+1] << '\n';
                        j++;
                }
        }

        return sstream.str();
}

//display matrix
void ED::print()
{
    for(unsigned i = 0; i < opt.size(); i ++)
    {
        for(unsigned j = 0; j < opt[i].size(); j++)
        {
            cout << " " << opt[i][j];
        }
              cout << endl;
    }
}
```

PS6: Markov Model of Natural Language

The Assignment:

The goal of this assignment is to analyze a string for transitions between k-grams and the following letter. We produce a probabilistic model of the text given a number of k-grams and generate a reasonable string using the calculated ratios.

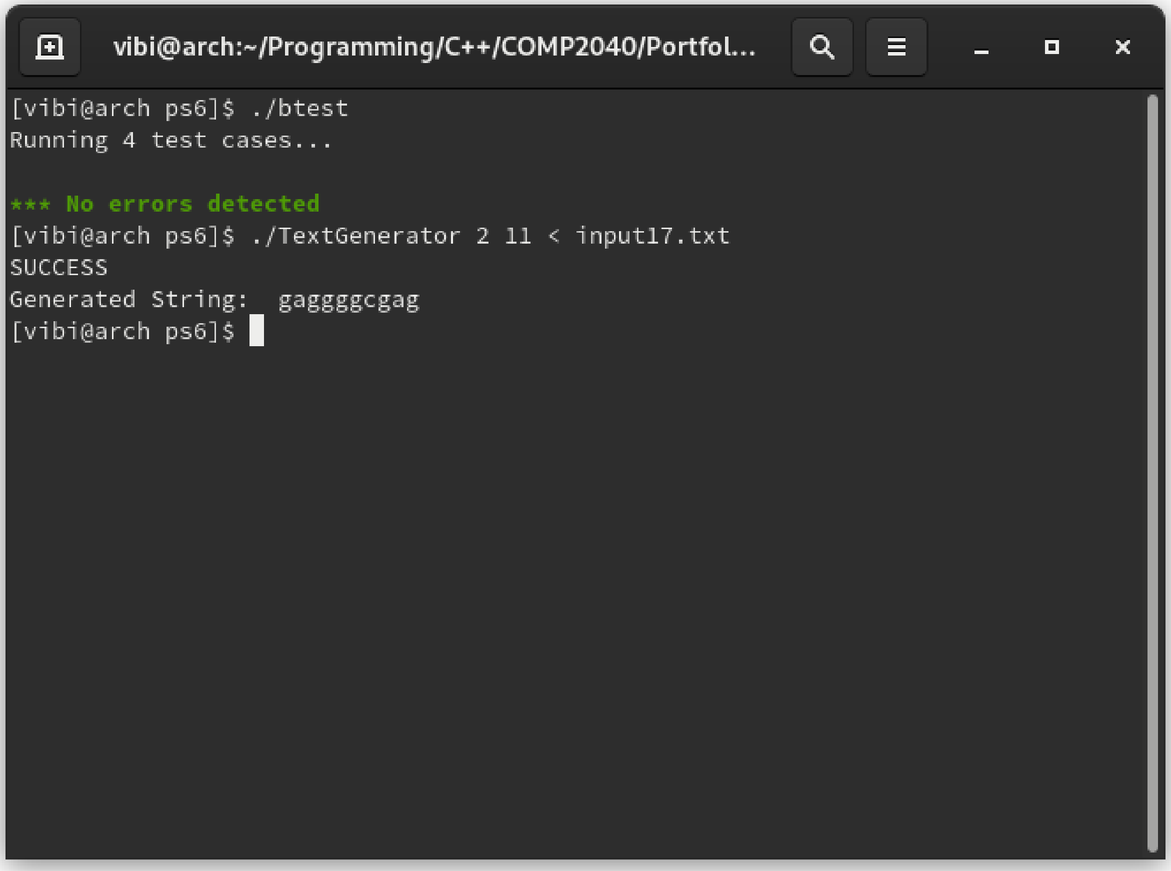
Central Designs:

Key concepts for this program include a deep understanding of Markov chains which utilize probabilistic models to produce reasonable strings of text. By using a generation function and random character function as well a frequency function we can create and utilize statistical likelihoods for character in a string which we then output.

What I learned:

Through this assignment I learned the concept of Markov chains and how to create statistical data from input data. I also learned to create test cases for these models and further increased my understanding of exception handling to catch errors. This program although simple can easily be advanced as such implementation is the basis of functions like google's auto complete addon.

Output:



vibi@arch:~/Programming/C++/COMP2040/Portfol...

```
[vibi@arch ps6]$ ./btest
Running 4 test cases...

*** No errors detected
[vibi@arch ps6]$ ./TextGenerator 2 11 < input17.txt
SUCCESS
Generated String:  gaggggcgag
[vibi@arch ps6]$ 
```

```
#Vibishan Wigneswaran MAKEFILE
Boost = -lboost_unit_test_framework
all:
	make TextGenerator
	make btest

TextGenerator: MModel.o TextGenerator.o
	g++ -o TextGenerator TextGenerator.o MModel.o -ansi -pedantic -Wall -Werror

btest: test.o MModel.o
	g++ test.o MModel.o -o btest $(Boost)

TextGenerator.o: MModel.h TextGenerator.cpp
	g++ -c TextGenerator.cpp -ansi -pedantic -Wall -Werror

MModel.o: MModel.h MModel.cpp
	g++ -c MModel.cpp -ansi -pedantic -Wall -Werror

test.o: test.cpp
	g++ -c test.cpp $(Boost)

clean:
	rm -f *.o *~ TextGenerator *~btest
```

```cpp
#include <iostream>
#include <string>
#include <sstream>
#include <cstdlib>
#include "MModel.h"

using namespace std;

int main(int argc, char* argv[]) {
  if (argc < 3) {
    cout << "Not enough arguements" << endl;
    return 1;
  }
  else if (argc > 3) {
    cout << "Too many arguements" << endl;
    return 1;
  }
  int k = atoi(argv[1]);
  int L = atoi(argv[2]);

  string input;
  string cursor;

  while (cin >> cursor) {
    input += " " + cursor;
    cursor = "";
  }
  MModel model(input, k);
  string str;

  for (int i = 0; i < k; i++){
    str.push_back(input[i]);
  }
  cout << "SUCCESS" << endl;
  cout << "Generated String: " << model.generate(str, L) << endl;

  return 0;
}
```

```
#ifndef MModel_H
#define MModel_H

#include <iostream>
#include <string>
#include <map>

using namespace std;

class MModel{
public:
  MModel(string text, int k);

  int kOrder() const;
  int freq(string kgram);
  int freq(string kgram, char c);
  char kRand(string kgram);
  string generate(string kgram, int L);

  friend ostream& operator<<(ostream& in, MModel& m);

private:
  string alphabet;
  string init;
  int order;

  map <string,int> kgrams;

};
#endif
```

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <stdexcept>
#include <cstdlib>
#include <ctime>
#include "MModel.h"

using namespace std;

MModel::MModel(string text, int k){
  order = k;
  init = text;
  srand(time(NULL));

  for (unsigned i = 0; i < text.size(); i++)
    if (string::npos == alphabet.find(text[i]))
      alphabet.push_back(text[i]);

  for (unsigned i = 0; i < text.size(); i++) {

    string newStr;
    string newStr2;

    //create new string
    for (unsigned j = i; j < i + k; j++)
      if (j >= text.size())
        newStr.push_back(text[j - text.size()]);
      else
        newStr.push_back(text[j]);

    // string repeat in new string
    if (kgrams.end() == kgrams.find(newStr))
      kgrams[newStr] = 1;
    else
      kgrams[newStr] += 1;

    // all possible k+1 strings
    for (unsigned j = 0; j < alphabet.size(); j++)
      if (kgrams.end() == kgrams.find(newStr + alphabet[j]))
        kgrams[newStr + alphabet[j]] = 0;

    for (unsigned j = i; j < i + k + 1; j++)
      if (j >= text.size())
        newStr2.push_back(text[j - text.size()]);
      else
        newStr2.push_back(text[j]);

    kgrams[newStr2] += 1;
  }
}

int MModel::kOrder() const{
  return order;
}

int MModel::freq(string kgram){
  if (kgram.size() != (unsigned)order)
    throw runtime_error("kgram not equal to k");

  if (order == 0)
    return init.size();
  else
    return kgrams[kgram];
}

int MModel::freq(string kgram, char c){
  if (kgram.size() != (unsigned)order)
    throw runtime_error("kgram not equal to k");

  if (order == 0) {
    int count = 0;
    for (unsigned i = 0; i < init.size(); i++)
      if (init[i] == c)
        count++;
    return count;
  } else {
    return kgrams[kgram + c];
  }

  return 0;
}

char MModel::kRand(string kgram){
  if (kgram.size() != (unsigned)order || kgrams.end() == kgrams.find(kgram))
    throw runtime_error("kRand: kgram not valid");

  string newStr;
  for (unsigned i = 0; i < alphabet.size(); i++)
    for (int j = 0; j < kgrams[kgram + alphabet[i]]; j++)
      newStr.push_back(alphabet[i]);

  return newStr[rand() % newStr.size()];
}

string MModel::generate(string kgram, int L){
  string newStr = kgram;
  string ret = kgram;
  char rc;

  for (int i = 0; i < L - order; i++) {
    rc = kRand(newStr);
    ret.push_back(rc);
    newStr.erase(newStr.begin());
    newStr.push_back(rc);
  }

  return ret;
}

ostream& operator<<(ostream& in, MModel& m){
  in << "\n" << "String: \"" << m.init << "\"" << endl;
  in << "Order:" << m.order << endl;
  in << "Alphabet: \"" << m.alphabet << "\"" << "\n" << endl;

  in << "Markov Map" << endl;
  map <string, int> newStr = m.kgrams;
  for (map<string, int>::iterator it = newStr.begin();
       it != newStr.end(); ++it) {
    in << it->first << " " << it->second << " => ";
    for (unsigned i = 0; i < m.alphabet.size(); i++) {
      it++;
      in << it->first << " " << it->second << " ";
    }

    in << endl;
  }

  return in;
}
```

```
//Vibishan Wigneswaran Boost test
#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>

#include "MModel.h"

using namespace std;

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(constructorTest) {
  // normal constructor
  BOOST_REQUIRE_NO_THROW(MModel("abababab aa", 0));

  MModel test("abab", 0);

  //BOOST_REQUIRE(test.order() == 0);
  BOOST_REQUIRE(test.freq("") == 4);

  BOOST_REQUIRE_THROW(test.freq("y"), std::runtime_error);

  BOOST_REQUIRE(test.freq("", 'a') == 2);
  BOOST_REQUIRE(test.freq("", 'b') == 2);
  BOOST_REQUIRE(test.freq("", 'c') == 0);
}

BOOST_AUTO_TEST_CASE(test2) {
  // normal constructor
  BOOST_REQUIRE_NO_THROW(MModel("qaqqqaqaqqcqaqaaa", 1));

  MModel newTest("qaqqqaqaqqcqaqaaa", 1);

  //BOOST_REQUIRE(newTest.order() == 1);
  BOOST_REQUIRE_THROW(newTest.freq(""), std::runtime_error);
  BOOST_REQUIRE_THROW(newTest.freq("xx"), std::runtime_error);

  BOOST_REQUIRE(newTest.freq("a") == 7);
  BOOST_REQUIRE(newTest.freq("q") == 9);
  BOOST_REQUIRE(newTest.freq("c") == 1);

  BOOST_REQUIRE(newTest.freq("a", 'a') == 2);
  BOOST_REQUIRE(newTest.freq("a", 'c') == 0);
  BOOST_REQUIRE(newTest.freq("a", 'q') == 5);

  BOOST_REQUIRE(newTest.freq("c", 'a') == 0);
  BOOST_REQUIRE(newTest.freq("c", 'c') == 0);
  BOOST_REQUIRE(newTest.freq("c", 'q') == 1);

  BOOST_REQUIRE(newTest.freq("q", 'a') == 5);
  BOOST_REQUIRE(newTest.freq("q", 'c') == 1);
  BOOST_REQUIRE(newTest.freq("q", 'q') == 3);

  BOOST_REQUIRE_NO_THROW(newTest.kRand("a"));
  BOOST_REQUIRE_NO_THROW(newTest.kRand("c"));
  BOOST_REQUIRE_NO_THROW(newTest.kRand("q"));

  BOOST_REQUIRE_THROW(newTest.kRand("x"), std::runtime_error);

  BOOST_REQUIRE_THROW(newTest.kRand("xx"), std::runtime_error);

}

BOOST_AUTO_TEST_CASE(test3) {
  // normal constructor
  BOOST_REQUIRE_NO_THROW(MModel("lalllalallclalaaa", 2));

  MModel newTest("lalllalallclalaaa", 2);

  BOOST_REQUIRE_THROW(newTest.freq(""), std::runtime_error);
  BOOST_REQUIRE_NO_THROW(newTest.freq("xx"));
  BOOST_REQUIRE_THROW(newTest.freq("", 'l'), std::runtime_error);
  BOOST_REQUIRE_THROW(newTest.freq("x", 'l'), std::runtime_error);
  BOOST_REQUIRE_THROW(newTest.freq("xxx", 'l'), std::runtime_error);


  BOOST_REQUIRE(newTest.freq("aa") == 2);
  BOOST_REQUIRE(newTest.freq("aa", 'a') == 1);
  BOOST_REQUIRE(newTest.freq("aa", 'c') == 0);
  BOOST_REQUIRE(newTest.freq("aa", 'l') == 1);

  BOOST_REQUIRE(newTest.freq("al") == 5);
  BOOST_REQUIRE(newTest.freq("al", 'a') == 3);
  BOOST_REQUIRE(newTest.freq("al", 'c') == 0);
  BOOST_REQUIRE(newTest.freq("al", 'l') == 2);

  BOOST_REQUIRE(newTest.freq("cl") == 1);
  BOOST_REQUIRE(newTest.freq("cl", 'a') == 1);
  BOOST_REQUIRE(newTest.freq("cl", 'c') == 0);
  BOOST_REQUIRE(newTest.freq("cl", 'l') == 0);

  BOOST_REQUIRE(newTest.freq("la") == 5);
  BOOST_REQUIRE(newTest.freq("la", 'a') == 1);
  BOOST_REQUIRE(newTest.freq("la", 'c') == 0);
  BOOST_REQUIRE(newTest.freq("la", 'l') == 4);

}

BOOST_AUTO_TEST_CASE(test4) {
  MModel newTest("zyzyzyzz", 8);
  BOOST_REQUIRE_THROW(newTest.freq("x"), std::runtime_error);
  BOOST_REQUIRE_THROW(newTest.freq("", 'l'), std::runtime_error);
  BOOST_REQUIRE_THROW(newTest.freq("x", 'l'), std::runtime_error);
  BOOST_REQUIRE_THROW(newTest.freq("xxx", 'l'), std::runtime_error);
}
```