Anas Ahmed

COMP IV: Project Portfolio
Spring 2020

Contents:

PS0: Hello World with SFML

The goal of this assignment was to display a sprite, as well as adding the ability to move the sprite with the arrow keys. Additionally, I was instructed to add an extra feature for functionality. For my PS0, I ended up adding text mentioning the famous result of Super Bowl 51, where the New England Patriots made a comeback against the Atlanta Falcons when they were up 28-3.

The purpose of this assignment was to get familiar with the SFML library. I was able to learn how to load fonts, images, and sprites as well as learn how to move sprites according to keyboard inputs.

Makefile

```
1   CC = g++
2   CFLAGS = -c -g -Og _ Wall -Werror -ansi -pedantic
3   OBJ = main.o
4   SFML = -lsfml-graphics -lsfml-window -lsfml-system
5   EXE = ps0
6
7   all: $(OBJ)
8       $(CC) $(OBJ) -o $(EXE) $(SFML)
9
10  clean:
11      rm $(OBJ) $(EXE)
12
```

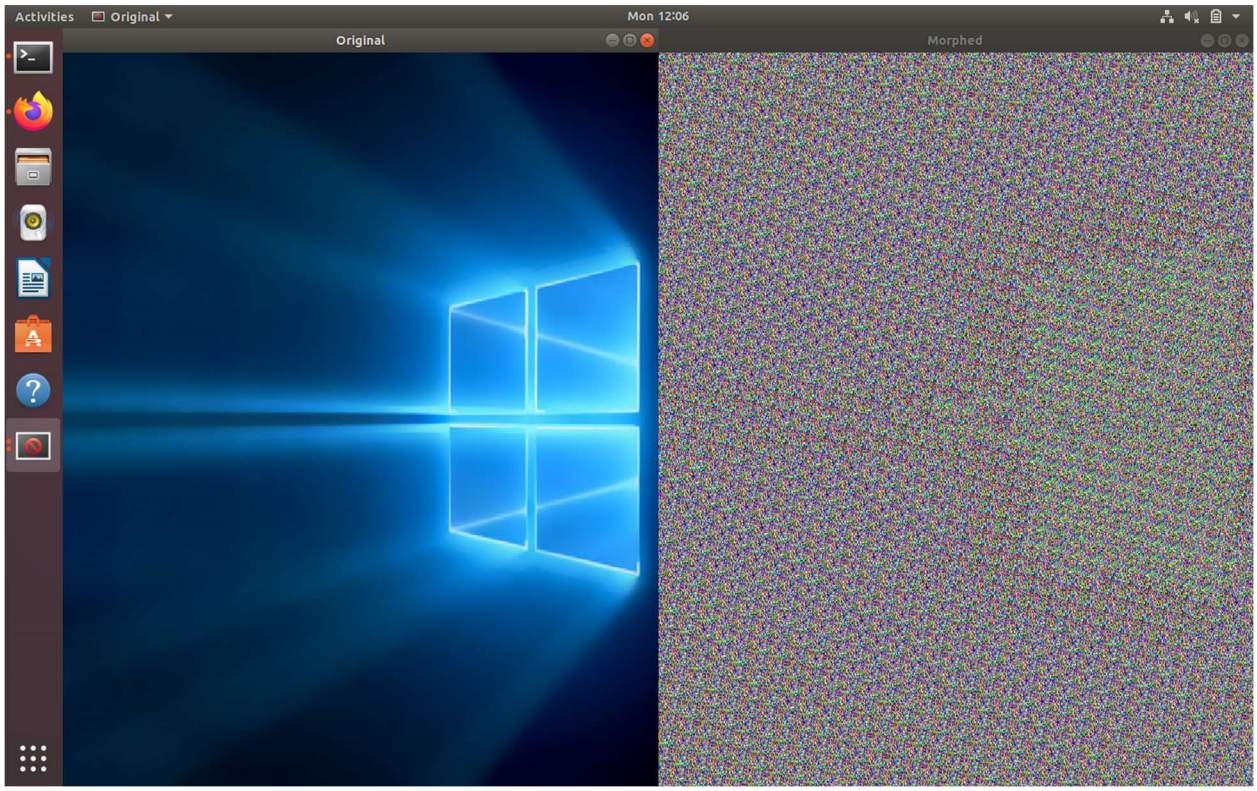
main.cpp

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

int main()
{
    sf::RenderWindow window(sf::VideoMode(1280, 720), "SFML works!");
    sf::Texture texture;

    window.setVerticalSyncEnabled(true);  //Enable V-sync

    if(!texture.loadFromFile("sprite.jpg"))  //Load Sprite
      return EXIT_FAILURE;
    sf::Sprite sprite(texture);

    sf::Font font;
    if(!font.loadFromFile("arial.ttf")) // Load Font
      return EXIT_FAILURE;
    sf::Text text("The Falcons blew a 28-3 lead", font, 50);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }
  // Search for inputs
  if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
    sprite.move(2, 0);
  else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
    sprite.move(-2, 0);
  else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
    sprite.move(0, -2);
  else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
    sprite.move(0, 2);

  // Display sprite and font
  window.clear();
  window.draw(sprite);
  window.draw(text);
  window.display();
    }

    return 0;
}
```

PS1: Linear Feedback Shift Register

The goal of the assignment was to apply a Fibonacci Linear Feedback Shift Register (FibLSFR). in order to encrypt and decrypt an image. That was done by feeding the input file name, output file name, and seed number in the command line when calling the program. Once started, the program will initialize a FibLSFR with the stated seed and proceed to run the generate(8) function for each pixel as part of the transform function. In doing so, each pixel is given a completely new value. Once the transform function is finished, two windows will appear one with the morphed image, another with the original. Executing the program again with the same seed on the encrypted image will transform it back into the original.

This assignment helped acquaint me with the image transformation functions within the SFML library. I also learned how to output an image to file in C++.



Makefile

```
CC = g++
CFLAGS = -c -g -Og -Wall -Werror -ansi -pedantic
SFML = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
ALLOBJ = fiblsfr.o PhotoMagic.o
EXE = ps1a PhotoMagic

all: PhotoMagic

PhotoMagic: PhotoMagic.o
    $(CC) -Wall PhotoMagic.o  -o PhotoMagic $(SFML)
ps1a: fiblsfr.o
    $(CC) fiblsfr.cpp -c fiblsfr.o -o ps1a
clean:
    \rm *.o *~ $(EXE) output-file.png

```

PhotoMagic.cpp (main)

```cpp
#include <iostream>
#include <string>
#include <cstdlib>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "fiblsfr.hpp"
using namespace std;

sf::Image transform(sf::Image image, FibLSFR *bitShift);

int main(int argc, char *argv[]){
  int x, y;
  FibLSFR colorCounter;
  sf::Image image1, image2;

  if(!image1.loadFromFile(argv[1]))
    return -1;
  if(!image2.loadFromFile(argv[1]))
    return -1;

  colorCounter.setSeed(argv[argc - 1]);

  image2 = transform(image2, &colorCounter);  // Transforms the image once they are loaded

  // Converts the images to sprites to prepare them for displaying

  sf::Texture texture1, texture2;
  texture1.loadFromImage(image1);
  texture2.loadFromImage(image2);

  sf::Sprite sprite1, sprite2;
  sprite1.setTexture(texture1);
  sprite2.setTexture(texture2);

  sf::RenderWindow window1(sf::VideoMode(800, 600), "Original"), window2(sf::VideoMode(800, 600), "Morphed");  // Opening the windows after the transform function is called helps a lot with optimization

  while(window1.isOpen() && window2.isOpen()){  // Runs draws both sprites in both windows
    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();
      }
    // Now loads the sprites
    window1.clear();
    window1.draw(sprite1);
    window1.display();

    window2.clear();
    window2.draw(sprite2);
    window2.display();
  }

  if(!image2.saveToFile(argv[2])) // Special case if the program is unable to save the encrypted image
    return -1;

  return 0;
}

sf::Image transform(sf::Image image, FibLSFR *bitShift){
      sf::Color p;
      unsigned int x, y;
      sf::Vector2u v1;
      v1 = image.getSize();    // Gets the size of the image and uses it for the bounds of the for loops


    for(x = 0; x < v1.x; x++){
        for(y = 0; y < v1.y; y++){
        p = image.getPixel(x, y);
        // The sum of the generate function will be applied for each color of pixel p
        p.r = p.r ^ bitShift->generate(8);
        p.g = p.g ^ bitShift->generate(8);
        p.b = p.b ^ bitShift->generate(8);
        image.setPixel(x, y, p);    // Sets the pixel
        }
    }
      return image;
}
```

fiblsfr.hpp

```
#include <iostream>
#include <string>
using namespace std;

class FibLSFR{
public:
  FibLSFR(string seed);

  FibLSFR();

  int setSeed(string newSeed);

  int step();

  int generate(int k);

  int bitstoint();

  friend ostream &operator <<(ostream& output, const FibLSFR see);

private:
  string LSFRseed;
};

FibLSFR::FibLSFR(string seed){
  LSFRseed = seed;
}

FibLSFR::FibLSFR(){
}

int FibLSFR::setSeed(string newSeed){
  LSFRseed = newSeed;
  return 0;
}

int FibLSFR::step(){
  char thirteen, twelve, ten, last;
  string newSeed = LSFRseed;

  thirteen = LSFRseed.at(2);
  twelve = LSFRseed.at(3);
  ten = LSFRseed.at(5);
  last = LSFRseed.front(); // Temporarily assigns last to the front bit before the XOR process begins

  // Checks fibonacci tap locations, if and else statements act as XOR function
  if(last == thirteen){last = '0';}
  else{last = '1';}
  if(last == twelve){last = '0';}
  else{last = '1';}
  if(last == ten){last = '0';}
  else{last = '1';}
  // last = char(front ^ thirteen ^ twelve ^ ten);

  // newSeed.at(0) = last;
  for(int i = 0; i < (LSFRseed.size() - 1); i++){
    // Special case, if at the end of the loop, back character is set
    if(i == (LSFRseed.size() - 2))
      newSeed.back() = last;
    newSeed.at(i) = LSFRseed.at(i + 1);
  }

  LSFRseed = newSeed;
  return int(last - 48); // last = 48 || 49, so must subtract it by 48 to get the return bit to 1 or 0
}
```

```
66
67  int FibLSFR::generate(int k){
68    int i, re;
69
70    // Runs the step function k amount of times
71    for(i = 0; i < k; i++){
72      this->step();
73    }
74    re = this->bitstoint();
75    cout << LSFRseed << " " << re << endl;
76    return re;
77  }
78
79  int FibLSFR::bitstoint(){
80    int sum = 0, twoExponent = 1;
81
82    // Loop adds the 2s power of each bit to the sum while increasing the exponent every time the loop increments
83    for(int i = 0; i < 8; i++){
84      sum += ((LSFRseed.at(LSFRseed.size() - 1 - i)) - 48) * twoExponent;
85      twoExponent *= 2;
86    }
87
88    return sum;
89  }
90  ostream &operator <<(ostream &output, const FibLSFR see){
91    output << see.LSFRseed;
92    return output;
93  }
```

PS2: Recursive Pythagoras Tree

The goal of this assignment was to recursively draw a Pythagoras Tree using the SFML library. A Pythagoras Tree is created by drawing a base square, and then drawing one additional square on each side. The process is then repeated until the tree is complete.

Originally, I used SFML's draw convex shape method, but I then switched to the draw rectangle method which ended up benefiting my code greatly, as it took less steps to draw each square.

I learned how to draw shapes using the SFML library and gained exposure to an advanced application of recursion. I normally prefer iterative programming, so it was beneficial to venture outside of my "comfort zone" in this project.

Makefile

```
1   CC = g++
2   CFLAGS = -c -g -Og _ Wall -Werror -ansi -pedantic
3   OBJ = main.o PTree.o
4   LIBS = -lsfml-graphics -lsfml-window -lsfml-system
5   EXE = PTree
6
7   all: $(OBJ)
8       $(CC) $(OBJ) -o $(EXE) $(LIBS)
9
10  clean:
11      rm $(OBJ) $(EXE)
12
```

main.cpp

```
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include "PTree.hpp"

using namespace std;
using namespace sf;

int main( int argc, char* argv[])
{
  double L = 0.0; // Length of square sides
  int N = 0; // Number of times to call recursive function

  L = atol(argv[1]);
  N = atoi(argv[2]);
  sf::Vector2f vPoint;
  vPoint.x = 0;
  vPoint.y = 0;

  // Create and Display Window
  PTree tree(L, N);
  return 0;
}
```

PTree.hpp

```
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <math.h>
#include <cmath>
#include "PTree.hpp"
#include <iostream>

using namespace std;
using namespace sf;

// Constructor
PTree::PTree(double L, int N)
{
  // Sets the window size by multiplying L
  width = (6 * L);
  height = (4 * L);
  /*
  Vector2f vPoint = {width/2, height - 1};
  Vector2f vOrigin;
  vOrigin.x = L/2;
  vOrigin.y = L;
  Code from my first attempt using ConvexShape*/

  window.create(VideoMode(width, height), "Pythagoras Fractal Tree");
  // Once window is created time to generate the tree
  while(window.isOpen())
  {
    for(sf::Event event; window.pollEvent(event);)
      {
        if(event.type == sf::Event::Closed)
          window.close();
      }
    window.clear(sf::Color::White);
    PTree::drawPtree(window, L, N);
    window.display();
  }
}

// Optimized destructor
PTree::~PTree(){}

/* Originally was going to use the ConvexShape functions to draw the tree, but using rectangles was much easier, I'm keeping the source code here

void PTree::pTree(double L, int N, Vector2f vPoint, Vector2f vOrigin, float rotation)
{
  Vector2f vPointR;

  // Ends recursion once tree is finished
  if(N < 1) return;

  // Define a convex shape called convexSquare
  ConvexShape convexSquare(4);
  convexSquare.setPoint(0, Vector2f(0, 0 ));
  convexSquare.setPoint(1, Vector2f(0, L));
  convexSquare.setPoint(2, Vector2f(L, L));
  convexSquare.setPoint(3, Vector2f(L, 0));

  convexSquare.setOutlineThickness(1.f);
  convexSquare.setFillColor(Color::Black);
  convexSquare.setOutlineColor(Color::White);

  convexSquare.setPosition(vPoint);
  convexSquare.setOrigin(vOrigin);
  convexSquare.setRotation(rotation);

  while(window.isOpen())
  {
    Event event;
    while(window.pollEvent(event))
    {
      if(event.type == Event::Closed)
      {
        window.close();
      }
    }
    if(N >= 0)
    {
    window.draw(convexSquare);
```

```
    window.display();
    L = (L * (sqrt(2)/2));
    N = N - 1;
    rotation = rotation - 135;

    // Generate left side
    vPoint = convexSquare.getTransform().transformPoint(convexSquare.getPoint(0));
    vOrigin = convexSquare.getPoint((angle1));
    pTree(L, N, vPoint, vOrigin, rotation);
    angle1 = ((angle1 + 1) % 4);

    // Generate right side
    vPointR = convexSquare.getTransform().transformPoint(convexSquare.getPoint(3));
    vOrigin = convexSquare.getPoint(2);
    pTree(L, N, vPointR, vOrigin, rotation-90);
    }
  }
}
*/
void PTree::drawPtree(sf::RenderTarget& target, const int N, const sf::RectangleShape& parent)
{
  static const float halfSqrt2 = sqrt(2.f) / 2;

  if(N < 1) return; // Ends the recursion
  target.draw(parent);
  auto const& sz = parent.getSize();
  auto const& tf = parent.getTransform();

  // Left side recursion call
  auto childL = parent;
  childL.setSize(sz * halfSqrt2);
  childL.setOrigin(0, childL.getSize().y);
  childL.setPosition(tf.transformPoint({0, 0}));
  childL.rotate(-45);
  PTree::drawPtree(target, N - 1, childL);

  // Right side recursion call
  auto childR = parent;
  childR.setSize(sz * halfSqrt2);
  childR.setOrigin(childR.getSize());
  childR.setPosition(tf.transformPoint({sz.x, 0}));
  childR.rotate(45);
  PTree::drawPtree(target, N - 1, childR);
}

void PTree::drawPtree(sf::RenderTarget& target, const float L, const int N)
{
  // Set origin to center of the rectangle
  sf::RectangleShape rect{{L, L}};

  rect.setOrigin(rect.getSize() / 2.f);
  rect.setPosition(target.getSize().x / 2.f, target.getSize().y - L / 2.f);
  rect.setFillColor(sf::Color::Blue); // EXTRA CREDIT: Add color
  PTree::drawPtree(target, N, rect);
}
```

PTree.cpp

```cpp
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <math.h>
#include <cmath>
#include "PTree.hpp"
#include <iostream>

using namespace std;
using namespace sf;

// Constructor
PTree::PTree(double L, int N)
{
  // Sets the window size by multiplying L
  width = (6 * L);
  height = (4 * L);
  /*
  Vector2f vPoint = {width/2, height - 1};
  Vector2f vOrigin;
  vOrigin.x = L/2;
  vOrigin.y = L;
  Code from my first attempt using ConvexShape*/

  window.create(VideoMode(width, height), "Pythagoras Fractal Tree");
  // Once window is created time to generate the tree
  while(window.isOpen())
  {
    for(sf::Event event; window.pollEvent(event);)
      {
        if(event.type == sf::Event::Closed)
          window.close();
      }
    window.clear(sf::Color::White);
    PTree::drawPtree(window, L, N);
    window.display();
  }
}

// Optimized destructor
PTree::~PTree(){}

/* Originally was going to use the ConvexShape functions to draw the tree, but using rectangles was much easier, I'm keeping the source code here

void PTree::pTree(double L, int N, Vector2f vPoint, Vector2f vOrigin, float rotation)
{
  Vector2f vPointR;

  // Ends recursion once tree is finished
  if(N < 1) return;

  // Define a convex shape called convexSquare
  ConvexShape convexSquare(4);
  convexSquare.setPoint(0, Vector2f(0, 0 ));
  convexSquare.setPoint(1, Vector2f(0, L));
  convexSquare.setPoint(2, Vector2f(L, L));
  convexSquare.setPoint(3, Vector2f(L, 0));

  convexSquare.setOutlineThickness(1.f);
  convexSquare.setFillColor(Color::Black);
  convexSquare.setOutlineColor(Color::White);

  convexSquare.setPosition(vPoint);
  convexSquare.setOrigin(vOrigin);
  convexSquare.setRotation(rotation);

  while(window.isOpen())
  {
    Event event;
    while(window.pollEvent(event))
    {
      if(event.type == Event::Closed)
      {
        window.close();
      }
    }
    if(N >= 0)
    {
    window.draw(convexSquare);
```

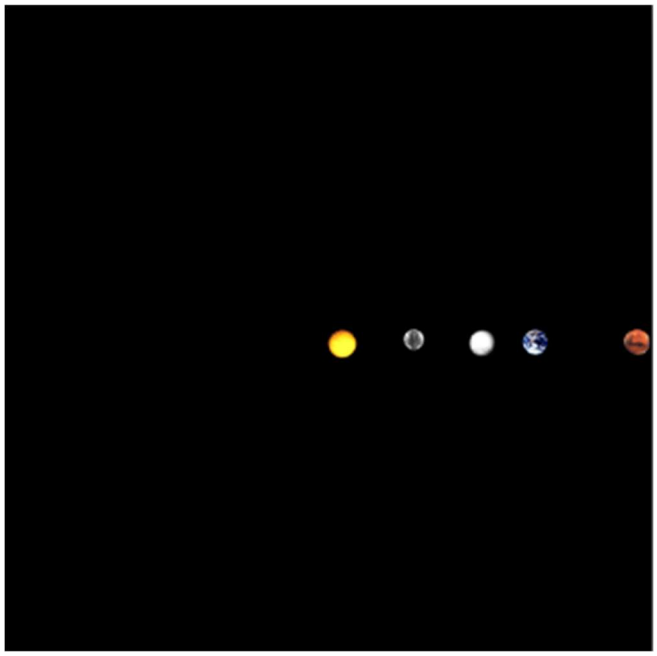
```
    window.display();
    L = (L * (sqrt(2)/2));
    N = N - 1;
    rotation = rotation - 135;

    // Generate left side
    vPoint = convexSquare.getTransform().transformPoint(convexSquare.getPoint(0));
    vOrigin = convexSquare.getPoint((angle1));
    pTree(L, N, vPoint, vOrigin, rotation);
    angle1 = ((angle1 + 1) % 4);

    // Generate right side
    vPointR = convexSquare.getTransform().transformPoint(convexSquare.getPoint(3));
    vOrigin = convexSquare.getPoint(2);
    pTree(L, N, vPointR, vOrigin, rotation-90);
    }
  }
 }
 */
void PTree::drawPtree(sf::RenderTarget& target, const int N, const sf::RectangleShape& parent)
{
  static const float halfSqrt2 = sqrt(2.f) / 2;

  if(N < 1) return; // Ends the recursion
  target.draw(parent);
  auto const& sz = parent.getSize();
  auto const& tf = parent.getTransform();

  // Left side recursion call
  auto childL = parent;
  childL.setSize(sz * halfSqrt2);
  childL.setOrigin(0, childL.getSize().y);
  childL.setPosition(tf.transformPoint({0, 0}));
  childL.rotate(-45);
  PTree::drawPtree(target, N - 1, childL);

  // Right side recursion call
  auto childR = parent;
  childR.setSize(sz * halfSqrt2);
  childR.setOrigin(childR.getSize());
  childR.setPosition(tf.transformPoint({sz.x, 0}));
  childR.rotate(45);
  PTree::drawPtree(target, N - 1, childR);
}

void PTree::drawPtree(sf::RenderTarget& target, const float L, const int N)
{
  // Set origin to center of the rectangle
  sf::RectangleShape rect{{L, L}};

  rect.setOrigin(rect.getSize() / 2.f);
  rect.setPosition(target.getSize().x / 2.f, target.getSize().y - L / 2.f);
  rect.setFillColor(sf::Color::Blue); // EXTRA CREDIT: Add color
  PTree::drawPtree(target, N, rect);
}
```

PS3: N-Body Simulation

The goal of this assignment was to render a galaxy using two classes, universe and cBody. cBody is a class which is made to represent any body rendered in the universe, whether it be the sun or a planet. Universe is a class which contains a vector of cBodies and has a function to draw the entire universe by accessing the vector. The universe is drawn by reading through a text file and assigning values from the file.

The second part of this assignment was to add physics simulation and animation. The concepts applied were Newton's law of universal gravitation and leapfrog finite difference approximation.

I used a smart pointer to represent the universe which helped avoid memory troubles.



Makefile

```
1   CC= g++
2   CFLAGS= -Wall -Werror -ansi -pedantic
3   GFLAGS= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
4   c11 = -std=c++11
5
6   all:
7       make NBody
8   NBody: body.o main.o
9       g++ body.o main.o -o NBody $(GFLAGS) $(c11)
10  body.o: body.cpp body.hpp
11      g++ -c body.cpp -o body.o $(CFLAGS) $(c11)
12  main.o: main.cpp body.hpp
13      g++ -c main.cpp -o main.o $(CFLAGS) $(c11)
14  clean:
15      rm *.o *NBody *~
16
```

main.cpp

```cpp
#include "body.hpp"
#include <string>
#include <iostream>
#include <memory>
#include <iterator>

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

 if(argc != 3){
  cout << "Error! Please restart the program with the correct number of command line arguments" << std::endl;
  return 1;
 }

 int uParticles;
 double uRadius;
 double dispTime = std::stod(argv[2]);
 double maxTime = std::stod(argv[1]);

 sf::Image backgroundimage;
 backgroundimage.loadFromFile("starfield.jpg");

 // Loads audio file
 sf::SoundBuffer music;
 if(!music.loadFromFile("2001.wav"))
  std::cout << "Error music file not found!" << endl;

 // Loads font used for time counter
 sf::Font counterFont;
 if(!counterFont.loadFromFile("arial.ttf"))
  std::cout << "Error! No font file found!" << endl;

 sf::Texture background_texture;
 background_texture.loadFromImage(backgroundimage);

 sf::Sprite background_sprite;
 background_sprite.setTexture(background_texture);

 // Loads text used for time counter
 sf::Text counterText;
 counterText.setFont(counterFont);
 counterText.setCharacterSize(20);

 sf::RenderWindow window1(sf::VideoMode(500, 500), "Space");
 window1.setFramerateLimit(40);
 sf::Vector2u window1size = window1.getSize();

 std::cout << std::scientific;

 // Uses smart pointers for universe
 std::unique_ptr<universe>u(new universe());
 cBody* planet;

 std::cin >> uParticles >> uRadius;
 cout << "Input: " << uParticles << std::endl << " uRadius: " << uRadius << std::endl;

 // Loads clock used for time
 sf::Clock timeCounter;

 double ixpos, iypos, ixvel, iyvel, imass;
 std::string iplanet;
 for(int i = 0; i < uParticles;i++)
  {
    std::cin >> ixpos >> iypos >> ixvel >> iyvel >> imass >> iplanet;
    planet = new cBody(ixpos, iypos, ixvel, iyvel, imass, iplanet);
    (*u).pushback(*planet);
    cout << *planet << std::endl;
  }
 //  window1.draw(background_sprite);
 //  (*u).setPositions(uRadius, window1size);
 cout << *u << std::endl;
```

```
71    // (*u).draw(window1);
72
73
74    while (window1.isOpen())
75      {
76        sf::Event event;
77        while(window1.pollEvent(event)){
78     if (event.type == sf::Event::Closed)
79       {
80         window1.close();
81       }
82     else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
83       {
84         window1.close();
85       }
86        }
87        window1.draw(background_sprite);
88        sf::Time counterTime = timeCounter.getElapsedTime();                    // Refreshes time counter
89        float time = counterTime.asSeconds();
90        dispTime += time;
91        if(dispTime < maxTime){
92    (*u).findForces();
93    (*u).step(dispTime);
94        }
95        (*u).setPositions(uRadius, window1size);
96        counterText.setString(std::to_string(time));
97        (*u).draw(window1);
98        window1.draw(counterText);
99        window1.display();
100       window1.clear();
101     }
102   return 0;
103 }
```

body.hpp

```
#ifndef BODY_HPP
#define BODY_HPP
#include <iostream>
#include <math.h>
#include <cmath>
#include <stdlib.h>
#include <string>
#include <fstream>
#include <vector>
#include <memory>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
using namespace std;

class cBody: public sf::Drawable
{
public:
  cBody();
  cBody(double x_position, double y_position, double x_velocity, double y_velocity, double bmass, std::string planet);
  void setXPos(double newPosition);
  void setYPos(double newPosition);
  void setXVel(double newVelocity);
  void setYVel(double newVelocity);
  double getXPos();
  double getYPos();
  double getXVel();
  double getYVel();
  double getMass();
  void setPositions(double radius, sf::Vector2u windowSize);
  void findForces(double reg_mass, double xSpot, double ySpot);
  void step(double seconds);
  friend ostream& operator <<(std::ostream &output, cBody& c);
  friend istream& operator >>(std::istream &input, cBody& d)
  {
    input >> d.xpos >> d.ypos >> d.xvel >> d.yvel >> d.mass >> d.picture;

    if(!d.pImage.loadFromFile(d.picture)){
      std::cerr << "Picture did not load";
    }
    d.pTexture.loadFromImage(d.pImage);
    d.pSprite.setTexture(d.pTexture);

  return input;
 }
  void draw(sf::RenderTarget& obj, sf::RenderStates status) const;
private:
  double xpos;
  double ypos;
  double xvel;
  double yvel;
  double mass;
  std::string picture;

  sf::Image pImage;
  sf::Sprite pSprite;
  sf::Texture pTexture;
  double xforce = 0.0;
  double yforce = 0.0;
};

class universe{
public:
  void draw(sf::RenderWindow& window1){
```

```
66      for(unsigned int i = 0; i < uni.size();i++){
67        window1.draw(uni.at(i));
68      }
69    }
70    void pushback(cBody obj){uni.push_back(obj);}
71
72    void step(double timePassed){
73      for(unsigned int i = 0; i < uni.size(); i++){
74        uni.at(i).step(timePassed);
75      }
76    }
77
78    void setPositions(double radius, sf::Vector2u windowSize){
79      for(unsigned int i = 0; i < uni.size(); i++){
80        uni.at(i).setPositions(radius, windowSize);
81      }
82    }
83
84    cBody begin(){return uni.at(0);}
85    cBody end(){return uni.at(uni.size() - 1);}
86
87    friend ostream& operator <<(std::ostream &output, universe& c){
88      for(unsigned int i = 0; i < c.uni.size(); i++){
89        output << c.uni.at(i) << endl;
90      }
91      return output;
92    }
93
94    void findForces(){
95      unsigned int i;
96      for(i = 0; i < (uni.size() - 1); i++){
97        uni.at(i).findForces(uni.at(i+1).getMass(), uni.at(i+1).getXPos(), uni.at(i+1).getYPos());
98      }
99    }
100
101 private:
102   std::vector<cBody> uni;
103 };
104
105 #endif
```

body.cpp

```cpp
#include "body.hpp"

cBody::cBody(){
  return;
}

cBody::cBody(double x_position, double y_position, double x_velocity, double y_velocity, double bmass, std::string planet)
{
  xpos = x_position;
  ypos = y_position;
  xvel = x_velocity;
  yvel = y_velocity;
  mass = bmass;
  picture = planet;



  if(!pImage.loadFromFile(picture)){
    return;
  }
  pTexture.loadFromImage(pImage);
  pSprite.setTexture(pTexture);

  std::cout << xpos << std::endl << ypos << std::endl;
}

std::ostream& operator <<(std::ostream &output, cBody& c){
  output << "current x Position: " << c.xpos << std::endl;
  output << "current y Postion: " << c.ypos << std::endl;
  output << "current x Velocity: " << c.xvel << std::endl;
  output << "current y Velocity: " << c.yvel << std::endl;
  output << "Particle Mass: " << c.mass << std::endl;
  output << "Particle Name: " << c.picture << std::endl;
  return output;
}
void cBody::draw(sf::RenderTarget& obj, sf::RenderStates status) const
{
  obj.draw(pSprite);
}

void cBody::setXPos(double newPosition){xpos = newPosition;}

void cBody::setYPos(double newPosition){ypos = newPosition;}

void cBody::setXVel(double newVelocity){xvel = newVelocity;}

void cBody::setYVel(double newVelocity){yvel = newVelocity;}

void cBody::setPositions(double radius, sf::Vector2u windowSize){
  double xW, yW;
  xW = windowSize.x;
  yW = windowSize.y;
  xpos = ((xW/radius) * xpos/2) + (xW/ 2);
  ypos = ((yW/radius) * ypos/2) + (yW / 2);
  pImage.loadFromFile(picture);
  pTexture.loadFromImage(pImage);
  pSprite.setTexture(pTexture);

  pSprite.setPosition(sf::Vector2f(xpos,ypos));
  }

void cBody::findForces(double reg_mass, double xSpot, double ySpot){
  const double G = 6.67e-11;
  double deltaX = xSpot - xpos;
  double deltaY = ySpot - ypos;
  double r_square = pow(deltaX, 2) + pow(deltaY, 2);
```

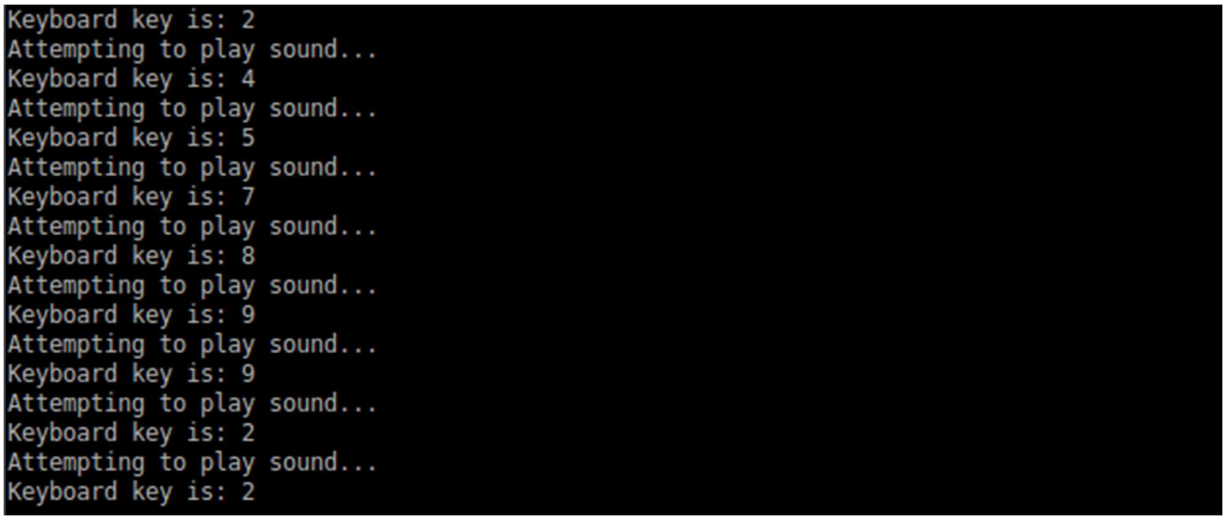
```
67    double r = sqrt(r_square);
68    double F = (G * mass * reg_mass) / r_square;
69
70    xforce = xforce + (F * (deltaX/r));
71    yforce = yforce + (F * (deltaY/r));
72  }
73
74  void cBody::step(double seconds){
75    double xacc, yacc;
76    xacc = (xforce / mass) * seconds;
77    yacc = (yforce / mass) * seconds;
78
79    xvel += xacc;
80    yvel += yacc;
81
82    xpos += (xvel * seconds);
83    ypos += (yvel * seconds);
84  }
85
86  double cBody::getXPos(){return xpos;}
87
88  double cBody::getYPos(){return ypos;}
89
90  double cBody::getXVel(){return xvel;}
91
92  double cBody::getYVel(){return yvel;}
93
94  double cBody::getMass(){return mass;}
```

PS4: Synthesizing a Plucked String Sound

The goal of this assignment was to implement the Karplus-Strong algorithm to represent audio and add the ability for users to be able to play notes from keyboard inputs.

The first class used was the ringBuffer class which implemented a queue data structure using the 16-bit integer data type to represent frequencies. Next, I created the StringSound class which, using the ringBuffer class, applies the Karplus-Strong algorithm to properly represent the values of the frequencies.

This assignment taught me how to use SFML's sound functions. I also formatted my code to comply with Google's cpplint coding guidelines.



Makefile

```
1   GuitarHero: ringbuffer.o guitarstring.o main.o
2       g++ -o guitarhero main.o guitarstring.o ringbuffer.o -lsfml-system -lsfml-audio
        -lsfml-graphics -lsfml-window
3
4   main.o: ringbuffer.hpp guitarstring.hpp main.cpp
5       g++ -c main.cpp -ansi -pedantic -Wall -Werror
6
7   GuitarString.o: ringbuffer.hpp guitarstring.hpp guitarstring.cpp
8       g++ -c guitarstring.cpp -ansi -pedantic -Wall -Werror
9
10  ringBuffer.o: ringbuffer.hpp ringbuffer.cpp
11      g++ -c ringbuffer.cpp -ansi -pedantic -Wall -Werror
12
13  clean:
14      rm -f *.o *~ GuitarHero
15
```

main.cpp

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

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

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

#include "ringbuffer.hpp"
#include "guitarstring.hpp"

#define CONCERT_A 220.0
#define SAMPLES_PER_SEC 44100

using namespace std;

vector<int16_t> makeSamplesFromString(GuitarString gs) {
  vector<int16_t> samples;

  gs.pluck();
  int duration = 8;  // seconds
  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 Guitar Hero Lite");
  sf::Event event;
  double freq;
  string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";

  vector<vector<int16_t> > samples(37);
  vector<sf::Sound> sounds(37);
  vector<sf::SoundBuffer> soundBuffers(37);

  for(int i = 0; i < 37; i++) {
    freq = 440 * pow(2, (i - 24) / 12.0);
    GuitarString gs(freq);
    samples[i] = makeSamplesFromString(gs);

    if (!soundBuffers[i].loadFromSamples(&samples[i][0], samples[i].size(), 2,
                                         SAMPLES_PER_SEC))
      throw std::runtime_error("sf::SoundBuffer: failed to load from samples.");

    sounds[i].setBuffer(soundBuffers[i]);
  }

  /*  vector<int16_t> sample;

  double freq = CONCERT_A;
  GuitarString gs1 = GuitarString(freq);
  sf::Sound sound1;
  sf::SoundBuffer buf1;
  sample = makeSamplesFromString(gs1);
```

```
  if (!buf1.loadFromSamples(&sample[0], sample.size(), 2, SAMPLES_PER_SEC))
    throw std::runtime_error("sf::SoundBuffer: failed to load from samples.");
    sound1.setBuffer(buf1); */

  while (window.isOpen()) {
    while (window.pollEvent(event)) {
      switch (event.type) {
      case sf::Event::Closed:
        window.close();
        break;

      case sf::Event::TextEntered:
        if (event.text.unicode < 128) {
          string temp;
          temp += static_cast<char>(event.text.unicode);
          cout << "Playing sound associated with " << temp << endl;
          int index = keyboard.find(temp);
          sounds[index].play();
        }
        break;
      default:
        break;
      }


      window.clear();
      window.display();
    }
  }

  return 0;
}
```

ringbuffer.hpp

```
#ifndef _RING_BUFFER_HPP
#define _RING_BUFFER_HPP

#include <stdint.h>
#include <vector>

class ringBuffer {
 private:
  std::vector<int16_t> buffer;
  int _size;
  int _capacity;
  int first;
  int last;
 public:
  explicit ringBuffer(int capacity);
  int size() const;
  bool isEmpty() const;     // Checks to see if the queue is empty
  bool isFull() const;     // Checks to see if the queue is full
  void enqueue(int16_t arg);   // Adds to the end and throws an exception if the queue is full
  int16_t dequeue();     // Subtracts from the front and throws an exception if the queue is empty
  int16_t peek() const;
  void emptyBuffer();
};

#endif
```

ringbuffer.cpp

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

ringBuffer::ringBuffer(int capacity) {
  try {          // Throws an exception if the capacity given is less than 1
    if (capacity < 1)
      throw std::invalid_argument("Capacity must be greater than zero");
  } catch(std::invalid_argument& ia) {
    std::cerr << "RB constructor: capacity must be greater than zero.";
    std::cerr << std::endl;
    throw ia;
  }

  buffer.reserve(capacity);
  for (int i = 0; i < capacity; i++)
    buffer.push_back(0);

  first = 0;
  last = 0;
  _size = 0;
  _capacity = capacity;
}

int ringBuffer::size() const {
  return _size;
}

bool ringBuffer::isEmpty() const {
  return _size > 0 ? false : true;
}

bool ringBuffer::isFull() const {
  return _size < _capacity ? false : true;
}

void ringBuffer::enqueue(int16_t arg) {
  try {          // Throws an exception when the queue is full
    if (isFull())
      throw std::runtime_error("Buffer is full");
  } catch(std::runtime_error& re) {
    std::cerr << "Enqueue Error: Buffer is full" << std::endl;
    throw re;
  }

  buffer[last] = arg;

  if (last == _capacity - 1)
    last = 0;
  else
    last++;

  _size++;
}

int16_t ringBuffer::dequeue() {
  try {          // Throws an exception when the queue is empty
    if (isEmpty())
      throw std::runtime_error("Buffer is empty");
  } catch(std::runtime_error& re) {
    std::cerr << "Dequeue Error: Buffer is empty" << std::endl;
    throw re;
  }
```

```
66   int16_t temp = peek();
67
68   if (first == _capacity - 1)
69     first = 0;
70   else
71     first++;
72
73   _size--;
74
75   return temp;
76 }
77
78 int16_t ringBuffer::peek() const {
79   try {          // Throws an exception when the queue is empty
80     if (isEmpty())
81       throw std::runtime_error("Buffer is empty");
82   } catch(std::runtime_error& re) {
83     std::cerr << "Peek Error: Buffer is empty" << std::endl;
84     throw re;
85   }
86
87   return buffer[first];
88 }
89
90 void ringBuffer::emptyBuffer() {
91   first = 0;
92   last = 0;
93   _size = 0;
94 }
```

guitarstring.hpp

```
#ifndef _GUITARSTRING_HPP
#define _GUITARSTRING_HPP

#include <stdint.h>
#include <vector>
#include "ringbuffer.hpp"

class GuitarString {
 private:
  ringBuffer buffer;
  int time;
 public:
  explicit GuitarString(double frequency);
  explicit GuitarString(std::vector<int16_t> init);
  void pluck();
  void tic();
  int16_t sample() const;
  int time() const;
};

#endif
```

guitarstring.cpp

```cpp
#include <stdint.h>
#include <vector>
#include <cmath>
#include <cstdlib>
#include "guitarstring.hpp"
#include "ringbuffer.hpp"

GuitarString::GuitarString(double frequency) : buffer(ceil(44100/frequency)) {
  time = 0;
}

GuitarString::GuitarString(std::vector<int16_t> init) : buffer(init.size()) {
  for (unsigned i = 0; i < init.size(); i++)
    buffer.enqueue(init[i]);

  time = 0;
}

void GuitarString::pluck() {
  buffer.emptyBuffer();

  while (!buffer.isFull())
    buffer.enqueue((int16_t)(rand() & 0xffff));
}

void GuitarString::tic() {
  int n1 = buffer.dequeue();
  int n2 = buffer.peek();

  buffer.enqueue(0.5 * 0.996 * (n1 + n2));

  time++;
}

int16_t GuitarString::sample() const {
  return buffer.peek();
}

int GuitarString::time() const {
  return time;
}
```

PS6: Markov Model of Natural Language

The goal of this assignment was to build a Markov model used to predict occurrences of letters of the alphabet with a fixed probability.

To accomplish this, I created a class called MModel which was made to represent a Markov Chain.

Using a k-gram which represents any string of k characters this program attempts to predict the next letter.

Makefile

```
CC = g++
CFLAGS = -g -Wall -Werror -std=c++11 -pedantic
BOOST = -lboost_unit_test_framework

all: ps6 mmtest

ps6: textGen.o model.o
    $(CC) textGen.o model.o -o ps6

mmtest: mmtest.o model.o
    $(CC) mmtest.o model.o -o mmtest $(BOOST)

TextGenerator.o: textGen.cpp model.hpp
    $(CC) -c textGen.cpp model.hpp $(CFLAGS)

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

mmtest.o: mmtest.cpp
    $(CC) -c mmtest.cpp $(BOOST)

clean:
    rm *.o
    rm *~
    rm ps6
    rm mmtest

```

main.cpp

```
/*  Copyright Anas Ahmed  */

#include <string>
#include "model.hpp"

int main(int argc, const char* argv[]) {
  // Make sure that the program was called properly
  if (argc != 3) {
    std::cout << "Usage: ./TextGenerator (int K) (int T)\n";
    return 0;
  }

  // For some reason, has to be a string to please -fpermissive
  // error: cast from ‘const char*’ to ‘int’ loses precision [-fpermissive]
  std::string str_k(argv[1]);
  std::string str_t(argv[2]);

  // Converts the command line arguments as ints
  int k = std::stoi(str_k);
  int t = std::stoi(str_t);

  // Now takes input
  std::string input = "";
  std::string current_txt = "";   // Set inputs to NULL before, to be safe

  while (std::cin >> current_txt) {
    input += " " + current_txt;
    current_txt = "";
  }

  // Outputs the users input for troubleshooting
  std::cout << "ORIGINAL INPUT TEXT: \n\n";

  // Only show the first T characters
  for (int a = 0; a < t; a++) {
    std::cout << input[a];

    // Fomrats the text better
    if (input[a] == '.' || input[a] == '!') {
      std::cout << "\n";
    }
  }

  // First make a final output string to use.
  std::string output_string = "";

  // Initializes an mModel object using input as the string and k as the int
  mModel amazing(input, k);

  output_string += "" + amazing.gen(input.substr(0, k), t);

  // Creates space before the output for visibility
  std::cout << "\n\nFINAL OUTPUT TEXT: \n\n";

  // Formats output similar to the input
  for (int a = 0; a < t; a++) {
    std::cout << output_string[a];

    // More text formatting
    if (output_string[a] == '.' || output_string[a] == '!') {
      std::cout << "\n";
    }
  }
  std::cout << "\n";

```

```
66    return 0;
67  }
```

model.hpp

```
1  /*  Copyright Anas Ahmed  */
2
3  #ifndef MODEL_HPP
4  #define MODEL_HPP
5  #include <algorithm>
6  #include <iostream>
7  #include <map>
8  #include <string>
9  #include <stdexcept>
10
11
12 class mModel {
13  public:
14   /* Creates a Markov model of order k from the given text.
15    * Assume that the text has a length of at least k.              */
16   mModel(std::string text, int k);
17
18   // Order k of Markov model
19   int order();
20
21   /* Returns number of occurrences of kgram in text.
22    * Throws an exception if kgram is not of length k              */
23   int freq(std::string kgram);
24
25   /* Number of times that character c follows kgram
26    * If order = 0, return number of times char c appears
27    * -> throw an exception if kgram is not of length k            */
28   int freq(std::string kgram, char c);
29
30   /* Random character following given kgram
31    * -> Throw an exception if kgram is not of length k.
32    * -> Throw an exception if no such kgram.                      */
33   char randk(std::string kgram);
34
35   /* Generate a string of length T characters by simulating a
36    * trajectory through the corresponding Markov chain.           */
37   std::string gen(std::string kgram, int T);
38
39   /* overload the stream insertion operator and display the internal
40    * state of the Markov Model. Print out the order, the alphabet,
41    * and the frequencies of the k-grams and k+1-grams.            */
42   friend std::ostream& operator<< (std::ostream &out, mModel &mm);
43
44  private:
45   int _order;
46   std::map <std::string, int> kgrams;
47   std::string alphabet;
48 };
49 #endif
```

model.cpp

```
/*  Copyright Anas Ahmed  */

#include "model.hpp"
#include <algorithm>
#include <map>
#include <string>
#include <stdexcept>
#include <vector>
#include <utility>

/* Creates a Markov model of order k from the given text.
 * Assumes that the text has a length of at least k.             */
mModel::mModel(std::string text, int k) {
  // Set the order.
  _order = k;

  // Seed the random number generator
  srand((int)time(NULL)); //NOLINT

  std::string text_circular = text;   // Make a copy.

  for (int a = 0; a < _order; a++) {
    text_circular.push_back(text[a]);
  }

  int text_len = text.length();  // Find the text's length

  // Now we need to set the alphabet.
  char tmp;
  bool inAlpha = false;

  // Go through the entire text and pick out all the individual letters
  for (int i = 0; i < text_len; i++) {
    tmp = text.at(i);
    inAlpha = false;

    // Check if this letter has been added to the alphabet
    for (unsigned int y = 0; y < alphabet.length(); y++) {
      // tmp is already in the alphabet!
      // Also ignore upper case,
      if (alphabet.at(y) == tmp) {
        inAlpha = true;   // Match it as being in the alphabet.
      }
    }

    // If tmp isn't in the alphabet, isAlpha should be false
    if (!inAlpha) {
      alphabet.push_back(tmp);
    }
  }

  // Sorts alphabet, alphabetically
  std::sort(alphabet.begin(), alphabet.end());

  std::string tmp_str;
  int x, y;

  // Do up to text.length() substring comparisons.
  // This first part just "finds" kgrams and puts a "0" next to them.
  for (x = _order; x <= _order + 1; x++) {
    // Go through the entire text.
    for (y = 0; y < text_len; y++) {
      // This collects all given kgrams, and adds a "0" that we can use
      // later on to increment. We basically find ALL the kgrams, then
      // find their occurrences after.

```

```
66        // current kgram we want.
67        tmp_str.clear();
68        tmp_str = text_circular.substr(y, x);
69
70        // Insert the 0.
71        kgrams.insert(std::pair<std::string, int>(tmp_str, 0));
72      }
73    }
74
75    // Need an iterator for going through the kgrams map.
76    std::map<std::string, int>::iterator it;
77    int count_tmp = 0;
78
79    // Uses same loop to count the kgrams
80    for (x = _order; x <= _order + 1; x++) {
81      for (y = 0; y < text_len; y++) {
82        // Let's get the current kgram we're comparing against.
83
84        tmp_str.clear();
85        tmp_str = text_circular.substr(y, x);
86
87        // Gets kgrams current count
88        it = kgrams.find(tmp_str);
89        count_tmp = it->second;
90
91        // Increment the count by 1.
92        count_tmp++;
93        kgrams[tmp_str] = count_tmp;
94      }
95    }
96  }
97
98
99  // Returns _order
100 int mModel::order() {
101   return _order;
102 }
103
104
105 /* Number of occurrences of kgram in text.      */
106 int mModel::freq(std::string kgram) {
107   // Throws an exception if kgram does not have length k
108   if (kgram.length() != (unsigned)_order) {
109     throw
110       std::runtime_error("Error - kgram not of length k.");
111   }
112
113   // Uses std::map::find to search for the kgram
114   std::map<std::string, int>::iterator it;
115   it = kgrams.find(kgram);
116
117   // Returns 0 if unable to find
118   if (it == kgrams.end()) {
119     return 0;
120   }
121
122   return it->second;
123 }
124
125
126 /* Number of times that character c follows kgram     */
127 int mModel::freq(std::string kgram, char c) {
128   // Throws an exception if kgram does not have length k
129   if (kgram.length() != (unsigned)_order) {
130     throw
131       std::runtime_error("Error - kgram not of length k.");
132
```

```
133   }
134
135   // use std::map::find to see if we can find the kgram + c.
136   std::map<std::string, int>::iterator it;
137   kgram.push_back(c);
138   it = kgrams.find(kgram);
139
140   // If it equals map::end, we didn't find it, so return 0.
141   if (it == kgrams.end()) {
142     return 0;
143   }
144
145   return it->second;
146 }
147
148
149 /* Returns a random character following the given kgram   */
150 char mModel::randk(std::string kgram) {
151   // Throws an exception if kgram does not have length k
152   if (kgram.length() != (unsigned)_order) {
153     throw std::runtime_error("Error - kgram not of length k (randk)");
154   }
155
156   // Create an iterator for the map
157   std::map<std::string, int>::iterator it;
158
159   // Search through and see if we find the given kgram.
160   it = kgrams.find(kgram);
161
162   // Throws an exception if kgram is not found
163   if (it == kgrams.end()) {
164     throw std::runtime_error("Error - Could not find the given kgram! (randk)");
165   }
166
167   // Get the freq of the given kgram.
168   int kgram_freq = freq(kgram);
169
170   // This should be an int from 1 to the total number of possible letters.
171   int random_value = rand() % kgram_freq; //NOLINT
172
173   double test_freq = 0;
174   double random_num = static_cast<double>(random_value) / kgram_freq;
175   double last_values = 0;
176
177   // Go through all the letters.
178   for (unsigned int a = 0; a < alphabet.length(); a++) {
179     // Calculates the probability as a double
180     test_freq =  static_cast<double>(freq(kgram, alphabet[a])) / kgram_freq;
181
182
183     if (random_num < test_freq + last_values && test_freq != 0) {
184       // Returns matching letter
185       return alphabet[a];
186     }
187
188     last_values += test_freq;
189   }
190
191   // Used for error checking only
192   return '-';
193 }
194
195
196
197 std::string mModel::gen(std::string kgram, int T) {
198   // Throw an exception if kgram is not of length k.
199
```

```
133   }
134
135   // use std::map::find to see if we can find the kgram + c.
136   std::map<std::string, int>::iterator it;
137   kgram.push_back(c);
138   it = kgrams.find(kgram);
139
140   // If it equals map::end, we didn't find it, so return 0.
141   if (it == kgrams.end()) {
142     return 0;
143   }
144
145   return it->second;
146 }
147
148
149 /* Returns a random character following the given kgram   */
150 char mModel::randk(std::string kgram) {
151   // Throws an exception if kgram does not have length k
152   if (kgram.length() != (unsigned)_order) {
153     throw std::runtime_error("Error - kgram not of length k (randk)");
154   }
155
156   // Create an iterator for the map
157   std::map<std::string, int>::iterator it;
158
159   // Search through and see if we find the given kgram.
160   it = kgrams.find(kgram);
161
162   // Throws an exception if kgram is not found
163   if (it == kgrams.end()) {
164     throw std::runtime_error("Error - Could not find the given kgram! (randk)");
165   }
166
167   // Get the freq of the given kgram.
168   int kgram_freq = freq(kgram);
169
170   // This should be an int from 1 to the total number of possible letters.
171   int random_value = rand() % kgram_freq; //NOLINT
172
173   double test_freq = 0;
174   double random_num = static_cast<double>(random_value) / kgram_freq;
175   double last_values = 0;
176
177   // Go through all the letters.
178   for (unsigned int a = 0; a < alphabet.length(); a++) {
179     // Calculates the probability as a double
180     test_freq =  static_cast<double>(freq(kgram, alphabet[a])) / kgram_freq;
181
182
183     if (random_num < test_freq + last_values && test_freq != 0) {
184       // Returns matching letter
185       return alphabet[a];
186     }
187
188     last_values += test_freq;
189   }
190
191   // Used for error checking only
192   return '-';
193 }
194
195
196
197 std::string mModel::gen(std::string kgram, int T) {
198   // Throw an exception if kgram is not of length k.
199
```

```
200   if (kgram.length() != (unsigned)_order) {
201     throw std::runtime_error("Error - kgram not of length k. (gen)");
202   }
203
204   // The final string we will return. We'll build it up over time.
205   std::string final_string = "";
206
207   // Temp char
208   char return_char;
209
210   // Add the kgram to it.
211   final_string += "" + kgram;
212
213
214   for (unsigned int a = 0; a < (T - (unsigned)_order); a++) {
215     return_char = randk(final_string.substr(a, _order));
216
217     final_string.push_back(return_char);
218   }
219
220   return final_string;
221 }
222
223 std::ostream& operator<< (std::ostream &out, mModel &mm) {
224   out << "\n_Order: " << mm._order << "\n";
225   out << "Alphabet: "<< mm.alphabet << "\n";
226
227   out << "Kgrams map: \n\n";
228
229   std::map<std::string, int>::iterator it;
230
231   for (it = mm.kgrams.begin(); it != mm.kgrams.end(); it++) {
232     out << it->first << "\t" << it->second << "\n";
233   }
234
235   return out;
      }
```