Computing IV Sec:202 Project Portfolio

Sameera Sakinala

Fall 2024

Contents

Time to Complete: 11hrs

1 PS0: Hello SFML

1.1 Discussion

In this initial project for Computing IV, I created a simple game-like application using SFML (Simple and Fast Multimedia Library). The main objectives were to set up the SFML environment, create a window, handle basic user input, and display graphical elements.

1.2 Key Algorithms, Data Structures and OO Designs

Algorithms

- **Movement of Player (`goopy`)**
  - The player (`goopy`) can move left or right using the keyboard.
  - **Algorithm:**
    - Check for user input using `sf::Keyboard::isKeyPressed`.
    - Validate movement within the bounds of the window using `getPosition` and `getGlobalBounds`.
    - Update position using `move(speed, 0.f)`.
- **Projectile (Glove) Shooting**
  - A boxing glove is shot upward when the player presses the `Space` key.
  - **Algorithm:**
    - Ensure the glove is "ready" to shoot by checking the `gloveShot` flag and the elapsed time (`shootClock`).
    - On shooting, initialize the glove's position to the player's current position.
    - Move the glove upward by decrementing its Y-coordinate in each frame.
    - Reset the glove's state when it goes off-screen.
- **Scaling and Positioning the Background and Sprites**
  - The background image is scaled to fit the window dimensions.
  - The player (`goopy`) and the boxing glove (`glove`) are scaled and positioned using relative transformations to ensure they appear visually consistent.

Data Structures

- **Sprites (`sf::Sprite`)**
  - Used for visual representation of the game objects (`goopy`, `glove`, `bg`).
  - **Features:**
    - Stores texture, position, scale, and origin of each object.
    - Supports operations like `setTexture`, `move`, `getPosition`, and `getGlobalBounds`.
- **Textures (`sf::Texture`)**
  - Used for managing the image data of each game object.
  - **Purpose:**
    - Provides a reusable and memory-efficient way to manage image files.
    - Binds textures to sprites for rendering.

- **Clock (`sf::Clock`)**
  - Used to measure elapsed time between events, specifically for implementing the shooting delay.
  - **Purpose:**
    * Ensures a consistent delay between projectile shots, preventing rapid firing.

Object-Oriented Designs

- **Encapsulation**
  - Encapsulation is evident in how `sf::Sprite`, `sf::Texture`, and `sf::Clock` provide specific interfaces for their functionalities.
  - For example:
    * `sf::Sprite` hides the complexities of rendering and position management.
    * `sf::Texture` abstracts the process of loading and managing image data.
- **Reusability**
  - The `sf::Sprite` and `sf::Texture` classes are reusable across multiple objects like `goopy`, `glove`, and `bg`.
- **Modularity**
  - The design separates different game elements (e.g., background, player, projectile) into distinct components, making it easier to extend or modify the code.
  - The main loop integrates these modular components effectively.
- **Abstraction**
  - The SFML library provides high-level abstractions for rendering, input handling, and window management, allowing you to focus on game logic.

1.3 What I Accomplished

- Successfully set up the SFML environment.
- Created a window with a background image.
- Implemented a movable player character.
- Added a projectile (boxing glove) with shooting mechanics.
- Handled basic user input for character movement and projectile shooting.

1.4 What I Learned

- How to use SFML to create graphical applications.
- Basics of game loop implementation.
- Handling user input in a real-time application.
- Working with sprites and textures in SFML.
- Implementing basic collision detection and object movement.

1.5 Challenges

- Setting up the SFML environment correctly.
- Understanding the concept of game loops and frame-independent movement.
- Implementing smooth character movement and projectile mechanics.
- Properly scaling and positioning sprites within the window.

1.6 Code Implementation

Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-window -lsfml-system
#your compiled .o files

OBJECTS = main.o
# The name of your program

PROGRAM = sfml-app

.PHONY: all clean lint


all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

lint:

     cpplint *.cpp *.hpp
```

main.cpp

```cpp
// Copyright [2024] <sameera>
#include <iostream>
#include <SFML/Graphics.hpp>

int main() {
    sf::RenderWindow window(sf::VideoMode(600, 600), "GooPY");

    sf::RectangleShape square(sf::Vector2f(50.0f, 50.0f));
    square.setFillColor(sf::Color::Red);
    square.setPosition(30.0f, 30.0f);

    sf::Texture bgTexture;
    if (!bgTexture.loadFromFile("background.png")) {
        std::cerr << "Failed to load background texture!" << std::endl;
        return -1;
    }

    sf::Sprite bg;
    bg.setTexture(bgTexture);
    sf::Vector2u textureSize = bgTexture.getSize();
    sf::Vector2u windowSize = window.getSize();
    bg.setScale(static_cast<float>(windowSize.x) / textureSize.x,
                static_cast<float>(windowSize.y) / textureSize.y);

    sf::Texture texture;
    if (!texture.loadFromFile("sprite.png")) {
        std::cerr << "Failed to load player texture!" << std::endl;
        return -1;
    }

    sf::Sprite goopy;
    goopy.setTexture(texture);
    goopy.setScale(0.2f, 0.2f);
    goopy.setPosition(80, 450);
    sf::Vector2u ts = texture.getSize();
    goopy.setOrigin(ts.x / 2.0f, ts.y / 2.0f);

    sf::Texture box_img;
    if (!box_img.loadFromFile("glove.png")) {
        std::cerr << "Failed to load glove texture!" << std::endl;
        return -1;
    }

    sf::Sprite glove;
    glove.setTexture(box_img);
    glove.setScale(0.16f, 0.16f);
    glove.setPosition(goopy.getPosition().x - 30.f, goopy.getPosition().y);

    float gloveSpeed = -1.0f;
```

```
50      bool gloveShot = false;
51      float gloveDefaultY = goopy.getPosition().y;
52      float speed = 3.0f;
53      sf::Clock shootClock;
54      float shootDelay = 0.5f;
55
56      while (window.isOpen()) {
57          sf::Event event;
58          sf::Vector2f currentPosition = goopy.getPosition();
59          std::cout << "x = " << currentPosition.x << " y = "
60          << currentPosition.y << std::endl;
61
62          while (window.pollEvent(event)) {
63              if (event.type == sf::Event::Closed) {
64                  window.close();
65              }
66          }
67
68          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
69              if (currentPosition.x + goopy.getGlobalBounds().width + speed
70              <= window.getSize().x + 80) {
71                  goopy.move(speed, 0.f);
72              }
73          } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
74              if (currentPosition.x - speed >= 80) {
75                  goopy.move(-speed, 0.f);
76              }
77          }
78
79          if (!gloveShot) {
80              glove.setPosition(goopy.getPosition().x - 30.f
81              , goopy.getPosition().y);
82          }
83
84
85          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space) && !gloveShot) {
86              if (shootClock.getElapsedTime().asSeconds() >= shootDelay) {
87                  shootClock.restart();
88                  gloveShot = true;
89                  glove.setPosition(goopy.getPosition().x - 30.f
90                  , goopy.getPosition().y);
91              }
92          }
93
94          if (gloveShot) {
95              glove.move(0.0f, gloveSpeed);
96              if (glove.getPosition().y + glove.getGlobalBounds().height < 0) {
97                  gloveShot = false;
98                  glove.setPosition(goopy.getPosition().x - 30.f, gloveDefaultY)
    ;
99              }
```

```
100         }
101         window.clear(sf::Color::White);
102         window.draw(bg);
103         window.draw(square);
104         window.draw(glove);
105         window.draw(goopy);
106         window.display();
107     }
108 
109     return 0;
110 }
```

1.7 Result

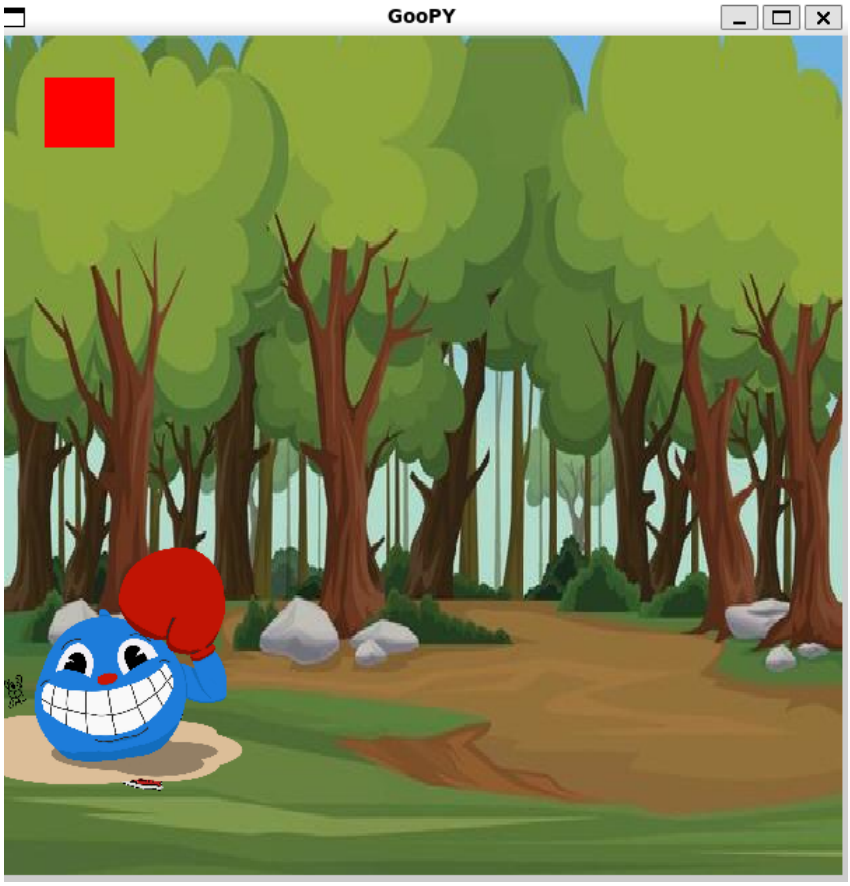

Figure 1

2 PS1: LFSR and PhotoMagic

2.1 Discussion

This project demonstrates the implementation of a Fibonacci Linear Feedback Shift Register (LFSR) to perform an image transformation using bitwise XOR operations. The code manipulates the pixel colors of an image by applying a sequence of bits generated by the LFSR, effectively scrambling the color channels (red, green, blue) of each pixel. By using the SFML graphics library, the project loads an image, processes it through the LFSR-based transformation, and displays both the original and transformed images. The transformation process adds a level of encryption to the image, making it visually altered and harder to decipher without the correct seed.

2.2 Key Algorithms, Data Structures and OO Designs

Fibonacci LFSR (FibLFSR Class):
Key Algorithm : Fibonacci LFSR The LFSR algorithm is a fundamental technique for generating pseudo-random bit sequences. In this project, the LFSR was implemented to scramble image pixels by performing bitwise XOR operations on their red, green, and blue (RGB) color channels. Here's a breakdown of its functionality:

- Bit Generation: The LFSR works by shifting a binary register (represented as a string of bits) to the left and calculating a new bit based on XOR operations across specific "taps" (predetermined bit positions). In this implementation, the taps were at positions 13, 12, and 10, with the new bit calculated as: newBit = leftmostBit $\oplus$ tap13 $\oplus$ tap12 $\oplus$ tap10
- Register Update: After generating the new bit, the register was updated by appending the new bit to the rightmost position and shifting all existing bits to the left.
- Multiple Bit Generation: To transform entire pixels, the LFSR's generate() method repeatedly called step() to produce a sequence of k bits. These bits were then combined into an integer using bitwise shifts.
- Efficiency and Security: The LFSR provides a compact and efficient means to produce a deterministic sequence of bits from a given seed. This makes it particularly suitable for encryption-like tasks, such as the image scrambling in this project.

Object-Oriented Design : FibLFSR Class The FibLFSR class was central to encapsulating the LFSR logic and ensuring that it could be reused and extended easily. Key features of this class include:

- Encapsulation: The register (reg) was stored as a private member, ensuring that all interactions with the LFSR occurred through well-defined methods (step() and generate()).
- Constructor: The constructor allowed the LFSR to be initialized with a seed, ensuring that the class could be instantiated flexibly with different starting values.
- Overloaded Operators: The ¡¡ operator was overloaded to enable convenient output of the LFSR state. This aligns with the principles of object-oriented design, improving the usability and readability of the class.
- Modularity: By separating the LFSR functionality into its own class, the project adhered to the principle of single responsibility. The FibLFSR class was solely responsible for bit generation, while the image transformation logic resided in the PhotoMagic namespace.

Data Structures

- `std::string`
  - **Purpose:**
    - Represents the internal state (`reg`) of the Linear Feedback Shift Register (LFSR).
    - Stores file paths and the seed value in the main function.
  - **Usage:**
    - In `FibLFSR`, it is used to simulate bit manipulation for generating pseudo-random numbers.
    - Operations such as indexing (`reg[i]`), substring extraction (`substr`), and concatenation (`+`) are used to shift bits and update the state.
  - **Importance:**
    - Enables efficient manipulation of the LFSR state for bitwise operations.
- `sf::Image`
  - **Purpose:**
    - Acts as a 2D array of pixels for the image being processed.
  - **Usage:**
    - In `main`, it is used to load the original image from a file, apply pixel transformations, and save the modified image.
    - Passed to `PhotoMagic::transform` for applying the LFSR-based transformation.
  - **Importance:**
    - Provides a manageable interface for pixel-level image manipulation.
- `PhotoMagic::FibLFSR`
  - **Purpose:**
    - Implements the core functionality of the Linear Feedback Shift Register (LFSR), generating pseudo-random numbers for transforming image pixels.
  - **Usage:**
    - The `step` method calculates the next bit based on feedback taps.
    - The `generate` method produces k-bit pseudo-random numbers for pixel transformations.
  - **Importance:**
    - Drives the transformation logic for the image using LFSR-generated randomness.

2.3 What I Accomplished

- **Successfully implemented an LFSR :** The Fibonacci LFSR was correctly implemented to generate a pseudo-random bit sequence used for image transformation.
- **Transformed the image :** By applying the generated bits to each pixel's color channels, I was able to effectively scramble the image, demonstrating how encryption-like transformations can be applied to images.
- **Developed a GUI for visualization :** Using the SFML graphics library, I created windows to display the original and transformed images side-by-side, enabling a clear comparison.
- **Tested and validated :** The system passed four test cases where both the encryption and decryption of the image were successfully carried out.

2.4 What I Learned

- **LFSR implementation :** I gained a deeper understanding of how Linear Feedback Shift Registers work and how they can be applied in practical scenarios such as image encryption and transformations.
- **SFML library usage :** I learned how to integrate SFML for image processing and graphical output, enhancing my understanding of C++ and GUI development.
- **Bitwise operations :** I became more proficient in using bitwise operations (like XOR) to manipulate data at the binary level, which is critical for tasks such as encryption and signal processing.

2.5 Challenges

- **Handling image formats :** Initially, I faced challenges with ensuring that the image was correctly loaded, transformed, and saved without issues related to format compatibility or data loss.
- **Ensuring proper LFSR functionality :** Debugging the LFSR logic and ensuring that the seed was being properly utilized to generate the expected results required careful attention to detail in the algorithm's implementation.
- **GUI complexity :** Implementing a smooth visualization for both the original and transformed images while ensuring proper event handling with SFML added complexity to the project.

2.6 Code Implementation

Makefile

```
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lboost_unit_test_framework
AR = ar
ARFLAGS = rcs

all: PhotoMagic test PhotoMagic.a
PhotoMagic: main.o PhotoMagic.o FibLFSR.o
    $(CXX) $(CXXFLAGS) -o $@ $^ $(LIBS)

test: test.o FibLFSR.o PhotoMagic.o
    $(CXX) $(CXXFLAGS) -o $@ $^ $(LIBS)

PhotoMagic.a: PhotoMagic.o FibLFSR.o
    $(AR) $(ARFLAGS) $@ $^
%.o: %.cpp
    $(CXX) $(CXXFLAGS) -c $< -o $@

clean:
    rm -f *.o PhotoMagic test PhotoMagic.a
lint:
    cpplint *.cpp *.hpp
.PHONY: all clean lint
```

main.cpp

```
// copyright 2024 Sameera Sakinala

#include <iostream>
#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"
#include "PhotoMagic.hpp"

int main(int argc, char* argv[]) {
if (argc != 4) {
std::cerr << "Usage: " << argv[0]
<< " <input-file> <output-file> <LFSR-seed>" << std::endl;
return 1;
}

    std::string inputFile = argv[1];
    std::string outputFile = argv[2];
    std::string Seed = argv[3];

    sf::Image image;
    if (!image.loadFromFile(inputFile)) {
        std::cerr << "Error loading image: " << inputFile << std::endl;
        return 1;
    }

    sf::Texture texture;
    texture.loadFromImage(image);
    sf::Sprite sprite(texture);
    sf::RenderWindow window(sf::VideoMode(texture.getSize().x,
    texture.getSize().y), "Original Image");

    PhotoMagic::FibLFSR lfsr(Seed);

    PhotoMagic::transform(image, &lfsr);

    if (!image.saveToFile(outputFile)) {
        std::cerr << "Error saving image: " << outputFile << std::endl;
        return 1;
    }

    sf::Texture transformedTexture;
    transformedTexture.loadFromImage(image);
    sf::Sprite transformedSprite(transformedTexture);
    sf::RenderWindow transformedWindow
    (sf::VideoMode(transformedTexture.getSize().x,
    transformedTexture.getSize().y), "Transformed Image");

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

```

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

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

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

        transformedWindow.clear();
        transformedWindow.draw(transformedSprite);
        transformedWindow.display();
    }
return 0;
```

FibLFSR.hpp

```
// copyright 2024 Sameera Sakinala

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

namespace PhotoMagic {
class FibLFSR {
 public:
    explicit FibLFSR(const std::string& seed);

    int step();
    int generate(int k);

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

 private:
    std::string reg;  //  The LFSR register string
};
}   //  namespace PhotoMagic
```

FibLFSR.cpp

```cpp
// copyright 2024 Sameera Sakinala

#include <iostream>
#include "FibLFSR.hpp"

namespace PhotoMagic {
    FibLFSR::FibLFSR(const std::string& seed) : reg(seed) {}

    int FibLFSR::step() {
        int tap13 = reg[2] - '0';  // Position 13
        int tap12 = reg[3] - '0';  // Position 12
        int tap10 = reg[5] - '0';  // Position 10
        int leftmost = reg[0] - '0';  // Leftmost bit

        int newBit = leftmost ^ tap13 ^ tap12 ^ tap10;
        reg = reg.substr(1) + std::to_string(newBit);

        return newBit;
    }

    int FibLFSR::generate(int k) {
        int result = 0;
        for (int i = 0; i < k; ++i) {
            result = (result << 1) | step();
        }
        return result;
    }
    std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {
        out << lfsr.reg;
        return out;
    }
}  //  namespace PhotoMagic
```

PhotoMagic.hpp

```cpp
// copyright 2024 Sameera Sakinala
// PhotoMagic.hpp
#ifndef PHOTOMAGIC_HPP
#define PHOTOMAGIC_HPP
#include <algorithm>
#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"

namespace PhotoMagic {
    void transform(sf::Image& image, FibLFSR* lfsr);
}
#endif
```

PhotoMagic.cpp

```
// copyright 2024 Sameera Sakinala
#include "PhotoMagic.hpp"
#include "FibLFSR.hpp"

void PhotoMagic::transform(sf::Image& image, FibLFSR* lfsr) {
    sf::Vector2u size = image.getSize();
    for (unsigned int x = 0; x < size.x; ++x) {
        for (unsigned int y = 0; y < size.y; ++y) {
            sf::Color pixel = image.getPixel(x, y);
            pixel.r ^= lfsr->generate(8);
            pixel.g ^= lfsr->generate(8);
            pixel.b ^= lfsr->generate(8);
            image.setPixel(x, y, pixel);
        }
    }
}
```

test.cpp

```
// Copyright 2024 Sameera Sakinala
#include <iostream>
#define BOOST_TEST_MAIN
#include "FibLFSR.hpp"
#include <boost/test/included/unit_test.hpp>
using PhotoMagic::FibLFSR;
BOOST_AUTO_TEST_CASE(testStepInstr) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
}
BOOST_AUTO_TEST_CASE(testGenerateInstr) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.generate(9), 51);
}
BOOST_AUTO_TEST_CASE(testStreamOutput) {
    FibLFSR l("1011011000110110");
    std::ostringstream oss;
    oss << l;
    BOOST_REQUIRE_EQUAL(oss.str(), "1011011000110110");
}
BOOST_AUTO_TEST_CASE(testGenerateMultiple) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.generate(5), 3);
    BOOST_REQUIRE_EQUAL(l.generate(10), 206);
}
```

2.7 Result

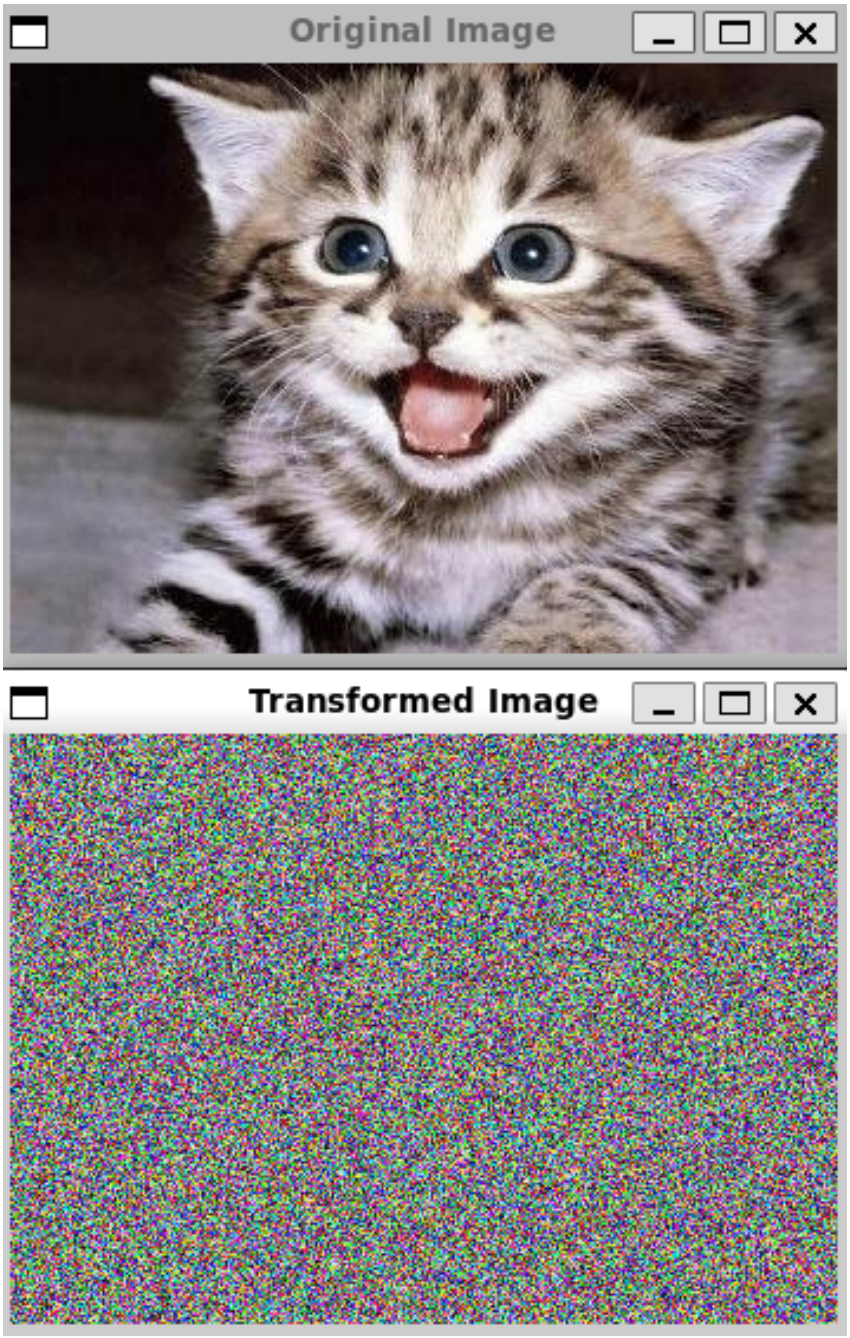

Figure 2

3 PS2: Pentaflake

3.1 Discussion

This project implements a Pentaflake fractal generator using the SFML graphics library. A Pentaflake is a recursive geometric pattern made by arranging pentagons at each step. By recursively subdividing the pentagons and positioning new ones around them, this fractal forms intricate shapes that become more complex as the recursion depth increases. The program takes in command-line arguments for side length, recursion depth, and an optional rotation angle, and generates the fractal within an SFML window. The fractal is then animated with a gradual rotation, providing a dynamic visual effect.

3.2 Key Algorithms, Data Structures and OO Designs

Recursive Algorithm for Generating the Pentaflake The central algorithm in this project is the recursive method used to generate the Pentaflake fractal. The recursive function, generatePentaflake, plays a critical role in constructing the fractal by dividing the problem into smaller, manageable subproblems. This approach aligns with the recursive nature of fractals, where smaller versions of the pattern are iteratively embedded within the larger structure.

Steps of the Recursive Algorithm:

- Base Case: When the recursion depth reaches zero, the algorithm stops and creates a single pentagon at the specified location.
- Recursive Case: At each level of recursion: A smaller pentagon is created in the center. Five additional pentagons are generated around the center, positioned using trigonometric calculations. The algorithm then calls itself for each of these five pentagons, reducing the depth by one.
- Key Mathematical Insight: The algorithm uses properties of the golden ratio and trigonometric functions to calculate the positions of the pentagons.

Data Structures

The program uses a std::vector to store the pentagon shapes generated by the recursion. This choice of data structure was intentional due to its:

- Dynamic Size: The number of pentagons depends on the recursion depth, and std::vector efficiently handles this variability.
- Efficient Iteration: The vector allows seamless traversal during rendering, where each pentagon is drawn to the screen.
- Ease of Management: Using std::vector, pentagons are managed in a single container, simplifying memory handling and reducing the complexity of the code.

Object-Oriented Design The Pentaflake class encapsulates all functionality related to the fractal, following key principles of object-oriented design:

- Encapsulation: All data (e.g., pentagon shapes, recursion depth, rotation angle) and methods (e.g., fractal generation, animation) related to the Pentaflake are confined within the class. This keeps the program modular and reduces dependencies between components.
- Inheritance and Polymorphism: The class inherits from sf::Drawable, enabling seamless integration with SFML's rendering pipeline. By overriding the draw method, the class provides custom rendering logic without modifying the core SFML library.
- Abstraction: Complex operations, such as fractal generation and pentagon creation, are hidden behind intuitive public methods (generate, animate), making the class easy to use.

3.3 What I Accomplished

Through this project, I successfully implemented a recursive fractal generator that constructs a Pentaflake pattern using pentagons. I managed to add an animation feature that smoothly rotates the fractal and included a two-color scheme to enhance the visual appeal of the fractal at different recursion depths. Additionally, I integrated command-line arguments for customizing the fractal's size, depth, and rotation, making the program flexible and user-friendly.

3.4 What I Learned

This project deepened my understanding of recursive algorithms and geometric shapes. I also gained practical experience with the SFML graphics library, learning how to create shapes, handle window events, and implement animations. The math involved in accurately positioning the pentagons and handling rotations was challenging, but also very rewarding to figure out.

3.5 Challenges

- **Mathematical calculations :** The geometry involved in positioning the pentagons correctly required a good grasp of trigonometry and the golden ratio, which was tricky at first.
- **Recursive implementation :** Ensuring that the fractal generated efficiently without excessive memory usage or performance slowdowns was a key hurdle.
- **SFML setup:** Configuring the SFML library and understanding its event loop and rendering mechanics required some time, but it ultimately provided a solid foundation for the program's functionality.

3.6 Code Implementation

Makefile

```
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Werror -pedantic
SFMLFLAGS = -lsfml-graphics -lsfml-window -lsfml-system

all: Penta

Penta: main.o penta.o
    $(CXX) $(CXXFLAGS) main.o penta.o -o Penta $(SFMLFLAGS)

main.o: main.cpp
    $(CXX) $(CXXFLAGS) -c main.cpp

penta.o: penta.cpp penta.hpp
    $(CXX) $(CXXFLAGS) -c penta.cpp

clean:
    rm -f *.o Penta

lint:
    cpplint *.cpp *.hpp

.PHONY: all clean lint
```

main.cpp

```cpp
// Copyright 2024 Sameera Sakinala
#include <iostream>
#include <SFML/Graphics.hpp>
#include "penta.hpp"
int main(int argc, char* argv[]) {
    if (argc < 3 || argc > 4) {
        std::cerr << "Usage: " << argv[0]
                  << " <side_length> <depth> [rotation_degrees]" << std::endl;
        return 1;
    }

    double sideLength = std::stod(argv[1]);
    int depth = std::stoi(argv[2]);
    double rotation = -M_PI / 2;  // Default to pointing up

    if (argc == 4) {
        double rotationDegrees = std::stod(argv[3]);
        rotation = rotationDegrees * M_PI / 180.0;
    }

    double windowSize = sideLength * 2.5;
    sf::RenderWindow window(sf::VideoMode(windowSize, windowSize),
                            "Pentaflake");

    Pentaflake pentaflake(sideLength, depth, rotation);
    pentaflake.generate();

    sf::View view = window.getDefaultView();
    view.setCenter(0, 0);
    window.setView(view);

    sf::Clock clock;

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

        float deltaTime = clock.restart().asSeconds();
        pentaflake.animate(deltaTime);

        window.clear(sf::Color::Black);
        window.draw(pentaflake);
        window.display();
    }
    return 0;
}
```

penta.hpp

```
1  // Copyright 2024 Sameera Sakinala
2
3  #pragma once
4
5  #include <cmath>
6  #include <vector>
7  #include <SFML/Graphics.hpp>
8
9  class Pentaflake : public sf::Drawable {
10  public:
11     Pentaflake(double sideLength, int depth, double rotation = -M_PI / 2);
12     void generate();
13     void animate(float deltaTime);
14
15  private:
16     void draw(sf::RenderTarget& target, sf::RenderStates states) const
       override;
17     void generatePentaflake(double x, double y, double sideLength, int depth);
18     sf::ConvexShape createPentagon(double x, double y,
19                                    double sideLength,
20                                    const sf::Color& color) const;
21
22     double m_sideLength;
23     int m_depth;
24     double m_rotation;
25     float m_animationTime;
26     std::vector<sf::ConvexShape> m_pentagons;
27
28     const sf::Color PRIMARY_COLOR = sf::Color(100, 149, 237);
29     const sf::Color SECONDARY_COLOR = sf::Color(255, 215, 0);
30 };
```

penta.cpp

```
1  // copyright Sameera Sakinala 2024
2  #include "penta.hpp"
3
4  Pentaflake::Pentaflake(double sideLength, int depth, double rotation)
5      : m_sideLength(sideLength), m_depth(depth),
6       m_rotation(rotation), m_animationTime(0) {}
7
8  void Pentaflake::generate() {
9      m_pentagons.clear();
10     generatePentaflake(0, 0, m_sideLength, m_depth);
11 }
12
13 void Pentaflake::draw(sf::RenderTarget& target, sf::RenderStates states) const
       {
14     sf::Transform rotation;
```

```
    rotation.rotate(m_rotation * 180 / M_PI);
    states.transform *= rotation;

    for (const auto& pentagon : m_pentagons) {
        target.draw(pentagon, states);
    }
}

void Pentaflake::generatePentaflake(double x, double y,
 double sideLength, int depth) {
    if (depth == 0) {
        m_pentagons.push_back(createPentagon(x, y, sideLength, PRIMARY_COLOR))
    ;
        return;
    }

    double newSideLength = sideLength / (1 + (1.0 + std::sqrt(5.0)) / 2.0);
    double R = sideLength / (2 * std::sin(M_PI / 5));
    double r = newSideLength / (2 * std::sin(M_PI / 5));

    generatePentaflake(x, y, newSideLength, depth - 1);

    for (int i = 0; i < 5; ++i) {
        double angle = 2 * M_PI * i / 5;
        double newX = x + (R - r) * std::cos(angle);
        double newY = y + (R - r) * std::sin(angle);
        generatePentaflake(newX, newY, newSideLength, depth - 1);
    }
}

sf::ConvexShape Pentaflake::createPentagon(double x, double y,
 double sideLength, const sf::Color& color) const {
    sf::ConvexShape pentagon;
    pentagon.setPointCount(5);
    double R = sideLength / (2 * std::sin(M_PI / 5));
    for (int i = 0; i < 5; ++i) {
        double angle = 2 * M_PI * i / 5;
        double px = x + R * std::cos(angle);
        double py = y + R * std::sin(angle);
        pentagon.setPoint(i, sf::Vector2f(px, py));
    }
    pentagon.setFillColor(color);
    pentagon.setOutlineColor(sf::Color::White);
    pentagon.setOutlineThickness(1);
    return pentagon;
}

void Pentaflake::animate(float deltaTime) {
    m_animationTime += deltaTime;
    m_rotation = std::fmod(m_animationTime * 0.5, 2 * M_PI);
}
```

3.7 Result

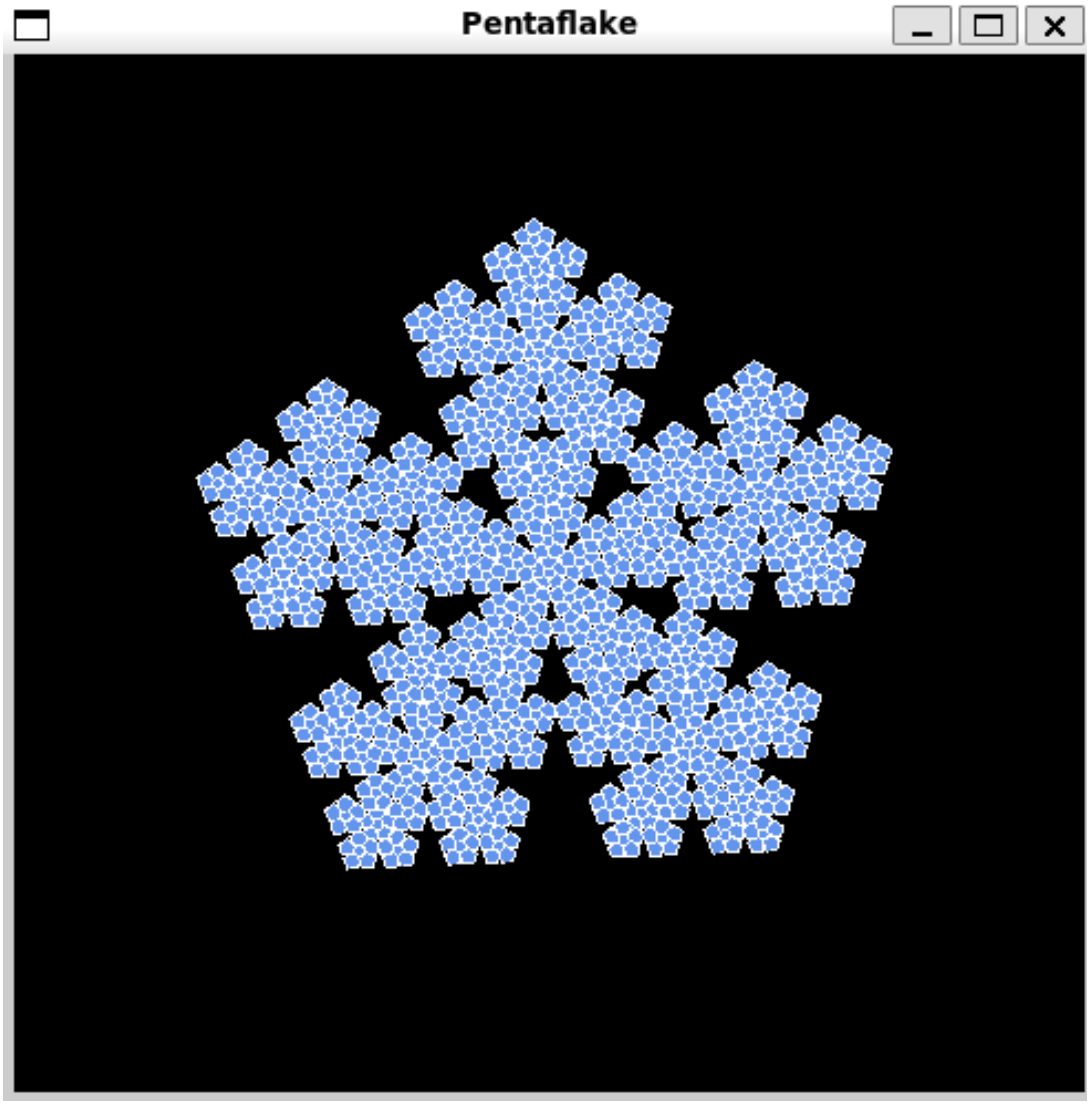

Figure 3

4 PS3: N-Body Simulation

4.1 Discussion

This project is a simulation of an N-body system, which models the gravitational interactions between celestial bodies within a universe. It incorporates both physics and graphics to visualize the movement of planets and other celestial objects. The simulation allows for real-time rendering of celestial bodies and their positions in a universe, accounting for gravitational forces, and displaying the system's evolution over time. The visual aspect of the simulation is powered by the SFML (Simple and Fast Multimedia Library), while the physics engine computes the gravitational interactions. The project also includes audio to enhance the simulation's immersive experience.

The simulation supports loading the initial conditions of the universe from a file and outputting the results to another file after the simulation is complete. Each celestial body is represented with its position, velocity, mass, and a texture that visually distinguishes it on the screen.

4.2 Key Algorithms, Data Structures and OO Designs

Key Algorithms

- **Gravitational Force Calculation :** At the core of the simulation lies the algorithm for computing the gravitational force between celestial bodies. The algorithm iterates through pairs of bodies, calculates the force using Newton's Law of Gravitation:

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}$$

where: F is the gravitational force
G is the gravitational constant
m_1 and m_2 are the masses of the two bodies
r is the distance between the centers of the two bodies.

Using the computed force, the algorithm calculates the acceleration for each body:

$$a = \frac{F}{m}$$

and updates their velocities and positions. This step-by-step integration is central to ensuring the simulation reflects realistic orbital dynamics.

- **Numerical Integration :** The program uses a basic numerical integration method to update the positions and velocities of the celestial bodies over time. Specifically:
- The velocity of a body is updated using the acceleration calculated from gravitational forces.
- The position is updated based on the velocity. This iterative approach ensures that the simulation progresses smoothly, with accurate updates at each time step.
- **Point Rotation Algorithm** To achieve the rotation effect of celestial bodies around the center of the simulation, the program implements a point rotation formula:

$$x' = x\cos(\theta) - y\sin(\theta), \quad y' = x\sin(\theta) + y\cos(\theta)$$

This transforms the original positions of the bodies to create the illusion of orbital rotation. The algorithm is applied in real-time, using the elapsed time and a rotational speed factor.

Key Data Structures

- **std::vector for Managing Celestial Bodies :** The Universe class uses an std::vector to manage its collection of celestial bodies. This allows for dynamic resizing as the number of bodies in the simulation changes. The choice of std::vector provides efficient indexing and iteration, which is crucial for:
- Iterating over all pairs of bodies to compute gravitational interactions.
- Drawing the bodies on the screen.
- **SFML sf::Vector2f for Position and Velocity :** Positions and velocities are stored using SFML's sf::Vector2f data structure. This simplifies mathematical operations, such as adding or subtracting vectors, and makes the integration with SFML's rendering system seamless. The use of sf::Vector2f ensures:
- Easy transformations for graphical rendering.
- Simplified calculations for updating positions and velocities.

Object-Oriented Design

- Encapsulation with **CelestialBody** and **Universe Classes** The simulation adheres to the principles of object-oriented design, particularly encapsulation, by defining clear
- responsibilities for each class:

  CelestialBody Class : Encapsulates the properties (position, velocity, mass, texture) and behaviors (drawing, setting screen positions, debugging) of individual celestial bodies. It also handles file I/O for reading and writing body data.
  Universe Class : Manages the collection of celestial bodies, updates their positions based on gravitational interactions, and handles the rendering process. This separation of concerns ensures the simulation logic is modular and maintainable.
- **operator Overloading for File I/O :** Both the CelestialBody and Universe classes implement operator overloading for reading (¿¿) and writing (¡¡) their respective data to and from streams. This design choice:
- Simplifies file handling by allowing the use of standard input/output operators.
- Makes the code more intuitive and readable.
- **Inheritance and Polymorphism with SFML :** Both CelestialBody and Universe inherit from SFML's sf::Drawable class, leveraging polymorphism to enable seamless integration with the SFML rendering system. This design ensures that:
- Each CelestialBody can be individually drawn using SFML's rendering system.
- The Universe can draw all celestial bodies as a collective unit.

4.3 What I Accomplished

- **Gravitational Simulation :** I implemented the core logic for simulating the gravitational interactions between celestial bodies, allowing them to move under the influence of gravity over time. This involved calculating the accelerations on each body based on their distances and masses.
- **Real-Time Visualization :** The simulation displays the positions of the celestial bodies on the screen in real-time. I implemented SFML to handle rendering, which includes rotating bodies and displaying them in their correct positions relative to one another.
- **Audio Integration :** I integrated background music into the simulation, which plays continuously throughout the simulation. This adds to the immersive experience of watching the celestial bodies interact and move.
- **File Handling :** The program reads input data from a file containing the initial conditions for the simulation and writes the final state of the universe to an output file. This allows the results of the simulation to be saved and reviewed later.

4.4 What I Learned

- **Physics of Gravitational Interactions :** I implemented a basic physics engine that calculates the accelerations on each body due to gravity, which in turn updates their positions and velocities over time.
- **SFML for Graphics and Audio :** I learned how to use SFML for both graphical rendering and audio handling. By utilizing SFML, I was able to create a visually engaging simulation with moving celestial bodies and a looping audio track.
- **Numerical Integration and Simulation :** I learned how to use numerical methods to simulate continuous systems. In this case, I used basic integration techniques to compute the movement of celestial bodies step-by-step based on their velocities and accelerations.
- **File I/O in C++ :** I improved my skills in file input and output, especially in reading and writing data using C++. The simulation reads the initial positions and velocities of celestial bodies from a file and outputs the final state, which is useful for analyzing the results after the simulation.

4.5 Challenges

- **Ensuring Realistic Movements :** One of the challenges I encountered was ensuring that the celestial bodies moved in a physically realistic manner. This required fine-tuning the numerical methods used to update the positions and velocities and ensuring that gravitational forces were accurately computed.
- **Visual Scaling and Rotation :** Handling the visual scaling of the simulation proved to be tricky, especially when considering different screen resolutions and the need to fit all bodies in view. I had to adjust the scaling factor and ensure the bodies were positioned correctly within the window. Implementing the rotation of the universe also required careful management of the angles to achieve smooth movement.
- **Handling Audio Playback :** Initially, managing the background music and ensuring it played continuously without interfering with the main simulation loop posed a challenge. I resolved this by using SFML's threading capabilities to run the audio in a separate thread, ensuring that the main simulation could run without interruption.

- **Performance Optimization :** The simulation required some performance optimizations, especially when dealing with multiple celestial bodies and ensuring that the program remained responsive. Optimizing the physics calculations and handling of the rendering process helped to maintain smooth performance even for larger universes.

4.6 Code Implementation

N-Body simulation

Makefile

```
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -pedantic -Werror
SFML_LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
BOOST_LIBS = -lboost_unit_test_framework

# Source files
SRCS = main.cpp Universe.cpp CelestialBody.cpp
OBJS = $(SRCS:.cpp=.o)

all: NBody NBody.a test lint

NBody: main.o NBody.a
    $(CXX) $(CXXFLAGS) -o $@ $^ $(SFML_LIBS)

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

%.o: %.cpp
    $(CXX) $(CXXFLAGS) -c $< -o $@

test: test.cpp NBody.a
    $(CXX) $(CXXFLAGS) -o $@ $^ $(SFML_LIBS) $(BOOST_LIBS)

lint:
    cpplint --filter=-legal/copyright *.cpp *.hpp

clean:
    rm -f *.o NBody NBody.a test

.PHONY: all clean lint test
```

main.cpp

```cpp
// copyright 2024 Sameera Sakinala

#include <iostream>
#include <fstream>
#include <cmath>
#include <chrono>
#include <iomanip>
#include <thread>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include "Universe.hpp"

const int SCREEN_WIDTH = 800;
const int SCREEN_HEIGHT = 800;

void playAudio(sf::Music& music) {
    music.play();
}

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: ./NBody T dt < universe_file" << std::endl;
        return 1;
    }

    sf::RenderWindow window(sf::VideoMode(SCREEN_WIDTH, SCREEN_HEIGHT), "N-Body Simulation");
    window.setFramerateLimit(60);

    sf::Texture backgroundTexture;
    if (!backgroundTexture.loadFromFile("starfield.jpg")) {
        std::cerr << "Failed to load background image" << std::endl;
        return 1;
    }
    sf::Sprite backgroundSprite(backgroundTexture);
    backgroundSprite.setScale(
        static_cast<float>(SCREEN_WIDTH) / backgroundTexture.getSize().x,
        static_cast<float>(SCREEN_HEIGHT) / backgroundTexture.getSize().y);

    sf::Music backgroundMusic;
    if (backgroundMusic.openFromFile("2001.wav")) {
        backgroundMusic.setLoop(true);
        std::thread audioThread(playAudio, std::ref(backgroundMusic));
        audioThread.detach();
    } else {
        std::cerr << "Failed to load background music" << std::endl;
    }

```

```
49      double T = std::stod(argv[1]);
50      double dt = std::stod(argv[2]);
51
52      NB::Universe universe;
53      std::cin >> universe;
54
55      std::vector<sf::Texture> textures(universe.size());
56      std::vector<sf::Sprite> sprites(universe.size());
57
58      for (size_t i = 0; i < universe.size(); ++i) {
59          textures[i].loadFromFile(universe[i].getFilename());
60          sprites[i].setTexture(textures[i]);
61          sprites[i].setOrigin(textures[i].getSize().x / 2.0f, textures[i].
        getSize().y / 2.0f);
62      }
63
64      sf::Font font;
65      if (!font.loadFromFile("arial.ttf")) {
66          std::cerr << "Failed to load font" << std::endl;
67          return 1;
68      }
69
70      sf::Text timeText;
71      timeText.setFont(font);
72      timeText.setCharacterSize(20);
73      timeText.setFillColor(sf::Color::White);
74      timeText.setPosition(10, 10);
75
76      auto start_time = std::chrono::high_resolution_clock::now();
77      double t = 0.0;
78      sf::Clock clock;
79
80      while (window.isOpen() && t < T) {
81          sf::Event event;
82          while (window.pollEvent(event)) {
83              if (event.type == sf::Event::Closed)
84                  window.close();
85          }
86
87          universe.step(dt);
88          t += dt;
89
90          window.clear();
91          window.draw(backgroundSprite);
92
93          for (size_t i = 0; i < universe.size(); ++i) {
94              double scaleFactor = SCREEN_WIDTH / (2 * universe.getRadius());
95              sf::Vector2<double> pos = universe[i].getPosition();
96              sprites[i].setPosition(
97                  (pos.x * scaleFactor) + SCREEN_WIDTH / 2,
98                  (pos.y * scaleFactor) + SCREEN_HEIGHT / 2);
```

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

        auto current_time = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> elapsed_time = current_time - start_time;
        timeText.setString("Elapsed time: " + std::to_string(elapsed_time.count()) + " s");
        window.draw(timeText);

        window.display();}

    std::cout << universe;

    std::ofstream newUniverseFile("new_universe.txt");
    newUniverseFile << universe;
    newUniverseFile.close();

    return 0;
}
```

Universe.hpp

```
// copyright 2024 Sameera Sakinala
#pragma once
#include <vector>
#include <memory>
#include "CelestialBody.hpp"
namespace NB {

class Universe {
 private:
    std::vector<std::unique_ptr<CelestialBody>> bodies;
    double radius;
    static constexpr double G = 6.67430e-11;

 public:
    Universe() = default;
    size_t size() const { return bodies.size(); }
    double getRadius() const { return radius; }
    CelestialBody& operator[](size_t index) { return *bodies[index]; }
    const CelestialBody& operator[](size_t index) const { return *bodies[index]; }
    void step(double dt);
    sf::Vector2<double> computeAcceleration(const CelestialBody& body) const;
    friend std::istream& operator>>(std::istream& is, Universe& universe);
    friend std::ostream& operator<<(std::ostream& os, const Universe& universe);
};
}  // namespace NB
```

Universe.cpp

```cpp
// copyright 2024 Sameera Sakinala
#include "Universe.hpp"
#include <cmath>
#include <iomanip>

namespace NB {

void Universe::step(double dt) {
    std::vector<sf::Vector2<double>> accelerations;
    for (const auto& body : bodies) {
        accelerations.push_back(computeAcceleration(*body));
    }

    for (size_t i = 0; i < bodies.size(); ++i) {
        auto& body = bodies[i];
        sf::Vector2<double> halfStepVelocity = body->getVelocity() + accelerations[i] * (dt / 2);
        sf::Vector2<double> newPosition = body->getPosition() + halfStepVelocity * dt;
        body->setPosition(newPosition.x, newPosition.y);
    }

    for (size_t i = 0; i < bodies.size(); ++i) {
        auto& body = bodies[i];
        sf::Vector2<double> newAcceleration = computeAcceleration(*body);
        sf::Vector2<double> avgAcceleration = (accelerations[i] + newAcceleration) / 2.0;
        sf::Vector2<double> newVelocity = body->getVelocity() + avgAcceleration * dt;
        body->setVelocity(newVelocity.x, newVelocity.y);
    }
}

sf::Vector2<double> Universe::computeAcceleration(const CelestialBody& body) const {
    sf::Vector2<double> acceleration(0, 0);
    for (const auto& otherBody : bodies) {
        if (otherBody.get() != &body) {
            sf::Vector2<double> delta = otherBody->getPosition() - body.getPosition();
            double distance = std::sqrt(delta.x * delta.x + delta.y * delta.y);
            if (distance > 0) {
                double force = G * body.getMass() * otherBody->getMass() / (distance * distance);
                acceleration += (delta / distance) * (force / body.getMass());
            }
        }
    }
```

```
    return acceleration;
}

std::istream& operator>>(std::istream& is, Universe& universe) {
    int n;
    is >> n >> universe.radius;
    universe.bodies.clear();
    for (int i = 0; i < n; ++i) {
        auto body = std::make_unique<CelestialBody>();
        is >> *body;
        universe.bodies.push_back(std::move(body));
    }
    return is;
}

std::ostream& operator<<(std::ostream& os, const Universe& universe) {
    os << universe.size() << "\n";
    os << std::scientific << std::setprecision(4) << universe.getRadius() << "
    \n";
    for (const auto& body : universe.bodies) {
        os << *body << "\n";
    }
    return os;
}

}  // namespace NB
```

CelestialBody.hpp

```
// copyright 2024 Sameera Sakinala
#pragma once
#include <string>
#include <SFML/System/Vector2.hpp>

namespace NB {

class CelestialBody {
 private:
    sf::Vector2<double> position;
    sf::Vector2<double> velocity;
    double mass;
    std::string filename;

 public:
    CelestialBody() = default;
    sf::Vector2<double> getPosition() const { return position; }
    sf::Vector2<double> getVelocity() const { return velocity; }
    double getMass() const { return mass; }
    std::string getFilename() const { return filename; }
    void setPosition(double x, double y) { position = {x, y}; }
    void setVelocity(double x, double y) { velocity = {x, y}; }
```

```
23
24      friend std::istream& operator>>(std::istream& is, CelestialBody& body);
25      friend std::ostream& operator<<
26      (std::ostream& os, const CelestialBody& body);
27  };
28
29  }  // namespace NB
```

CelestialBody.cpp

```
1   // copyright 2024 Sameera Sakinala
2   #include "CelestialBody.hpp"
3   #include <iomanip>
4
5   namespace NB {
6
7   std::istream& operator>>(std::istream& is, CelestialBody& body) {
8       is >> body.position.x >> body.position.y
9          >> body.velocity.x >> body.velocity.y
10         >> body.mass >> body.filename;
11      return is;
12  }
13
14  std::ostream& operator<<(std::ostream& os, const CelestialBody& body) {
15      os << std::scientific << std::setprecision(4)
16         << body.position.x << " " << body.position.y << " "
17         << body.velocity.x << " " << body.velocity.y << " "
18         << body.mass << " " << body.filename;
19      return os;
20  }
21
22  }  // namespace NB
```

test.cpp

```
// copyright 2024 Sameera Sakinala

#define BOOST_TEST_MODULE GalaxyModelTests
#include <sstream>
#include <cmath>
#include <boost/test/included/unit_test.hpp>
#include "Universe.hpp"
#include "CelestialBody.hpp"


// Constant for gravitational acceleration
const double G = 6.67430e-11;

BOOST_AUTO_TEST_SUITE(GalaxyTests)

// Test case for galaxy construction and I/O operations
BOOST_AUTO_TEST_CASE(GalaxyConstructionAndIO) {
    NB::Universe galaxyModel;
    std::stringstream dataStream("2 1.0000e+06\n"
    "1.0000e+00 2.0000e+00 3.0000e+00 4.0000e+00 5.0000e+00 earth.gif\n"
    "6.0000e+00 7.0000e+00 8.0000e+00 9.0000e+00 1.0000e+01 mars.gif");
    dataStream >> galaxyModel;

    BOOST_CHECK_EQUAL(galaxyModel.size(), 2);
    BOOST_CHECK_CLOSE(galaxyModel.getRadius(), 1.0000e+06, 0.001);

    std::stringstream outputStream;
    outputStream << galaxyModel;

    std::string expectedOutput = "2\n1.0000e+06\n"
    "1.0000e+00 2.0000e+00 3.0000e+00 4.0000e+00 5.0000e+00 earth.gif\n"
    "6.0000e+00 7.0000e+00 8.0000e+00 9.0000e+00 1.0000e+01 mars.gif\n";

    BOOST_CHECK_EQUAL(outputStream.str(), expectedOutput);
}

// Test case for galaxy index operator
BOOST_AUTO_TEST_CASE(GalaxyIndexOperator) {
    NB::Universe galaxyModel;
    std::stringstream dataStream("2 1.0000e+06\n"
    "1.0000e+00 2.0000e+00 3.0000e+00 4.0000e+00 5.0000e+00 earth.gif\n"
    "6.0000e+00 7.0000e+00 8.0000e+00 9.0000e+00 1.0000e+01 mars.gif");
    dataStream >> galaxyModel;

    BOOST_CHECK_CLOSE(galaxyModel[0].getPosition().x, 1.0000e+00, 0.001);
    BOOST_CHECK_CLOSE(galaxyModel[1].getPosition().y, 7.0000e+00, 0.001);
}

// New test case for computing acceleration
```

```
50 BOOST_AUTO_TEST_CASE(ComputeAccelerationTest) {
51     NB::Universe universe;
52     std::stringstream dataStream("2 1.0000e+11\n"
53     "0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.9890e+30 sun.gif\n"
54     "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif");
55     dataStream >> universe;
56 
57     sf::Vector2<double> acc = universe.computeAcceleration(universe[1]);
58     // Expected acceleration of Earth towards Sun
59     double expectedAcc = G * 1.9890e+30 / (1.4960e+11 * 1.4960e+11);
60     BOOST_CHECK_CLOSE(acc.x, -expectedAcc, 0.1);
61     BOOST_CHECK_SMALL(acc.y, 1e-10);
62 }
63 
64 // New test case for step method
65 BOOST_AUTO_TEST_CASE(StepMethodTest) {
66     NB::Universe universe;
67     std::stringstream dataStream("2 1.0000e+11\n"
68     "0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.9890e+30 sun.gif\n"
69     "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif");
70     dataStream >> universe;
71 
72     double dt = 86400;   // One day in seconds
73     universe.step(dt);
74 
75     // Check if Earth's position has changed
76     BOOST_CHECK_GT(universe[1].getPosition().y, 0);
77     // Check if Earth's velocity has changed
78     BOOST_CHECK_LT(universe[1].getVelocity().x, 2.9800e+04);
79 }
80 
81 
82 // New test case for precision (to catch subtle errors)
83 BOOST_AUTO_TEST_CASE(PrecisionTest) {
84     NB::Universe universe;
85     std::stringstream dataStream("2 1.0000e+11\n"
86     "0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.9890e+30 sun.gif\n"
87     "1.4960e+11 0.0000e+00 0.0000e+00 2.9800e+04 5.9740e+24 earth.gif");
88     dataStream >> universe;
89 
90     double initialPosition = universe[1].getPosition().x;
91     for (int i = 0; i < 365; ++i) {
92         universe.step(86400);   // Simulate for one year
93     }
94     double finalPosition = universe[1].getPosition().x;
95     // Check if Earth returns to its initial position after one year (within
    1%)
96     BOOST_CHECK_CLOSE(initialPosition, finalPosition, 1.0);
97 }
98 BOOST_AUTO_TEST_SUITE_END()
```

4.7 Result

Figure 4

5 PS4: Sokoban

5.1 Discussion

In this project, I implemented a Sokoban game using C++ and the SFML graphics library. Sokoban is a classic puzzle game where the player controls a character trying to push crates onto storage locations. The goal is to move crates around a grid while navigating obstacles to complete the puzzle.
The game features a dynamic environment where the player can interact with the game world by moving the character and crates, undoing and redoing moves, and resetting the game when necessary. The game's user interface shows the grid, player, crates, and storage areas, and updates the player's movements in real-time. Additionally, the application tracks and displays the elapsed time during gameplay and alerts the player when they win.

5.2 Key Algorithms, Data Structures and OO Designs

- **Algorithms**
  - **Game Initialization:**
    - The game initializes by reading a level file containing the layout of the grid. This layout is used to populate a 2D grid representing the game board.
    - Textures and sounds are preloaded to ensure efficient rendering and user feedback during gameplay.
  - **Player Movement Logic:**
    - Player movement is governed by keyboard input (W/A/S/D or arrow keys).
    - Before moving, the algorithm checks:
      - If the destination tile is walkable (`.` or `a`).
      - If it is a box (`A` or `*`), whether the box can be pushed further in the same direction.
  - **Box Push Logic:**
    - If the player pushes a box:
      - The box is moved to the adjacent tile in the same direction.
      - The current tile is updated to reflect whether it is now empty or a storage location.
    - This is implemented using a helper function `doPushBox`.
  - **Game State Management:**
    - States (grid and player position) are stored in stacks to support **undo/redo** functionality.
    - When a move is made:
      - The current state is pushed onto an undo stack.
      - The redo stack is cleared to prevent inconsistent states.
    - Undo/redo operations pop/push from respective stacks to revert or reapply changes.
  - **Win Condition:**
    - The game checks if all boxes (`A`) are placed on storage locations (`a`) by iterating through the grid. If true, the game is considered won.
  - **Rendering and User Interface:**

    - The game board is rendered as a grid of sprites, with specific textures for each tile type.
    - A timer and win message are displayed using SFML's text-rendering capabilities.

- **Data Structures**
  - **2D Grid (`std::vector<std::vector<char>>`):**
    - Represents the game board where each character encodes the tile type:
      - `'#'`: Wall.
      - `'.'`: Empty floor.
      - `'a'`: Storage location.
      - `'A'`: Box.
      - `'@'`: Player on floor.
      - `'+'`: Player on storage.
    - This grid is the core representation of the game state.
  - **State Management (Stacks):**
    - **Undo Stack (`std::stack<std::vector<std::vector<char>>>`):** Stores previous states of the grid for undo operations.
    - **Redo Stack (`std::stack<std::vector<std::vector<char>>>`):** Stores future states for redo functionality.
    - **Player Position Stack (`std::stack<sf::Vector2u>`):** Tracks player positions for undo/redo actions.
  - **Texture Map (`std::map<char, sf::Texture>`):** Maps tile types (e.g., `'#'`, `'A'`, `'@'`) to corresponding textures for rendering.
  - **Direction Enum (`Direction`):** Encodes possible movement directions: Up, Down, Left, Right.

- **Object-Oriented Used**
  - **Encapsulation:**
    - The `Sokoban` class encapsulates the game logic, data, and rendering functionality. This makes the code modular and easier to manage.
    - Private members (e.g., `m_grid`, `m_playerPosition`) prevent external modification, ensuring state consistency.
  - **Abstraction:**
    - High-level functions like `movePlayer` and `reset` abstract away the details of how player movement or game reset is implemented.
    - Users of the `Sokoban` class interact with these simple interfaces without needing to know the internal logic.
  - **Inheritance:**
    - While not explicitly shown, `Sokoban` inherits from `sf::Drawable` to integrate seamlessly with SFML's rendering pipeline.
  - **Polymorphism:**
    - The `draw` function leverages polymorphism through `sf::Drawable`, enabling the `Sokoban` object to be drawn directly via the SFML rendering framework.
  - **Modularity:**
    - Game logic (`Sokoban` class) and UI handling (`main` function) are separated for clarity and maintainability.

5.3 What I Accomplished

- Developed a fully functional Sokoban game in C++ with SFML.
- Implemented player movement, crate pushing, and level reset features.
- Integrated undo and redo functionality for a better user experience.
- Designed the user interface with a timer and a win condition display.
- Added sound effects to enhance the gaming experience, including sounds for movement and victory.
- Managed game state, including saving and loading the level from a file and providing the ability to reset the level.

5.4 What I Learned

- **Game Design and State Management :** I gained a deeper understanding of how to manage game states effectively, especially when it comes to implementing undo and redo features. Using stacks to store previous game states was a helpful way to provide smooth gameplay for the player.
- **SFML Graphics Handling :** I learned how to use SFML for game development, specifically how to load and display textures, handle window events, and use sprite rendering to create a visually interactive game.
- **Game Flow and User Interaction :** I refined my understanding of how players interact with games and how to handle user input in real-time. I also learned how to synchronize the game mechanics (movement, crate pushing) with visual feedback on the screen.
- **File Handling :** By implementing file reading and writing features, I learned how to store and load game levels dynamically, which is essential for creating reusable and scalable game content.

5.5 Challenges

- **Texture and Asset Management :** Initially, loading and managing the game's assets (textures and sounds) posed a challenge. Ensuring that the correct assets were used for each tile and character required careful attention to file paths and texture assignments.
- **Undo/Redo Mechanism :** Implementing the undo and redo mechanism was tricky. I had to make sure that not only the game grid was reverted but also the player's position, ensuring the game restored to the correct state after each action.
- **Player and Box Interaction :** Handling the interaction between the player and crates was complex, as it required checking if a crate could be pushed and if the destination tile was valid. This involved carefully checking neighboring tiles and ensuring that the player could only push crates in valid directions.
- **Sound Effects Timing :** Timing the sound effects appropriately (e.g., only playing the victory sound once the player has completed the level) required synchronizing the gameplay with the sound cues.

5.6 Code Implementation

Makefile

```
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -Werror -pedantic
SFML_LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
BOOST_LIBS = -lboost_unit_test_framework

all: Sokoban Sokoban.a test

Sokoban: main.o Sokoban.o
    $(CXX) $(CXXFLAGS) -o $@ $^ $(SFML_LIBS)

test: test.o Sokoban.o
    $(CXX) $(CXXFLAGS) -o $@ $^ $(SFML_LIBS) $(BOOST_LIBS)

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

main.o: main.cpp Sokoban.hpp
    $(CXX) $(CXXFLAGS) -c $<

Sokoban.o: Sokoban.cpp Sokoban.hpp
    $(CXX) $(CXXFLAGS) -c $<

test.o: test.cpp Sokoban.hpp
    $(CXX) $(CXXFLAGS) -c $<

lint:
    cpplint *.cpp *.hpp

clean:
    rm -f *.o Sokoban Sokoban.a test

.PHONY: all lint clean test
```

main.cpp

```cpp
// Copyright 2024 Sameera Sakinala
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include "Sokoban.hpp"

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

    std::ifstream levelFile(argv[1]);
    if (!levelFile) {
        std::cerr << "Error: Could not open level file." << std::endl;
        return 1;
    }

    SB::Sokoban game;
    levelFile >> game;

    sf::RenderWindow window(sf::VideoMode(
                                game.width() * SB::Sokoban::TILE_SIZE,
                                game.height() * SB::Sokoban::TILE_SIZE),
                                "Sokoban");

    sf::Clock clock;  // Clock to track elapsed time

    // Text to display elapsed time
    sf::Font font;
    if (!font.loadFromFile("arial.ttf")) {
        std::cerr << "Error: Could not load font." << std::endl;
        return 1;
    }

    sf::Text timerText;
    timerText.setFont(font);
    timerText.setCharacterSize(24);
    timerText.setFillColor(sf::Color::White);
    timerText.setPosition(10, 10);  // Display timer at the top-left corner

    sf::Text winText;
    winText.setFont(font);
    winText.setString("You Won!");
    winText.setCharacterSize(48);
    winText.setFillColor(sf::Color::Green);
```

```
    winText.setPosition(
        (window.getSize().x - winText.getLocalBounds().width) / 2,
        (window.getSize().y - winText.getLocalBounds().height) / 2);

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            } else if (event.type == sf::Event::KeyPressed) {
                switch (event.key.code) {
                    case sf::Keyboard::W:
                    case sf::Keyboard::Up:
                        game.movePlayer(SB::Direction::Up);
                        break;
                    case sf::Keyboard::S:
                    case sf::Keyboard::Down:
                        game.movePlayer(SB::Direction::Down);
                        break;
                    case sf::Keyboard::A:
                    case sf::Keyboard::Left:
                        game.movePlayer(SB::Direction::Left);
                        break;
                    case sf::Keyboard::D:
                    case sf::Keyboard::Right:
                        game.movePlayer(SB::Direction::Right);
                        break;
                    case sf::Keyboard::R:
                        game.reset();
                        clock.restart();  // Reset the timer
                        break;
                    case sf::Keyboard::Z:
                        if (game.canUndo()) game.undo();
                        break;
                    case sf::Keyboard::Y:
                        if (game.canRedo()) game.redo();
                        break;
                    default:
                        break;
                }
            }
        }

        // Format elapsed time in MM:SS
        sf::Time elapsed = clock.getElapsedTime();
        int minutes = static_cast<int>(elapsed.asSeconds()) / 60;
        int seconds = static_cast<int>(elapsed.asSeconds()) % 60;
        std::ostringstream timeStream;
        timeStream << std::setw(2) << std::setfill('0') << minutes << ":"
                   << std::setw(2) << std::setfill('0') << seconds;
        timerText.setString(timeStream.str());
```

```

        window.clear();
        window.draw(game);
        window.draw(timerText);  // Draw the elapsed time

        if (game.isWon()) {
            window.draw(winText);
        }

        window.display();
    }

    return 0;
}
```

Sokoban.hpp

```
// Copyright 2024 Sameera Sakinala
#pragma once

#include <vector>
#include <map>
#include <string>
#include <stack>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

namespace SB {

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

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

    Sokoban();
    unsigned int width() const;
    unsigned int height() const;
    sf::Vector2u playerLoc() const;
    bool isWon() const;
    void movePlayer(Direction dir);
    void reset();
    char getTile(unsigned int x, unsigned int y) const;
    void undo();
    bool canUndo() const;
    void redo();
    bool canRedo() const;

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

```

```
35  protected:
36      void draw(sf::RenderTarget& target, sf::RenderStates states) const
        override;
37
38  private:
39      unsigned int m_width;
40      unsigned int m_height;
41      sf::Vector2u m_playerPosition;
42      std::vector<std::vector<char>> m_grid;
43      std::vector<std::vector<char>> m_initialGrid;
44      std::map<char, sf::Texture> m_textures;
45      Direction m_playerDirection;
46      sf::SoundBuffer m_moveSoundBuffer;
47      sf::Sound m_moveSound;
48      sf::SoundBuffer m_victorySoundBuffer;
49      sf::Sound m_victorySound;
50      std::stack<std::vector<std::vector<char>>> m_undoStack;
51      std::stack<sf::Vector2u> m_playerPositionStack;
52      std::stack<std::vector<std::vector<char>>> m_redoStack;
53      std::stack<sf::Vector2u> m_redoPlayerPositionStack;
54
55      void loadTextures();
56      bool canMove(int x, int y) const;
57      bool canPushBox(int x, int y, Direction dir) const;
58      void doPushBox(int x, int y, Direction dir);
59      bool isStorage(char tile) const;
60      bool isBoxOnStorage(char tile) const;
61      void saveState();
62      void playMoveSound();
63      void playVictorySound();
64  };
65
66  std::istream& operator>>(std::istream& in, Sokoban& s);
67  std::ostream& operator<<(std::ostream& out, const Sokoban& s);
68
69  }  // namespace SB
```

Sokoban.cpp

```cpp
// Copyright 2024 Sameera Sakinala
#include "Sokoban.hpp"
#include <iostream>
#include <fstream>
#include <utility>

namespace SB {

Sokoban::Sokoban() : m_width(0), m_height(0),
m_playerPosition(0, 0), m_playerDirection(Direction::Down) {
    loadTextures();
    if (!m_moveSoundBuffer.loadFromFile("move.wav")) {
        std::cerr << "Failed to load move sound" << std::endl;
    }
    m_moveSound.setBuffer(m_moveSoundBuffer);

    if (!m_victorySoundBuffer.loadFromFile("victory.wav")) {
        std::cerr << "Failed to load victory sound" << std::endl;
    }
    m_victorySound.setBuffer(m_victorySoundBuffer);
}

void Sokoban::loadTextures() {
    std::vector<std::pair<char, std::string>> texturePaths = {
        {'#', "block_06.png"},
        {'A', "crate_03.png"},
        {'.', "ground_01.png"},
        {'a', "ground_04.png"},
        {'*', "crate_03.png"},
        {'+', "player_05.png"},
    };

    for (const auto& [key, path] : texturePaths) {
        sf::Texture texture;
        if (!texture.loadFromFile(path)) {
            std::cerr << "Failed to load texture: " << path << std::endl;
        } else {
            m_textures[key] = texture;
        }
    }

    std::vector<std::pair<Direction, std::string>> playerTextures = {
        {Direction::Up, "player_08.png"},
        {Direction::Down, "player_05.png"},
        {Direction::Left, "player_20.png"},
        {Direction::Right, "player_17.png"}
    };

    for (const auto& [dir, path] : playerTextures) {
```

```
50          sf::Texture texture;
51          if (!texture.loadFromFile(path)) {
52              std::cerr << "Failed to load player texture: " << path << std::
    endl;
53          } else {
54              m_textures[static_cast<char>(dir)] = texture;
55          }
56      }
57  }
58
59  unsigned int Sokoban::width() const { return m_width; }
60  unsigned int Sokoban::height() const { return m_height; }
61  sf::Vector2u Sokoban::playerLoc() const { return m_playerPosition; }
62
63  bool Sokoban::isWon() const {
64      for (unsigned int y = 0; y < m_height; ++y) {
65          for (unsigned int x = 0; x < m_width; ++x) {
66              if (m_grid[y][x] == 'A' || m_grid[y][x] == 'a') {
67                  return false;
68              }
69          }
70      }
71      return true;
72  }
73
74  void Sokoban::movePlayer(Direction dir) {
75      saveState();
76      // Clear redo stacks when a new move is made
77      while (!m_redoStack.empty()) m_redoStack.pop();
78      while (!m_redoPlayerPositionStack.empty()) m_redoPlayerPositionStack.pop()
    ;
79
80      m_playerDirection = dir;
81      int newX = m_playerPosition.x;
82      int newY = m_playerPosition.y;
83
84      switch (dir) {
85          case Direction::Up: newY--; break;
86          case Direction::Down: newY++; break;
87          case Direction::Left: newX--; break;
88          case Direction::Right: newX++; break;
89      }
90
91      if (canMove(newX, newY)) {
92          char currentTile = m_grid[m_playerPosition.y][m_playerPosition.x];
93          m_grid[m_playerPosition.y][m_playerPosition.x] =
94              (currentTile == '@' || currentTile == '+') ?
95              (isStorage(m_initialGrid[m_playerPosition.y]
96              [m_playerPosition.x]) ? 'a' : '.') :
97              currentTile;
98          if (m_grid[newY][newX] == 'A' || m_grid[newY][newX] == '*') {
```

```
99              doPushBox(newX, newY, dir);
100         }
101         m_playerPosition = {static_cast<unsigned int>(newX),
102         static_cast<unsigned int>(newY)};
103         char newTile = m_grid[newY][newX];
104         m_grid[newY][newX] = isStorage(newTile) ? '+' : '@';
105         playMoveSound();
106 
107         if (isWon()) {
108             playVictorySound();
109         }
110     }
111 }
112 
113 void Sokoban::reset() {
114     while (!m_undoStack.empty()) m_undoStack.pop();
115     while (!m_playerPositionStack.empty()) m_playerPositionStack.pop();
116     while (!m_redoStack.empty()) m_redoStack.pop();
117     while (!m_redoPlayerPositionStack.empty()) m_redoPlayerPositionStack.pop()
    ;
118     m_grid = m_initialGrid;
119     for (unsigned int y = 0; y < m_height; ++y) {
120         for (unsigned int x = 0; x < m_width; ++x) {
121             if (m_grid[y][x] == '@' || m_grid[y][x] == '+') {
122                 m_playerPosition = {x, y};
123                 m_playerDirection = Direction::Down;
124                 return;
125             }
126         }
127     }
128 }
129 
130 char Sokoban::getTile(unsigned int x, unsigned int y) const {
131     if (x < m_width && y < m_height) {
132         return m_grid[y][x];
133     }
134     return ' ';
135 }
136 
137 void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
138     for (unsigned int y = 0; y < m_height; ++y) {
139         for (unsigned int x = 0; x < m_width; ++x) {
140             char tile = m_grid[y][x];
141             if (tile == '@' || tile == '+') {
142                 sf::Sprite floorSprite(m_textures.at(tile == '+' ? 'a' : '.'))
    ;
143                 floorSprite.setPosition(x * TILE_SIZE, y * TILE_SIZE);
144                 target.draw(floorSprite, states);
145                 sf::Sprite playerSprite
146                 (m_textures.at(static_cast<char>(m_playerDirection)));
147                 playerSprite.setPosition(x * TILE_SIZE, y * TILE_SIZE);
```

```
148                 target.draw(playerSprite, states);
149             } else if (m_textures.count(tile) > 0) {
150                 sf::Sprite sprite(m_textures.at(tile));
151                 sprite.setPosition(x * TILE_SIZE, y * TILE_SIZE);
152                 target.draw(sprite, states);
153             }
154         }
155     }
156 }
157
158 bool Sokoban::canMove(int x, int y) const {
159     if (x < 0 || x >= static_cast<int>(m_width)
160     || y < 0 || y >= static_cast<int>(m_height)) {
161         return false;
162     }
163
164     char cell = m_grid[y][x];
165     if (cell == '#') return false;
166     if (cell == 'A' || cell == '*') {
167         return canPushBox(x, y, m_playerDirection);
168     }
169     return true;
170 }
171
172 bool Sokoban::canPushBox(int x, int y, Direction dir) const {
173     int newX = x, newY = y;
174     switch (dir) {
175         case Direction::Up: newY--; break;
176         case Direction::Down: newY++; break;
177         case Direction::Left: newX--; break;
178         case Direction::Right: newX++; break;
179     }
180
181     if (newX < 0 || newX >= static_cast<int>(m_width)
182     || newY < 0 || newY >= static_cast<int>(m_height)) {
183         return false;
184     }
185
186     char nextCell = m_grid[newY][newX];
187     return (nextCell == '.' || nextCell == 'a');
188 }
189
190 void Sokoban::doPushBox(int x, int y, Direction dir) {
191     int newX = x, newY = y;
192     switch (dir) {
193         case Direction::Up: newY--; break;
194         case Direction::Down: newY++; break;
195         case Direction::Left: newX--; break;
196         case Direction::Right: newX++; break;
197     }
198
```

```
199     char nextCell = m_grid[newY][newX];
200     m_grid[newY][newX] = isStorage(nextCell) ? '*' : 'A';
201     m_grid[y][x] = isStorage(m_initialGrid[y][x]) ? 'a' : '.';
202 }
203 
204 bool Sokoban::isStorage(char tile) const {
205     return tile == 'a' || tile == '*' || tile == '+';
206 }
207 
208 bool Sokoban::isBoxOnStorage(char tile) const {
209     return tile == '*';
210 }
211 
212 void Sokoban::saveState() {
213     m_undoStack.push(m_grid);
214     m_playerPositionStack.push(m_playerPosition);
215 }
216 
217 void Sokoban::undo() {
218     if (!m_undoStack.empty()) {
219         m_redoStack.push(m_grid);
220         m_redoPlayerPositionStack.push(m_playerPosition);
221 
222         m_grid = m_undoStack.top();
223         m_undoStack.pop();
224         m_playerPosition = m_playerPositionStack.top();
225         m_playerPositionStack.pop();
226     }
227 }
228 
229 void Sokoban::redo() {
230     if (!m_redoStack.empty()) {
231         saveState();
232         m_grid = m_redoStack.top();
233         m_redoStack.pop();
234         m_playerPosition = m_redoPlayerPositionStack.top();
235         m_redoPlayerPositionStack.pop();
236     }
237 }
238 
239 bool Sokoban::canUndo() const {
240     return !m_undoStack.empty();
241 }
242 
243 bool Sokoban::canRedo() const {
244     return !m_redoStack.empty();
245 }
246 
247 void Sokoban::playMoveSound() {
248     m_moveSound.play();
249 }
```

```
250
251 void Sokoban::playVictorySound() {
252     m_victorySound.play();
253 }
254
255 std::istream& operator>>(std::istream& in, Sokoban& s) {
256     in >> s.m_height >> s.m_width;
257     s.m_grid.resize(s.m_height, std::vector<char>(s.m_width));
258     s.m_initialGrid.resize(s.m_height, std::vector<char>(s.m_width));
259     for (unsigned int y = 0; y < s.m_height; ++y) {
260         for (unsigned int x = 0; x < s.m_width; ++x) {
261             in >> s.m_grid[y][x];
262             s.m_initialGrid[y][x] = s.m_grid[y][x];
263             if (s.m_grid[y][x] == '@' || s.m_grid[y][x] == '+') {
264                 s.m_playerPosition = {x, y};
265             }
266         }
267     }
268     return in;
269 }
270
271 std::ostream& operator<<(std::ostream& out, const Sokoban& s) {
272     out << s.height() << " " << s.width() << "\n";
273     for (unsigned int y = 0; y < s.height(); ++y) {
274         for (unsigned int x = 0; x < s.width(); ++x) {
275             out << s.getTile(x, y);
276         }
277         out << "\n";
278     }
279     return out;
280 }
281 }  // namespace SB
```

test.cpp

```
1  // copyrightr 2024 Sakinala Sameera
2  #include <sstream>
3  #include <ostream>
4  #include <iostream>
5  #include <SFML/System/Vector2.hpp>
6
7  namespace sf {
8  template <typename T>
9  std::ostream& operator<<(std::ostream& os, const Vector2<T>& vec) {
10 return os << "(" << vec.x << ", " << vec.y << ")";
11 }
12 }
13
14 #define BOOST_TEST_MODULE SokobanTests
15 #include <boost/test/included/unit_test.hpp>
16 #include "Sokoban.hpp"
```

```
17
18 // Fixture setup for consistent game state initialization across tests
19 struct GameFixture {
20     SB::Sokoban game;
21     GameFixture() {
22         std::stringstream ss;
23         ss << "5 5\n"
24            << "#####\n"
25            << "#@.A#\n"
26            << "#.a.#\n"
27            << "#...#\n"
28            << "#####\n";
29         ss >> game;
30     }
31 };
32
33 BOOST_FIXTURE_TEST_SUITE(SokobanTests, GameFixture)
34
35 BOOST_AUTO_TEST_CASE(BasicMovement) {
36     auto initialPos = game.playerLoc();
37     std::cout << "Initial position: " << initialPos << std::endl;
38
39     game.movePlayer(SB::Direction::Right);
40     auto newPos = game.playerLoc();
41     std::cout << "Position after moving right: " << newPos << std::endl;
42
43     BOOST_TEST(newPos.x == initialPos.x + 1);
44     BOOST_TEST(newPos.y == initialPos.y);
45 }
46
47 BOOST_AUTO_TEST_CASE(WallCollision) {
48     auto initialPos = game.playerLoc();
49     game.movePlayer(SB::Direction::Up);
50     BOOST_TEST(game.playerLoc() == initialPos);
51 }
52
53 BOOST_AUTO_TEST_CASE(BoxPushing) {
54     auto initialPos = game.playerLoc();
55     game.movePlayer(SB::Direction::Right);
56     auto posAfterOneMove = game.playerLoc();
57     std::cout << "Position after one move right: "
58     << posAfterOneMove << std::endl;
59
60     game.movePlayer(SB::Direction::Right);
61     auto posAfterTwoMoves = game.playerLoc();
62     std::cout << "Position after two moves right: "
63     << posAfterTwoMoves << std::endl;
64
65     BOOST_TEST(posAfterTwoMoves.x == initialPos.x + 2);
66     BOOST_TEST(posAfterTwoMoves.y == initialPos.y);
67     BOOST_CHECK_EQUAL(game.getTile(3, 1), 'A');
```

```
68  }
69
70  BOOST_AUTO_TEST_CASE(WinCondition) {
71      std::cout << "Initial game state:" << std::endl << game << std::endl;
72
73      game.movePlayer(SB::Direction::Right);
74      game.movePlayer(SB::Direction::Right);
75      game.movePlayer(SB::Direction::Down);
76      BOOST_CHECK(!game.isWon());
77      std::cout << "Game state before final move:"
78      << std::endl << game << std::endl;
79
80      game.movePlayer(SB::Direction::Right);
81      std::cout << "Game state after final move:"
82      << std::endl << game << std::endl;
83      std::cout << "Is won: " << (game.isWon() ? "true" : "false") << std::endl;
84
85      BOOST_CHECK_MESSAGE(game.isWon(),
86      "Game should be won when all boxes are on storage locations");
87  }
88
89  BOOST_AUTO_TEST_CASE(ResetGame) {
90      auto initialPos = game.playerLoc();
91      game.movePlayer(SB::Direction::Right);
92      game.movePlayer(SB::Direction::Right);
93      game.reset();
94      BOOST_TEST(game.playerLoc() == initialPos);
95  }
96
97  BOOST_AUTO_TEST_CASE(TestSwap) {
98      std::stringstream ss;
99      ss << "5 5\n"
100        << "#####\n"
101        << "#@aA#\n"
102        << "#...#\n"
103        << "#...#\n"
104        << "#####\n";
105     SB::Sokoban game;
106     ss >> game;
107
108     game.movePlayer(SB::Direction::Up);
109     game.movePlayer(SB::Direction::Down);
110     game.movePlayer(SB::Direction::Right);
111     game.movePlayer(SB::Direction::Right);
112     game.movePlayer(SB::Direction::Up);
113     game.movePlayer(SB::Direction::Left);
114
115     BOOST_CHECK(game.isWon());
116 }
117 BOOST_AUTO_TEST_SUITE_END()
```

5.7 Result

Figure 5

6 PS5 : DNA String Alignment

6.1 Discussion

This project implements the Edit Distance (Levenshtein distance) algorithm for DNA sequence alignment. The main objective is to compute the minimum number of operations (insertions, deletions, or substitutions) required to transform one DNA string into another. This is achieved through dynamic programming for optimal performance and accuracy.

6.2 Key Algorithms, Data Structures and OO Designs

Dynamic Programming Algorithm for Edit Distance:

- **Edit Distance Algorithm :** The primary algorithm used in the project is a dynamic programming solution to compute the edit distance between two strings x and y. The edit distance measures the minimum number of operations (insertions, deletions, substitutions) required to transform one string into another. The algorithm is implemented using a two-dimensional table (opt) where opt[i][j] holds the minimum number of operations required to transform the first i characters of string x into the first j characters of string y.
  Initialization : The base cases are initialized with values representing the cost of converting a string to an empty string (insertions or deletions). For example, transforming the first i characters of x into an empty string requires 2 * i deletions.
  Filling the DP Table : The table is filled in a bottom-up manner. For each pair of characters from x and y, the algorithm computes the minimum cost of three operations: substitution (if characters differ), insertion, or deletion. The recursive relationship used here is:

```
1 opt[i][j] = min3(opt[i + 1][j + 1] + penalty(x[i], y[j]),
2             opt[i + 1][j] + 2, opt[i][j + 1] + 2);
```

  penalty(x[i], y[j]) is 0 if the characters are the same and 1 otherwise.
  The function min3 computes the minimum of three values, ensuring the least-cost operation is chosen.

  Data Structures :

- **2D Vector (opt) :** The opt matrix is a two-dimensional vector `std::vector<std::vector<int>>` that stores the intermediate results for the dynamic programming approach. It is sized to hold the lengths of the strings x and y (i.e., `opt[x.length() + 1][y.length() + 1]`). This ensures that the table can store values for all prefix combinations of the two strings. **String Storage :** The strings x and y are stored as std::string objects. These are used as inputs to the EDistance class, and their characters are compared throughout the algorithm.

- **Time and Memory Tracking : Time Tracking :** The execution time is tracked using `std::chrono::high_resolution_clock::now()` at the start and end of the main computation.
  The duration is then computed and output, giving insight into the performance of the algorithm

  .

```
auto start = std::chrono::high\textunderscore resolution\textunderscore
    clock::now();
auto end = std::chrono::high\textunderscore resolution
\textunderscore clock::now();
std::chrono::duration<double> diff = end - start;
```

- **Memory Usage :** Memory usage is tracked using the getCurrentMemoryUsage() function, which queries the system's memory usage through getrusage(RUSAGE_SELF, &usage). This helps monitor the memory footprint of the algorithm.

```
double initialMemory = getCurrentMemoryUsage();
double finalMemory = getCurrentMemoryUsage();
```

- **Key Functions : Penalty Function :** The penalty function computes the cost of substituting one character for another. It returns 0 if the characters are the same and 1 if they differ. **Alignment :** The alignment function reconstructs the sequence of operations that transform string x into string y based on the opt table. This is useful for visualizing the transformations between the two strings.

- **Object-Oriented Design :** The design of the project follows an Object-Oriented approach. +e dynamic programming table, and providing the final alignment. This class structure makes the code modular and easily extendable.

6.3 What I Accomplished

- Implemented the Edit Distance algorithm with dynamic programming.
- Designed a backtracking function to generate DNA sequence alignments.
- Created a robust testing suite to verify correctness and edge cases.
- Evaluated performance metrics, including execution time and memory usage.
- Documented results for various input sizes, showcasing the algorithm's scalability and limitations.

6.4 What I Learned

- **Dynamic Programming :** Learned how to decompose complex problems into overlapping subproblems. Gained experience in designing and optimizing DP solutions.
- **Algorithm Analysis :** Improved understanding of time and space complexity. Gained skills in measuring and interpreting performance metrics.
- **Memory Optimization :** Learned about trade-offs between memory usage and computational efficiency. Explored tools like Valgrind and Massif Visualizer to debug memory issues.
- **Problem-Solving Skills :** Enhanced ability to debug, test, and refine algorithms iteratively.

6.5 Challenges

- **Memory Usage :** The quadratic space requirement of the DP matrix caused crashes for large DNA sequences (¿50,000 characters). Attempted space optimization techniques, but they introduced additional complexity.
- **Performance Scaling :** Execution time scaled poorly for larger inputs, limiting the program's practicality. Explored optimization strategies, including iterative DP and memory-efficient implementations, but with limited success.
- **Alignment Validation :** Ensuring the correctness of generated alignments required extensive testing. Addressed edge cases, such as empty strings and highly divergent sequences.
- **Tool Familiarity :** Initial difficulty in using memory profiling tools like Valgrind. Overcame challenges through extensive documentation and trial-and-error experimentation.

6.6 Code Implementation

Makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -g -Werror -pedantic
LDFLAGS = -lsfml-system

all: EDistance test EDistance.a

EDistance: main.o EDistance.o
    $(CXX) $(CXXFLAGS) -o $@ $^ $(LDFLAGS)

test: test.o EDistance.o
    $(CXX) $(CXXFLAGS) -o $@ $^ -lboost_unit_test_framework

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

%.o: %.cpp
    $(CXX) $(CXXFLAGS) -c $<

clean:
    rm -f *.o EDistance test EDistance.a

lint:
    cpplint --filter=-legal/copyright *.cpp *.hpp

.PHONY: all clean lint
```

main.cpp

```cpp
// copyright 2024 Sameera Sakinala
#include <sys/resource.h>
#include <unistd.h>
#include <iostream>
#include <string>
#include <chrono>
#include <fstream>
#include "EDistance.hpp"

// Function to get current memory usage
double getCurrentMemoryUsage() {
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    return usage.ru_maxrss / 1024.0;
}

// Function to get CPU information
std::string getCPUInfo() {
    std::ifstream cpuinfo("/proc/cpuinfo");
    std::string line;
    while (std::getline(cpuinfo, line)) {
        if (line.substr(0, 10) == "model name") {
            return line.substr(line.find(":") + 2);
        }
    }
    return "Unknown";
}

int main() {
    std::string x, y;
    std::cin >> x >> y;
    double initialMemory = getCurrentMemoryUsage();
    auto start = std::chrono::high_resolution_clock::now();
    EDistance ed(x, y);
    int distance = ed.optDistance();
    std::string alignment = ed.alignment();
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> diff = end - start;
    double finalMemory = getCurrentMemoryUsage();
std::cout << "Edit distance = " << distance << std::endl;
    std::cout << alignment;
    std::cout << "Execution time is " << diff.count() << " seconds" << std::
    endl;
    std::cout << "Memory used: " << (finalMemory - initialMemory) << " MB" <<
    std::endl;

    auto minComparisonResults = ed.minComparisonAnalysis();
    std::cout << "\nMin Comparison Analysis:" << std::endl;
```

```
    for (const auto& [a, b, c, min3_result, direct_result, is_equal] :
    minComparisonResults) {
std::cout <<
"(" << a << ", " << b << ", " << c << "): min3 = " << min3_result
<< ", direct = " << direct_result << ", equal: " << (is_equal ? "true" : "
    false") << std::endl;
    }

    auto [unoptimized_time, optimized_time] = ed.
    optimizationComparisonAnalysis();
    std::cout << "\nOptimization Comparison Analysis:" << std::endl;
    std::cout << "Unoptimized time: " << unoptimized_time << " seconds" << std
    ::endl;
    std::cout << "Optimized time: " << optimized_time << " seconds" << std::
    endl;

    return 0;
}
```

EDistance.hpp

```
// copyright 2024 Sameera Sakinala

#ifndef EDISTANCE_HPP
#define EDISTANCE_HPP
#include <string>
#include <vector>
#include <tuple>

class EDistance {
 private:
    std::string x, y;
    std::vector<std::vector<int>> opt;

 public:
    EDistance(const std::string& x, const std::string& y);
    ~EDistance();

    static int penalty(char a, char b);
    static int min3(int a, int b, int c);

    int optDistance();
    std::string alignment();

    std::vector<std::tuple<int, int, int, int, int, bool>>
    minComparisonAnalysis();
    std::pair<double, double> optimizationComparisonAnalysis();
};

#endif   //   EDISTANCE_HPP
```

EDistance.cpp

```cpp
// copyright 2024 Sameera Sakinala
#include "EDistance.hpp"
#include <algorithm>
#include <sstream>
#include <iomanip>
#include <chrono>

EDistance::EDistance(const std::string& x, const std::string& y) : x(x), y(y) {
    opt.resize(x.length() + 1, std::vector<int>(y.length() + 1, 0));
}

EDistance::~EDistance() {}

int EDistance::penalty(char a, char b) {
    return (a == b) ? 0 : 1;
}

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

int EDistance::optDistance() {
    int m = x.length(), n = y.length();
    // Initialize the base cases
    for (int i = 0; i <= m; i++) opt[i][n] = 2 * (m - i);
    for (int j = 0; j <= n; j++) opt[m][j] = 2 * (n - j);
    // Fill the matrix
    for (int i = m - 1; i >= 0; i--) {
        for (int j = n - 1; j >= 0; j--) {
            opt[i][j] = min3(
                opt[i + 1][j + 1] + penalty(x[i], y[j]),
                opt[i + 1][j] + 2,
                opt[i][j + 1] + 2);
        }
    }
    return opt[0][0];
}

std::string EDistance::alignment() {
    std::ostringstream result;
    std::string::size_type i = 0, j = 0;
    while (i < x.size() || j < y.size()) {
        if (i < x.size() && j < y.size() && opt[i][j] == opt[i + 1][j + 1] + penalty(x[i], y[j])) {
            result << x[i] << " " << y[j] << " " << penalty(x[i], y[j]) << "\n";
            ++i;
            ++j;
```

```
        } else if (i < x.size() && opt[i][j] == opt[i + 1][j] + 2) {
            result << x[i] << " - 2\n";
            ++i;
        } else {
            result << "- " << y[j] << " 2\n";
            ++j;
        }
    }
    return result.str();
}

std::vector<std::tuple<int, int, int, int, int, bool>> EDistance::
    minComparisonAnalysis() {
    std::vector<std::tuple<int, int, int, int, int, bool>> results;
    std::vector<std::tuple<int, int, int>> test_values = {{3, 5, 2}, {10, 1,
    7}, {6, 6, 6}};
    for (const auto& [a, b, c] : test_values) {
        int min3_result = min3(a, b, c);
        int direct_result = std::min({a, b, c});
        results.emplace_back(a, b, c, min3_result, direct_result, min3_result
    == direct_result);
    }
    return results;
}

std::pair<double, double> EDistance::optimizationComparisonAnalysis() {
    auto start = std::chrono::high_resolution_clock::now();
    optDistance();  // Run the unoptimized version
    auto end = std::chrono::high_resolution_clock::now();
    double unoptimized_time = std::chrono::duration<double>(end - start).count
    ();

    // Placeholder for optimized version
    start = std::chrono::high_resolution_clock::now();
    optDistance();
    end = std::chrono::high_resolution_clock::now();
    double optimized_time = std::chrono::duration<double>(end - start).count()
    ;

    return {unoptimized_time, optimized_time};
}
```

test.cpp

```
1
2  // copyright 2024 Sameera Sakinala
3  #include <algorithm>
4  #include <sstream>
5  #include <vector>
6  #define BOOST_TEST_MODULE EDistanceTest
7  #include <boost/test/included/unit_test.hpp>
8  #include "EDistance.hpp"
9
10
11 BOOST_AUTO_TEST_CASE(test_penalty) {
12     BOOST_CHECK_EQUAL(EDistance::penalty('A', 'A'), 0);
13     BOOST_CHECK_EQUAL(EDistance::penalty('A', 'T'), 1);
14 }
15
16 BOOST_AUTO_TEST_CASE(test_min3) {
17     BOOST_CHECK_EQUAL(EDistance::min3(1, 2, 3), 1);
18     BOOST_CHECK_EQUAL(EDistance::min3(3, 1, 2), 1);
19     BOOST_CHECK_EQUAL(EDistance::min3(3, 2, 1), 1);
20 }
21
22 BOOST_AUTO_TEST_CASE(test_optDistance) {
23     EDistance ed1("AACAGTTACC", "TAAGGTCA");
24     BOOST_CHECK_EQUAL(ed1.optDistance(), 7);
25     EDistance ed2("", "ACATAG");
26     BOOST_CHECK_EQUAL(ed2.optDistance(), 12);
27     EDistance ed3("AGTACG", "");
28     BOOST_CHECK_EQUAL(ed3.optDistance(), 12);
29 }
30
31 BOOST_AUTO_TEST_CASE(test_alignment) {
32     EDistance ed("AACAGTTACC", "TAAGGTCA");
33     ed.optDistance();
34     std::string alignment = ed.alignment();
35     BOOST_CHECK(!alignment.empty());
36 BOOST_CHECK(alignment.find("A T") < alignment.rfind("C A"));
37 BOOST_CHECK(alignment.find("A T") != std::string::npos);
38     BOOST_CHECK(alignment.rfind("C A") != std::string::npos);
39     int lineCount = std::count(alignment.begin(), alignment.end(), '\n');
40     BOOST_CHECK_EQUAL(lineCount, 10);
41 }
42
43 BOOST_AUTO_TEST_CASE(test_swapped_strings) {
44     EDistance ed1("AACAGTTACC", "TAAGGTCA");
45     EDistance ed2("TAAGGTCA", "AACAGTTACC");
46     BOOST_CHECK_EQUAL(ed1.optDistance(), ed2.optDistance());
47     std::string alignment1 = ed1.alignment();
48     std::string alignment2 = ed2.alignment();
49     BOOST_CHECK_NE(alignment1, alignment2);
```

```
}

BOOST_AUTO_TEST_CASE(test_alignment_direction) {
    EDistance ed("AGT", "ACA");
    ed.optDistance();
    std::string alignment = ed.alignment();
    BOOST_CHECK(alignment.find("A A") < alignment.find("G C"));
    BOOST_CHECK(alignment.find("G C") < alignment.find("T A"));
}

BOOST_AUTO_TEST_CASE(test_alignment_completeness) {
    EDistance ed("AACAGTTACC", "TAAGGTCA");
    ed.optDistance();
    std::string alignment = ed.alignment();
    std::vector<std::string> lines;
    std::istringstream iss(alignment);
    for (std::string line; std::getline(iss, line); ) {
        lines.push_back(line);
    }
    BOOST_CHECK_EQUAL(lines.size(), 10);
BOOST_CHECK(lines.front().find('A') != std::string::npos ||
lines.front().find('T') != std::string::npos);
BOOST_CHECK(lines.back().find('C') != std::string::npos ||
lines.back().find('A') != std::string::npos);
    std::string seq1, seq2;
    for (const auto& line : lines) {
        if (line[0] != '-') seq1 += line[0];
        if (line[2] != '-') seq2 += line[2];
    }
    BOOST_CHECK_EQUAL(seq1, "AACAGTTACC");
    BOOST_CHECK_EQUAL(seq2, "TAAGGTCA");
    std::string expected_chars = "AACAGTTACCTAAGGTCA-";
    for (char c : expected_chars) {
        BOOST_CHECK(alignment.find(c) != std::string::npos);
    }
}
```

6.7 Result

Input:

AACAGTTACC
TAAGGTCA

Output:

```
 Edit distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
Execution time is 4.1185e-05 seconds
Memory used: 0 MB

Min Comparison Analysis:
(3, 5, 2): min3 = 2, direct = 2, equal: true
(10, 1, 7): min3 = 1, direct = 1, equal: true
(6, 6, 6): min3 = 6, direct = 6, equal: true

Optimization Comparison Analysis:
Unoptimized time: 3.466e-06 seconds
Optimized time: 3.126e-06 seconds
```

7 PS6: RandWriter

7.1 Discussion

This project was all about creating a Markov Chain-based text generator—a tool that takes in a piece of text, analyzes its patterns, and produces new, random text that mimics the input's style. The code tackles several interesting challenges, such as efficiently managing k-grams, dealing with circular text seamlessly, and allowing for transformations using lambda functions. Through careful design, implementation, and testing, I was able to create a tool that's both robust and flexible.

7.2 Key Algorithms, Data Structures and OO Designs

Markov Chain Algorithm for Random Text Generation At the core of this project lies the Markov Chain algorithm, which models the generation of random text based on the statistical properties of the input text. Specifically, the algorithm focuses on k-grams (substrings of length k). For every k-gram, it keeps track of the characters that follow it, creating a frequency distribution of these transitions.

The algorithm generates text by :

- **Selecting a seed k-gram :** Starting with the initial k-gram, the algorithm uses the frequency map to determine the most likely next character based on the transitions.
- **Stochastic Selection :** A random choice is made from the possible next characters, weighted by their frequencies.
- **Circular Text Handling :** When the text reaches the end, it continues from the beginning, treating the text as circular to maintain continuity of k-grams. This algorithm relies on the principle that given the previous k characters, the next character can be predicted with a certain probability.

Data Structures

Several data structures were essential in efficiently implementing the Markov Chain algorithm and ensuring the program's performance:

- **Maps (std::map)**

  The main data structure used for storing the frequency distribution of k-grams is a map of `maps: std::map<std::string, std::map<char, int>>`. The outer map's key is a k-gram (a substring of length k), and the inner map stores characters that follow that k-gram as keys, with their frequency counts as values.
  This data structure provides efficient lookups, insertions, and updates, which is crucial when building the frequency map from the input text and when performing random text generation.

- **String Handling :** The input text and the generated text are handled using `std::string`. Efficient manipulation of strings, especially when working with substrings, is key for both building k-grams and generating text of the desired length. The use of `std::ostringstream` allows for efficient reading of the input text from standard input, ensuring that the program can handle varying input sizes seamlessly.

7.3 What I Accomplished

- **Built a Functional Text Generator :** I successfully designed and implemented a program that uses a Markov Chain to mimic text patterns. It can take any input string and produce new, statistically similar text.
- **Handled Edge Cases and Errors :** From managing short or circular input text to validating the randomness of outputs, I ensured the program was resilient to unexpected situations.
- **Comprehensive Testing :** I wrote detailed unit tests to validate all parts of the implementation, which gave me confidence in the program's reliability.
- **Clean Code and Documentation :** I prioritized clear, maintainable code and included documentation so others could easily understand how the program works.

7.4 What I Learned

- **How Markov Chains Work :** I now have a solid understanding of how Markov Chains model text and how transitions between k-grams are managed.
- **Advanced C++ Features :** I became more comfortable using C++ maps, random number generators, and lambda functions, which were crucial to the program's flexibility.
- **Testing Probabilistic Systems :** Testing functions like kRand that rely on randomness was challenging but rewarding, as I learned to use statistical methods to verify correctness.
- **Error Handling in C++ :** By handling exceptions for invalid input or runtime issues, I gained more confidence in writing robust C++ code.

7.5 Challenges

- **Managing Circular Text :** Ensuring that the program treated the input as circular (so the end wraps around to the start) required careful attention to detail in constructing the frequency map.
- **Testing Randomness :** Verifying that random text generation worked as intended was tricky, since randomness inherently involves variability. I had to test by running simulations and checking overall patterns.
- **Generating Text of Exact Length :** Initially, the program struggled to produce output of the exact requested length. Fixing this required refining the logic in the generate method.
- **Input Validation :** Ensuring that the program handled edge cases—like very short input strings or invalid arguments—took more time than expected but was worth it for the final robustness.

7.6 Code Implementation

Makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -Wextra -pedantic -Werror
LDFLAGS = -lboost_unit_test_framework

all: TextWriter TextWriter.a test

TextWriter: TextWriter.o RandWriter.o
    $(CXX) $(CXXFLAGS) -o $@ $^
TextWriter.a: RandWriter.o
    ar rcs $@ $^
test: test.o RandWriter.o
    $(CXX) $(CXXFLAGS) -o $@ $^ $(LDFLAGS)

%.o: %.cpp
    $(CXX) $(CXXFLAGS) -c $<
lint:
    cpplint --filter=-legal/copyright *.cpp *.hpp

clean:
    rm -f *.o TextWriter TextWriter.a test

.PHONY: all clean lint
```

TextWriter.cpp

```
// Copyright 2024 Sameera Sakinala
#include <iostream>
#include <sstream>
#include "RandWriter.hpp"

int main(int argc, char* argv[]) {
    // Check for correct usage
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <k> <L>\n";
        return 1;
    }

    const size_t k = static_cast<size_t>(std::stoul(argv[1]));
    const size_t L = static_cast<size_t>(std::stoul(argv[2]));

    std::ostringstream oss;
    oss << std::cin.rdbuf();

    // Validate input text is not empty
    const std::string inputText = oss.str();
    if (inputText.empty()) {
        std::cerr << "Error: Input text cannot be empty.\n";
```

```
        return 1;
    }

    try {
        RandWriter rw(inputText, k);
        // Generate output starting with the first 'k' characters as seed
        std::cout << rw.generate(inputText.substr(0, k), L) << '\n';
    } catch (const std::exception& e) {
        // Handle any exceptions thrown by RandWriter
        std::cerr << "Error: " << e.what() << '\n';
        return 1;
    }

    return 0;
}
```

RandWriter.hpp

```
// Copyright 2024 Sameera Sakinala
#ifndef RANDWRITER_HPP
#define RANDWRITER_HPP

#include <string>
#include <map>
#include <random>
#include <functional>

class RandWriter {
 public:
    RandWriter(const std::string& text, size_t k);
    size_t orderK() const;
    int freq(const std::string& kgram) const;
    int freq(const std::string& kgram, char c) const;
    char kRand(const std::string& kgram);
    std::string generate(const std::string& kgram, size_t L);

    // Lambda function parameter usage
    void transformAlphabet(const std::function<char(char)>& func);

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

 private:
    size_t k;
    std::string alphabet;
    std::map<std::string, std::map<char, int>> freqMap;
    std::mt19937 rng;
};
#endif  // RANDWRITER_HPP
```

RandWriter.cpp

```cpp
// Copyright 2024 Sameera Sakinala
#include "RandWriter.hpp"
#include <stdexcept>
#include <algorithm>
#include <iostream>
#include <random>

RandWriter::RandWriter(const std::string& text, size_t k)
    : k(k), rng(std::random_device { } ()) {
    if (text.length() < k) {
        throw std::invalid_argument
        ("Text length must be at least the value of k.");
    }

    const size_t n = text.length();
    for (size_t i = 0; i < n; ++i) {
        std::string kgram = text.substr(i, k);
        if (kgram.length() < k) {
            kgram += text.substr(0, k - kgram.length());
        }
        const char nextChar = text[(i + k) % n];
        freqMap[kgram][nextChar]++;

        if (alphabet.find(nextChar) == std::string::npos) {
            alphabet += nextChar;
        }
    }
}

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

int RandWriter::freq(const std::string& kgram) const {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram must be of length k.");
    }

    const auto it = freqMap.find(kgram);
    if (it != freqMap.end()) {
        int total = 0;
        for (const auto& pair : it->second) {
            total += pair.second;
        }
        return total;
    }
    return 0;
}

```

```
int RandWriter::freq(const std::string& kgram, char c) const {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram must be of length k.");
    }

    const auto it = freqMap.find(kgram);
    if (it != freqMap.end()) {
        const auto charIt = it->second.find(c);
        if (charIt != it->second.end()) {
            return charIt->second;
        }
    }
    return 0;
}

char RandWriter::kRand(const std::string& kgram) {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram must be of length k.");
    }

    const auto it = freqMap.find(kgram);
    if (it == freqMap.end()) {
        throw std::runtime_error("No such kgram exists in the frequency map.")
    ;
    }

    const int total = freq(kgram);
    if (total == 0) {
        throw std::runtime_error("No valid transitions from this kgram.");
    }

    std::uniform_int_distribution<> dis(1, total);
    const int r = dis(rng);

    int sum = 0;
    for (const auto& pair : it->second) {
        sum += pair.second;
        if (r <= sum) {
            return pair.first;
        }
    }
    throw std::runtime_error
    ("Unexpected error: kRand failed to select a character.");
}
std::string RandWriter::generate(const std::string& kgram, size_t L) {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram must be of length k.");
    }
    if (L < k) {
        throw std::invalid_argument("L must be at least the value of k.");
    }
```

```

    std::string result = kgram;  // Start with the provided k-gram
    result.reserve(L);           // Reserve space for efficiency

    while (result.length() < L) {
        std::string currentKgram = result.substr(result.length() - k, k);
        char nextChar = kRand(currentKgram);
        result += nextChar;
    }

    return result;
}
void RandWriter::transformAlphabet(const std::function<char(char)>& func) {
    for (char& c : alphabet) {
        c = func(c);  // Apply the lambda to each character
    }

    // Ensure the alphabet remains ordered
    std::sort(alphabet.begin(), alphabet.end());
}


std::ostream& operator<<(std::ostream& os, const RandWriter& rw) {
    os << "Order: " << rw.k << "\n";
    os << "Alphabet: " << rw.alphabet << "\n";
    os << "Frequencies:\n";

    for (const auto& outer : rw.freqMap) {
        for (const auto& inner : outer.second) {
            os << outer.first << inner.first << ": " << inner.second << "\n";
        }
    }

    return os;
}
```

test.cpp

```
// Copyright 2024 Sameera Sakinala
#define BOOST_TEST_MODULE RandWriterTest
#include <boost/test/included/unit_test.hpp>
#include "RandWriter.hpp"

BOOST_AUTO_TEST_CASE(constructor_test) {
    BOOST_REQUIRE_NO_THROW(RandWriter("abcde", 2));
    BOOST_REQUIRE_THROW(RandWriter("abc", 4), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(order_k_test) {
    RandWriter rw("abcde", 2);
    BOOST_REQUIRE_EQUAL(rw.orderK(), 2);
```

```
14 }
15
16 BOOST_AUTO_TEST_CASE(freq_test) {
17     RandWriter rw("gagggagaggcgagaaa", 2);
18     BOOST_REQUIRE_EQUAL(rw.freq("ga"), 5);
19     BOOST_REQUIRE_EQUAL(rw.freq("gg"), 3);
20     BOOST_REQUIRE_EQUAL(rw.freq("aa"), 2);
21     BOOST_REQUIRE_EQUAL(rw.freq("zz"), 0);
22     BOOST_REQUIRE_THROW(rw.freq("a"), std::invalid_argument);
23 }
24
25 BOOST_AUTO_TEST_CASE(freq_char_test) {
26     RandWriter rw("gagggagaggcgagaaa", 2);
27     BOOST_REQUIRE_EQUAL(rw.freq("ga", 'g'), 4);
28     BOOST_REQUIRE_EQUAL(rw.freq("ga", 'a'), 1);
29     BOOST_REQUIRE_EQUAL(rw.freq("gg", 'g'), 1);
30     BOOST_REQUIRE_EQUAL(rw.freq("gg", 'a'), 1);
31     BOOST_REQUIRE_EQUAL(rw.freq("aa", 'a'), 1);
32     BOOST_REQUIRE_EQUAL(rw.freq("zz", 'a'), 0);
33     BOOST_REQUIRE_THROW(rw.freq("a", 'a'), std::invalid_argument);
34 }
35
36 BOOST_AUTO_TEST_CASE(krand_test) {
37     RandWriter rw("gagggagaggcgagaaa", 2);
38     std::string kgram = "ga";
39     std::map<char, int> counts;
40     for (int i = 0; i < 1000; ++i) {
41         counts[rw.kRand(kgram)]++;
42     }
43     BOOST_REQUIRE(counts['g'] > 700);
44     BOOST_REQUIRE(counts['a'] > 100);
45     BOOST_REQUIRE(counts['c'] < 100);
46     BOOST_REQUIRE_THROW(rw.kRand("zz"), std::runtime_error);
47     BOOST_REQUIRE_THROW(rw.kRand("a"), std::invalid_argument);
48 }
49
50 BOOST_AUTO_TEST_CASE(generate_test) {
51     RandWriter rw("gagggagaggcgagaaa", 2);
52     std::string generated = rw.generate("ga", 20);
53     BOOST_REQUIRE_EQUAL(generated.length(), 20);
54     BOOST_REQUIRE_EQUAL(generated.substr(0, 2), "ga");
55     BOOST_REQUIRE_THROW(rw.generate("a", 20), std::invalid_argument);
56     BOOST_REQUIRE_THROW(rw.generate("ga", 1), std::invalid_argument);
57 }
58 BOOST_AUTO_TEST_CASE(generate_length_test) {
59     RandWriter rw("gagggagaggcgagaaa", 2);
60
61     std::string generated = rw.generate("ga", 300);
62     BOOST_REQUIRE_EQUAL(generated.length(), 300);
63     BOOST_REQUIRE_EQUAL(generated.substr(0, 2), "ga");
64 }BOOST_AUTO_TEST_CASE(transform_alphabet_test) {
```

```
65      RandWriter rw("abcdef", 2);
66
67      rw.transformAlphabet([](char c) { return std::toupper(c); });
68
69      std::ostringstream oss;
70      oss << rw;
71      std::string output = oss.str();
72      std::cout << "Generated output: " << output << std::endl;
73
74      // Check if the alphabet contains uppercase letters
75      BOOST_REQUIRE_MESSAGE(
76          output.find("ABCDEF") != std::string::npos,
77          "Alphabet transformation failed. Output: " + output);
78  }
```

7.7 Result

Input:
Two households, both alike in dignity
(In fair Verona, where we lay our scene),
From ancient grudge break to new mutiny,
Where civil blood makes civil hands unclean.
From forth the fatal loins of these two foes
A pair of star-crossed lovers take their life;
Whose misadventured piteous overthrows
Doth with their death bury their parents' strife.
The fearful passage of their death-marked love
And the continuance of their parents' rage,
Which, but their children's end, naught could remove,
Is now the two hours' traffic of our stage;
The which, if you with patient ears attend,
What here shall miss, our toil shall strive to mend.

```
1  Two households, both alike in dignity
2  (In fair Verona, where we lay our scene),
3  From ancient grudge break to new mutiny,
4  Where civil blood makes civil hands unclean.
5  From forth the fatal loins of these two foes
6  A pair of star-crossed lovers take their life;
7  Whose misadventured piteous overthrows
8  Doth with their death bury their parents' strife.
9  The fearful passage of their death-marked lov
```

8 PS7: Kronos Log Parsing

8.1 Discussion

This project was all about digging into Kronos InTouch time clock logs to figure out when the devices started up and how long it took them to complete. The goal was to extract useful information from raw log files, match up start and completion events, and generate a summary that tells the story of the boot processes. Using regular expressions was key to parsing the logs, but it came with its challenges. I also had to handle cases where the logs didn't tell the whole story, like when a boot start didn't have a matching completion.

8.2 Key Algorithms, Data Structures and OO Designs

Key Algorithm: Log Parsing with Regular Expressions
The core algorithm driving the log parsing functionality relies heavily on regular expressions (regex). This algorithm involves scanning each line of the log files to search for specific patterns representing boot start and boot completion events. The regex patterns used were: **Boot Start Event :** The pattern `\(\log\.c\.166\)\s+server` started was used to identify the beginning of a boot process.

Boot Completion Event : The pattern `oejs\.AbstractConnector:Started SelectChannelConnector@0\.0\.0\.0:\d+` was used to identify the end of a boot process.

Timestamp Extraction : A timestamp regex `(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})` was utilized to extract time values from each relevant log line. These regular expressions allowed the algorithm to efficiently identify and extract timestamps from the logs, mapping the start and completion events to calculate boot durations.

Key Data Structure :
BootEvent Struct The BootEvent struct was crucial in organizing the log data. It stored the following properties:
LineNumber : Tracks the line number in the log file for debugging and reporting purposes.
Timestamp : Stores the timestamp extracted from the log, representing the exact time of the event.
isStart : A boolean flag that distinguishes between a boot start event and a boot completion event. This data structure allowed for easy management of each event's key information and facilitated the pairing of boot start and completion events during the analysis.

Event Pairing and Boot Duration Calculation
A key challenge was efficiently pairing boot start events with their corresponding completion events. The algorithm loops through the events and pairs a start event with the next corresponding completion event. When both are found, the algorithm calculates the time difference using the `boost::posix_time` library. This calculation is based on the following steps:
Start Event : When a boot start event is found, the timestamp is saved.
Completion Event : The algorithm looks for the next non-start event to pair it with the saved start event.
Time Difference : The difference in time is calculated using Boost's ptime objects to handle date and time manipulation and to ensure accurate millisecond calculations.

Design Choices :
The design of the program was based on simplicity and clarity. The parsing, pairing, calculation, and reporting processes were separated into distinct functions to promote modularity and

readability. Specifically:
Modular Functions : Functions like parseLogFile and generateReport were designed to handle specific tasks, reducing complexity and allowing for easy testing and debugging.
Error Handling : The program includes basic error handling to manage cases where files can't be opened, or where the expected events are incomplete, ensuring robustness.

8.3 What I Accomplished

- Built a parser that could reliably find boot start and completion events in the logs.
- Calculated how long it took for devices to boot up when both start and completion events were present.
- Created a clean and detailed report summarizing the findings.
- Made the program handle tricky cases, like when a boot start didn't have a matching completion, without crashing

8.4 What I Learned

- Regular expressions are powerful but can be tricky to get just right, especially for parsing logs.
- Parsing structured text requires attention to detail and a good understanding of the data format.
- Boost's date-time library is incredibly useful for time-based calculations, but there is a bit of a learning curve.
- Breaking the problem into smaller tasks—parsing, pairing, calculating, and reporting—makes the whole process more manageable.

8.5 Challenges

- **Incomplete Events :** Some logs didn't have a boot completion after the start. Figuring out how to handle those cases without breaking the program was a real puzzle.
- **Time Calculations :** I had never used Boost's date-time library before, so understanding how to parse timestamps and calculate differences took some time.
- **Debugging Large Logs :** Working with bigger log files made it harder to spot issues. Debugging became a slow process of checking the output line by line.

8.6 Code Implementation

Makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -Wall -Wextra -Werror -pedantic
LDFLAGS = -lboost_date_time

all: ps7

ps7: ps7.cpp
    $(CXX) $(CXXFLAGS) -o ps7 ps7.cpp $(LDFLAGS)

clean:
    rm -f ps7 *.o

lint:
    cpplint --filter=-legal/copyright ps7.cpp

.PHONY: all clean lint
```

ps7.cpp

```
// Copyright 2024 Sameera Sakinala
#include <iostream>
#include <fstream>
#include <regex>
#include <string>
#include <vector>
#include <boost/date_time/posix_time/posix_time.hpp>

struct BootEvent {
    int lineNumber;
    std::string timestamp;
    bool isStart;
};

std::vector<BootEvent> parseLogFile(const std::string& filename) {
    std::ifstream file(filename);
    if (!file.is_open()) {
        std::cerr << "Error: Unable to open file " << filename << "\n";
        return {};
    }

    std::string line;
    std::vector<BootEvent> events;
    int lineNumber = 0;

    std::regex startPattern(R"(\(log\.c\.166\)\s+server started)");
    std::regex endPattern(R"(oejs\.AbstractConnector:Started
   SelectChannelConnector@0\.0\.0\.0:\d+)");
```

```
28      std::regex timestampPattern(R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}))");
29
30      while (std::getline(file, line)) {
31          lineNumber++;
32          std::smatch match;
33          if (std::regex_search(line, match, timestampPattern)) {
34              std::string timestamp = match[1];
35              if (std::regex_search(line, startPattern)) {
36                  events.push_back({lineNumber, timestamp, true});
37              } else if (std::regex_search(line, endPattern)) {
38                  events.push_back({lineNumber, timestamp, false});
39              }
40          }
41      }
42
43      return events;
44  }
45
46  void generateReport
47  (const std::string& inputFilename, const std::vector<BootEvent>& events) {
48      std::string outputFilename = inputFilename + ".rpt";
49      std::ofstream outFile(outputFilename);
50      if (!outFile.is_open()) {
51  std::cerr
52  << "Error: Unable to create report file " << outputFilename << "\n";
53          return;
54      }
55
56      int bootCount = 0;
57      int completeBootCount = 0;
58      int incompleteBootCount = 0;
59
60      for (size_t i = 0; i < events.size(); ++i) {
61          if (events[i].isStart) {
62              bootCount++;
63              outFile << events[i].lineNumber << "(" << inputFilename << "): "
64                      << events[i].timestamp << " Boot Start\n";
65
66              bool bootCompleted = false;
67              for (size_t j = i + 1; j < events.size(); ++j) {
68                  if (events[j].isStart) {
69                      break;
70                  }
71                  if (!events[j].isStart) {
72                      bootCompleted = true;
73
74                      boost::posix_time::ptime start =
75  boost::posix_time::time_from_string(events[i].timestamp);
76                      boost::posix_time::ptime end =
77  boost::posix_time::time_from_string(events[j].timestamp);
78                      boost::posix_time::time_duration diff = end - start;
```

```
outFile << events[j].lineNumber << "(" << inputFilename << "): "
                            << events[j].timestamp << " Boot Completed   Time:
    "
                            << diff.total_milliseconds() << "ms\n";
                    completeBootCount++;
                    i = j;
                    break;
                }
            }

            if (!bootCompleted) {
                outFile << "Boot Incomplete\n";
                incompleteBootCount++;
            }
            outFile << "\n";
        }
    }

    outFile << "Device Boot Report\n";
    outFile << "InTouch log file: " << inputFilename << "\n";
    outFile << "Summary:\n";
    outFile << "Total number of boots: " << bootCount << "\n";
    outFile << "Number of completed boots: " << completeBootCount << "\n";
    outFile << "Number of incomplete boots: " << incompleteBootCount << "\n\n"
    ;
}

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

    std::string filename = argv[1];
    std::vector<BootEvent> events = parseLogFile(filename);

    if (events.empty()) {
        std::cerr << "Error: No events found. Exiting.\n";
        return 1;
    }

    generateReport(filename, events);

    return 0;
}
```

8.7 Result

Output of Device 1

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

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

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

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

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

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

Device Boot Report
InTouch log file: device1_intouch.log
Summary:
Total number of boots: 6
Number of completed boots: 6
Number of incomplete boots: 0
```