Christopher Nguyen

COMPIV Section 204: Project Portfolio

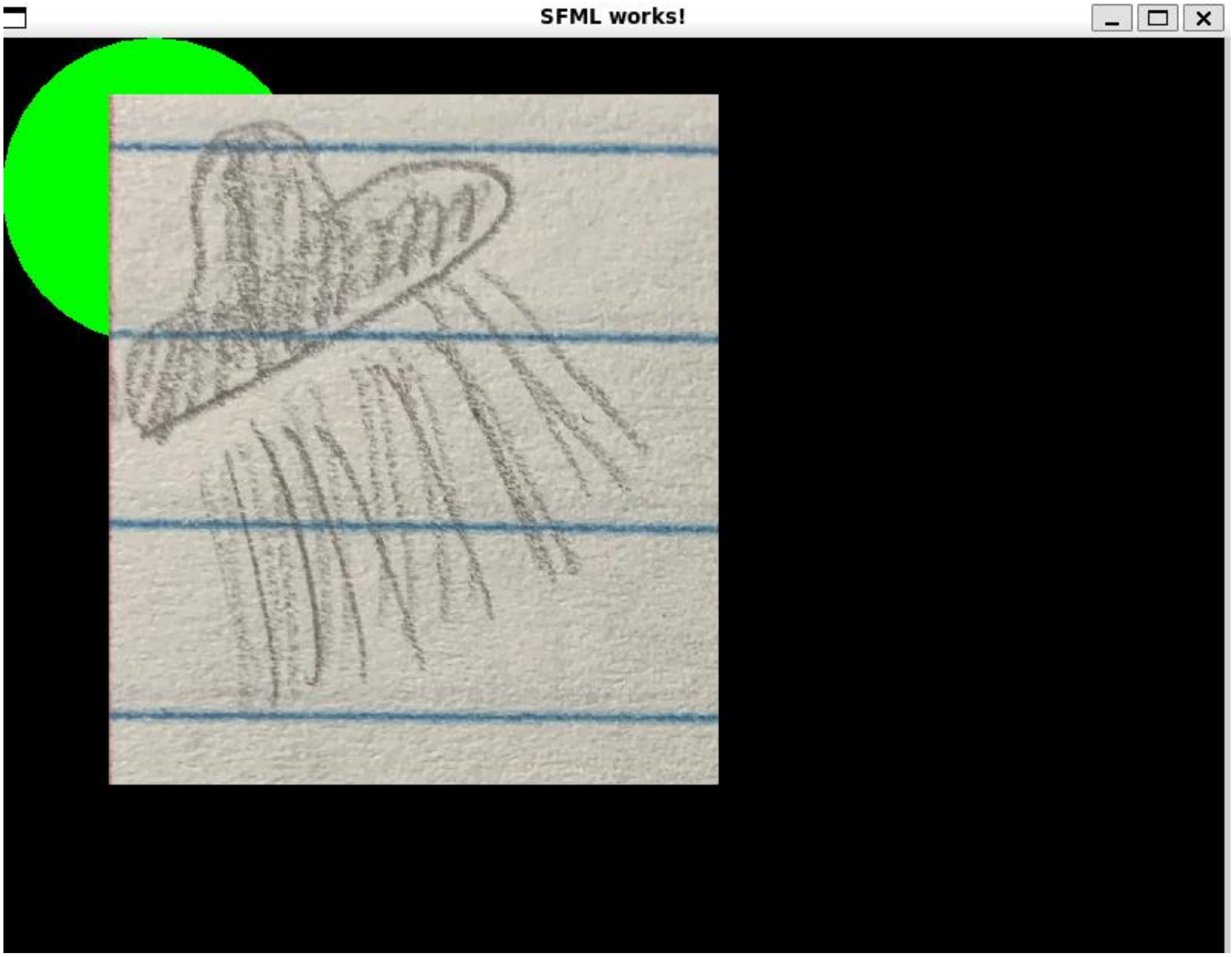
Spring 2025

Contents:

PS0 Hello SFML

This assignment is about showing my ability to set up Ubuntu through WSL and learn some features of SFML. The assignment required me to load a sprite, so I was able to load a sprite. This assignment also required me to move a sprite so if no buttons are pressed, the sprite will by default go lower from the screen. The sprite will move up, down, left, and right from the keyboard up arrow, down arrow, left arrow, and right arrow respectively.

The central thing crucial to complete the assignment is learning how to set up an SFML window. I also learned how to load a sprite and learned how to change colors of a shape. It is done by using the setFillColor. A screenshot of my code working below:



Code for main.cpp is below:

PS0 Hello SFML

This assignment is about showing my ability to set up Ubuntu through WSL and learn some features of SFML. The assignment required me to load a sprite, so I was able to load a sprite. This assignment also required me to move a sprite so if no buttons are pressed, the sprite will by default go lower from the screen. The sprite will move up, down, left, and right from the keyboard up arrow, down arrow, left arrow, and right arrow respectively.

The central thing to complete the assignment is learning how to set up an SFML window. I also learned how to load a sprite and learned how to change colors of a shape. It is done by using the setFillColor. A screenshot of my code working below:

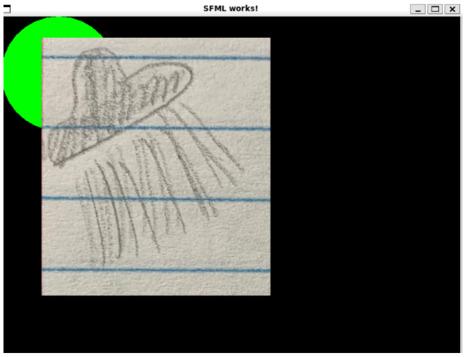


Figure 1: Image of the output of the program

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
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
```

Now, let's look at a simple C++ program that prints "Hello, world!" to the console.

Listing 2: main.cpp

```
// Copyright 2025 Christopher Nguyen
#include <SFML/Graphics.hpp>
int main() {
    sf::RenderWindow window(sf::VideoMode(800, 600), "
       ↪ SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

     // Load a sprite to display
    sf::Texture texture;
    if (!texture.loadFromFile("sprite.png"))
        return EXIT_FAILURE;
    sf::Sprite sprite(texture);

    while (window.isOpen()) {
        sf::Event event;
    while (window.pollEvent(event)) {
        if (event.type == sf::Event::Closed)
            window.close();
    }
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)
       ↪ ) {
    sprite.move(-5, 0);
    } else if (sf::Keyboard::isKeyPressed(sf::Keyboard
       ↪ ::Right)) {
    sprite.move(5, 0);
    } else if (sf::Keyboard::isKeyPressed(sf::Keyboard
       ↪ ::Up)) {
    sprite.move(0, -5);
    } else if (sf::Keyboard::isKeyPressed(sf::Keyboard
       ↪ ::Down)) {
    sprite.move(0, 5);
    }
    if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
        sprite.scale(0.5, 0.5);
    } else if (sf::Mouse::isButtonPressed(sf::Mouse::
       ↪ Right)) {
        sprite.scale(1.5, 1.5);
    }
       window.clear();
       window.draw(shape);
       window.draw(sprite);
       sprite.move(0, 1.f);
       window.display();
    }

    return 0;
}
```

PS1 LFSR with Photo Magic

I was able to use a seed and shift it and have position 0 based on the XORs of the taps. That process is called step. Generate returns a certain bit integer after a certain amount of steps have occurred. The LFSR is used to create random like bits.

I used the features from LFSR to be implemented for the transform function. This is how the encryption process works. For every pixel in the image, I was able to use the generate(8) three times for three new integers for the colors red, blue, and green. I then XORed the value of the colors with the three new int values that came from generate to change the color of each pixel. For the main function, I was able to set up two windows one for the input image and another for when altered image. I used argv values to call the three command-line arguments. One for the input, one for the encrypted image, and one for the seed. After altering the image, a new png file will be stored. Use the new png file with the same seed you can get the photo back into original form.

The encryption process works because when you after altering the image the seed will be a certain 16 bit. If you repeat the three command-line arguments with the stored altered image with the same seed as the first time you encrypt the image, the values of the 16 bit from generate after the XOR process will be the same value as the original image.

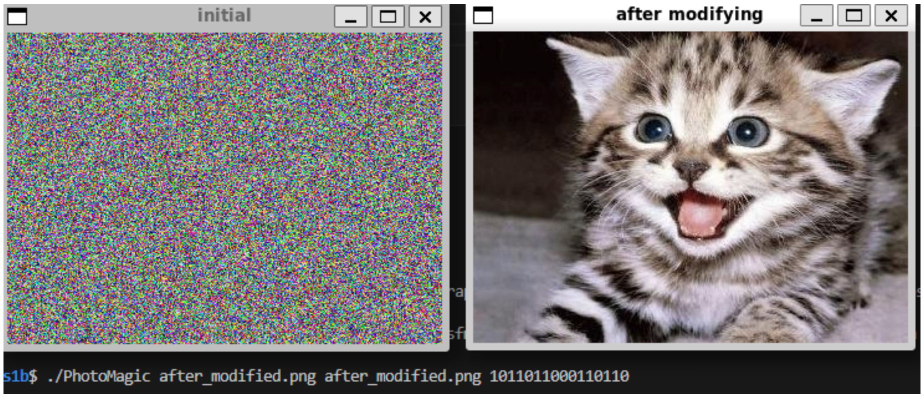
I used a bitset as a private field. I used it for the step operation because I used the ¡¡ operator to shift the bits and before that XOR the farthest position with the taps and stored it to position 0 of the bitset and return that value. It is much less work. involved than having a for loop and assigning the value of each index of the array to the next one.

For the FibLFSR constructor, I had to check for possible errors in the seed such as the seed containing a non binary or its not 16 bits. It will throw and invalid argument if either the two issues exists.

For the ostream operator, I was able to print out the seed of the FibLFSR object so whenever ¡¡ is loaded with the object, the seed will be printed out. By using the ostream operator, I won't need to be required to use getters to print out the seed of the object.

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
# Your .hpp files
DEPS = FibLFSR.hpp PhotoMagic.hpp
# Your compiled .o files
OBJECTS = FibLFSR.o test.o PhotoMagic.o
OBJ_PHOTO = main.o FibLFSR.o PhotoMagic.o
# The name of your program
PROGRAM = test
```



```
s1b$ ./PhotoMagic after_modified.png after_modified.png 1011011000110110
```

Figure 1: Image of the output of the program

```
PROGRAM2 = PhotoMagic

.PHONY: all clean lint


all: $(PROGRAM) $(PROGRAM2) PhotoMagic.a


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

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

PhotoMagic: $(OBJ_PHOTO)
        $(CC) $(CFLAGS) -o $@ $^ $(LIB)

# To create static library NOT including main.cpp
PhotoMagic.a: FibLFSR.o PhotoMagic.o
        ar rcs $@ $^

clean:
        rm *.o $(PROGRAM) $(PROGRAM2) PhotoMagic.a

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

Listing 2: main.cpp

```cpp
// Copyright 2025 <Christopher Nguyen>
#include <iostream>
#include "FibLFSR.hpp"
#include "PhotoMagic.hpp"
int main(int argc, char* argv[]) {
    using PhotoMagic::FibLFSR;
    sf::Image inPic;
    sf::Image outPic;
    FibLFSR l(argv[3]);
    FibLFSR ex("1010110011101010");

    std::cout << "after generate ex is " << ex.generate(50) << std::endl;
    std::cout << "after generate ex is " << ex.generate(50) << std::endl;

    if (!inPic.loadFromFile(argv[1])) {
        return -1;
    }
    if (!outPic.loadFromFile(argv[2])) {
        return -1;
    }
    PhotoMagic::transform(outPic, &l);
    sf::Vector2u size = inPic.getSize();
    // Window 1 is for inPic
    // Window 2 is for outPic
    sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "initial");
    sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "after modifying");

    // for inPic
    sf::Texture texture;
    texture.loadFromImage(inPic);
    sf::Sprite sprite;
    sprite.setTexture(texture);

    // for outPic
    sf::Texture texture1;
    texture1.loadFromImage(outPic);

    // for outPic
    sf::Sprite sprite1;
    sprite1.setTexture(texture1);

    outPic.saveToFile("after_modified.png");

    while (window1.isOpen() && window2.isOpen()) {
    sf::Event event;
    while (window1.pollEvent(event)) {
    if (event.type == sf::Event::Closed)
    window1.close();
    }
    while (window2.pollEvent(event)) {
    if (event.type == sf::Event::Closed)
    window2.close();
    }
    window1.clear();
    window1.draw(sprite);
    window1.display();
    window2.clear();
    window2.draw(sprite1);
    window2.display();
    }

    std::cout << "Hey chris!\n";
    return 0;
}
```

Listing 3: Photomagic.hpp

```
// Copyright 2025 <Christopher Nguyen>
#pragma once
#ifndef PHOTOMAGIC_H
#define PHOTOMAGIC_H
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include "FibLFSR.hpp"

namespace PhotoMagic {
void transform(sf::Image& img, FibLFSR* lfsr);
}

#endif
```

Listing 4: Photomagic.cpp

```
// Copyright 2025 <Christopher Nguyen>
#include "PhotoMagic.hpp"
```

```cpp
#include "FibLFSR.hpp"
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
using PhotoMagic::FibLFSR;
namespace PhotoMagic {
void transform(sf::Image& img, FibLFSR* lfsr) {
    sf::Vector2u size = img.getSize();
    sf::Color p;
    int new_int, new_int_g, new_int_b;
    for (unsigned int x = 0; x < size.x; x++) {
        for (unsigned int y = 0; y < size.y; y++) {
            new_int = lfsr->generate(8);
            new_int_g = lfsr->generate(8);
            new_int_b = lfsr->generate(8);
            p = img.getPixel(x, y);
            p.r = p.r ^ new_int;
            p.g = p.g ^ new_int_g;
            p.b ^= new_int_b;
            img.setPixel(x, y, p);
        }
    }
}
}  // namespace PhotoMagic
```

Listing 5: FibLFSR.hpp

```cpp
// Copyright 2025 <Christopher Nguyen>
#pragma once
#ifndef FIBLFSR_H
#define FIBLFSR_H
#include <bitset>
#include <iostream>
#include <stdexcept>
#include <string>

namespace PhotoMagic {
class FibLFSR {
 public:
  explicit FibLFSR(const std::string &seed);
  explicit FibLFSR(unsigned int seed);  // Optional
  static FibLFSR fromPassword(const std::string &password);  // Optional

  int step();
  int generate(int k);
  friend std::ostream &operator<<(std::ostream &out,
     ↪ const FibLFSR &lfsr);

 private:
  std::string the_seed;
  std::bitset<16> b;
};

}  // namespace PhotoMagic
#endif
```

Listing 6: FibLFSR.cpp

```
// Copyright 2025 <Christopher Nguyen>
#include "FibLFSR.hpp"
#include <string>
namespace PhotoMagic {
FibLFSR::FibLFSR(const std::string& seed) {
        if (seed.length() != 16) {
            throw std::invalid_argument("length must
               ↪ be 16 bits");
        }
        for (char c : seed) {
            if (c != '0' && c!= '1') {
            throw std::invalid_argument("must consist
               ↪ of only 0s and 1s");
            }
        }
        the_seed = seed;
        b = std::bitset<16>(the_seed);
}

int FibLFSR::step() {
        int result = b[15]^b[13]^b[12]^b[10];
        b<<=1;
        b[0] = result;
        return result;
}

int FibLFSR::generate(int k) {
    if (k <= 0) {
        throw std::invalid_argument("invalid, must be
           ↪ greater than 0");
    }
    int var = 0;
    for (int i = 0; i < k; i++) {
        var = (var << 1)|step();
    }
    return var;
}

std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {
    out << lfsr.b;
    return out;
}
}  // namespace PhotoMagic
```

Listing 7: FibLFSR.cpp

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

#include <iostream>
#include <string>
#include <stdexcept>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include "./FibLFSR.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);
}
```

```cpp


// Own test cases are below this comment

/* Checking to see if this fails because k
(-1 in this case) because k has to be greater than 0 */
BOOST_AUTO_TEST_CASE(test1) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_THROW(l.generate(-1), std::invalid_argument);
}

// Tests if the string has invalid characters such as 2
BOOST_AUTO_TEST_CASE(test2) {
  BOOST_REQUIRE_THROW(FibLFSR l("1011011000110120"), std::invalid_argument);
}

/* Tests the ostream operator if it
 matches 0010110110001101 */
BOOST_AUTO_TEST_CASE(test3seedmatch) {
  FibLFSR l("0010110110001101");
  std::string eS = "0010110110001101";

  std::ostringstream oss;
  oss << l;
  std::string tS = oss.str();

  BOOST_REQUIRE_EQUAL_COLLECTIONS(tS.begin(), tS.end(), eS.begin(), eS.end());
}

/* Test object if it the string
length is not 16 is supposed to
throw an invalid argument */
BOOST_AUTO_TEST_CASE(test4) {
  BOOST_REQUIRE_THROW(FibLFSR l("1"), std::invalid_argument);
}
```

PS2 Triangle Fractal with recursion

To create a triangle, I used the CircleShape class. Then I used set origin to be the center of the triangle rather than the top left corner. I used setPointCount to make the CircleShape object to draw a triangle.

I calculated the radius of the triangle by dividing the length of the triangle, by 2cos30 where 30 is 30 degrees. The height of the triangle is calculated as cotangent of 30 degrees times the radius. F is the difference between the height and the radius of the base triangle. For the x position, I computed the positions of the top of the child triangle by using the x position of the base triangle which can be acquired by using getPosition().x, then subtract it by the length of the side of the child triangle divided by 2. For the y position of the top child triangle, I used the the y position of the base triangle which can be acquired by using getPosition().y, then subtract it by the height of the base triangle divided by 2, then subtracted it by the F of the child triangle.

To calculate the x position of the bottom left triangle, I acquired the x position of the base triangle using getPosition().x then subtract it by the quotient of the length of the side of the base triangle divided by 2. To calculate the y position of the bottom left triangle, I acquired the y position of the base triangle using getPosition().y then add it with the radius of the base triangle.

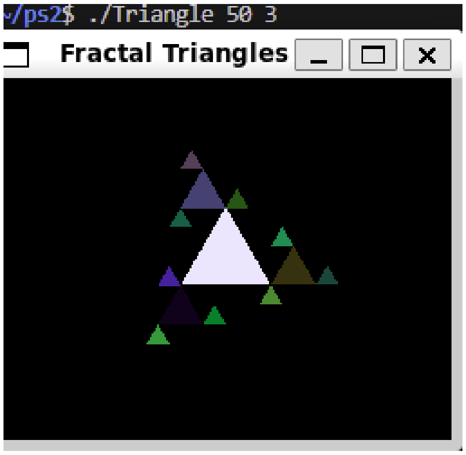
To calculate the x position of the bottom right triangle, I acquired the x position of the base triangle using getPosition().x then add it by the quotient of the length of the size of the side of the triangle divided by 2, then add the length of the side of the child triangle divided by 2. To calculate the y position of the bottom right triangle, I acquired the y position of the base triangle using getPosition().y then add it with the quotient of the F value of the base triangle divided by 2.

I used drawTriangle as a helper function to draw the base triangle, then set the positions of the top, bottom left, and bottom right triangles. Inside the helper function, I called the drawTriangle within the function with the top, bottom left, and bottom right triangles as one argument for each drawTriangle calls. If the depth is not 0, it will draw the triangle from the argument then set the positions of the next three triangles. This process is done by a key Computer Science concept of recursion.

I initially had trouble with how to move the three child triangles from the base. I was provided with advice to one, create 3 triangle objects for top, bottom left, and bottom right and use set position to move the three child triangles from the base to its appropriate destinations then call drawTriangle for each direction.

I wanted to have a color pattern where each base triangle has a different color from the child triangle. I used generate from FibLFSR class to change the color but it had a different output than I expected it to be. The triangles keeps changing colors while the window is open.

I put color on the triangles. When the window is displayed, it will make an animation design where all the triangles will keep changing colors. This is considered to be an animation design because the triangles will keep changing colors as long as the window is open. I used FibLFSR from ps1 to help me generate random numbers to use to set the fillColors for the triangle object.



```
~/ps2$ ./Triangle 50 3
```

Figure 1: image of Sierpinski triangle

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
# Your .hpp files
DEPS = triangle.hpp FibLFSR.hpp
# Your compiled .o files
OBJECTS = main.o triangle.o FibLFSR.o
# The name of your program
PROGRAM = Triangle

.PHONY: all clean lint


all: $(PROGRAM)

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

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

```

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

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

Now, let's look at a simple C++ program that prints ”Hello, world!” to the console.

Listing 2: main.cpp

```
// Copyright 2025 <Christopher Nguyen>
#include <iostream>
#include <cmath>
#include "triangle.hpp"
#include "FibLFSR.hpp"
using PhotoMagic::FibLFSR;

int main(int argc, char *argv[]) {
    int x_window = std::stoi(argv[1])* 4 + 50;
    int y_window = std::stoi(argv[1])* 4;
    FibLFSR l("1010110011101010");
    sf::RenderWindow window1(sf::VideoMode(x_window,
       ↪ y_window), "Fractal Triangles");
    while (window1.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window1.close();
        }

        window1.clear();
        fractal(window1, std::stoi(argv[1]), std::stoi
           ↪ (argv[2]), &l);
        window1.display();
    }

    return 0;
}
```

Listing 3: triangle.hpp

```
// Copyright 2025 <Christopher Nguyen>
#pragma once
#ifndef TRIANGLE_H
#define TRIANGLE_H
#include <iostream>
```

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

using PhotoMagic::FibLFSR;
void fractal(sf::RenderTarget& window, double length,
    ↪ int d, FibLFSR* obj);
void drawTriangle(sf::RenderTarget& window, double
    ↪ length, int d, sf::CircleShape t, FibLFSR* obj);

#endif
```

Listing 4: triangle.cpp

```
// Copyright 2025 <Christopher Nguyen>
#include <iostream>
#include <cmath>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include "triangle.hpp"
#include "FibLFSR.hpp"
#define _USE_MATH_DEFINES  // need this
// for M_PI
using PhotoMagic::FibLFSR;
void fractal(sf::RenderTarget& window, double length,
    ↪ int d, FibLFSR* obj) {
    if (d <= 0) return;
    sf::CircleShape triangle;
    // convert 30 degrees to radians
    sf::Vector2u size = window.getSize();
    float radians = (30 * M_PI)/ 180;
    // rad is radius of circle
    float rad = ((length)/(2*cos(radians)));
    triangle.setRadius(rad);
    triangle.setOrigin(rad, rad);
    triangle.setPosition((size.x/2), (size.y/2));
    triangle.setPointCount(3);
    window.draw(triangle);
    drawTriangle(window, length, d, triangle, obj);
}

void drawTriangle(sf::RenderTarget& window, double
    ↪ length, int d, sf::CircleShape t, FibLFSR* obj)
    ↪ {
    if (d == 0) return;
    sf::CircleShape tA;
    sf::CircleShape tB;
    sf::CircleShape tC;

    int result = obj->generate(50);
    result = (result > 0)? result: -result;

    t.setFillColor(sf::Color(result));

    // convert 30 degrees to radians
    float radians = (30 * M_PI)/ 180;
    // rad is radius of circle
    float rad = ((length)/(2*cos(radians)));
    t.setRadius(rad);
    t.setOrigin(rad, rad);
    t.setPointCount(3);
    window.draw(t);

    float h = (1/tan(radians)) * rad;
    float f = h - rad;
    float c_size = length/2;
    // for top
    float r_small = c_size/sqrt(3);
    float child_h = (1/tan(radians)) * r_small;
    float child_f = child_h - r_small;
    float c_half = c_size/2.0;

    float x_tA = t.getPosition().x - c_half;
    float x_tC = t.getPosition().x + c_size + c_half;

    tA.setPosition(x_tA, t.getPosition().y - h/2 -
        ↪ child_f);
    tB.setPosition(t.getPosition().x-c_size, t.
        ↪ getPosition().y + rad);
    tC.setPosition(x_tC, t.getPosition().y + (f/3));

    drawTriangle(window, c_size, d-1, tA, obj);
    drawTriangle(window, c_size, d-1, tB, obj);
    drawTriangle(window, c_size, d-1, tC, obj);

    t.setFillColor(sf::Color(result));
}
```

Listing 5: FibLFSR.hpp

```cpp
// Copyright 2025 <Christopher Nguyen>
#pragma once
#ifndef FIBLFSR_H
#define FIBLFSR_H
#include <bitset>
#include <iostream>
#include <stdexcept>
#include <string>

namespace PhotoMagic {
class FibLFSR {
 public:
  explicit FibLFSR(const std::string &seed);
  explicit FibLFSR(unsigned int seed);  // Optional
  static FibLFSR fromPassword(const std::string &
      ↪ password);  // Optional

  int step();
  int generate(int k);
  friend std::ostream &operator<<(std::ostream &out,
      ↪ const FibLFSR &lfsr);

 private:
  std::string the_seed;
  std::bitset<16> b;
};

}  // namespace PhotoMagic
#endif
```

Listing 6: FibLFSR.cpp

```cpp
// Copyright 2025 <Christopher Nguyen>
#include "FibLFSR.hpp"
#include <string>
namespace PhotoMagic {
FibLFSR::FibLFSR(const std::string& seed) {
        if (seed.length() != 16) {
            throw std::invalid_argument("length must
                ↪ be 16 bits");
        }
        for (char c : seed) {
            if (c != ’0’ && c!= ’1’) {
            throw std::invalid_argument("must consist
                ↪ of only 0s and 1s");
            }
        }
        the_seed = seed;
        b = std::bitset<16>(the_seed);
}

int FibLFSR::step() {
        int result = b[15]^b[13]^b[12]^b[10];
        b<<=1;
        b[0] = result;
        return result;
}

int FibLFSR::generate(int k) {
    if (k <= 0) {
        throw std::invalid_argument("invalid, must be greater than 0");
    }
    int var = 0;
    for (int i = 0; i < k; i++) {
        var = (var << 1)|step();
    }
    return var;
}

std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {
    out << lfsr.b;
    return out;
}
}  // namespace PhotoMagic
```

PS3 N-Body Simulation

Part A reads a text file, the first two lines of the text file is the number of celestial bodies and the second line is the radius of the universe. The first two inputs are stored to the Universe class. For every number of celestial bodies, a Celestial body will be called. That means that each subsequent lines will have the data of the body's features. In the main, there is a universe object called obj. In the while loop, it will display the celestial bodies with the given positions scaled to fit the window.

The step function gets defined. What it does is takes a time parameter in seconds and moves each CelestialBody based on the Leapfrog method which is to find the pairwise forces, use the sums to fin the net force. Find the net force of the x and y direction which is Fx and Fy respectively. Then use Fx and Fy to find the acceleration of the x and y directions given as ax and ay. Then use ax and ay to find the new velocity. Then use the new velocity and the time and the old position to find the new positions of the x and y directions. Then sets the positions and velocity of the x and y directions of each celestial body.

When it's implemented in the main, the planets should revolve around the sun and will keep moving until dt (time in seconds) is greater than T (the maximum allocated total time).

I used the provided sf::Velocity2f position function from the CelestialBody class. It is used to get the x and y positions of the celestial body and the position function is used in the draw function of the Universe.cpp to set position of a celestial body. In the extraction operator in the Universe.cpp file, I used a for loop that is of length of the number of celestial bodies and for each iteration it will be of the extration operator of the CelestialBody object. By doing that, it will read each line of each CelestialBody and provide its properties.

To make the data from the files fit the size of the window, I used the length of the window, divided by 2* the provided radius of the .txt file

To set the new x position, new y position, new x velocity, and new y velocity after the step function is applied to the CelestialBody, I made a setter function called set CB for CelestialBody which took new x position, new y position, new x velocity, and new y velocity as its parameters.

I used the command lines for T and dt with argv[1] being T and argv[2] being dt. In the while loop, the planets will keep revolving and the current will be incremented by dt. I will keep moving until current is greater than T and then the window will close then it will output the features of each celestial body in the universe.

In the extraction overload function in the CelestialBody.cpp file, a make shared ptr is stored to texture. I then set the Sprite to be the deferenced value of the texture. Because the texture is a shared ptr, the sprite and texture are accessing the same memory address and the risk of having sprites being white square are mitigated.

I initially had an issue where all the Celestial bodies loaded in the window but were all in the same place. This is because I made a copy of states stored as t but called the window.draw() as window.draw(body, state) instead of window.draw(body, state). I also learned how there is a way to set a position of a transform object which is different from setting a position of a sprite object. I also had an error where I forgot to initialize a new body when iterating through the loop. The initialization took place before the for loop which means that Universe would only accept one celestial body. I fixed the issue by putting the Celestial body inside the for loop so it will output many celestial bodies in the window.

Additionally, in Gradescope, I got points reduced because I got tests that failed. These tests explained that it failed to read x-pos from ostream even though the insertion operator from the CelestialBody class reads it by means of uni.x pos. I found out the solution was in the Universe insertion overload function. I applied CelestialBody overloading inside the Universe insertion operator.

I had issues with how to store Fx and Fy the same element of a vector. I figured out you need to do a vector pair. Additionally, I had issues that the celestial bodies didn't move even though I modified step to what the instructions say and that step has been called in main. I figured out you call step in the while loop.

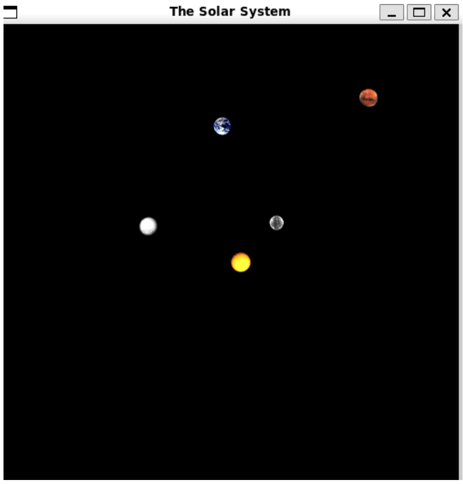


Figure 1: Image of the Universe with planets revolving around the Sun

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework

# Your .hpp files
DEPS = CelestialBody.hpp Universe.hpp
```

```makefile

# Your compiled .o files
TEST_OBJ = test.o CelestialBody.o Universe.o
MAIN_OBJ = main.o CelestialBody.o Universe.o

# The name of the program
PROGRAM_T = test
PROGRAM = NBody

.PHONY: all clean lint

all: $(PROGRAM_T) $(PROGRAM) NBody.a

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

test: $(TEST_OBJ)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)

NBody: $(MAIN_OBJ)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)

NBody.a: CelestialBody.o Universe.o
	ar rcs $@ $^

clean:
	rm *.o $(PROGRAM_T) $(PROGRAM) NBody.a

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

Listing 2: main.cpp

```cpp
// Copyright 2025 <Christopher Nguyen>
#include <iostream>
#include <string>
#include <SFML/Graphics.hpp>
#include "CelestialBody.hpp"
#include "Universe.hpp"
using NB::CelestialBody;
using NB::Universe;
int main(int argc, char* argv[]) {
  double BigT = std::stod(argv[1]);
  double dt = std::stod(argv[2]);
  double current = 0.0;
  Universe obj;
  std::cin >> obj;

  sf::RenderWindow window(sf::VideoMode(500, 500), "
      ↪ The Solar System");
  sf::View view(sf::Vector2f(0.f, 0.f),
                sf::Vector2f(window.getSize().x,
                    ↪ window.getSize().y));
  window.setView(view);
  while (window.isOpen() && BigT >= current) {
    sf::Event event;
    while (window.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
        window.close();
    }

    window.clear();
    window.draw(obj);
    obj.step(std::stod(argv[2]));
    current += dt;
    window.display();
  }

  std::cout << obj << std::endl;
  return 0;
}
```

Listing 3: Universe.hpp

```cpp
// Copyright 2025 <Christopher Nguyen>
#pragma once

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

namespace NB {
class Universe : public sf::Drawable {
 public:
  Universe();                                          //
      ↪ Required
  explicit Universe(const std::string &filename);  //
      ↪ Optional

  size_t size() const;   // Optional
  double radius() const;  // Optional

  const CelestialBody &operator[](size_t i) const;  // Optional

  void step(double dt);  // Implemented in part b, behavior for part a is undefined

  friend std::istream &operator>>(std::istream &is, Universe &uni);
  friend std::ostream &operator<<(std::ostream &os, const Universe &uni);

 protected:
  void draw(sf::RenderTarget &window, sf::RenderStates states) const override;
  // From sf::Drawable
 private:
  // Fields and helper functions go here
  int planets;
  std::vector<CelestialBody> p;
  float uni_rad;
};

}  // namespace NB
```

Listing 4: Universe.cpp

```
// Copyright 2025 <Christopher Nguyen>
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include "Universe.hpp"

using NB::CelestialBody;
namespace NB {

const double GRAVITY_C = 6.67e-11;

Universe::Universe() : planets(0), uni_rad(0.0) { p.resize(planets); }

// void Universe::step(double dt) {
```

```cpp
//   double dx, dy, dr2,
//   F, Fx, Fy, Fnum, dr,
//   Fx_total, Fy_total,
//   ax, ay, vxI, vyI,
//   pxI, pyI;
//   for (int i = 0; i < planets; i++) {
//     std::vector<std::pair<double, double>> CB_F;
//     Fx_total = 0.0; Fy_total = 0.0;
//     for (int j = 0; j < planets; j++) {
//       if (i == j) continue;
//       //  For net force
//       dx = p[j].position().x - p[i].position().x;
//       dy = p[j].position().y - p[i].position().y;

//       dr2 = dx*dx + dy*dy;     dr = sqrt(dr2);
//       Fnum = GRAVITY_C* p[i].mass()*p[j].mass();
//       F = Fnum/dr2;    Fx = F*dx/dr;
//       Fy = F*dy/dr;
//       CB_F.push_back(std::make_pair(Fx, Fy));
//     }

//     for (auto& cb : CB_F) {
//       Fx_total += cb.first;
//       Fy_total += cb.second;
//     }

//     ax = Fx_total/p[i].mass();
//     ay = Fy_total/p[i].mass();

//     vxI = p[i].velocity().x + dt*ax;
//     vyI = p[i].velocity().y + dt*ay;

//     pxI = p[i].position().x + dt*vxI;
//     pyI = p[i].position().y + dt*vyI;

//     p[i].set_CB(pxI, pyI, vxI, vyI);
//   }
// }

void Universe::step(double dt) {
  // Variables to store forces and accelerations
  double dx, dy, dr2, dr, F, Fx, Fy, Fnum;
  double Fx_total, Fy_total, ax, ay;

  // Arrays to store the updated velocities and positions
  std::vector<std::pair<double, double>> new_velocities(planets);
  std::vector<std::pair<double, double>> new_positions(planets);

  for (int i = 0; i < planets; i++) {
    Fx_total = 0.0;
    Fy_total = 0.0;

    for (int j = 0; j < planets; j++) {
      if (i == j) continue;

      dx = p[j].position().x - p[i].position().x;
      dy = p[j].position().y - p[i].position().y;
      dr2 = dx * dx + dy * dy;
      dr = sqrt(dr2);

      Fnum = GRAVITY_C * p[i].mass() * p[j].mass();
      F = Fnum / dr2;
      Fx = F * dx / dr;
      Fy = F * dy / dr;

      Fx_total += Fx;
      Fy_total += Fy;
    }

    ax = Fx_total / p[i].mass();
    ay = Fy_total / p[i].mass();

    new_velocities[i].first = p[i].velocity().x + dt * ax;
    new_velocities[i].second = p[i].velocity().y + dt * ay;
  }

  for (int i = 0; i < planets; i++) {
    new_positions[i].first = p[i].position().x + dt * new_velocities[i].first;
    new_positions[i].second = p[i].position().y + dt * new_velocities[i].second;

    p[i].set_CB(new_positions[i].first, new_positions[i].second,
                new_velocities[i].first, new_velocities[i].second);
  }
```

```
}


Universe::Universe(const std::string &filename) {
  std::ifstream is(filename);
  is >> planets >> uni_rad;
  for (int i = 0; i < planets; i++) {
    CelestialBody body;
    is >> body;
    p.push_back(body);
  }
}

std::istream &operator>>(std::istream &is, Universe &uni) {
  is >> uni.planets >> uni.uni_rad;
  for (int i = 0; i < uni.planets; i++) {
    CelestialBody body;
    is >> body;
    uni.p.push_back(body);
  }
  return is;
}

std::ostream &operator<<(std::ostream &os, const Universe &uni) {
  os << uni.planets << std::endl << uni.uni_rad <<std::endl;
  for (const auto& bod : uni.p) {
    os << bod << std::endl;  // p is a vector holding Celestial Bodies
  }
  return os;
}

void Universe::draw(sf::RenderTarget &window, sf::RenderStates states) const {
  double scale_num = window.getSize().x;
  double s_dem = 2.0 * uni_rad;
  double scale_f = scale_num / s_dem;
  for (const auto &body : p) {
    sf::RenderStates t = states;
    sf::Transform transform;
    transform.translate(body.position().x * scale_f,
                        -body.position().y * scale_f);
    t.transform = transform;
    window.draw(body, t);  // used to delegate each celestial body
  }
}

}  // namespace NB
```

Listing 5: CelestialBody.hpp

```
// Copyright 2025 <Christopher Nguyen>
#pragma once

#include <iostream>
#include <memory>
#include <string>

#include <SFML/Graphics.hpp>

namespace NB {
class CelestialBody : public sf::Drawable {
 public:
    explicit CelestialBody();  // Required

    sf::Vector2f position() const;  // Optional
    sf::Vector2f velocity() const;  // Optional
    float mass() const;            // Optional
    friend std::istream &operator>>(std::istream &is, CelestialBody &uni);
    friend std::ostream &operator<<(std::ostream &os, const CelestialBody &uni);

    void set_CB(double x, double y, double v_x, double v_y);

 protected:
    void draw(sf::RenderTarget &window, sf::RenderStates states) const override;
    // From sf::Drawable
 private:
    // Fields and helper methods go here
    double x_pos;
    double y_pos;
    double x_vel;
    double y_vel;
    double c_mass;
    std::string img;
    sf::Sprite sprite;
    std::shared_ptr<sf::Texture> texture;
};

}  // namespace NB
```

Listing 6: CelestialBody.cpp

```
// Copyright 2025 <Christopher Nguyen>
#include <iostream>
#include <string>
#include "CelestialBody.hpp"
#include <SFML/Graphics.hpp>

namespace NB {

CelestialBody::CelestialBody()
: x_pos(0.0), y_pos(0.0), x_vel(0.0), y_vel(0.0), c_mass(0.0), img("") {}

sf::Vector2f CelestialBody::position() const {
  return sf::Vector2f(x_pos, y_pos);
}

sf::Vector2f CelestialBody::velocity() const {
  return sf::Vector2f(x_vel, y_vel);
}

float CelestialBody::mass() const { return c_mass;}

void CelestialBody::set_CB(double x, double y, double v_x, double v_y) {
  x_pos = x; y_pos = y;
  x_vel = v_x; y_vel = v_y;
}

void CelestialBody::draw(sf::RenderTarget &window,
sf::RenderStates states) const {
window.draw(sprite, states);
}

std::istream &operator>>(std::istream &is, CelestialBody &uni) {
  is >> uni.x_pos >> uni.y_pos >> uni.x_vel >> uni.y_vel >> uni.c_mass >>
      uni.img;
  uni.texture = std::make_shared<sf::Texture>();
  if (!uni.texture->loadFromFile(uni.img))
    exit(1);
  uni.sprite = sf::Sprite(*uni.texture);
  return is;
}

std::ostream &operator<<(std::ostream &os, const CelestialBody &uni) {
  os << uni.x_pos << " " << uni.y_pos << " " << uni.x_vel << " " << uni.y_vel
     << " " << uni.c_mass << " " << uni.img;
  return os;
}

}  // namespace NB
```

Listing 7: test.cpp

```
// Copyright 2025 <Christopher Nguyen>
#include <iostream>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include "CelestialBody.hpp"
#include "Universe.hpp"
#include <boost/test/unit_test.hpp>
using NB::CelestialBody;
using NB::Universe;

BOOST_AUTO_TEST_CASE(test_CelestialBody) {
  std::istringstream is(" 1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24    earth.gif");
  CelestialBody body;
  is >> body;

  std::ostringstream output;
  output << body;

  BOOST_REQUIRE_EQUAL(output.str(),
   "1.496e+11 0 0 29800 5.974e+24 earth.gif");
}

BOOST_AUTO_TEST_CASE(test_CB_not_planets_txt) {
    std::istringstream is("1.496e11 0.000e00 0.000e00 2.980e04 5.974e24 electron.png");
    CelestialBody body;
    is >> body;

    std::ostringstream output;
    output << body;

    BOOST_REQUIRE_EQUAL(output.str(),
    "1.496e+11 0 0 29800 5.974e+24 electron.png");
}

BOOST_AUTO_TEST_CASE(CBs_after_animation) {
    double dt = 0;
    std::string expected_output =
        "5\n2.5e+11\n"
        "1.49584e+11 -2.14338e+09 427.029 29797.3
           ↪ 5.974e+24 earth.gif\n"
        "-2.21799e+11 -4.77345e+10 5080.04 -23669
           ↪ 6.419e+23 mars.gif\n"
        "3.56998e+10 4.56052e+10 -37672.9 29568.4
           ↪ 3.302e+23 mercury.gif\n"
        "597179 6.22961e+06 -0.0584217 0.163719 1.989e
           ↪ +30 sun.gif\n"
        "-7.49384e+10 -7.78189e+10 25226.6 -24335.5
           ↪ 4.869e+24 venus.gif\n";

    std::string planets_text = "5\n"
    "2.50e+11\n"
    "1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04
       ↪ 5.9740e+24    earth.gif\n"
    "2.2790e+11  0.0000e+00  0.0000e+00  2.4100e+04
       ↪ 6.4190e+23     mars.gif\n"
    "5.7900e+10  0.0000e+00  0.0000e+00  4.7900e+04
       ↪ 3.3020e+23  mercury.gif\n"
    "0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00
       ↪ 1.9890e+30      sun.gif\n"
    "1.0820e+11  0.0000e+00  0.0000e+00  3.5000e+04
       ↪ 4.8690e+24    venus.gif\n";
    std::istringstream is(planets_text);
    std::ostringstream output;
    Universe uni;
    is >> uni;
    while (dt <= 31557600) {
        uni.step(1000.0);
        dt+=1000;
    }
    output << uni;

    std::cout << output.str() << std::endl;
    BOOST_REQUIRE_EQUAL(output.str(), expected_output)
        ↪ ;
}
```

PS4 Sokoban

The extraction function takes in a file to read the width and height of the level and it will set the height and width of the level to what is provided on the file. Depending on the character, the Sokoban draw function will draw a particular picture. The end result is after a level file is read, it will display a level which contains walls, the floor, and the crates displayed based on the level file.

Part B is to have the player move across the map. The player can only push the box and the tile adjacent to the box must be clear for the player and the box to move. There is a isWon() that checks the two winning conditions, either all boxes have been placed on storage area spaces or all storage areas contains boxes. If you beat the level, a celebratory message appears and you cannot move the player but are allowed to reset. When R is pressed it resets meaning that level goes back to the original grid from the text file.

I used a setLoc function to set the private fields of the x and y locations of the player. I used the x and y locations for the getLoc function. I used a vector that holds a vector of strings to keep track of the level tiles. The 2d vector is useful because accessing it is based on the double for loop and accessing an element of a 2d vector is managable for me. It is also dynamic so there is no need to worry about fixed size.

Additionally, I made several textures for each required image that is needed for each level. This is used to load every image that is needed for Part a, and when a certain image is needed, a setTexture is set to that image. I loaded the textures in the constructor. This was implemented because when I set textures in other files, the textures will be already defined.

The extraction operator is used to read a level file and store the elements for its height and width and the locations and the type of the tiles.

The insertion operator is used to print out the level data from the level file. It uses the 2d vector to print it out because the 2d vector is used to represent the grid of the level.

I added a private field of type vector vector named original v. It is used to represent the grid of the inital map of the level file. To reset the level, the 2d vector v is set to the contents of original v.

For the movePlayer function, I had to check if the tile adjacent of the player has a block or a wall. The walls will block movement so it cannot move. If there is 2 blocks next to each other and are parallel the player cannot move. This is represent by setting the next values of the v vector to be the element that represents the character. Depending on each movement there is a direction. Based on the direction, the player's image will move based on the direction. I had cases that move the player based on each direction.

To prevent the player from moving after beating the level, I had to set a function named postWin() that sets afterWin to true. In the movePlayer function, the first thing was to check if the afterWin is true. If it's true, it will just automatically return meaning that once the player beats a level, the player cannot move.

To store the textures, I used smart pointers so all the sprites (many are duplicates) will be able to be referenced to the same texture.

I used the count if algorithm. It is used to track the amount of a's, A's, and 1's which are then used to check if the player beat the level. I used a lambda inside each count if algorithm. each lambda returns a predicate that checks if each index is the right character.

I had issues with figuring out how to spawn different images based on different characters from the level data. I learned that I needed to make different textures for each unique image as one step to spawn different images. It is used to set different textures based on the character.

I initially had issues with moving through a storage area. Once I pass the storage area, the storage area would disappear. I learned that I can use the original v and check if the the index at original v was a storage area. If true, the v tile will be set to a, the storage area. That way the storage area will appear once I leave it.

I had issues with the sound playing every frame. This is because every time the player beats a level, the sound will always play. To resolve this solution, I had to check for two conditions, if isWon() is true (if the player beat the level) and sound flag is false, it will play the sound only once and set sound flag to true.

For extra credit I had the window the player is able to change directions while moving. This is done by making a private field of type direction called player dir and for each case in the movePlayer function, player dir will be initialized to the respective direction and keyboard input. In the draw function the player's direction has a switch statement that sets a sprite to the appropriate texture that gives the player its direction.

I also loaded sound to ps4b. The purpose of this is to play the sound once the player beats the level.

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework

# Your .hpp files
DEPS = Sokoban.hpp

# Your compiled .o files
MAIN_OBJ = main.o Sokoban.o
TEST_OBJ = test.o Sokoban.o

# The name of the program
PROGRAM = Sokoban
TEST_PROGRAM = test

```

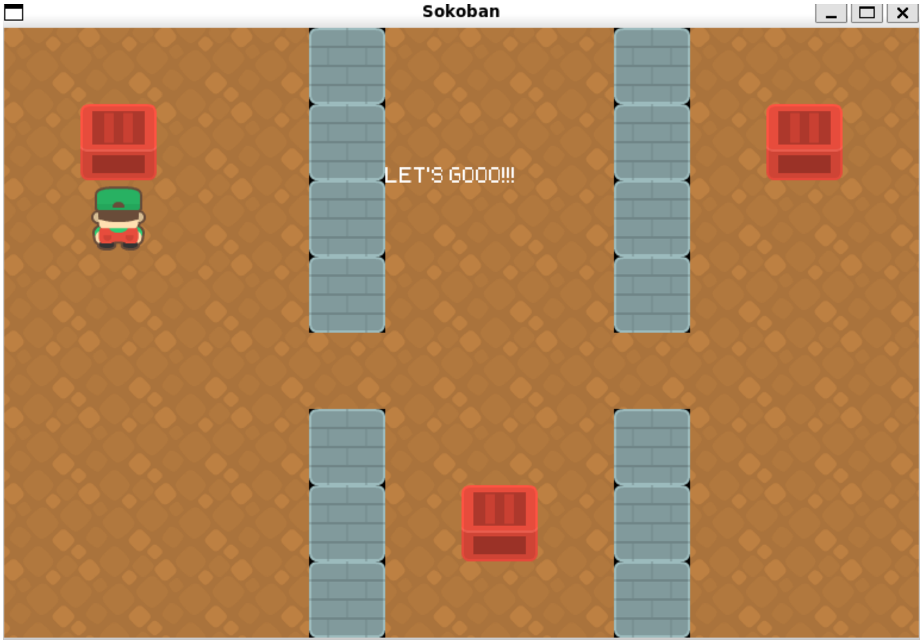


Figure 1: Image player beating a level in Sokoban

```
.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) Sokoban.a

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

test: $(TEST_OBJ)
        $(CC) $(CFLAGS) -o $@ $^ $(LIB)

Sokoban: $(MAIN_OBJ)
        $(CC) $(CFLAGS) -o $@ $^ $(LIB)

Sokoban.a: Sokoban.o
        ar rcs $@ $^

clean:
        rm *.o $(PROGRAM) Sokoban.a

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

Listing 2: main.cpp

```cpp
// Copyright 2025 Christopher Nguyen
#include <iostream>
#include <string>
#include <fstream>
#include <SFML/Audio.hpp>
#include "Sokoban.hpp"
using SB::Sokoban;
int main(int argc, char * argv[]) {
    sf::Font Pixeltype;
    sf::Text victory;
    bool flag = false;
    bool sound_flag = false;
    if (!Pixeltype.loadFromFile("Pixeltype.ttf")) {
        return -1;
    }
    sf::SoundBuffer buffer;
    if (!buffer.loadFromFile("mixkit-instant-win-2021.wav")) {
        return -1;
    }
    sf::Sound sound(buffer);
    Sokoban obj;
    // Open file stream and pass it to Sokoban
    std::ifstream file(argv[1]);
    if (!file) {
        std::cerr << "Error: Could not open file " << argv[1] << std::endl;
        return 1;
    }
    file >> obj;
    unsigned int the_h = obj.height()*obj.TILE_SIZE;
    unsigned int the_w = obj.width()*obj.TILE_SIZE;
    sf::RenderWindow window(sf::VideoMode(the_w, the_h), "Sokoban");

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::
           ↪ Key::D)) {
            obj.movePlayer(SB::Direction::Right);
        } else if (sf::Keyboard::isKeyPressed(sf::
           ↪ Keyboard::Key::W)) {
            obj.movePlayer(SB::Direction::Up);
        } else if (sf::Keyboard::isKeyPressed(sf::
           ↪ Keyboard::Key::A)) {
            obj.movePlayer(SB::Direction::Left);
        } else if (sf::Keyboard::isKeyPressed(sf::
           ↪ Keyboard::Key::S)) {
            obj.movePlayer(SB::Direction::Down);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::
           ↪ Key::R)) {
            obj.reset();
            flag = false;
            sound_flag = false;
        }
        if (obj.isWon() && !sound_flag) {
            victory.setFont(Pixeltype);
            victory.setString("LET'S GOOO!!!");
            victory.setPosition(the_w/2-64, 100);
            flag = true;
            sound_flag = true;
            obj.postWin();
            sound.play();
        }

        window.clear();
        window.draw(obj);
        if (flag) window.draw(victory);
        window.display();
    }
    std::cout << obj.playerLoc().x << " "<< obj.
       ↪ playerLoc().y << std::endl;
    std::cout << obj;
    return 0;
}
```

Listing 3: Sokoban.hpp

```
// Copyright 2025 Christopher Nguyen
#pragma once
#ifndef SOKOBAN_HPP
#define SOKOBAN_HPP
#include <iostream>
```

```cpp
#include <memory>
#include <string>
#include <vector>
#include <algorithm>
#include <SFML/Graphics.hpp>

namespace SB {
enum class Direction {
    Up, Down, Left, Right
};

class Sokoban : public sf::Drawable {
 public:
    static const int TILE_SIZE = 64;

    Sokoban();
    explicit Sokoban(const std::string&);  // Optional

    unsigned int pixelHeight() const;  // Optional
    unsigned int pixelWidth() const;  // Optional

    unsigned int height() const;
    unsigned int width() const;

    sf::Vector2u playerLoc() const;

    bool isWon() const;
    void postWin();

    void movePlayer(Direction dir);
    void reset();

    void undo();  // Optional XC
    void redo();  // Optional XC

    friend std::ostream& operator<<(std::ostream& out, const Sokoban& s);
    friend std::istream& operator>>(std::istream& in, Sokoban& s);

    void setLoc(unsigned int x, unsigned int y);

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

 private:
    // Any fields you need go here.
    unsigned int w;
    unsigned int h;
    unsigned int p_x, p_y;
    sf::Vector2u loc;
    sf::Sprite sprite;
    std::shared_ptr<sf::Texture> texture, t_empty,
    t_p_bottom, t_box, t_loc, t_done, t_p_top, t_p_left,
    t_p_right;
    std::string char_t;
    std::vector<std::vector<std::string>> v;
    std::vector<std::vector<std::string>> original_v;
    SB::Direction player_dir;
    bool afterWin = false;
};


}  // namespace SB

#endif
```

Listing 4: Sokoban.cpp

```
// Copyright 2025 Christopher Nguyen
#include <iostream>
#include "Sokoban.hpp"
#include <SFML/Graphics.hpp>

namespace SB {

Sokoban::Sokoban(): w(10), h(10), player_dir(Direction::Down) {
    texture = std::make_shared<sf::Texture>();
    t_empty = std::make_shared<sf::Texture>();
    t_p_bottom = std::make_shared<sf::Texture>();
    t_box = std::make_shared<sf::Texture>();
    t_loc = std::make_shared<sf::Texture>();
    t_done = std::make_shared<sf::Texture>();
    t_p_top = std::make_shared<sf::Texture>();
    t_p_left = std::make_shared<sf::Texture>();
    t_p_right = std::make_shared<sf::Texture>();
    if (!texture->loadFromFile("block_06.png")) {
        std::cerr << "Error: Could not load texture " << "block_06.png" << std::endl;
```

```cpp
    }
    if (!t_empty->loadFromFile("ground_01.png")) {
        std::cerr << "Error: Could not load texture " << "ground1.png" << std::endl;
    }
    if (!t_p_bottom->loadFromFile("player_05.png")) {
        std::cerr << "Error: Could not load texture " << "player_05.png" << std::endl;
    }
    if (!t_box->loadFromFile("crate_03.png")) {
        std::cerr << "Error: Could not load texture " << "crate_03.png" << std::endl;
    }
    if (!t_loc->loadFromFile("ground_04.png")) {
        std::cerr << "Error: Could not load texture " << "ground_04.png" << std::endl;
    }
    if (!t_p_top->loadFromFile("player_08.png")) {
        std::cerr << "Error: Could not load texture " << "player_08.png" << std::endl;
    }
    if (!t_p_left->loadFromFile("player_20.png")) {
        std::cerr << "Error: Could not load texture " << "player_20.png" << std::endl;
    }
    if (!t_p_right->loadFromFile("player_17.png")) {
        std::cerr << "Error: Could not load texture " << "player_08.png" << std::endl;
    }
}

// Sokoban::Sokoban(const std::string&):w(10), h(10) {

// }

unsigned int Sokoban::height() const {
    return h;
}

unsigned int Sokoban::width() const {
    return w;
}

sf::Vector2u Sokoban::playerLoc() const {
    return sf::Vector2u(p_x, p_y);
}
```

```

bool Sokoban:: isWon() const {
    unsigned int oc_a = 0;
    unsigned int oc_A = 0;
    unsigned int vc_1 = 0;
    unsigned int required = 0;
    for (const auto& row : original_v) {
        oc_a += std::count_if(row.begin(), row.end(),
           ↪  [](std::string st){
            return st == "a";
        });
        oc_A += std::count_if(row.begin(), row.end(),
           ↪  [](std::string st){
            return st == "A";
        });
    }
    required = (oc_a < oc_A)? oc_a:oc_A;
    for (const auto& row : v) {
        vc_1 += std::count_if(row.begin(), row.end(),
           ↪  [](std::string st){
            return st == "1";
        });
    }
    return (vc_1 == required)? true:false;
}

void Sokoban:: postWin() {
    afterWin = true;
}

void Sokoban::movePlayer(Direction dir) {
    unsigned int newX = p_x;
    unsigned int newY = p_y;
    unsigned int boxX = 0;
    unsigned int boxY = 0;

    if (afterWin) return;
    switch (dir) {
        case Direction::Up:
            player_dir = Direction::Up;
            if (p_y == 0) return;
            if (v[p_y - 1][p_x] == "A" || v[p_y - 1][
               ↪  p_x] == "1") {
                if (p_y - 1 == 0) return;
                boxX = p_x;
                boxY = p_y - 2;
                if (v[boxY][boxX] != "#" && v[boxY][boxX] != "A") {
                    if (v[boxY][boxX] == "a") {
                        v[boxY][boxX] = "1";
                    } else {
                        v[boxY][boxX] = "A";
                    }
                    v[p_y - 1][p_x] = ".";
                } else {
                    return;
                }
            }
            if (!(v[p_y - 1][p_x] == "A" && v[p_y - 2][p_x] == "#")) {
                newY = (p_y > 0) ? p_y - 1 : p_y;
            }
            break;

        case Direction::Left:
            player_dir = Direction::Left;
            if (p_x == 0) return;
            if (v[p_y][p_x - 1] == "A" || v[p_y][p_x - 1] == "1") {
                if (p_x - 1 == 0) return;
                boxX = p_x - 2;
                boxY = p_y;
                if (v[boxY][boxX] != "#" && v[boxY][boxX] != "A") {
                    if (v[boxY][boxX] == "a") {
                        v[boxY][boxX] = "1";
                    } else {
                        v[boxY][boxX] = "A";
                    }
                    v[p_y][p_x - 1] = ".";
                } else {
                    return;
                }
            }
            newX = (p_x > 0) ? p_x - 1 : p_x;
            break;

        case Direction::Right:
            player_dir = Direction::Right;
            if (p_x == w - 1) return;
            if (v[p_y][p_x + 1] == "A" || v[p_y][p_x + 1] == "1") {
                if (p_x + 1 == w - 1) return;
                boxX = p_x + 2;
                boxY = p_y;
                if (v[boxY][boxX] != "#" && v[boxY][boxX] != "A") {
                    if (v[boxY][boxX] == "a") {
                        v[boxY][boxX] = "1";
                    } else {
                        v[boxY][boxX] = "A";
                    }
                    v[p_y][p_x + 1] = ".";
                } else {
                    return;
                }
            }
            if (!(v[p_y][p_x + 1] == "A" && v[p_y][p_x + 2] == "#") &&
                !(v[p_y][p_x + 1] == "A" && v[p_y][p_x + 2] == "A")) {
                newX = (p_x < w - 1) ? p_x + 1 : p_x;
            }
            break;

        case Direction::Down:
            player_dir = Direction::Down;
            if (p_y == h-1) return;
            if (v[p_y + 1][p_x] == "A") {
                if (p_y + 1 == h-1) return;
                boxY = p_y + 2;
                boxX = p_x;
                if (v[boxY][boxX] != "#" && v[boxY][boxX] != "A") {
                    if (v[boxY][boxX] == "a") {
                        v[boxY][boxX] = "1";
                    } else {
                        v[boxY][boxX] = "A";
                    }
                    v[p_y + 1][p_x] = ".";
                } else {
                    return;
                }
            }
            newY = (p_y < h - 1) ? p_y + 1 : p_y;
            break;
    }
        if (v[newY][newX] != "#" && v[newY][newX] != "
A") {
        v[p_y][p_x] = (original_v[p_y][p_x] == "a")? "a":".";
        v[newY][newX] = "@";
        setLoc(newX, newY);
    }
}




void Sokoban:: reset() {
    for (unsigned int i = 0; i < h; i++) {
        for (unsigned int j = 0; j < w; j++) {
            if (original_v[i][j] == "@") {
                setLoc(j, i);
            }
            v[i][j] = original_v[i][j];
        }
    }
    afterWin = false;
    return;
}

void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    sf::Sprite tileSprite(*texture);
    for (unsigned int i = 0; i < h; i++) {
        for (unsigned int j = 0; j < w; j++) {
            if (v[i][j] != "#") {
                tileSprite.setTexture(*t_empty);
                tileSprite.setPosition(j * TILE_SIZE, i * TILE_SIZE);
                target.draw(tileSprite, states);
            }
        }
    }
    for (unsigned int i = 0; i < h; i++) {
        for (unsigned int j = 0; j < w; j++) {
                if (v[i][j] == "A" || v[i][j] == "1") {
                    tileSprite.setTexture(*t_box);
                } else if (v[i][j] == ".") {
                    tileSprite.setTexture(*t_empty);
                } else if (v[i][j] == "#") {
                    tileSprite.setTexture(*texture);
                } else if (v[i][j] == "a") {
                    tileSprite.setTexture(*t_loc);
                } else if (v[i][j] == "@") {
                    switch (player_dir) {
                        case Direction::Up:
                           ↪  tileSprite.setTexture(*
                           ↪  t_p_top); break;
                        case Direction::Down:
                           ↪  tileSprite.setTexture(*
                           ↪  t_p_bottom); break;
                        case Direction::Left:
                           ↪  tileSprite.setTexture(*
                           ↪  t_p_left); break;
                        case Direction::Right:
                           ↪  tileSprite.setTexture(*
                           ↪  t_p_right); break;
                    }
                }
                tileSprite.setPosition(j * TILE_SIZE,
                   ↪  i * TILE_SIZE);
                target.draw(tileSprite, states);
            }
    }
}

std::istream& operator>>(std::istream& in, Sokoban& s)
   ↪  {
    in >> s.h >> s.w;
    in.ignore();
    s.v.clear();

    std::string line;
    for (unsigned int i = 0; i < s.h; i++) {
        std::getline(in, line);
        std::vector<std::string> row;
        for (unsigned int j = 0; j < s.w; j++) {
            std::string cell(1, line[j]);
            row.push_back(cell);
            if (cell == "@") {
                s.setLoc(j, i);
            }
        }

        s.v.push_back(row);
        s.original_v.push_back(row);
    }
```

```cpp
    return in;
}


std::ostream& operator<<(std::ostream& out, const Sokoban& s) {
    for (const auto& row : s.v) {
        for (const auto& cell : row) {
            out << cell;
        }
        out << std::endl;
    }
    return out;
}

void Sokoban:: setLoc(unsigned int x, unsigned int y) {
    p_x = x;
    p_y = y;
}

}  // namespace SB
```

Listing 5: test.cpp

```cpp
// Copyright 2025 <Christopher Nguyen>
#include <iostream>
#include <string>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include "Sokoban.hpp"
#include <boost/test/unit_test.hpp>
using SB::Sokoban;

BOOST_AUTO_TEST_CASE(play_moves_right_works_as_intended) {
    sf::Event event;
    Sokoban obj;
    std::istringstream l1(
    "10 10\n"
    "##########\n"
    "#....a...#\n"
    "#....A...#\n"
    "#........#\n"
    "#...##...#\n"
    "#...##...#\n"
    "#..@..A..#\n"
    "#.......a#\n"
    "#........#\n"
    "##########\n");
    l1 >> obj;
    event.type = sf::Event::KeyPressed;
    event.key.code = sf::Keyboard::D;
    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::D) {
            obj.movePlayer(SB::Direction::Right);
        }
    }
    unsigned int x_pos = obj.playerLoc().x;
    unsigned int y_pos = obj.playerLoc().y;
    BOOST_CHECK_EQUAL(x_pos, 4);
    BOOST_CHECK_EQUAL(y_pos, 6);
}

BOOST_AUTO_TEST_CASE(player_cant_move_through_wall) {
    sf::Event event;
    Sokoban obj;
    std::istringstream l1(
    "10 10\n"
    "##########\n"
    "#....a...#\n"
    "#....A...#\n"
    "#........#\n"
    "#...##...#\n"
    "#...##...#\n"
    "#@....A..#\n"
    "#.......a#\n"
    "#........#\n"
    "##########\n");
    l1 >> obj;
    event.type = sf::Event::KeyPressed;
    event.key.code = sf::Keyboard::A;
    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::A) {
            obj.movePlayer(SB::Direction::Left);
        }
    }
    unsigned int x_pos = obj.playerLoc().x;
    unsigned int y_pos = obj.playerLoc().y;
    BOOST_CHECK_EQUAL(x_pos, 1);
    BOOST_CHECK_EQUAL(y_pos, 6);
}
```

```

BOOST_AUTO_TEST_CASE(player_cant_go_offscreen) {
    sf::Event event;
    Sokoban obj;
    std::istringstream l1(
    "10 10\n"
    "##########\n"
    "#....a...#\n"
    "#....A...#\n"
    "#........#\n"
    "#...##...#\n"
    "#...##...#\n"
    "@.....A..#\n"
    "#.......a#\n"
    "#........#\n"
    "##########\n");
    l1 >> obj;
    event.type = sf::Event::KeyPressed;
    event.key.code = sf::Keyboard::A;
    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::A) {
            obj.movePlayer(SB::Direction::Left);
        }
    }
    unsigned int x_pos = obj.playerLoc().x;
    unsigned int y_pos = obj.playerLoc().y;
    BOOST_CHECK_EQUAL(x_pos, 0);
    BOOST_CHECK_EQUAL(y_pos, 6);
}

BOOST_AUTO_TEST_CASE(box_cannot_go_offscreen) {
    sf::Event event;
    sf::Event event1;
    Sokoban obj;
    std::istringstream l1(
    "10 10\n"
    "##########\n"
    "#A@..a...#\n"
    "#....A...#\n"
    "#........#\n"
    "#...##...#\n"
    "#...##...#\n"
    "#.....A..#\n"
    "#.......a#\n"
    "#........#\n"
    "##########\n");
    l1 >> obj;
    event.type = sf::Event::KeyPressed;
    event.key.code = sf::Keyboard::A;
    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::A) {
            obj.movePlayer(SB::Direction::Left);
        }
    }
    event1.type = sf::Event::KeyPressed;
    event1.key.code = sf::Keyboard::A;
    if (event1.type == sf::Event::KeyPressed) {
        if (event1.key.code == sf::Keyboard::A) {
            obj.movePlayer(SB::Direction::Left);
        }
    }
    unsigned int x_pos = obj.playerLoc().x;
    unsigned int y_pos = obj.playerLoc().y;
    BOOST_CHECK_EQUAL(x_pos, 2);
    BOOST_CHECK_EQUAL(y_pos, 1);
}

BOOST_AUTO_TEST_CASE(
   ↪ adjacent_parallel_boxes_cannot_move) {
    sf::Event event;
    sf::Event event1;
    Sokoban obj;
    std::istringstream l1(
    "10 10\n"
    "##########\n"
    "#.AA@a...#\n"
    "#....A...#\n"
    "#........#\n"
    "#...##...#\n"
    "#...##...#\n"
    "#.....A..#\n"
    "#.......a#\n"
    "#........#\n"
    "##########\n");
    l1 >> obj;
    event.type = sf::Event::KeyPressed;
    event.key.code = sf::Keyboard::A;
    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::A) {
            obj.movePlayer(SB::Direction::Left);
        }
    }
```

```cpp
    event1.type = sf::Event::KeyPressed;
    event1.key.code = sf::Keyboard::A;
    if (event1.type == sf::Event::KeyPressed) {
        if (event1.key.code == sf::Keyboard::A) {
            obj.movePlayer(SB::Direction::Left);
        }
    }
    unsigned int x_pos = obj.playerLoc().x;
    unsigned int y_pos = obj.playerLoc().y;
    BOOST_CHECK_EQUAL(x_pos, 4);
    BOOST_CHECK_EQUAL(y_pos, 1);
}

BOOST_AUTO_TEST_CASE(more_boxes_than_targets_victory) {
    Sokoban obj;
    std::istringstream l1(
    "10 10\n"
    "##########\n"
    "#....a...#\n"
    "#....A...#\n"
    "#....@...#\n"
    "#...##...#\n"
    "#...##...#\n"
    "#.A......#\n"
    "#........#\n"
    "#........#\n"
    "##########\n");
    l1 >> obj;
    sf::Event event;
    event.type = sf::Event::KeyPressed;
    event.key.code = sf::Keyboard::W;
    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::W) {
            obj.movePlayer(SB::Direction::Up);
        }
    }
    bool win = (obj.isWon())? true:false;
    BOOST_CHECK_EQUAL(win, true);
}

BOOST_AUTO_TEST_CASE(more_targets_than_boxes_victory) {
    Sokoban obj;
    std::istringstream l1(
    "10 10\n"
    "##########\n"
    "#...a....#\n"
    "#...A....#\n"
    "#...@....#\n"
    "#...##...#\n"
    "#...##...#\n"
    "#........#\n"
    "#...a....#\n"
    "#...a....#\n"
    "##########\n");
    l1 >> obj;
    sf::Event event;
    event.type = sf::Event::KeyPressed;
    event.key.code = sf::Keyboard::W;
    if (event.type == sf::Event::KeyPressed) {
        if (event.key.code == sf::Keyboard::W) {
            obj.movePlayer(SB::Direction::Up);
        }
    }
    bool win = (obj.isWon())? true:false;
    BOOST_CHECK_EQUAL(win, true);
}


// BOOST_AUTO_TEST_CASE(t1_check_insertion_outputs_file) {
//     Sokoban obj;
//     std::istringstream l1(
//     "10 10\n"
//     "##########\n"
//     "#....a...#\n"
//     "#....A...#\n"
//     "#........#\n"
//     "#...##...#\n"
//     "#...##...#\n"
//     "#..@..A..#\n"
//     "#.......a#\n"
//     "#........#\n"
//     "##########\n");
//     l1 >> obj;

//     std::ostringstream output;
//     output << obj;


//     std::string expected =  "10 10\n"
//                              "##########\n"
//                              "#....a...#\n"
//                              "#....A...#\n"
//                              "#........#\n"
//                              "#...##...#\n"
//                              "#...##...#\n"
//                              "#..@..A..#\n"
//                              "#.......a#\n"
//                              "#........#\n"
//                              "##########\n";
//     BOOST_REQUIRE_EQUAL(output.str(), expected);
// }
```

PS5 DNA Alignment

This project is supposed to align two DNA sequences as similarly as possible with the lowest amount of cost. Then it displays the editing distance of the two sequences. Then it shows the alignment of the two sequences. Additionally, the execution time of aligning two sequences will be displayed. The opt distance is the function that returns the cost of aligning the two DNA sequences. The alignment function returns the sequences after alignment.

We used two vectors,, so we can store two rows at a time. Then we set the values of the diagonal, down, and right. The current value will be the minimum of the diagonal, down,, or right Hence there is a function min3 that returns the smallest value of diagonal, down,wn, and right. To make the penalty function, I return where the arguments equal one another, if they are equal, it returns a 1. Otherwise it returns a 0. For the alignment function, it returns a string. Alignment function creates a 2d vector which represents the grid of the 2 sequences. Check if the current box matches the down, right, or diagonal box. The current will be whichever is the right case. This process repeats until it reaches all the the indexes that are the length of the first sequence and the length of the second sequence plus. For each iteration, the string will increase in content.

To get the optDistance() we used a method to store two rows at a time. Both are vectors of type int. We made a base case for the first row that follows opt[M][j] = 2(N - j) from the pdf. We used a nested for loop to start at the second lowest row and to set the current row from right to left. Then set the current row to follow opt[i][N] = 2(M - i) from pdf. Then compare the adjacent elements and set the current element to be the lowest cost of the diagonal, down, or right. then you swap the previous vector to have the contents of the current one.

For the EDistance constructor, I had to remove new lines for strings x and y and then set the sizes for the two sequences that are to be used for further calculations.

In the main we used getline to set the first sequence to be a string object and another getline with another. We used the two updated string objects as arguments for the EDistance object. We then used sf::Clock to get the elapsed time of execution for aligning two sequences.

Penalty Function Test

To verify the correctness of the `penalty` function, I created a test case where the input parameters do not match. The function is expected to return a penalty value of `1` in this case. The return value is stored in an integer variable `result`, and the following assertion is used:

```
BOOST_CHECK_EQUAL(result, 1);
```

Since the parameters differ, the `penalty` function correctly returns `1`, making the assertion pass and confirming that the function behaves as intended.

Test Case: min3_01

Another test case I created is min3_01, which verifies the behavior of the min3 function across several scenarios using BOOST_CHECK_EQUAL. One example from this test case is:

```
BOOST_CHECK_EQUAL(obj.min3(1, 2, 3), 1);
```

This test passes, as 1 is the smallest of the three input values. All other BOOST_CHECK_EQUAL checks in this test case also pass, confirming that the min3 function works correctly in various situations.

Test Case: optDistance

Another test case checks whether the optDistance function works as intended. To do this, I created an EDistance object with the following input strings:

```
EDistance obj("AACAGTTACC", "TAAGGTCA");
```

Then, I computed the optimal distance:

```
int res = obj.optDistance();
```

To verify the result, I used the following assertion:

```
BOOST_CHECK_EQUAL(res, 7);
```

Since the expected optimal distance is 7, this test passes, confirming that the optDistance function is working correctly for this input.

Test Case: alignment_function

For the alignment_function test case, I created an EDistance object with the following input strings:

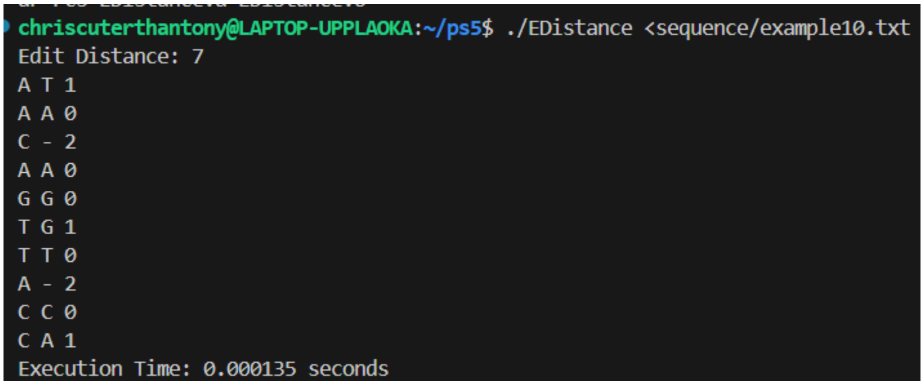
```
EDistance ed("TACAGTTACC", "TAAGGTCA");
```

Next, I created a string containing the expected alignment result. I then used the following assertion to compare the actual alignment with the expected output:

```
BOOST_CHECK_EQUAL(ed.alignment(), expected_string);
```

The comparison returns true, meaning the alignment matches the expected result, and the test case passes successfully.

I then compared if its optDistance is correct then checked its alignment with the expected string and test case passes.



```
chriscuterthantony@LAPTOP-UPPLAOKA:~/ps5$ ./EDistance <sequence/example10.txt
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: 0.000135 seconds
```

Figure 1: Image of the output of the program

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g -O3
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework

# Your .hpp files
DEPS = EDistance.hpp

# Your compiled .o files
MAIN_OBJ = main.o EDistance.o
TEST_OBJ = test.o EDistance.o

# The name of the program
PROGRAM = EDistance
TEST_PROGRAM = test

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) EDistance.a

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

test: $(TEST_OBJ)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)

EDistance: $(MAIN_OBJ)
```

```
        $(CC) $(CFLAGS) -o $@ $^ $(LIB)

EDistance.a: EDistance.o
        ar rcs $@ $^

clean:
        rm *.o $(PROGRAM) EDistance.a

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

Now, let’s look at a simple C++ program that prints ”Hello, world!” to the console.

Listing 2: main.cpp

```
// Copyright 2025 Christopher Nguyen
#include <iostream>
#include <sstream>
#include <SFML/System.hpp>

#include "EDistance.hpp"

int main(int argc, char* argv[]) {
    if (argc != 1) {
        std::cerr << "Usage: " << argv[0] << " < <
           ↪ filename.txt>" << std::endl;
        return 1;
    }

    std::string a, b;
    std::getline(std::cin, a);
    std::getline(std::cin, b);

    sf::Clock clock;  // start the clock
    EDistance edObj(a, b);  // create an EDistance
       ↪ object
    int edDist = edObj.optDistance();  // calculate
       ↪ the edit distance
    std::string optStr = edObj.alignment();  // get
       ↪ the optimal alignment

    std::cout << "Edit Distance: " << edDist << std::
       ↪ endl;
    std::cout << optStr;  // print the optimal
       ↪ alignment
    sf::Time t = clock.getElapsedTime();  // get the
```

```
        ↪ elapsed time
    std::cout << "Execution Time: " << t.asSeconds()
        ↪ << " seconds" << std::endl;

    return 0;
}
```

Listing 3: EDistance.hpp

```
// Copyright 2025 Christopher Nguyen
#pragma once
#ifndef EDISTANCE_HPP
#define EDISTANCE_HPP
#include <iostream>
#include <string>
#include <vector>
// #include <algorithm>
#include <SFML/Graphics.hpp>

class EDistance {
 public:
    EDistance(const std::string& s1, const std::string
        ↪ & s2)
    : _x(removeNewlines(s1)), _y(removeNewlines(s2)),
      _M(_x.size()), _N(_y.size()) {}

    inline static int penalty(char a, char b) {  //
        ↪ returns 0 if equal, 1 otherwise
      return (a == b) ? 0 : 1;
    }

    inline static int min3(int a, int b, int c) {  //
        ↪ returns minimum of three integers
      // int result = (a < b) ? a : b;
      // return (result < c) ? result : c;
      return std::min({a, b, c});
    }

    int optDistance() const;  // returns optimal
        ↪ distance |0||0|

    std::string alignment();  // returns string to be
        ↪ displayed

    //  int dp(int i, int j) const;

 private:
    const std::string _x, _y;
    const int _M, _N;
    // std::vector<std::vector<int>> _grid;
    std::vector<int> _grid;
    inline static std::string removeNewlines(const std::string& input) {
      std::string result = input;
      result.erase(std::remove_if(result.begin(), result.end(), isspace), result.end());
      return result;
  }
};

#endif
```

Listing 4: EDistance.cpp

```
// Copyright 2025 Christopher Nguyen
#include <iostream>
#include <algorithm>

#include "EDistance.hpp"

int EDistance::optDistance() const {
    if (_M == 0) return 2 * _N;
    if (_N == 0) return 2 * _M;
    if (_x == _y) return 0;
    // store two rows at a time (current and previous)
    std::vector<int> prev(_N + 1);
    std::vector<int> curr(_N + 1);

    // std::cout << "\n\n" << _M << std::endl;
    // std::cout << "" << _N;

    // first row (base case)
    for (int j = 0; j <= _N; j++) {
        prev[j] = 2 * (_N - j);
    }
    // std::cout << "     ";
    // for (int j = 0; j <= _N; j++) {
    //      std::cout << std::setw(4) << j;
    // }
    // std::cout << std::endl;
    // std::cout << "    ";
    // for (int j = 0; j <= _N; j++) {
```

```cpp
    //     std::cout << "----";
    // }
    // std::cout << std::endl;
    // std::cout << std::setw(3) << 0 << " |";
    // for (int j = 0; j <= _N; j++) {
    //     std::cout << std::setw(4) << prev[j];
    // }
    // std::cout << std::endl;

    // fill the matrix row by row
    // int displayRow = 1; // start display row at 1
    for (int i = _M - 1; i >= 0; i--) {
        // rightmost element of the current row
        curr[_N] = 2 * (_M - i);

        // fill the current row from right to left
        for (int j = _N - 1; j >= 0; j--) {
            int diag = prev[j + 1] + penalty(_x[i], _y[j]);
            int down = prev[j] + 2;
            int right = curr[j + 1] + 2;
            curr[j] = min3(diag, down, right);
        }
        // std::cout << std::setw(3) << displayRow << " |";
        // for (int j = 0; j <= _N; j++) {
        //     std::cout << std::setw(4) << curr[j];
        // }
        // std::cout << std::endl;
        // displayRow++;
        // current row is now previous row for next iteration, so swap
        prev.swap(curr);
    }

    return prev[0];
}

/* **********************************************************************
*   Starting from the (0, 0) cell, backtrack through the matrix to find      *
*   the optimal alignment. Traverse the matrix as follows:                   *
*   1. Move diagonally (i+1, j+1) for a MATCH (0) or
```

```
SUBSTITUTION (1).        *
*   2. Move down (i+1, j) for a GAP (2) in y.                        *
*   3. Move right (i, j+1) for a GAP (2) in x.                      *
*   Continue until you reach the bottom-right cell of the matrix.             *
****************************************************************************/
// std::string EDistance::alignment() {
//     _grid.resize(_M + 1, std::vector<int>(_N + 1, 0));

//     /********************************************************************
//     *   Bottom-Up (Right-Left) Dynamic Programming using a n*m matrix   *
//     *   - s1 is string _x of size M                                      *
//     *   - s2 is string _y of size N                                      *
//     ********************************************************************/

//     /****************************
//     *   Base Cases              *
//     *   1. opt[M][j] = 2(N-j)   *
//     *   2. opt[i][N] = 2(M-i)   *
//     ****************************/
//     for (int j = 0; j < _N; j++) {
//         _grid[_M][j] = 2 * (_N - j);
//     }  // 1
//     for (int i = 0; i < _M; i++) {
//         _grid[i][_N] = 2 * (_M - i);
//     }  // 2

//     /**********************************************************************
//     *   Each cell is filled with the minimum of three possibilities:       *
//     *   1. Diagonal (i+1, j+1) + penalty(x[i], y[j])                       *
//     *   2. Down (i+1, j) + 2
```

```
                                                         *
//     *   3. Right (i, j+1) + 2                                  *
//     *   The penalty is 0 if the characters are equal, otherwise it is 1.   *
//     *   The edit distance is stored in the top-left cell (0, 0).           *
//     ************************************************************************
//     for (int i = _M - 1; i >= 0; i--) {
//         for (int j = _N - 1; j >= 0; j--) {
//             _grid[i][j] = min3(
//                 _grid[i+1][j+1] + penalty(_x[i], _y[j]),  // 1
//                 _grid[i+1][j] + 2,  // 2
//                 _grid[i][j+1] + 2);  // 3
//         }
//     }

//     int i = 0, j = 0;
//     std::string result;
//     result.reserve((_M + _N) * 5);  // Reserve space for the result string.
//     while (i < _M  && j < _N) {
//         int current = _grid[i][j];
//         int costDiag   = _grid[i+1][j+1] + penalty(_x[i], _y[j]);
//         int costDelete = _grid[i+1][j]   + 2;

//         if (current == costDiag) {  // 1
//             result += std::string(1, _x[i]) + " " + std::string(1, _y[j]) + " " +
//             std::to_string(penalty(_x[i], _y[j])) + "\n";
//             i++;
//             j++;
//         } else if (current == costDelete) {  // 2
//             result += std::string(1, _x[i]) + " - 2\n";
//             i++;
//         } else {  // 3
//             result += "- " + std::string(1, _y[j]) + " 2\n";
//             j++;
//         }
//      }
//      i++;
//      j++;
//      while (i < _M) {
//          result += std::string(1, _x[i]) + " - 2\n";
//          i++;
//      }
//      while (j < _N) {
//          result += "- " + std::string(1, _y[j]) + " 2\n";
//          j++;
//      }
//      std::cout << "Total Cost: " << _grid[0][0] << std::endl;
//      return result;
// }

/* **********************************************************************
*   Starting from the (0, 0) cell, backtrack through the matrix to find       *
*   the optimal alignment. Traverse the matrix as follows:                    *
*   1. Move diagonally (i+1, j+1) for a MATCH (0) or SUBSTITUTION (1).        *
*   2. Move down (i+1, j) for a GAP (2) in y.                                 *
*   3. Move right (i, j+1) for a GAP (2) in x.                                *
*   Continue until you reach the bottom-right cell of the matrix.             *
**************************************************************************** */
std::string EDistance::alignment() {
    auto index =  [this](int i, int j) {
        return i * (_N + 1) + j;
    };

    _grid.resize((_M + 1) * (_N + 1), 0);

    /* ******************************************************************
    *   Bottom-Up (Right-Left) Dynamic Programming using a n*m matrix   *
    *   - s1 is string _x of size M
        ↪                                     *
    *   - s2 is string _y of size N
        ↪                                     *
    *******************************************************************
        ↪ */

    /****************************
    *   Base Cases              *
    *   1. opt[M][j] = 2(N-j)   *
    *   2. opt[i][N] = 2(M-i)   *
    ****************************/

    // base cases: last row and last column.
    for (int j = 0; j <= _N; j++) {
        _grid[index(_M, j)] = 2 * (_N - j);
    }

    for (int i = 0; i <= _M; i++) {
        _grid[index(i, _N)] = 2 * (_M - i);
    }

    /*
        ↪ ******************************************************************
        ↪
    *   Each cell is filled with the minimum of three
        ↪ possibilities:       *
    *   1. Diagonal (i+1, j+1) + penalty(x[i], y[j])
        ↪                       *
    *   2. Down (i+1, j) + 2
        ↪
        ↪ *
    *   3. Right (i, j+1) + 2
        ↪
        ↪ *
    *   The penalty is 0 if the characters are equal,
        ↪ otherwise it is 1.   *
    *   The edit distance is stored in the top-left
        ↪ cell (0, 0).         *
    **********************************************************************
        ↪ */
    for (int i = _M - 1; i >= 0; i--) {
        for (int j = _N - 1; j >= 0; j--) {
            int diag = _grid[index(i + 1, j + 1)] +
                ↪ penalty(_x[i], _y[j]);
            int down = _grid[index(i + 1, j)] + 2;
            int right = _grid[index(i, j + 1)] + 2;
            _grid[index(i, j)] = min3(diag, down, right);
        }
    }

    std::string result;
    result.reserve((_M + _N) * 6 + 1);
    // backtrack from (0, 0) to (M, N)
    int i = 0, j = 0;
    while (i < _M && j < _N) {
        int current = _grid[index(i, j)];
        int diag = _grid[index(i + 1, j + 1)] + penalty(_x[i], _y[j]);
        int down = _grid[index(i + 1, j)] + 2;
        int right = _grid[index(i, j + 1)] + 2;
        if (current == diag) {
            // match/sub
            result += std::string(1, _x[i]) + " " + std::string(1, _y[j]) + " " +
                      std::to_string(penalty(_x[i], _y[j])) + "\n";
            i++;
            j++;
        } else if (current == down) {
            // gap in second string (insert '-' in y).
            result += std::string(1, _x[i]) + " - 2\n";
            i++;
        } else if (current == right) {
            // gap in the first string (insert '-' in x).
            result += "- " + std::string(1, _y[j]) + " 2\n";
            j++;
        } else {
            break;
        }
    }
    while (i < _M) {
        result += std::string(1, _x[i]) + " - 2\n";
        i++;
    }
    while (j < _N) {
        result += "- " + std::string(1, _y[j]) + " 2\n";
        j++;
    }
    return result;
}
```

Listing 5: test.cpp

```
// Copyright 2025 Christopher Nguyen
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include "EDistance.hpp"
#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(penalty_function) {
    // characters don't match, penalty should
    // be 1
    char i = 'i';
    char j = 'j';
    EDistance obj("i", "j");
    int result = obj.penalty(i, j);
    BOOST_REQUIRE_NO_THROW(obj.penalty(i, j));
    BOOST_CHECK_EQUAL(result, 1);
}

BOOST_AUTO_TEST_CASE(min3) {
    // tests min3 function
    // duhh!! it's in the name
    EDistance obj("i", "j");

    int res = obj.min3(1, 2, 3);
    BOOST_CHECK_EQUAL(res, 1);
    BOOST_CHECK_NO_THROW();
}

BOOST_AUTO_TEST_CASE(min3_01) {
    // tests min3 function
    // duhh!! it's in the name
    EDistance obj("AACAGTTACC", "TAAGGTCA");

     // Test different scenarios
     BOOST_CHECK_EQUAL(obj.min3(1, 2, 3), 1);
     BOOST_CHECK_EQUAL(obj.min3(3, 2, 1), 1);
     BOOST_CHECK_EQUAL(obj.min3(2, 3, 1), 1);
     BOOST_CHECK_EQUAL(obj.min3(0, -1, 1), -1);
     BOOST_CHECK_EQUAL(obj.min3(-1, -2, -3), -3);

     BOOST_CHECK_EQUAL(obj.min3(5, 5, 5), 5);  // All values are equal, should return 5

    // Test when two numbers are equal and the third is different
    BOOST_CHECK_EQUAL(obj.min3(2, 2, 3), 2);  // Two equal numbers, the smallest should be returned
    BOOST_CHECK_EQUAL(obj.min3(3, 2, 2), 2);  // The same case with positions swapped
    BOOST_CHECK_EQUAL(obj.min3(2, 3, 2), 2);  // Another arrangement

    // Test with zero and negative numbers
    BOOST_CHECK_EQUAL(obj.min3(0, -1, -2), -2);  // Negative numbers with zero, -2 is the smallest
    BOOST_CHECK_EQUAL(obj.min3(-5, 0, -3), -5);  // Mix of negative and zero, -5 is the smallest

    // Test when two numbers are very close
    BOOST_CHECK_EQUAL(obj.min3(1000000, 999999, 1000001), 999999);
    // Very large numbers, 999999 is the smallest
    BOOST_CHECK_EQUAL(obj.min3(999999, 1000000, 1000001), 999999);
    // Different order, still the same smallest number

    BOOST_CHECK_NO_THROW(obj.min3(1000000, 999999, 1000001));
}


BOOST_AUTO_TEST_CASE(optDistance_works_as_intended) {
    EDistance obj("AACAGTTACC", "TAAGGTCA");
    int res = obj.optDistance();
    BOOST_REQUIRE_NO_THROW(obj.optDistance());
    BOOST_CHECK_EQUAL(res, 7);
}

BOOST_AUTO_TEST_CASE(alignment_function) {
    EDistance ed("TACAGTTACC", "TAAGGTCA");
    std::string expected = "T T 0\n"
    "A A 0\n"
    "C - 2\n"
    "A A 0\n"
    "G G 0\n"
    "T G 1\n"
    "T T 0\n"
    "A - 2\n"
    "C C 0\n"
    "C A 1\n";
    BOOST_CHECK_EQUAL(ed.alignment(), expected);
}

BOOST_AUTO_TEST_CASE(alignment_function_01) {
    EDistance ed("CAGCTACAA", "CAGACAA");
    std::string ext = "C C 0\n"
    "A A 0\n"
    "G G 0\n"
    "C - 2\n"
    "T - 2\n"
    "A A 0\n"
    "C C 0\n"
    "A A 0\n"
    "A A 0\n";
    BOOST_CHECK_EQUAL(ed.alignment(), ext);
    BOOST_REQUIRE_NO_THROW(ed.alignment());
}

BOOST_AUTO_TEST_CASE(alignment_with_nothing) {
    EDistance e("", "");
    BOOST_CHECK_EQUAL(e.optDistance(), 0);
    BOOST_CHECK_EQUAL(e.alignment(), "");
    BOOST_REQUIRE_NO_THROW(e.optDistance());
}

BOOST_AUTO_TEST_CASE(alignment_end_gaps) {
    EDistance e_gaps("atattat", "tattata");

    std::string exp = "a - 2\n"
    "t t 0\n"
    "a a 0\n"
    "t t 0\n"
    "t t 0\n"
    "a a 0\n"
    "t t 0\n"
    "- a 2\n";

    BOOST_CHECK_EQUAL(e_gaps.optDistance(), 4);
    BOOST_CHECK_EQUAL(e_gaps.alignment(), exp);
    BOOST_REQUIRE_NO_THROW(e_gaps.optDistance());
}
```

PS6 Random Writer

The program reads an input from the file, then it is able to to produce a sequence. The sequence is based on a substrings frequencies that is able to produce random strings.

For the RandWriter constructor, I had to check if the input's length is less than the k gram. If it is true, it will throw an exepction. I then have to handle the pre wrap around traversal of the string. It is done by a for loop and the substring is at index i and the k-gram. I have two maps, one for the frequencies of the substring and another map for the frequencies of the substring and the character following it. I then use I for another for loop that maps the maps for the wrap around.

For the orderK function, I had a private member named underscore order which is the stores of the k gram. The orderK function returns underscore order.

For the freq functions, I map through the map and checks if the substring is in the map, if it's found, it will return the amount of times the substring occurs.

For the kRand function, I first have to check if the size of the substring does not equal to the order. If the conditional statement is true, it will throw an execption. It will also check if the substring is not in the map. If this conditional statement is true, it will also throw an execption. I made two vectors on that represents the next char after the substring and another one that represents the frequency of the next char. I used a for loop and to traverse through the entire map that holds the substring. If the substring matches the kgram, it will push the next character to nChars and will increment the weight. To setup the random character based on a character's frequency, I used std::mt19937 engine to set the state. Then I used discrete distribution that creates a weighted distribution that mirrors the frequencies. I made an int variable that returns the corresponding next character.

For the generate function, I initialized an empty string. Then I added the kgram to the string. Then L is subtracted by the length of the substring I used a while loop that iterates until the size L is less than or equal to 0. Inside the while loop, the string gets added by the kRand function. Then I decrement L and increased I to update the substring.

For the insertion operator, I used an unordered set to find the distinct elements of the string. For the entire string I insert the character to the set. Then I print out all the contents of the set, and they are all unique. To print out the k gram frequncies I print out the substring and its frequency. To print out the k plus one frequencies I iterate through a map and print out the substring and its frequencies.

I have 5 test cases. For one case, I check to see if calling a function to the RandWriter class that has a string length that is not equivalent to the order. I used BOOST REQUIRE THROW to check if it throws an invalid argument.

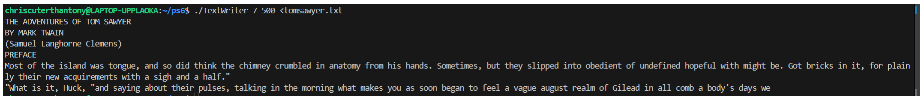
For the second test case, I made a RandWriter object that has an order of 0. I checked the freq function that has an empty string and the character c. I then Check if the numbers of c's occured from the freq function is equivalent to 3.

For the third test case. I test the generate function to see if it returns a length of 5.

For the fourth test case, I test if the first three characters of a string after generate is the first 3 elements of the input string.

For the fifth test case, I tested if the substring agg follows up to be g.

I used my lambda expression for the freq function that has the kgram and c. If the order is 0, I used the count if algorithm to traverse through the input string and the lambda check if the character in the input string is equivalent to c.



```
chriscuterthantony@LAPTOP-UPPLAOKA:~/ps6$ ./TextWriter 7 500 <tomsawyer.txt
THE ADVENTURES OF TOM SAWYER
BY MARK TWAIN
(Samuel Langhorne Clemens)
PREFACE
Most of the island was tongue, and so did think the chimney crumbled in anatomy from his hands. Sometimes, but they slipped into obedient of undefined hopeful with might be. Got bricks in it, for plain
ly their new acquirements with a sigh and a half."
"What is it, Huck, "and saying about their pulses, talking in the morning what makes you as soon began to feel a vague august realm of Gilead in all comb a body's days we
```

Figure 1: Image of the output of the program

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework

# Your .hpp files
DEPS = RandWriter.hpp

# Your compiled .o files
MAIN_OBJ = TextWriter.o RandWriter.o
TEST_OBJ = test.o RandWriter.o

# The name of the program
PROGRAM = TextWriter
TEST_PROGRAM = test

.PHONY: all clean lint

all: $(PROGRAM) $(TEST_PROGRAM) TextWriter.a

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

test: $(TEST_OBJ)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)

```

```
TextWriter: $(MAIN_OBJ)
	$(CC) $(CFLAGS) -o $@ $^ $(LIB)

TextWriter.a: TextWriter.o RandWriter.o
	ar rcs $@ $^

clean:
	rm *.o $(PROGRAM) $(TEST_PROGRAM) TextWriter.a

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

The Textwritter.cpp is the equivalent of main.cpp in this case.

Listing 2: TextWriter.cpp

```cpp
// Copyright Christopher Nguyen 2025
#include <iostream>
#include "RandWriter.hpp"

int main(int argc, char* argv[]) {
    if (argc != 3) {
        throw std::invalid_argument("Usage: ./TextWriter k l < input.txt");
    }

    size_t k = std::stoul(argv[1]);  // For the order
    size_t l = std::stoul(argv[2]);  // For the amount of chars printed
    std::string in;
    for (std::string line; std::getline(std::cin, line);) {
        in.append(line).push_back('\n');
    }
    if (!in.empty()) {
        in.pop_back();
    }

    RandWriter ran(in, k);
    std::string sub_str = in.substr(0, k);
    std::string res = ran.generate(sub_str, l);
    std::cout << res << std::endl;
    return 0;
}
```

Listing 3: RandtWriter.hpp

```
// Copyright 2025 Christopher Nguyen
#pragma once
#ifndef RANDWRITER_HPP
#define RANDWRITER_HPP
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <unordered_map>
#include <stdexcept>
#include <chrono>
#include <random>
#include <unordered_set>

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& str, size_t k);

    size_t orderK() const;  // Order k of Markov model

    // Number of occurences of kgram in text
    // Throw an exception if kgram is not length k
    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
       ↪ )
    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,
        ↪ const RandWriter &ran);

 private:
    // Private member variables go here
    size_t _order;
    std::string obj_str;
    std::unordered_map<std::string, int> freq_k_gram;
    std::unordered_map<std::string, int> freq_kp1_gram
        ↪ ;
};

#endif
```

Listing 4: RandtWriter.cpp

```
// Copyright 2025 Christopher Nguyen
#include <iostream>
#include "RandWriter.hpp"

RandWriter::RandWriter(const std::string& str, size_t
    ↪ k): _order(k), obj_str(str) {
    std::string sub, sub_p1;
    size_t i, j;

    if (obj_str.length() < k + 1) {
        throw std::logic_error("input must be at least
            ↪  the length k");
    }
    // Handle non-wraparound part
    for (i = 0; i <= str.length() - k - 1; i++) {
        sub = obj_str.substr(i, k);
        sub_p1 = obj_str.substr(i, k + 1);
        freq_k_gram[sub]++;
        freq_kp1_gram[sub_p1]++;
    }

    // Wrap-around part (to include remaining k-grams
        ↪ that wrap around)
    for (; i < str.length(); i++) {
        std::string res = "", res1 = "";
        for (j = 0; j < k; j++) {
            res += obj_str[(i + j) % str.length()];
            res1 += obj_str[(i + j) % str.length()];
        }
        res1 += obj_str[(i + k) % str.length()];  // Add one more char for (k+1)-gram

        // std::cout << "i is " << i << " j is " << j << " k gram is "
        // << res << std::endl;

        // std::cout << "i is " << i << " j is " << j << " k+1_gram is "
        // << res1 << std::endl;
        freq_k_gram[res]++;
        freq_kp1_gram[res1]++;
    }
}


size_t RandWriter::orderK() const {
    return _order;
}

int RandWriter::freq(const std::string& kgram) const {
    // Throw an exception if kgram is not of length _order
    if (kgram.length() != _order) {
        throw std::invalid_argument("kgram must be of length k");
    }

    auto it = freq_k_gram.find(kgram);
    if (it != freq_k_gram.end()) {
        return it->second;
    }
    return 0;  // Return 0 if kgram is not found in the map
}

int RandWriter::freq(const std::string& kgram, char c) const {
    // Throw an exception if kgram is not of length _order
    if (kgram.length() != _order) {
        throw std::invalid_argument("kgram must be of length k");
    }

    if (_order == 0) {
      return std::count_if(obj_str.begin(), obj_str.end(),
    [c](char ch) {return ch == c;});
    }

     // Combine kgram + c to form a (k+1)-gram
     std::string kgram_plus_c = kgram + c;

     auto it = freq_kp1_gram.find(kgram_plus_c);
     if (it != freq_kp1_gram.end()) {
         return it->second;
     }
     return 0;
}

char RandWriter::kRand(const std::string& kgram) {
     // (throw an exception if kgram is not of length k)
    if (kgram.size() != _order) throw std::invalid_argument("kgram must be of length k");

    // (throw an exception if no such kgram)
    if (freq_k_gram.find(kgram) == freq_k_gram.end()) {
        throw std::invalid_argument("The kgram does not exist in the text");
    }

    std::vector<char> nChars;  // the next char after kgram
    std::vector<int> weights;  // frequency of the next char

    for (const auto& pair : freq_kp1_gram) {
        const std::string& kp1 = pair.first;
        if (kp1.size() == _order + 1 && kp1.substr(0, _order) == kgram) {
            nChars.push_back(kp1[_order]);
            weights.push_back(pair.second);
        }
    }

    static std::mt19937 engine(static_cast<unsigned>(
        std::chrono::system_clock::now().time_since_epoch().count()));

    // Create a weighted distribution that mirrors the frequencies.
    std::discrete_distribution<int> dist(weights.begin(), weights.end());

    // Pick an index and return the corresponding next character.
    int index = dist(engine);
    return nChars[index];
}

std::string RandWriter::generate(const std::string& kgram, size_t L) {
    std::string current = "";
    size_t i = 0;
    if (kgram.length() != _order) throw std::invalid_argument("kgram must be of length k");
    if (L < _order) throw std::invalid_argument("L must be at least k");
    current += kgram;
    L -= kgram.length();
    while (L > 0) {
        std::string sub_cur = current.substr(i, _order);
        current += std::string(1, kRand(sub_cur));
        L--;
        i++;
    }
    return current;
}

std::ostream &operator<<(std::ostream &os, const RandWriter &ran) {
    os << "Order "<< ran._order << std::endl;
    std::unordered_set<char> alphabet;
    for (char ch : ran.obj_str) {
        alphabet.insert(ch);
    }

    os << "Alphabet: ";
    for (char ch : alphabet) {
        os << ch << " ";
    }
    os << std::endl;

    os << "K-gram Frequencies:" << std::endl;
    for (const auto& pair : ran.freq_k_gram) {
        os << pair.first << ": " << pair.second << std
            ↪ ::endl;
    }

    os << std::endl;
    os << "K+1-gram Frequencies:" << std::endl;
    for (const auto& pair : ran.freq_kp1_gram) {
        os << pair.first << ": " << pair.second << std
            ↪ ::endl;
    }
    return os;
}
```

Listing 5: test.cpp

```
// Copyright 2025 Christopher Nguyen
#include <iostream>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include "RandWriter.hpp"

BOOST_AUTO_TEST_CASE(a_substring_is_less_than_k_gram)
    ↪ {
    RandWriter obj("aggcgagggagcggcagggg", 3);
    BOOST_REQUIRE_THROW(obj.freq("ag"), std::
        ↪ invalid_argument);
}

BOOST_AUTO_TEST_CASE(zero_gram) {
    RandWriter obj("aggcgagggagcggcagggg", 0);
    BOOST_REQUIRE_NO_THROW(obj.freq("", ’c’));
    BOOST_REQUIRE_EQUAL(obj.freq("", ’c’), 3);
}

BOOST_AUTO_TEST_CASE(string_length) {
    RandWriter obj("aggcgagggagcggcagggg", 3);
    std::string o = obj.generate("agg", 5);
    BOOST_REQUIRE_EQUAL(o.length(), 5);
}

```

```
BOOST_AUTO_TEST_CASE(right_string_start) {
    RandWriter obj("aggcgagggagcggcagggg", 3);
    std::string o = obj.generate("agg", 5);
    BOOST_REQUIRE_EQUAL(o.substr(0, 3), "agg");
}

BOOST_AUTO_TEST_CASE(test_for_right_distribution) {
    RandWriter obj("aggcgagggagcggcagggg", 3);
    BOOST_REQUIRE_NO_THROW(obj.generate("agg", 4));
    std::string o = obj.generate("agc", 4);
    BOOST_REQUIRE_EQUAL(o, "agcg");
}
```

PS7 Kronos Log Parsing

This project simulates a Kronos InTouch time-clock log. It is tracked by using regular expressions to parse the file. It also checks the device boot up timing. It is able to read through a log file, and produce a report file that tracks the server started and whether it gets completed to it fails. It can return its line number and the time and the time required for the completion should the start up complete.

Features that I implemented is adding date track.hpp and date track.cpp for this assignment. This is because I want to store the date and the time of the line should it get accepted. I can call an object that gets the whole sequence of date and time to be used to find the difference in time.

To accept the line with (1og.number.c) server started, I used a regular expression named re that follows any 4 digits that represents the year followed by a dash followed by any 2 digits followed by a dash and 2 digits. This is for the date. To represent the time, I used (2̣:2̣:2̣):. For the rest of the sequence, I used log and the .c and d + server started.

To accept a complete line, I follow 4 digits for the year, dashed, 2 digits for the month and 2 digits for the day with a space and 2 digits for the hour, 2 digits for the minutes and 2 digits for the seconds followed by d3:INFO:oejsȦbstractConnector:Started SelectChannelConnector@00̇0̇0̇:9080*$.

To store the date and time, I put parenthesis between the sub-parts that represent the date and the sub-parts that represent the time, excluding the milliseconds.

Before the while loop in the main file, I initialized a value that represents a line number to one. Each time the program traverses the log, the line tracker will also increase. So, when it is appropriate to store the line in the report file, it will display the right line number.

The overall approach to achieve the assignment's task is to traverse through a log file. You initialize a report file in the main. Then track down server started, completion, or incomplete into a report file. You can track the line number by setting an int number then incremented in the while loop.

The regular expression I used is:

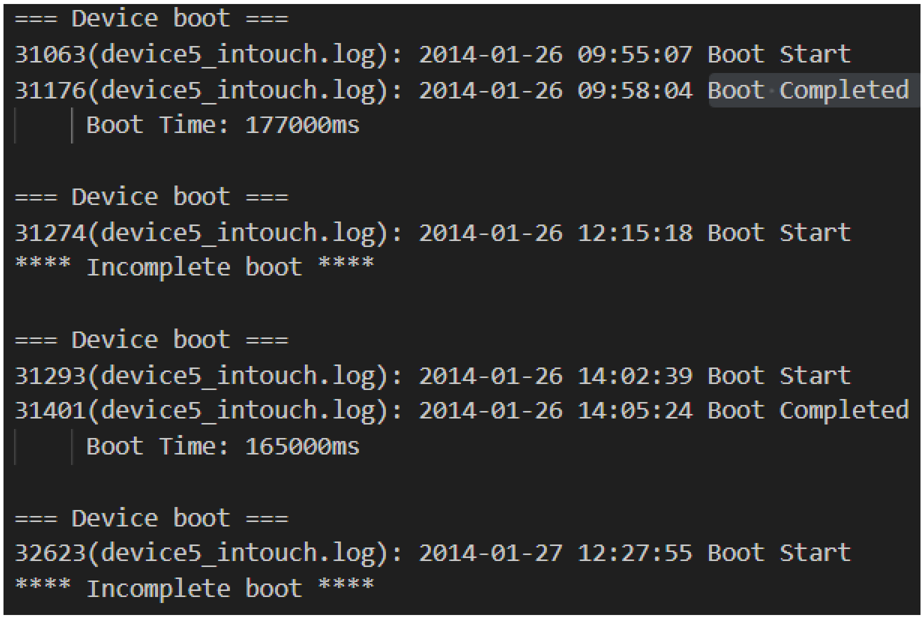
```
std::regex re(R"(^(\d{4}-\d{2}-\d{2}) (\d{2}:\d{2}:\d{2}):
\(log\.c\.\d+\) server started\s*$)");
```

This is used to check whether a line indicates that the log server has started.

I also used:

```
std::regex complete(R"(^(\d{4}-\d{2}-\d{2}) (\d{2}:\d{2}:\d{2})\.\d{3}:INFO:
oejs\.AbstractConnector:Started SelectChannelConnector@0\.0\.0\.0:9080\s*$)");
```

It is used to find the line that accepts a line that contains oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080.



```
=== Device boot ===
31063(device5_intouch.log): 2014-01-26 09:55:07 Boot Start
31176(device5_intouch.log): 2014-01-26 09:58:04 Boot Completed
        Boot Time: 177000ms

=== Device boot ===
31274(device5_intouch.log): 2014-01-26 12:15:18 Boot Start
**** Incomplete boot ****

=== Device boot ===
31293(device5_intouch.log): 2014-01-26 14:02:39 Boot Start
31401(device5_intouch.log): 2014-01-26 14:05:24 Boot Completed
        Boot Time: 165000ms

=== Device boot ===
32623(device5_intouch.log): 2014-01-27 12:27:55 Boot Start
**** Incomplete boot ****
```

Figure 1: Sample from the report file

Listing 1: Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework

# Your .hpp files
DEPS = date_track.hpp

# Your compiled .o files
MAIN_OBJ = main_ps7.o date_track.o

# The name of the program
PROGRAM = ps7

.PHONY: all clean lint

all: $(PROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
```

```
        $(CC) $(CFLAGS) -c $<

ps7: $(MAIN_OBJ)
        $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
        rm *.o $(PROGRAM)

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

Listing 2: main_ps7.cpp

```
// Copyright 2025 Christopher Nguyen
#include <iostream>
#include <string>
#include <fstream>
#include <regex>
#include "date_track.hpp"

// using namespace boost::posix_time;

int main(int argc, char* argv[]) {
    /* argv[1] holds the name of the log
    file in the command line
    example: device1_intouch.log */
    std::ifstream file(argv[1]);
    if (!file.is_open()) {
        std::cerr << "Failed to open file.\n";
        return 1;
    }

    std::string out = std::string(argv[1]) + ".rpt";
    std::ofstream report(out);

    if (!report) {
        std::cerr << "Error opening file.\n";
        return 1;
    }

    std::regex re(R"(^(\d{4}-\d{2}-\d{2}) (\d{2}:\d
        ↪ {2}:\d{2}): \(log\.c\.\d+\) server started\s
        ↪ *$)");
    std::regex complete(R"(^(\d{4}-\d{2}-\d{2}) (\d
        ↪ {2}:\d{2}:\d{2})\.\d{3}:INFO:oejs\.
        ↪ AbstractConnector:Started
```

```
SelectChannelConnector@0\.0\.0\.0:9080\s*$)");

    std::string line;
    std::smatch match;
    date_track obj;  // Needs a default constructor
    std::string start_line;
    int line_track = 1;
    bool flag = false;

    while (std::getline(file, line)) {
        if (std::regex_match(line, match, re) && !flag) {
            obj = date_track(match);  // Assign it here
            start_line = line;
            report << "=== Device boot ===" << std::endl;
            report << std::to_string(line_track) << "("<<
                std::string(argv[1]) << "): ";
            report << obj.get_full_sequence() << " Boot Start" << std::endl;
            flag = true;

        // if server started sucessfully boots
        } else if (std::regex_match(line, match, complete) /*&& flag*/) {
            date_track com_obj(match);
            boost::posix_time::ptime d2 =
                boost::posix_time::time_from_string(com_obj.get_full_sequence());

            boost::posix_time::ptime d1 =
                boost::posix_time::time_from_string(obj.get_full_sequence());

            // boost::posix_time::time_period tp(d1, d2);
            // boost::posix_time::time_duration dur = tp.length();
            boost::posix_time::time_duration dur = d2 - d1;
            report << std::to_string(line_track) << "("
                << std::string(argv[1]) << "): ";
            report << com_obj.get_full_sequence() << "
               ↪  Boot Completed" << std::endl;
            report << "\t Boot Time: " << dur.
               ↪ total_milliseconds() << "ms" << std
               ↪ ::endl
                << std::endl;
            flag = false;
        } else if (std::regex_match(line, match, re)
           ↪ && flag) {
            obj = date_track(match);  // Assign it
               ↪ here
            report << "**** Incomplete boot ****" <<
               ↪ std::endl <<std::endl;
            report << "=== Device boot ===" << std::
               ↪ endl;
            report << std::to_string(line_track) << "(
               ↪ "<<
                std::string(argv[1]) << "): ";
            report << obj.get_full_sequence() << "
               ↪ Boot Start" << std::endl;
            flag = false;
        }
        line_track++;
    }

    if (flag) {
        report << "**** Incomplete boot ****" << std::
           ↪ endl <<std::endl;
    }
    file.close();  // optional, since destructor
       ↪ closes it
    report.close();
    // std::cout << "Chris!!!!!\n";
    return 0;
}
```

Listing 3: date_track.hpp

```
// Copyright 2025 Christopher Nguyen
#pragma once
#ifndef DATE_TRACK_HPP
#define DATE_TRACK_HPP
#include <iostream>
#include <string>
#include <regex>
#include <boost/date_time/gregorian/gregorian.hpp>
```

```
#include <boost/date_time/posix_time/posix_time.hpp>

// class date_track {
//  public:
//     date_track(): match() {}
//     explicit date_track(const std::smatch& m): match(m) {}

//     std::string get_date() const;
//     std::string get_time() const;
//     void clear();

//     // Used to display full line of start
//     std::string get_full_sequence() const;
//  private:
//     std::smatch match;
// };

class date_track {
 public:
      date_track() = default;

      // Copy the substrings right away
      explicit date_track(const std::smatch& m)
         : date_{ m[1].str() },
            time_{ m[2].str() }
      {}
      std::string get_full_sequence() const;
 private:
      std::string date_, time_;
};



#endif
```

Listing 4: date_track.cpp

```
// Copyright 2025 Christopher Nguyen
#include "date_track.hpp"

std::string date_track::get_full_sequence() const {
    return date_ + " " + time_;
}
```