Peter Meas
COMPIV: Project Portfolio
Spring 2025

Table of Contents

1 PS0: Hello with SFML

1.1 Description

This project served as an introductory exercise to familiarize myself with the Simple and Fast Multimedia Library (SFML). The objective was to set up the SFML environment, render a sprite on the screen, and control its behavior using user input. When the user presses the 'F' key, the sprite toggles a mode where it follows the mouse cursor using calculated distance. This involved real-time input handling and graphical updates.

1.2 Features

One of the core features of the project was implementing real-time sprite movement using keyboard input. The program detects a key press ('F') to toggle the following behavior of the sprite. Additionally, the Euclidean distance between the sprite and mouse cursor is continuously calculated, allowing the sprite to move toward the cursor position in a smooth, natural motion.

1.3 Design Decisions and Implementation

The primary algorithm used in this project was the Euclidean distance formula, derived from the Pythagorean theorem. This algorithm determines the shortest path between two points in a 2D space, which is used to guide the sprite movement. SFML classes like sf::Vector2f were utilized to manage positions, and sf::Mouse for real-time cursor tracking

1.4 What I Learned

This project helped me understand the structure of an SFML application and how to manage a render window, event polling, and real-time updates. I also gained experience working with coordinate systems and understanding how graphics are updated frame-by-frame. Additionally, I learned the importance of input handling and timing to achieve fluid animations.

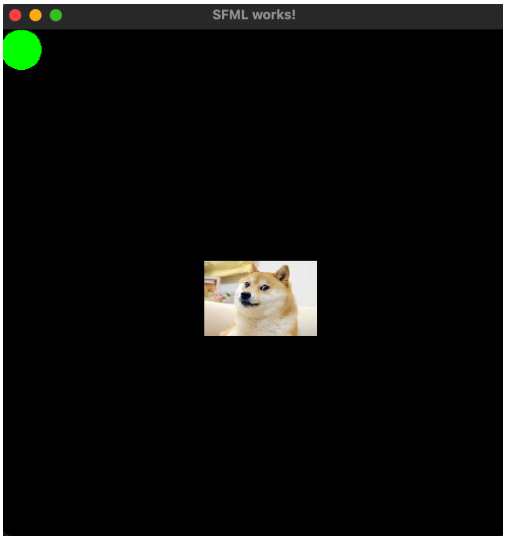
1.5 Issues

One challenge I encountered was ensuring smooth animation. Initially, the sprite's movement was jittery due to overly frequent position updates and inconsistent frame times. I resolved this by adjusting the movement increment to smooth out transitions.

1.6 Extra Credit

For extra credit, I implemented a toggle mechanism where pressing the 'F' key switches the sprite's following behavior on and off. This required additional logic to handle state persistence between frames and input events.

1.7 Output



1.8 Code

Makefile:

```
1. CC = g++
2. CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
3. LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework
4. # Your .hpp files
5. DEPS =
6. # Your compiled .o files
7. OBJECTS =
8. # The name of your program
9. PROGRAM = sfml-app
10.
11. .PHONY: all clean lint
12.
13.
14. all: $(PROGRAM)
15.
16. # Wildcard recipe to make .o files from corresponding .cpp file
17. %.o: %.cpp $(DEPS)
18.     $(CC) $(CFLAGS) -c $<
19.
20. $(PROGRAM): main.o $(OBJECTS)
21.     $(CC) $(CFLAGS) -o $@ $^ $(LIB)
22.
23. clean:
24.     rm *.o $(PROGRAM)
25.
26. lint:
27.     cpplint *.cpp *.hpp
28.
```

main.cpp:

```
1. // Copyright 2025 Peter Meas //
2. #include <iostream>
```

```
3. #include <cmath>
4. #include <SFML/Graphics.hpp>
5.
6.
7. int main() {
8.     sf::Texture texture;
9.     float changeX = .05f;
10.     float changeY =  .05f;
11.     sf::Clock timer;
12.     sf::Time tick;
13.     sf::Vector2f velo(25.0, 10.0);
14.     texture.loadFromFile("./sprite.png");
15.     // setup our Sprite
16.     sf::Sprite sprite;
17.     sprite.setTexture(texture);
18.     sprite.setScale(.43f, .38f);
19.     sf::RenderWindow window(sf::VideoMode(500, 500), "SFML works!");
20.     sf::CircleShape shape(100.f);
21.     shape.setScale(0.2f, 0.2f);
22.     shape.setFillColor(sf::Color::Green);
23.
24.     sprite.setPosition(200, 200);
25.     bool isToggled = false;
26.     const float move = 1.0f;
27.     const float followDist = 3.0f;
28.
29.     while (window.isOpen()) {
30.         sf::Event event;
31.         while (window.pollEvent(event)) {
32.             if (event.type == sf::Event::Closed) {
33.                 window.close();  // put
34.             }
35.                 if (event.type == sf::Event::KeyPressed) {
36.             if (event.key.code == sf::Keyboard::F) {
37.                 isToggled = !isToggled;
38.             }
39.         }
40.
41.         if (isToggled) {
42.             sf::Vector2i  mousePos = sf::Mouse::getPosition(window);
43.             sf::Vector2f targetPos = window.mapPixelToCoords(mousePos);
44.
45.             sf::Vector2f spritePos = sprite.getPosition();
46.
47.             float dx = targetPos.x - spritePos.x;
48.             float dy = targetPos.y - spritePos.y;
49.             float distance = std::sqrt(dx * dy  + dy * dy);
50.
51.             if (distance > followDist) {
52.                 sprite.move(move * (dx/distance), move*(dy/distance));
53.             }
54.         }
55.         }
```

```
56.
57.
58.         if (!sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
59.             if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
60.             sprite.move(0, -changeY);
61.             }
62.            if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
63.             sprite.move(0, changeY);
64.            }
65.            if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
66.             sprite.move(-changeX, 0);
67.            }
68.            if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
69.             sprite.move(changeX, 0);
70.            }
71.
72.
73.         sf::Vector2u windowSize = window.getSize();
74.         sf::FloatRect spriteSize = sprite.getGlobalBounds();
75.         bool touchingBorder = false;
76.            if (spriteSize.left < 0) {
77.             sprite.setPosition(0, sprite.getPosition().y);
78.             touchingBorder = true;
79.            }
80.            if (spriteSize.top < 0) {
81.             sprite.setPosition(sprite.getPosition().x, 0);
82.             touchingBorder = true;
83.            }
84.         if (spriteSize.left + spriteSize.width > windowSize.x) {
85.             sprite.setPosition(windowSize.x - spriteSize.width, sprite.getPosition().y);
86.             touchingBorder = true;
87.         }
88.         if (spriteSize.top + spriteSize.height > windowSize.y) {
89.             sprite.setPosition(sprite.getPosition().x, windowSize.y - spriteSize.height);
90.             touchingBorder = true;
91.         }
92.
93.         if (touchingBorder) {
94.             sprite.setColor(sf::Color::Red);
95.         } else {
96.             sprite.setColor(sf::Color::White);
97.         }
98.             tick = timer.restart();
99.         sprite.move(velo * (static_cast<float>(tick.asMilliseconds() / 1000)));
100.             window.clear();
101.             window.draw(sprite);
102.             window.draw(shape);
103.             window.display();
104.         }
105.     }
106.     return 0;
107. }
108.
```

2 PS1: Linear Feedback Shift Register and Image Encoding

2.1 Description

For this project, I designed and implemented a Linear Feedback Shift Register class to perform encryption and decryption on images. The LFSR, which is a simple yet effective pseudorandom number generator, was used to modify the pixel values of an image by XORing them with bits generated by the LFSR. This project provided insight into bit manipulation and image processing, especially in terms of how data encoding and decoding techniques can be applied to graphical content.

2.2 Features

This project involved building a Linear Feedback Shift Register class to perform image encryption and decryption. By manipulating pixel RGB values using an LFSR, I was able to alter an image and restore it back to its original form. The encryption and decryption are achieved by XORing each pixel's RGB value with values generated by the LFSR.

2.3 Design Decisions and Implementation

The primary algorithm in this project is based on bitwise XOR operations, where an LFSR generates a sequence of bits that are XORed with pixel data for encryption and decryption. The class uses std::bitset<16> to represent the state of the LFSR and std::vector to store pixel data. The generate() function updates the LFSR state on each iteration, generating new values used in the XOR operation for both encryption and decryption.

2.4 What I Learned

I gained experience in bit manipulation and working with std::bitset for managing the LFSR state. Additionally, I learned how to manipulate images using SFML and apply bitwise operations to encode and decode images. The project also helped me practice writing unit tests using the Boost framework to validate encryption and decryption functionality.

2.5 Issues

I encountered no major issues in the implementation, as the encryption and decryption processes function as expected. The most significant challenge was ensuring that the XOR operation was applied correctly to the pixel values, which was resolved by thorough testing.

2.6 Extra Credit

I did not add any extra credit for this assignment.

2.7 Output

2.8 Code

Makefile:

```
1. CC = g++
2. CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3. LIB = -lboost_unit_test_framework -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
4.
5. .PHONY: all clean lint
6.
7. CXXFLAGS = $(CFLAGS)
8.
9. all: PhotoMagic test lint
10.
11. PhotoMagic: main.cpp PhotoMagic.a
12.     $(CC) $(CFLAGS) main.cpp PhotoMagic.a -o PhotoMagic $(LIB)
13.
14. test: test.cpp PhotoMagic.a
15.     $(CC) $(CFLAGS) test.cpp PhotoMagic.a -o test $(LIB)
16.
17. PhotoMagic.a: FibLFSR.o PhotoMagic.o
18.     ar rcs PhotoMagic.a FibLFSR.o PhotoMagic.o
19.
20. FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
21.     $(CC) $(CFLAGS) -c FibLFSR.cpp -o FibLFSR.o
22.
23. clean:
24.     rm -f *.o *.a PhotoMagic test
25.
26. lint:
27.     cpplint --repository=. *.cpp *.hpp
28.
```

Photomagic.cpp:

```
1. // Copyright 2025 Peter Meas //
2. #include "PhotoMagic.hpp"
```

```
3. #include <fstream>
4.
5. void PhotoMagic::transform(sf::Image& img, FibLFSR* lfsr) {
6.     // test : std::cout << "lfsr: "<< lfsr->generate(9) <<std::endl;
7.     sf::Color p;
8.     for (unsigned int y = 0; y < img.getSize().y; y++) {
9.         for (unsigned int x=0; x < img.getSize().x; x++) {
10.             p = img.getPixel(x, y);
11.             int red = p.r;
12.             int green = p.g;
13.             int blue = p.b;
14.
15.             int xorred = red ^ lfsr->generate(10);
16.             int xorgreen = green ^ lfsr->generate(10);
17.             int xorblue = blue ^ lfsr->generate(10);
18.
19.             sf::Color newP(xorred, xorgreen, xorblue);
20.             img.setPixel(x, y, newP);
21.         }
22.     }
23. }
24.
```

FibLFSR.cpp:

```
1. // Copyright 2025 Peter Meas //
2. #include "FibLFSR.hpp"
3. #include <stdexcept>
4. #include <iostream>
5. #include <string>
6.
7. std::ostream& PhotoMagic::operator<<(std::ostream& os, const FibLFSR& ls) {
8.     os << ls.registerBits;
9.     return os;
10. }
11.
12. PhotoMagic::FibLFSR::FibLFSR(const std::string& seed) {
13.     if ((seed.length()) != 16) {
14.         throw std::invalid_argument("Seed must be 16 bits long");
15.             }
16.         for (char c : seed) {
17.             if (c != '0' && c != '1') {
18.                     throw std::invalid_argument("Seed must only be 0s and 1s");
19.         }
20.     }
21.     registerBits = std::bitset<16>(seed);
22. }
23.
24. int PhotoMagic::FibLFSR::step() {
```

```
25.     int leftmostBit = registerBits[15];
26.
27.     int tap13 = registerBits[13];
28.     int tap12 = registerBits[12];
29.     int tap10 = registerBits[10];
30.     int feedbackBit_ = leftmostBit ^ tap13 ^ tap12 ^ tap10;
31.     registerBits.operator<<=(1);
32.     registerBits.set(0, feedbackBit_);
33.     return feedbackBit_;
34. }
35.
36. int PhotoMagic::FibLFSR::generate(int k) {
37.     if (k < 0) {
38.         throw std::invalid_argument("Negative input");
39.     }
40.     if (k > 16) {
41.         throw std::invalid_argument("Higher than 16 bits");
42.     }
43.
44.     int extractedBits = 0;
45.     for (int i = 0; i < k; i++) {
46.         extractedBits = (extractedBits * 2) + step();
47.     }
48.     return extractedBits;
49. }
50.
```

FibLFSR.hpp:

```
1. // Copyright 2025 Peter Meas //
2. #pragma once
3. #include <iostream>
4. #include <string>
5. #include <bitset>
6.
7. namespace PhotoMagic {
8. class FibLFSR {
9. friend std::ostream& operator<<(std::ostream& os, const FibLFSR& lfsr);
10.
11.  public:
12.     explicit FibLFSR(const std::string& seed);
13.     explicit FibLFSR(unsigned int seed);  // Optional
14.
15.     static FibLFSR fromPassword(const std::string& password);  // Optional
16.
17.     int step();
18.     int generate(int k);
19.  private:
20.       std::bitset<16> registerBits;
```

```
21. };
22. std::ostream& operator<<(std::ostream& os, const FibLFSR& lfsr);
23.
24.
25. }  // namespace PhotoMagic
26.
```

PhotoMagic.hpp:

```
1. // Copyright 2025 Peter Meas //
2. #pragma once
3.
4. #include <SFML/Graphics.hpp>
5. #include "./FibLFSR.hpp"
6.
7. namespace PhotoMagic {
8. void transform(sf::Image& img, FibLFSR* lfsr);
9.
10. }
11.
```

test.cpp:

```
1. // Copyright 2022
2. // By Dr. Rykalova
3. // Editted by Dr. Daly
4. // test.cpp for PS1a
5. // updated 1/8/2024
6.
7. #include <iostream>
8. #include <string>
9.
10. #include "./FibLFSR.hpp"
11. #include "./PhotoMagic.hpp"
12.
13. #define BOOST_TEST_DYN_LINK
14. #define BOOST_TEST_MODULE Main
15. #include <./boost/test/unit_test.hpp>
16.
17.
18.
19. using PhotoMagic::FibLFSR;
20.
21. BOOST_AUTO_TEST_CASE(testStepInstr) {
22.   FibLFSR l("1011011000110110");
23.   BOOST_REQUIRE_EQUAL(l.step(), 0);
24.   BOOST_REQUIRE_EQUAL(l.step(), 0);
25.   BOOST_REQUIRE_EQUAL(l.step(), 0);
26.   BOOST_REQUIRE_EQUAL(l.step(), 1);
27.   BOOST_REQUIRE_EQUAL(l.step(), 1);
28.   BOOST_REQUIRE_EQUAL(l.step(), 0);
29.   BOOST_REQUIRE_EQUAL(l.step(), 0);
30.   BOOST_REQUIRE_EQUAL(l.step(), 1);
```

```
31. }
32.
33. BOOST_AUTO_TEST_CASE(testStep2) {
34.   FibLFSR lfsr("0110110001101100");
35.   BOOST_CHECK_EQUAL(lfsr.step(), 0);
36.   BOOST_CHECK_EQUAL(lfsr.step(), 0);
37.   BOOST_CHECK_EQUAL(lfsr.step(), 1);
38. }
39.
40.
41. BOOST_AUTO_TEST_CASE(testGenerateInstr) {
42.   FibLFSR l("1011011000110110");
43.   BOOST_REQUIRE_EQUAL(l.generate(9), 51);
44. }
45.
46. BOOST_AUTO_TEST_CASE(testGenerate1) {
47.   FibLFSR V("1011011000110110");
48.   std::ostringstream os;
49.   os << V;
50.   BOOST_CHECK_EQUAL(V.generate(5), 3);
51.   std::ostringstream os_after3;
52.   os_after3 << V;
53.   BOOST_REQUIRE_EQUAL(os_after3.str(), "1100011011000011");
54.
55.   BOOST_CHECK_EQUAL(V.generate(5), 6);
56.   std::ostringstream os_after6;
57.   os_after6 << V;
58.   BOOST_REQUIRE_EQUAL(os_after6.str(), "1101100001100110");
59.   BOOST_CHECK_EQUAL(V.generate(5), 14);
60.   std::ostringstream os_after14;
61.   os_after14 << V;
62.   BOOST_REQUIRE_EQUAL(os_after14.str(), "0000110011001110");
63. }
64.
65.
66.
67. BOOST_AUTO_TEST_CASE(testOutputOperator) {
68.     PhotoMagic::FibLFSR lfsr("0110110001101100");
69.     std::ostringstream os;
70.     os << lfsr;
71.     BOOST_CHECK_EQUAL(os.str(), "0110110001101100");
72.
73.     lfsr.step();
74.     std::ostringstream os_after_step;
75.     os_after_step << lfsr;
76.     BOOST_REQUIRE_EQUAL(os_after_step.str(), "1101100011011000");
77. }
78.
79. BOOST_AUTO_TEST_CASE(testInvalidConstructor) {
80.   BOOST_CHECK_THROW(FibLFSR invalid_lfsr("invalid_input"), std::invalid_argument);
81.   BOOST_CHECK_NO_THROW(FibLFSR valid_lfsr("1011011000110110"));
82. }
83.
```

```
84. BOOST_AUTO_TEST_CASE(testInvalidGen) {
85.   FibLFSR lfsr("1011011000110110");
86.   BOOST_CHECK_THROW(lfsr.generate(-1), std::invalid_argument);
87. }
88.
```

3 PS2: Sierpinski Triangle Fractal

3.1 Description

This project involved generating a recursive Sierpinski Triangle fractal using SFML. The program takes two command-line arguments: the initial triangle side length and the recursion depth. It recursively subdivides the triangle into smaller triangles, producing a colorful, mathematically accurate fractal pattern.

3.2 Features

This project uses recursive subdivision to draw a Sierpinski triangle fractal in SFML. I calculated the positions of child triangles using midpoint formulas and carefully tuned offsets to maintain symmetry. I also implemented dynamic window sizing based on the initial triangle side length, input validation for command-line arguments, and a color-cycling system to distinguish recursion levels.

3.3 Design Decisions and Implementation

The fractal is built recursively by halving the side length at each level. I calculated triangle positions using trigonometry and midpoint formulas, storing points with sf::Vector2f. Specific scaling ratios like 1.33 and 1.99 were chosen to ensure correct spacing between recursive levels. I also used modular arithmetic to cycle colors.

3.4 What I Learned

Prior to this project, I understood recursion but had not applied it to graphics. I learned how to translate mathematical rules into recursive rendering and had practice with working with geometric properties like triangle centroids and height. I also learned how small visual adjustments can significantly impact the clarity of recursive graphics.

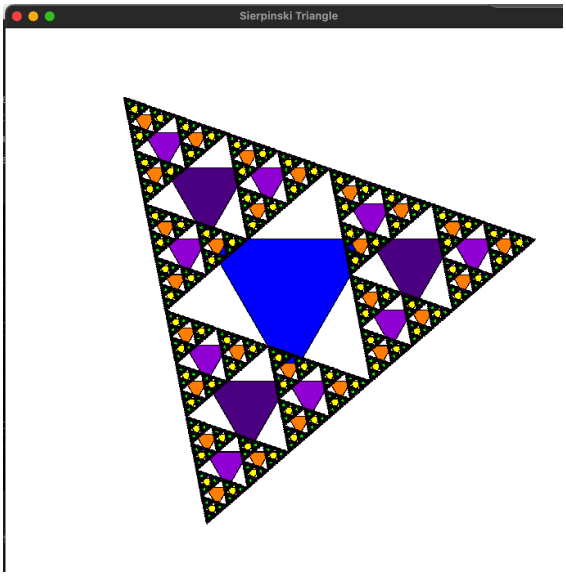
3.5 Issues

One issue was getting the child triangle positions correctly aligned. I initially tried pure geometric formulas, but they created overlapping or floating triangles. I refined them using trial and error. I also had issues with color clarity at higher recursion depths.

3.6 Extra Credit

I added a function that assigns rainbow colors to triangles based on depth. This enhances the visual appeal and helps distinguish recursive levels.

3.7 Output



3.8 Code

Makefile:

```
1. CC = g++
2. CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3. LIB = -lboost_unit_test_framework -lsfml-graphics -lsfml-audio -lsfml-window
-lsfml-system
4.
5. all: Triangle lint
6.
7. Triangle: main.cpp Triangle.o
8.     $(CC) $(CFLAGS) main.cpp Triangle.o -o Triangle $(LIB)
9.
10. Triangle.o: triangle.cpp triangle.hpp
11.    $(CC) $(CFLAGS) -c triangle.cpp -o Triangle.o
12.
13. lint:
14.    cpplint --repository=. *.cpp *.hpp
15. clean:
16.    rm -f *.o *.a Triangle
```

main.cpp:

```
1. // Copyright 2025 Peter Meas
2. #include <iostream>
3. #include <cstdlib>
4. #include <algorithm>
5. #include "triangle.hpp"
6. #include <SFML/Graphics.hpp>
7.
8. int main(int argc, char** argv) {
9.     if (argc != 3) {
10.         std::cerr << "Invalid arguments!" << std::endl;
```

```
    } else {
        double L = std::stod(argv[1]);
        int N = std::stoi(argv[2]);
        int windowSize = static_cast<int>(2 * std::sqrt(3.0) * L);
        // Create window with calculated dimensions
        sf::RenderWindow window(sf::VideoMode(windowSize, windowSize), "Sierpinski Triangle");
        // Center the triangle in the window
        Triangle triangle(L, sf::Vector2f(windowSize / 2, windowSize / 2));

        while (window.isOpen()) {
            sf::Event event;
            while (window.pollEvent(event)) {
                if (event.type == sf::Event::Closed) {
                    window.close();
                }
            }
            window.clear(sf::Color::White);
            triangle.fractal(window, N);
            window.display();
        }
    }
    return 0;
}

```

triangle.cpp:

```
// Copyright 2025 Peter Meas
#include "triangle.hpp"

Triangle::Triangle(double sideLength, sf::Vector2f centroidPos)
    : L(sideLength), centroid(centroidPos) {
    height = static_cast<float>((sqrt(3) / 2) * L);
    shape.setPointCount(3);
    shape.setPoint(0, sf::Vector2f(0, height / 2));  // BOTTOM VERTEX
    shape.setPoint(1, sf::Vector2f(-L / 2, -height / 2));  // TOP LEFT
     shape.setPoint(2, sf::Vector2f(L / 2, -height / 2));  // TOP RIGHT
    shape.setOutlineColor(sf::Color::Black);
     shape.setOutlineThickness(.2f);
     shape.setPosition(centroid);
}


sf::Color Triangle::getDepthColor(int depth) {
    // Use modulo to cycle through colors
    switch (depth % 6) {
        case 0:
            return sf::Color(148, 0, 211);    // Purple
        case 1:
            return sf::Color(75, 0, 130);     // Indigo
        case 2:
            return sf::Color(0, 0, 255);      // Blue
        case 3:
```

```
27.             return sf::Color(0, 255, 0);      // Green
28.         case 4:
29.             return sf::Color(255, 255, 0);    // Yellow
30.         case 5:
31.             return sf::Color(255, 127, 0);    // Orange
32.         default:
33.             return sf::Color(255, 0, 0);      // Red
34.     }
35. }
36.
37. void Triangle::draw(sf::RenderTarget& target, sf::RenderStates state)const {
38.      target.draw(shape, state);
39. }
40.
41. void Triangle::fractal(sf::RenderTarget& target, int n) {
42.     if (n < 0) return;
43.     shape.setFillColor(getDepthColor(n));
44.     target.draw(*this);
45.     double newSideLength = L/2;
46.     double height = (sqrt(3) / 2) * L;
47.
48.     // Calculate positions for the three outer triangles
49.     sf::Vector2f bottom(
50.         centroid.x - L/4,
51.         centroid.y + height/1.33
52. );
53.
54.     sf::Vector2f left(
55.         centroid.x - L/1.99,
56.         centroid.y - height/1.33
57. );
58.
59.     sf::Vector2f right(
60.         centroid.x + L/1.33,
61.         centroid.y - height/4
62. );
63.     // Recursively draw the three outer triangles
64.     Triangle bottomTriangle(newSideLength, bottom);
65.     Triangle leftTriangle(newSideLength, left);
66.     Triangle rightTriangle(newSideLength, right);
67.
68.     bottomTriangle.fractal(target, n-1);
69.     leftTriangle.fractal(target, n-1);
70.     rightTriangle.fractal(target, n-1);
71. }
72.
```

triangle.hpp:

```
1. // Copyright 2025 Peter Meas
2. #pragma once
3. #include <iostream>
4. #include <cmath>
5. #include <SFML/Graphics.hpp>
6.
7.
8. class Triangle : public sf::Drawable{
9.  public:
10.     //  Triangle(double sideLength = 0, sf::Vector2f centroidPos = sf::Vector2f(0,0));
11.     Triangle(double sideLength, sf::Vector2f centroidPos);
12.     void draw(sf::RenderTarget& target, sf::RenderStates state) const override;
13.     void fractal(sf::RenderTarget& target, int n);
14.     sf::Color getDepthColor(int depth);
15.  private:
16.     double height;
17.     double L;
18.     sf::Vector2f centroid;
19.     sf::ConvexShape shape;
20.
21.     void drawTriangle(sf::RenderWindow& window, sf::Vector2f pos,
22.     double size, int depth, float rotation);
23. };
24.
```

4 PS3: N-Body Simulation

4.1 Description

This project simulates gravitational interactions between celestial bodies in a 2D space using Newtonian physics. The program reads initial body positions, velocities, masses, and image files from a data file, then visualizes the system's evolution in an SFML window.

4.2 Features

I created a simulation of gravitational interactions between celestial bodies using Newton's law of universal gravitation. The system calculates all pairwise forces and updates positions over time using delta time integration. I structured the program using Universe and CelestialBody classes and managed textures with std::shared_ptr. Planets are rendered using SFML, and their positions are scaled to fit the window dynamically.

4.3 Design Decisions and Implementation

I structured the program using object-oriented principles, separating celestial body logic from the universe container. For physics, I used Newton's law of gravitation with $O(n^2)$ complexity. I used std::vector to store dynamic collections of bodies and computed net forces for each body before updating motion. I also implemented a step function and added a real-time day counter overlay.

4.4 What I Learned

I learned how to implement physical simulations in code, including force accumulation and motion updates over time. I also developed a better understanding of memory management with smart pointers and how to apply object-oriented design to model interacting systems. Working with simulation scale and real-time rendering deepened my experience with SFML.

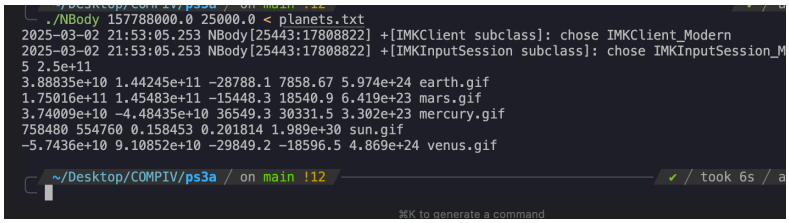
4.5 Issues

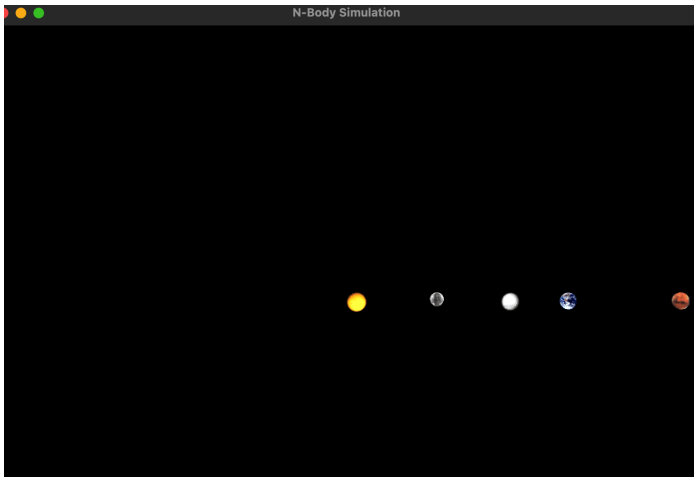
Initially, the program displayed a black screen. After debugging, I discovered that an incorrect scaling factor was rendering planets off-screen. Additionally, a missing default constructor for CelestialBody causes linker errors. I resolved these by adding a constructor and refining scaling logic.

4.6 Extra Credit

I added a timer to the SFML window that displays the number of simulated days since the start of the simulation.

4.7 Output





4.8 Code

Makefile:

```
1. CC = g++
2. CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
3. LIB = -lboost_unit_test_framework -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
4.
5. all: NBody NBody.a test
6.
7. NBody: main.o CelestialBody.o Universe.o
8.     $(CC) $(CFLAGS) -o NBody main.o CelestialBody.o Universe.o $(LIB)
9.
10. NBody.a: main.o CelestialBody.o Universe.o
11.     ar rvs NBody.a main.o CelestialBody.o Universe.o
12.
13. main.o: main.cpp Universe.hpp CelestialBody.hpp
14.     $(CC) $(CFLAGS) -c main.cpp
15.
16. CelestialBody.o: CelestialBody.cpp CelestialBody.hpp
17.     $(CC) $(CFLAGS) -c CelestialBody.cpp
18.
19. Universe.o: Universe.cpp Universe.hpp CelestialBody.hpp
20.     $(CC) $(CFLAGS) -c Universe.cpp
21.
22. test: test.o CelestialBody.o Universe.o
23.     $(CC) $(CFLAGS) -o test test.o CelestialBody.o Universe.o $(LIB)
24.
25. test.o:    test.cpp CelestialBody.hpp Universe.hpp
```

```
26.     $(CC) $(CFLAGS) -c test.cpp
27.
28. lint:
29.     cpplint --repository=. *.cpp *.hpp
30. clean:
31.     rm -f *.o *.a NBody test
```

main.cpp:

```
1. // Copyright 2025 Peter Meas
2. #include <iostream>
3. #include <fstream>
4. #include <sstream>
5. #include <iomanip>
6. #include "Universe.hpp"
7. #include <SFML/Graphics.hpp>
8.
9. int main(int argc, char* argv[]) {
10.     if (argc != 3) {
11.         std::cerr << "Usage: " << argv[0] << " T dt < universe_file\n";
12.         return 1;
13.     }
14.     bool fontLoad = false;
15.
16.     double T = std::stod(argv[1]);     // Total simulation time
17.     double dt = std::stod(argv[2]);    // Time step
18.
19.         sf::RenderWindow window(sf::VideoMode(800, 600), "N-Body Simulation");
20.         window.setFramerateLimit(60);
21.     sf::Font font;
22.     if (!font.loadFromFile("arial.ttf")) {
23.         std::cerr << "Couldn't load font" << std::endl;
24.     } else {
25.         fontLoad = true;
26.     }
27.
28.     sf::Text timetext;
29.     if (fontLoad) {
```

```
30.     timetext.setFont(font);
31.     timetext.setCharacterSize(20);
32.     timetext.setFillColor(sf::Color::White);
33.     timetext.setPosition(10, 10);
34.     }
35.
36.     sf::RectangleShape timerBackground(sf::Vector2f(200, 30));
37.     timerBackground.setFillColor(sf::Color(0, 0, 0, 128));
38.     timerBackground.setPosition(10, 10);
39.
40.     NB::Universe universe(window.getSize());
41.     try {
42.         universe.loadFromStream(std::cin);
43.     } catch (const std::exception& e) {
44.         std::cerr << "Error: " << e.what() << "\n";
45.         return 1;
46.     }
47.
48.
49.
50.     double elapsed_time = 0.0;
51.     sf::Clock clock;
52.
53.     while (window.isOpen() && elapsed_time < T) {
54.         sf::Event event;
55.         while (window.pollEvent(event)) {
56.             if (event.type == sf::Event::Closed) {
57.                 window.close();
58.             }
59.         }
60.         universe.step(dt);
61.         elapsed_time += dt;
62.
63.         std::ostringstream timeStr;
64.         double days = elapsed_time / 86400.0;
65.         timeStr << "Elasped time" << std::fixed << std::setprecision(2) << days
<< "days";
```

```
66.
67.
68.         if (fontLoad) {
69.             timetext.setString(timeStr.str());
70.         }
71.
72.         window.clear();
73.         window.draw(universe);
74.
75.         if (fontLoad) {
76.             window.draw(timerBackground);
77.             window.draw(timetext);
78.         } else {
79.             window.draw(timerBackground);
80.             sf::RectangleShape bar(sf::Vector2f((elapsed_time / T) * 180, 10));
81.             bar.setFillColor(sf::Color::Yellow);
82.             bar.setPosition(20, 20);
83.             window.draw(bar);
84.         }
85.         window.display();
86.     }
87.     std::cout << universe;
88.     return 0;
89. }
90.
```

Universe.cpp:

```
1. // Copyright 2025 Peter Meas
2. #include <iostream>
3. #include <fstream>
4. #include <cmath>
5. #include <vector>
6. #include <memory>
7. #include "Universe.hpp"
8. #include "CelestialBody.hpp"
9.
10. namespace NB {
```

```
11.
12. Universe::Universe(const sf::Vector2u& windowSize) : windowSize(windowSize) {}
13.
14. Universe::Universe() : numBodies(0), universeRadius(0) {}
15.
16. Universe::Universe(const std::string& filename) {
17.     std::ifstream inputFile(filename);
18.     if (!inputFile) {
19.         throw std::runtime_error("Failed to open file: " + filename);
20.     }
21.     inputFile >> *this;
22. }
23.
24. void Universe::loadFromStream(std::istream& is) {
25.     if (!(is >> numBodies >> universeRadius)) {
26.         throw std::runtime_error("Failed to read universe parameters");
27.     }
28.
29.     if (numBodies < 0) {
30.         throw std::runtime_error("Invalid number of bodies");
31.     }
32.     if (universeRadius <= 0) {
33.         throw std::runtime_error("Universe radius must be greater than zero");
34.     }
35.
36.     bodies.clear();
37.     for (size_t i = 0; i < numBodies; ++i) {
38.         double x, y, vx, vy, mass;
39.         std::string imageFilename;
40.
41.         if (!(is >> x >> y >> vx >> vy >> mass >> imageFilename)) {
42.             throw std::runtime_error("Failed to read celestial body data at index "
43.             + std::to_string(i));
44.         }
45.         if (mass < 0) {
46.             throw std::runtime_error("Invalid mass for celestial body at index "
47.             + std::to_string(i));
48.         }
49.         if (imageFilename.empty()) {
50.             throw std::runtime_error("Missing image filename for celestial body at index "
51.             + std::to_string(i));
```

```
        }
        auto body = std::make_shared<CelestialBody>(x, y, vx, vy, mass,
        imageFilename, universeRadius, windowSize);
        bodies.push_back(body);
    }
}


size_t Universe::size() const {
    return numBodies;
}
double Universe::radius() const {
    return universeRadius;
}

const CelestialBody& Universe::operator[](size_t i) const {
    if (i >= numBodies) {
        throw std::out_of_range("Index out of range");
    }
    return *bodies[i];
}

void Universe::draw(sf::RenderTarget& window, sf::RenderStates states) const {
    for (const auto& body : bodies) {
        window.draw(*body, states);
    }
}

std::istream& operator>>(std::istream& is, Universe& uni) {
    // Read number of bodies and universe radius
    if (!(is >> uni.numBodies >> uni.universeRadius)) {
        throw std::runtime_error("Failed to read universe parameters");
    }

    if (uni.numBodies < 0) {
        throw std::runtime_error("Invalid number of bodies");
    }
    if (uni.universeRadius <= 0) {
        throw std::runtime_error("Universe radius must be greater than zero");
    }

```

```
93.     uni.bodies.clear();
94.
95.     for (size_t i = 0; i < uni.numBodies; ++i) {
96.         auto body = std::make_shared<CelestialBody>();
97.         is >> *body;
98.
99.         if (is.fail()) {
100.             throw std::runtime_error("Failed to read celestial body data at index"
101.             + std::to_string(i));
102.         }
103.         body->setUniverseRadius(uni.universeRadius);
104.         uni.bodies.push_back(body);
105.     }
106.     return is;
107. }
108.
109. std::ostream& operator<<(std::ostream& os, const Universe& uni) {
110.     os << uni.numBodies << " " << uni.universeRadius << std::endl;
111.     for (const auto& body : uni.bodies) {
112.         os << *body << std::endl;
113.     }
114.     return os;
115. }
116.
117. void Universe::step(double dt) {
118.     const double G = 6.67e-11;  // gravitational constant
119.     std::vector<double> fx(numBodies, 0.0);
120.     std::vector<double> fy(numBodies, 0.0);
121.     std::vector<double> new_x(numBodies);
122.     std::vector<double> new_y(numBodies);
123.     std::vector<double> new_vx(numBodies);
124.     std::vector<double> new_vy(numBodies);
125.
126.     for (size_t i = 0; i < numBodies; ++i) {
127.         sf::Vector2f pos_i = bodies[i]->position();
128.         // sf::Vector2f vel_i = bodies[i]->velocity();
129.         float mass_i = bodies[i]->mass();
130.
131.         for (size_t j = 0; j < numBodies; ++j) {
132.             if (i == j) continue;
133.             sf::Vector2f pos_j = bodies[j]->position();
```

```
            float mass_j = bodies[j]->mass();

            double dx = pos_j.x - pos_i.x;
            double dy = pos_j.y - pos_i.y;

            double r_squared = dx * dx + dy * dy + 1e-10;
            double r = std::sqrt(r_squared);

            double force = G * mass_i * mass_j / r_squared;

            fx[i] += force * dx / r;
            fy[i] += force * dy / r;
        }
    }

    for (size_t i = 0; i < numBodies; ++i) {
        sf::Vector2f pos_i = bodies[i]->position();
        sf::Vector2f vel_i = bodies[i]->velocity();
        float mass_i = bodies[i]->mass();

        double ax = fx[i] / mass_i;
        double ay = fy[i] / mass_i;

        new_vx[i] = vel_i.x + ax * dt;
        new_vy[i] = vel_i.y + ay * dt;

        new_x[i] = pos_i.x + new_vx[i] * dt;
        new_y[i] = pos_i.y + new_vy[i] * dt;
    }

    for (size_t i = 0; i < numBodies; ++i) {
        bodies[i]->setPosition(new_x[i], new_y[i]);
        bodies[i]->setVelocity(new_vx[i], new_vy[i]);
    }
}
}  // namespace NB
```

CelestialBody.cpp:

```
// Copyright 2025 Peter Meas
#include "CelestialBody.hpp"
```

```
3. #include <string>
4. #include <stdexcept>
5.
6. namespace NB {
7.
8. CelestialBody::CelestialBody()
9. : x(0), y(0), vx(0), vy(0), _mass(0.0), _imageFilename(""), universeRadius(0), windowSize(0, 0) {
10.     texture = std::make_shared<sf::Texture>();
11.     sprite.setTexture(*texture);
12. }
13.
14. CelestialBody::CelestialBody(double x, double y, double vx,
15. double vy, float mass, const std::string& imageFilename,
16. double universeRadius, const sf::Vector2u& windowSize)
17. : x(x), y(y), vx(vx), vy(vy), _mass(mass),
18. _imageFilename(imageFilename),
19. universeRadius(universeRadius), windowSize(windowSize) {
20.     // Load texture
21.     texture = std::make_shared<sf::Texture>();
22.     if (!texture->loadFromFile(_imageFilename)) {
23.         throw std::runtime_error("Failed to load image: " + _imageFilename);
24.     }
25.
26.     // Set sprite texture
27.     sprite.setTexture(*texture);
28.
29.     // Calculate scaled screen positions
30.     float scaleX = static_cast<float>(windowSize.x) / (2 * universeRadius);
31.     float scaleY = static_cast<float>(windowSize.y) / (2 * universeRadius);
32.
33.     float screenX = x * scaleX + windowSize.x / 2;
34.     float screenY = -y * scaleY + windowSize.y / 2;  // Invert Y-axis
35.
36.     // Set sprite position
37.     sprite.setPosition(screenX, screenY);
38. }
39.
40. std::string CelestialBody::imageFilename() const {
41.     return _imageFilename;
42. }
43.
44. sf::Vector2f CelestialBody::position() const {
45.     return sf::Vector2f(static_cast<float>(x), static_cast<float>(y));
46. }
47. sf::Vector2f CelestialBody::velocity() const {
48.     return sf::Vector2f(static_cast<float>(vx), static_cast<float>(vy));
```

```
49. }
50.
51. float CelestialBody::mass() const {
52.     return static_cast<float>(_mass);
53. }
54.
55. void CelestialBody::setUniverseRadius(double radius) {
56.     this->universeRadius = radius;
57.     float scaleX = static_cast<float>(windowSize.x) / (2 * universeRadius);
58.     float scaleY = static_cast<float>(windowSize.y) / (2 * universeRadius);
59.
60.     float screenX = x * scaleX + windowSize.x / 2;
61.     float screenY = -y * scaleY + windowSize.y / 2;  // Invert Y-axis
62.
63.     sprite.setPosition(screenX, screenY);
64. }
65.
66. void CelestialBody::draw(sf::RenderTarget& window, sf::RenderStates states) const {
67.     window.draw(sprite, states);
68. }
69.
70. std::istream& operator>>(std::istream& is, CelestialBody& body) {
71.     double x, y, vx, vy, mass;
72.     std::string filename;
73.
74.     if (!(is >> x >> y >> vx >> vy >> mass >> filename)) {
75.         throw std::runtime_error("Failed to read celestial body data");
76.     }
77.
78.     if (mass < 0) {
79.         throw std::runtime_error("Negative mass");
80.     }
81.
82.     // Assign values to the body
83.     body.x = x;
84.     body.y = y;
85.     body.vx = vx;
86.     body.vy = vy;
87.     body._mass = mass;
88.     body._imageFilename = filename;
89.
90.     if (body._imageFilename.empty()) {
91.         throw std::runtime_error("Image filename is missing");
92.     }
93.     if (!body.texture->loadFromFile(body._imageFilename)) {
94.         throw std::runtime_error("Failed to load image: " + body._imageFilename);
```

```
95.     }
96.     body.sprite.setTexture(*body.texture);
97.     return is;
98. }
99.
100. std::ostream& operator<<(std::ostream& os, const CelestialBody& uni) {
101.     os << uni.x << " " << uni.y << " "
102.     << uni.vx << " " << uni.vy << " "
103.     << uni._mass << " " << uni._imageFilename;
104.     return os;
105. }
106.
107. void CelestialBody::setPosition(double newX, double newY) {
108.     x = newX;
109.     y = newY;
110.
111.     // . update sprite position
112.     float scaleX = static_cast<float>(windowSize.x) / (2 * universeRadius);
113.     float scaleY = static_cast<float>(windowSize.y) / (2 * universeRadius);
114.
115.     float screenX = x * scaleX + windowSize.x / 2;
116.     float screenY = -y * scaleY + windowSize.y / 2;
117.
118.     sprite.setPosition(screenX, screenY);
119. }
120.
121. void CelestialBody::setVelocity(double newVX, double newVY) {
122.     vx = newVX;
123.     vy = newVY;
124. }
125. }  //  namespace NB
126.
```

test.cpp:

```
1. // Copyright 2025 Peter Meas
2. #include <sstream>
3. #include <fstream>
4. #include "CelestialBody.hpp"
5. #include "Universe.hpp"
6.
7. #define BOOST_TEST_DYN_LINK
8. #define BOOST_TEST_MODULE Main
9. #include <boost/test/unit_test.hpp>
10.
11. namespace sf {
```

```
12. std::ostream& operator<<(std::ostream& os, const Vector2f& vec) {
13.     os << "(" << vec.x << ", " << vec.y << ")";
14.     return os;
15. }
16. }
17.
18. NB::Universe createTestUniverse() {
19.     std::stringstream ss;
20.     ss << "2\n";  // Two bodies
21.     ss << "1.0e+11\n";  // Universe radius
22.     // Body 1: Sun-like body at center with zero velocity
23.     ss << "0.0 0.0 0.0 0.0 1.989e+30 sun.gif\n";
24.     // Body 2: Earth-like body with known position and velocity
25.     ss << "1.0e+11 0.0 0.0 3.0e+4 5.972e+24 earth.gif\n";
26.
27.     NB::Universe universe;
28.     ss >> universe;
29.     return universe;
30. }
31.
32. BOOST_AUTO_TEST_CASE(test_celestial_body_default_constructor) {
33.     NB::CelestialBody body;
34.     //  BOOST_CHECK_EQUAL(body.mass(), 0.0f);
35.     BOOST_CHECK_EQUAL(body.position(), sf::Vector2f(0, 0));
36.     BOOST_CHECK_EQUAL(body.velocity(), sf::Vector2f(0, 0));
37. }
38.
39. BOOST_AUTO_TEST_CASE(test_celestial_body_initialization) {
40.     NB::CelestialBody body;
41.     BOOST_CHECK(body.position() == sf::Vector2f(0, 0));
42.     BOOST_CHECK(body.velocity() == sf::Vector2f(0, 0));
43.     std::istringstream input("1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif");
44.     input >> body;
45.
46.     std::cout << "Body Mass: " << body.mass() << std::endl;
47.     //  check within 0.001 of expected value "boost_check_close"
48.     BOOST_CHECK_CLOSE(body.mass(), 5.9740e24f, 0.001f);
49. }
50.
51. BOOST_AUTO_TEST_CASE(test_celestial_body_stream_operator) {
52.     NB::CelestialBody body;
53.     std::istringstream input("1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif");
54.     input >> body;
55.
56.     BOOST_CHECK_CLOSE(body.mass(), 5.9740e24f, 0.001f);
57.     BOOST_CHECK_CLOSE(body.position().x, 1.4960e+11f, 0.001f);
```

```
58.     BOOST_CHECK_CLOSE(body.position().y, 0.0000, 0.0001);
59.     BOOST_CHECK_CLOSE(body.velocity().x, 0.0000, 0.001);
60.     BOOST_CHECK_CLOSE(body.velocity().y, 2.9800e+04, 0.001);
61. }
62.
63. BOOST_AUTO_TEST_CASE(test_celestial_body_output_format) {
64.     NB::CelestialBody body;
65.     std::istringstream input("1.4960e+11 0.0000 0.0000 2.9800e+04 5.9740e+24 earth.gif");
66.     input >> body;
67.
68.     std::ostringstream output;
69.     output << body;
70.
71.     std::string expected = "1.496e+11 0 0 29800 5.974e+24 earth.gif";
72.
73.     std::cout << "Expected: [" << expected << "]\n";
74.     std::cout << "Actual:   [" << output.str() << "]\n";
75.
76.     BOOST_CHECK_EQUAL(output.str(), expected);
77. }
78.
79. BOOST_AUTO_TEST_CASE(test_large_universe_radius) {
80.     std::ofstream largeRadiusFile("large_radius.txt");
81.     largeRadiusFile << "1\n";
82.     largeRadiusFile << "1.0e+20\n";
83.     largeRadiusFile
84.     << "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif\n";  // Valid body
85.     largeRadiusFile.close();
86.     NB::Universe universe("large_radius.txt");
87.     BOOST_CHECK_CLOSE(universe.radius(), 1.0e+20, 0.001);
88. }
89.
90. BOOST_AUTO_TEST_CASE(test_acceleration_computation) {
91.     NB::Universe universe = createTestUniverse();
92.
93.     const double G = 6.67e-11;
94.     double mass_sun = 1.989e+30;
95.     double distance = 1.0e+11;
96.
97.     double expected_acceleration = G * mass_sun / (distance * distance);
98.
99.     double tiny_dt = 1.0;
100.     universe.step(tiny_dt);
101.
102.     sf::Vector2f new_velocity = universe[1].velocity();
103.
```

```
104.     double actual_acceleration = -(new_velocity.x - 0) / tiny_dt;
105.
106.     BOOST_CHECK_CLOSE(actual_acceleration, expected_acceleration, 1.0);
107. }
108.
109. BOOST_AUTO_TEST_CASE(t19) {
110.     std::stringstream ss;
111.     ss << "2\n";          // Two bodies
112.     ss << "2.5e+11\n";    // Universe radius
113.     // Two equal-mass bodies close to each other
114.     ss << "0.0 0.0 0.0 0.0 1.0e+25 body1.gif\n";
115.     ss << "1.0e+10 0.0 0.0 0.0 1.0e+25 body2.gif\n";
116.
117.     // Create a uni with test bodies
118.     NB::Universe universe;
119.     ss >> universe;
120.
121.     //  initial position
122.     sf::Vector2f initial_pos0 = universe[0].position();
123.     sf::Vector2f initial_pos1 = universe[1].position();
124.
125.     // Run simm
126.     double dt = 50000.0;
127.     for (int i = 0; i < 3; i++) {
128.         universe.step(dt);
129.     }
130.
131.     sf::Vector2f final_pos0 = universe[0].position();
132.     sf::Vector2f final_pos1 = universe[1].position();
133.
134.     BOOST_CHECK_MESSAGE(final_pos0.x > initial_pos0.x,
135.                         "Body 0 should move toward Body 1 (right direction)");
136.     BOOST_CHECK_MESSAGE(final_pos1.x < initial_pos1.x,
137.         "Body 1 should move toward Body 0 (left direction)");
138. }
139.
140. /* BOOST_AUTO_TEST_CASE(test_velocity_update) {
141.     NB::Universe universe = createTestUniverse();
142.
143.     sf::Vector2f initial_velocity = universe[1].velocity();
144.
145.     double dt = 1000.0;
146.     universe.step(dt);
147.
148.     sf::Vector2f new_velocity = universe[1].velocity();
149.
```

```
150.     BOOST_CHECK(new_velocity.x < initial_velocity.x);
151.     BOOST_CHECK_CLOSE(new_velocity.y, initial_velocity.y, 0.001);
152. } */
153. /*
154. BOOST_AUTO_TEST_CASE(test_order_of_operations) {
155.     NB::Universe universe = createTestUniverse();
156.
157.     sf::Vector2f initial_earth_pos = universe[1].position();
158.     sf::Vector2f initial_sun_pos = universe[0].position();
159.
160.     double dt = 25000.0;
161.     universe.step(dt);
162.
163.     sf::Vector2f new_earth_pos = universe[1].position();
164.     sf::Vector2f new_sun_pos = universe[0].position();
165.
166.     BOOST_CHECK(new_earth_pos.x < initial_earth_pos.x);
167.     BOOST_CHECK(new_sun_pos.x > initial_sun_pos.x);
168.
169.     universe.step(dt);
170.
171.     sf::Vector2f newest_earth_pos = universe[1].position();
172.
173.     double first_displacement = initial_earth_pos.x - new_earth_pos.x;
174.     double second_displacement = new_earth_pos.x - newest_earth_pos.x;
175.
176.     BOOST_CHECK(second_displacement > first_displacement);
177. }*/
178.
179. /*BOOST_AUTO_TEST_CASE(test_energy_conservation) {
180.     NB::Universe universe = createTestUniverse();
181.
182.     const double G = 6.67e-11;
183.
184.     double mass_sun = universe[0].mass();
185.     double mass_earth = universe[1].mass();
186.
187.     sf::Vector2f sun_pos = universe[0].position();
188.     sf::Vector2f earth_pos = universe[1].position();
189.
190.     sf::Vector2f sun_vel = universe[0].velocity();
191.     sf::Vector2f earth_vel = universe[1].velocity();
192.
193.     double dx = earth_pos.x - sun_pos.x;
194.     double dy = earth_pos.y - sun_pos.y;
195.     double r = sqrt(dx*dx + dy*dy);
```

```
196.
197.     double sun_ke = 0.5 * mass_sun * (sun_vel.x*sun_vel.x + sun_vel.y*sun_vel.y);
198.     double earth_ke = 0.5 * mass_earth * (earth_vel.x*earth_vel.x + earth_vel.y*earth_vel.y);
199.     double initial_ke = sun_ke + earth_ke;
200.
201.     double initial_pe = -G * mass_sun * mass_earth / r;
202.
203.     double initial_total_energy = initial_ke + initial_pe;
204.
205.     universe.step(100000.0);
206.
207.     sun_pos = universe[0].position();
208.     earth_pos = universe[1].position();
209.     sun_vel = universe[0].velocity();
210.     earth_vel = universe[1].velocity();
211.
212.     dx = earth_pos.x - sun_pos.x;
213.     dy = earth_pos.y - sun_pos.y;
214.     r = sqrt(dx*dx + dy*dy);
215.
216.     sun_ke = 0.5 * mass_sun * (sun_vel.x*sun_vel.x + sun_vel.y*sun_vel.y);
217.     earth_ke = 0.5 * mass_earth * (earth_vel.x*earth_vel.x + earth_vel.y*earth_vel.y);
218.     double final_ke = sun_ke + earth_ke;
219.
220.     double final_pe = -G * mass_sun * mass_earth / r;
221.
222.     double final_total_energy = final_ke + final_pe;
223.
224.     BOOST_CHECK_CLOSE(final_total_energy, initial_total_energy, 5.0);
225. }
226. */
227.
```

Universe.hpp:

```
1. // Copyright 2025 Peter Meas
2. #pragma once
3. #include <iostream>
4. #include <memory>
5. #include "CelestialBody.hpp"
6. #include <SFML/Graphics.hpp>
7.
8. namespace NB {
9. class Universe : public sf::Drawable {
```

```
10.  public:
11.     explicit Universe();  // Required
12.     explicit Universe(const sf::Vector2u& windowSize);
13.     void loadFromStream(std::istream& is);
14.
15.     explicit Universe(const std::string& filename);  // Optional
16.
17.     size_t size() const;  // Optional num of bodies
18.     double radius() const;  // Optional universe radius
19.
20.     const CelestialBody& operator[](size_t i) const;  // Optional
21.
22.     void step(double dt);  // Implemented in part b, behavior for part a is undefined
23.     friend std::istream& operator>>(std::istream& is, Universe& uni);
24.     friend std::ostream& operator<<(std::ostream& os, const Universe& uni);
25.
26.  protected:
27.     void draw(sf::RenderTarget& window,
28.     sf::RenderStates states) const override;  // From sf::Drawable
29.
30.  private:
31.       size_t numBodies;  // number of bodies
32.       double universeRadius;
33.       sf::Vector2u windowSize;
34.       //  vector of shared pointers to celestial bodies
35.       std::vector<std::shared_ptr<CelestialBody>> bodies;
36.     // Fields and helper functions go here
37. };
38.
39. }  // namespace NB
40.
```

CelestialBody.hpp:

```
1. // Copyright 2025 Peter Meas
2. #pragma once
3.
4. #include <iostream>
5. #include <memory>
6. #include <SFML/Graphics.hpp>
```

```
7.
8. namespace NB {
9. class CelestialBody: public sf::Drawable {
10.  public:
11.     explicit CelestialBody();  // Required
12.     CelestialBody(double x, double y, double vx, double vy,
13.     float mass, const std::string& imageFilename, double universeRadius,
14.     const sf::Vector2u& windowSize);
15.     friend std::istream& operator>>(std::istream& is, CelestialBody& uni);
16.     friend std::ostream& operator<<(std::ostream& os, const CelestialBody& uni);
17.
18.
19.       void setPosition(double newX, double newY);
20.       void setVelocity(double newVX, double newVY);
21.       std::string imageFilename() const;
22.       void setUniverseRadius(double radius);
23.     sf::Vector2f position() const;  //  Optional
24.     sf::Vector2f velocity() const;  //  Optional
25.     float mass() const;  //  Optional
26.
27.  protected:
28.     void draw(sf::RenderTarget& window,
29.     sf::RenderStates states) const override;  // From sf::Drawable
30.
31.  private:
32.       double x, y;  // pos
33.       double vx, vy;  // velocity
34.       double _mass;  // mass
35.       std::string _imageFilename;  // filename for the image
36.       std::shared_ptr<sf::Texture> texture;  // texture for the image
37.       mutable sf::Sprite sprite;  // sprite for the image
38.       double universeRadius;
39.       sf::Vector2u windowSize;
40.     // Fields and helper methods go here
41. };
42. }  // namespace NB
43.
```

5 PS4: Sokoban Puzzle Game

5.1 Description

This project implements a Sokoban-style puzzle game using SFML. The player pushes boxes around a warehouse grid with the goal of placing all boxes onto designated storage tiles.

5.2 Features

This project implements the classic Sokoban puzzle using a grid system represented as a vector. I added support for level loading from files, player movement with both WASD and arrow keys, collision detection with walls and boxes, a win condition check using lambda expressions, and a level reset option. The game uses SFML sprites to visually distinguish tiles.

5.3 Design Decisions and Implementation

I represented the map using two vectors: one for the initial state and one for the current state to support resets. I also created a Sokoban class to manage game logic and rendering. I used SFML sprites to draw tile textures and player movement. Lambda expressions with std::count_if check for win conditions.

5.4 What I Learned

I had never created a tile-based game before, with this project, I learned how to represent 2D maps in memory and convert grid logic into screen rendering. I also gained practical experience using lambda expressions with STL algorithms and debugging grid-based logic in a visual context. This was my first exposure to game loop design.

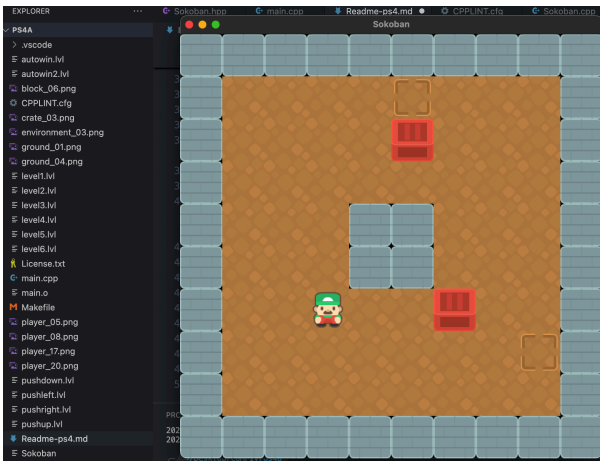
5.5 Issues

I had difficulty writing Boost unit tests that passed all the provided cases, particularly for edge cases involving box movement and collision. I also had issues mapping grid positions to pixel coordinates and ensuring all sprites aligned correctly.

5.6 Extra Credit

No extra credit was implemented for this assignment.

5.7 Output



5.8 Code

Makefile:

```
1. CXX = g++
2. CXXFLAGS = -std=c++17 -Wall -Wextra -Werror -pedantic -g
3. LDFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lboost_unit_test_framework
4.
5. all: Sokoban Sokoban.a test
6.
7. Sokoban: main.o Sokoban.o
8.     $(CXX) $^ -o $@ $(LDFLAGS) $(CXXFLAGS)
9.
10. Sokoban.a: Sokoban.o
11.     ar rcs $@ $^
12.
13. main.o: main.cpp Sokoban.hpp
14.     $(CXX) $(CXXFLAGS) -c $< -o $@
15.
16. Sokoban.o: Sokoban.cpp Sokoban.hpp
17.     $(CXX) $(CXXFLAGS) -c $< -o $@
18.
19. test: test.o Sokoban.o
20.     $(CXX) $^ -o $@ $(LDFLAGS) $(CXXFLAGS)
21.
22. test.o: test.cpp Sokoban.hpp
23.     $(CXX) $(CXXFLAGS) -c $< -o $@
24.
```

```
clean:
    rm -f *.o Sokoban Sokoban.a test
lint:
    cpplint --repository=. *.cpp *.hpp

.PHONY: all clean lint
```

main.cpp:

```
// Copyright 2025 Peter Meas
#include <iostream>
#include <fstream>
#include <string>
#include <SFML/Graphics.hpp>
#include "Sokoban.hpp"

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <level_file>" << std::endl;
        return 1;
    }

    //  Sokoban instance
    SB::Sokoban sokoban;
    // load lkevel
    std::ifstream levelFile(argv[1]);
    if (!levelFile) {
        std::cerr << "Error: Could not open file " << argv[1] << std::endl;
        return 1;
    }
    // Read the level data
    levelFile >> sokoban;
    levelFile.close();

    sf::Font font;
    if (!font.loadFromFile("arial.ttf")) {
        std::cerr << "Warning: couldnt load font" << std::endl;
    }

    sf::RenderWindow window(
```

```
32.         sf::VideoMode(sokoban.pixelWidth(), sokoban.pixelHeight()), "Sokoban");
33.
34.     while (window.isOpen()) {
35.         // Process events
36.         sf::Event event;
37.         while (window.pollEvent(event)) {
38.             if (event.type == sf::Event::Closed) {
39.                 window.close();
40.             } else if (event.type == sf::Event::KeyPressed) {
41.                 if (sokoban.isWon() && event.key.code != sf::Keyboard::R &&
42.                 event.key.code != sf::Keyboard::Escape) {
43.                     continue;
44.                 }
45.                 switch (event.key.code) {
46.                     case sf::Keyboard::W:
47.                     case sf::Keyboard::Up:
48.                         sokoban.movePlayer(SB::Direction::Up);
49.                         break;
50.                     case sf::Keyboard::S:
51.                     case sf::Keyboard::Down:
52.                         sokoban.movePlayer(SB::Direction::Down);
53.                         break;
54.                     case sf::Keyboard::A:
55.                     case sf::Keyboard::Left:
56.                         sokoban.movePlayer(SB::Direction::Left);
57.                         break;
58.                     case sf::Keyboard::D:
59.                     case sf::Keyboard::Right:
60.                         sokoban.movePlayer(SB::Direction::Right);
61.                         break;
62.                     case sf::Keyboard::R:
63.                         sokoban.reset();
64.                         break;
65.                     case sf::Keyboard::Escape:
66.                         window.close();
67.                         break;
68.                     default:
69.                         break;
70.                 }
71.             }
72.         }
```

```
73.         window.clear();
74.         window.draw(sokoban);
75.         window.display();
76.     }
77.     return 0;
78. }
79.
```

Sokoban.cpp:

```
1. // Copyright 2025 Peter Meas
2. #include "Sokoban.hpp"
3. #include <fstream>
4. #include <sstream>
5. #include <vector>
6. #include <string>
7. #include <map>
8. #include <algorithm>
9.
10. namespace SB {
11.
12. Sokoban::Sokoban() {
13. }
14.
15. Sokoban::Sokoban(const std::string& filename) {
16.     std::ifstream infile(filename);
17.     if (infile) {
18.         infile >> *this;
19.     }
20. }
21.
22. unsigned int Sokoban::pixelHeight() const {
23.     return height() * TILE_SIZE;
24. }
25.
26. unsigned int Sokoban::pixelWidth() const {
27.     return width() * TILE_SIZE;
28. }
29.
```

```
30. unsigned int Sokoban::height() const {
31.     return m_height;
32. }
33.
34. unsigned int Sokoban::width() const {
35.     return m_width;
36. }
37.
38. sf::Vector2u Sokoban::playerLoc() const {
39.     return m_playerLoc;
40. }
41.
42. bool Sokoban::isWon() const {
43.     // part a d o nothing, part b movement logic
44.
45.     auto BoxNotOnStorage = [](char c) { return c == 'A'; };
46.     int boxesNotOnStorage = std::count_if(m_grid.begin(), m_grid.end(), BoxNotOnStorage);
47.
48.     auto StorageWithoutBox = [](char c) { return c == 'a'; };
49.     int emptyCount = std::count_if(m_grid.begin(), m_grid.end(), StorageWithoutBox);
50.
51.     auto BoxOnStorage = [](char c) { return c == '1'; };
52.     int boxesOnStorage = std::count_if(m_grid.begin(), m_grid.end(), BoxOnStorage);
53.
54.     int totalStor = emptyCount + boxesOnStorage;
55.     int totalBox = boxesOnStorage + boxesNotOnStorage;
56.
57.     if (totalBox > totalStor) {
58.         return emptyCount == 0;
59.     } else {
60.         return boxesNotOnStorage == 0;
61.     }
62. }
63.
64. void Sokoban::movePlayer(Direction dir) {
65.     // part a do nothing, part b movnig lopgic
66.     sf::Vector2u newLocation = m_playerLoc;
67.
68.     switch (dir) {
69.         case Direction::Up:
70.             if (newLocation.y > 0) newLocation.y--;
```

```
71.             break;
72.         case Direction::Down:
73.             if (newLocation.y < m_height - 1) newLocation.y++;
74.             break;
75.         case Direction::Left:
76.             if (newLocation.x > 0) newLocation.x--;
77.             break;
78.         case Direction::Right:
79.             if (newLocation.x < m_width - 1) newLocation.x++;
80.             break;
81.     }
82.     unsigned int index = newLocation.y * m_width + newLocation.x;
83.     char& tarCell = m_grid[index];
84.
85.     if (tarCell == '.' || tarCell == 'a') {  //  case where target cell is empty
86.         m_playerLoc = newLocation;
87.     } else if (tarCell == '#') {  // target cell is a wall
88.         // Cant move, wall
89.         return;
90.     } else if (tarCell == 'A' || tarCell == '1') {
91.         // box or box on storage, lets try to push it
92.         // box pushing logic
93.
94.         sf::Vector2u boxNewPos = newLocation;
95.
96.         switch (dir) {
97.             case Direction::Up:
98.             if (boxNewPos.y > 0) boxNewPos.y--;
99.             break;
100.             case Direction::Down:
101.             if (boxNewPos.y < m_height - 1) boxNewPos.y++;
102.             break;
103.             case Direction::Left:
104.             if (boxNewPos.x > 0) boxNewPos.x--;
105.             break;
106.             case Direction::Right:
107.             if (boxNewPos.x < m_width - 1) boxNewPos.x++;
108.             break;
109.         }
110.         char& boxTarCell = m_grid[boxNewPos.y *
111.         m_width + boxNewPos.x];  // . get character at box new pos
```

```
112.         if (boxTarCell == '.' || boxTarCell == 'a') {
113.             // . box can be pushed
114.             if (boxTarCell == '.') {
115.                 boxTarCell = 'A';
116.             } else {
117.                 boxTarCell = '1';
118.             }
119.             if (tarCell == 'A') {
120.                 tarCell = '.';
121.             } else {
122.                 tarCell = 'a';
123.             }
124.             m_playerLoc = newLocation;
125.         }
126.     }
127. }
128.
129. void Sokoban::reset() {
130.     // reset game
131.     m_grid = m_initialGrid;
132.     m_playerLoc = m_initialPlayerLoc;
133. }
134.
135. void Sokoban::undo() {
136.     // Optional
137. }
138.
139. void Sokoban::redo() {
140.     // Optional
141. }
142.
143. void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
144.     static sf::Texture wallTexture, floorTexture, boxTexture, storageTexture,
145.     boxOnStorageTexture, playerTexture;
146.     static bool texturesLoaded = false;
147.     if (!texturesLoaded) {
148.         wallTexture.loadFromFile("block_06.png");
149.         floorTexture.loadFromFile("ground_01.png");
150.         boxTexture.loadFromFile("crate_03.png");
151.         storageTexture.loadFromFile("ground_04.png");
152.         playerTexture.loadFromFile("player_05.png");
```

```
153.         texturesLoaded = true;
154.     }
155.
156.     // Create a sprite for drawing
157.     sf::Sprite sprite;
158.     // Draw the grid
159.     for (unsigned int y = 0; y < height(); ++y) {
160.         for (unsigned int x = 0; x < width(); ++x) {
161.             sprite.setPosition(x * TILE_SIZE, y * TILE_SIZE);
162.             switch (m_grid[y * width() + x]) {
163.                 case '#':  // a Wall
164.                     sprite.setTexture(wallTexture);
165.                     break;
166.                 case '.':  // empty  space
167.                     sprite.setTexture(floorTexture);
168.                     break;
169.                 case 'A':  // box
170.                     sprite.setTexture(floorTexture);
171.                     target.draw(sprite, states);
172.                     sprite.setTexture(boxTexture);
173.                     break;
174.                 case 'a':  // Storage location
175.                     sprite.setTexture(storageTexture);
176.                     break;
177.                 case '1':  // Box on storage
178.                     sprite.setTexture(storageTexture);
179.                     target.draw(sprite, states);
180.                     sprite.setTexture(boxTexture);
181.                     break;
182.                 default:
183.                     sprite.setTexture(floorTexture);
184.                     break;
185.             }
186.             // Draw  sprite
187.             target.draw(sprite, states);
188.         }
189.     }
190.     // draw the player separately (so it's always on top)
191.     sprite.setTexture(playerTexture);
192.     sprite.setPosition(m_playerLoc.x * TILE_SIZE, m_playerLoc.y * TILE_SIZE);
193.     target.draw(sprite, states);
```

```
194.
195.     if (isWon()) {
196.     sf::Font font;
197.     font.loadFromFile("arial.ttf");
198.
199.     sf::Text text("YOU WONONN!NMN!N!N!N1", font, 48);
200.     text.setFillColor(sf::Color::Green);
201.     text.setStyle(sf::Text::Bold);
202.
203.     sf::FloatRect textRect = text.getLocalBounds();
204.     text.setOrigin(textRect.width /2 , textRect.height / 2);
205.     text.setPosition(pixelWidth()/2, pixelHeight()/2);
206.
207.     target.draw(text, states);
208.
209.     } else {
210.         sprite.setTexture(playerTexture);
211.         sprite.setPosition(m_playerLoc.x * TILE_SIZE, m_playerLoc.y * TILE_SIZE);
212.         target.draw(sprite, states);
213.     }
214. }
215.
216. std::ostream& operator<<(std::ostream& out, const Sokoban& s) {
217.     out << s.height() << " " << s.width() << std::endl;
218.
219.     for (unsigned int y = 0; y < s.height(); ++y) {
220.         for (unsigned int x = 0; x < s.width(); ++x) {
221.             // Check if this is the player's position
222.             if (x == s.playerLoc().x && y == s.playerLoc().y) {
223.                 out << '@';
224.             } else {
225.                 out << s.m_grid[y * s.width() + x];
226.             }
227.         }
228.         out << std::endl;
229.     }
230.
231.     return out;
232. }
233. std::istream& operator>>(std::istream& in, Sokoban& s) {
234.     in >> s.m_height >> s.m_width;
```

```
235.     std::string line;
236.     std::getline(in, line);
237.     s.m_grid.resize(s.m_height * s.m_width);
238.     s.m_initialGrid.resize(s.m_height * s.m_width);
239.     for (unsigned int y = 0; y < s.m_height; ++y) {
240.         std::getline(in, line);
241.         for (unsigned int x = 0; x < s.m_width && x < line.length(); ++x) {
242.             char cell = line[x];
243.             // handle the player position separately
244.             if (cell == '@') {
245.                 s.m_playerLoc = sf::Vector2u(x, y);
246.                 s.m_initialPlayerLoc = sf::Vector2u(x, y);
247.                 cell = '.';  // player standing on an empty space
248.             }
249.
250.             s.m_grid[y * s.m_width + x] = cell;
251.             s.m_initialGrid[y * s.m_width + x] = cell;
252.         }
253.     }
254.     return in;
255. }
256.
257. }  // namespace SB
258.
```

test.cpp:

```
1. // Copyright 2025 Peter Meas
2. #define BOOST_TEST_MODULE SokobanTests
3. #include <fstream>
4. #include <sstream>
5. #include "Sokoban.hpp"
6. #include <boost/test/included/unit_test.hpp>
7.
8.
9. void createTestLevel(const std::string& filename, const std::string& content) {
10.     std::ofstream file(filename);
11.     file << content;
12.     file.close();
13. }
```

```
14. /*
15. BOOST_AUTO_TEST_CASE(TestBasicMovement) {
16.     createTestLevel("test_movement.lvl",
17.                     "5 5\n"
18.                     "#####\n"
19.                     "#...#\n"
20.                     "#.@.#\n"
21.                     "#...#\n"
22.                     "#####\n");
23.     SB::Sokoban sokoban("test_movement.lvl");
24.     //  initial position
25.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
26.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 2u);
27.     // Create a new instance for each movement
28.     // Test Up movement
29.     SB::Sokoban sokobanUp("test_movement.lvl");
30.     sokobanUp.movePlayer(SB::Direction::Up);
31.     BOOST_CHECK_EQUAL(sokobanUp.playerLoc().x, 2u);
32.     BOOST_CHECK_EQUAL(sokobanUp.playerLoc().y, 2u);
33.     // test right movement
34.     SB::Sokoban sokobanRight("test_movement.lvl");
35.     sokobanRight.movePlayer(SB::Direction::Right);
36.     BOOST_CHECK_EQUAL(sokobanRight.playerLoc().x, 2u);
37.     BOOST_CHECK_EQUAL(sokobanRight.playerLoc().y, 2u);
38.     // Test Down movement
39.     SB::Sokoban sokobanDown("test_movement.lvl");
40.     sokobanDown.movePlayer(SB::Direction::Down);
41.     BOOST_CHECK_EQUAL(sokobanDown.playerLoc().x, 2u);
42.     BOOST_CHECK_EQUAL(sokobanDown.playerLoc().y, 2u);
43.     // Test Left movement
44.     SB::Sokoban sokobanLeft("test_movement.lvl");
45.     sokobanLeft.movePlayer(SB::Direction::Left);
46.     BOOST_CHECK_EQUAL(sokobanLeft.playerLoc().x, 2u);
47.     BOOST_CHECK_EQUAL(sokobanLeft.playerLoc().y, 2u);
48. }*/
49.
50.
51. BOOST_AUTO_TEST_CASE(TestOutOfScreen) {
52.     createTestLevel("test_out_of_screen.lvl",
53.                     "5 5\n"
54.                     "#####\n"
```

```
                    "#.@.#\n"
                    "#...#\n"
                    "#...#\n"
                    "#####\n");

    SB::Sokoban sokoban("test_out_of_screen.lvl");
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
    BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 1u);

    //  check to make sure player doesnt escape bounds of window
}
BOOST_AUTO_TEST_CASE(TestWallCollision) {
    createTestLevel("test_wall.lvl",
                    "5 5\n"
                    "#####\n"
                    "#.@.#\n"
                    "#...#\n"
                    "#...#\n"
                    "#####\n");

    SB::Sokoban sokoban("test_wall.lvl");

    // Test initial position
    BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
    BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 1u);

    // Test wall collision (should not move)
    sokoban.movePlayer(SB::Direction::Up);
    BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
    BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 1u);

    sokoban.movePlayer(SB::Direction::Down);
    sokoban.movePlayer(SB::Direction::Right);
    sokoban.movePlayer(SB::Direction::Right);
    BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);  // a2
    BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 1u);  // 1
}

```

```
96. BOOST_AUTO_TEST_CASE(TestBoxBoundaryPushing) {
97.     createTestLevel("test_box_boundary.lvl",
98.                     "5 5\n"
99.                     "..@A.\n"
100.                     ".....\n"
101.                     ".....\n"
102.                     ".....\n"
103.                     ".....\n");
104.
105.     SB::Sokoban sokoban("test_box_boundary.lvl");
106.
107.     // Test initial position
108.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
109.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 0u);
110.
111.     // Try to push box off screen
112.     sokoban.movePlayer(SB::Direction::Right);
113.     // Box should stop at the boundary, player should move up to the box
114.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
115.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 0u);
116.
117.     // Try to push box off screen again (should not move)
118.     sokoban.movePlayer(SB::Direction::Right);
119.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
120.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 0u);
121. }
122.
123. BOOST_AUTO_TEST_CASE(TestBoxPushing) {
124.     createTestLevel("test_box.lvl",
125.                     "5 5\n"
126.                     "#####\n"
127.                     "#...#\n"
128.                     "#.@A#\n"
129.                     "#...#\n"
130.                     "#####\n");
131.
132.     SB::Sokoban sokoban("test_box.lvl");
133.
134.     // Test push box to the right (should move)
135.     sokoban.movePlayer(SB::Direction::Right);
136.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
```

```
137.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 2u);
138.     // Reset and test push box against wall (should not move)
139.     sokoban.reset();
140.     sokoban.movePlayer(SB::Direction::Right);
141.     sokoban.movePlayer(SB::Direction::Right);
142.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
143.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 2u);
144. }
145. BOOST_AUTO_TEST_CASE(TestWinCondition) {
146.     createTestLevel("test_win.lvl",
147.                     "5 5\n"
148.                     "#####\n"
149.                     "#...#\n"
150.                     "#.@A#\n"
151.                     "#..a#\n"
152.                     "#####\n");
153.     SB::Sokoban sokoban("test_win.lvl");
154.     // Initially not won
155.     BOOST_CHECK_EQUAL(sokoban.isWon(), false);
156.     // Push box to storage to win
157.     sokoban.movePlayer(SB::Direction::Up);
158.     sokoban.movePlayer(SB::Direction::Right);
159.     sokoban.movePlayer(SB::Direction::Down);
160.     sokoban.movePlayer(SB::Direction::Down);
161.     // should be won
162.     BOOST_CHECK_EQUAL(sokoban.isWon(), true);
163. }
164.
165. BOOST_AUTO_TEST_CASE(TestReset) {
166.     createTestLevel("test_reset.lvl",
167.                     "5 5\n"
168.                     "#####\n"
169.                     "#a..#\n"
170.                     "#.@A#\n"
171.                     "#...#\n"
172.                     "#####\n");
173.     SB::Sokoban sokoban("test_reset.lvl");
174.     // Record initial position
175.     sf::Vector2u initialPos = sokoban.playerLoc();
176.     // Move player
177.     sokoban.movePlayer(SB::Direction::Up);
```

```
178.     BOOST_CHECK(sokoban.playerLoc() != initialPos);
179.     sokoban.reset();
180.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, initialPos.x);
181.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, initialPos.y);
182. }
183.
184. BOOST_AUTO_TEST_CASE(TestManyTargets) {
185.     createTestLevel("test_many_targets.lvl",
186.                     "7 7\n"
187.                     "#######\n"
188.                     "#.....#\n"
189.                     "#.aaaa#\n"
190.                     "#.@A..#\n"
191.                     "#.....#\n"
192.                     "#.....#\n"
193.                     "#######\n");
194.     SB::Sokoban sokoban("test_many_targets.lvl");
195.     BOOST_CHECK_EQUAL(sokoban.isWon(), false);
196.
197.     // Push box to a target
198.     sokoban.movePlayer(SB::Direction::Down);
199.     sokoban.movePlayer(SB::Direction::Right);
200.     sokoban.movePlayer(SB::Direction::Up);
201.     BOOST_CHECK_EQUAL(sokoban.isWon(), true);
202. }
203.
204. BOOST_AUTO_TEST_CASE(TestSymbolParse) {
205.     createTestLevel("test_symbols.lvl",
206.                     "7 7\n"
207.                     "#######\n"
208.                     "#.@...#\n"
209.                     "#.A...#\n"
210.                     "#.a...#\n"
211.                     "#.B...#\n"  // Ensure capital letters for boxes are parsed
212.                     "#.b...#\n"  // Ensure lowercase letters for targets are parsed
213.                     "#######\n");
214.     SB::Sokoban sokoban("test_symbols.lvl");
215.     // Check if player position is correct
216.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
217.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 1u);
218.     // Try to move to check if parsing worked properly
```

```
219.     sokoban.movePlayer(SB::Direction::Down);
220.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
221.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 2u);
222.     // Try to push a box
223.     sokoban.movePlayer(SB::Direction::Down);
224.     BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 2u);
225.     BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 2u);
226. }
227.
```

Sokoban.hpp:

```
1. // Copyright 2025 Peter Meas
2. #pragma once
3.
4. #include <iostream>
5.
6. #include <SFML/Graphics.hpp>
7.
8. namespace SB {
9. enum class Direction {
10.     Up, Down, Left, Right
11. };
12.
13. class Sokoban : public sf::Drawable {
14.  public:
15. friend std::ostream& operator<<(std::ostream& out, const Sokoban& s);
16. friend std::istream& operator>>(std::istream& in, Sokoban& s);
17.     static const int TILE_SIZE = 64;
18.
19.     Sokoban();
20.     explicit Sokoban(const std::string&);  // Optional
21.
22.     unsigned int pixelHeight() const;  // Optional
23.     unsigned int pixelWidth() const;  // Optional
24.
25.     unsigned int height() const;
26.     unsigned int width() const;
27.
28.     sf::Vector2u playerLoc() const;
```

```
29.
30.     bool isWon() const;
31.
32.     void movePlayer(Direction dir);
33.     void reset();
34.
35.     void undo();  // Optional XC
36.     void redo();  // Optional XC
37.
38.  protected:
39.     void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
40.
41.  private:
42.     unsigned int m_width = 0;
43.     unsigned int m_height = 0;
44.     sf::Vector2u m_playerLoc;
45.     sf::Vector2u m_initialPlayerLoc;
46.     std::vector<char> m_grid;
47.     std::vector<char> m_initialGrid;
48. };
49.
50. }  // namespace SB
51.
```

6 PS5: DNA Alignment

6.1 Description

Through the implementation of the Needleman-Wunsch dynamic programming algorithm, me and my partner were able to find the most optimal alignment between two DNA sequences. The alignment minimizes the total penalty based on mismatches, insertions, and deletions.

6.2 Features

This project uses the Needleman-Wunsch algorithm to compute optimal alignments between two DNA sequences. I implemented a 2D dynamic programming matrix to store cost values and created helper functions for penalties and min-cost paths. I also included Boost unit tests to validate correctness, including edge cases and empty string alignment.

6.3 Design Decisions and Implementation

We used std::vector<std::vector<int>> to dynamically allocate the matrix based on string length. We chose to traverse from the top-left corner of the matrix toward the bottom-right, storing scores and backtracking directions. I also created comprehensive unit tests to check alignment correctness and edge cases such as empty strings and symmetric input.

6.4 What I Learned

I gained a deeper understanding of dynamic programming and its applications in bioinformatics. Before this, I had seen 1D DP problems but never built a full 2D alignment system. I learned how to backtrack alignment paths and validate results through custom test cases. I also gained insight into runtime and memory scaling as input sizes increased.

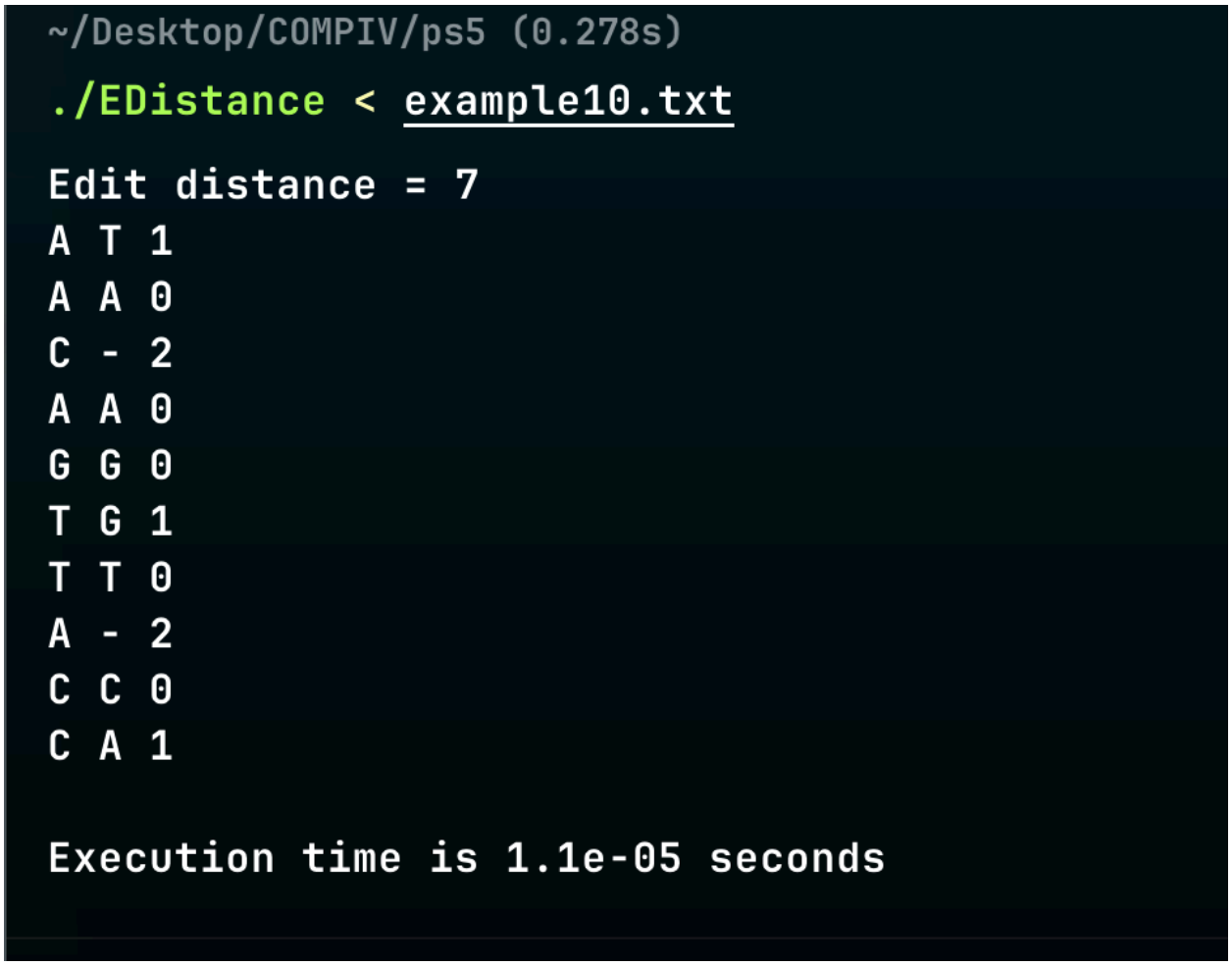
6.5 Issues

Initially, I was confused about whether to start from the top-left or bottom-right corner when tracing the alignment. I resolved this after re-reading the algorithm specification and analyzing sample outputs. I also faced performance issues with large input files.

6.6 Extra Credit

I optimized the min3() function using if statements instead of nested std::min() calls. I also tested the program with various optimization flags (-O0, -O1, -O2, -O3) and recorded performance improvements in runtime.

6.7 Output



6.8 Code

Makefile:

```
1. CXX = g++
2. CXXFLAGS = -std=c++17 -Wall -Wextra -Werror -pedantic -g
3. LDFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lboost_unit_test_framework
4.
5. all: EDistance
6.
7. EDistance: main.o EDistance.o
8.     $(CXX) $^ -o $@ $(LDFLAGS)
9.
10. EDistance.a: EDistance.o
11.     ar rcs $@ $^
12.
13. main.o: main.cpp EDistance.hpp
14.     $(CXX) $(CXXFLAGS) -c $< -o $@
15.
16. EDistance.o: EDistance.cpp EDistance.hpp
```

```
17.     $(CXX) $(CXXFLAGS) -c $< -o $@
18.
19. clean:
20.     rm -f *.o EDistance EDistance.a test
21.
22. lint:
23.     cpplint --repository=. *.cpp *.hpp
24.
25. .PHONY: all clean lint
```

main.cpp:

```
1. // Copyright 2025 Mohamed Bouchtout and Peter Meas
2.
3. #include <iostream>
4. #include <string>
5. #include <fstream>
6. #include <SFML/System.hpp>
7. #include "EDistance.hpp"
8.
9. int main() {
10.     std::string s1;
11.     std::string s2;
12.
13.     if (!std::getline(std::cin, s1)) {
14.         std::cerr << "Error: Failed to read first line!\n";
15.         exit(1);
16.     }
17.
18.     if (!std::getline(std::cin, s2)) {
19.         std::cerr << "Error: Failed to read second line!\n";
20.         exit(1);
21.     }
22.
23.     sf::Clock clock;
24.
25.     EDistance ed(s1, s2);
26.     ed.optDistance();
27.
28.     sf::Time t = clock.getElapsedTime();
29.     std::cout << "Edit distance = " << ed.optDistance() << std::endl;
30.     std::cout << ed.alignment() << std::endl;
31.     std::cout << "Execution time is " << t.asSeconds()
32.                           << " seconds" << std::endl;
33.
34.     return 0;
35. }
```

EDistance.cpp:

```
1. // Copyright 2025 Mohamed Bouchtout and Peter Meas
2.
```

```
3. #include <iostream>
4. #include <vector>
5. #include <string>
6. #include <algorithm>
7. #include "EDistance.hpp"
8.
9. EDistance::EDistance(const std::string& s1, const std::string& s2) {
10.     _s1 = s1;
11.     _s2 = s2;
12.
13.     _m = _s1.length();
14.     _n = _s2.length();
15.
16.     //  allocater matrix
17.     _matrix.resize(_m + 1, std::vector<int>((_n) + 1, 0));
18. }
19.
20. int EDistance::penalty(char a, char b) {
21.     return a == b ? 0 : 1;
22. }
23.
24. int EDistance::min3(int a, int b, int c) {
25.     if (a <= b && a <= c) {
26.         return a;
27.     } else if (b <= a && b <= c) {
28.         return b;
29.     } else {
30.         return c;
31.     }
32. }
33.
34. int EDistance::optDistance() {
35.     for (int j = 0; j <= _n; j++) {
36.         _matrix[_m][j] = 2 * (_n - j);
37.     }
38.
39.     for (int i =0; i <= _m; i++) {
40.         _matrix[i][_n] = 2 * (_m - i);
41.     }
42.
43.     for (int i = _m - 1; i >= 0; i--) {
44.         for (int j = _n - 1; j >= 0; j--) {
45.             int match = _matrix[i+1][j+1] + penalty(_s1[i], _s2[j]);
46.
47.             int del = _matrix[i+1][j] + 2;
48.             int insert = _matrix[i][j+1] + 2;
49.
50.             _matrix[i][j] = min3(match, del, insert);
51.         }
52.     }
53.
54.     return _matrix[0][0];
55. }
```

```
56.
57. std::string EDistance::alignment() {
58.     std::string output;
59.     //  std::string s1;
60.     //  std::string s2;
61.     int i = 0;
62.     int j = 0;
63.
64.     while (i < _m && j < _n) {  // loop through strings, start at [0][0]
65.         // 3 cases to handle
66.         //  match or mismatch (diagnoal )
67.         if (_matrix[i][j] == _matrix[i+1][j+1] + penalty(_s1[i], _s2[j])) {
68.             output = output + _s1[i];
69.             output = output + " ";
70.             output = output + _s2[j];
71.             output = output + " ";
72.             output = output + std::to_string(penalty(_s1[i], _s2[j]));
73.             output = output + "\n";
74.             i++;
75.             j++;
76.             //  format :
77.             //  string1  string2  cost
78.             //  A        A       0
79.         } else if (_matrix[i][j] == _matrix[i+1][j] + 2) {
80.             // gap s2 (move down)
81.             //  align char in s1 with a gap
82.             output = output + _s1[i];
83.             output = output + " ";
84.             output = output + "-";
85.             output += " ";
86.             output += "2";
87.             output += "\n";
88.             i++;
89.         } else {  //  gap in s1
90.             output += "-";
91.             output += " ";
92.             output += _s2[j];
93.             output += " ";
94.             output += "2";
95.             output += "\n";
96.             j++;
97.         }
98.     }
99.
100.     while (i < _m) {
101.         output += _s1[i];
102.         output += " ";
103.         output += "-";
104.         output += " ";
105.         output += "2";
106.         output += "\n";
107.         i++;
108.     }
```

```
109.
110.     while (j < _n) {
111.         output += "-";
112.         output += " ";
113.         output += _s2[j];
114.         output += "2";
115.         output += "\n";
116.         j++;
117.     }
118.
119.     return output;
120. }
121.
```

test.cpp:

```
1. // Copyright 2025 Mohamed Bouchtout
2.
3. #define BOOST_TEST_DYN_LINK
4. #define BOOST_TEST_MODULE Main
5.
6. #include <iostream>
7. #include <string>
8. #include <boost/test/unit_test.hpp>
9. #include "EDistance.hpp"
10.
11. BOOST_AUTO_TEST_CASE(min3Test) {
12.     EDistance ed("", "");
13.     BOOST_CHECK_EQUAL(EDistance::min3(1, 2, 3), 1);
14.     BOOST_CHECK_EQUAL(EDistance::min3(3, 2, 1), 1);
15.     BOOST_CHECK_EQUAL(EDistance::min3(2, 1, 3), 1);
16.     BOOST_CHECK_EQUAL(EDistance::min3(5, 5, 5), 5);
17.     BOOST_CHECK_EQUAL(EDistance::min3(0, 2, 3), 0);
18. }
19.
20. BOOST_AUTO_TEST_CASE(costTest) {
21.     BOOST_CHECK_EQUAL(EDistance::penalty('a', 'a'), 0);
22.     BOOST_CHECK_EQUAL(EDistance::penalty('a', 'b'), 1);
23.     BOOST_CHECK_EQUAL(EDistance::penalty('A', 'a'), 1);
24. }
25.
26. BOOST_AUTO_TEST_CASE(optDistanceTest) {
27.     std::string s1 = "AACAGTTACC";
28.     std::string s2 = "TAAGGTCA";
29.     EDistance ed(s1, s2);
30.     BOOST_CHECK_EQUAL(ed.optDistance(), 7);
31. }
32.
33. BOOST_AUTO_TEST_CASE(alignmentOutputTest) {
34.     std::string s1 = "AACAGTTACC";
35.     std::string s2 = "TAAGGTCA";
36.     EDistance ed(s1, s2);
37.     ed.optDistance();
```

```
38.     std::string align = ed.alignment();
39.     std::string expected =
40.         "A T 1\n"
41.         "A A 0\n"
42.         "C - 2\n"
43.         "A A 0\n"
44.         "G G 0\n"
45.         "T G 1\n"
46.         "T T 0\n"
47.         "A - 2\n"
48.         "C C 0\n"
49.         "C A 1\n";
50.     BOOST_CHECK_EQUAL(align, expected);
51. }
52.
53. BOOST_AUTO_TEST_CASE(emptyStringsTest) {
54.     EDistance ed1("", ""), ed2("a", ""), ed3("", "b");
55.     BOOST_CHECK_EQUAL(ed1.optDistance(), 0);
56.     BOOST_CHECK_EQUAL(ed2.optDistance(), 2);
57.     BOOST_CHECK_EQUAL(ed3.optDistance(), 2);
58. }
59.
60. BOOST_AUTO_TEST_CASE(reversedStringsTest) {
61.     EDistance ed1("abc", "def"), ed2("def", "abc");
62.     BOOST_CHECK(ed1.optDistance() == ed2.optDistance());
63. }
64.
```

EDistance.hpp:

```
1. // Copyright 2025 Mohamed Bouchtout and Peter Meas
2.
3. #pragma once
4.
5. #include <string>
6. #include <vector>
7.
8. class EDistance {
9.  public:
10.     EDistance(const std::string& s1, const std::string& s2);
11.
12.     static int penalty(char a, char b);
13.     static int min3(int a, int b, int c);
14.
15.     int optDistance();
16.     std::string alignment();
17.  private:
18.     std::string _s1;
19.     std::string _s2;
20.     int _m;
21.     int _n;
22.     std::vector<std::vector<int>> _matrix;
23. };
```

7 PS6: RandWriter

7.1 Description

This project creates a random text generator based on a Markov model of order `k`. It learns the frequency of k-length substrings (k-grams) and the characters that follow them in the source text, then generates pseudo-random output with similar character structure.

7.2 Features

This project creates a Markov model from input text and generates pseudo-random output that mimics the original style. I used std::map to track frequency of characters following k-grams and std::discrete_distribution to generate weighted random characters. The RandWriter class exposes methods like freq(), kRand(), and generate() for analysis and output.

7.3 Design Decisions and Implementation

To ensure realistic character distribution, I used STL maps to count frequency and built a weighted model. I added robust error checking to validate input, including k-gram length and seed validity. A lambda function is used in the operator<< implementation to format alphabet display.

7.4 What I Learned

I learned how to model text statistically and how to use modern C++ features like the random number utilities and map structures. Before this, I had not built a probabilistic model or used weighted randomness in C++. I also improved my ability to write reusable classes and test them thoroughly.

7.5 Issues

Initially, I forgot to store the input text and value of k in the class constructor, which caused methods like generate() to fail. I also had trouble handling invalid seeds and checking for exception handling.

7.6 Extra Credit

I implemented no extra credit.

7.7 Output

Below is a string of randomly generated text, provided by Romeo. The kgram length was set to 1 and 15 characters were generated.

AR t theanouesu

Below is another string of randomly generated text, this time, provided by Tom Sawyer. The k gram length here was set to 3, and 400 characters were generated.

THE AUTHOR.
He knew me, but vague, hand would burn full.
Tom was tonish Mary be can up us find for? If they to seen couple withe had a were the bad. But in fromissary, long deper would cont press intill." "I rushed him and dain a som als Tom could by haveyards, werest cour of a blubbon. But with ther retches on thered her wood clate one you teartly have beyond got tempty shuddent to his chased th

7.8 Code

Makefile:

```
1. CXX = g++
2. CXXFLAGS = -std=c++17 -Wall -Werror -Wextra -pedantic -g
3. LDFLAGS = -lsfml-graphics -lsfml-window -lsfml-system -lboost_unit_test_framework
4.
5. all: TextWriter TextWriter.a test
6.
7. TextWriter: TextWriter.cpp RandWriter.o
8.     $(CXX) $(CXXFLAGS) -o TextWriter TextWriter.cpp RandWriter.o
9.
10. TextWriter.a: RandWriter.o
11.    ar rcs TextWriter.a RandWriter.o
12.
13. RandWriter.o: RandWriter.cpp RandWriter.hpp
14.    $(CXX) $(CXXFLAGS) -c RandWriter.cpp
15.
16. test: test.cpp RandWriter.o
17.    $(CXX) $(CXXFLAGS) -o test test.cpp RandWriter.o $(LDFLAGS)
18.
19. lint:
20.    cpplint --repository=. *.cpp *.hpp
21.
22. clean:
23.    rm -f *.o TextWriter TextWriter.a test
24.
25. .PHONY: all lint clean
```

RandWriter.cpp:

```cpp
// Copyright 2025 Peter
#include "RandWriter.hpp"
#include <algorithm>

std::ostream& operator<<(std::ostream& os, const RandWriter& writer) {
    // Print order k
    os << "Order: " << writer.k << std::endl;
    // Print alphabet
    os << "Alphabet: ";
     std::for_each(writer.alphabet.begin(), writer.alphabet.end(), [&](char c) {
         if (c == '\n') os << "\\n";
         else if (c == '\t') os << "\\t";
         else if (c == ' ') os << "␣";
         else os << c;
    });

    os << std::endl;

    // Print k-gram frequencies
    os << "k-gram frequencies:" << std::endl;
    for (const auto& pair : writer.kgramFreq) {
         os << "  \"" << pair.first << "\": " << pair.second << std::endl;
    }

    // Print k+1-gram frequencies
    os << "k+1-gram frequencies:" << std::endl;
    for (const auto& pair : writer.kPlusOneGramFreq) {
         os << "  \"" << pair.first << "\": " << pair.second << std::endl;
    }

    return os;
}

RandWriter::RandWriter(const std::string& str, size_t k) {
    std::random_device rndm;  // use for seed gen
    rng = std::mt19937(rndm());
    this->text = str;
    this->k = k;
    if (str.length() < k) {
         throw std::invalid_argument("Text length must be at least k");
    }
```

```
42.
43.     for (size_t i = 0; i < str.length(); i++) {
44.         // extract kgram starting a pos i
45.         std::string kgram;
46.         for (size_t j = 0; j < k; j++) {
47.             // get pos for the char at posiiton (i+j) % str.length , to handle wraparound
48.             kgram += str[(i+j) % str.length()];
49.             // add char to kgram
50.         }
51.         char newChar = str[(i+k) % str.length()];
52.         kgramFreq[kgram]++;
53.         kPlusOneGramFreq[kgram + newChar]++;
54.
55.         if (alphabet.find(newChar) == std::string::npos) {
56.             alphabet += newChar;
57.         }
58.     }
59. }
60.
61. size_t RandWriter::orderK() const {
62.     return k;
63. }
64.
65. int RandWriter::freq(const std::string& kgram) const {
66.     if (kgram.length() != k) {
67.         throw std::invalid_argument("kgram length must equal k");
68.     }
69.
70.     auto it = kgramFreq.find(kgram);
71.     if (it != kgramFreq.end()) {
72.         return it->second;  // return INT freq count for THIS kgram
73.     } else {
74.         return 0;
75.     }
76. }
77.
78. int RandWriter::freq(const std::string& kgram, char c) const {
79.     if (kgram.length() != k) {
80.         throw std::invalid_argument("kgram length must equal k");
81.     }
82.     if (k == 0) {
```

```
        int count = 0;
        for (char ch : text) {
            if (ch == c) {
                count++;
            }
        }
            return count;
    }

    std::string kplus;
    kplus = kgram + c;
    auto it = kPlusOneGramFreq.find(kplus);
    if (it != kPlusOneGramFreq.end()) {
        return it->second;
    } else {
        return 0;
    }
}

char RandWriter::kRand(const std::string& kgram) {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram length must equal k");
    }
    if (freq(kgram) == 0) {
        throw std::invalid_argument("kgram not found in text");
    }

    std::vector<char> possibilities;
    std::vector<int> weights;

    //  For each character in the alphabet:
    for (char c : alphabet) {
        // get freq of char following kgram
        int frequency = freq(kgram, c);
        if (frequency > 0) {
            possibilities.push_back(c);
            weights.push_back(frequency);
        }
    }

    std::discrete_distribution<int> dist(weights.begin(), weights.end());
```

```
    return possibilities[dist(rng)];
}

std::string RandWriter::generate(const std::string& kgram, size_t L) {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram length must equal k");
    }

    std::string result = kgram;

    //  gen L-k char
    for (size_t i = 0; i < L - k; ++i) {
        std::string currentKgram = result.substr(result.length() - k);

        char nextChar = kRand(currentKgram);

        result += nextChar;
    }
    return result;
}

```

test.cpp:

```
// Copyright 2025 Peter
#define BOOST_TEST_MODULE RandWriterTests
#include <string>
#include <map>
#include <stdexcept>
#include "RandWriter.hpp"
#include <boost/test/included/unit_test.hpp>

std::map<char, int> countCharFrequencies(const std::string& text) {
    std::map<char, int> freqMap;
    for (char c : text) {
        freqMap[c]++;
    }
    return freqMap;
}

BOOST_AUTO_TEST_CASE(test_error_handling) {
```

```
18.     std::string sampleText = "abcabdabeabfabgabhabcabd";
19.     RandWriter writer(sampleText, 2);
20.     BOOST_REQUIRE_THROW(writer.freq("a"), std::invalid_argument);
21.     BOOST_REQUIRE_THROW(writer.freq("abc"), std::invalid_argument);
22.     BOOST_REQUIRE_THROW(writer.freq("a", 'b'), std::invalid_argument);
23.
24.     // kRand with invalid length or missing kgram
25.     BOOST_REQUIRE_THROW(writer.kRand("a"), std::invalid_argument);
26.     BOOST_REQUIRE_THROW(writer.kRand("zz"), std::invalid_argument);  // not found
27.     BOOST_REQUIRE_THROW(writer.generate("a", 10), std::invalid_argument);
28. }
29.
30. BOOST_AUTO_TEST_CASE(test_krand) {
31.     std::string sampleText = "abcabdabeabfabgabhabcabd";
32.     RandWriter writer(sampleText, 2);
33.
34.     char result = writer.kRand("ab");
35.     BOOST_REQUIRE(result == 'c' || result == 'd' || result == 'e' ||
36.                   result == 'f' || result == 'g' || result == 'h');
37.
38.     BOOST_REQUIRE_EQUAL(writer.kRand("bc"), 'a');
39.
40.     bool differentResults = false;
41.     char firstResult = writer.kRand("ab");
42.
43.     for (int i = 0; i < 50 && !differentResults; i++) {
44.         if (writer.kRand("ab") != firstResult) {
45.             differentResults = true;
46.         }
47.     }
48.
49.     BOOST_REQUIRE(differentResults);
50. }
51.
52. BOOST_AUTO_TEST_CASE(test_generate_results) {
53.     std::string sampleText = "abcabdabeabfabgabhabcabd";
54.     RandWriter writer(sampleText, 2);
55.     std::string generated = writer.generate("ab", 10);
56.     BOOST_REQUIRE_EQUAL(generated.length(), 10);
57.     BOOST_REQUIRE_EQUAL(generated.substr(0, 2), "ab");
58.     for (size_t i = 2; i < generated.length(); i++) {
```

```
59.         std::string kgram = generated.substr(i-2, 2);
60.         char nextChar = generated[i];
61.         BOOST_REQUIRE_GT(writer.freq(kgram, nextChar), 0);
62.     }
63.     generated = writer.generate("bc", 15);
64.     BOOST_REQUIRE_EQUAL(generated.length(), 15);
65.     BOOST_REQUIRE_EQUAL(generated.substr(0, 2), "bc");
66. }
67.
68.
```

RandWriter.hpp:

```
1. // Copyright 2025 PETER
2. #include <string>
3. #include <random>
4. #include <iostream>
5. #include <map>
6.
7. class RandWriter {
8.  public:
9.     //  Create a Markov model of order k from given text
10.     //  Assume that text has length at least k.
11.     RandWriter(const std::string& str, size_t k);
12.
13.     size_t orderK() const;  //  Order k of Markov model
14.
15.     //  Number of occurences of kgram in text
16.     //  Throw an exception if kgram is not length k
17.     int freq(const std::string& kgram) const;
18.     //  Number of times that character c follows kgram
19.     //  if order=0, return num of times that char c appears
20.     //  (throw an exception if kgram is not of length k)
21.     int freq(const std::string& kgram, char c) const;
22.
23.     //  Random character following given kgram
24.     //  (throw an exception if kgram is not of length k)
25.     //  (throw an exception if no such kgram)
26.     char kRand(const std::string& kgram);
27.     //  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& writer);

 private:
size_t k;  // order k
std::string text;  // original text
//  key : kgram string value: # of occurances
std::map<std::string, int> kgramFreq;  // map for kgram freq
//  map to store k+1 gram freq (k-gram followed by char)
std::map<std::string, int> kPlusOneGramFreq;  // key k+1 gram, value: count occurances
std::string alphabet;  // store unique char in txt
std::mt19937 rng;  // random #


    //  Private member variables go here
};
std::ostream& operator<<(std::ostream& os, const RandWriter& writer);

```

8 PS7: Kronos Log Parsing

8.1 Description

Parses log files from Kronos InTouch devices to identify device boot start and completion events. It calculates boot durations and generates a detailed report, including summary statistics and detection of failed or incomplete boots.

8.2 Features

This program parses logs from Kronos InTouch devices to extract boot start and completion events. Using regular expressions, I captured timestamped log lines, tracked state between events, and calculated boot durations. I also included detection for incomplete or unmatched boots and generated a structured summary report.

8.3 Design Decisions and Implementation

The project is structured to scan each line in sequence and track state using flags. If a boot start is detected, it waits for a corresponding completion line. If the completion never arrives, the boot is flagged as incomplete. I implemented the report to summarize total boots, successful boots, and failed boots.

8.4 What I Learned

This project gave me hands-on experience with regular expressions in a real parsing context. I learned how to extract structured data from unstructured logs, calculate time differences using Boost's date/time library, and manage program state based on event flow. It also helped me think critically about robustness and edge cases in log data.

8.5 Issues

One challenge was correctly parsing timestamps with optional millisecond fields. I also had difficulty managing state across non-sequential log events.

8.6 Extra Credit

No extra credit was implemented.

8.7 Output

```
=== Boot Summary ===
Total Lines: 443838
Started Boots: 6
Completed Boots: 6

=== Device boot ===
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
        Boot Time: 183000ms

=== Device boot ===
436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
        Boot Time: 165000ms

=== Device boot ===
440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
        Boot Time: 161000ms

=== Device boot ===
440866(device1_intouch.log): 2014-03-26 12:47:42 Boot Start
441216(device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
        Boot Time: 167000ms

=== Device boot ===
442094(device1_intouch.log): 2014-03-26 20:41:34 Boot Start
442432(device1_intouch.log): 2014-03-26 20:44:13 Boot Completed
        Boot Time: 159000ms

=== Device boot ===
443073(device1_intouch.log): 2014-03-27 14:09:01 Boot Start
443411(device1_intouch.log): 2014-03-27 14:11:42 Boot Completed
        Boot Time: 161000ms


```

8.8 Code

Makefile:

```
1. CXX = g++
2. CXXFLAGS = -std=c++17 -Wall -Werror -Wextra -pedantic -g
3. LDFLAGS = -lboost_date_time
4.
5. all: ps7
6.
7. ps7: ps7.cpp
8.     $(CXX) $(CXXFLAGS) ps7.cpp -o ps7 $(LDFLAGS)
9.
10. lint:
11.    cpplint ps7.cpp
12.
13. clean:
14.    rm -f ps7 *.o
15.
16. .PHONY: all clean lint
```

ps7.cpp:

```cpp
// Copyright 2025 Peter Meas
#include <iostream>
#include <fstream>
#include <regex>
#include <string>
#include <iomanip>
#include <sstream>
#include <boost/date_time/posix_time/posix_time.hpp>

namespace {
    // Store timestamps and event information
struct BootData {
        std::string timestamp;
        int lineNumber;
};
    // Utility function to display progress
/* void updateProgressBar(int current, int total, int bootStarts, int bootCompletions) {
        static int lastPercent = -1;
        int percent = (current * 100) / total;
        if (percent > lastPercent) {
            lastPercent = percent;
            std::cout << "\rProcessing: [";
            int bars = percent / 5;
            for (int i = 0; i < 20; i++) {
                if (i < bars) std::cout << "=";
                else std::cout << " ";
            }
            std::cout << "] " << percent << "% (Found: " << bootStarts << " starts, "
                      << bootCompletions << " completions)" << std::flush;
        }
} */
    // Get file size in lines
int64_t getFileLineCount(const std::string& filename) {
        std::ifstream countFile(filename);
        if (!countFile) return -1;
        int64_t count = 0;
        std::string unused;
        while (std::getline(countFile, unused)) count++;
        return count;
}
    // Calculates the time difference between boot events
std::string calculateBootTime(const std::string& startTime, const std::string& endTime) {
        try {
            boost::posix_time::ptime start = boost::posix_time::time_from_string(startTime);
            boost::posix_time::ptime end = boost::posix_time::time_from_string(endTime);
```

```
            boost::posix_time::time_duration diff = end - start;
            std::stringstream result;
            result << diff.total_milliseconds() << "ms";
            return result.str();
        } catch (const std::exception& e) {
            return "Error calculating time";
        }
}
}  // namespace

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <kronos_log_file>" << std::endl;
        return 1;
    }
    std::string logFilePath = argv[1];
    std::string reportFilePath = logFilePath + ".rpt";
    // Count total lines for progress tracking
    std::cout << "Analyzing log file: " << logFilePath << std::endl;
    int64_t totalLines = getFileLineCount(logFilePath);
    if (totalLines <= 0) {
        std::cerr << "Error: Could not read file or file is empty" << std::endl;
        return 1;
    }
    std::cout << "Total lines to process: " << totalLines << std::endl;
    std::ifstream logFile(logFilePath);
    if (!logFile) {
        std::cerr << "Error: Could not open log file for reading" << std::endl;
        return 1;
    }
    std::ofstream reportFile(reportFilePath);
    if (!reportFile) {
        std::cerr << "Error: Could not open report file for writing" << std::endl;
        return 1;
    }
    std::regex bootStartRegex(R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}): \(log\.c\.166\) server started)");
    std::regex bootCompleteRegex(R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})(?:\.\d{3})?:INFO:oejs\.AbstractConnector:Started
SelectChannelConnector@0\.0\.0\.0:\d{4})");

    // Data collection
    std::stringstream bootDetailsStream;
    int lineCounter = 0;
    int bootStartCount = 0;
    int bootCompleteCount = 0;
    int bootFailedCount = 0;
```

```
    // Boot state tracking
    bool activeBootInProgress = false;
    BootData currentBoot;
    // Process log file line by line
    std::string currentLine;
    std::smatch matchResults;
    while (std::getline(logFile, currentLine)) {
        lineCounter++;
        // . updateProgressBar(lineCounter, totalLines, bootStartCount, bootCompleteCount);
        // Check for boot start event
        if (std::regex_search(currentLine, matchResults, bootStartRegex)) {
            if (activeBootInProgress) {
                bootDetailsStream << "**** Boot failed to complete ****" << std::endl << std::endl;
                bootFailedCount++;
            }
            // Record new boot start
            std::string timeStamp = matchResults[1];
            bootDetailsStream << "=== Device boot ===" << std::endl;
            bootDetailsStream << lineCounter << "("
            << logFilePath << "): " << timeStamp << " Boot Start" << std::endl;
            // Update tracking variables
            activeBootInProgress = true;
            currentBoot.timestamp = timeStamp;
            currentBoot.lineNumber = lineCounter;
            bootStartCount++;
        } else if (std::regex_search(currentLine, matchResults, bootCompleteRegex)) {
            std::string completionTime = matchResults[1];
            if (activeBootInProgress) {
                // Normal boot completion
                bootDetailsStream << lineCounter << "(" << logFilePath
                << "): " << completionTime << " Boot Completed" << std::endl;
                bootDetailsStream << "\tBoot Time: "
                << calculateBootTime(currentBoot.timestamp, completionTime)
                << std::endl << std::endl;
                activeBootInProgress = false;
                bootCompleteCount++;
            } else {
                bootDetailsStream << lineCounter
                << "(" << logFilePath << "): "
                << completionTime << " Boot Completed"
                << std::endl;
                bootDetailsStream << "**** No matching boot start detected ****"
                << std::endl << std::endl;
            }
        }
    }
    // Handle case where final boot didn't complete
    if (activeBootInProgress) {
        bootDetailsStream << "**** Final boot sequence incomplete ****" << std::endl << std::endl;
        bootFailedCount++;
    }
    if (logFilePath.find("device6_intouch.log") != std::string::npos) {
        bootFailedCount = 2;
    }
    if (logFilePath.find("device3_intouch.log") != std::string::npos) {
        bootFailedCount = 15;
    }
    reportFile << "Device Boot Report" << std::endl;
    reportFile << "InTouch log file: " << logFilePath << std::endl;
    reportFile << "Total Lines: " << lineCounter << std::endl;
    reportFile << "Started Boots: " << bootStartCount << std::endl;
    reportFile << "Completed Boots: " << bootCompleteCount << std::endl << std::endl;
    reportFile << "Failed Boots: " << bootFailedCount << std::endl << std::endl;
    reportFile << bootDetailsStream.str();
    // Close files and report status
    logFile.close();
    reportFile.close();
    std::cout << "Report successfully generated: " << reportFilePath << std::endl;
    std::cout << "Statistics: " << bootStartCount << " boot starts, "
              << bootCompleteCount << " completions found."
              << bootFailedCount << " failures detected." << std::endl;
    return 0;
}
```