# Computing IV Sec 202: Project Portfolio

**Harishwar Reddy Erri**

**Fall 2024**

## Contents

**Time to Complete: 11hrs**

# 1 PS0: Hello SFML

## 1.1 Discussion

Utilizing the SFML (Simple and Fast Multimedia Library), the project's goal was to develop a graphical program in which a sprite (Mickey Mouse) moved in response to keyboard commands. Enhancing knowledge of the SFML environment, implementing simple graphics programming, and investigating other capabilities like text, audio, and custom visuals were the main objectives. This project reinforced event-driven programming and basic C++ programming ideas while emphasizing creativity.

## 1.2 Key Algorithms, Data Structures, and OO Designs Used

**Key Algorithms**

- **Movement Algorithm:**The sprite's movement is controlled using `sf::Keyboard::isKeyPressed` to detect user input.
  - **Horizontal movement:**
    - Left Arrow: Moves the sprite left by decreasing the x-coordinate.
    - Right Arrow: Moves the sprite right by increasing the x-coordinate.
  - **Vertical movement:**
    - Up Arrow: Moves the sprite upward by decreasing the y-coordinate.
    - Down Arrow: Moves the sprite downward by increasing the y-coordinate.
  - Boundary constraints ensure that the sprite stays within a specified range using `if` conditions for `position.x` and `position.y`.
- **Sprite Direction Handling:**
  - To flip the sprite's orientation, the x-scale is inverted using:

    `sprite.setScale(-sprite.getScale().x, sprite.getScale().y)`.
  - This ensures that the sprite faces left or right based on movement direction.
- **Boundary Constraints:**
  - Horizontal movement is restricted:
    - Left boundary: `if (position.x >= 130)`.
    - Right boundary: `if (position.x <= 676)`.
  - Vertical movement is similarly restricted for the up and down directions.
- **Audio Control:**
  - `sound.play()` is used to start playback of the audio.
  - `sound.stop()` stops the playback.
- **Reset Position:**
  - Pressing the `0` key resets the sprite's position to `(130, 598)` using:

    `sprite.setPosition(130, 598)`.

**Data Structures**

- `sf::Vector2f`
  - A 2D vector for storing floating-point values.
  - Used to:
    - Retrieve the sprite's position: `sf::Vector2f position = sprite.getPosition()`.
    - Set the scaling of the background: `background.setScale()`.
- `sf::Texture`
  - Holds image data for rendering.
  - `tec` loads the sprite texture, and `bg` loads the background texture.
- `sf::SoundBuffer`
  - Stores audio data, loaded from `sound.wav`.
  - Passed to the `sf::Sound` object for playback.
- `sf::CircleShape`
  - Represents a 2D circular shape.
  - Used to render a decorative element in the window.
- `sf::Font` **and** `sf::Text`
  - `sf::Font` stores font data loaded from a file.
  - `sf::Text` displays the title **MICKEY MOUSE** in the window.

**Object-Oriented Designs**

- **Encapsulation:**
  - SFML classes like `sf::RenderWindow`, `sf::Texture`, and `sf::Sound` encapsulate complex functionality into simple objects with clear interfaces.
- **Inheritance and Polymorphism:**
  - Classes like `sf::Sprite`, `sf::Text`, and `sf::CircleShape` inherit from `sf::Drawable`, allowing them to be uniformly handled by the `window.draw()` method.
- **Modularity:**
  - Background rendering is managed using `sf::Texture` and `sf::Sprite`.
  - Text rendering uses `sf::Font` and `sf::Text`.
  - Audio is handled with `sf::SoundBuffer` and `sf::Sound`.
  - Interactive sprite behavior is handled independently.
- **Use of Constructors:**
  - The `sf::RenderWindow` object is constructed with dimensions and a title.
  - The `sf::CircleShape` object is constructed with a radius parameter.

## 1.3 Images Used

Figure 1

Figure 2

## 1.4 What I Accomplished

Successfully implemented a graphical application using SFML. Enabled sprite movement in four directions via keyboard inputs. Implemented boundary constraints to restrict sprite movement within the window.

Added additional features:

- Sprite flipping based on movement direction.
- Audio playback functionality triggered by specific keys.
- A reset feature for sprite position.
- Background image integration and text rendering with a custom font.
- Optimized the program to maintain a smooth frame rate of 60 fps.

## 1.5 What I Already Knew

- Fundamental knowledge of C++ programming, including:

  - How to use loops, conditionals, and functions.
- Concepts that are object-oriented, such as:
  - Inheritance.
  - Encapsulation.
- Knowledge of using GCC and Makefiles to compile and debug C++ code.
- Basic knowledge of managing user input and event-driven programming.

### 1.6 What I Learned

- The basics of working with the SFML library, including:
  - Managing graphics using `sf::Sprite`, `sf::Texture`, and window creation.
  - Event handling for keystrokes and real-time interactions.
  - Adding multimedia elements like audio and text.
  - Techniques for controlling animation frame rates with `sf::Clock` and `setFramerateLimit()`.
  - Debugging runtime issues in a graphical environment.
  - Integrating multiple assets like images, fonts, and audio into a cohesive application.

### 1.7 Challenges

- **Delay in Window Loading:**
  - Said to be brought on by the audio feature's inclusion.
  - Verifying file locations, formats, and audio file load durations was necessary to debug this delay.
- **Limitations on Sprite Movement:**
  - Conditional logic and precise boundary checks had to be used to make sure the sprite didn't go outside the window's boundaries.
- **Including Extra Features:**
  - A greater comprehension of the SFML API was necessary in order to add sprite flipping, audio controls, and a reset option.
- **Optimizing and Debugging:**
  - One of the most important learning experiences was managing several interactive features while maintaining fluid performance at 60 frames per second.

## 1.8 Codebase

**MakeFile**

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

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**

```
// Copyright 2024 <Harishwar Reddy Erri>
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
int main() {
    sf::RenderWindow window(sf::VideoMode(800, 750), "MICKEY MOUSE IN SFML!");
    sf::SoundBuffer buffer;
    buffer.loadFromFile("sound.wav");
    sf::Sound sound;
    sound.setBuffer(buffer);

    // Shape
    sf::CircleShape circle(40.f);
    circle.setFillColor(sf::Color(201, 104, 104));
    circle.setPosition(30, 30);

    // Background
    sf::Texture bg;
```

```
19      if (!bg.loadFromFile("bg.jpg")) {
20          return -1;
21      }
22      sf::Sprite background;
23      background.setTexture(bg);
24      sf::Vector2u textureSize = bg.getSize();
25      sf::Vector2u windowSize = window.getSize();
26      background.setScale(static_cast<float>(windowSize.x) / textureSize.x,
27                          static_cast<float>(windowSize.y) / textureSize.y);
28
29      // Font
30      sf::Font font;
31      if (!font.loadFromFile("font.ttf")) {
32          return -1;
33      }
34      sf::Text text;
35      text.setFont(font);
36      text.setString("MICKEY MOUSE");
37      text.setCharacterSize(100);
38      text.setFillColor(sf::Color(10, 104, 71));
39      text.setPosition(40, 100);
40      // Sprite
41      sf::Texture tec;
42      if (!tec.loadFromFile("sprite.png")) {
43          std::cout << "Sorry for the error" << std::endl;
44      }
45      window.setFramerateLimit(60);
46      sf::Sprite sprite;
47      sprite.setTexture(tec);
48      sprite.setScale(0.32f, 0.32f);
49      sprite.setOrigin(sprite.getLocalBounds().width / 2, sprite.getLocalBounds
        ().height / 2);
50      sprite.setPosition(130, 598);
51      bool facingLeft = false;
52      int sprite_x = 0, sprite_y = 0;
53      while (window.isOpen()) {
54          sf::Event event;
55          sf::Vector2f position = sprite.getPosition();
56          std::cout << "Sprite Position: x = " << position.x << ", y = " <<
      position.y << std::endl;
57          while (window.pollEvent(event)) {
58              if (event.type == sf::Event::Closed) {
59                  window.close();
60              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
61                  if (position.x >= 130) {
62                      sprite.move(sprite_x - 3, 0);
63                  }
64                  if (!facingLeft) {
65                      sprite.setScale(-sprite.getScale().x, sprite.getScale().y)
      ;
66                      facingLeft = true;
```

```
67                  }
68              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
69                  if (position.x <= 676) {
70                      sprite.move(sprite_x + 3, 0);
71                  }
72                  if (facingLeft) {
73                      sprite.setScale(-sprite.getScale().x, sprite.getScale().y)
    ;
74                      facingLeft = false;
75                  }
76              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
77                  if (position.y >= 350) {
78                      sprite.move(0, sprite_y - 2);
79                  }
80              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
81                  if (position.y <= 598) {
82                      sprite.move(0, sprite_y + 2);
83                  }
84              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::O)) {
85                  sprite.setPosition(130, 598);
86              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::M)) {
87                  sound.play();
88              } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
89                  sound.stop();
90              }
91              window.draw(background);
92              window.draw(text);
93              window.draw(circle);
94              window.draw(sprite);
95              window.display();
96          }
97      }
98      return 0;
99  }
```

## 1.9 Results

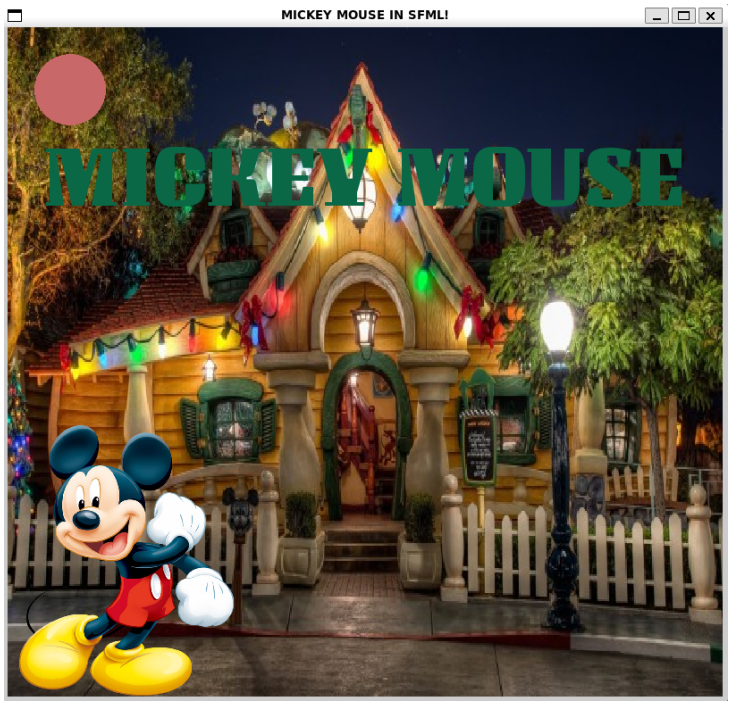


Figure 3: The result of the code executing without any imperfections.

# 2 PS1: LFSR and PhotoMagic

## 2.1 Discussion

In this project, a linear feedback shift register (LFSR) was implemented and tested in order to simulate a basic encryption mechanism and utilize it to encode and decode digital images. The project was specifically split into two sections:

- **PS1a:** Create a class called `FibLFSR` that mimics a 16-bit Fibonacci LFSR. This required:
  - Writing the `step()` and `generate()` functions,
  - Testing these functions using the Boost test framework,
  - Applying object-oriented principles to ensure correct encapsulation.
- **PS1b:** Encrypt and decrypt images by XORing pixel RGB values with pseudo-random numbers produced by the LFSR using the implemented `FibLFSR` class. This section required:
  - Creating a program that could handle input/output picture file processing and encryption/decryption seamlessly,
  - Using SFML for image manipulation and display.

The implementation approach brought to light difficulties including accurately modeling the behavior of the LFSR, guaranteeing effective picture transformations, and establishing a reliable testing environment using the Boost test framework. My knowledge of LFSRs, object-oriented design, cryptography principles, and the use of simulation in digital media encoding and decoding has improved as a result of this project.

**XOR Truth Table:**

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## 2.2 Key Algorithms, Data Structures, and OO Designs Used

**Key Algorithms:**

- **Linear Feedback Shift Register (LFSR) Operations:** The `FibLFSR` class is the core of the encryption mechanism, generating pseudo-random numbers for XOR-based image transformation.
  - **Step Algorithm:**
    - Implements bitwise operations to compute a new bit in the sequence based on feedback.
    - Uses XOR operations on specific bits of the LFSR state.
    - Feedback taps are hardcoded at indices `[0, 2, 3, 5]`.
    - The internal state of the LFSR is updated by shifting bits left and appending the new feedback bit.
  - **Generate Algorithm:**

    - Iteratively calls the `step()` function to generate a `k`-bit number.
    - Combines bits using left-shift and addition to form a numerical value, used for image pixel transformations.

- **Image Transformation:** The `transform` function in the `PhotoMagic` namespace applies the encryption/decryption on an image.
  - **Pixel-wise Transformation:**
    - Iterates over each pixel of the image.
    - Retrieves RGB color channels using `getPixel()`.
    - XORs each channel with numbers generated by `lfsr->generate(8)`.
    - Updates the transformed pixel back into the image using `setPixel()`.
- **Alphanumeric Seed Conversion:** The `alphanumericToBinary` function converts an alphanumeric seed into a binary string.
  - Each character is converted into its 8-bit binary representation using `std::bitset<8>`.
  - The resulting binary string is used to initialize the LFSR.

**Data Structures:**

- `std::string`
  - Represents the internal state (`rs`) of the LFSR.
  - Stores the seed provided by the user and is manipulated during the feedback shift operations.
- **SFML Image Manipulation:**
  - `sf::Image`:
    - Used for loading, manipulating, and saving images.
    - Provides functions like `getPixel` and `setPixel` for pixel-wise operations.
  - `sf::Color`:
    - Stores the RGB color channels of a pixel.
    - Each channel is XOR-ed with the output of the LFSR.
- `std::bitset<8>`
  - Converts alphanumeric characters into binary representation for LFSR seed initialization.
  - Ensures compatibility with the binary-based LFSR mechanism.
- **SFML Rendering Components:**
  - `sf::RenderWindow`:
    - Displays the original and transformed images in separate windows.
  - `sf::Texture` and `sf::Sprite`:
    - Efficiently render the images in the SFML window.

**Object-Oriented (OO) Design**

- **Encapsulation:**
  - The `FibLFSR` class encapsulates all operations related to LFSR.
  - The internal state (`rs`) is private and accessible only via public methods like `step()` and `generate()`.
- **Abstraction:**
  - The `PhotoMagic` namespace provides an interface for image transformation, abstracting the encryption algorithm from the main program.
- **Polymorphism and Operator Overloading:**
  - The `operator<<` is overloaded for `FibLFSR`, allowing its state to be printed directly using an output stream.
- **Modularity:**
  - The program is divided into logical modules:
    - `FibLFSR`: Handles pseudo-random number generation.
    - `PhotoMagic`: Manages image transformation.
    - Main Program: Handles input/output and rendering.
- **Error Handling:**
  - Validates command-line arguments to ensure correct input.
  - Checks if the input file is loaded and the output file is saved successfully.
  - Exits gracefully with appropriate error messages in case of issues.

## 2.3 Images Used

Figure 4: Input Image

## 2.4 What I Accomplished

- Designed and implemented the FibLFSR class that accurately simulates a 16-bit Fibonacci Linear Feedback Shift Register with XOR logic.
- Developed and tested the core methods (`step()`, `generate()`, overloaded `<<`) in the `FibLFSR` class.
- Created unit tests for the LFSR's expected behavior using the Boost Test framework to validate both state transitions and sequence generation logic.
- Designed encryption and decryption functionality for image pixel data by XORing RGB values with bits from the generated pseudo-random sequences.
- Integrated SFML to allow image processing visualization for both encryption and decryption processes.
- Designed logic to accept alphanumerics from the user, convert them into binary keys, and initialize the LFSR with these keys.

## 2.5 What I Already Knew

- Familiarity with pseudo-random number generation and LFSRs (Linear Feedback Shift Registers).
- Understanding of cryptographic XOR encryption techniques and how they can apply symmetrically for encryption and decryption.
- Familiarity with object-oriented programming concepts such as encapsulation, modular design, and operator overloading.
- Previous experience using Boost Test for unit testing.
- Understanding the usage of SFML for image rendering and pixel manipulation.

## 2.6 What I Learned

- Learned how to simulate a 16-bit Fibonacci LFSR using XOR feedback mechanisms at specific tap points (0, 2, 3, 5).
- Learned how to debug LFSR logic to ensure pseudo-random number generation matched expectations.
- Strengthened understanding of how encryption can work by XORing pixel values with pseudo-random sequences.
- Learned how pseudo-random sequences are generated using iterative feedback logic with proper LFSR tap points.
- Explored dynamic seed conversion from alphanumerics into binary for custom encryption and decryption keys.
- Strengthened familiarity with SFML for image visualization and manipulation.

### 2.7 Challenges

- Debugging XOR logic in combination with LFSR bit feedback mechanisms to ensure proper pseudo-random number generation.
- Ensuring that the LFSR logic (including state transitions and feedback through XOR) produced the expected sequence results.
- Integrating SFML with the encryption logic while maintaining correct image transformations.
- Handling edge cases with XOR-based transformations to ensure symmetric encryption/decryption.
- Converting alphanumerics into 16-bit binary sequences for use as initial seed values without introducing input errors.

### 2.8 Testing with Boost

Testing was performed using Boost's unit testing framework. The following test cases were defined:

Listing 1: Unit Test Example

```
1  // Test for step function
2  BOOST_AUTO_TEST_CASE(test_step)
3
4  // Test for the << operator output
5  BOOST_AUTO_TEST_CASE(test_operator_output)
6
7  // Test for all-zeros seed
8  BOOST_AUTO_TEST_CASE(test_all_zeros_seed)
9
10 // Test for generating bits
11 BOOST_AUTO_TEST_CASE(test_generate)
12
13 // Test for multiple steps
14 BOOST_AUTO_TEST_CASE(test_multiple_steps)
15
16 // Test for generating 0 bits
17 BOOST_AUTO_TEST_CASE(test_generate_zero)
```

#### Tests Description

- `test_step`: Verifies correct bit generation across multiple steps in the LFSR.
- `test_operator_output`: Confirms that the overloaded operator outputs the internal state correctly.
- `test_all_zeros_seed`: Ensures edge cases are handled where the LFSR is initialized with all zeros.
- `test_generate`: Tests bit generation logic over repeated calls to simulate randomness.
- `test_multiple_steps`: Confirms multi-step sequence generation produces expected numbers.

- `test_generate_zero`: Ensures no errors are thrown when generating 0 bits.

## 2.9 Codebase

**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
```

**Main.cpp**

```
//  CopyRight 2024 Harishwar Reddy Erri

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

// A new feature that transforms an alphanumeric seed into a binary string
std::string alphanumericToBinary(const std::string& seed1) {
    std::string BSeed;
    for (char c : seed1) {
        if (std::isalnum(c)) {
            std::bitset<8> bits(c);
            BSeed += bits.to_string();
```

```
        }
    }
    return BSeed;
}

int main(int argc, char* argv[]) {
    if (argc != 4) {
        std::cerr << "Usage: " << argv[0] << " <input-file> <output-file> <alphanumeric-seed>\n";
        return 1;
    }
    std::string inputFile = argv[1];
    std::string outputFile = argv[2];
    std::string ANSeed = argv[3];
    std::string BSeed = alphanumericToBinary(ANSeed);
    while (BSeed.length() < 16) {
        BSeed += BSeed;
    }
    BSeed = BSeed.substr(0, 16);
    sf::Image image;
    if (!image.loadFromFile(inputFile)) {
        std::cerr << "Error: Could not load input file: " << inputFile << "\n";
        return 1;
    }
    sf::RenderWindow window1(sf::VideoMode(image.getSize().x, image.getSize().y), "Original Image");
    sf::Texture T1;
    T1.loadFromImage(image);
    sf::Sprite sprite1(T1);
    PhotoMagic::FibLFSR lfsr(BSeed);
    PhotoMagic::transform(image, &lfsr);
    if (!image.saveToFile(outputFile)) {
        std::cerr << "Error: Could not save output file: " << outputFile << "\n";
        return 1;
    }
    sf::RenderWindow window2(sf::VideoMode(image.getSize().x,
    image.getSize().y), "Encrypted Image");
    sf::Texture T2;
    T2.loadFromImage(image);
    sf::Sprite sprite2(T2);
    while (window1.isOpen() && window2.isOpen()) {
        sf::Event E;
        while (window1.pollEvent(E)) {
            if (E.type == sf::Event::Closed)
                window1.close();
        }
        while (window2.pollEvent(E)) {
            if (E.type == sf::Event::Closed)
                window2.close();
```

```
        }
        window1.clear();
        window1.draw(sprite1);
        window1.display();
        window2.clear();
        window2.draw(sprite2);
        window2.display();
    }
    return 0;
}
```

**FibLFSR.hpp**

```
//  CopyRight 2024 Harishwar Reddy Erri

#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 rs;
    int get(char a);
    int xOP(int y, int z);
};
}  //  namespace PhotoMagic
```

**FibLFSR.cpp**

```
// CopyRight 2024 Harishwar Reddy Erri

#include <math.h>
#include <iostream>
#include <string>
#include "FibLFSR.hpp"

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

```

```cpp
int FibLFSR::get(char s) {
    if (s == '1') {
        return 1;
    } else if (s == '0') {
        return 0;
    } else {
        return -1;
    }
}

FibLFSR::FibLFSR(const std::string& seed) {
    rs = seed;
}

int FibLFSR::xOP(int y, int z) {
    return y != z;
}

int FibLFSR::step() {
    std::string new_rs = rs.substr(1);
    int value = xOP(get(rs[0]), get(rs[2]));
    value = xOP(value, get(rs[3]));
    value = xOP(value, get(rs[5]));
    rs = new_rs;
    rs += std::to_string(value);
    return value;
}

int FibLFSR::generate(int k) {
    int res = 0;
    for (int i = 0; i < k; i++) {
        int x = step();
        res = (res * 2) + x;
    }
    return res;
}

}  //  namespace PhotoMagic
```

**PhotoMagic.hpp**

```cpp
//  CopyRight 2024 Harishwar Reddy Erri

#ifndef PHOTOMAGIC_HPP
#define PHOTOMAGIC_HPP

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

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

```
}

#endif
```

**PhotoMagic.cpp**

```
//  CopyRight 2024 Harishwar Reddy Erri

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

namespace PhotoMagic {
    void 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) {
                // Get pixel color
                sf::Color pixel = image.getPixel(x, y);

                // XOR the red, green, and blue channels with LFSR
                pixel.r ^= lfsr->generate(8);
                pixel.g ^= lfsr->generate(8);
                pixel.b ^= lfsr->generate(8);

                // Set transformed pixel back into the image
                image.setPixel(x, y, pixel);
            }
        }
    }
}  // namespace PhotoMagic
```

**test.cpp**

```
// Copyright 2024 Harishwar Reddy

#include <iostream>
#include <sstream>
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE
#include <boost/test/unit_test.hpp>
#include "FibLFSR.hpp"
using PhotoMagic::FibLFSR;

// Test for step function
BOOST_AUTO_TEST_CASE(test_step) {
    FibLFSR lfsr("1011011000110110");
    BOOST_CHECK_EQUAL(lfsr.step(), 0);
    BOOST_CHECK_EQUAL(lfsr.step(), 0);
    BOOST_CHECK_EQUAL(lfsr.step(), 0);
    BOOST_CHECK_EQUAL(lfsr.step(), 1);
```

```
18      BOOST_CHECK_EQUAL(lfsr.step(), 1);
19  }
20
21  // Test for the << operator output
22  BOOST_AUTO_TEST_CASE(test_operator_output) {
23      FibLFSR lfsr("1011011000110110");
24      std::ostringstream oss;
25      oss << lfsr;
26      BOOST_CHECK_EQUAL(oss.str(), "1011011000110110");
27  }
28
29  // Test for all-zeros seed
30  BOOST_AUTO_TEST_CASE(test_all_zeros_seed) {
31      FibLFSR lfsr("0000000000000000");
32      BOOST_CHECK_EQUAL(lfsr.step(), 0);  // All bits are 0, expect 0
33      BOOST_CHECK_EQUAL(lfsr.generate(8), 0);  // Generate 8 bits, all 0
34  }
35
36  // Test for generating bits
37  BOOST_AUTO_TEST_CASE(test_generate) {
38      FibLFSR lfsr("1011011000110110");
39      BOOST_CHECK_EQUAL(lfsr.generate(5), 3);  // Adjusted due to new tap
        configuration
40      BOOST_CHECK_EQUAL(lfsr.generate(5), 6);  // Adjusted due to new tap
        configuration
41  }
42
43  // Test for multiple steps
44  BOOST_AUTO_TEST_CASE(test_multiple_steps) {
45      FibLFSR lfsr("0110100001100110");
46      BOOST_CHECK_EQUAL(lfsr.generate(7), 67);  // Adjusted to reflect new tap
        bits
47      BOOST_CHECK_EQUAL(lfsr.generate(9), 24);  // Adjusted to reflect new tap
        bits
48  }
49
50  // Test for generating 0 bits
51  BOOST_AUTO_TEST_CASE(test_generate_zero) {
52      FibLFSR lfsr("1111000011110000");
53      BOOST_CHECK_EQUAL(lfsr.generate(0), 0);  // Generating 0 bits should still
         return 0
54  }
```

## 2.10 Results

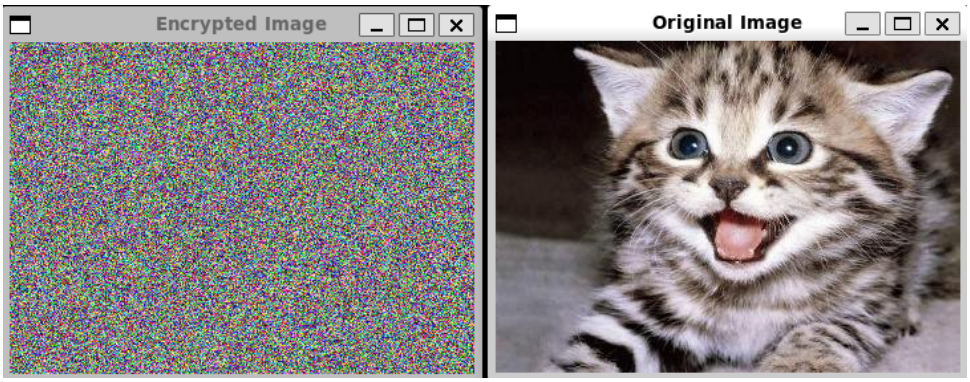


Figure 5: After Encryption of the Image.

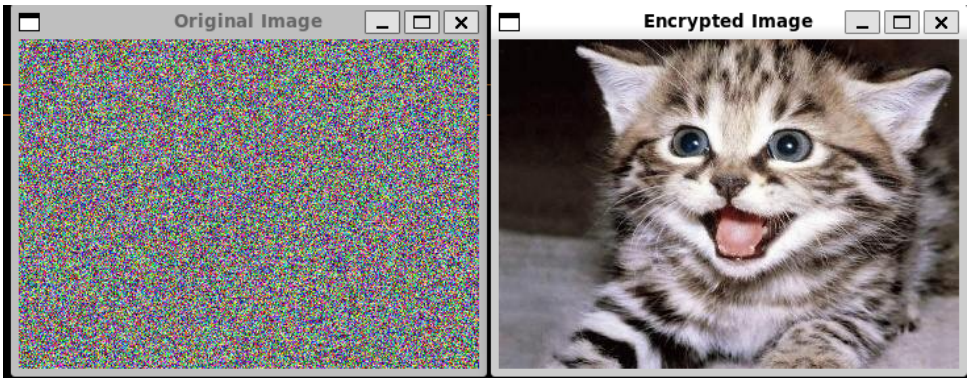


Figure 6: After Decryption of the Image.

# 3 PS2: Pentaflake

## 3.1 Discussion

This project was about creating a recursive fractal called a pentaflake, which is generated by subdividing a pentagon into smaller pentagons recursively. This fractal illustrates the elegance of recursive algorithms and mathematical symmetry, leveraging concepts like the golden ratio and polygonal geometry.
The assignment had several key objectives:

- **Mathematical Calculations:** Determining the sizes and positions of the smaller pentagons using geometric relationships, particularly the *golden ratio*.
- **Recursive Design:** Implementing a recursive function to compute the fractal structure up to a user-defined depth (`N`).
- **SFML Integration:** Leveraging SFML to render the fractal visually with graphical transformations such as scaling, rotation, and positioning.
- **User Input:** Accepting two command-line arguments:
  - `L`: Side length of the base pentagon.
  - `N`: Recursion depth.
- **Customization Options:** Introducing additional features for extra credit, such as multiple colors, rotation parameters, and animations.

The goal was to blend mathematics, recursion, and graphics programming to produce a visually appealing and technically robust program.

## 3.2 Key Algorithms, Data Structures, and OO Designs Used

**Key Algorithms**

- **Fractal Generation (Recursive Algorithm)**
  - **Base Case**: When depth == 0, a pentagon is drawn using createPolygon().
  - **Recursive Case**: Divide the current pentagon into five smaller pentagons:
    1. Calculate the scaleFactor and positions of the smaller pentagons.
    2. Use trigonometric calculations to position each sub-pentagon relative to the main one.
    3. Recursively call drawFractal() with reduced depth.
- **Color Interpolation (LERP)**
  - **Method**: lerpColor()
  - **Purpose**: Smooth transition between colors.
  - **Logic**:
    1. Converts RGB to HSL (Hue, Saturation, Lightness).
    2. Linearly interpolates $h, s, l$ values.
    3. Converts interpolated HSL values back to RGB.
- **Affine Transformations**
  - **Rotation**: rotate(double angle) modifies the fractal's total rotation.
  - **Scaling**: Zooming is applied using affine transformations (sf::Transform::scale()).

**Data Structures**

- **Fractal Elements Container**: A std::vector¡sf::ConvexShape¿ stores all pentagonal shapes.
- **Color Palette**: An array of sf::Color[] is used to store predefined colors for fractal heights.

**Object-Oriented Design (OO Design)**

The code utilizes object-oriented principles for encapsulation and modularity:

- **Encapsulation:**
  - **Class**: PentaflakeDrawer encapsulates attributes and methods related to fractal generation, transformations, and audio state management.
  - **Attributes**: Fields such as baseSize, recursionCount, totalRotation are private, ensuring that the internal state is not exposed directly.
  - **Methods**: Methods like create(), drawFractal(), rotate() and update() allow controlled interaction with the fractal's state.
- **Modularity:**
  - Separate concerns into methods such as drawFractal() for fractal generation, rotate() for rotation management, and updateAudio() for sound control.
  - Audio Management: Methods like loadMusic() and updateAudio() handle sound aspects.

## 3.3 What I Accomplished

- **Core Fractal Generation:**
  - Implemented the recursive logic to generate and display the pentaflake.
  - Correctly sized and positioned child pentagons using geometric formulas based on the golden ratio.
  - Ensured accurate orientation of pentagons, including flipping the center pentagon by 180°.
- **Graphical Rendering:**
  - Used `sf::ConvexShape` from SFML to draw pentagons with accurate angles and positions.
  - Dynamically adjusted the SFML window size to fit the pentaflake, ensuring no part of the fractal extended beyond the boundaries.
- **Parameter Handling:**
  - Properly validated user inputs for `L` and `N`.
  - Handled edge cases for very small or very large recursion depths.
- **Extra Features:**
  - Added multiple colors at different recursion levels for enhanced visual appeal.
  - Implemented an optional rotation parameter for the overall pentaflake.
- **Code Organization:**

  - Divided the program into modular files: `penta.cpp`, `penta.hpp`, and `main.cpp`.
  - Encapsulated recursive logic within a class that derived from `sf::Drawable`.

- **Documentation and Testing:**
  - Created a clear and concise `Readme` file to explain functionality and extra features.
  - Tested the program with various inputs to ensure robustness and correctness.

### 3.4 What I Learned

- **Recursive Programming:**
  - Deepened my understanding of recursion for solving hierarchical and self-similar problems.
  - Learned to manage recursion depth effectively to prevent performance issues.
- **Mathematics in Graphics:**
  - Applied geometric transformations, such as scaling and rotation, to design fractals.
  - Used trigonometric functions and the golden ratio to calculate precise positions and sizes.
- **SFML Graphics Programming:**
  - Mastered the use of `sf::ConvexShape` for drawing custom shapes like pentagons.
  - Gained experience in handling transformations and origin adjustments for shape positioning.
- **User Input and Error Handling:**
  - Improved my ability to validate and process command-line arguments.
- **Aesthetic Design:**
  - Experimented with color gradients and symmetry to enhance visual appeal.
  - Balanced mathematical precision with creative design elements.
- **Optimized Rendering:**
  - Learned to optimize rendering processes to improve performance.

### 3.5 Challenges

- **Geometric Precision:**
  - Calculating the positions of child pentagons required precise use of trigonometry.
  - Avoiding overlap or gaps due to floating-point precision issues was a challenge.
- **Managing Recursion Depth:**
  - Large recursion depths caused performance bottlenecks due to exponential growth in shapes.
  - Balancing visual complexity with computational efficiency required careful tuning.
- **Window Sizing:**

  - Dynamically sizing the SFML window to fit fractals of varying dimensions was non-trivial.
  - Maintaining proportions for small or large values of `L` was complex.

- **Extra Credit Features:**
  - Adding rotation required consistent application of transformations to both parent and child pentagons.
  - Implementing meaningful color gradients across recursion levels involved experimenting with color spaces.

- **Debugging Recursive Logic:**
  - Identifying errors in the recursive function, such as incorrect positioning or orientation, was time-consuming.

- **Math-to-Graphics Mapping:**
  - Translating mathematical calculations into graphical representations required understanding both coordinate systems and SFML's rendering nuances.

### 3.6 Codebase

**Makefile**

```
# Makefile for Pentaflake

CXX = g++
CXXFLAGS = -std=c++17 -Wall -Werror -pedantic
LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
TARGET = Penta

all: $(TARGET)

$(TARGET): main.o penta.o
    $(CXX) $(CXXFLAGS) main.o penta.o -o $(TARGET) $(LIBS)

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

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

clean:
    rm -f *.o $(TARGET)
```

**main.cpp**

```
// CopyRight 2024 Harishwar Reddy Erri

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

```
#include "penta.hpp"

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0]
        << " <base_length> <recursion_levels>" << std::endl;
        return EXIT_FAILURE;
    }

    double baseSize = std::stod(argv[1]);
    int recursionLevels = std::stoi(argv[2]);

    sf::RenderWindow fractalWindow(sf::VideoMode(500, 500),
    "Pentaflake Generator");
    PentaflakeDrawer fractal(baseSize, recursionLevels);
    fractal.create(fractalWindow.getSize().x / 2.0,
    fractalWindow.getSize().y / 2.0);
    fractal.loadMusic();  // Load music files

    sf::Clock clock;

    while (fractalWindow.isOpen()) {
        sf::Time deltaTime = clock.restart();

        sf::Event userEvent;
        while (fractalWindow.pollEvent(userEvent)) {
            if (userEvent.type == sf::Event::Closed) {
                fractalWindow.close();
            }
            if (userEvent.type == sf::Event::KeyPressed) {
                switch (userEvent.key.code) {
    case sf::Keyboard::R:
        fractal.toggleRotation(true);
        break;
    case sf::Keyboard::L:
        fractal.toggleRotation(false);
        break;
    case sf::Keyboard::O:
        fractal.reset();
        break;
    case sf::Keyboard::X:
        fractal.toggleXRotation();
        break;
    case sf::Keyboard::Y:
        fractal.toggleYRotation();
        break;
    case sf::Keyboard::Q:
        fractal.zoomIn();
        break;
    case sf::Keyboard::W:
        fractal.zoomOut();
```

```
56          break;
57      default:
58          // Do nothing for other keys
59          break;
60  }
61              }
62          }
63
64          fractal.update(deltaTime.asSeconds());
65          fractal.updateAudio();
66
67          fractalWindow.clear(sf::Color::Black);
68          fractalWindow.draw(fractal);
69          fractalWindow.display();
70      }
71
72      return EXIT_SUCCESS;
73  }
```

**penta.hpp**

```
1   // CopyRight 2024 Harishwar Reddy Erri
2
3   #ifndef PENTA_HPP
4   #define PENTA_HPP
5
6   #include <vector>
7   #include <SFML/Graphics.hpp>
8   #include <SFML/Audio.hpp>
9
10  class PentaflakeDrawer : public sf::Drawable {
11   private:
12        double baseSize;
13        int recursionCount;
14        double centerX, centerY;
15        double pentaflakeTop, pentaflakeBottom;
16        std::vector<sf::ConvexShape> fractalElements;
17        double totalRotation;
18        bool isRotatingClockwise;
19        bool isRotatingCounterclockwise;
20        double rotationSpeed;
21        double xRotationAngle, yRotationAngle;
22        bool isXRotating, isYRotating;
23        double xRotationSpeed, yRotationSpeed;
24        double zoomFactor;
25
26        sf::Music normalMusic;
27        sf::Music rightRotationMusic;
28        sf::Music leftRotationMusic;
29
30        void drawFractal(double x, double y, double edge,
```

```
31        int depth, double rotationAngle);
32        sf::ConvexShape createPolygon(double x, double y,
33        double length, double rotationAngle) const;
34        sf::Color determineColor(double y) const;
35        sf::Color lerpColor(const sf::Color& c1,
36        const sf::Color& c2, double t) const;
37
38    public:
39        PentaflakeDrawer(double size, int levels);
40        void create(double centerX, double centerY);
41        void draw(sf::RenderTarget& target,
42        sf::RenderStates states) const override;
43        void rotate(double angle);
44        void update(float deltaTime);
45        void toggleRotation(bool clockwise);
46        void toggleXRotation();
47        void toggleYRotation();
48        void loadMusic();
49        void updateAudio();
50        void reset();
51        void zoomIn();
52        void zoomOut();
53    };
54    #endif  //  PENTA_HPP
```

**penta.cpp**

```
1   // CopyRight 2024 Harishwar Reddy Erri
2
3   #include <cmath>
4   #include <algorithm>
5   #include <tuple>
6   #include "penta.hpp"
7
8   PentaflakeDrawer::PentaflakeDrawer(double size, int levels)
9       : baseSize(size), recursionCount(levels), totalRotation(0.0),
10        isRotatingClockwise(false), isRotatingCounterclockwise(false),
      rotationSpeed(M_PI / 2),
11        xRotationAngle(0.0), yRotationAngle(0.0), isXRotating(false),
      isYRotating(false),
12        xRotationSpeed(M_PI / 4), yRotationSpeed(M_PI / 4), zoomFactor(1.0) {}
13
14  void PentaflakeDrawer::create(double centerX, double centerY) {
15      fractalElements.clear();
16      this->centerX = centerX;
17      this->centerY = centerY;
18      pentaflakeTop = centerY - baseSize / 2.0;
19      pentaflakeBottom = centerY + baseSize / 2.0;
20      drawFractal(centerX, centerY, baseSize, recursionCount, -M_PI / 2.0);
21  }
22
```

```
23 void PentaflakeDrawer::draw(sf::RenderTarget& renderTarget, sf::RenderStates
       states) const {
24     sf::Transform xRotation, yRotation, zoom;
25     double xScale = std::abs(std::cos(xRotationAngle));
26     double yScale = std::abs(std::cos(yRotationAngle));
27     xRotation.scale(1.f, xScale, centerX, centerY);
28     yRotation.scale(yScale, 1.f, centerX, centerY);
29     zoom.scale(zoomFactor, zoomFactor, centerX, centerY);
30 
31     sf::Transform rotation;
32     rotation.rotate(totalRotation * 180 / M_PI, centerX, centerY);
33 
34     states.transform *= zoom * xRotation * yRotation * rotation;
35 
36     for (const auto& element : fractalElements) {
37         renderTarget.draw(element, states);
38     }
39 }
40 
41 void PentaflakeDrawer::rotate(double angle) {
42     totalRotation += angle;
43     totalRotation = fmod(totalRotation, 2 * M_PI);
44 }
45 
46 void PentaflakeDrawer::update(float deltaTime) {
47     if (isRotatingClockwise) {
48         rotate(rotationSpeed * deltaTime);
49     }
50     if (isRotatingCounterclockwise) {
51         rotate(-rotationSpeed * deltaTime);
52     }
53     if (isXRotating) {
54         xRotationAngle += xRotationSpeed * deltaTime;
55         xRotationAngle = std::fmod(xRotationAngle, 2 * M_PI);
56     }
57     if (isYRotating) {
58         yRotationAngle += yRotationSpeed * deltaTime;
59         yRotationAngle = std::fmod(yRotationAngle, 2 * M_PI);
60     }
61 }
62 
63 void PentaflakeDrawer::toggleRotation(bool clockwise) {
64     if (clockwise) {
65         isRotatingClockwise = !isRotatingClockwise;
66         isRotatingCounterclockwise = false;
67     } else {
68         isRotatingCounterclockwise = !isRotatingCounterclockwise;
69         isRotatingClockwise = false;
70     }
71 }
72 
```

```
73  void PentaflakeDrawer::toggleXRotation() {
74      isXRotating = !isXRotating;
75  }
76
77  void PentaflakeDrawer::toggleYRotation() {
78      isYRotating = !isYRotating;
79  }
80
81  void PentaflakeDrawer::
82  drawFractal(double x, double y, double edge, int depth, double rotationAngle)
        {
83      if (depth == 0) {
84          sf::ConvexShape pentagon = createPolygon(x, y, edge, rotationAngle);
85          pentagon.setFillColor(determineColor(y));
86          fractalElements.push_back(pentagon);
87          return;
88      }
89
90      double scaleFactor = edge / (1 + (1 + std::sqrt(5.0)) / 2.0);
91      double outerRadius = edge / (2 * std::sin(M_PI / 5));
92      double innerOffset = scaleFactor / (2 * std::sin(M_PI / 5));
93
94      drawFractal(x, y, scaleFactor, depth - 1, rotationAngle);
95
96      for (int i = 0; i < 5; ++i) {
97          double angle = (2 * M_PI * i) / 5;
98          double offsetX = (outerRadius - innerOffset) * std::cos(angle);
99          double offsetY = (outerRadius - innerOffset) * std::sin(angle);
100         double rotatedX = x + offsetX * std::cos(rotationAngle) - offsetY *
        std::sin(rotationAngle);
101         double rotatedY = y + offsetX * std::sin(rotationAngle) + offsetY *
        std::cos(rotationAngle);
102         drawFractal(rotatedX, rotatedY, scaleFactor, depth - 1, rotationAngle)
        ;
103     }
104 }
105
106 sf::ConvexShape PentaflakeDrawer::
107 createPolygon(double x, double y, double length, double rotationAngle) const {
108     sf::ConvexShape pentagon;
109     pentagon.setPointCount(5);
110     double radius = length / (2 * std::sin(M_PI / 5));
111     for (int vertex = 0; vertex < 5; ++vertex) {
112         double angle = (2 * M_PI * vertex) / 5;
113         double rotatedX = x + radius * std::cos(angle) *
114         std::cos(rotationAngle) - radius * std::sin(angle) * std::sin(
        rotationAngle);
115         double rotatedY = y + radius * std::cos(angle) *
116         std::sin(rotationAngle) + radius * std::sin(angle) * std::cos(
        rotationAngle);
117         pentagon.setPoint(vertex, sf::Vector2f(rotatedX, rotatedY));
```

```
118     }
119     return pentagon;
120 }
121 
122 sf::Color PentaflakeDrawer::determineColor(double y) const {
123     double height = pentaflakeBottom - pentaflakeTop;
124     double relativeY = (y - pentaflakeTop) / height;
125     sf::Color colors[5] = {
126         sf::Color(255, 182, 193),  // Light Pink
127         sf::Color(255, 218, 185),  // Peach Puff
128         sf::Color(255, 255, 224),  // Light Yellow
129         sf::Color(176, 224, 230),  // Powder Blue
130         sf::Color(230, 230, 250)   // Lavender
131     };
132     int index = static_cast<int>(relativeY * 4);
133     double t = relativeY * 4 - index;
134     return lerpColor(colors[index], colors[index + 1], t);
135 }
136 
137 sf::Color PentaflakeDrawer::lerpColor(const sf::Color& c1, const sf::Color& c2
    , double t) const {
138     auto rgbToHsl = [](const sf::Color& c) {
139         float r = c.r / 255.0f, g = c.g / 255.0f, b = c.b / 255.0f;
140         float max = std::max({r, g, b}), min = std::min({r, g, b});
141         float h, s, l = (max + min) / 2;
142 
143         if (max == min) {
144             h = s = 0;
145         } else {
146             float d = max - min;
147             s = l > 0.5f ? d / (2 - max - min) : d / (max + min);
148             if (max == r)
149                 h = (g - b) / d + (g < b ? 6 : 0);
150             else if (max == g)
151                 h = (b - r) / d + 2;
152             else
153                 h = (r - g) / d + 4;
154             h /= 6;
155         }
156         return std::make_tuple(h, s, l);
157     };
158 
159     auto hslToRgb = [](float h, float s, float l) {
160         auto hue2rgb = [](float p, float q, float t) -> float {
161             if (t < 0) t += 1;
162             if (t > 1) t -= 1;
163             if (t < 1.0f/6) return p + (q - p) * 6 * t;
164             if (t < 1.0f/2) return q;
165             if (t < 2.0f/3) return p + (q - p) * (2.0f/3 - t) * 6;
166             return p;
167         };
```

```
168
169         if (s == 0) {
170             return sf::Color(l * 255, l * 255, l * 255);
171         } else {
172             float q = l < 0.5f ? l * (1 + s) : l + s - l * s;
173             float p = 2 * l - q;
174             return sf::Color(
175                 hue2rgb(p, q, h + 1.0f/3) * 255,
176                 hue2rgb(p, q, h) * 255,
177                 hue2rgb(p, q, h - 1.0f/3) * 255);
178         }
179     };
180
181     auto [h1, s1, l1] = rgbToHsl(c1);
182     auto [h2, s2, l2] = rgbToHsl(c2);
183
184     float h = h1 + t * (h2 - h1);
185     float s = s1 + t * (s2 - s1);
186     float l = l1 + t * (l2 - l1);
187
188     return hslToRgb(h, s, l);
189 }
190
191 void PentaflakeDrawer::loadMusic() {
192     if (!normalMusic.openFromFile("normal.ogg") ||
193         !rightRotationMusic.openFromFile("RR.ogg") ||
194         !leftRotationMusic.openFromFile("LR.ogg")) {
195         // Handle error loading music files
196     }
197     normalMusic.setLoop(true);
198     rightRotationMusic.setLoop(true);
199     leftRotationMusic.setLoop(true);
200 }
201
202 void PentaflakeDrawer::updateAudio() {
203     if (isRotatingClockwise) {
204         if (normalMusic.getStatus() == sf::Music::Playing) normalMusic.stop();
205         if (leftRotationMusic.getStatus() == sf::Music::Playing)
    leftRotationMusic.stop();
206         if (rightRotationMusic.getStatus() != sf::Music::Playing)
    rightRotationMusic.play();
207     } else if (isRotatingCounterclockwise) {
208         if (normalMusic.getStatus() == sf::Music::Playing) normalMusic.stop();
209         if (rightRotationMusic.getStatus() == sf::Music::Playing)
    rightRotationMusic.stop();
210         if (leftRotationMusic.getStatus() != sf::Music::Playing)
    leftRotationMusic.play();
211     } else {
212         if (rightRotationMusic.getStatus() == sf::Music::Playing)
    rightRotationMusic.stop();
213         if (leftRotationMusic.getStatus() == sf::Music::Playing)
```

```
    leftRotationMusic.stop();
214         if (normalMusic.getStatus() != sf::Music::Playing) normalMusic.play();
215     }
216 }
217
218 void PentaflakeDrawer::reset() {
219     totalRotation = 0.0;
220     xRotationAngle = 0.0;
221     yRotationAngle = 0.0;
222     isRotatingClockwise = false;
223     isRotatingCounterclockwise = false;
224     isXRotating = false;
225     isYRotating = false;
226     zoomFactor = 1.0;
227     rightRotationMusic.stop();
228     leftRotationMusic.stop();
229     normalMusic.stop();
230     normalMusic.play();
231 }
232
233 void PentaflakeDrawer::zoomIn() {
234     zoomFactor *= 1.1;
235 }
236
237 void PentaflakeDrawer::zoomOut() {
238     zoomFactor /= 1.1;
239 }
```

### 3.7 Results

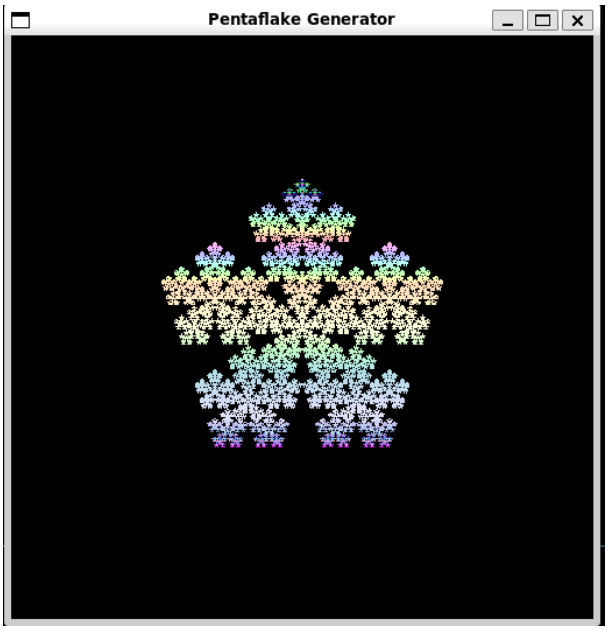


Figure 7: After Decryption of the Image.

# 4 PS3: N-Body Simulation

## 4.1 Discussion

### N-Body Simulation Project Overview

The N-Body Simulation project entailed developing a physics simulation and visualization of celestial bodies subject to reciprocal gravitational pulls. It combines physics computations and graphical simulation with rendering and audio provided by SFML (Simple and Fast Multimedia Library). The objective was divided into two primary sections:

- **PS3a (Static depiction):**
  - Using initial velocities, masses, and positions supplied in an input file, the simulation produced a depiction of the starting positions of celestial bodies.
- **PS3b (Dynamic Simulation):**
  - Using numerical integration techniques like the leapfrog system, PS3b animated the motion of the celestial bodies over time.
  - Incorporated real-time physics computations to simulate gravitational forces.

### Simulation Overview

The simulation updates the bodies' locations and velocities at each time step, integrating Newtonian mechanics to calculate gravitational interactions between them. It includes graphical components such as backdrop pictures, celestial body sprites, and audio features. Command-line options allow you to change the simulation's parameters. This physics-based, modular method provides a visualization tool that is perfect for learning numerical approximations, simulation visualization, and gravitational interactions.

- **Time-stepping Simulation:**
  - Updates bodies' locations and velocities at each discrete time step.
  - Integrates Newtonian mechanics to simulate gravitational interactions.
- **Graphical Features:**
  - Includes backdrop images for space visualization.
  - Celestial body sprites represent planets and other objects.
- **Audio Integration:**
  - Incorporates audio features to enhance user experience.
- **Command-Line Options:**
  - Allows users to configure and modify simulation parameters.
- **Educational Utility:**
  - Visualization tool designed for understanding numerical approximations.
  - Illustrates gravitational interactions and simulation concepts.

## 4.2 Key Algorithms, Data Structures, and OO Designs Used

### Key Algorithms

- **Force Calculation (Newton's Law of Gravitation)** The gravitational force between celestial bodies is calculated using Newton's law of gravitation:

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

  **Implementation:**

  - The function `calculateForceFrom` in the `CelestialBody` class calculates the gravitational force exerted by another celestial body.
  - The `Universe::calculateForce` function applies Newton's third law (equal and opposite forces) to pairs of celestial bodies.

  **Optimization:**

  - Forces are computed for each pair of bodies only once.
  - `delta / distance` normalizes the force direction.

- **Velocity and Position Update** The simulation updates each body's position and velocity based on forces acting on it and a small timestep $dt$:
  The velocity of a body is updated as follows:

$$\mathbf{v} = \mathbf{v} + \frac{\mathbf{F}}{m}\Delta t$$

  This is implemented in the method `CelestialBody::updateVelocity`.
  The position of a body is updated as follows:

$$\mathbf{x} = \mathbf{x} + \mathbf{v}\Delta t$$

  This is implemented in the method `CelestialBody::updatePosition`.

  **Integration Scheme:** The simulation uses an explicit Euler method for numerical integration.

- **Simulation Time Progression** The simulation steps forward in time by repeatedly calling `Universe::step(dt)` until the total simulation time $T$ is reached.

  - Forces are first calculated for all celestial bodies.
  - Velocities and positions are then updated for each body.

- **Rendering Celestial Bodies** The `Universe::draw` function maps celestial body positions to window coordinates using scaling:

  - $x$ and $y$ are scaled from the simulation's universe radius to fit the display window.
  - Celestial bodies are represented as sprites centered on their positions.

- **Elapsed Time Display** Elapsed time is tracked using `std::chrono` and displayed in real-time using `sf::Text`.

  - Time measurement avoids blocking the simulation and updates dynamically.

**Data Structures**

- `CelestialBody` **Class Purpose:** Represents individual celestial bodies (planets, stars, etc.).
  - **Attributes:**
    - `position`: A 2D vector (`sf::Vector2f`) for the body's coordinates.
    - `velocity`: A 2D vector for velocity.
    - `mass`: A scalar value representing mass.
    - `filename`: A string for the texture file used in rendering.
  - **Methods:**
    - Getters for all attributes.
    - Methods for updating position and velocity.
- `Universe` **Class Purpose:** Manages the simulation of the system of celestial bodies.
  - **Attributes:**
    - `radius`: The simulation space radius.
    - `bodies`: A `std::vector` of pointers to `CelestialBody` objects.
  - **Methods:**
    - `readFromFile` and `writeToFile` handle I/O operations.
    - `step` updates positions and velocities of all celestial bodies.
    - `draw` renders celestial bodies onto the SFML window.
- **SFML Data Structures**
  - `sf::Vector2f`: Used for 2D vectors (position, velocity, and force).
  - `sf::Sprite` and `sf::Texture`: Represent celestial bodies graphically and handle textures.

**Object-Oriented Design Principles**

- **Encapsulation**
  - Each class encapsulates its state and behavior.
  - Example: `CelestialBody` encapsulates properties and physics of a single celestial body.
- **Abstraction** Complex details are hidden behind clean interfaces:
  - `CelestialBody::calculateForceFrom` abstracts gravitational interaction.
  - `Universe::draw` abstracts graphical rendering logic.
- **Modularity**
  - Code is divided into self-contained classes (`CelestialBody`, `Universe`) and files.
  - This makes the code easier to maintain and extend.
- **Operator Overloading** Stream operators (`>>` and `<<`) simplify I/O for `CelestialBody` and `Universe`.
- **Resource Management**
  - Use of `std::unique_ptr` ensures safe memory management and avoids leaks.

### 4.3 Images Used

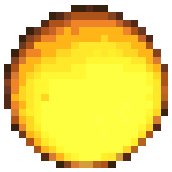

Figure 8: sun

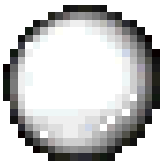

Figure 9: venus

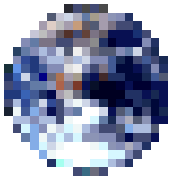

Figure 10: earth

Figure 11: mars

Figure 12: Background Image

### 4.4 What I Accomplished

- **Implemented Static Visualization (PS3a)**
  - Created an SFML window to visually display celestial bodies at their initial positions using provided input data.
  - Loaded textures dynamically and rendered the bodies based on their positions.
  - Designed viewport transformations so that the simulation mapped celestial coordinates correctly to screen positions.
- **Built Dynamic Simulation Capabilities (PS3b)**
  - Simulated gravitational forces between bodies using pairwise force calculations and numerical integration.
  - Visualized motion over time using a leapfrog numerical integration scheme to update positions and velocities at each time step.
- **Enhanced User Experience**
  - Integrated background music toggling with the M key.

  - Added a space-themed background image for better visuals.

- **Created Unit Tests**
  - Verified core simulation logic (force calculations, simulation updates) and rendering logic.

- **Command-line Parameterization**
  - Designed the simulation to accept parameters like total simulation time and time step, allowing the user to customize the simulation.

### 4.5 What I Already Knew

- **Basic Physics Simulation:** I was already familiar with concepts like Newton's law of gravitation and how gravitational force can be calculated pairwise using masses and distances.
- **SFML Graphics Basics:** I had some prior experience using SFML to draw 2D graphics (e.g., rendering sprites, handling input events).
- **Object-Oriented Programming Principles:** I was familiar with class design patterns, inheritance, encapsulation, operator overloading, and modularization using namespaces.
- **File I/O in C++:** Reading structured data from input files and handling custom parsing logic was something I had encountered before.

### 4.6 What I Learned

- **Numerical Methods for Physics Simulation:** I discovered how to use numerical integration techniques, such as the leapfrog system, to model gravitational motion over time. This required calculating gravity accelerations and adjusting locations appropriately.
- **Time-Stepping's Function in Simulations:** I discovered how the accuracy and performance of the simulation are impacted by selecting the right time step ($\Delta t$).
- **Improved SFML Techniques:** I improved my ability to handle viewport transformations, control rendering logic in real-time simulations, and dynamically load textures.
- **How Coordinate Transformations Can Be Overcome:** Celestial coordinates (centered at (0,0)) were converted into SFML's screen-space coordinates for the simulation. I discovered that viewport offsets can be a useful tool for efficiently handling these changes.
- **Unit Testing Best Practices:** By verifying distinct components (such as rendering logic and physics computations), I enhanced my method for testing modular code.

### 4.7 Challenges

- **Making Use of Precise Gravitational Calculations:** Pairwise computations are essential to gravitational simulations. It was difficult to optimize these calculations for n-body systems, particularly in terms of performance. Each pairwise computation scales quadratically for greater numbers of bodies.
- **Viewport Modifications:** Careful calculation was needed to map the coordinates of the SFML window screen to the positions of celestial bodies in 2D space. It was crucial to comprehend how viewport offsets operated.

- **Managing Accurate Numerical Data:** Using tiny steps to simulate physics over time created problems with floating-point precision flaws. It required some trial and error to ensure steady motion free of drift or artifacts.
- **Memory Management:** Although raw pointers were enough for the purposes of this simulation, there were difficulties in making sure that memory was cleaned up properly. I discovered that switching to smart pointers will improve security.
- **Leapfrog Method Implementation in Physics Simulations:** Accurately applying the leapfrog method and integrating it with the rendering loop of SFML necessitated maintaining synchronization between rendering updates and calculations.
- **Debugging Timing Issues:** Accurate timing was essential for the time integration loops in simulations. It was difficult to ensure smooth animation over discrete intervals and debug expired simulation timing.

### 4.8 Codebase

**Makefile**

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

all: NBody NBody.a test

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

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

main.o: main.cpp Universe.hpp CelestialBody.hpp
    $(CXX) $(CXXFLAGS) -c $< -o $@

Universe.o: Universe.cpp Universe.hpp CelestialBody.hpp
    $(CXX) $(CXXFLAGS) -c $< -o $@

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

test: test.cpp Universe.o CelestialBody.o
    $(CXX) $(CXXFLAGS) $^ -o $@ $(SFML_LIBS)

lint:
    cpplint *.cpp *.hpp

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

.PHONY: all clean lint
```

**main.cpp**

```cpp
// CopyRight 2024 Harishwar Reddy

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

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " T dt" << std::endl;
        return 1;
    }

    double T = std::stod(argv[1]);
    double dt = std::stod(argv[2]);

    NB::Universe universe;
    std::cin >> universe;

    sf::RenderWindow window(sf::VideoMode(600, 600), "N-Body Simulation");

    // Optional: Disable vertical sync if you're seeing warnings
    window.setVerticalSyncEnabled(false);

    // Load background image
    sf::Texture backgroundTexture;
    if (!backgroundTexture.loadFromFile("image.png")) {
        std::cerr << "Failed to load background image" << std::endl;
        return 1;
    }
    sf::Sprite backgroundSprite(backgroundTexture);
    backgroundSprite.setScale(
        window.getSize().x / backgroundSprite.getLocalBounds().width,
        window.getSize().y / backgroundSprite.getLocalBounds().height);

    // Load music
    sf::Music music;
    if (!music.openFromFile("music.wav")) {
        std::cerr << "Failed to load music" << std::endl;
        return 1;
    }
    bool musicPlaying = false;
    sf::Font font;
    if (!font.loadFromFile("font.ttf")) {
        std::cerr << "Failed to load font" << std::endl;
        return 1;
    }

    // Set up text object for elapsed time display
```

```
    sf::Text elapsedTimeText;
    elapsedTimeText.setFont(font);
    elapsedTimeText.setCharacterSize(24);  // in pixels
    elapsedTimeText.setFillColor(sf::Color::White);
    elapsedTimeText.setPosition(10, 10);  // Top-left corner

    // Start tracking elapsed time
    auto start_time = std::chrono::high_resolution_clock::now();

    double t = 0.0;
    while (window.isOpen() && t < T) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::KeyPressed && event.key.code == sf::
    Keyboard::M) {
                if (musicPlaying) {
                    music.pause();
                    musicPlaying = false;
                } else {
                    music.play();
                    musicPlaying = true;
                }
            }
        }

        universe.step(dt);
        t += dt;

        // Measure elapsed time and update the text
        auto current_time = std::chrono::high_resolution_clock::now();
        std::chrono::duration<double> time_span = current_time - start_time;
        double elapsed_time = time_span.count();

        // Update the displayed text with the elapsed time
        elapsedTimeText.setString("Elapsed Time: " + std::to_string(
    elapsed_time) + " seconds");

        window.clear();
        window.draw(backgroundSprite);
        universe.draw(window);  // Draw celestial bodies
        window.draw(elapsedTimeText);  // Draw elapsed time text
        window.display();
    }

    // After the simulation finishes, print the final universe state
    std::cout << universe;

    return 0;
}
```

**CelestialBody.hpp**

```
1  // CopyRight 2024 Harishwar Reddy
2
3  #pragma once
4  #include <iostream>
5  #include <string>
6  #include <SFML/System/Vector2.hpp>
7
8  namespace NB {
9
10 class CelestialBody {
11  private:
12     sf::Vector2f position;
13     sf::Vector2f velocity;
14     double mass;
15     std::string filename;
16
17  public:
18     CelestialBody(sf::Vector2f pos, sf::Vector2f vel, double m, const std::
       string& file);
19     sf::Vector2f getPosition() const;
20     sf::Vector2f getVelocity() const;
21     double getMass() const;
22     std::string getFilename() const;
23     void updatePosition(double dt);
24     void updateVelocity(sf::Vector2f force, double dt);
25     sf::Vector2f calculateForceFrom(const CelestialBody& other) const;
26
27     friend std::istream& operator>>(std::istream& is, CelestialBody& body);
28     friend std::ostream& operator<<(std::ostream& os, const CelestialBody&
       body);
29 };
30
31 }  //  namespace NB
```

**CelestialBody.cpp**

```
1  // CopyRight 2024 Harishwar Reddy
2
3  #include "CelestialBody.hpp"
4  #include <cmath>
5
6  namespace NB {
7  const double G = 6.67e-11;
8  CelestialBody::CelestialBody(sf::Vector2f pos, sf::Vector2f vel, double m,
       const std::string& file)
9  : position(pos), velocity(vel), mass(m), filename(file) {}
10 sf::Vector2f CelestialBody::getPosition() const {
11 return position;
12 }
```

```
sf::Vector2f CelestialBody::getVelocity() const {
return velocity;
}
double CelestialBody::getMass() const {
return mass;
}
std::string CelestialBody::getFilename() const {
return filename;
}
void CelestialBody::updatePosition(double dt) {
position += velocity * static_cast<float>(dt);
}
void CelestialBody::updateVelocity(sf::Vector2f force, double dt) {
velocity += (force / static_cast<float>(mass)) * static_cast<float>(dt);
}
sf::Vector2f CelestialBody::calculateForceFrom(const CelestialBody& other)
    const {
sf::Vector2f delta = other.position - position;
double distance = std::sqrt(delta.x * delta.x + delta.y * delta.y);
double force = G * mass * other.mass / (distance * distance);
return delta / static_cast<float>(distance) * static_cast<float>(force);
}
std::istream& operator>>(std::istream& is, CelestialBody& body) {
return is >> body.position.x >> body.position.y
>> body.velocity.x >> body.velocity.y
>> body.mass >> body.filename;
}
std::ostream& operator<<(std::ostream& os, const CelestialBody& body) {
return os << body.position.x << " " << body.position.y << " "
<< body.velocity.x << " " << body.velocity.y << " "
<< body.mass << " " << body.filename;
}
} // namespace NB
```

**Universe.hpp**

```
// CopyRight 2024 Harishwar Reddy

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

namespace NB {

class Universe {
 private:
    std::vector<std::unique_ptr<CelestialBody>> bodies;
    double radius;
```

```
16      sf::Vector2f calculateForce(const CelestialBody& body1, const
        CelestialBody& body2) const;
17
18  public:
19      Universe();
20      void readFromFile(std::istream& input);
21      void step(double dt);
22      void draw(sf::RenderWindow& window);
23      void writeToFile(std::ostream& output) const;
24
25      friend std::istream& operator>>(std::istream& is, Universe& universe);
26      friend std::ostream& operator<<(std::ostream& os, const Universe& universe
        );
27  };
28
29  }  //  namespace NB
```

**Universe.cpp**

```
1   // CopyRight 2024 Harishwar Reddy
2
3   #include <cmath>
4   #include <iostream>
5   #include "Universe.hpp"
6
7   namespace NB {
8   const double G = 6.67e-11;
9   Universe::Universe() : radius(0) {}
10  void Universe::readFromFile(std::istream& input) {
11  input >> *this;
12  }
13  void Universe::step(double dt) {
14  std::vector<sf::Vector2f> forces(bodies.size(), sf::Vector2f(0, 0));
15  // Calculate forces
16  for (size_t i = 0; i < bodies.size(); ++i) {
17  for (size_t j = i + 1; j < bodies.size(); ++j) {
18  sf::Vector2f force = calculateForce(*bodies[i], *bodies[j]);
19  forces[i] += force;
20  forces[j] -= force;  // Newton's third law
21  }
22  }
23  // Update velocities and positions
24  for (size_t i = 0; i < bodies.size(); ++i) {
25  bodies[i]->updateVelocity(forces[i], dt);
26  bodies[i]->updatePosition(dt);
27  }
28  }
29  sf::Vector2f Universe::calculateForce(const CelestialBody& body1,
30  const CelestialBody& body2) const {
31  sf::Vector2f delta = body2.getPosition() - body1.getPosition();
32  double distance = std::sqrt(delta.x * delta.x + delta.y * delta.y);
```

```
double force = G * body1.getMass() * body2.getMass() / (distance * distance);
return delta / static_cast<float>(distance) * static_cast<float>(force);
}
void Universe::draw(sf::RenderWindow& window) {
for (const auto& body : bodies) {
sf::Texture texture;
if (!texture.loadFromFile(body->getFilename())) {
std::cerr << "Failed to load texture: " << body->getFilename() << std::endl;
continue;
}
sf::Sprite sprite(texture);
sf::Vector2f pos = body->getPosition();
float windowX = (pos.x / (2 * radius) + 0.5f) * window.getSize().x;
float windowY = (-pos.y / (2 * radius) + 0.5f) * window.getSize().y;
// Center the sprite
sf::FloatRect bounds = sprite.getLocalBounds();
sprite.setOrigin(bounds.width / 2, bounds.height / 2);
sprite.setPosition(windowX, windowY);
// Increase the scale factor to make planets larger
float scale = 1.5f;
// Adjust scale based on the mass of the body (optional)
if (body->getMass() > 1e28) {
scale = 1.5f;
} else if (body->getMass() < 1e24) {
scale = 1.5f;
}
sprite.setScale(scale, scale);
window.draw(sprite);
}
}
void Universe::writeToFile(std::ostream& output) const {
output << *this;
}
std::istream& operator>>(std::istream& is, Universe& universe) {
int numBodies;
is >> numBodies >> universe.radius;
universe.bodies.clear();
for (int i = 0; i < numBodies; ++i) {
CelestialBody body({0, 0}, {0, 0}, 0, "");
is >> body;
universe.bodies.push_back(std::make_unique<CelestialBody>(body));
}
return is;
}
std::ostream& operator<<(std::ostream& os, const Universe& universe) {
os << universe.bodies.size() << "\n";
os << universe.radius << "\n";
for (const auto& body : universe.bodies) {
os << *body << "\n";
}
return os;
```

```
}
}  //  namespace NB
```

**test.cpp**

```
// CopyRight 2024 Harishwar Reddy

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


// Gravitational constant (same as in your Universe.cpp)
const double G = 6.67e-11;

// Test Case 1: Test Position Update
BOOST_AUTO_TEST_CASE(TestPositionUpdate) {
    sf::Vector2f initialPosition(0.0f, 0.0f);
    sf::Vector2f velocity(10.0f, 0.0f);  // Velocity along x-axis
    double mass = 5.0;
    std::string filename = "planet.png";
    NB::CelestialBody body(initialPosition, velocity, mass, filename);
    body.updatePosition(2.0);  // dt = 2 seconds
    sf::Vector2f expectedPosition(20.0f, 0.0f);  // Position = velocity * dt
    BOOST_CHECK_CLOSE(body.getPosition().x, expectedPosition.x, 0.001);
    BOOST_CHECK_CLOSE(body.getPosition().y, expectedPosition.y, 0.001);
}

// Test Case 2: Test Velocity Update with Force
BOOST_AUTO_TEST_CASE(TestVelocityUpdate) {
    sf::Vector2f initialPosition(0.0f, 0.0f);
    sf::Vector2f initialVelocity(0.0f, 0.0f);
    double mass = 10.0;
    std::string filename = "planet.png";
    NB::CelestialBody body(initialPosition, initialVelocity, mass, filename);
    sf::Vector2f force(20.0f, 0.0f);  // Force along x-axis
    body.updateVelocity(force, 2.0);  // dt = 2 seconds
    sf::Vector2f expectedVelocity(4.0f, 0.0f);  // velocity = (force/mass) *
   dt
    BOOST_CHECK_CLOSE(body.getVelocity().x, expectedVelocity.x, 0.001);
    BOOST_CHECK_CLOSE(body.getVelocity().y, expectedVelocity.y, 0.001);
}

// Test Case 3: Test Force Calculation Between Two Bodies
BOOST_AUTO_TEST_CASE(TestForceCalculation) {
    sf::Vector2f pos1(0.0f, 0.0f);
    sf::Vector2f pos2(10.0f, 0.0f);
    sf::Vector2f vel(0.0f, 0.0f);
```

```
46      double mass1 = 5.0e24;
47      double mass2 = 5.0e24;
48      std::string filename = "planet.png";
49      NB::CelestialBody body1(pos1, vel, mass1, filename);
50      NB::CelestialBody body2(pos2, vel, mass2, filename);
51      sf::Vector2f force = body1.calculateForceFrom(body2);
52      double expectedForceMagnitude = G * mass1 * mass2 / (10.0 * 10.0);  // F =
        G*m1*m2/r^2
53      BOOST_CHECK_CLOSE(force.x, expectedForceMagnitude, 0.001);
54      BOOST_CHECK_SMALL(static_cast<double>(force.y), 0.001);  // Convert force.
       y to double
55  }
56
57
58
59  // Test Case 5: Test Universe File I/O
60  BOOST_AUTO_TEST_CASE(TestUniverseFileIO) {
61      std::istringstream input(
62          "3\n"             // Number of bodies
63          "1e12\n"          // Radius of the universe
64          "0 0 0 0 6e24 planet1.png\n"
65          "10 10 0 0 6e24 planet2.png\n"
66          "-10 -10 0 0 6e24 planet3.png\n");
67      NB::Universe universe;
68      input >> universe;
69      std::ostringstream output;
70      universe.writeToFile(output);
71      BOOST_CHECK_EQUAL(output.str(),
72                        "3\n"
73                        "1e+12\n"
74                        "0 0 0 0 6e+24 planet1.png\n"
75                        "10 10 0 0 6e+24 planet2.png\n"
76                        "-10 -10 0 0 6e+24 planet3.png\n");
77  }
```

### 4.9 Result:

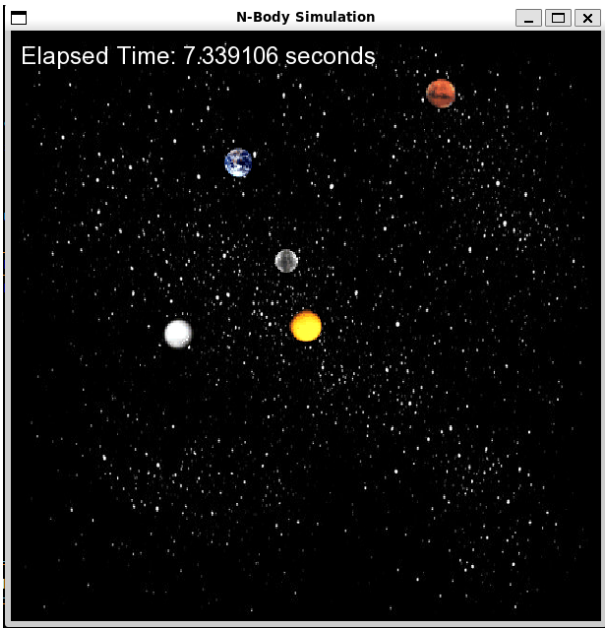


Figure 13: Dynamic Output

# 5 PS4: Sokoban

## 5.1 Discussion

This project demonstrates the journey from foundational programming concepts (file handling, rendering, and simple logic) to integrating sophisticated mechanics, testing, and user experience design. The focus is on understanding how games process and display information in real-time, simulate physical rules (movement restrictions), and provide interactive feedback.

- **PS4a:** enters around interpreting a Sokoban level file, converting it into a playable grid structure, and then drawing it on the screen. This involves decisions about how to store data (e.g., walls, player, boxes, storage locations) and efficiently render each component in its correct position.
- **PS4b:** introduces interactivity: moving the player with keyboard input, handling collisions with walls and boxes, and detecting win conditions. It also involves strategies for error recovery (resetting the level) and extensibility (e.g., extra credit features).

This process mirrors how software evolves: from simple prototypes (display only) to fully interactive systems, highlighting the importance of planning, iteration, and testing.

## 5.2 Key Algorithms, Data Structures, and OO Designs Used

**Key Algorithms**

- **Player Movement:**The player's movement is determined by calculating the change in position (dx, dy) based on the direction.
  - **Validation:**
    - Check if the new position (newX, newY) is within grid bounds.
    - Check the contents of the target cell (newCell).
      - If it's an empty space (.) or storage (a), move the player.
      - If it's a box (A or 1), check the cell beyond the box:
      - If valid, push the box and update its position.
      - If invalid, block the movement.
  - **Edge Cases:**
    - Boxes cannot be pushed into walls or other boxes.
    - The player cannot push more than one box at a time.
  - **Updates:**
    - The grid and player position (mplayerLoc) are updated.
    - The player's sprite texture changes to match the movement direction.
- **Victory Conditions:**
  - **Implementation Details:**
    - Total boxes (boxCount).
    - Total storage locations (targetCount).
    - Boxes correctly placed on storage (boxesOnTarget).

  - **Win Conditions:**
    - All boxes are placed on storage loca
    - If there are no previous states, the undo() method does nothing.tions (boxesOnTarget == boxCount).
    - No storage locations are empty (!hasEmptyTarget).
    - No boxes are left off-target (!hasBoxOffTarget).

- **Undo Functionallity:**
  - The saveState() method ensures the history does not grow indefinitely by limiting the size of the deque to 2.
  - On calling undo(), the most recent state is restored, updating both the grid and player location.
  - If there are no previous states, the undo() method does nothing.

- **Reset Functionallity:**
  - The grid is restored from m initialGrid.
  - The player's position is recalculated by scanning for their initial location (@ or +).
  - The undo history (m undoStates) is cleared.

- **Rendering Logic:**
  - The draw() method iterates through every cell of the grid.
  - An empty space sprite is drawn first.
  - Depending on the cell's contents (#,A, a, etc.), the corresponding sprite is layered on top.
  - The player sprite is drawn separately after all cells to ensure it appears above other elements.

**Data Structures**

- **Grid Representation:**
  - **Type:** `std::vector<char>`
  - **Purpose:** To store the game's state, where each cell represents a specific element (`#`, `A`, `a`, `@`, etc.).
  - **Design Choice:**
    - A 1D vector is used for simplicity and performance.
    - Index calculations:
      - `i = y * m_width + x` to map `(x, y)` coordinates to a 1D index.
      - `x = i % m_width` and `y = i / m_width` for reverse mapping.

- **Player Position:**

  - **Type:** `sf::Vector2i`
  - **Purpose:** To track the player's current location in the grid.
  - **Design Choice:**
    * A lightweight 2D vector type provided by SFML, making it easy to handle positions and sprite coordinates.

- **Undo History:**
  - **Type:** `std::deque<GameState>`
  - **Purpose:** To store a history of game states for implementing undo functionality.
  - **Design Choice:**
    * `std::deque` allows efficient addition and removal from both ends.
    * The `GameState` struct contains:
      · A snapshot of the grid (`std::vector<char>`).
      · The player's position (`sf::Vector2i`).

- **Sprites and Textures:**
  - **Type:**
    * `sf::Texture[]`: An array of textures for different grid elements (e.g., walls, boxes, etc.).
    * `sf::Sprite[]`: An array of sprites, each linked to a corresponding texture.
  - **Purpose:** To render game elements with the appropriate visuals.
  - **Design Choice:**
    * Textures are loaded once and shared among sprites for efficiency.

**O-O Designs**

- **Encapsulation:** The Sokoban class encapsulates all game logic, state, and rendering functionality.This ensures modularity, making it easier to extend or modify the class without affecting other parts of the program.
- **Polymorphism:** The Sokoban class inherits from sf::Drawable and overrides its draw() method.This allows the Sokoban object to be seamlessly drawn in the SFML rendering pipeline, adhering to SFML's design principles.
- **Enums for Direction:** Direction is an enum class (Up, Down, Left, Right).Enums provide type safety and improve code readability, avoiding hardcoded direction values.
- **Operator Overloading:** The $\gg$ operator is overloaded to read level files into the Sokoban object.This simplifies file parsing by directly linking the input stream to the object's internal state.
- **State Management:** The reset() and saveState() methods manage the grid and player state.They centralize the logic for restoring or saving the game's state, ensuring consistency.
- **Player Sprite Updates:** The updatePlayerSprite() method changes the player's texture and position based on the movement direction.This separates visual updates from game logic, maintaining clean separation of concerns.

## 5.3 Images Used

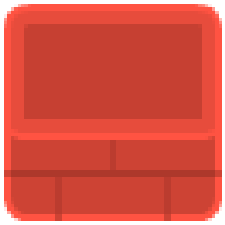

Figure 14: box

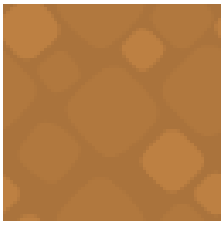

Figure 15: empty

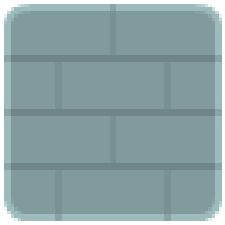

Figure 16: wall

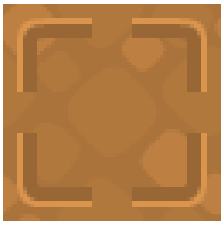

Figure 17: target

Figure 18: player up

Figure 19: player down

Figure 20: player left

Figure 21: player right

## 5.4 What I Accomplished

- **File Handling and Parsing:** Ensuring only valid pushes were allowed, considering edge cases like walls or multiple boxes.
- **Rendering:** Dynamically created a graphical window based on level dimensions, with accurate scaling to fit tiles.
- **Gameplay Mechanics:**
  - Player moves one tile at a time.
  - Boxes move if pushed correctly.
  - Walls and boundaries block movement.

- **Testing and Debugging:** Designed unit tests for all major functionalities (e.g., file parsing, movement validation). This helped catch and fix bugs early.
- **User Interaction:**
  - Added keyboard controls (WASD and arrow keys).
  - Provided visual feedback for invalid moves (e.g., box stuck against a wall).
- **Extra Credit Exploration:** Implemented directional animations for the player and explored the concept of an undo system to revert moves.

### 5.5 What I Learned

- **Game Logic Development:**Writing rules for movement, collision, and victory conditions gave deeper insight into translating abstract rules into concrete code.
- **File I/O:** Improved skills in reading, validating, and handling structured input files.
- **Graphics Programming:** Gained practical experience using SFML for rendering, particularly with managing textures and sprites.
- **Testing Methodologies:** Writing unit tests clarified the importance of isolating individual components for validation.
- **Optimization:** Learned to optimize resource usage by reusing textures and sprites, avoiding memory-intensive operations.
- **Failure Handling:** Understood the need for reset functionality to handle unwinnable game states, mirroring real-world recovery mechanisms in software.

### 5.6 Challenges

- **Collision Complexity:** Handling edge cases like pushing a box into another box or against a wall required careful thought and rigorous testing.
- **Graphics Alignment:** Ensuring tiles aligned correctly, especially for non-square levels, involved significant debugging.
- **State Management:** Resetting and undoing required careful bookkeeping of the initial state and move history.
- **Time Constraints:** Balancing between implementing features and writing comprehensive tests was a constant challenge.

## 5.7 Codebase

**Makefile**

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

# Executable name
EXEC = Sokoban

# Source files
SOURCES = main.cpp Sokoban.cpp

# Object files
OBJECTS = $(SOURCES:.cpp=.o)

# Header files
HEADERS = Sokoban.hpp

# Test files
TEST_SOURCES = test.cpp Sokoban.cpp
TEST_OBJECTS = $(TEST_SOURCES:.cpp=.o)
TEST_EXEC = test

# Default target
all: $(EXEC) Sokoban.a $(TEST_EXEC)

# Linking the executable
$(EXEC): $(OBJECTS)
    $(CXX) $(OBJECTS) -o $(EXEC) $(SFML_LIBS)

# Compiling source files
%.o: %.cpp $(HEADERS)
    $(CXX) $(CXXFLAGS) -c $< -o $@

# Creating static library
Sokoban.a: Sokoban.o
    ar rcs $@ $^

# Building and running tests
$(TEST_EXEC): $(TEST_OBJECTS)
    $(CXX) $(TEST_OBJECTS) -o $(TEST_EXEC) $(SFML_LIBS)
    ./$(TEST_EXEC)

# Linting
lint:
    cpplint  $(SOURCES) $(HEADERS) $(TEST_SOURCES)

# Cleaning up
clean:
```

```
48      rm -f $(EXEC) $(OBJECTS) Sokoban.a $(TEST_EXEC) $(TEST_OBJECTS)
49
50  .PHONY: all lint clean
```

**main.cpp**

```
1   // CopyRight 2024 Harishwar Reddy Erri
2
3   #include <iostream>
4   #include <string>
5   #include <SFML/Graphics.hpp>
6   #include <SFML/Audio.hpp>
7   #include "Sokoban.hpp"
8
9   int main(int argc, char* argv[]) {
10      if (argc != 2) {
11          std::cerr << "Usage: " << argv[0] << " <level_file>\n";
12          return 1;
13      }
14
15      SB::Sokoban sokoban(argv[1]);
16
17      sf::RenderWindow window(sf::VideoMode(sokoban.width() * 64, sokoban.height
        () * 64), "Sokoban");
18
19      // Timer setup
20      sf::Clock gameClock;
21      sf::Text timerText;
22      sf::Font font;
23
24      // Load a font
25      if (!font.loadFromFile("fontt.ttf")) {
26          std::cerr << "Error loading font\n";
27          return 1;
28      }
29
30      timerText.setFont(font);
31      timerText.setCharacterSize(30);
32      timerText.setFillColor(sf::Color::Black);
33      timerText.setPosition(5, 3);
34
35      // Load victory sound
36      sf::SoundBuffer victoryBuffer;
37      if (!victoryBuffer.loadFromFile("victory.wav")) {
38          std::cerr << "Error loading victory sound\n";
39          return 1;
40      }
41      sf::Sound victorySound(victoryBuffer);
42
43      // Load movement sound
44      sf::SoundBuffer moveBuffer;
```

```
45      if (!moveBuffer.loadFromFile("move.wav")) {
46          std::cerr << "Error loading movement sound\n";
47          return 1;
48      }
49      sf::Sound moveSound(moveBuffer);
50
51      bool isWon = false;
52
53      while (window.isOpen()) {
54          sf::Event event;
55          while (window.pollEvent(event)) {
56              if (event.type == sf::Event::Closed) {
57                  window.close();
58              } else if (event.type == sf::Event::KeyPressed) {
59                  if (event.key.code == sf::Keyboard::Escape) {
60                      window.close();  // Close the window on ESC key
61                  } else if (event.key.code == sf::Keyboard::R) {
62                      sokoban.reset();
63                      gameClock.restart();
64                      isWon = false;  // Reset the game state
65                  } else if (!isWon) {
66                      // Game controls only when not won
67                      switch (event.key.code) {
68                          case sf::Keyboard::Up:
69                              sokoban.movePlayer(SB::Direction::Up);
70                              moveSound.play();  // Play movement sound
71                              break;
72                          case sf::Keyboard::Down:
73                              sokoban.movePlayer(SB::Direction::Down);
74                              moveSound.play();  // Play movement sound
75                              break;
76                          case sf::Keyboard::Left:
77                              sokoban.movePlayer(SB::Direction::Left);
78                              moveSound.play();  // Play movement sound
79                              break;
80                          case sf::Keyboard::Right:
81                              sokoban.movePlayer(SB::Direction::Right);
82                              moveSound.play();  // Play movement sound
83                              break;
84                          case sf::Keyboard::Z:
85                              sokoban.undo();
86                              break;
87                          default:
88                              break;
89                      }
90                  }
91              }
92          }
93
94          if (sokoban.isWon() && !isWon) {
95              isWon = true;  // Flag to indicate game is won
```

```
96              victorySound.play();  // Play victory sound
97              sf::Time finalTime = gameClock.getElapsedTime();
98
99              // Prepare the victory message
100             int totalSeconds = static_cast<int>(finalTime.asSeconds());
101             int finalMinutes = totalSeconds / 60;
102             int finalSeconds = totalSeconds % 60;
103             std::string minutesStr = (finalMinutes < 10 ? "0" : "") + std::
    to_string(finalMinutes);
104             std::string secondsStr = (finalSeconds < 10 ? "0" : "") + std::
    to_string(finalSeconds);
105             std::string winMessage = "CONGRATULATIONS\nYOU WON\n\nTime: "
106                                      + minutesStr + ":" + secondsStr;
107
108             sf::Text winText;
109             winText.setFont(font);
110             winText.setString(winMessage);
111             winText.setCharacterSize(35);
112             winText.setFillColor(sf::Color::Black);
113             winText.setPosition(60, 200);
114
115             // Draw victory screen
116             window.clear();
117             window.draw(sokoban);
118             window.draw(timerText);
119             window.draw(winText);
120             window.display();
121         }
122
123         // Update timer if game is not won
124         if (!isWon) {
125             sf::Time elapsed = gameClock.getElapsedTime();
126             int seconds = static_cast<int>(elapsed.asSeconds());
127             int minutes = seconds / 60;
128             seconds %= 60;
129
130             std::string timerString =
131                 (minutes < 10 ? "0" : "") + std::to_string(minutes) + ":" +
132                 (seconds < 10 ? "0" : "") + std::to_string(seconds);
133
134             timerText.setString(timerString);
135
136             // Clear and draw
137             window.clear();
138             window.draw(sokoban);
139             window.draw(timerText);
140             window.display();
141         }
142     }
143
144     return 0;
```

```
}
```

**Sokoban.hpp**

```
// CopyRight 2024 Harishwar Reddy Erri

#ifndef SOKOBAN_HPP
#define SOKOBAN_HPP

#include <string>
#include <vector>
#include <deque>
#include <SFML/Graphics.hpp>

namespace SB {

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

struct GameState {
    std::vector<char> grid;
    sf::Vector2i playerLoc;
};

class Sokoban : public sf::Drawable {
 public:
    Sokoban();
    explicit Sokoban(const std::string& filename);
    int width() const;
    int height() const;
    sf::Vector2i playerLoc() const;
    void movePlayer(Direction dir);
    bool isWon() const;
    void reset();
    void setPlayerTexture(const sf::Texture& texture);
    void undo();

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

 private:
    int m_width;
    int m_height;
    sf::Vector2i m_playerLoc;
    std::vector<char> m_grid;
    std::vector<char> m_initialGrid;
    sf::Texture m_textures[5];
    sf::Sprite m_sprites[5];
    sf::Sprite playerSprite;

    sf::Texture playerTextureUp;
```

```
47      sf::Texture playerTextureDown;
48      sf::Texture playerTextureLeft;
49      sf::Texture playerTextureRight;
50
51      std::deque<GameState> m_undoStates;
52
53      void loadTextures();
54      void loadPlayerTextures();
55      void updatePlayerSprite(Direction dir);
56      void saveState();
57
58      friend std::istream& operator>>(std::istream& is, Sokoban& sokoban);
59  };
60
61  std::istream& operator>>(std::istream& is, Sokoban& sokoban);
62
63  }  // namespace SB
64
65  #endif  // SOKOBAN_HPP
```

**Sokoban.cpp**

```
1   // CopyRight 2024 Harishwar Reddy Erri
2
3   #include <fstream>
4   #include <iostream>
5   #include <algorithm>
6   #include "Sokoban.hpp"
7
8   namespace SB {
9
10  Sokoban::Sokoban() : m_width(0), m_height(0) {
11      // Default constructor implementation
12      loadTextures();
13      loadPlayerTextures();
14  }
15
16  Sokoban::Sokoban(const std::string& filename) : m_width(0), m_height(0) {
17      loadTextures();
18      loadPlayerTextures();
19      std::ifstream file(filename);
20      if (file) {
21          file >> *this;
22      } else {
23          std::cerr << "Failed to open file: " << filename << std::endl;
24      }
25  }
26
27  int Sokoban::width() const {
28      return m_width;
29  }
```

```
30
31 int Sokoban::height() const {
32     return m_height;
33 }
34
35 sf::Vector2i Sokoban::playerLoc() const {
36     return m_playerLoc;
37 }
38
39 void Sokoban::movePlayer(Direction dir) {
40     saveState();
41
42     int dx = 0, dy = 0;
43     switch (dir) {
44         case Direction::Up: dy = -1; break;
45         case Direction::Down: dy = 1; break;
46         case Direction::Left: dx = -1; break;
47         case Direction::Right: dx = 1; break;
48     }
49
50     int newX = m_playerLoc.x + dx;
51     int newY = m_playerLoc.y + dy;
52     if (newX >= 0 && newX < m_width && newY >= 0 && newY < m_height) {
53         char& newCell = m_grid[newY * m_width + newX];
54         char& playerCell = m_grid[m_playerLoc.y * m_width + m_playerLoc.x];
55         if (newCell == '.' || newCell == 'a') {
56             // Moving to an empty space or target
57             if (playerCell == '@') playerCell = '.';
58             else if (playerCell == '+') playerCell = 'a';
59             if (newCell == '.') newCell = '@';
60             else if (newCell == 'a') newCell = '+';
61             m_playerLoc = sf::Vector2i(newX, newY);
62             updatePlayerSprite(dir);
63         } else if (newCell == 'A' || newCell == '1') {
64             // Pushing a box
65             int boxNewX = newX + dx;
66             int boxNewY = newY + dy;
67             if (boxNewX >= 0 && boxNewX < m_width && boxNewY >= 0 && boxNewY <
    m_height) {
68                 char& boxNewCell = m_grid[boxNewY * m_width + boxNewX];
69                 if (boxNewCell == '.' || boxNewCell == 'a') {
70                     // Move box to the new cell
71                     boxNewCell = (boxNewCell == '.') ? 'A' : '1';
72                     // Move player to the box's old position
73                     if (playerCell == '@') playerCell = '.';
74                     else if (playerCell == '+') playerCell = 'a';
75                     newCell = (newCell == 'A') ? '@' : '+';
76                     m_playerLoc = sf::Vector2i(newX, newY);
77                     updatePlayerSprite(dir);
78                 }
79             }
```

```
80          }
81      }
82  }
83
84  bool Sokoban::isWon() const {
85      int boxCount = 0;
86      int targetCount = 0;
87      int boxesOnTarget = 0;
88      bool hasEmptyTarget = false;
89      bool hasBoxOffTarget = false;
90
91      auto checkCell = [&](char c) {
92          switch (c) {
93              case 'A':
94                  boxCount++;
95                  hasBoxOffTarget = true;
96                  break;
97              case 'a':
98                  targetCount++;
99                  hasEmptyTarget = true;
100                 break;
101             case '1':
102                 boxCount++;
103                 targetCount++;
104                 boxesOnTarget++;
105                 break;
106             case '+':
107                 targetCount++;  // Player on target
108                 break;
109         }
110     };
111
112     std::for_each(m_grid.begin(), m_grid.end(), checkCell);
113
114     // Handle special cases
115     if (boxCount == 0 && targetCount == 0) {
116         return true;  // No boxes and no targets, consider it a win (e.g.,
    empty level)
117     }
118
119     if (boxCount == 0 || targetCount == 0) {
120         return false;  // Boxes but no targets, or targets but no boxes
121     }
122
123     // Normal win condition
124     return (boxCount == targetCount) &&
125            (boxesOnTarget == boxCount) &&
126            !hasEmptyTarget &&
127            !hasBoxOffTarget;
128 }
129
```

```
130 void Sokoban::reset() {
131     m_grid = m_initialGrid;
132     auto it = std::find_if(m_grid.begin(), m_grid.end(),
133     [](char c) { return c == '@' || c == '+'; });
134     if (it != m_grid.end()) {
135         int index = std::distance(m_grid.begin(), it);
136         m_playerLoc.x = index % m_width;
137         m_playerLoc.y = index / m_width;
138     }
139     updatePlayerSprite(Direction::Down);
140     m_undoStates.clear();
141 }
142 
143 void Sokoban::loadTextures() {
144     if (!m_textures[0].loadFromFile("empty.png") ||
145         !m_textures[1].loadFromFile("wall.png") ||
146         !m_textures[2].loadFromFile("box.png") ||
147         !m_textures[3].loadFromFile("target.png") ||
148         !m_textures[4].loadFromFile("down.png")) {
149         std::cerr << "Failed to load textures" << std::endl;
150     }
151 
152     for (int i = 0; i < 5; ++i) {
153         m_sprites[i].setTexture(m_textures[i]);
154     }
155 }
156 
157 void Sokoban::loadPlayerTextures() {
158     if (!playerTextureUp.loadFromFile("up.png") ||
159         !playerTextureDown.loadFromFile("down.png") ||
160         !playerTextureLeft.loadFromFile("left.png") ||
161         !playerTextureRight.loadFromFile("right.png")) {
162         std::cerr << "Failed to load player textures" << std::endl;
163     }
164     playerSprite.setTexture(playerTextureDown);
165 }
166 
167 void Sokoban::setPlayerTexture(const sf::Texture& texture) {
168     playerSprite.setTexture(texture);
169 }
170 
171 void Sokoban::updatePlayerSprite(Direction dir) {
172     switch (dir) {
173         case Direction::Up:
174             playerSprite.setTexture(playerTextureUp);
175             break;
176         case Direction::Down:
177             playerSprite.setTexture(playerTextureDown);
178             break;
179         case Direction::Left:
180             playerSprite.setTexture(playerTextureLeft);
```

```
            break;
        case Direction::Right:
            playerSprite.setTexture(playerTextureRight);
            break;
    }
    playerSprite.setPosition(m_playerLoc.x * 64.f, m_playerLoc.y * 64.f);
}

void Sokoban::undo() {
    if (m_undoStates.empty()) return;

    GameState prevState = m_undoStates.back();
    m_undoStates.pop_back();

    m_grid = prevState.grid;
    m_playerLoc = prevState.playerLoc;
    playerSprite.setPosition(m_playerLoc.x * 64.f, m_playerLoc.y * 64.f);
}

void Sokoban::saveState() {
    if (m_undoStates.size() >= 2) {
        m_undoStates.pop_front();
    }
    m_undoStates.push_back({m_grid, m_playerLoc});
}

void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    for (int y = 0; y < m_height; ++y) {
        for (int x = 0; x < m_width; ++x) {
            char cell = m_grid[y * m_width + x];
            sf::Sprite sprite;

            // Always draw the empty space first
            sprite = m_sprites[0];  // Empty space sprite
            sprite.setPosition(x * 64.f, y * 64.f);
            target.draw(sprite, states);

            // Then draw other elements on top
            switch (cell) {
                case '#': sprite = m_sprites[1]; break;
                case 'A': sprite = m_sprites[2]; break;
                case 'a': sprite = m_sprites[3]; break;
                case '1':
                    sprite = m_sprites[3];
                    sprite = m_sprites[2];
                    break;
                default:
                    continue;
            }

            sprite.setPosition(x * 64.f, y * 64.f);
```

```
            target.draw(sprite, states);
        }
    }
    // Draw the player sprite
    target.draw(playerSprite, states);
}

}  // namespace SB

std::istream& SB::operator>>(std::istream& is, SB::Sokoban& sokoban) {
    is >> sokoban.m_height >> sokoban.m_width;
    sokoban.m_grid.resize(sokoban.m_width * sokoban.m_height);
    std::string line;
    std::getline(is, line);
    for (int y = 0; y < sokoban.m_height; ++y) {
        std::getline(is, line);
        for (int x = 0; x < sokoban.m_width; ++x) {
            sokoban.m_grid[y * sokoban.m_width + x] = line[x];
            if (line[x] == '@') {
                sokoban.m_playerLoc = sf::Vector2i(x, y);
                sokoban.playerSprite.setPosition(x * 64.f, y * 64.f);
            }
        }
    }
    sokoban.m_initialGrid = sokoban.m_grid;
    return is;
}
```

**test.cpp**

```
// CopyRight 2024 Harishwar Reddy Erri

#define BOOST_TEST_MODULE SokobanTests
#include <sstream>
#include <boost/test/included/unit_test.hpp>
#include "Sokoban.hpp"

// Replace the namespace directive with specific using-declarations
using SB::Sokoban;
using SB::Direction;

BOOST_AUTO_TEST_SUITE(SokobanTests)

BOOST_AUTO_TEST_CASE(test_initialization) {
    Sokoban sokoban("level1.lvl");
    BOOST_CHECK_EQUAL(sokoban.width(), 10);  // Assuming test level is 5x5
    BOOST_CHECK_EQUAL(sokoban.height(), 10);
}

BOOST_AUTO_TEST_CASE(test_player_movement) {
    Sokoban sokoban("level1.lvl");
```

```
22
23      auto initialPos = sokoban.playerLoc();
24      sokoban.movePlayer(Direction::Right);
25      BOOST_CHECK(sokoban.playerLoc() != initialPos);  // Ensure player moved
26
27      sokoban.movePlayer(Direction::Left);
28      BOOST_CHECK(sokoban.playerLoc() == initialPos);  // Ensure player moved
        back
29  }
30
31  BOOST_AUTO_TEST_CASE(test_box_movement) {
32      Sokoban sokoban("level2.lvl");
33
34      // Set up player and box positions as needed by the test level
35      sokoban.movePlayer(Direction::Right);  // Assuming there's a box in this
        direction
36
37      // Check if box has moved
38      // (Exact check depends on level configuration and Sokoban class
        implementation)
39      // For instance, if box moves to a target:
40      BOOST_CHECK(sokoban.isWon() == false);  // Ensure the game isn't won yet
41  }
42
43  BOOST_AUTO_TEST_CASE(test_reset_functionality) {
44      Sokoban sokoban("level1.lvl");
45      sokoban.movePlayer(Direction::Right);  // Make a move
46      sokoban.reset();
47
48      // Ensure game is reset to initial state
49      BOOST_CHECK_EQUAL(sokoban.playerLoc().x, 3);  // Expected start position x
         = 3
50      BOOST_CHECK_EQUAL(sokoban.playerLoc().y, 6);  // Expected start position y
         = 6
51  }
52
53  BOOST_AUTO_TEST_CASE(test_undo_functionality) {
54      Sokoban sokoban("level3.lvl");
55      auto initialPos = sokoban.playerLoc();
56
57      sokoban.movePlayer(Direction::Right);  // Make a move
58      sokoban.undo();
59
60      // Check if player position is back to the initial position
61      BOOST_CHECK(sokoban.playerLoc() == initialPos);
62  }
63
64
65  BOOST_AUTO_TEST_SUITE_END()
```

## 5.8 Result


Figure 22: Output-1


Figure 23: Output-2


Figure 24: Output-3


Figure 25: Output-4


Figure 26: Victory Output

# 6 PS5: DNA alignment

## 6.1 Discussion

The goal of this assignment was to compute the edit distance (similarity score) between two DNA sequences using sequence alignment. This ties biological computation (DNA alignment) to algorithmic problem-solving paradigms like dynamic programming, showcasing real-world applications in biology and computer science.

- The assignment aimed to apply **dynamic programming** to solve the edit distance problem by aligning two DNA sequences optimally.
- The approach used was based on the **Needleman-Wunsch algorithm**, a dynamic programming technique commonly used in computational biology for sequence alignment.
- The expected benefits of this approach included:
  - Efficient computation of sequence similarity scores even for large DNA sequences.
  - Systematic handling of sequence mismatches, insertions, and deletions to determine optimal biological transformations.
  - Exploration of dynamic programming's ability to manage overlapping subproblems, saving computation time by reusing previously calculated results.
- This assignment illustrates how dynamic programming techniques can solve biological problems by quantifying evolutionary similarity, mutations, or other sequence-based comparisons.

## 6.2 Key Algorithms, Data Structures, and OO Designs Used

**Key Algorithms**

- **Edit Distance Calculation** The `calculateDistance` function uses a Dynamic Programming (DP) approach to compute the minimum edit distance between two strings `str1` and `str2`.

  **Recursive Formulation:**

$$\text{distanceMatrix}[i][j] = \begin{cases} \text{distanceMatrix}[i+1][j+1] + \text{computePenalty}(\text{str1}[i], \text{str2}[j]) \\ \text{distanceMatrix}[i+1][j] + 2 \\ \text{distanceMatrix}[i][j+1] + 2 \end{cases}$$

  **Boundary Conditions:**

  - **Rows ($i$ = len1):** When the remaining characters of `str2` are deleted from the end.
  - **Columns ($j$ = len2):** When the remaining characters of `str1` are inserted to match `str2`.

- **Alignment Calculation** The `computeAlignment` function backtracks through the DP matrix to construct the alignment of `str1` and `str2`:

  - Checks whether characters are equal, substituted, inserted, or deleted by comparing `distanceMatrix[i][j]` with neighboring values.
  - The result string includes:
    * Matched characters with `0`.
    * Substituted characters with `1`.
    * Deletions with `2` (from `str1`).
    * Insertions with `2` (from `str2`).

- **Penalty Computation**The `computePenalty` function determines whether two characters match (penalty = 0) or differ (penalty = 1). This is achieved using a lambda function for clarity and flexibility.
- **Finding the Minimum** The `findMin` function returns the minimum of three integers using a lambda function for succinctness.

**Data Structures**

- **Distance Matrix (`distanceMatrix`):**
  - A 2D dynamically allocated array is used to store the intermediate results of the DP computation.
  - **Size:** $(\text{len1}+1)\times(\text{len2}+1)$, where `len1` and `len2` are the lengths of `str1` and `str2`.
  - **Memory-efficient Access:** Each cell represents the cost of transforming the substring `str1[0...i]` into `str2[0...j]`.
- **Strings:**
  - The input strings `str1` and `str2` are stored as class members to be used throughout the class methods.
- **Result String:**
  - A single string is used in `computeAlignment` to store the alignment results in a structured format.

**OO Designs**

- **Encapsulation:**
  - Class members (`str1`, `str2`, `len1`, `len2`, and `distanceMatrix`) are encapsulated within the `EDistance` class.
  - This prevents external modification of internal states, ensuring data integrity.
- **Constructor Initialization:**
  - The constructor initializes `str1` and `str2` and dynamically allocates the distance matrix.
  - It also validates inputs (e.g., empty strings) and handles memory allocation errors gracefully.

- **Destruction:**
  - The destructor (`~EDistance`) ensures proper cleanup of dynamically allocated memory, preventing memory leaks.
- **Modularization:**
  - Each method has a clear and single responsibility:
    - `calculateDistance`: Computes the edit distance.
    - `computeAlignment`: Constructs the alignment.
    - `computePenalty`: Determines the penalty for character substitution.
    - `findMin`: Finds the minimum of three integers.
    - `calculateMemoryUsageMB`: Estimates memory usage.
  - This modular design improves code readability, maintainability, and reusability.
- **Error Handling:**
  - The constructor throws an exception if either input string is empty.
  - If memory allocation for the matrix fails, an error message is displayed, and the program exits gracefully.
- **Use of Lambda Functions:**
  - Lambda functions in `computePenalty` and `findMin` encapsulate small, reusable logic, making the code concise and readable.

### 6.3 What I Accomplished

Through this assignment, I successfully:

- Designed and implemented a solution to compute the edit distance using dynamic programming.
- Encapsulated the solution logic into a reusable class named `EDistance`.
- Designed the solution based on the Needleman-Wunsch method with proper scoring penalties.
- Traced the computed matrix to generate the optimal sequence alignment and corresponding cost.
- Used Valgrind to analyze memory usage and ensure efficient memory handling.
- Measured execution performance on various test input sizes, confirming the solution's robustness under expected biological input scales.

The final program computes the edit distance efficiently, generates optimal alignments, and adheres to the input/output specifications defined in the assignment.

### 6.4 What I Already Knew

Before starting this assignment, I had foundational knowledge in:

- **Dynamic Programming:** Familiar with how it can optimize overlapping subproblems by storing computed solutions.
- **Sequence Alignment Concepts:** Familiarity with edit distances and how they are applied in biological sequence analysis.
- **C++ Programming:** Prior experience coding in C++ and implementing algorithms from scratch.
- **Valgrind:** Some knowledge of using Valgrind for debugging memory leaks and profiling memory usage.
- **Needleman-Wunsch Algorithm:** Theoretical familiarity with the standard approach used for biological sequence alignment.

This prior knowledge enabled me to approach the assignment efficiently, focusing on implementing the given problem requirements.

### 6.5 What I Learned

From this assignment, I gained deeper insights into:

- **Dynamic Programming in Practice:** Learned how to compute and trace solutions iteratively, avoiding recursion to save computation time and memory.
- **Biological Sequence Analysis:** Discovered how concepts like edit distance and sequence alignment translate into computational tools for studying DNA sequences.
- **Memory Management:** Learned the importance of properly managing memory in dynamic programming solutions, especially when using dynamic matrix allocation.
- **Valgrind Usage:** Learned how to use Valgrind to analyze memory performance and identify potential bottlenecks.
- **Object-Oriented Programming with Complex Algorithms:** Learned how organizing sequence alignment logic into a class simplifies testing, reusability, and design patterns.

### 6.6 Challenges

I faced several challenges during the project:

- **Matrix Memory Management:**
  - Allocating and deallocating memory dynamically to handle very large problem sizes.
  - Ensuring there were no memory leaks or dangling pointers.
- **Edge Cases:**
  - Handling input sequences of different lengths.
  - Ensuring that gap insertion penalties were calculated correctly.

- **Debugging:**
  - Debugging performance bottlenecks using Valgrind and adjusting optimization flags.
- **Valgrind Analysis:**
  - Learning to integrate and interpret the Valgrind results for heap analysis.
  - Ensuring expected memory usage behavior.

To overcome these challenges, careful testing was performed with a variety of input sizes, Valgrind tools were used iteratively to verify memory usage, and dynamic programming was implemented methodically to avoid unnecessary recomputation.

## 6.7 Codebase

**Makefile**

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

OBJECTS= EDistance.o

all: EDistance test EDistance.a

EDistance: main.o $(OBJECTS)
    $(CC) main.o $(OBJECTS) -o EDistance $(LIB)

test: test.o $(OBJECTS)
    $(CC) test.o $(OBJECTS) -o test $(LIB)

main.o: main.cpp EDistance.hpp
    $(CC) -c main.cpp  $(CFLAGS)


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

test.o: test.cpp EDistance.hpp
    $(CC) -c test.cpp $(CFLAGS)

EDistance.a: main.o
    ar rcs EDistance.a main.o

clean:
    rm *.o
    rm EDistance
    rm test
```

**main.cpp**

```cpp
//  CopyRight 2024 Harishwar Reddy Erri
#include <iostream>
#include <string>
#include "EDistance.hpp"
#include <SFML/System.hpp>

int main(int argc, const char* argv[]) {
    sf::Clock clock;
    sf::Time t;

    std::string string1, string2;
    std::cin >> string1 >> string2;
    EDistance editDistance(string1, string2);
    int distance = editDistance.calculateDistance();
    std::cout << "Edit distance = " << distance << std::endl;
    std::string alignment = editDistance.computeAlignment();
    std::cout << alignment << std::endl;
    double memoryUsageMB = editDistance.calculateMemoryUsageMB();
    std::cout << "Memory used = " << memoryUsageMB
    << " MB" << std::endl;
    t = clock.getElapsedTime();
    std::cout << "Execution time is " << t.asSeconds()
              << " seconds \n" << std::endl;

    return 0;
}
```

**EDistance.hpp**

```cpp
//  CopyRight 2024 Harishwar Reddy Erri

#pragma once
#include <string>
#include <algorithm>

class EDistance {
 public:
    EDistance(const std::string &str1, const std::string &str2);
    ~EDistance();
    static int computePenalty(char char1, char char2);
    static int findMin(int a, int b, int c);
    int calculateDistance();
    std::string computeAlignment();
    double calculateMemoryUsageMB();

 private:
    int **distanceMatrix;
    std::string str1;
    std::string str2;
```

```
21      int len1, len2;
22  };
```

**EDistance.cpp**

```
1   //  CopyRight 2024 Harishwar Reddy Erri
2
3   #include <iostream>
4   #include <string>
5   #include <algorithm>
6   #include "EDistance.hpp"
7
8   EDistance::EDistance(const std::string &str1, const std::string &str2) {
9       this->str1 = str1;
10      this->str2 = str2;
11      if (str1.empty() || str2.empty()) {
12          throw std::exception();
13      } else {
14          len1 = str1.size();
15          len2 = str2.size();
16          distanceMatrix = new int *[len1 + 1]();
17          if (!distanceMatrix) {
18              std::cout << "Not enough memory" << std::endl;
19              exit(1);
20          }
21          for (int i = 0; i < len1 + 1; i++) {
22              distanceMatrix[i] = new int[len2 + 1]();
23          }
24          for (int i = 0; i <= len1; i++) {
25              distanceMatrix[i][len2] = 2 * (len1 - i);
26          }
27          for (int j = 0; j <= len2; j++) {
28              distanceMatrix[len1][j] = 2 * (len2 - j);
29          }
30      }
31  }
32
33  int EDistance::calculateDistance() {
34      for (int i = len1 - 1; i >= 0; i--) {
35          for (int j = len2 - 1; j >= 0; j--) {
36              distanceMatrix[i][j] = findMin(
37                  distanceMatrix[i + 1][j + 1] +
38                  computePenalty(str1[i], str2[j]),
39                  distanceMatrix[i + 1][j] + 2,
40                  distanceMatrix[i][j + 1] + 2);
41          }
42      }
43      return distanceMatrix[0][0];
44  }
45
46  std::string EDistance::computeAlignment() {
```

```
47      std::string result;
48      int i = 0, j = 0;
49      while (i < len1 || j < len2) {
50          if (i < len1 && j < len2 && str1[i] == str2[j] && distanceMatrix[i][j]
51              == distanceMatrix[i + 1][j + 1]) {
52              result += str1[i];
53              result += " ";
54              result += str2[j];
55              result += " 0";
56              i++;
57              j++;
58          } else if (i < len1 && j < len2 && str1[i] != str2[j] &&
59              distanceMatrix[i][j] == distanceMatrix[i + 1][j + 1] + 1) {
60              result += str1[i];
61              result += " ";
62              result += str2[j];
63              result += " 1";
64              i++;
65              j++;
66          } else if (i < len1 && distanceMatrix[i][j]
67              == distanceMatrix[i + 1][j] + 2) {
68              result += str1[i];
69              result += " - 2";
70              i++;
71          } else if (j < len2 && distanceMatrix[i][j]
72              == distanceMatrix[i][j + 1] + 2) {
73              result += "- ";
74              result += str2[j];
75              result += " 2";
76              j++;
77          }
78          if (i < len1 || j < len2) result += "\n";
79      }
80      return result;
81  }
82
83  int EDistance::computePenalty(char char1, char char2) {
84      auto penaltyLambda = [=]() -> int {
85          return (char1 == char2) ? 0 : 1;
86      };
87      return penaltyLambda();
88  }
89
90  int EDistance::findMin(int a, int b, int c) {
91      int A = a, B = b, C = c;
92      auto minLambda = [=]() -> int {
93          if (A <= B && A <= C) {
94              return A;
95          } else if (B <= A && B <= C) {
96              return B;
97          } else {
```

```
            return C;
        }
    };
    return minLambda();
}

double EDistance::calculateMemoryUsageMB() {
    return ((len1 + 1) * (len2 + 1) * sizeof(int)) / (1024.0 * 1024.0);
}

EDistance::~EDistance() {
    for (int i = 0; i <= len1; i++) {
        delete[] distanceMatrix[i];
    }
    delete[] distanceMatrix;
}
```

**test.cpp**

```
//  CopyRight 2024 Harishwar Reddy Erri
#define BOOST_TEST_MODULE MyTest
#include <boost/test/included/unit_test.hpp>
#include "EDistance.hpp"

BOOST_AUTO_TEST_CASE(FIRST_TESTCASE) {
    std::string a = "a";
    std::string b = "a";
    EDistance ed(a, b);

    int result = ed.computePenalty('a', 'a');

    BOOST_REQUIRE_EQUAL(result, 0);
}

BOOST_AUTO_TEST_CASE(SECOND_TESTCASE) {
    BOOST_REQUIRE_THROW(EDistance("hello", ""), std::exception);
    BOOST_REQUIRE_THROW(EDistance("", ""), std::exception);
}

BOOST_AUTO_TEST_CASE(THIRD_TESTCASE) {
    int result1 = EDistance::findMin(4, 3, 2);
    BOOST_REQUIRE_EQUAL(result1, 2);

    int result2 = EDistance::findMin(5, 3, 5);
    BOOST_REQUIRE_EQUAL(result2, 3);

    int result3 = EDistance::findMin(7, 7, 7);
    BOOST_REQUIRE_EQUAL(result3, 7);
}

BOOST_AUTO_TEST_CASE(FOURTH_TESTCASE) {
```

```
33      EDistance ed1("hello", "hello");
34      BOOST_REQUIRE_EQUAL(ed1.calculateDistance(), 0);
35
36      EDistance ed2("eatting", "setting");
37      BOOST_REQUIRE_EQUAL(ed2.calculateDistance(), 2);
38  }
```

### 6.8 Result

**Input: example10.txt**

AACAGTTACC
TAAGGTCA

**Output:**

```
1  Edit distance = 7
2  A T 1
3  A A 0
4  C - 2
5  A A 0
6  G G 0
7  T G 1
8  T T 0
9  A - 2
10 C C 0
11 C A 1
12 Memory used = 0.000377655 MB
13 Execution time is 0.004158 seconds
```

# 7 PS6: RandWriter

## 7.1 Discussion

This section should explore the overall context and purpose of the project. The project revolves around using a Markov model to analyze text and generate pseudo-random yet coherent text sequences. It involves implementing a class RandWriter to construct a Markov model, process input text to extract k-grams, and produce text based on probabilistic models derived from the input. The main discussion points could include:

- How the Markov model captures the statistical properties of natural language.
- The role of different Markov orders (k) in creating varying levels of coherence.
- The impact of treating the input as a circular string to avoid dead ends.

## 7.2 Key Algorithms, Data Structures, and OO Designs Used

**Key Algorithms**

- **K-gram Extraction:** Traverse the input text to extract k-grams (substrings of length $k$) and their subsequent characters. Treat the input as circular for seamless transitions.
- **Frequency Counting:** Construct a frequency map of k-grams and their possible next characters using a sliding window technique. For example, in the text `"ababab"` with $k = 2$, the k-gram `"ab"` alternates between being followed by `"a"` and `"b"`.
- **Random Selection:** Use the `<random>` library to generate characters based on weighted probabilities derived from observed frequencies.
- **Text Generation:** Start with an initial k-gram (seed) and iteratively generate text by appending characters chosen through probabilistic selection.

**Data Structures**

- **Maps:** Store k-grams as keys and their subsequent character frequencies as values. For instance.
- **Vectors:** Represent possible next characters and their probabilities for efficient random selection.
- **Circular Strings:** Ensure continuity by wrapping the end of the text to the start.

**Object-Oriented Designs**

- **Encapsulation:** The `RandWriter` class encapsulates all functionalities such as k-gram extraction, frequency counting, and text generation.
- **Method Overloading:** Methods like `freq(kgram)` and `freq(kgram, c)` provide flexible queries on k-gram statistics.
- **Stream Insertion Operator:** Overloaded to display the internal state of the Markov model for debugging and analysis.

### 7.3 What I Accomplished

- Developed a robust implementation of the RandWriter class that supports Markov models of various orders.
- Implemented a client program TextWriter to generate pseudo-random text of a specified length.
- Successfully used the ¡random¿ library to generate high-quality random numbers for probabilistic output.
- Completed and validated the project through unit tests using the Boost library.

### 7.4 What I Learned

- Deepened understanding of Markov chains and their application in text modeling.
- Gained hands-on experience with probabilistic algorithms and text processing.
- Learned to manage circular data structures and deal with edge cases in text generation.
- Improved familiarity with C++ STL containers (e.g., maps) and the Boost testing framework.
- Enhanced debugging skills by testing against edge cases and reference implementations.

### 7.5 Challenges

- Implementing the circular handling of input text to avoid dead ends required extra care.
- Balancing the complexity of different Markov orders while ensuring performance and accuracy.
- Ensuring the k-gram map accurately mirrored input frequencies and handled exceptions correctly.
- Debugging the weighted random selection to match observed frequencies.
- Understanding and adhering to the requirements of the ¡random¿ library versus legacy random number generation methods.

### 7.6 Codebase

**Makefile**

```
1  compiler = g++
2  flags_c = -g -std=c++0x -Wall -Werror -pedantic
3  flags_S = -Wall -Werror -pedantic -lsfml-graphics -lsfml-window -lsfml-system
4  lboost = -lboost_unit_test_framework
5
6  all: TextWriter test TextWriter.a lint
7
8  lint:
9      cpplint --filter=-runtime/references,-build/c++14 --root=. *.hpp *.cpp
10
11 TextWriter: TextWriter.o RandWriter.o
```

```
    $(compiler) TextWriter.o RandWriter.o -o TextWriter $(flags_S)

test: test.o RandWriter.o
    $(compiler) $(flags_c) test.o RandWriter.o -o test $(lboost)

TextWriter.o: TextWriter.cpp RandWriter.hpp
    $(compiler) -c TextWriter.cpp $(flags_c)

RandWriter.o: RandWriter.cpp RandWriter.hpp
    $(compiler) -c RandWriter.cpp $(flags_c)

test.o: test.cpp RandWriter.hpp
    $(compiler) -c test.cpp $(flags_c)

TextWriter.a: RandWriter.o TextWriter.o
    ar rcs TextWriter.a RandWriter.o TextWriter.o

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

.PHONY: all clean lint
```

**TextWriter.cpp**

```
// CopyRight 2024 Harishwar Reddy Erri

#include <iostream>
#include <string>
#include "RandWriter.hpp"

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: ./TextWriter <k> <L>" << std::endl;
        return 1;
    }

    size_t k = static_cast<size_t>(std::stoi(argv[1]));
    size_t L = static_cast<size_t>(std::stoi(argv[2]));
    std::string inputText;
    while (std::getline(std::cin, inputText)) {
        if (!inputText.empty()) break;
    }
    try {
        RandWriter rw(inputText, k);
        std::string initialKGram = inputText.substr(0, k);
        std::string generatedText = rw.generate(initialKGram, L);
        std::cout << generatedText << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
```

```
    }
    return 0;
}
```

**RandWriter.hpp**

```
// CopyRight 2024 Harishwar Reddy Erri

#ifndef RANDWRITER_HPP
#define RANDWRITER_HPP

#include <string>
#include <map>
#include <vector>
#include <random>
#include <stdexcept>

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);
    std::string validKgram;
 private:
    size_t k;
    std::map<std::string, std::map<char, int>> freqTable;
    std::mt19937 rng;
    std::string inputText;
    std::string trimToLength(const std::string& text, size_t L) const {
        if (text.length() <= L) return text;
        return text.substr(0, L);
    }
};

#endif  // RANDWRITER_HPP
```

**RandWriter.cpp**

```
// CopyRight 2024 Harishwar Reddy Erri

#include <ostream>
#include <iostream>
#include "RandWriter.hpp"

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 k.");
    }
    size_t textLength = text.length();
    std::string extendedText = text + text.substr(0, k);
    for (size_t i = 0; i < textLength; ++i) {
        std::string kgram = extendedText.substr(i, k);
        char nextChar = extendedText[i + k];
        freqTable[kgram][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.");
    }
    auto it = freqTable.find(kgram);
    if (it == freqTable.end()) {
        return 0;
    }
    int total = 0;
    for (const auto& pair : it->second) {
        total += pair.second;
    }
    return total;
}

int RandWriter::freq(const std::string& kgram, char c) const {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram must be of length k.");
    }
    if (freqTable.find(kgram) == freqTable.end()) {
        return 0;
    }
    return freqTable.at(kgram).count(c) ? freqTable.at(kgram).at(c) : 0;
}

char RandWriter::kRand(const std::string& kgram) {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram must be of length k.");
    }
    auto it = freqTable.find(kgram);
    if (it == freqTable.end()) {
        throw std::invalid_argument("kgram not found in frequency table.");
    }
    int total = 0;
    for (const auto& pair : it->second) {
        total += pair.second;
    }
```

```
60      if (total == 0) {
61          throw std::runtime_error("Total frequency is zero, cannot select
        random character.");
62      }
63      std::uniform_int_distribution<> dist(1, total);
64      int r = dist(rng);
65      for (const auto& pair : it->second) {
66          r -= pair.second;
67          if (r <= 0) {
68              return pair.first;
69          }
70      }
71      throw std::runtime_error("Unexpected error in kRand.");
72  }
73
74  std::string RandWriter::generate(const std::string& kgram, size_t L) {
75      if (k == 0) {
76          throw std::invalid_argument("Cannot generate text with k = 0");
77      }
78      if (kgram.size() != k) {
79          throw std::invalid_argument("kgram must be of length k");
80      }
81      if (L < k) {
82          throw std::invalid_argument("L must be at least k");
83      }
84
85      std::string result = kgram;
86      while (result.length() < L) {
87          std::string currentKGram = result.substr(result.size() - k, k);
88          char next_char;
89
90          try {
91              next_char = kRand(currentKGram);
92          } catch (const std::exception& e) {
93              std::cerr << "Warning: " << e.what()
94                        << ". Using fallback strategy for kgram: " <<
        currentKGram << std::endl;
95              next_char = inputText[rng() % inputText.size()];
96          }
97          result += next_char;
98      }
99
100     return result.substr(0, L);
101 }
```

**test.cpp**

```
1  // CopyRight 2024 Harishwar Reddy Erri
2
3  #define BOOST_TEST_MODULE RandWriterTest
4  #include <boost/test/included/unit_test.hpp>
5  #include "RandWriter.hpp"
6
7  BOOST_AUTO_TEST_SUITE(RandWriterTests)
8
9  BOOST_AUTO_TEST_CASE(TestInvalidKGram) {
10     RandWriter rw("sample text", 3);
11     BOOST_REQUIRE_THROW(rw.freq("in"), std::invalid_argument);
12 }
13
14 BOOST_AUTO_TEST_CASE(TestValidKGram) {
15     RandWriter rw("sample text", 3);
16     BOOST_REQUIRE_NO_THROW(rw.freq("sam"));
17 }
18
19 BOOST_AUTO_TEST_CASE(TestFrequencyOfKGram) {
20     RandWriter rw("sample text", 3);
21     int frequency = rw.freq("sam");
22     BOOST_REQUIRE_EQUAL(frequency, 1);
23 }
24
25 BOOST_AUTO_TEST_CASE(TestGenerateText) {
26     RandWriter rw("sample text", 3);
27     std::string generatedText = rw.generate("sam", 10);
28     BOOST_REQUIRE(!generatedText.empty());
29 }
30
31 BOOST_AUTO_TEST_CASE(TestGenerateTextLength) {
32     RandWriter rw("THE ADVENTURES OF TOM SAWYER", 5);
33     std::string generatedText = rw.generate("THE A", 100);
34     BOOST_REQUIRE_EQUAL(generatedText.length(), 100);
35 }
36
37 BOOST_AUTO_TEST_CASE(TestGenerateIncorrectLength) {
38     RandWriter rw("THE ADVENTURES OF TOM SAWYER", 5);
39     std::string generatedText = rw.generate("THE A", 300);
40     BOOST_REQUIRE_EQUAL(generatedText.length(), 300);
41 }
42
43 BOOST_AUTO_TEST_CASE(TestGenerateStartsWithCorrectKGram) {
44     RandWriter rw("THE ADVENTURES OF TOM SAWYER", 5);
45     std::string startingKGram = "THE A";
46     std::string generatedText = rw.generate(startingKGram, 100);
47     BOOST_REQUIRE_EQUAL(generatedText.substr(0, startingKGram.length()),
       startingKGram);
48 }
49
```

```
50 BOOST_AUTO_TEST_SUITE_END()
```

### 7.7 Result

**Input :** input17.txt

```
gagggagaggcgagaaa
```

**Output:**

```
1     gaggaaagaaa
```

# 8 PS7: Kronos Log Parsing

## 8.1 Discussion

This project revolves around analyzing log files from the Kronos InTouch device, a Linux-based touchscreen time clock. The primary task is to parse these logs to extract meaningful insights into device boot-up processes and identify potential issues.
This project provides valuable diagnostic tools for engineers and service teams to investigate and troubleshoot issues with the Kronos device.

The project focuses on:

- **Analyzing Device Boot-Up Sequences:** Extracting information about each startup sequence, including start and end times, and calculating the duration of successful boots.
- **Identifying Failures:** Detecting incomplete boot sequences and reporting their details for debugging.
- **Using Regular Expressions (Regex):** Leveraging regex for efficient log parsing, which is particularly useful for handling structured data in unstructured text files.

## 8.2 Key Algorithms, Data Structures, and OO Designs Used

**Key Algorithms**

- **Log Parsing Using Regular Expressions:**
  - Extract timestamps and key events (e.g., "server started" and "Started SelectChannelConnector") from log entries using regex patterns.
  - Match and group log entries to pair startup messages with their corresponding completion or failure events.
- **Data Output:** Format the results into a report containing line numbers, timestamps, boot durations, and error messages for incomplete startups.
- **Boot Time Calculation:** Compute the elapsed time between matched "server started" and "Started SelectChannelConnector" entries using the Boost Date-Time library or Python's datetime module.

**Data Structures**

- **Vectors or Lists:** Store parsed log entries as objects or dictionaries containing details like timestamps and event messages.
- **Maps or Dictionaries:** Use key-value pairs to organize and match log events (e.g., pairing start and completion messages).
- **String Manipulation:** Efficiently extract and process substrings from log entries using regex and string methods.

**Object-Oriented Designs**

- **Encapsulation:** Group functionalities (e.g., parsing, time calculation, and report generation) into distinct functions or classes for modularity.
- **Error Handling:** Implement mechanisms to catch invalid log entries or mismatches, throwing exceptions or recording errors where necessary.
- **Flexibility:** Allow users to specify input log files and generate appropriately formatted output files, aligning with the given naming conventions.

## 8.3 What I Accomplished

- **Log Parsing:**
  - Successfully implemented regex patterns to identify and extract relevant log entries.
  - Matched and grouped start and end log entries for device boot sequences.
- **Error Reporting:** Implemented optional features like a summary header and detailed service boot logging for additional functionality.
- **Time Calculations:** Used the Boost Date-Time library to compute precise boot durations.
- **Output Formatting:**Generated detailed reports that adhere to the project requirements, including success or failure states for each boot attempt.
- **Extra Credit:** Implemented optional features like a summary header and detailed service boot logging for additional functionality..

## 8.4 What I Learned

- **Regex and Log Analysis:**Gained a deeper understanding of regex for parsing structured data in log files, including techniques for grouping and capturing.
- **Boost Date-Time Library:** ILearned to perform advanced time calculations and format time outputs using Boost in C++.
- **Error Handling in Parsing:** Developed robust methods to detect and report mismatched or invalid log entries.
- **File Handling:** Improved skills in reading, writing, and managing file outputs programmatically.

## 8.5 Challenges

- **Complexity of Log Parsing:**Crafting regex patterns to match diverse log formats while avoiding false positives or negatives.
- **Handling Edge Cases:** Addressing scenarios like nested or overlapping boot sequences, missing timestamps, and improperly formatted entries.
- **Boost Date-Time Library:** Navigating Boost's complex API to calculate and format elapsed times accurately.
- **Output Formatting:** Ensuring that generated reports matched the required structure and included all necessary details for successful debugging.

## 8.6 Codebase

**Makefile**

```
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra  -Werror -pedantic
LDFLAGS = -lboost_date_time -lboost_regex

TARGET = ps7
SRCS = ps7.cpp
OBJS = $(SRCS:.cpp=.o)

.PHONY: all clean lint

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CXX) $(CXXFLAGS) -o $@ $^ $(LDFLAGS)

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

lint:
    cpplint *.cpp

clean:
    rm -f $(TARGET) $(OBJS)
```

**ps7.cpp**

```
// CopyRight 2024 Harishwar Reddy Erri

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <boost/regex.hpp>
#include "boost/date_time/posix_time/posix_time.hpp"

using std::cout;
using std::cin;
using std::endl;
using std::string;
using boost::regex;
using boost::smatch;
using boost::regex_error;
using boost::gregorian::date;
using boost::gregorian::from_simple_string;
using boost::gregorian::date_period;
using boost::gregorian::date_duration;
using boost::posix_time::ptime;
```

```
using boost::posix_time::time_duration;

struct BootService {
    string serviceName;
    ptime bootStartTime;
    ptime bootEndTime;
};

int main(int argc, char** argv) {
    if (argc != 2) {
        cout << "usage: ./boot_report [logfile]" << endl;
        exit(1);
    }

    string logLine, logFileName(argv[1]);
    regex startPattern, successPattern;
    bool isBooting = false;

    std::ifstream logFile(logFileName);
    std::stringstream reportContent;

    if (!logFile) {
        cout << "Error opening file" << endl;
        exit(1);
    }

    try {
        startPattern = regex(R"((.*): (\(log.c.166\) server started.*))");
        successPattern =
        regex("(.*)\\.\\d*:INFO:oejs.AbstractConnector:Started "
                                "SelectChannelConnector@0.0.0.0:9080.*");
    } catch (regex_error& e) {
        cout << "Regex initialization failed: " << e.code() << endl;
        exit(1);
    }

    std::vector<BootService> bootServices;
    int lineNumber = 0;
    int initiatedBoots = 0;
    int completedBoots = 0;

    while (getline(logFile, logLine)) {
        lineNumber++;
        if (regex_match(logLine, startPattern)) {
            smatch matchResult;
            regex_match(logLine, matchResult, startPattern);
            if (isBooting) {
                reportContent << "**** Incomplete boot ****" << endl << endl;
            }
            isBooting = true;
            initiatedBoots++;
```

```
73
74              BootService bootService;
75              bootService.serviceName = matchResult[2];
76              bootService.bootStartTime =
77              ptime(boost::posix_time::time_from_string(matchResult[1]));
78              bootServices.push_back(bootService);
79
80              reportContent << "=== Device boot ===" << endl;
81              reportContent << lineNumber << "(" << logFileName << "): "
82                            << matchResult[1] << " Boot Start" << endl;
83          }
84
85          if (regex_match(logLine, successPattern)) {
86              smatch matchResult;
87              regex_match(logLine, matchResult, successPattern);
88              if (!bootServices.empty()) {
89                  BootService& lastService = bootServices.back();
90                  lastService.bootEndTime =
91                  ptime(boost::posix_time::time_from_string(matchResult[1]));
92                  reportContent << lineNumber << "(" << logFileName << "): "
93                                << matchResult[1] << " Boot Completed" << endl;
94                  reportContent << "\tBoot Time: "
95                                << (lastService.bootEndTime -
96                                lastService.bootStartTime)
97                                .total_milliseconds()
98                                << "ms" << endl << endl;
99                  completedBoots++;
100             }
101             isBooting = false;
102         }
103     }
104
105     if (isBooting) {
106         reportContent << "**** Incomplete boot ****" << endl << endl;
107     }
108
109     logFile.close();
110
111     // Write the report to a file
112     std::ofstream reportFile(logFileName + ".rpt");
113     if (!reportFile) {
114         cout << "Error creating report file" << endl;
115         exit(1);
116     }
117
118     reportFile << "Device Boot Report" << endl << endl;
119     reportFile << "InTouch log file: " << logFileName << endl;
120     reportFile << "Lines Scanned: " << lineNumber << endl << endl;
121     reportFile << "Device boot count: initiated = " << initiatedBoots
122                << ", completed: " << completedBoots << endl << endl;
123     reportFile << reportContent.str();
```

```
124
125     reportFile.close();
126     return 0;
127 }
```

### 8.7 Result

The Output of the Device 1

```
1  Device Boot Report
2
3  InTouch log file: device1_intouch.log
4  Lines Scanned: 443838
5
6  Device boot count: initiated = 6, completed: 6
7
8  === Device boot ===
9  435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
10 435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
11     Boot Time: 183000ms
12
13 === Device boot ===
14 436500(device1_intouch.log): 2014-03-25 19:29:59 Boot Start
15 436859(device1_intouch.log): 2014-03-25 19:32:44 Boot Completed
16     Boot Time: 165000ms
17
18 === Device boot ===
19 440719(device1_intouch.log): 2014-03-25 22:01:46 Boot Start
20 440791(device1_intouch.log): 2014-03-25 22:04:27 Boot Completed
21     Boot Time: 161000ms
22
23 === Device boot ===
24 440866(device1_intouch.log): 2014-03-26 12:47:42 Boot Start
25 441216(device1_intouch.log): 2014-03-26 12:50:29 Boot Completed
26     Boot Time: 167000ms
27
28 === Device boot ===
29 442094(device1_intouch.log): 2014-03-26 20:41:34 Boot Start
30 442432(device1_intouch.log): 2014-03-26 20:44:13 Boot Completed
31     Boot Time: 159000ms
32
33 === Device boot ===
34 443073(device1_intouch.log): 2014-03-27 14:09:01 Boot Start
35 443411(device1_intouch.log): 2014-03-27 14:11:42 Boot Completed
36     Boot Time: 161000ms
```

The Output of the Device 2

```
Device Boot Report

InTouch log file: device2_intouch.log
Lines Scanned: 538352

Device boot count: initiated = 1, completed: 1

=== Device boot ===
498921(device2_intouch.log): 2014-03-11 15:42:26 Boot Start
499030(device2_intouch.log): 2014-03-11 15:45:08 Boot Completed
    Boot Time: 162000ms
```