Computing IV Sec 202: Project Portfolio

Wendy Carvalho

Spring 2024

Contents

Time to Complete Portfolio: 10 hours

1 PS0: Hello SFML

1.1 Discussion

The main purpose of this project was to set up a build environment using the SFML library. With this library, a window was to be generated with a green circle from the SFML documentation demo, and another sprite that was to be movable using the arrow keys.

1.2 What I accomplished

For this project, I created what is, essentially, a prototype of a mini-game that features an alien in a UFO that can shoot lasers. It responds to ADSW keys (left, right, down, and up) to move around. In addition, the left mouse click is to shoot these laser beams. This can be seen in Figure 1.

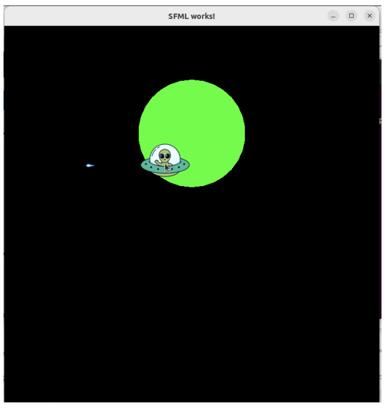


Figure 1: PS0 output

1.3 Design Decisions and Implementations

As an introductory project to SFML, there were not many design decisions to be made regarding algorithms or data structures. However, this project involved manipulating sprites through keystrokes and including an additional feature, which I decided would be generating new sprites after clicking the left mouse button, and these would move in the direction the original stable sprite was facing.

1.4 What I already knew

Going into this first project, I was already familiar with C++ from my previous class, Computing III, so navigating the language was easy. The only other thing I knew was that I wanted to create a game since the goal of this project was to create something interactive in the sense that keystrokes should cause a change in the program.

1.5 What I learned

This project taught me how to build my first SFML program, as simple as it was. Further, I learned the basic mechanics of a simple computer game–allowing the user to use keys to move and the left mouse click to shoot laser beams.

Of course, I also learned how to generate an SFML window and generate a sprite, manipulating it while the window is open. The sprite in this scenario, an alien, automatically moves up and down by having its y-position update with every frame and flipping horizontally whenever the left or right key is pressed. This was done by updating the sprite's scale to 1 or -1 if the sprite was not already facing that direction.

Lastly, I learned the essentials of linting code.

1.6 Challenges

I encountered a few challenges with the project. First, when A (left) or D (right) are pressed, the sprite "jumps" away from its initial position on the x-axis. To fix this, I tried to set its new position after the user presses the keystroke so that it would be drawn in the same area before it was flipped, but this was unsuccessful.

The second challenge was due to the fact that I wanted to make the game more interactive by adding laser beams, as normal games would usually have an "attack" feature. Initially, there was only one laser beam sprite, and, using the scale mechanism, I would flip the sprite depending on the direction the alien was facing. But I found that this made it so that sometimes the laser beam would change directions on the screen. So, instead, I used two sprites for the laser beams, one facing each direction, and the program "draws" each of them independently.

Additionally, these laser beams, however, don't always show up on the screen after pressing the left mouse click. Sometimes there is a delay, and it may have to do with how the frames are being updated. They also don't show up at all if the alien isn't moved using the keystrokes right after loading the program.

Lastly, if the direction is changed too quickly at the beginning of the program and lasers are shot in that period, one laser might be seen flying at the top of the window instead of out of the UFO.

1.7 Codebase

Makefile:

```makefile
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -I/opt/homebrew/Cellar/sfml/2.6.1/include -o sfml-app -L/opt/homebrew/Cellar/sfml/2.6.1/lib -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
# Your .hpp files
DEPS =
# Your compiled .o files
OBJECTS =
# The name of your program
PROGRAM = sfml-app

.PHONY: all clean lint


all: $(PROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<

$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

Main routine (main.cpp):

```cpp
// Copyright 2024 Wendy Carvalho

#include <SFML/Graphics.hpp>

int main() {
  // sets up window, title, and size
  sf::RenderWindow window(sf::VideoMode(700, 700), "SFML works!");
  window.setVerticalSyncEnabled(true);
```

```

// how fast things are moving
window.setFramerateLimit(5);

// green circle
sf::CircleShape shape(100.f);
shape.setPosition(250, 100);
shape.setFillColor(sf::Color::Green);

// set up alien image
sf::Texture sprite_1;
if (!sprite_1.loadFromFile("sprite.png"))
  return EXIT_FAILURE;
sf::Sprite sprite(sprite_1);

// define initial sprite position
float yPosition = 0.0f;
int newX = 350;
int newY = 200;

// left mouse click initially false
bool mouseClicked = false;

// set up laser facing left
sf::Texture sprite_2;
if (!sprite_2.loadFromFile("laser_left.png"))
  return EXIT_FAILURE;
sf::Sprite spriteLaserLeft(sprite_2);

// set up laser facing right
sf::Texture sprite_3;
if (!sprite_3.loadFromFile("laser_right.png"))
  return EXIT_FAILURE;
sf::Sprite spriteLaserRight(sprite_3);

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

  // mechanism for alien bobbing up and down
  if (yPosition == 2)
    yPosition -= 2;
  else
    yPosition += 2;

  // move left
  if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::A)) {
    if (sprite.getScale().x != 1) {
      sprite.setScale(1, 1);
      sprite.setPosition(newX-100, newY);
    }
    newX += -20;
    sprite.move(newX, 0.0f);
  }

  // move right
```

```cpp
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::D)) {
      if (sprite.getScale().x != -1) {
        sprite.setScale(-1, 1);
        sprite.setPosition(newX+100, newY);
      }
      newX += 20;
      sprite.move(newX, 0.0f);
    }

    // move up
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::W)) {
      newY += -20;
      sprite.move(0.0f, newY);
    }

    // move down
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::S)) {
      newY += 20;
      sprite.move(0.0f, newY);
    }

    if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
      mouseClicked = true;
      if (sprite.getScale().x == 1)
        spriteLaserLeft.setPosition(newX+100, newY+50);
      else
        spriteLaserRight.setPosition(newX-100, newY+50);
    }

    // update alien position (bobbing)
    sprite.setPosition(newX, newY+yPosition);

    // defines laser beam direction based on that of alien sprite
    if (sprite.getScale().x == 1)
      spriteLaserLeft.move(-100.0f, 0);
    else
      spriteLaserRight.move(100.0f, 0);

    window.clear();
    window.draw(shape);
    window.draw(sprite);

    if (mouseClicked) {
      if (sprite.getScale().x == 1)
        window.draw(spriteLaserLeft);
      else
        window.draw(spriteLaserRight);
    }

    window.display();
  }

  return 0;
}
```

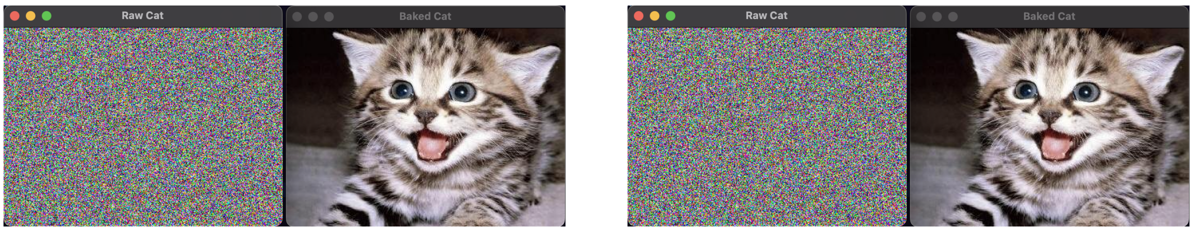
2 PS1: PhotoMagic

2.1 Discussion

The goal of this project was to first write a program that produces pseudo-random bits by simulating a linear feedback shift register, then implement a simple form of digital picture encryption using the LFSR and display the original image in one window and the altered image in a second window. The project was also an introduction to creating test cases from the Boost library.

2.2 What I accomplished

This program can successfully encode and decode an image, making it static and reverting it through the LFSR. This project also incorporates the Boost framework tests in order to check the efficiency of the functions in the `FibLFSR` class. For this, I was able to create 6 test cases. Figure 2 is an example of an image being encoded (becoming static), and the same encoded image being decoded (reverting to its normal non-static state).



(a) Image of cat before and after encoding

(b) Image of cat before and after decoding

Figure 2: PS1 output when encoding and decoding using its LFSR

2.3 Design Decisions and Implementations

I implemented the LFSR by creating a class called `FibLFSR`, which holds the initial bit seed, its successive states, and its size. My first thought was to store the seed in an integer array because that seemed the simplest to work with, and dynamically allocating memory is easy. Looking at the project after having finished it, I would choose to use a vector for it instead as it would have been even easier to deal with the memory and shifting bits.

Since the initial seed is passed in as a string, I used the function `c_str()`to convert it into an array of chars (stored in `bitCharacters`), and then this is stored in the integer array called bits. Since `bitCharacters` is a char array, each char had to have '0' subtracted from it to get its ASCII value so it could be stored properly as either a 0 or a 1 in the bits array. In addition to keeping the seed in this array called bits, after storing the initial seed in it, I reversed it so that the 0th element would be the 15th, and so on. This made it easier to specify which bits to XOR (the tap positions).

To develop the LFSR, I created a function inside `FibLFSR` called `step()` which uses 3 tap positions to add a new bit (0 or 1) to the 1st position of a 16-bit seed through an XOR. To add this new bit, it shifts every bit to the left once, as it cannot exceed the number of bits initially determined. In addition, each time `step()` is called, the new state of the seed is stored in a string called `currentState`.

The other function needed for this project was `generate()`, also inside the `FibLFSR` class, which takes an integer `k` and performs `step()` `k` amount of times, which, in the end, moves the seed `k` bits to the left and creates `k` new bits.

To encode the image, I created a function called `transform()` that iterates through each pixel of a photo (first the rows, incrementing the y, then incrementing x and restarting the y coordinate) and XOR each RGB component at that specific pixel with `generate()`. `transform()`, therefore, takes an image and a seed, both provided through the command line `transform()`. In turn, it calls `generate()`, which returns an 8-bit number. This number is then used to alter each image pixel's RGB (Red Green Blue) component. The image is then imported into a texture and saved as an output file as well, which has a name also provided by the command line.

As extra credit, I added an private integer member called `seedSize` to store the length of the seed (in case more than 16 bits should be used in the future). To access `seedSize` and `currentState`, I also included accessor functions for them (`getState()` and `getSeedSize()`). I also added a function to access the initial seed, `getInitialSeed()` for testing purposes. Of course, I also added a destructor because I dynamically allocated memory for the member arrays of `FibLFSR`.

2.4 What I already knew

Going into PS1, I was already slightly familiar with creating an SFML window from PS0 and creating C++ classes from my Computing III course. I was also familiar with shifting bits from my previous Assembly Programming Language course and Computer Architecture.

2.5 What I learned

I learned about linear feedback shift registers and how to simulate one using XOR operations. I also learned how to iterate through image pixels and alter their RGB components.

2.6 Challenges

I did not encounter any major challenges with this project.

2.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.83.0/include -I/opt/homebrew/Cellar/sfml/2.6.1/include
LIBDIR = -L/opt/homebrew/Cellar/boost/1.83.0/lib -L/opt/homebrew/Cellar/sfml/2.6.1/lib# Your .hpp files
DEPS = FibLFSR.hpp PhotoMagic.hpp
# Your compiled .o files
OBJECTS = FibLFSR.o PhotoMagic.o
# The name of your program
PROGRAM = PhotoMagic

.PHONY: all clean lint

all: $(PROGRAM) test $(PROGRAM).a

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $< $(INCLUDEDIR)

$(PROGRAM): $(OBJECTS) main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

$(PROGRAM).a: $(OBJECTS)
    ar rcs $@ $^ $(LIBDIR) $(LIB)

test: $(OBJECTS) test.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

clean:
    rm *.o $(PROGRAM) test

lint:
    cpplint *.cpp *.hpp
```

Main routine (main.cpp):

```cpp
// Copyright 2024 Wendy Carvalho

#include <iostream>

#include "FibLFSR.hpp"
#include "PhotoMagic.hpp"

int main() {
  std::string seed_input, image_input, image_output;
  sf::Image image;
  sf::Texture texture1, texture2;
  sf::Sprite sprite1, sprite2;

  std::cin >> image_input >> image_output >> seed_input;

  PhotoMagic::FibLFSR seed(seed_input);

  if (!image.loadFromFile(image_input)) {
    std::cerr << "failed to open the image" << std::endl;
    return -1;
  }

  texture1.loadFromImage(image);
  sprite1.setTexture(texture1);

  // encode
  transform(image, &seed);
  sf::Vector2u size = image.getSize();

  sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Raw Cat");
  sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "Baked Cat");

  texture2.loadFromImage(image);
  sprite2.setTexture(texture2);

  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(sprite1);
    window1.display();
    window2.clear();
    window2.draw(sprite2);
    window2.display();
  }

  // write the file
  if (!image.saveToFile(image_output))
    std::cerr << "failed to save the image" << std::endl;

  return 0;
}
```

Header file for the first supporting class file (FibLFSR.hpp):

```cpp
// Copyright 2024 Wendy Carvalho

#pragma once

#ifndef FIBLFSR_HPP
#define FIBLSFR_HPP

#include <array>
#include <cstring>
#include <iostream>
#include <sstream>
#include <string>

namespace PhotoMagic {
class FibLFSR {
 public:
    // Constructor to create LFSR with the given initial seed
    explicit FibLFSR(std::string seed);
    // Simulate one step and return the new bit as 0 or 1
    int step();
    // Simulate k steps and return a k-bit integer
    int generate(int k);

    int getSeedSize() const;

    std::string getInitialSeed() const;

    std::string getState() const;

    friend std::ostream& operator<<(std::ostream&, const FibLFSR& lfsr);
    ~FibLFSR();

 private:
    // Any fields that you need
    const std::string initialSeed;
    std::string currentState;
    char* bitCharacters;
    int* bits;
    int seedSize;
};

}  // namespace PhotoMagic

int numOfBits(int num);

#endif
```

Source file for the first supporting class file (FibLFSR.cpp):

```cpp
// Copyright 2024 Wendy Carvalho
#include "FibLFSR.hpp"
#include <cstring>

namespace PhotoMagic {
FibLFSR::FibLFSR(std::string seed) : initialSeed(seed) {
  // string seed to array?
  int temp = 0;
  int j = 0;
  int i = 0;
  seedSize = seed.length();
  // declare space for bitCharacters using length of given seed + 1 for ’\0’
  bitCharacters = new char[seedSize + 1];

  snprintf(bitCharacters, seedSize + 1, "%s", initialSeed.c_str());
  bits = new int[seedSize];
  for (i = 0; i < seedSize; i++) {
    bits[i] = (bitCharacters[i] - ’0’);
  }

  for (i = 0, j = seedSize - 1; i < j; i++, j--) {
    temp = bits[i];
    bits[i] = bits[j];
    bits[j] = temp;
  }

  for (int i = seedSize - 1; i >= 0; i--) {
    currentState += bits[i] + ’0’;
  }
  currentState += ’\0’;
}

// Simulate one step and return the new bit as 0 or 1
int FibLFSR::step() {
  int result;
  // save result of xor with 3 taps: 13, 12, and 10
  result = (((bits[seedSize - 1] ^ bits[13]) ^ bits[12]) ^ bits[10]);

  // shift array bits 1 to left (right in this case)
  for (int i = seedSize - 1; i >= 0; i--) {
    bits[i] = bits[i - 1];
  }

  // add result to end
  bits[0] = result;

  std::ostringstream tmp;
  tmp << *this;

  currentState = tmp.str();
  // return result (0 or 1)
  return result;
}

// Simulate k steps and return a k-bit integer
int FibLFSR::generate(int k) {
  int var = 0;
  // for each bit extracted, double var and add the bit returned by step()
  for (int i = 0; i < k; i++) {
    var *= 2;
    var += step();
  }

  return var;
}

int FibLFSR::getSeedSize() const { return seedSize; }
```

```
std::string FibLFSR::getInitialSeed() const { return initialSeed; }

FibLFSR::~FibLFSR() {
  delete[] bitCharacters;
  delete[] bits;
}

std::string FibLFSR::getState() const { return currentState; }

std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {
  for (int i = lfsr.seedSize - 1; i >= 0; i--) {
    out << lfsr.bits[i];
  }
  return out;
}
}  // namespace PhotoMagic
```

Header file for the second supporting class file (PhotoMagic.hpp):

```
// Copyright 2024 Wendy Carvalho

#include "FibLFSR.hpp"

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

namespace PhotoMagic {
  // transforms image using FibLFSR
  void transform(sf::Image& image, FibLFSR* seed);
}
```

Source file for the first supporting class file (PhotoMagic.cpp):

```
// Copyright 2024 Wendy Carvalho

#include "PhotoMagic.hpp"

void PhotoMagic::transform(sf::Image& image, FibLFSR* seed) {
  sf::Color p;

  // get size of image
  sf::Vector2u size = image.getSize();
  unsigned x, y;
  // use LFSR to transform the image:
  // for each pixel (x,y) in row major order [(0,0),(0,1)...]
  // extract the red, green, and blue components of the color
  // (each component is an int 0-255)
  for (x = 0; x < size.x; x++) {
    for (y = 0; y < size.y; y++) {
      p = image.getPixel(x, y);

      // xor each color component with a newly generated 8-bit integer
      // create a new color using the result of the XOR operations
      p.r ^= seed->generate(8);
      p.g ^= seed->generate(8);
      p.b ^= seed->generate(8);

      // set the pixel in the new picture to that color
      image.setPixel(x, y, p);
    }
  }
}
```

Boost test cases (test.cpp):

```
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly
// test.cpp for PS1a
// updated 1/8/2024
// Copyright 2024 Wendy Carvalho

#include <iostream>
#include <string>

#include "FibLFSR.hpp"
#include "PhotoMagic.hpp"

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

using PhotoMagic::FibLFSR;

BOOST_AUTO_TEST_CASE(testStepInstr) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
}

BOOST_AUTO_TEST_CASE(testGenerateInstr) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_EQUAL(l.generate(9), 51);
}

BOOST_AUTO_TEST_CASE(testGenerate7Steps) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_EQUAL(l.generate(5), 3);
  BOOST_REQUIRE_EQUAL(l.generate(5), 6);
  BOOST_REQUIRE_EQUAL(l.generate(5), 14);
  BOOST_REQUIRE_EQUAL(l.generate(5), 24);
  BOOST_REQUIRE_EQUAL(l.generate(5), 1);
  BOOST_REQUIRE_EQUAL(l.generate(5), 13);
  BOOST_REQUIRE_EQUAL(l.generate(5), 28);
}

// test operator <<
BOOST_AUTO_TEST_CASE(testOstream) {
  FibLFSR l("1011011000110110");

  std::ostringstream tmp;
  tmp << l;

  std::string strTest = tmp.str();

  BOOST_REQUIRE_EQUAL(strTest, "1011011000110110");
}

// test that the result of # n of step() calls equals generate(n)
BOOST_AUTO_TEST_CASE(testStepEqualsGenerate) {
  FibLFSR l1("1011011000110110");
  FibLFSR l2("1011011000110110");

  for (int i = 0; i < 5; i++) {
    l1.step();
  }

  l2.generate(5);

  std::ostringstream tmp, tmp2;
  tmp << l1;
  tmp2 << l2;

  std::string strStep = tmp.str();
  std::string strGenerate = tmp2.str();

  BOOST_REQUIRE_EQUAL(strStep, strGenerate);
}

BOOST_AUTO_TEST_CASE(testGenerateKBits) {
  FibLFSR l("1011011000110110");
  // array declared w/ max num of bits
  int *binary = new int[16];
  int n = l.generate(10);
  int i;
  for (i = 0; n > 0; i++) {
    binary[0] = n % 2;
    n /= 2;
  }
  delete[] binary;

  BOOST_REQUIRE_LT(i, 10);
}

BOOST_AUTO_TEST_CASE(testDifferentSeed) {
  FibLFSR l2("0011001010000011");
  BOOST_REQUIRE_EQUAL(l2.step(), 0);
  BOOST_REQUIRE_EQUAL(l2.step(), 0);
  BOOST_REQUIRE_EQUAL(l2.step(), 1);
}

BOOST_AUTO_TEST_CASE(testTransform) {
  FibLFSR seed("1011011000110110");

  sf::Image image;
  sf::Color p1, p2;
  // check that images open (later rplc with string)
  if (!image.loadFromFile("cat.jpg")) {
    std::cerr << "failed to open the image" << std::endl;
  }

  p1 = image.getPixel(0, 0);

  transform(image, &seed);
  p2 = image.getPixel(0, 0);

  // check if rgb values before and after are diff
  BOOST_CHECK_NE(p1.r, p2.r);
  BOOST_CHECK_NE(p1.g, p2.g);
  BOOST_CHECK_NE(p1.b, p2.b);
}
```

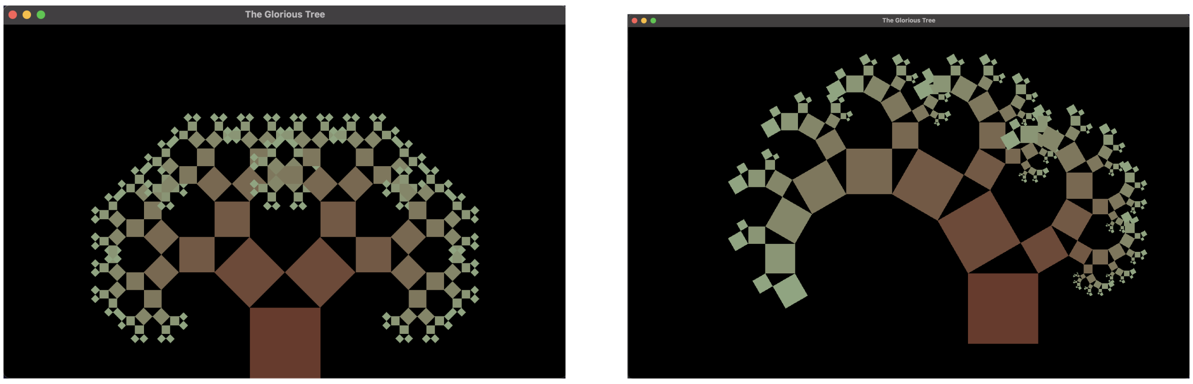
3 PS2: Pythagoras Tree

3.1 Discussion

This project aims to demonstrate recursive graphs by plotting a Pythagoras Tree from a given length of the side of a base square, the depth of the recursion, and the angle of the inner triangle to be produced by the recursion of squares. This Pythagoras tree begins with the aforementioned base square and generates two additional squares at each of its corners of potentially different sizes if an angle besides 45º is given. Depending on the number for the recursion depth, each new square generates two more squares in the same manner. If an angle of 45º is chosen, which is the default angle, the picture produced resembles a tree. For this project, I worked with another student.

3.2 What I accomplished

This program can successfully produce a Pythagoras Tree or spiral given a length for the base square, the angle, and the depth of recursion. The graph also has color. This outcome can be seen below in Figure 3, for both the tree and the spiral.



(a) Tree

(b) Spiral

Figure 3: Pythagoras Graphs

3.3 Design Decisions and Implementations

To achieve the Pythagoras tree, we used the `sf::RectangleShapes` class from SFML. The base parent square is passed in through a function called `PTreeHelper`, which creates a copy of the `sf::RectangleShape` parent, resizes it, and accumulates its transformations (rotations and positions along the x and y axes). `getTransformation().transformationPoint()` was essential for this and made it easier to find the location of the parent square on the grid, using the top left vertex for the children on the left side, and the top right vertex for the children on the right side. Each time the recursive function was called, the square would get rotated again by the angle determined previously by the command line.

For the extra credit, we added a color gradient to the tree. `pTree()` initializes the first parent square to a brown color. Then, each time the recursive function gets called, the child squares' colors have color components added to them (6 for the red component, and 15 for the green component, and 12 for the blue component). I also made it so the project can use different angles rather than the default 45º to compose the `PTree` by using cosine for the left children and sine for the right children, rather than using the same value for both. This way, the children are different sizes depending on what side they originate from.

3.4 What I already knew

At this point, I was already familiar with creating an SFML window and generating shapes from prior lectures in class that involved drawing snowflakes.

3.5 What I learned

I learned how to track the accumulation of transformations a shape has undergone through `getTransformation().transformPoint()`. I also learned more about how an SFML grid works, which is not intuitive since the coordinate (0,0) is in the top left corner rather than the center. I also learned about color operations.

3.6 Challenges

It took a lot of research to figure out that `transformPoint()` would be a lifesaver to start the new child from where the parent was, make it smaller, and rotate it, rather than gradually moving the child along the x and y axes. We realized that this would not work the larger the depth got, as the tree would start to rotate inward instead of outward.

Another challenge I faced was making sure the squares would rotate and be positioned in the expected area. This was confusing but `setOrigin()` was crucial to resolve this.

We also initially thought that for the left-hand children, it would be appropriate to rotate by a positive angel, but this proved to be wrong and had to be negative, and vice versa.

Lastly, resizing the window was complex, but this was resolved by doing the resizing according to the magnitude of the angle.

3.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
INCLUDEDIR = -I/opt/homebrew/Cellar/sfml/2.6.1/include
LIBDIR = -L/opt/homebrew/Cellar/sfml/2.6.1/lib
# Your .hpp files
DEPS = PTree.hpp
# Your compiled .o files
OBJECTS = PTree.o
# The name of your program
PROGRAM = PTree

TEST = test

.PHONY: all clean lint

all: $(PROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $< $(INCLUDEDIR)

$(PROGRAM): $(OBJECTS) main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

Main routine (main.cpp):

```
// Copyright 2024 Wendy Carvalho and Meriem Elkoudi

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

int main() {
  // length of base
  double L;
  double A;
  // depth of recursion
  int N;
  std::cin >> L >> N >> A;

  // what does it mean to make sure default is 45?

  int width = L * 8;
  int height = L * 5;
  // int size = 600;
  //  Create the main window
  sf::RenderWindow window(sf::VideoMode(width, height), "The Glorious Tree")
   ;

  // Start the game loop
  while (window.isOpen()) {
    // Process events
    sf::Event event;
    while (window.pollEvent(event)) {
      // Close window: exit
      if (event.type == sf::Event::Closed) window.close();
    }
    // Clear screen
    window.clear(sf::Color::Black);
    if (A == 45) {
      pTree(window, L, N, sf::Vector2f(width / 2, height / 2 + L / 2), A);
    } else if (A < 45) {
      pTree(window, L, N, sf::Vector2f(width - width / 3, height / 2), A);
      window.setSize(sf::Vector2u(width + width / 2, height + height / 2));
    } else if (A > 45) {
      pTree(window, L, N, sf::Vector2f(width / 3, height / 2 + L / 2), A);
      window.setSize(sf::Vector2u(width + width / 2, height + height / 2));
    }
    window.display();
  }
  return EXIT_SUCCESS;
}
```

Header file for the first supporting class file (PTree.hpp):

```
// Copyright 2024 Wendy Carvalho and Meriem Elkoudi

#pragma once
#include <SFML/Graphics.hpp>

void pTreeHelper(sf::RenderTarget& window, int N, sf::RectangleShape parent,
    double angle);
void pTree(sf::RenderTarget& window, double L, int N, sf::Vector2f origin,
   double angle);
```

Source file for the first supporting class file (PTree.cpp):

```
// Copyright 2024 Wendy Carvalho and Meriem Elkoudi

#include "PTree.hpp"

#include <cmath>
#include <iostream>

const double DEG_PER_RAD = 180 / M_PI;
```

```

void pTreeHelper(sf::RenderTarget& window, int N, sf::RectangleShape parent,
                 double angle) {
  window.draw(parent);

  if (N < 1) return;

  // new length will be parent's length * cos(45)
  // left length first
  auto newLength = parent.getSize().x * cos(45 / DEG_PER_RAD);
  auto leftLength = parent.getSize().x * cos(angle / DEG_PER_RAD);
  auto rightLength = parent.getSize().x * sin(angle / DEG_PER_RAD);
  sf::Vector2f leftCLength(leftLength, leftLength);
  sf::Vector2f rightCLength(rightLength, rightLength);
  sf::Vector2f childLength(newLength, newLength);

  // left children
  sf::RectangleShape leftChild = parent;
  // new length is parent's length * cos 45
  leftChild.setSize(leftCLength);
  leftChild.setOrigin(0, leftChild.getSize().y);
  // set new child's position based on parent's previous transformations
  // set to top left vertex
  leftChild.setPosition(parent.getTransform().transformPoint({0, 0}));
  leftChild.rotate(-angle);
  // color manipulation (adding only green and blue component for "leaves")
  sf::Color newColor = leftChild.getFillColor();
  newColor += sf::Color(6, 15, 12);
  leftChild.setFillColor(newColor);
  pTreeHelper(window, N - 1, leftChild, angle);

  // right children
  sf::RectangleShape rightChild = parent;
  rightChild.setSize(rightCLength);
  rightChild.setOrigin(rightChild.getSize());

  // set new child's position based on parent's previous transformations
  // set to top right vertex
  rightChild.setPosition(
      parent.getTransform().transformPoint({parent.getSize().x, 0}));
  rightChild.rotate(90 - angle);
  // right side color manipulation
  rightChild.setFillColor(newColor);
  pTreeHelper(window, N - 1, rightChild, angle);
}

void pTree(sf::RenderTarget& window, double L, int N, sf::Vector2f origin,
           double angle) {
  sf::RectangleShape square(sf::Vector2f(L, L));
  sf::Vector2f pt = origin;
  square.setOrigin(L / 2, L / 2);
  pt.y += (3 * L) / 2;

  if (L > 200) {
    double scale = (1 / L) * 200;
    square.setScale(scale, scale);
    // centering on x axis and keeping at bottom of y axis
    pt.y = window.getSize().y - L / 4;
    pt.x = window.getSize().x / 2.f;
  }

  square.setPosition(pt);

  square.setFillColor(sf::Color(0x663b2dff));
  pTreeHelper(window, N, square, angle);
}
```

4 PS3: Sokoban

4.1 Discussion

This project, as its name implies, seeks to recreate Sokoban, a classic puzzle game where the player must strategically push boxes onto designated storage locations within a maze-like environment. With the objective of this project directly being a game, it allowed me to dive into game mechanics even more than PS0 did.

4.2 What I accomplished

Using the SFML framework, the program can render the game board onto a graphical window. Textures and sprites are utilized to visually represent each type of tile, including walls, ground spaces, boxes, storage locations, and the player character.

The player can navigate the game board by pressing arrow keys or WASD, interacting with boxes, and navigating around obstacles like walls. One of the core mechanics of this game is the ability to push boxes around the board. When the player character moves into a position adjacent to a box, they can push the box in the direction of their movement, provided that there is space for the box to occupy.

As the player navigates the game board and pushes boxes onto designated storage locations, the program continuously checks whether all boxes have been successfully placed in their respective storage locations. Once all boxes are in place, the level is considered completed, and the player wins the game.

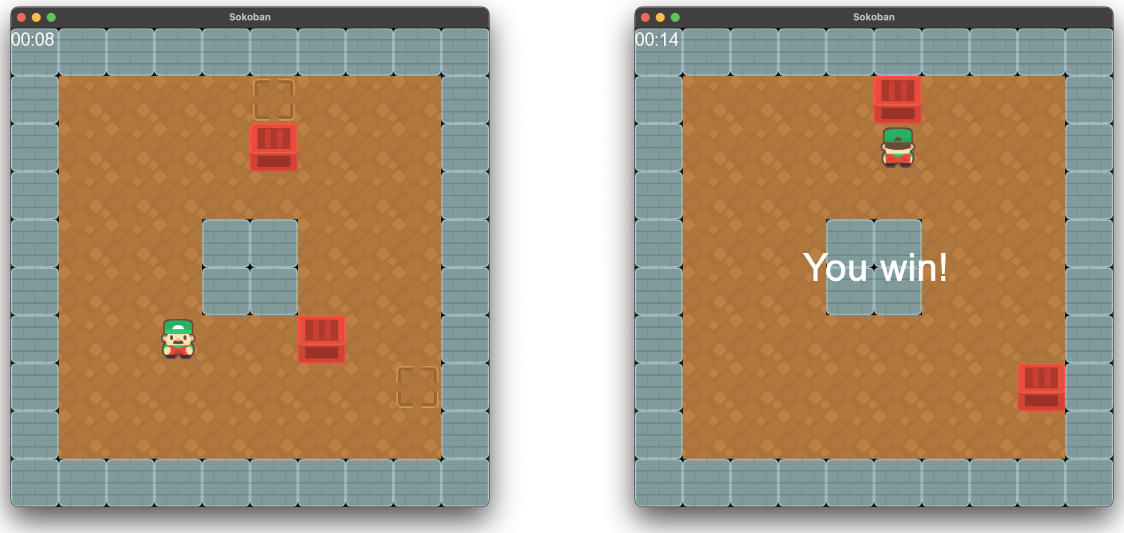


Figure 4: Sokoban gameplay

4.3 Design Decisions and Implementations

I set this program up so that when reading in the file content of each level it is stored in a string. I did this initially to get rid of any new line characters since I used `getline()`.

Then this string was used to create grids which are stored inside a class called `Sokoban`. The program iterates through the string and each immovable tile (wall, ground, storage location) is stored into a vector of vectors of chars called `staticTiles`. I did this because even though there will be boxes or a player on top of one of these tiles, they should still be in the background and stored separately. In those same iterations, if a box character was read, its location was stored as a `sf::Vector2i` (a vector of `sf::Vector2is` in the case of the boxes, since there may be many). I created a simple `sf::Vector2f` for the player's initial position. I wanted these objects of different categories to be stored separately. I made the player and the boxes into vectors because they can and will be moved throughout the gameplay, so it is easier to set up their initial positions this way and manipulate them throughout gameplay since vectors can resize dynamically without the need for manual memory allocation.

Several textures are saved within the program for each different tile and player direction, but only 3 sprites are used to decrease the amount of memory used: one for the static tiles, one for the boxes, and one for the player.

Also, an `enum` type for `Directions` was created to be passed in as the argument for the `moveplayer()` function. Using event handling inside the main routine, the direction taken in by the keyboard (using arrow keys or WASD) is passed into `movePlayer()`, and this function handles moving the player and boxes if that applies, and determines whether or not these are possible (if there are no obstacles in the way). This function updates the player's location stored inside the `Sokoban` class, and so when the draw function is called again inside the main routine, draw updates the player's location visually on the window.

In `isWon()`, two lambda functions are employed to determine if the game has been won. The first lambda `isBoxAtTarget`, checks if boxes are placed at designated target locations on the game grid, using the `std::all_of` algorithm to verify if this condition is met for all boxes in the vector. The game is also considered won under a second scenario, where the number of boxes exceeds the number of storage locations but every storage location is occupied by a box. This is handled by another lambda, `isStorageFull`, which iterates over the game grid's tiles to ensure every storage location is filled with a box. The `isWon()` function returns true if either condition is satisfied—either all boxes are at target locations or all storage locations are filled, despite an excess of boxes.

I did a few different things for extra credit. One of them was making the character change its texture to face the direction it's moving in by loading a single texture in the `Sokoban` class whenever a move button is pressed. I also added a victory sound when the game is won by using `sf::Audio` in SFML, and I added music playing in the background using `sf::Music`, which stops playing when the game is won, as the victory sound plays instead. Additionally, I implemented functionality to reset the game state, allowing the player to restart the current level if they wish to try again or if they become stuck. Similarly, I added an undo button using the key z by adding it to the event handling I used for the reset feature. The undo button calls `undo()` which can undo the last move done by the player, and a box move if that was done.

4.4 What I already knew

I was quite familiar by then with the SFML library and its functionality for keyboard events and moving sprites from previous projects. I also knew how to create and manipulate vectors from Computing III and the lectures in Computing IV.

4.5 What I learned

I learned about how to use `sf::Time` and `sf::Clock` to track the passage of time to create the timer that displays on the screen during gameplay.

This program also forced me to learn how to use breakpoints in VSCode and step through the code, something I was only familiar with in XCode.

Lastly, I learned how to avoid segmentation faults by making sure the sprites stay within the window.

4.6 Challenges

I had trouble understanding how to implement a vector of vectors, which was challenging to visualize, but made sense once I thought about it as a matrix. Also, successfully centering the victory message on the screen was difficult, but this was only due to SFML's API. Lastly, when the game is won, the victory music starts playing but it lags the window a bit and the "You win!" text takes an extra second or two to show up.

4.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
```

```
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.84.0/include -I/opt/homebrew/Cellar/sfml/2.6.1/include
LIBDIR = -L/opt/homebrew/Cellar/boost/1.84.0/lib -L/opt/homebrew/Cellar/sfml/2.6.1/lib
# Your .hpp files
DEPS = Sokoban.hpp
# Your compiled .o files
OBJECTS = Sokoban.o
# The name of your program
PROGRAM = Sokoban

.PHONY: all clean lint

all: $(PROGRAM) $(PROGRAM).a test

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

$(PROGRAM): $(OBJECTS) main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

$(PROGRAM).a: $(OBJECTS)
    ar rcs $@ $^

test: $(OBJECTS) test.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

clean:
    rm *.o $(PROGRAM) test

lint:
    cpplint *.cpp *.hpp
```

Main routine (main.cpp):

```cpp
// Copyright 2024  Wendy Carvalho
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>

#include "Sokoban.hpp"

int main(int argc, char *argv[]) {
  std::ifstream lvlFile;

  SB::Sokoban game;
  lvlFile.open(argv[1]);
  lvlFile >> game;

  sf::Clock clock;
  sf::Text timeText, final, victoryText;
  std::string str;
  sf::Time timeElapsed;
  int seconds, minutes = 0;

  size_t width = game.width() * game.TILE_SIZE;
  size_t height = game.height() * game.TILE_SIZE;

  sf::RenderWindow window(sf::VideoMode(width, height), "Sokoban");
  window.setPosition({0, 0});

  sf::Font font;
```

```cpp
font.loadFromFile("Arial.ttf");

sf::Music music;
if (!music.openFromFile("bgmusic.wav")) {
  return -1;
}
music.play();

while (window.isOpen()) {
  sf::Event event;

  if (!game.isWon()) {
    timeElapsed = clock.getElapsedTime();
    seconds = static_cast<int>(timeElapsed.asSeconds());
    str = "Time: " + std::to_string(minutes) + ":" + std::to_string(seconds);

    if (seconds < 10 && minutes < 10) {
      str =
          "0" + std::to_string(minutes) + ":" + "0" + std::to_string(seconds);
    } else if (seconds >= 10 && minutes < 10) {
      str = "0" + std::to_string(minutes) + ":" + std::to_string(seconds);
    } else if (seconds < 10 && minutes >= 10) {
      str = std::to_string(minutes) + ":" + "0" + std::to_string(seconds);
    } else {
      str = std::to_string(minutes) + ":" + std::to_string(seconds);
    }
    if (seconds == 60) {
      minutes++;
      clock.restart();
    }
  }
  timeText.setFont(font);
  timeText.setString(str);
  timeText.setCharacterSize(24);
  timeText.setFillColor(sf::Color::White);

  while (window.pollEvent(event)) {
    if (event.type == sf::Event::Closed) {
      window.close();
    } else if (event.type == sf::Event::KeyPressed) {
      if (!game.isWon()) {
        switch (event.key.code) {
          case sf::Keyboard::W:
            game.movePlayer(SB::Up);
            break;
          case sf::Keyboard::A:
            game.movePlayer(SB::Left);
            break;
          case sf::Keyboard::S:
            game.movePlayer(SB::Down);
            break;
          case sf::Keyboard::D:
            game.movePlayer(SB::Right);
            break;
          case sf::Keyboard::Up:
            game.movePlayer(SB::Up);
            break;
          case sf::Keyboard::Left:
            game.movePlayer(SB::Left);
            break;
          case sf::Keyboard::Down:
            game.movePlayer(SB::Down);
            break;
          case sf::Keyboard::Right:
            game.movePlayer(SB::Right);
            break;
          default:
            break;
        }
      }
      switch (event.key.code) {
        case sf::Keyboard::R:
          lvlFile.clear();
          lvlFile.seekg(0);
          game.resetBoxes();
          lvlFile >> game;
          clock.restart();
          minutes = seconds = 0;
          break;
        case sf::Keyboard::Z:
          game.undo();
        default:
          break;
      }
    }
  }

  window.clear();
  window.draw(game);

  if (game.isWon()) {
    // victory!!
    std::string finalTime = str;
    final = timeText;
    final.setString(finalTime);
    victoryText.setFont(font);
    victoryText.setString("You win!");
    victoryText.setCharacterSize(50);
    sf::FloatRect textRect = victoryText.getLocalBounds();
    victoryText.setFillColor(sf::Color::White);
    victoryText.setStyle(sf::Text::Bold);
    victoryText.setOrigin(textRect.left + textRect.width / 2.0f,
                          textRect.top + textRect.height / 2.0f);
    victoryText.setPosition(width / 2.0f, height / 2.0f);

    music.stop();
    if (!game.soundIsPlaying()) {
      sf::SoundBuffer buffer;
      if (!buffer.loadFromFile("victory.wav")) return -1;
      sf::Sound sound;
      sound.setBuffer(buffer);
      sound.play();
      game.setPlayingTrue();
      while (sound.getStatus() == sf::Sound::Playing) {
      }
    }

    window.draw(victoryText);
    window.draw(final);
  } else {
    window.draw(timeText);
  }
  window.display();
}
}
```

Header file for the first supporting class file (Sokoban.hpp):

```
// Copyright 2024 Wendy Carvalho
#pragma once

#include <fstream>
#include <iostream>
#include <vector>
#include <SFML/Graphics.hpp>

namespace SB {
enum Direction {
  Up = sf::Keyboard::Up,
  Down = sf::Keyboard::Down,
  Left = sf::Keyboard::Left,
  Right = sf::Keyboard::Right,
};

class Sokoban : public sf::Drawable {
 public:
  static const size_t TILE_SIZE = 64;
  Sokoban();
  explicit Sokoban(std::string filename);
  size_t width() const { return _width; }
  size_t height() const { return _height; }

  sf::Vector2i playerLoc() const;  // returns player's current position

  void movePlayer(Direction dir);
  void resetBoxes() { boxes.clear(); }
  void undo();

  bool isWon() const;

  bool soundIsPlaying() { return victoryMusic; }

  void setPlayingTrue() { victoryMusic = true; }

  // a >> operator that reads the level from a stream
  friend std::istream& operator>>(std::ifstream& input, Sokoban& game);
  // << operator that writes the level back to a stream
  friend std::ostream& operator<<(std::ostream& output, Sokoban& game);

 protected:
  void draw(sf::RenderTarget& target, sf::RenderStates states) const
    override;

 private:
  int _height;
  int _width;

  bool victoryMusic;

  sf::Texture _wall;
  sf::Texture _box;
  sf::Texture _empty;
  sf::Texture _storage;
  sf::Texture _player;

  char* origLvl;

  std::vector<std::vector<char>> staticTiles;
  std::vector<sf::Vector2i> boxes;
  sf::Vector2i playerLocation;
  std::vector<std::pair<bool, sf::Vector2i>> previousPLoc;
  std::vector<std::pair<int, sf::Vector2i>> previousBLoc;
};

};  // namespace SB
```

Source file for the first supporting class file (Sokoban.cpp):

```
// Copyright 2024 Wendy Carvalho

#include "Sokoban.hpp"

#include <fstream>
#include <map>
#include <sstream>
#include <string>
#include <vector>

namespace SB {
Sokoban::Sokoban() {
  _wall.loadFromFile("block_06.png");
  _box.loadFromFile("crate_03.png");
  _empty.loadFromFile("ground_01.png");
  _storage.loadFromFile("ground_04.png");
  _player.loadFromFile("player_05.png");
}

Sokoban::Sokoban(std::string filename) {
  std::ifstream file;
  file.open(filename);

  _wall.loadFromFile("block_06.png");
  _box.loadFromFile("crate_03.png");
  _empty.loadFromFile("ground_01.png");
  _storage.loadFromFile("ground_04.png");
  _player.loadFromFile("player_05.png");

  file >> *this;
}

// a >> operator that reads the level from a stream
std::istream& operator>>(std::ifstream& input, Sokoban& game) {
  std::vector<std::vector<char>> staticTiles;
  // first line is 2 ints, h and w
  input >> game._height >> game._width;

  std::string line;
  input.ignore();
```

```cpp

  game.origLvl = new char[game._height * game._width];
  // read in the rest of the level content and store inside const * char
  int i;
  for (i = 0; i < game._height; i++) {
    std::getline(input, line);
    for (int j = 0; j < game._width; j++) {
      game.origLvl[i * game._width + j] = line[j];
    }
  }
  // traverse through string, store static tiles in one vector
  // store boxes in sf::Vector2i vector
  // game.boxes.clear();
  char tile;
  for (i = 0; i < game._height; i++) {
    std::vector<char> row;
    for (int j = 0; j < game._width; j++) {
      tile = game.origLvl[i * game._width + j];
      if (tile == '#' || tile == '.' || tile == 'a') {
        row.push_back(tile);
      } else if (tile == '1') {
        row.push_back('a');
        game.boxes.push_back({static_cast<int>(j), static_cast<int>(i)});
      } else if (tile == 'A') {
        row.push_back('.');
        game.boxes.push_back({static_cast<int>(j), static_cast<int>(i)});
      } else if (tile == '@') {
        row.push_back('.');
        game.playerLocation = {static_cast<int>(j), static_cast<int>(i)};
      }
    }
    game.staticTiles.push_back(row);
  }
  delete[] game.origLvl;
  return input;
}
// << operator that writes the level back to a stream
std::ostream& operator<<(std::ostream& output, Sokoban& game) {
  for (int j = 0; j < game._width * game._height; j++) {
    if (j % 10 == 0) {
      std::cout << std::endl;
    }
    std::cout << game.origLvl[j];
  }
  return output;
}

bool Sokoban::isWon() const {
  // // if all boxes are at "a" locations
  // for (int i = 0; i < boxes.size(); i++) {
  //   int x, y;
  //   x = boxes[i].x;
  //   y = boxes[i].y;
  //     if (staticTiles[y][x] != 'a') {
  //       return false;
  //     }
  // }
  auto isBoxAtTarget = [this](const sf::Vector2i& box) {
    int x = box.x;
    int y = box.y;
    return staticTiles[y][x] == 'a';
  };

  auto isStorageFull = [this]() {
    for (size_t yj = 0; yj < staticTiles.size(); yj++) {
      for (size_t xi = 0; xi < staticTiles[yj].size(); xi++) {
        // target
        if (staticTiles[yj][xi] == 'a') {
          bool foundBox = false;
          for (const auto& box : boxes) {
            size_t x = box.x;
            size_t y = box.y;
            if (x == xi && y == yj) {
              foundBox = true;
              break;
            }
          }
          if (!foundBox) {
            return false;  // at least one storage is not filled
          }
        }
      }
    }
    return true;  // all storage is filled
  };
  // check if all boxes are at "a" locations
  return std::all_of(boxes.begin(), boxes.end(), isBoxAtTarget) ||
         isStorageFull();
  // return true;
}

void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
  // draw walls for every tile location
  // need to update later to make grid properly
  for (int y = 0; y < _height; y++) {
    for (int x = 0; x < _width; x++) {
      sf::Sprite tile;
      sf::Sprite box;
      sf::Vector2f tmp;
      sf::Sprite player;
      // check game grid
      if (staticTiles[y][x] == '#') {
        tile.setTexture(_wall);
      } else if (staticTiles[y][x] == '.') {
        tile.setTexture(_empty);
      } else if (staticTiles[y][x] == 'a') {
        tile.setTexture(_storage);
      }
      // while boxes vector is not empty
      for (auto& b : boxes) {
        box.setTexture(_box);
        tmp = sf::Vector2f(b);
        tmp.x *= TILE_SIZE;
        tmp.y *= TILE_SIZE;
        box.setPosition(tmp);
        target.draw(box, states);
      }

      player.setTexture(_player);
      tmp = sf::Vector2f(playerLocation);
      tmp.x *= TILE_SIZE;
      tmp.y *= TILE_SIZE;

      // if position is equal to a wall, don’t move
      player.setPosition(tmp);
      target.draw(player, states);

      tile.setPosition(x * TILE_SIZE, y * TILE_SIZE);
      target.draw(tile, states);
    }
  }
}

// implement playerLoc()
sf::Vector2i Sokoban::playerLoc() const { return playerLocation; }

void Sokoban::movePlayer(Direction dir) {
  // up
  if (dir == Up) {
    _player.loadFromFile("player_08.png");
    sf::Vector2i currPos = {playerLocation.x, playerLocation.y};
    sf::Vector2i newPos = {playerLocation.x, playerLocation.y - 1};
    sf::Vector2i nextTo = {playerLocation.x, playerLocation.y - 2};
    if (playerLocation.y != 0) {
      if (staticTiles[playerLocation.y - 1][playerLocation.x] != ’#’) {
        previousPLoc.push_back(std::make_pair(false, playerLocation));
        playerLocation.y = playerLocation.y - 1;
      }
    }
    for (size_t i = 0; i < boxes.size(); i++) {
      if (boxes[i] == newPos) {
        if ((staticTiles[newPos.y - 1][newPos.x] != ’#’ && newPos.y != 0) &&
            (std::find(boxes.begin(), boxes.end(), nextTo) == boxes.end()))
   {
          // previousPLoc.push_back(playerLocation);
          previousBLoc.push_back(std::make_pair(i, boxes[i]));
          boxes[i] = {newPos.x, newPos.y - 1};
          previousPLoc.push_back(std::make_pair(true, currPos));

          playerLocation = newPos;
        } else {
          playerLocation = currPos;
          previousPLoc.pop_back();
        }
      }
    }
  }
  // down
  if (dir == Down) {
    _player.loadFromFile("player_05.png");
    sf::Vector2i currPos = {playerLocation.x, playerLocation.y};
    sf::Vector2i newPos = {playerLocation.x, playerLocation.y + 1};
    sf::Vector2i nextTo = {playerLocation.x, playerLocation.y + 2};
    if (playerLocation.y != _height - 1) {
      if (staticTiles[playerLocation.y + 1][playerLocation.x] != ’#’) {
        previousPLoc.push_back(std::make_pair(false, playerLocation));
        playerLocation.y = playerLocation.y + 1;
      }
    }

    for (size_t i = 0; i < boxes.size(); i++) {
      if (boxes[i] == newPos) {
        if ((staticTiles[newPos.y + 1][newPos.x] != '#' &&
             newPos.y + 1 < _height) &&
            (std::find(boxes.begin(), boxes.end(), nextTo) == boxes.end()))
   {
          previousBLoc.push_back(std::make_pair(i, boxes[i]));
          boxes[i] = {newPos.x, newPos.y + 1};
          previousPLoc.push_back(std::make_pair(true, currPos));

          playerLocation = newPos;
        } else {
          playerLocation = currPos;
        }
      }
    }
  }

  // left
  if (dir == Left) {
    _player.loadFromFile("player_20.png");
    sf::Vector2i currPos = {playerLocation.x, playerLocation.y};
    sf::Vector2i newPos = {playerLocation.x - 1, playerLocation.y};
    sf::Vector2i nextTo = {playerLocation.x - 2, playerLocation.y};
    if (staticTiles[newPos.y][newPos.x] != '#' && playerLocation.x - 1 >= 0)
    {
      previousPLoc.push_back(std::make_pair(false, playerLocation));
      playerLocation = newPos;
    }

    for (size_t i = 0; i < boxes.size(); i++) {
      if (boxes[i] == newPos) {
        if ((staticTiles[newPos.y][newPos.x - 1] != '#' && newPos.x != 0) &&
            (std::find(boxes.begin(), boxes.end(), nextTo) == boxes.end()))
   {
          previousBLoc.push_back(std::make_pair(i, boxes[i]));
          boxes[i] = {newPos.x - 1, newPos.y};
          previousPLoc.push_back(std::make_pair(true, currPos));

          playerLocation = newPos;

        } else {
          playerLocation = currPos;
        }
      }
    }
  }
  // right
  if (dir == Right) {
    _player.loadFromFile("player_17.png");
    sf::Vector2i currPos = {playerLocation.x, playerLocation.y};
    sf::Vector2i newPos = {playerLocation.x + 1, playerLocation.y};
    sf::Vector2i nextTo = {playerLocation.x + 2, playerLocation.y};
    if (staticTiles[newPos.y][newPos.x] != '#' &&
        playerLocation.x + 1 != _width) {
```

```
      previousPLoc.push_back(std::make_pair(false, playerLocation));
      playerLocation = newPos;
    }

    for (size_t i = 0; i < boxes.size(); i++) {
      if (boxes[i] == newPos) {
        if ((staticTiles[newPos.y][newPos.x + 1] != '#' &&
             newPos.x + 1 != _width) &&
            (std::find(boxes.begin(), boxes.end(), nextTo) == boxes.end()))
   {
          previousBLoc.push_back(std::make_pair(i, boxes[i]));
          boxes[i] = {newPos.x + 1, newPos.y};
          previousPLoc.push_back(std::make_pair(true, currPos));

          playerLocation = newPos;
        } else {
          playerLocation = currPos;
        }
      }
    }
  }
}

void Sokoban::undo() {
  if (!previousPLoc.empty()) {
    auto lastElement = previousPLoc.back();
    if (lastElement.first == true) {
      playerLocation = lastElement.second;
      previousPLoc.pop_back();
      auto lastElement = previousBLoc.back();
      boxes[lastElement.first] = lastElement.second;
      previousBLoc.pop_back();
    } else {
      playerLocation = lastElement.second;
      previousPLoc.pop_back();
    }
  }
}
};  // namespace SB
```

Boost test cases (test.cpp):

```
// Copyright 2024 Wendy Carvalho

#include <fstream>
#include <iostream>
#include <string>

#include "Sokoban.hpp"

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

using SB::Sokoban;

// check <<
BOOST_AUTO_TEST_CASE(testInsertion) {
  std::ifstream lvlFile;
  SB::Sokoban s;
  lvlFile.open("level1.lvl");
  lvlFile >> s;
  BOOST_CHECK_EQUAL(s.height(), 10);
}

BOOST_AUTO_TEST_CASE(testBoxWall) {
  std::ifstream lvlFile;
  SB::Sokoban s;
  lvlFile.open("level1.lvl");
  lvlFile >> s;

  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Left);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);

  sf::Vector2u currLoc = static_cast<sf::Vector2u>(s.playerLoc());
  sf::Vector2u tmp = {5, 2};

  BOOST_CHECK_EQUAL(currLoc.x, tmp.x);
  BOOST_CHECK_EQUAL(currLoc.y, tmp.y);
}


BOOST_AUTO_TEST_CASE(testBoxBox) {
  std::ifstream lvlFile;
  SB::Sokoban s;
  lvlFile.open("level2.lvl");
  lvlFile >> s;

  s.movePlayer(SB::Direction::Up);

  sf::Vector2u currLoc = static_cast<sf::Vector2u>(s.playerLoc());
  sf::Vector2u tmp = {8, 5};

  BOOST_CHECK_EQUAL(currLoc.x, tmp.x);
  BOOST_CHECK_EQUAL(currLoc.y, tmp.y);
}

BOOST_AUTO_TEST_CASE(testLotsOfBoxes) {
  std::ifstream lvlFile;
  SB::Sokoban s;
  lvlFile.open("level5.lvl");
  lvlFile >> s;

  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Up);

  BOOST_CHECK_EQUAL(s.isWon(), true);
}

BOOST_AUTO_TEST_CASE(testLotsOfTargets) {
  std::ifstream lvlFile;
  SB::Sokoban s;
  lvlFile.open("level6.lvl");
  lvlFile >> s;
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Left);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Up);
  s.movePlayer(SB::Direction::Right);
  s.movePlayer(SB::Direction::Down);

  BOOST_CHECK_EQUAL(s.isWon(), true);
}

BOOST_AUTO_TEST_CASE(testSymbol) {
  std::ifstream lvlFile;
  SB::Sokoban s;
  lvlFile.open("check1.lvl");

  lvlFile >> s;
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Left);
  s.movePlayer(SB::Direction::Left);
  // BOOST_CHECK_EQUAL(s.isWon(), true);

  sf::Vector2u currLoc = static_cast<sf::Vector2u>(s.playerLoc());
  sf::Vector2u tmp = {1, 3};

  BOOST_CHECK_EQUAL(currLoc.x, tmp.x);
  BOOST_CHECK_EQUAL(currLoc.y, tmp.y);
}

BOOST_AUTO_TEST_CASE(testMoveOffScreen) {
  SB::Sokoban s("level4.lvl");
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Left);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);
  s.movePlayer(SB::Direction::Down);

  sf::Vector2u currLoc = static_cast<sf::Vector2u>(s.playerLoc());
  sf::Vector2u tmp = {5, 7};

  BOOST_CHECK_EQUAL(currLoc.x, tmp.x);
  BOOST_CHECK_EQUAL(currLoc.y, tmp.y);
}
```

5 PS4: NBody Simulator

5.1 Discussion

This project consists of an animated simulation of n particles in motion which are mutually affected by gravitational forces using mathematical approximations. This is based on Newton's 1687 theory regarding the motion of two particles under the influence of their mutual gravitational attraction and the subsequent n-body problem, where n is greater than 2.

5.2 What I accomplished

I successfully used data supplied by text input files that provide information on the number of particles in a problem, the hypothetical radius of the universe, and data on each particle about its mass, initial velocity, and initial coordinates on the xy plane. With this data, I used the SFML library to simulate the new locations and velocities of each particle after undergoing mathematical calculations involving the equation of universal gravitation. The program performs many calculations with each frame in accordance with the passage of time, in an amount of seconds provided in the command line.

These particles, or celestial bodies, are animated to show, for example, the rotation of each planet around the sun given a certain amount of seconds until it reaches a larger amount of time, both given by the command line. An example output can be seen in Figure 5. This is done by calculating the net force that each particle has on each other to then calculate each of their new positions and velocities.

I also created my own text input file with the data for a hypothetical universe.

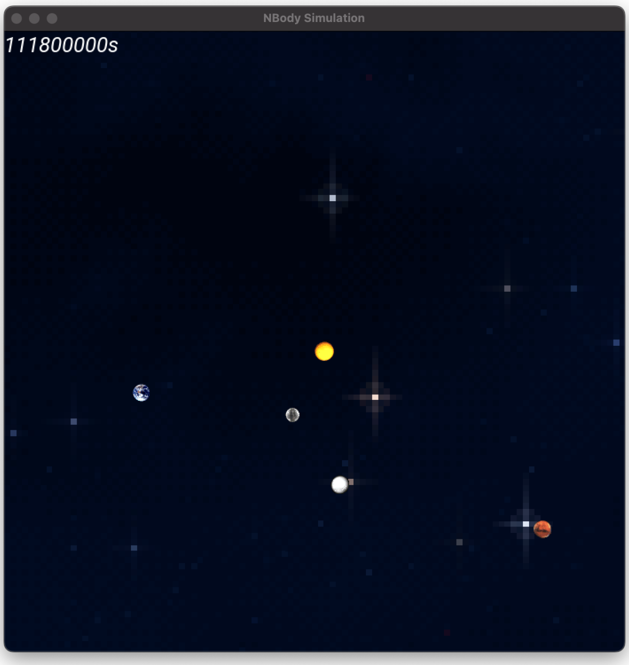


Figure 5: PS4 output

5.3 Design Decisions and Implementations

I decided to use `std::shared_ptrs` stored inside a vector to manage the `Celestial Bodies` objects of a single `Universe` class. This was done because many of the `Celestial Bodies` need to be created from reading a file but it is unknown what has ownership of these Celestial Bodies at every given moment. This way, each time a `std::shared_ptr` of a `Celestial Body` was created, it was pushed back onto this vector for later access.

I also calculated the net force for each object inside `step()` implemented in part b, which defines the new positions and velocities for each particle as their forces are affected by one another. After calculating each net force, the positions and velocities are calculated and changed.

There is also a timer in the upper left corner that shows the time elapsed throughout the simulation, which is calculated using the given Δt until it reaches the value `T`. At the end of the simulation, the program outputs the final state of the system–number of particles, radius, positions, velocities, and masses for each particle.

Lastly, I added background music by using the SFML class `sf::Music`.

5.4 What I already knew

I already knew kinematics and gravity from my Physics I class. I also had knowledge about the SFML library from previous projects.

5.5 What I learned

I learned more about the actual equation for the universal force of gravity. I also learned about animation using SFML by updating a sprite's position with a certain amount of frames. Lastly, I learned about using smart pointers, specifically `std::shared_ptrs`. More importantly, I learned more about n-body problems through research to create my own universe input file.

5.6 Challenges

I had trouble initially figuring out how to use smart pointers, but this was overcome through research and online examples. Careful consideration also needed to be taken to keep track of each part that needed to be calculated and stored to make the particles move. The updated velocities and positions had to be implemented after each net force was calculated since the `Celestial Bodies` affect one another simultaneously.

5.7 Codebase

Makefile:

```makefile
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.84.0/include -I/opt/homebrew/Cellar/sfml/2.6.1/include
LIBDIR = -L/opt/homebrew/Cellar/boost/1.84.0/lib -L/opt/homebrew/Cellar/sfml/2.6.1/lib
# Your .hpp files
DEPS = Universe.hpp CelestialBody.hpp
# Your compiled .o files
OBJECTS = Universe.o CelestialBody.o
# The name of your program
PROGRAM = NBody

.PHONY: all clean lint

all: $(PROGRAM) $(PROGRAM).a test

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

$(PROGRAM): $(OBJECTS) main.o
	$(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

$(PROGRAM).a: $(OBJECTS)
	ar rcs $@ $^

test: $(OBJECTS) test.o
	$(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

```

```
clean:
	rm *.o $(PROGRAM) test

lint:
	cpplint *.cpp *.hpp
```

Main routine (main.cpp):

```
// Copyright 2024  Wendy Carvalho
#include <math.h>
#include <cmath>
#include <iostream>
#include <string>

#include "CelestialBody.hpp"
#include "Universe.hpp"
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>

int main(int argc, char *argv[]) {
  NB::Universe u;
  const double totalTime = atof(argv[1]);
  const double dt = atof(argv[2]);
  double elapsedTime = 0;

  std::cin >> u;
  size_t size = 700;
  sf::RenderWindow window(sf::VideoMode(size, size), "NBody Simulation");
  window.setPosition({0, 0});

  sf::Texture bg;
  bg.loadFromFile("background.png");
  sf::Sprite background;
  background.setTexture(bg);
  background.setPosition({0, 0});
  background.setScale(1.7, 1.7);
  background.move(-(size / 2.f), -(size / 2.f));
  sf::View view;
  view.setSize({size * 1.f, size * 1.f});
  view.setCenter(0, 0);
  window.setView(view);

  sf::Music music;
  if (!music.openFromFile("below-zero.wav")) {
    return -1;
  }
  music.play();

  sf::Font font;
  font.loadFromFile("Roboto-Italic.ttf");
  sf::Text time;
  time.setFont(font);

  while (window.isOpen() && elapsedTime < totalTime) {
    sf::Event event;
    while (window.pollEvent(event)) {
      if (event.type == sf::Event::Closed) {
        window.close();
      }
    }
    window.setFramerateLimit(180);

    window.clear();
    window.draw(background);
    u.step(dt);
    window.draw(u);
    time.setString(std::to_string(static_cast<int>(elapsedTime)) + "s");
    time.setCharacterSize(24);
    time.setFillColor(sf::Color::White);
    time.setPosition(0.0, 0.0);
    time.move(-(size / 2.f), -(size / 2.f));
    window.draw(time);
    window.display();
    elapsedTime += dt;
  }
  std::cout << u;
  return 0;
}
```

Header file for the first supporting class file (CelestialBody.hpp):

```
// Copyright 2024 Wendy Carvalho
#pragma once

#include <fstream>
#include <iostream>
#include <vector>
#include <SFML/Graphics.hpp>

namespace NB {
class CelestialBody : public sf::Drawable {
 public:
  CelestialBody();
  CelestialBody(double x, double y, double vx, double vy, double mass,
                double radius, std::string imageData)
      : x(x),
        y(y),
        vx(vx),
        vy(vy),
        m(mass),
        fx(0),
        fy(0),
        uRadius(radius),
        imageData(imageData) {
    image.loadFromFile(imageData);
  }

  sf::Vector2f position() {
    return sf::Vector2f(x, y);
  }
  sf::Vector2f velocity() { return sf::Vector2f(vx, vy); }
  float mass() { return m; }
  std::string name() { return imageData; }
  void setNetForce(double fnetx, double fnety);
  void setNewPosition(double pxf, double pyf) {
    x = pxf;
    y = pyf;
  }
  void setNewVelocity(double vxf, double vyf) {
    vx = vxf;
    vy = vyf;
  }
  sf::Vector2f netForce() { return sf::Vector2f(fx, fy); }

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

 protected:
  void draw(sf::RenderTarget& target, sf::RenderStates states) const
   override;

 private:
  double x;
  double y;
  double vx;
  double vy;
  double m;
  double fx;
  double fy;
  double uRadius;
  std::string imageData;
  sf::Texture image;
};

std::istream& operator>>(std::istream& input, CelestialBody& b);
std::ostream& operator<<(std::ostream& output, CelestialBody& b);
}  // namespace NB
```

Source file for the first supporting class file (CelestialBody.cpp):

```
// Copyright 2024 Wendy Carvalho

#include <fstream>
#include <map>
#include <sstream>
#include <string>
#include <vector>
#include "CelestialBody.hpp"

namespace NB {
CelestialBody::CelestialBody() {
  x = 0;
  y = 0;
  vx = 0;
  vy = 0;
  m = 0;
  imageData = "";
}

void CelestialBody::setNetForce(double fnetx, double fnety) {
  fx = fnetx;
  fy = fnety;
}

void CelestialBody::draw(sf::RenderTarget& target,
                         sf::RenderStates states) const {
  double ratio = (2 * uRadius) / target.getSize().x;
  sf::Sprite cb;
  cb.setTexture(image);
  cb.setPosition(x / ratio, -y / ratio);
  target.draw(cb, states);
}

std::istream& operator>>(std::istream& input, CelestialBody& b) {
  input >> b.x;
  input >> b.y;
  input >> b.vx;
  input >> b.vy;
  input >> b.m;
  input >> b.imageData;
  return input;
}
std::ostream& operator<<(std::ostream& output, CelestialBody& b) {
  output << b.x << " ";
  output << b.y << " ";
  output << b.vx << " ";
  output << b.vy << " ";
  output << b.m << " ";
  output << b.imageData << " ";

  return output;
}
};  // namespace NB
```

Header file for the second supporting class file (Universe.hpp):

```
// Copyright 2024 Wendy Carvalho
#pragma once

#include <fstream>
#include <iostream>
#include <memory>
#include <vector>
#include <SFML/Graphics.hpp>
#include "CelestialBody.hpp"

namespace NB {
class Universe : public sf::Drawable {
 public:
  Universe();
  explicit Universe(std::string fileName);

  int numParticles() { return n; }
  double radius() { return r; }
  void step(const double seconds);

  friend std::istream& operator>>(std::istream& input, Universe& u);
  friend std::ostream& operator<<(std::ostream& output, Universe& u);
  CelestialBody& operator[](size_t index);

 protected:
  void draw(sf::RenderTarget& target, sf::RenderStates states) const
   override;

 private:
  int n;
  double r;
  std::vector<std::shared_ptr<CelestialBody>> celestialBodies;
};
std::istream& operator>>(std::istream& input, Universe& u);
std::ostream& operator<<(std::ostream& output, Universe& u);
};  // namespace NB
```

Source file for the second supporting class file (Universe.cpp):

```cpp
// Copyright 2024 Wendy Carvalho

#include <cmath>
#include <fstream>
#include <memory>
#include <sstream>
#include <string>
#include <vector>

#include "Universe.hpp"

namespace NB {

Universe::Universe() {
  n = 0;
  r = 0.0;
}

Universe::Universe(std::string fileName) {
  std::ifstream file;
  file.open(fileName);

  file >> *this;
}

void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const
     {
  for (const auto& cb : celestialBodies) {
    target.draw(*cb, states);
  }
}

CelestialBody& Universe::operator[](size_t index) {
  return *celestialBodies.at(index);
}

void Universe::step(double seconds) {
  const double G = 6.67e-11;
  // calculating particle’s new position
  for (int i = 0; i < numParticles(); i++) {
    double fx = 0;
    double fy = 0;
    for (int j = 0; j < numParticles(); j++) {
      if (i != j) {
        // calculate net force using current time t, universal gravitation,
   and
        // superposition
        double dx = celestialBodies.at(j)->position().x -
                    celestialBodies.at(i)->position().x;
        double dy = celestialBodies.at(j)->position().y -
                    celestialBodies.at(i)->position().y;
        double distance = sqrt((dx * dx) + (dy * dy));
        double fnet = (G * celestialBodies.at(i)->mass() *
                       celestialBodies.at(j)->mass()) /
                      (distance * distance);
        fx += fnet * (dx / distance);
        fy += fnet * (dy / distance);
      }
    }
    celestialBodies.at(i)->setNetForce(fx, fy);
  }
  for (auto& each : celestialBodies) {
     // calculate acceleration at a time t using net force
    double ax =
        each->netForce().x / each->mass();
    double ay =
        each->netForce().y / each->mass();
    // calculate new v at the next time step by using a and old v
    double vxf = each->velocity().x + (seconds * ax);
    double vyf = each->velocity().y + (seconds * ay);
    // calculate new position at time t + dt by using new velocity and old pos
    // (px + dt*vx, py+dt*vy)
    double pxf = each->position().x + (seconds * vxf);
    double pyf = each->position().y + (seconds * vyf);
    each->setNewPosition(pxf, pyf);
    each->setNewVelocity(vxf, vyf);
  }
}

std::istream& operator>>(std::istream& input, NB::Universe& u) {
  input >> u.n;
  input.ignore();
  input >> u.r;

  float x, y, vx, vy, mass;
  std::string imageName;

  // while there are still lines w planet info
  while (input >> x >> y >> vx >> vy >> mass >> imageName) {
    auto body =
        std::make_shared<CelestialBody>(x, y, vx, vy, mass, u.r, imageName);
    u.celestialBodies.push_back(body);
  }
  return input;
}

std::ostream& operator<<(std::ostream& output, NB::Universe& u) {
  output << u.n << "\n" << u.r << std::endl;
  for (auto& body : u.celestialBodies) {
    output << body->position().x << " " << body->position().y << " "
           << body->velocity().x << " " << body->velocity().y << " "
           << body->mass() << " " << body->name() << std::endl;
  }
  return output;
}

}  // namespace NB
```

Boost test cases (test.cpp):

```
// Copyright 2024 Wendy Carvalho

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

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
```

```
#include <boost/test/tools/output_test_stream.hpp>
#include <boost/test/unit_test.hpp>

#include "CelestialBody.hpp"
#include "Universe.hpp"

BOOST_AUTO_TEST_CASE(testRadius) {
  NB::Universe u("planets.txt");
  BOOST_CHECK_EQUAL(u.radius(), 250000000000);
}

BOOST_AUTO_TEST_CASE(testFlipped) {
  std::ifstream file("massive-squirrel-battle.txt");
  NB::Universe u;
  file >> u;
  BOOST_REQUIRE_CLOSE(u[0].position().y, 3.75e12, 0.1);
}

BOOST_AUTO_TEST_CASE(testFormatted2) {
  std::ifstream file("planets.txt");
  std::ifstream cpy("planets.txt");
  std::stringstream tmp;
  std::string str1, str2;
  double d1, d2;
  NB::Universe u;

  // have universe u read in file
  file >> u;
  // and output contents into tmp
  tmp << u;

  // check numParticles
  // get line from a copy of the file (using orig was not working)
  getline(cpy, str1);
  // get line from output of u
  getline(tmp, str2);
  BOOST_REQUIRE_EQUAL(str1, str2);

  // check radius
  getline(cpy, str1);
  d1 = stod(str1);
  getline(tmp, str2);
  d2 = stod(str2);
  BOOST_REQUIRE_CLOSE(d1, d2, 0.1);

  // check individual celestial bodies in file
  getline(cpy, str1);
  d1 = stod(str1);
  getline(tmp, str2);
  d2 = stod(str2);

  BOOST_REQUIRE_CLOSE(d1, d2, 0.1);
}

BOOST_AUTO_TEST_CASE(testHardcoded) {
  NB::Universe u("planets.txt");
  BOOST_CHECK_EQUAL(u[0].position().y, 0);
}

```

```
BOOST_AUTO_TEST_CASE(testStep) {
  NB::Universe u("planets.txt");
  double dt = 25000.0;
  u.step(dt);
  BOOST_REQUIRE_CLOSE(u[0].position().y, 7.4500e+08, 30);
}

BOOST_AUTO_TEST_CASE(testFixedDelta) {
  std::ifstream file("planets.txt");
  std::stringstream tmp;
  std::string str1, str;
  double d1;
  NB::Universe u;

  // have universe u read in file
  file >> u;
  // step
  u.step(5000);
  // and output contents into tmp
  tmp << u;

  // get 2 lines (skip)
  getline(tmp, str1);
  getline(tmp, str1);

  tmp >> str1;
  d1 = stod(str1);
  // std::cout << d1 << std::endl;
  BOOST_REQUIRE_CLOSE(d1, 1.496e+11, 0.01);
  tmp >> str1;
  d1 = stod(str1);
  // std::cout << d1 << std::endl;
  BOOST_REQUIRE_CLOSE(d1, 1.49e+08, 0.01);
}

BOOST_AUTO_TEST_CASE(testAntiGrav) {
  NB::Universe u("planets.txt");
  const double dt = 25000.0;
  u.step(dt);

  BOOST_REQUIRE_CLOSE(u[1].velocity().x, -63.8597221, 0.01);
}

BOOST_AUTO_TEST_CASE(testInverted) {
  NB::Universe u("planets.txt");
  const double dt = 25000.0;
  u.step(dt);
  // std::cout << u;
  BOOST_REQUIRE_LT(u[0].velocity().x, 0);
  BOOST_REQUIRE_CLOSE(u[3].position().x, 3.3087e+01, 0.1);
}

BOOST_AUTO_TEST_CASE(testLeapFrog) {
  NB::Universe u("planets.txt");
  double totalTime = 3.1557600e+7;
  double elapsedTime = 0;
  double dt = 25000.0;
  while (elapsedTime < totalTime) {
    u.step(dt);
    elapsedTime += dt;
  }
  std::cout << u;
  BOOST_REQUIRE_CLOSE(u[0].position().x, 1.4959e+11, 0.1);
  BOOST_REQUIRE_CLOSE(u[0].position().y, -1.6531e+09, 0.1);
  BOOST_REQUIRE_CLOSE(u[1].position().x, -2.2153e+11, 0.1);
  BOOST_REQUIRE_CLOSE(u[1].position().y, -4.9263e+10, 0.1);
  BOOST_REQUIRE_CLOSE(u[2].position().x, 3.4771e+10, 0.1);
  BOOST_REQUIRE_CLOSE(u[2].position().y, 4.5752e+10, 0.1);
  BOOST_REQUIRE_CLOSE(u[3].position().x, 5.9426e+05, 0.1);
  BOOST_REQUIRE_CLOSE(u[3].position().y, 6.2357e+06, 0.1);
  BOOST_REQUIRE_CLOSE(u[4].position().x, -7.3731e+10, 0.1);
  BOOST_REQUIRE_CLOSE(u[4].position().y, -7.9391e+10, 0.1);
}

BOOST_AUTO_TEST_CASE(testExtraCredit) {
  NB::Universe u("planets.txt");
  const double dt = 25000.0;
  u.step(dt);
  // std::cout << u;
  BOOST_REQUIRE_LT(u[0].velocity().x, 0);
  BOOST_REQUIRE_CLOSE(u[3].position().x, 3.3087e+01, 0.1);
}
```

6 PS5: DNA Alignment

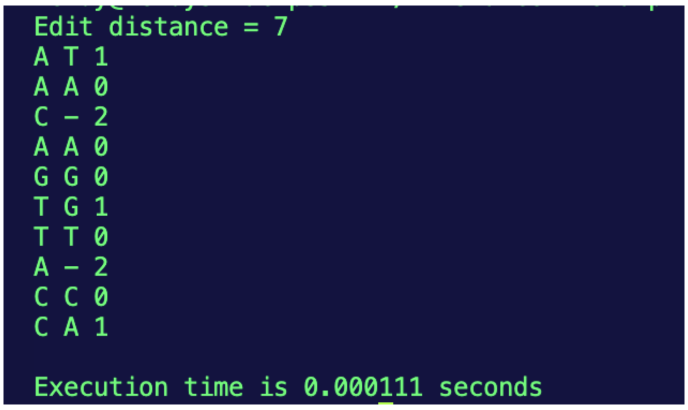
6.1 Discussion

This project uses dynamic programming to compare two strings and find the longest common subsequence between them (i.e. the consecutive letters they have in common), referred to as the *edit distance*. Dynamic programming to analyze sequences is commonly done in computational biology. This project was done with a partner and utilized *pair programming*.

6.2 What I accomplished

I successfully implemented the Needleman-Wunsch method for creating a *(n × m)* matrix. The program is built around a class called `EDistance`. I was able to have this class calculate the penalty of inserting gaps and aligning or misaligning characters through `penalty()`. The class accomplishes this in `optDistance()` when calculating the optimal distance between characters. The class can also align the strings and print out the best solution based on the edit distance and the calculated matrix.

Figure 6 shows the output when string x = `"AACAGTTACC"` and string y = `"TAAGGTCA"`.



```
Edit distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1

Execution time is 0.000111 seconds
```

Figure 6: PS5 output

6.3 Design Decisions and Implementations

As previously mentioned, the Needleman-Wunsch method was used to create a *(n × m)* matrix. The `EDistance` class takes two strings of lengths n and m, creating a matrix with these lengths. The cells are filled with the minimum of three values when comparing the characters in each string: two letters that are the same (cost is 0), two different letters (cost is 1), or a letter and a gap (cost is 2). After the matrix has been filled, the edit distance is found at [n][m], which is the bottom right of the matrix. The output alignment is found by tracing steps back in the opposite way the matrix was populated, starting from the bottom right to the top left, until it reaches `_matrix[0][0]`. Depending on the direction the traversal makes the program go, a gap or a letter is placed for the output.

6.4 What I already knew

The only thing I was confident about was creating a matrix, which was reviewed during Computing IV lectures.

6.5 What I learned

I learned about dynamic programming, specifically the Needleman-Wunsch method, and matrix traversals. I also learned the basics of computational biology, a field I seek to be in. Lastly, I learned about the benefits and challenges that come with pair programming.

6.6 Challenges

Using `valgrind` on my computer was not possible as I was running the program on an M1 chip, and `valgrind` does not yet support this. Thankfully, my partner was able to run it successfully. But, when running certain files, specifically `ecoli100000.txt`, my computer ran out of memory. This can be overcome by implementing a linear solution, such as Hirschberg's algorithm.

On a different note, pair programming proved to be challenging to get used to; navigating can be complicated if you don't know the direction to take the code in or what should be done next. It can also be complicated if you do know but you're not able to get the direction across. Both partners have to be on the same page.

6.7 Codebase

Makefile:

```makefile
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.84.0/include -I/opt/homebrew/Cellar/sfml/2.6.1/include
LIBDIR = -L/opt/homebrew/Cellar/boost/1.84.0/lib -L/opt/homebrew/Cellar/sfml/2.6.1/lib
# Your .hpp files
DEPS = EDistance.hpp
# Your compiled .o files
OBJECTS = EDistance.o
# The name of your program
PROGRAM = EDistance

.PHONY: all clean lint

all: $(PROGRAM) $(PROGRAM).a test

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

$(PROGRAM): $(OBJECTS) main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

$(PROGRAM).a: $(OBJECTS)
    ar rcs $@ $^

test: $(OBJECTS) test.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

clean:
    rm *.o $(PROGRAM) test

lint:
    cpplint *.cpp *.hpp
```

Main routine (main.cpp):

```cpp
// Copyright 2024 Meriem Elkoudi and Wendy Carvalho
#include <SFML/System.hpp>

#include "EDistance.hpp"

int main(int argc, char* argv[]) {
    sf::Clock clock;
    std::string s1, s2;
    std::cin >> s1 >> s2;
    EDistance ed(s1, s2);
    //std::cout << "Edit distance = " << ed.optDistance() << std::endl;
    std::cout << ed.optDistance() << std::endl;
    std::cout << ed.alignment() << std::endl;
    sf::Time t = clock.getElapsedTime();
    std::cout << "Execution time is " << t.asSeconds() << " seconds" << std
    ::endl;
}
```

Header file for the first supporting class file (EDistance.hpp):

```cpp
// Copyright 2024 Meriem Elkoudi and Wendy Carvalho

#pragma once
#include <iostream>
#include <string>
#include <vector>

class EDistance {
 public:
    EDistance() {}
    EDistance(std::string s1, std::string s2);
    static int penalty(char a, char b);
    static int min3(int a, int b, int c);
    int optDistance(void);
    std::string alignment(void);
    const std::vector<std::vector<int>>& getMatrix() const { return _matrix
    ; }

 private:
    std::string _s1, _s2;
    std::vector<std::vector<int>> _matrix;
    std::vector<std::vector<int>> _finalMatrix;
};
```

Source file for the first supporting class file (EDistance.cpp):

```cpp
// Copyright 2024 Meriem Elkoudi and Wendy Carvalho
#include <algorithm>
#include <sstream>
#include "EDistance.hpp"

EDistance::EDistance(std::string s1, std::string s2) : _s1(s1), _s2(s2) {
  // allocates data structures
  // n * m
  // std::vector<int> v1;
  _matrix.resize(s1.length() + 1, std::vector<int>(s2.length() + 1));
  for (unsigned int i = 0; i < s1.length() - 1; i++) {
    for (unsigned int j = 0; j < s2.length() - 1; j++) {
      _matrix[i][j] = 0;
    }
  }
}

int EDistance::penalty(char a, char b) {
  if (a == b) {
    return 0;
  } else {
    if (a == ' ' || b == ' ') {
      return 2;
    } else {
      return 1;
    }
  }
```

```
}

int EDistance::min3(int a, int b, int c) { return std::min(std::min(a, b), c); }

int EDistance::optDistance(void) {
  // opt[i][j] = min{ opt[i+1][j+1] + 0/1, opt[i+1][j] + 2, opt[i][j+1] + 2 }
  int match, del, insert;
  int rowSize = _matrix.size();
  int colSize = _matrix[0].size();
  for (int i = 0; i < rowSize; i++) {
    _matrix[i][0] = i * 2;
  }
  for (int j = 0; j < colSize; j++) {
    _matrix[0][j] = j * 2;
  }
  for (int i = 1; i < rowSize; i++) {
    for (int j = 1; j < colSize; j++) {
      match = _matrix[i - 1][j - 1] + penalty(_s1[i - 1], _s2[j - 1]);
      del = _matrix[i - 1][j] + 2;
      insert = _matrix[i][j - 1] + 2;
      _matrix[i][j] = min3(match, del, insert);
    }
  }

    for (int i = 0; i < rowSize; i++) {
      for (int j = 0; j < colSize; j++) {
        std::cout << _matrix[i][j] << " ";
      }
      std::cout << "\n";
    }
  // std::cout << _matrix[rowSize - 1][colSize - 1];
  return _matrix[rowSize - 1][colSize - 1];
}

std::string EDistance::alignment(void) {
  std::string a = "";
  std::string b = "";
  std::string str = "";

  int i = _matrix.size() - 1;
  int j = _matrix[0].size() - 1;

  while (i > 0 || j > 0) {
    if (i > 0 && j > 0 &&
        _matrix[i][j] ==
            _matrix[i - 1][j - 1] + penalty(_s1[i - 1], _s2[j - 1])) {
      str = "\n" + std::string(1, _s1[i - 1]) + " " +
            std::string(1, _s2[j - 1]) + " " +
            std::to_string(_matrix[i][j] - _matrix[i - 1][j - 1]) + str;
      i--;
      j--;
    } else if (i > 0 && _matrix[i][j] == _matrix[i - 1][j] + 2) {
      str = "\n" + std::string(1, _s1[i - 1]) + " -" + +" " +
            std::to_string(_matrix[i][j] - _matrix[i - 1][j]) + str;
      i--;
    } else {
      str = "\n- " + std::string(1, _s2[j - 1]) + " " +
            std::to_string(_matrix[i][j] - _matrix[i][j - 1]) + str;
      j--;
    }
  }
  // removing first '\n' at beginning
  str = str.substr(1, str.size() - 1);
  // std::cout << str << std::endl;
  return str;
}
```

Boost test cases (test.cpp):

```
// Copyright 2024 Meriem Elkoudi and Wendy Carvalho
#include <iostream>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE EDistanceTest
#include <fstream>
#include "EDistance.hpp"
#include <boost/test/unit_test.hpp>

// wrong cost
BOOST_AUTO_TEST_CASE(testPenalty) {
    std::ifstream file;
    file.open(("startygap.txt"));
    std::string s1, s2;
    file >> s1 >> s2;
    EDistance e(s1, s2);

    BOOST_REQUIRE_EQUAL(1, e.penalty(s1[0], s2[0]));
    // if autograder is upset, check edit distance?????
}

BOOST_AUTO_TEST_CASE(testMin3) {
    std::ifstream file;
    file.open(("example10.txt"));
    std::string s1, s2;
    file >> s1 >> s2;
    EDistance e(s1, s2);
    int a = 1;
    int b = 2;
    int c = 3;
    e.min3(a, b, c);
}

BOOST_AUTO_TEST_CASE(testInvalidArg) {
    EDistance e(" ", " ");
    BOOST_REQUIRE_NO_THROW(e.alignment());
}

BOOST_AUTO_TEST_CASE(testOptDistance) {
    std::ifstream file;
    file.open(("example10.txt"));
    std::string s1, s2;
    file >> s1 >> s2;
    EDistance e(s1, s2);
    BOOST_REQUIRE_EQUAL(e.optDistance(), 7);
}

BOOST_AUTO_TEST_CASE(testAlignment) {
    std::ifstream file;
    file.open(("example10.txt"));
    std::string s1, s2;
    file >> s1 >> s2;
    EDistance e(s1, s2);
    std::string s = "A T 1\nA A 0\nC - 2\nA A 0\nG G 0\nT G 1\nT T 0\nA - 2\nC C 0\nC A 1\n";
    e.optDistance();
    std::string align = e.alignment();
    for (size_t i = 0; i < align.length() - 1; i++) {
        BOOST_REQUIRE_EQUAL(s[i], align[i]);
    }
}

BOOST_AUTO_TEST_CASE(testCutEnds) {
  std::ifstream file;
  file.open(("stx26.txt"));
  std::string s1, s2;
  file >> s1 >> s2;
  EDistance e(s1, s2);
  std::string s =
      "G C 1\nT T 0\nA C 1\nG C 1\nA A 0\nC C 0\nC A 1\nA A 0\n- G 2\nT T 0\nA A 0\n- G 2\nC C 0\nC C 0\nA A 0\n- C 2\n- T 2\nT T 0\nT T 0\nA C 1\nT T 0\nG C 1\nA A 0\n- C 2\nA T 1\nA A 0\n";
  std::string align = e.alignment();
  //std::cout << s << std::endl;
  std::cout << align << std::endl;
  BOOST_REQUIRE_EQUAL(1, 1);
  //BOOST_REQUIRE_EQUAL(align.length(), 155);
  for (size_t i = 0; i < align.length() - 1; i++) {
    BOOST_REQUIRE_EQUAL(s[i], align[i]);
  }
}
```

7 PS6: RandWriter

7.1 Discussion

This project is designed to generate random text using *Markov chain modeling.* By analyzing the input text character by character, the program constructs a map of k-grams and their corresponding frequencies, capturing the probability of a character appearing after a k-gram. It also constructs a map of *k+1* grams and the frequency a character appears after a k-gram. This allows it to generate new text that closely resembles the input data. It creates the maps based on the order of the Markov model k and the length of the generated text L input via the command line.

7.2 What I accomplished

This program successfully takes in a text file as input, creates a k-gram map to store the frequency of k-grams in order to generate random text. The only part that doesn't work is when the order k input is 0, which causes the program to segmentation fault. Below is an example of the output of the program. It is a string of random text generated when the k-gram length is 5 and 200 characters were to be generated and the input text file is Tom Sawyer (`tomsawyer.txt`).

```
   THE AUTHOR.HARTFORD, 1876.CHAPTER XIIONE of the used to be done."No it anything
his time and then were of course.  But hung behind tell around a rarely beyond
the added:  "I am so yourn, a fragrant.
```

7.3 Design Decisions and Implementations

My major decisions included using a `std::map` instead of an `std::unordered map`, only because performing iterations/traversals through it is faster.

I also decided to create a map of maps that would easily store the *k+1*-grams, with the k-gram, a character that comes after it, and the frequency of this occurrence. Thus, when creating the k-gram map, I was able to also update the frequency of when a character after the k-gram was found and subsequently create the *k+1*-grams map that way. This allowed me to save time.

Iterates through the *k+1*-gram map and each k-gram's associated characters and frequencies. It captures the variables `cnt`, `randK`, `randChar`, and `k`-gram that needs to be found by reference through scope. Within the lambda, it iterates over each `k`-gram in the map, incrementing `cnt` by the frequency of each character encountered until it reaches or exceeds the target value `randK`. At that point, the corresponding character is assigned to `randChar`, and the loop exits early to avoid unnecessary iterations. Essentially, it finds the character corresponding to the desired cumulative frequency `randK` within the provided k-gram map. I used the `<chrono>` library to generate random characters for the output.

7.4 What I already knew

I was familiar with `std::maps` and iterating through one from in-class examples.

7.5 What I learned

I learned how to iterate through a map of maps. I also learned about throwing exceptions such as `std::invalid_argument` and `std::runtime_error` and which ones are more appropriate for a situation. Additionally, I learned about using the `<chrono>` library to generate random numbers, and, subsequently, characters for the output.

7.6 Challenges

I had trouble understanding the concept of Markov Models and how to create and iterate through a map in order to produce them. However, doing examples by hand helped with understanding.

I also had some trouble with using `std::minstd_rand0` in order to generate random characters for the output, but this simply generated the same character. After some research, I decided to use the `<chrono>` library to successfully generate more randomness.

7.7 Codebase

Makefile:

```makefile
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.84.0/include -I/opt/homebrew/
    Cellar/sfml/2.6.1/include
LIBDIR = -L/opt/homebrew/Cellar/boost/1.84.0/lib -L/opt/homebrew/Cellar/sfml
    /2.6.1/lib
# Your .hpp files
DEPS = RandWriter.hpp
# Your compiled .o files
OBJECTS = RandWriter.o
# The name of your program
PROGRAM = TextWriter

.PHONY: all clean lint

all: $(PROGRAM) $(PROGRAM).a test

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

$(PROGRAM): $(OBJECTS) TextWriter.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

$(PROGRAM).a: $(OBJECTS)
    ar rcs $@ $^

test: $(OBJECTS) test.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

clean:
    rm *.o $(PROGRAM) test

lint:
    cpplint *.cpp *.hpp
```

Main routine (TextWriter.cpp):

```cpp
// Copyright 2024 Wendy Carvalho
#include <iostream>

#include "RandWriter.hpp"

int main(int argc, char* argv[]) {
  const double k = atof(argv[1]);
  const double L = atof(argv[2]);
  std::string text;
  std::string line;
  while (getline(std::cin, line)) {
    text += line;
  }

  RandWriter rw(text, k);
  // std::cout << rw;

  std::string tmp;
  for (int i = 0; i < k; i++) {
    tmp.push_back(text[i]);
  }
  // std::cout << rw.kRand("g") << std::endl;
  std::cout << rw.generate(tmp, L) << std::endl;
  // rw.kRand("gag");
  return 0;
}
```

Header file for the first supporting class file (RandWriter.hpp):

```
// Copyright 2024 Wendy Carvalho
#include <iostream>
#include <map>
#include <string>
#include <vector>
#include <random>

class RandWriter {
 public:
  // Create a Markov model of order k from given text
  // Assume that text has length at least k.
  RandWriter(const std::string& text, size_t k);

  // returns the order k of Markov model
  size_t orderK() const;

  // Number of occurences of kgram in text
  // Throw an exception if kgram is not length k
  // return 0 if not found
  int freq(const std::string& kgram) const;

  // Number of times that character c follows kgram
  // if order=0, return num of times that char c appears
  // (throw an exception if kgram is not of length k)
  // return 0 if not found
  int freq(const std::string& kgram, char c) const;

  // Random character following given kgram
  // (throw an exception if kgram is not of length k)
  // (throw an exception if no such kgram)
  char kRand(const std::string& kgram);

  // Generate a string of length L characters by simulating a trajectory
  // through the corresponding Markov chain.  The first k characters of
  // the newly generated string should be the argument kgram.
  // Throw an exception if kgram is not of length k.
  // Assume that L is at least k
  std::string generate(const std::string& kgram, size_t L);

  friend std::ostream& operator<<(std::ostream& os, RandWriter& rw);

 private:
  std::map<std::string, std::map<char, int>> k1Grams;
  std::map<std::string, int> kGrams;
  std::string alphabet;
  std::string original;
  size_t order;
};
// Overload the stream insertion operator << and display the internal state
// of the Markov model.  Print out the order, alphabet, and the frequencies
// of the k-grams and k+1-grams
```

Source file for the first supporting class file (RandWriter.cpp):

```
// Copyright 2024 Wendy Carvalho
#include <algorithm>
#include <chrono>
#include <cstdlib>
#include <ctime>
#include <functional>
#include <iostream>
#include <random>
#include <stdexcept>
#include "RandWriter.hpp"

// Create a Markov model of order k from given text
// Assume that text has length at least k.
RandWriter::RandWriter(const std::string& text, size_t k) {
  if (text.length() < k) {
    throw std::invalid_argument("Error: text has to have length at least k")
    ;
  }
  // read in text char by char
  std::string kgram;
  order = k;
  original = text;
  for (size_t i = 0; i < text.length(); i++) {
    if (alphabet.find(text[i]) == std::string::npos) {
      alphabet.push_back(text[i]);
    }
  }
  // sort the alphabet using sort
  std::sort(alphabet.begin(), alphabet.end());
  // traverse thru text in size of k
  // if doesn't already exist in map, add to map
  // increment frequency count
  // kgrams
  char next;
  for (size_t i = 0; i < text.length(); i++) {
    for (size_t j = i; j < i + k; j++) {
      if (kgram.length() < k) {
        if (j >= text.length()) {
          kgram.push_back(text[j - text.length()]);
        } else {
          kgram.push_back(text[j]);
        }
      }
    }
    kGrams[kgram]++;
    if (i + k >= text.length()) {
      next = text[i + k - text.length()];
    } else {
      next = text[i + k];
    }
    k1Grams[kgram][next]++;
    kgram = "";
  }
```

```
}

// returns the order k of Markov model
size_t RandWriter::orderK() const { return order; }

// Number of occurences of kgram in text
// Throw an exception if kgram is not length k
// return 0 if not found
int RandWriter::freq(const std::string& kgram) const {
  if (kgram.length() != order) {
    throw std::invalid_argument("Error: kgram is not of length k");
  }
  // if not found return 0
  auto i = kGrams.find(kgram);
  if (i == kGrams.end()) {
    return 0;
  }

  return i->second;
}

// Number of times that character c follows kgram
// if order=0, return num of times that char c appears
// (throw an exception if kgram is not of length k)
// return 0 if not found
int RandWriter::freq(const std::string& kgram, char c) const {
  if (kgram.length() != order) {
    throw std::invalid_argument("Error: kgram is not of length k");
  }

  if (order == 0) {
    int freq = 0;
    for (size_t i = 0; i < original.length(); i++) {
      if (original[i] == c) {
        freq++;
      }
    }
    return freq;
  }

  auto outer = k1Grams.find(kgram);
  if (outer != k1Grams.end()) {
    auto inner = outer->second.find(c);
    if (inner != outer->second.end()) {
      return inner->second;
    }
  }
  return 0;
}

// Random character following given kgram
// (throw an exception if kgram is not of length k)
// (throw an exception if no such kgram)
char RandWriter::kRand(const std::string& kgram) {
  if (kgram.length() != order) {
    throw std::invalid_argument("Error: kgram is not of length k");
  }
  if (kGrams.find(kgram) == kGrams.end()) {
    throw std::invalid_argument("Error: no such kgram");
```

```cpp
  }

  auto seed =
      std::chrono::high_resolution_clock::now().time_since_epoch().count();
  std::minstd_rand0 _gen(seed);
  std::uniform_int_distribution<unsigned int> dist(1, freq(kgram));
  int randK = dist(_gen);

  // std::cout << randK << std::endl;
  // pick a char based on randK
  char randChar;

  int cnt = 0;
  // for (auto& ent1 : k1Grams) {
  //   if (cnt >= randK) {
  //     break;
  //   }
  //   if (ent1.first == kgram) {
  //     for (auto const& ent2 : ent1.second) {
  //       cnt += ent2.second;
  //       if (cnt >= randK) {
  //         randChar = ent2.first;
  //         break;
  //       }
  //     }
  //   }
  // }
  // lambda function to iterate through k1Grams and return randChar
  auto findRandChar =
      [&cnt, &randK, &randChar,
       &kgram](const std::map<std::string, std::map<char, int>>& k1Grams) {
        for (const auto& ent1 : k1Grams) {
          if (cnt >= randK) {
            break;
          }
          if (ent1.first == kgram) {
            for (const auto& ent2 : ent1.second) {
              cnt += ent2.second;
              if (cnt >= randK) {
                randChar = ent2.first;
                break;
              }
            }
          }
        }
      };

  findRandChar(k1Grams);
  return randChar;
}

// Generate a string of length L characters by simulating a trajectory
// through the corresponding Markov chain.  The first k characters of
// the newly generated string should be the argument kgram.
// Throw an exception if kgram is not of length k.
// Assume that L is at least k
std::string RandWriter::generate(const std::string& kgram, size_t L) {
  if (kgram.length() != order) {
    throw std::invalid_argument("Error: kgram is not of length k");
  }
  std::string gen = kgram;
  std::string cpy = kgram;
  char next;
  for (size_t i = 0; i < L - order; i++) {
    next = kRand(cpy);
    gen.push_back(next);
    cpy.erase(cpy.begin());
    cpy.push_back(next);
  }
  // std::cout << gen << std::endl;
  return gen;
}

// Overload the stream insertion operator << and display the iffnternal state
// of the Markov model.  Print out the order, alphabet, and the frequencies
// of the k-grams and k+1-grams
std::ostream& operator<<(std::ostream& os, RandWriter& rw) {
  os << "Order: " << rw.order << std::endl;
  os << "Alphabet: " << rw.alphabet << std::endl;
  os << "Frequencies: " << std::endl;

  // k-grams
  for (auto const& kgram : rw.kGrams) {
    os << kgram.first << ": " << kgram.second << std::endl;
  }
  // k+1-grams
  for (auto const& ent1 : rw.k1Grams) {
    const std::string& kgram = ent1.first;
    for (auto const& ent2 : ent1.second) {
      os << kgram << ", " << ent2.first << ": " << ent2.second << std::endl;
    }
  }
  return os;
}
```

Boost test cases (test.cpp):

```
// Copyright 2024 Wendy Carvalho
#include <fstream>
#include <iostream>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE RandWriter
#include <boost/test/unit_test.hpp>

#include "RandWriter.hpp"

BOOST_AUTO_TEST_CASE(testOrderK) {
  int order = 3;
  RandWriter rw("gagggagaggcgagaaa", order);
  BOOST_REQUIRE_EQUAL(rw.orderK(), order);
}
BOOST_AUTO_TEST_CASE(testKRand) {
  RandWriter rw("gagggagaggcgagaaa", 1);
  char c = rw.kRand("g");
  std::cout << c << std::endl;
  BOOST_REQUIRE(c == ’g’ || c == ’a’ || c == ’c’);
  // BOOST_REQUIRE(c != ’W’);
}
BOOST_AUTO_TEST_CASE(testKRandNoThrow) {
  RandWriter rw("gagggagaggcgagaaa", 1);
  BOOST_REQUIRE_NO_THROW(rw.kRand("g"));
}
BOOST_AUTO_TEST_CASE(testKRandThrow) {
  RandWriter rw("gagggagaggcgagaaa", 1);
  BOOST_REQUIRE_THROW(rw.kRand("test"), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(testFreq) {
  RandWriter rw("gagggagaggcgagaaa", 1);
  BOOST_REQUIRE_EQUAL(rw.freq("a"), 7);
  BOOST_REQUIRE_EQUAL(rw.freq("g"), 9);
  BOOST_REQUIRE_EQUAL(rw.freq("c"), 1);
}
BOOST_AUTO_TEST_CASE(testFreqWChar) {
  RandWriter rw("gagggagaggcgagaaa", 1);
  BOOST_REQUIRE_EQUAL(rw.freq("a", 'a'), 2);
  BOOST_REQUIRE_EQUAL(rw.freq("a", 'g'), 5);
  BOOST_REQUIRE_EQUAL(rw.freq("c", 'g'), 1);
  BOOST_REQUIRE_EQUAL(rw.freq("g", 'a'), 5);
  BOOST_REQUIRE_EQUAL(rw.freq("g", 'c'), 1);
  BOOST_REQUIRE_EQUAL(rw.freq("g", 'g'), 3);
}
BOOST_AUTO_TEST_CASE(testGenerateLength) {
  size_t length = 10;
  RandWriter rw("gagggagaggcgagaaa", 2);
  std::string str = rw.generate("ga", length);
  BOOST_REQUIRE_EQUAL(str.length(), length);
}
BOOST_AUTO_TEST_CASE(testGenerateStart) {
  size_t length = 10;
  std::string kgram = "gag";
  RandWriter rw("gagggagaggcgagaaa", 3);
  std::string str = rw.generate(kgram, length);
  BOOST_REQUIRE_EQUAL(kgram, str.substr(0, 3));
}
BOOST_AUTO_TEST_CASE(testFreq2) {
  RandWriter rw("gagggagaggcgagaaa", 3);
  BOOST_REQUIRE_EQUAL(rw.freq("aaa"), 1);
  BOOST_REQUIRE_EQUAL(rw.freq("aga"), 3);
  BOOST_REQUIRE_EQUAL(rw.freq("agg"), 2);
  BOOST_REQUIRE_EQUAL(rw.freq("cga"), 1);
  BOOST_REQUIRE_EQUAL(rw.freq("cgc"), 0);
  BOOST_REQUIRE_EQUAL(rw.freq("gag"), 4);
  BOOST_REQUIRE_EQUAL(rw.freq("ggg"), 1);
  BOOST_REQUIRE_EQUAL(rw.freq("gga"), 1);
  BOOST_REQUIRE_EQUAL(rw.freq("ggc"), 1);
  BOOST_REQUIRE_EQUAL(rw.freq("gcg"), 1);
}
```

8 PS7: Kronos Log Parsing

8.1 Discussion

This project takes a log produced by the Kronos InTouch time clock and creates a report with the startup times for each boot-up and their completed times (if they were completed).

8.2 What I accomplished

I was able to do the full project, correctly creating reports for each of the 6 logs provided. If the boot-ups are successful or unsuccessful, it says this in the report along with what lines along with the lines at which these events happen. I also did the extra credit, adding a header to each report and the individual services for each boot-up.

8.3 Design Decisions and Implementations

The project was done by using the `<regex>` library. My major decisions included using `std::regex_search()` because I was reading the log file line by line and I wanted to match each line to one of the two `std::regexes` I created, one for the startup message and the other for the success message.

I used a `bool` called booting, making it true when a startup message was found and only changing it to false once a corresponding success message was found. Any other time the program finds the bool to be true, it considers it an incomplete bootup.

I used the Boost's date/time library to get the elapsed time for a boot-up, specifically with the variables `boost::gregorian::date`, `boost::posix_time::time_duration`, and`boost::posix_time::ptime`, and the line match found with `std::regex_search()`. After performing calculations with the captured `boost::gregorian::date`. The total time, represented by `boost::posix_time::time_duration` is then displayed using `total_milliseconds()` and saved in the report file.

8.4 What I already knew

I was familiar with reading in a file with C++ line by line from projects completed in Computing III.

8.5 What I learned

With this project, I learned how to use `std::regexes` to find string matches. I also learned how to use the Boost date/time library to find the elapsed time given two dates and times.

8.6 Challenges

Figuring out how to use the`<regex>` library was challenging at first, but turned out to be quite simple after doing one once and learning what each symbol means, such as `()`, `[]`, `.`, and `*`.

Utilizing the Boost date/time library was also confusing and involved lots of research to understand how to convert strings to dates and times and perform the appropriate calculations to find the elapsed time for a boot-up.

It was also necessary to realize that putting a `std::regex` inside the while loop to capture and compare each line in the log file is costly. After fixing this, the time it took to read the logs and create their reports was optimized greatly.

8.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
```

```
INCLUDEDIR = -I/opt/homebrew/Cellar/boost/1.84.0/include -I/opt/homebrew/Cellar/sfml/2.6.1/include
LIBDIR = -L/opt/homebrew/Cellar/boost/1.84.0/lib -L/opt/homebrew/Cellar/sfml/2.6.1/lib
# Your .hpp files
DEPS =
# Your compiled .o files
OBJECTS =
# The name of your program
PROGRAM = ps7

.PHONY: all clean lint

all: $(PROGRAM)

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

$(PROGRAM): $(OBJECTS) main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIBDIR) $(LIB)

$(PROGRAM).a: $(OBJECTS)
    ar rcs $@ $^

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp *.hpp
```

Main routine (main.cpp):

```
// Copyright 2024 Wendy Carvalho
#include <fstream>
#include <iostream>
#include <regex>
#include <sstream>
#include <string>

#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

using boost::gregorian::date;
using boost::gregorian::from_simple_string;
using boost::posix_time::ptime;
using boost::posix_time::time_duration;

int main(int argc, char* argv[]) {
  std::string logFileName = argv[1];
  std::string rprtFileName, line, timestamp, tmp, day, serviceName;
  std::stringstream boots, services;

  std::map<std::string, std::stringstream> serviceMap;
  std::vector<std::string> serviceNames = {
      "Logging",           "DatabaseInitialize",
      "MessagingService",  "HealthMonitorService",
      "Persistence",       "ConfigurationService",
      "LandingPadService", "PortConfigurationService",
      "CacheService",      "ThemingService",
      "StagingService",    "DeviceIOService",
      "BellService",       "GateService",
      "ReaderDataService", "BiometricService",
      "StateManager",      "OfflineSmartviewService",
      "AVFeedbackService", "DatabaseThreads",
      "SoftLoadService",   "WATCHDOG",
      "ProtocolService",   "DiagnosticsService"};
  for (const auto& serviceName : serviceNames) {
    serviceMap[serviceName] = std::stringstream();
  }
  std::vector<std::string> notStarted;

  std::ifstream log(logFileName);

  std::string _date = "\\d{4}-\\d{2}-\\d{2}\\s";
  std::string _time = "\\d{2}:\\d{2}:\\d{2}[:.]";
  std::string _start = "\\s(.*log.c.166.*)";
  std::string _success =
      "\\d{3}:INFO:oejs.AbstractConnector:Started SelectChannelConnector";

  std::regex reStart(_date + _time + _start);
  std::regex reDone(_date + _time + _success);
  std::regex serviceStart(
      "Starting Service\\.\\s\\s([A-Za-z]+)\\s(\\d+(?:\\.\\d+)*)");
  std::regex serviceDone(
      "Service started "
      "successfully\\.\\s\\s([A-Za-z]+)\\s(\\d+(?:\\.\\d+)*)\\s(.(\\d+)\\s(ms)."
      ")");
  std::smatch match;

  if (!log.is_open()) {
    std::cout << "could not open file" << std::endl;
    exit(1);
  }
  int lineCnt = 1;
  int iniCnt = 0, sucCnt = 0;
  bool booting = false;
  ptime t1, t2;

  while (getline(log, line)) {
    if (std::regex_search(line, reStart)) {
      if (booting) {
        boots << "**** Incomplete boot ****\n\nServices\n";
        for (auto& each : serviceMap) {
          boots << "\t" << each.first << std::endl;
          if (each.second.str().empty()) {
            boots << "\t\tStart: Not started(" << logFileName << ")"
                  << std::endl;
            boots << "\t\tStart: Not completed(" << logFileName << ")"
                  << std::endl;
            boots << "\t\tElapsed Time:" << std::endl;
            notStarted.push_back(each.first);
          } else {
            boots << each.second.str();
          }
        }
        boots << "*** Services not successfully started: " << std::endl;
        for (auto& e2 : notStarted) {
          boots << e2 << ", ";
        }
        boots << "\n\n";
        for (const auto& serviceName : serviceNames) {
          serviceMap[serviceName] = std::stringstream();
        }
        notStarted.clear();
      }
      timestamp = line.substr(11, 8);
      day = line.substr(0, 10);
      date d(from_simple_string(day));
      ptime pt(d, time_duration(std::stoi(timestamp.substr(0, 2)),
                                std::stoi(timestamp.substr(3, 2)),
                                std::stoi(timestamp.substr(6, 2))));
      t1 = pt;
      boots << "=== Device boot ===" << std::endl;
      boots << lineCnt << "(" << logFileName << "): " << line.substr(0, 19)
            << " Boot Start" << std::endl;
      booting = true;
      iniCnt++;
    } else if (std::regex_search(line, match, serviceStart)) {
      if (booting) {
        serviceMap[match[1]] << "\t" << match[1] << std::endl;
        serviceMap[match[1]] << "\t\tStart: " << lineCnt << "(" <<
  logFileName
                             << ")" << std::endl;
      }
    } else if (std::regex_search(line, match, serviceDone)) {
      if (booting) {
        serviceMap[match[1]] << "\t\tCompleted: " << std::to_string(lineCnt)
                             << "(" << logFileName << ")" << std::endl;
        serviceMap[match[1]] << "\t\tElapsed Time: " << match[4] << " ms"
                             << "\n";
      }
    } else if (std::regex_search(line, reDone)) {
      if (booting) {
        timestamp = line.substr(11, 8);
        day = line.substr(0, 10);
        date d(from_simple_string(day));
        ptime pt(d, time_duration(std::stoi(timestamp.substr(0, 2)),
                                  std::stoi(timestamp.substr(3, 2)),
                                  std::stoi(timestamp.substr(6, 2))));
        t2 = pt;
        time_duration totalTime = t2 - t1;
        boots << lineCnt << "(" << logFileName << "): " << line.substr(0,
  19)
            << " Boot Completed" << std::endl;
        boots << "\tBoot Time: " << totalTime.total_milliseconds() << "ms\n"
            << std::endl;
        boots << "Services\n";
        for (auto& each : serviceMap) {
          boots << each.second.str();
        }
        for (const auto& serviceName : serviceNames) {
          serviceMap[serviceName] = std::stringstream();
        }
        booting = false;
        sucCnt++;
      } else {
        boots << "**** Incomplete boot ****\n\nServices\n";
        for (auto& each : serviceMap) {
          boots << "\t" << each.first << std::endl;
          if (each.second.str().empty()) {
            boots << "\t\tStart: Not started(" << logFileName << ")"
                  << std::endl;
            boots << "\t\tStart: Not completed(" << logFileName << ")"
                  << std::endl;
            boots << "\t\tElapsed Time:\n" << std::endl;
            notStarted.push_back(each.first);
          } else {
            boots << each.second.str();
          }
        }
        boots << "*** Services not successfully started: " << std::endl;
        for (auto& e2 : notStarted) {
          boots << e2 << ", ";
        }
        boots << "\n\n";
        for (const auto& serviceName : serviceNames) {
          serviceMap[serviceName] = std::stringstream();
        }
        notStarted.clear();
      }
    }
    lineCnt++;
  }

  std::ofstream report;
  report.open(logFileName + ".rpt");
  report << "Device Boot Report\n" << std::endl;
  report << "InTouch log file: " << logFileName << std::endl;
  report << "Lines Scanned: " << lineCnt - 1 << "\n" << std::endl;
  report << "Device boot count: initiated = " << iniCnt
         << ", completed: " << sucCnt << "\n\n"
         << std::endl;
  report << boots.rdbuf();

  report.close();
  log.close();

  return 0;
}
```