Computing IV Sec 201: Project Portfolio

Jason Ossai

December 12, 2024

Contents

Time to Complete Portfolio: 48 hours

1 PS0: Hello SFML

1.1 Discussion

This project introduced the basics of the SFML (Simple and Fast Multimedia Library). The goal was to create a simple graphical application where a sprite moves across a window in response to keyboard inputs. This helped understand graphical rendering and event handling in SFML.

1.2 Key Algorithms and Data Structures

The project utilized event polling to detect keyboard inputs. The sprite's movement was directly controlled based on these events. This required a simple structure and focused on real-time input processing without complex algorithms.

1.3 Learnings

The project taught me how to set up an SFML application, create a graphical window, render objects, and handle user inputs. This knowledge is foundational for building interactive graphical applications.

1.4 Makefile

Listing 1: Makefile for PS0

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
# Your .hpp files
DEPS =
# Your compiled .o files
OBJECTS = main.o
# The name of your program
PROGRAM = ./sfml-app

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

1.5 Main Source Code

Listing 2: Main Source Code for PS0

```
#include <SFML/Graphics.hpp>
#include <cstdlib>  // For EXIT_SUCCESS and EXIT_FAILURE
#include <iostream> // For std::cerr

int main()
```

```
{
    sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");

    sf::Texture texture;
    if (!texture.loadFromFile("sprite.png"))
    {
        std::cerr << "Error: Could not load sprite.png" << std::endl;
        return EXIT_FAILURE;
    }

    sf::Sprite sprite(texture);

    sprite.setPosition(400, 300);

    float speed = 5.0f;

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

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
        {
            sprite.move(-speed, 0);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
        {
            sprite.move(speed, 0);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
        {
            sprite.move(0, -speed);
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
        {
            sprite.move(0, speed);
        }

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

    return EXIT_SUCCESS;
}
```

1.6 Evidence of Code Execution

Figure 1: Evidence of the running code with the displayed sprite.

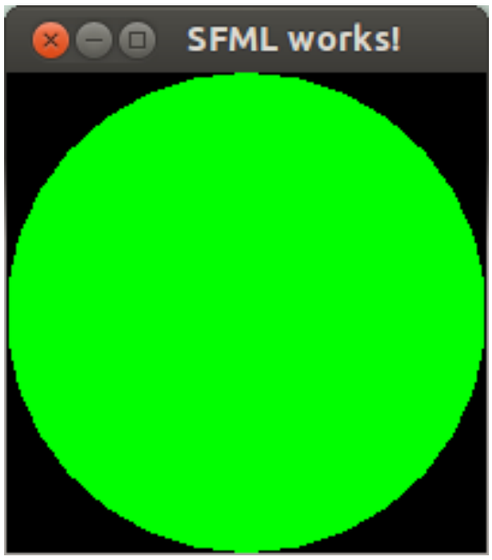

Figure 2: Original image

2 PS1a: Linear Feedback Shift Register

2.1 Discussion

This project focused on the implementation of a Linear Feedback Shift Register (LFSR) for digital signal processing and encryption. The main task was to create an LFSR that could generate pseudo-random numbers and manipulate data through bitwise operations.

2.2 Key Algorithms and Data Structures

The key algorithm used was the step function, which simulates the behavior of an LFSR by performing bitwise XOR operations on specific tap bits. This process shifts the bits of the register, creating a pseudo-random sequence. The LFSR's internal state was maintained using the `std::bitset` data structure for efficient bit manipulation.

2.3 Learnings

This assignment helped solidify my understanding of bitwise operations and their practical applications in pseudo-random number generation. Additionally, I learned about implementing custom operators like « for outputting the LFSR's state as a readable binary string.

2.4 Makefile

Listing 3: Makefile for PS1a

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lboost_unit_test_framework
# Your .hpp files
DEPS = FibLFSR.hpp
# Your compiled .o files
OBJECTS = FibLFSR.o PhotoMagic.o main.o test.o
# The name of your program
PROGRAM = ./test

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

2.5 FibLFSR Source Code

Listing 4: FibLFSR.cpp

```
#include "FibLFSR.hpp"
#include <bitset>
#include <sstream>
#include <string>

```

```cpp
namespace PhotoMagic {

    FibLFSR::FibLFSR(const std::string& seed) {
        if (seed.length() != SEED_LENGTH) {
            const std::string message =
                "The length of seed should be " + std::to_string(SEED_LENGTH);
            throw std::invalid_argument(message);
        }
        for (const char& bit : seed) {
            if (bit != '0' && bit != '1') {
                const std::string message =
                    "Each character in the seed should either be '0' or '1'";
                throw std::invalid_argument(message);
            }
        }

        lfsr = std::bitset<SEED_LENGTH>{ seed };
    }

    int FibLFSR::generate(const int k) {
        int ans = 0;
        for (int i = 0; i < k; ++i) {
            ans = (ans << 1) | step();
        }
        return ans;
    }

    int FibLFSR::step() {
        int ans = lfsr[SEED_LENGTH - 1];
        for (const int& tapIndex : tapIndexes) {
            ans ^= lfsr[tapIndex];
        }
        lfsr <<= 1;
        lfsr.set(0, ans);
        return ans;
    }

    std::string FibLFSR::getLfsrBinaryString() const { return lfsr.to_string(); }

    std::ostream& operator<<(std::ostream& os, const FibLFSR& lfsr) {
        os << lfsr.getLfsrBinaryString();
        return os;
    }

}  // namespace PhotoMagic
```

2.6 Unit Tests

Listing 5: test.cpp

```cpp
#include <sstream>
#include <string>
#include "FibLFSR.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
```

```
7
8  #include <boost/test/unit_test.hpp>
9
10 using PhotoMagic::FibLFSR;
11
12 BOOST_AUTO_TEST_CASE(testStepInstr) {
13     FibLFSR l("1011011000110110");
14     BOOST_REQUIRE_EQUAL(l.step(), 0);
15     BOOST_REQUIRE_EQUAL(l.step(), 0);
16     BOOST_REQUIRE_EQUAL(l.step(), 0);
17     BOOST_REQUIRE_EQUAL(l.step(), 1);
18 }
19
20 BOOST_AUTO_TEST_CASE(testGenerateInstr) {
21     FibLFSR l("1011011000110110");
22     BOOST_REQUIRE_EQUAL(l.generate(9), 51);
23 }
24
25 BOOST_AUTO_TEST_CASE(testOutputOperator) {
26     const std::string initialLFSR = "0110110001101100";
27     const FibLFSR l(initialLFSR);
28     std::stringstream ss;
29     ss << l;
30     BOOST_REQUIRE_EQUAL(ss.str(), initialLFSR);
31 }
32
33 BOOST_AUTO_TEST_CASE(testGenerateAndOutput) {
34     const std::string initialLFSR = "0110110001101100";
35     const std::string expectedLFSRAfterGenerate = "1101100001100110";
36     FibLFSR l(initialLFSR);
37     l.generate(9);
38
39     std::stringstream ss;
40     ss << l;
41     BOOST_REQUIRE_EQUAL(ss.str(), expectedLFSRAfterGenerate);
42 }
```

3 PS1b: PhotoMagic

3.1 Discussion

This project utilizes the FibLFSR class to encrypt and decrypt images through a process of bitwise manipulation. Using SFML, the program reads an image, applies transformations using the LFSR algorithm, and displays both the original and transformed images side by side. This demonstrates the practical applications of LFSR in data encryption.

3.2 Key Algorithms and Data Structures

The primary algorithm involves using the LFSR to generate pseudo-random binary sequences that are XORed with the pixel values of an image. The `FibLFSR::generate` function produces the binary sequences used for encrypting the RGB components of the image.

3.3 Learnings

This project expanded my understanding of image manipulation and the integration of graphical libraries with encryption algorithms. I also gained experience handling file I/O for reading and writing images and managing resources in SFML.

3.4 Makefile

Listing 6: Makefile for PS1b

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
DEPS = FibLFSR.hpp PhotoMagic.hpp
OBJECTS = FibLFSR.o PhotoMagic.o main.o test.o
PROGRAM = ./PhotoMagic

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

3.5 Main Source Code

Listing 7: main.cpp

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

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

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

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

    sf::RenderWindow window1(sf::VideoMode(image.getSize().x, image.getSize
   ().y),
        "Original Image");

    sf::Texture texture;
    texture.loadFromImage(image);
```

```
    sf::Sprite sprite(texture);

    FibLFSR lfsr(lfsrSeed);

    PhotoMagic::transform(&image, &lfsr);

    sf::RenderWindow window2(sf::VideoMode(image.getSize().x, image.getSize
    ().y),
        "Transformed Image");

    sf::Texture transformedTexture;
    transformedTexture.loadFromImage(image);
    sf::Sprite transformedSprite(transformedTexture);

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

        window1.clear();
        window1.draw(sprite);
        window1.display();

        window2.clear();
        window2.draw(transformedSprite);
        window2.display();
    }

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

    return 0;
}
```

3.6 PhotoMagic Implementation

Listing 8: PhotoMagic.cpp

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

namespace PhotoMagic {

    void transform(sf::Image* image, FibLFSR* lfsr) {
        unsigned int width = image->getSize().x;
        unsigned int height = image->getSize().y;

        for (unsigned int y = 0; y < height; ++y) {
            for (unsigned int x = 0; x < width; ++x) {
                sf::Color pixelColor = image->getPixel(x, y);

```

```
                uint8_t rKey = lfsr->generate(8);
                uint8_t gKey = lfsr->generate(8);
                uint8_t bKey = lfsr->generate(8);

                uint8_t newR = pixelColor.r ^ rKey;
                uint8_t newG = pixelColor.g ^ gKey;
                uint8_t newB = pixelColor.b ^ bKey;

                sf::Color newColor(newR, newG, newB);
                image->setPixel(x, y, newColor);
            }
        }
    }

}  // namespace PhotoMagic
```

3.7 Evidence of Code Execution

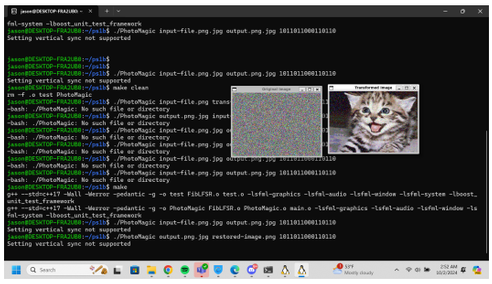

Figure 3: Original image

Figure 4: Original image

Figure 5: Original image

3.8 Test Cases

Figure 6: Original image

Figure 7: Original image

Listing 9: test.cpp

```
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include "FibLFSR.hpp"

BOOST_AUTO_TEST_CASE(testStepInstr1) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
}

BOOST_AUTO_TEST_CASE(testStepInstr2) {
    FibLFSR l2("1011011000110110");
    BOOST_REQUIRE_EQUAL(l2.generate(9), 51);
}

BOOST_AUTO_TEST_CASE(testConstructorInvalidSeed) {
    BOOST_CHECK_THROW(FibLFSR l("10110110001101"), std::runtime_error);
```

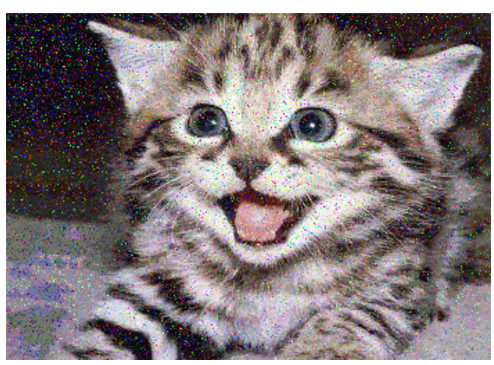

Figure 8: Original image

```
}
```

3.9 The Code Base

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

DEPS = FibLFSR.hpp PhotoMagic.hpp

OBJECTS = FibLFSR.o PhotoMagic.o test.o main.o

PROGRAM = test
PHOTOMAGIC_PROGRAM = PhotoMagic

LIBRARY = PhotoMagic.a

.PHONY: all clean lint

all: $(PROGRAM) $(LIBRARY) $(PHOTOMAGIC_PROGRAM)

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

$(PROGRAM): test.o FibLFSR.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

$(PHOTOMAGIC_PROGRAM): PhotoMagic.o FibLFSR.o main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

$(LIBRARY): FibLFSR.o PhotoMagic.o
    ar rcs $@ $^

clean:
    rm -f .o $(PROGRAM) $(PHOTOMAGIC_PROGRAM) $(LIBRARY)

lint:
    cpplint.cpp *.hpp
```

```
// Copyright 2024 Jason Ossai
#include <iostream>
#include "PhotoMagic.hpp"   //  Include directory for project headers
#include "FibLFSR.hpp"  //  Include directory for project headers

int main(int argc, char** argv) {
    if (argc != 4) {
        std::cerr << "Usage: " << argv[0]
            << " input-file.png output-file.png LFSR-seed"
            << std::endl;  //  Break line to avoid exceeding 80 characters
        return 1;
    }

    //  Read command line arguments
    const std::string inputFile = argv[1];
    const std::string outputFile = argv[2];
    const std::string lfsrSeed = argv[3];

    //  Load the source image
```

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

    //  Display the original image
    sf::RenderWindow window1(sf::VideoMode(image.getSize().x,
        image.getSize().y),
        "Original Image");

    sf::Texture texture;
    texture.loadFromImage(image);
    sf::Sprite sprite(texture);

    //  Create and initialize FibLFSR with the seed
    FibLFSR lfsr(lfsrSeed);  // Assuming it accepts a seed

    //  Transform the image using the FibLFSR
    PhotoMagic::transform(&image, &lfsr);

    //  Display the transformed image
    sf::RenderWindow window2(sf::VideoMode(image.getSize().x,
        image.getSize().y),
        "Transformed Image");

    sf::Texture transformedTexture;
    transformedTexture.loadFromImage(image);
    sf::Sprite transformedSprite(transformedTexture);

    //  Event loop for both windows
    while (window1.isOpen() && window2.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window1.close();
        }
        while (window2.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window2.close();
        }

        window1.clear();
        window1.draw(sprite);
        window1.display();

        window2.clear();
        window2.draw(transformedSprite);
        window2.display();
    }

    //  Save the transformed image to disk
    if (!image.saveToFile(outputFile)) {
        std::cerr << "Error saving image: " << outputFile << std::endl;
        return 1;
    }

    return 0;
}
```

```cpp
#include "PhotoMagic.hpp"  // Include the directory for the header file
#include "FibLFSR.hpp"     // Include the directory for the FibLFSR header

namespace PhotoMagic {

    // Transforms image using FibLFSR
    void transform(sf::Image* image, FibLFSR* lfsr) {
        // Get image dimensions
        unsigned int width = image->getSize().x;
        unsigned int height = image->getSize().y;

        for (unsigned int y = 0; y < height; ++y) {
            for (unsigned int x = 0; x < width; ++x) {
                // Get the original pixel color
                sf::Color pixelColor = image->getPixel(x, y);

                // Generate 8-bit keys for red, green, and blue components
                uint8_t rKey = lfsr->generate(8);  // Generate 8 bits for red
                uint8_t gKey = lfsr->generate(8);  // Generate 8 bits for green
                uint8_t bKey = lfsr->generate(8);  // Generate 8 bits for blue

                // XOR the pixel components with the generated keys
                uint8_t newR = pixelColor.r ^ rKey;
                uint8_t newG = pixelColor.g ^ gKey;
                uint8_t newB = pixelColor.b ^ bKey;

                // Set the new pixel color
                sf::Color newColor(newR, newG, newB);
                image->setPixel(x, y, newColor);
            }
        }
    }

}  // namespace PhotoMagic
```

```cpp
// Copyright 2024 Jason Ossai

#ifndef _HOME_JASON_PS1B_PHOTOMAGIC_HPP_
#define _HOME_JASON_PS1B_PHOTOMAGIC_HPP_

#include <algorithm>  // C++ system header
#include <SFML/Graphics.hpp>  // C++ system header

#include "FibLFSR.hpp"  // Project header file

namespace PhotoMagic {
    // Transforms the image using FibLFSR
    void transform(sf::Image* image, FibLFSR* lfsr);  // Change to pointer
}

#endif  // _HOME_JASON_PS1B_PHOTOMAGIC_HPP_
```

```cpp
// Copyright 2024 Aadit Engineer
#ifndef _HOME_JASON_PS1B_FIBLFSR_HPP_
#define _HOME_JASON_PS1B_FIBLFSR_HPP_
#include <iostream>
```

```hpp
#include <string>

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

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

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

    private:
        std::string seed;
        int tap;
    };

// Two spaces before the comment to follow the style guide
std::ostream& operator<<(std::ostream& out, const FibLFSR& fib);

#endif  // _HOME_JASON_PS1B_FIBLFSR_HPP_
```

```cpp
// Copyright 2024 Aadit Engineer

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

FibLFSR::FibLFSR(const std::string initialSeed) {
    seed = initialSeed;

    if (initialSeed.length() != 16) {
        throw std::runtime_error("Seed length is not 16 bits");
    }

    for (char c : initialSeed) {
        if (c != '0' && c != '1') {
            throw std::runtime_error(
                "Invalid seed input; must be either '0' or '1'");
        }
    }
}

int FibLFSR::step() {
    int outputBit = seed[0] ^ seed[2] ^ seed[3] ^ seed[5];

    for (size_t i = 0; i < 15; i++) {
        seed[i] = seed[i + 1];
    }

    seed[15] = (outputBit == 0) ? '0' : '1';

    return outputBit;
}

int FibLFSR::generate(int k) {
    int result = 0;
    for (int i = 0; i < k; ++i) {
        result *= 2;
        result += step();
```

```
    }
    return result;
}

std::ostream& operator<<(std::ostream& output, const FibLFSR& fib) {
    output << fib.seed;
    return output;
}
```

```
// Copyright 2022
// By Dr. Rykalova
// Edited by Dr. Daly
// test.cpp for PS1a
// updated 5/12/2022

#include <iostream>
#include <string>

#include "FibLFSR.hpp"

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

BOOST_AUTO_TEST_CASE(testStepInstr1) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 0);
    BOOST_REQUIRE_EQUAL(l.step(), 1);
}

BOOST_AUTO_TEST_CASE(testStepInstr2) {
    FibLFSR l2("1011011000110110");
    BOOST_REQUIRE_EQUAL(l2.generate(9), 51);
}

BOOST_AUTO_TEST_CASE(testConstructorInvalidSeed) {
    BOOST_CHECK_THROW(FibLFSR l("10110110001101"), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(testConstructorInvalidInput) {
    BOOST_CHECK_THROW(FibLFSR l("101A011000110110"), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(testGenerateZeroBits) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.generate(0), 0);
}

BOOST_AUTO_TEST_CASE(testGenerateMaxBits) {
    FibLFSR l("1011011000110110");
    BOOST_REQUIRE_EQUAL(l.generate(16), 6557);
}
```

4 PS2: Pentaflake

4.1 Discussion

This project implemented a recursive drawing of a Pentaflake, a fractal pattern based on the geometry of a pentagon. The goal was to explore geometric transformations, recursive programming, and graphical rendering using the SFML library.

4.2 Key Algorithms and Data Structures

The main algorithm involved a recursive function to draw smaller pentagons around a central pentagon. This recursive approach allowed for the creation of a fractal pattern. The SFML library was used for rendering, and `sf::ConvexShape` was utilized to define custom shapes like the pentagon.

4.3 Learnings

This project deepened my understanding of recursion in graphical programming. I also gained experience in using SFML for geometric transformations and rendering complex patterns. Working in a pair programming environment helped me learn from my partner's strengths and approach to problem-solving.

4.4 Makefile

Listing 10: Makefile for PS2

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-window -lsfml-system
# Your .hpp files
DEPS = penta.hpp
# Your compiled .o files
OBJECTS = penta.o main.o
# The name of your program
PROGRAM = ./Penta

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

4.5 Main Source Code

Listing 11: main.cpp

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

```

```
int main(int argc, char** argv) {
    if (argc < 3) {
        std::cerr << "Usage: Penta L N" << std::endl;
        return 1;
    }

    double L = std::stod(argv[1]);
    int N = std::stoi(argv[2]);

    sf::RenderWindow window(sf::VideoMode(2 * L, 2 * L), "Pentaflake");

    Pentaflake pentaflake(L, N);

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

        window.clear();
        window.draw(pentaflake);
        window.display();
    }

    return 0;
}
```

4.6 Pentaflake Class Source Code

Listing 12: penta.cpp

```
#include "penta.hpp"

const double PI = 3.141592653589793;
const double rd = 180 / PI;
const double angleOffset = 72.0;  // For a pentagon, 72 degrees between
   vertices

Pentaflake::Pentaflake(float l, int n) : L(l), N(n) {}

void Pentaflake::drawPentaflake(sf::RenderTarget& target, sf::Vector2f
   center,
    float size, int depth, double theta) const {
    if (depth <= 0) {
        // Draw a pentagon
        sf::ConvexShape pentagon(5);
        for (int i = 0; i < 5; ++i) {
            float angle = (angleOffset * i + theta * rd) * PI / 180.0;
            pentagon.setPoint(i, sf::Vector2f(size * cos(angle), size * sin(
   angle)));
        }
        pentagon.setOrigin(size, size);
        pentagon.setFillColor(sf::Color::White);
        pentagon.setPosition(center);
        pentagon.rotate(theta * rd);
        target.draw(pentagon);
    } else {
        drawPentaflake(target, center, size / 3.0, depth - 1, theta);
```

```
25
26          for (int i = 0; i < 5; ++i) {
27              double newTheta = (angleOffset * i) * PI / 180.0;
28              sf::Vector2f newCenter = center + sf::Vector2f(size * 2 / 3 *
    cos(newTheta),
29              size * 2 / 3 * sin(newTheta));
30              drawPentaflake(target, newCenter, size / 3.0, depth - 1, theta);
31          }
32      }
33  }
34
35  void Pentaflake::draw(sf::RenderTarget& target, sf::RenderStates states)
    const {
36      sf::Vector2f center(target.getSize().x / 2, target.getSize().y / 2);
37      drawPentaflake(target, center, L, N, 0);
38  }
```

4.7 Evidence of Code Execution

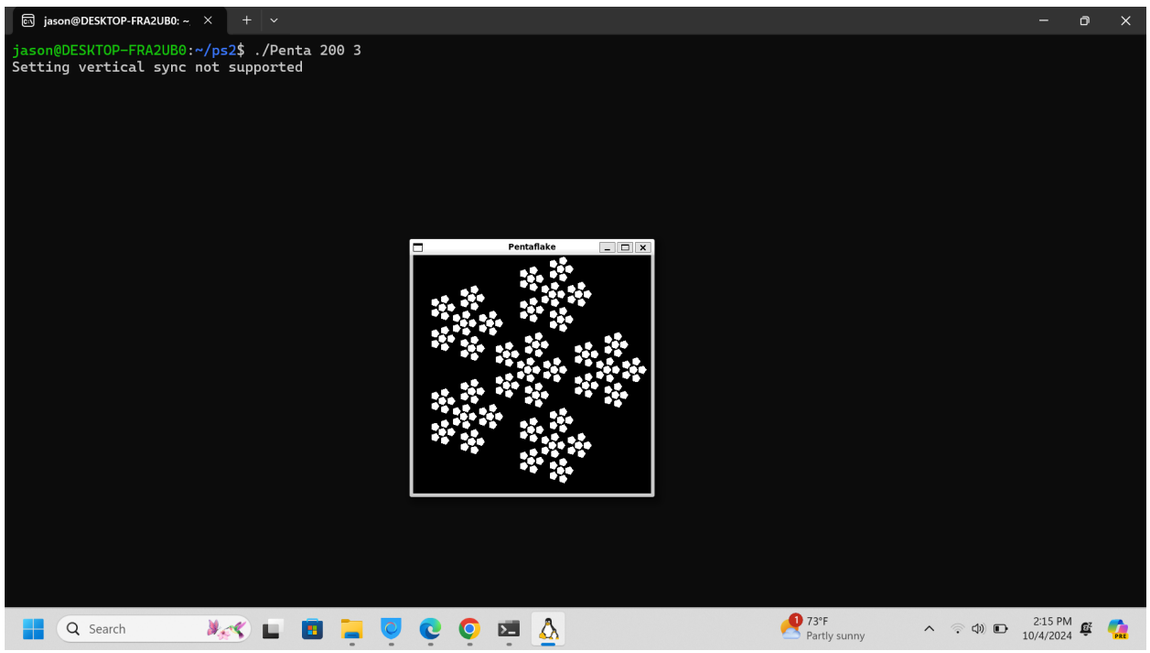

Figure 9: Pentaflake output with specified parameters.

5 PS3a: N-Body Simulation (Static)

5.1 Discussion

This project simulates and visualizes celestial mechanics using an N-body simulation. It uses SFML to render the positions and movements of celestial bodies in a static universe, showcasing their gravitational interactions and orbital dynamics. The universe data is read from an external file, and the celestial bodies are displayed graphically in real-time.

5.2 Key Algorithms and Data Structures

The main algorithms include reading celestial body data from a file, computing their positions relative to the window, and rendering them using SFML. The data structures used include `std::vector` to store celestial bodies and custom classes like `CelestialBody` and `Universe` for encapsulation and modularity.

5.3 Learnings

This project deepened my understanding of how to integrate graphical rendering with simulation logic. I learned about using SFML for creating visually engaging simulations, designing robust error handling, and managing class responsibilities effectively in a collaborative environment.

5.4 Makefile

Listing 13: Makefile for PS3a

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
DEPS = CelestialBody.hpp Universe.hpp
OBJECTS = CelestialBody.o Universe.o main.o test.o
PROGRAM = ./universe_sim

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

5.5 Main Source Code

Listing 14: main.cpp

```
#include <iostream>
#include <SFML/Graphics.hpp>
#include "Universe.hpp"
int main() {
    const int WINDOW_WIDTH = 800;
```

```
    const int WINDOW_HEIGHT = 600;

    sf::RenderWindow window(sf::VideoMode(WINDOW_WIDTH, WINDOW_HEIGHT), "
    SFML Universe");

    Universe universe;
    std::cin >> universe;

    double conversionFactor = WINDOW_WIDTH / (2.0 * universe.
    getUniverseRadius());

    for (auto& body : universe.getBodies()) {
        body.setPositionInWindow(conversionFactor);
    }

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

        window.clear();
        window.draw(universe);
        window.display();
    }

    return 0;
}
```

5.6 CelestialBody Implementation

Listing 15: CelestialBody.cpp

```
#include "CelestialBody.hpp"
#include <string>
CelestialBody::CelestialBody() {}

void CelestialBody::setPositionInWindow(double conversionFactor) {
    double pixelX = x * conversionFactor + 400;
    double pixelY = -y * conversionFactor + 300;
    sprite.setPosition(static_cast<float>(pixelX), static_cast<float>(pixelY
    ));
}
sf::Vector2f CelestialBody::getSpritePosition() const {
    return sprite.getPosition();
}

double CelestialBody::getX() const {
    return x;
}

double CelestialBody::getY() const {
    return y;
}

double CelestialBody::getMass() const {
    return mass;
}
```

```
25
26 std::istream& operator>>(std::istream& in, CelestialBody& body) {
27     std::string filename;
28     in >> body.x >> body.y >> body.xvel >> body.yvel >> body.mass >>
       filename;
29     body.texture.loadFromFile(filename);
30     body.sprite.setTexture(body.texture);
31     return in;
32 }
33
34 std::ostream& operator<<(std::ostream& out, const CelestialBody& body) {
35     out << body.x << " " << body.y << " " << body.xvel << " " << body.yvel
       << " " << body.mass;
36     return out;
37 }
38
39 void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states)
       const {
40     target.draw(sprite, states);
41 }
```

5.7 Test Cases

Listing 16: test.cpp

```
1  #define BOOST_TEST_MODULE UniverseTest
2  #include <boost/test/included/unit_test.hpp>
3  #include "CelestialBody.hpp"
4  #include "Universe.hpp"
5
6  BOOST_AUTO_TEST_CASE(CelestialBodyReadStreamTest) {
7      CelestialBody body;
8      std::stringstream ss;
9      ss << "1.0 2.0 3.0 4.0 5.0 earth.gif";
10     ss >> body;
11
12     BOOST_CHECK_EQUAL(body.getX(), 1.0);
13     BOOST_CHECK_EQUAL(body.getY(), 2.0);
14     BOOST_CHECK_EQUAL(body.getMass(), 5.0);
15 }
```

5.8 Evidence of Code Execution

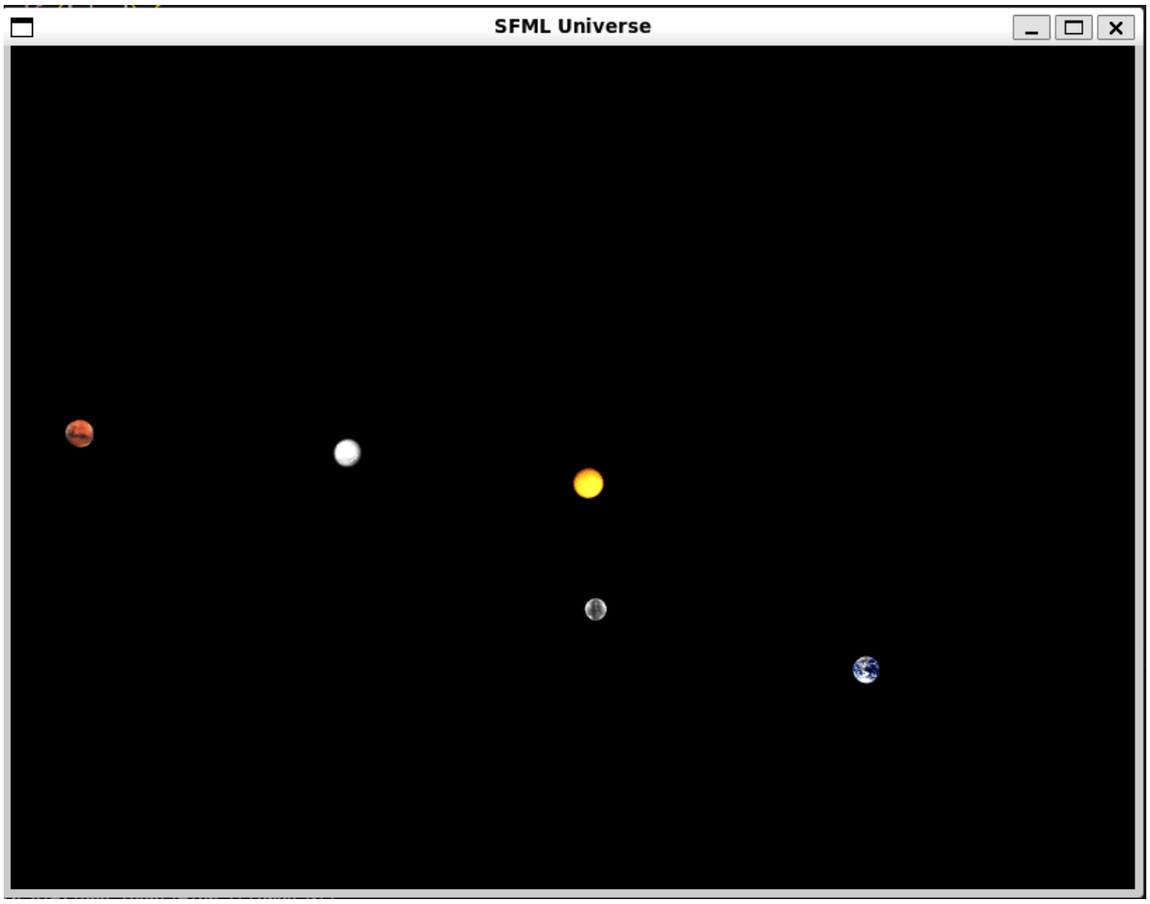

Figure 10: N-Body simulation displaying celestial bodies.

6 PS3b: N-Body Simulation (Dynamic)

6.1 Discussion

This project builds upon the static N-body simulation by introducing dynamic physics and animation. It models the motion of celestial bodies in 2D space, influenced by gravitational forces, and visualizes their interactions over time using SFML. The simulation iteratively updates the positions and velocities of the celestial bodies.

6.2 Key Algorithms and Data Structures

- **Physics Simulation**: Newton's law of universal gravitation was applied to compute the forces between all celestial bodies.
- **Velocity and Position Updates**: The velocity and position of each celestial body were updated at each time step using the equations of motion.
- **Smart Pointers**: Used to manage celestial body objects efficiently, avoiding memory leaks.
- **Animation**: The simulation continuously animates the bodies as they move under gravitational influence.

6.3 Learnings

This project enhanced my understanding of dynamic simulations and the application of physics in programming. I learned to implement real-time updates, handle numerical stability issues, and improve memory safety using smart pointers.

6.4 Makefile

Listing 17: Makefile for PS3b

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
DEPS = CelestialBody.hpp Universe.hpp
OBJECTS = CelestialBody.o Universe.o main.o test.o
PROGRAM = ./nbody_sim

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

6.5 Main Source Code

Listing 18: main.cpp

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

int main() {
    const int WINDOW_WIDTH = 800;
    const int WINDOW_HEIGHT = 600;
    const double dt = 25000.0;  // Time step for the simulation
    bool isPaused = false;

    sf::RenderWindow window(sf::VideoMode(WINDOW_WIDTH, WINDOW_HEIGHT), "
   SFML Universe");

    Universe universe;
    std::cin >> universe;

    double conversionFactor = WINDOW_WIDTH / (2.0 * universe.
   getUniverseRadius());

    for (auto& body : universe.getBodies()) {
        body.setPositionInWindow(conversionFactor);
    }

    while (window.isOpen()) {
        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::Space) {
                isPaused = !isPaused;
            }
        }

        window.clear();

        if (!isPaused) {
            universe.step(dt);
            for (auto& body : universe.getBodies()) {
                body.setPositionInWindow(conversionFactor);
            }
        }

        window.draw(universe);
        window.display();
    }

    return 0;
}
```

6.6 Universe Implementation

Listing 19: Universe.cpp

```
#include "Universe.hpp"
#include <cmath>

const double Universe::G = 6.67e-11;

Universe::Universe() : num_planets(0), universe_radius(2.50e+11) {
    bodies.resize(num_planets);
}

double Universe::getUniverseRadius() const {
    return universe_radius;
}

std::vector<CelestialBody>& Universe::getBodies() {
    return bodies;
}

std::istream& operator>>(std::istream& in, Universe& universe) {
    in >> universe.num_planets >> universe.universe_radius;
    universe.bodies.resize(universe.num_planets);
    for (int i = 0; i < universe.num_planets; ++i) {
        in >> universe.bodies[i];
    }
    return in;
}

void Universe::step(double dt) {
    std::vector<double> changesInXvel(bodies.size(), 0);
    std::vector<double> changesInYvel(bodies.size(), 0);

    for (size_t i = 0; i < bodies.size(); i++) {
        for (size_t j = 0; j < bodies.size(); j++) {
            if (i != j) {
```

```
                double dx = bodies[j].getX() - bodies[i].getX();
                double dy = bodies[j].getY() - bodies[i].getY();
                double r = sqrt(dx * dx + dy * dy);
                double F = G * bodies[i].getMass() * bodies[j].getMass() / (
    r * r * r);
                changesInXvel[i] += F * dx / bodies[i].getMass() * dt;
                changesInYvel[i] += F * dy / bodies[i].getMass() * dt;
            }
        }
    }

    for (size_t i = 0; i < bodies.size(); i++) {
        bodies[i].updateVel(changesInXvel[i], changesInYvel[i], 1);
        bodies[i].updatep(dt);
    }
}

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

6.7 Test Cases

Listing 20: test.cpp

```
#define BOOST_TEST_MODULE UniverseTest
#include <boost/test/included/unit_test.hpp>
#include "CelestialBody.hpp"
#include "Universe.hpp"

BOOST_AUTO_TEST_CASE(CelestialBodyUpdateTest) {
    CelestialBody body;
    std::stringstream ss;
    ss << "1.0 2.0 3.0 4.0 5.0 earth.gif";
    ss >> body;

    body.updateVel(0.5, 0.5, 1.0);
    body.updatep(1.0);

    BOOST_CHECK_CLOSE(body.getX(), 4.5, 1e-6);
    BOOST_CHECK_CLOSE(body.getY(), 6.5, 1e-6);
    BOOST_CHECK_CLOSE(body.getMass(), 5.0, 1e-6);
}
```

6.8 Evidence of Code Execution

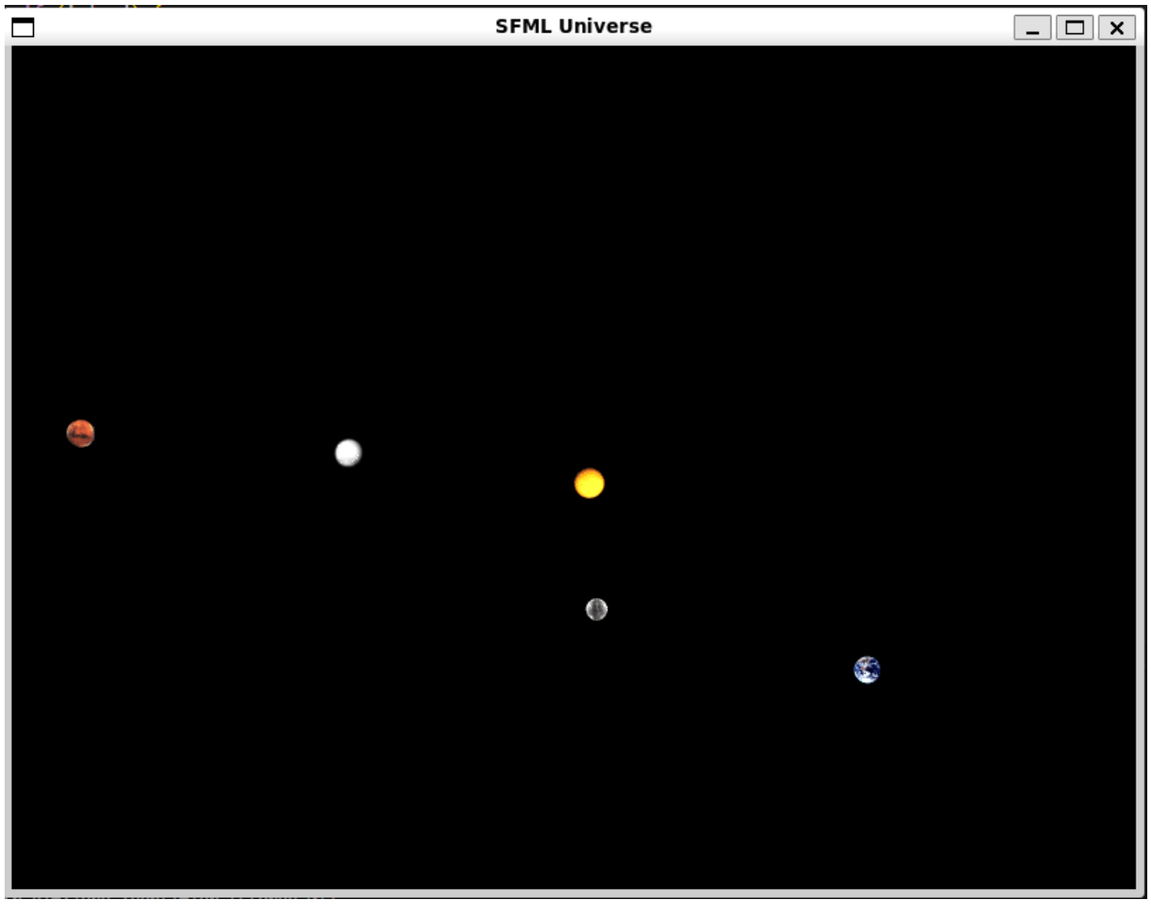

Figure 11: Dynamic N-Body simulation showing gravitational interactions.

7 PS4a: Sokoban

7.1 Discussion

This project implements a graphical Sokoban game using SFML. Players move boxes to designated storage locations in a grid-based environment. The project focuses on rendering, event handling, and implementing player movement mechanics.

7.2 Key Algorithms and Data Structures

- **Grid Representation**: The game grid is stored as a 2D vector of characters, where each character represents a tile type (e.g., wall, player, box, storage).
- **Event Handling**: Keypress events are used to determine player movement direction and validate moves.
- **Rendering**: SFML is used to render the game grid and sprites.

7.3 Learnings

This project enhanced my understanding of grid-based game logic, including collision detection and boundary checks. I also gained experience integrating SFML for graphics rendering and input handling.

7.4 Makefile

Listing 21: Makefile for PS4a

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-window -lsfml-system
DEPS = Sokoban.hpp
OBJECTS = Sokoban.o main.o
PROGRAM = ./sokoban_game

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

7.5 Main Source Code

Listing 22: main.cpp

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

int main() {
    Sokoban game("level1.lvl");
```

```
7
8       sf::RenderWindow window(sf::VideoMode(game.width() * Sokoban::TILE_SIZE,
9           game.height() * Sokoban::TILE_SIZE),
10          "Sokoban");
11
12      window.setVerticalSyncEnabled(true);
13
14      while (window.isOpen()) {
15          sf::Event event;
16
17          while (window.pollEvent(event)) {
18              if (event.type == sf::Event::Closed) {
19                  window.close();
20              }
21
22              if (event.type == sf::Event::KeyPressed) {
23                  Direction dir;
24                  bool validMove = true;
25
26                  switch (event.key.code) {
27                  case sf::Keyboard::Left:
28                      dir = Left;
29                      break;
30                  case sf::Keyboard::Right:
31                      dir = Right;
32                      break;
33                  case sf::Keyboard::Up:
34                      dir = Up;
35                      break;
36                  case sf::Keyboard::Down:
37                      dir = Down;
38                      break;
39                  default:
40                      validMove = false;
41                      break;
42                  }
43
44                  if (validMove) {
45                      game.movePlayer(dir);
46                  }
47              }
48          }
49
50          window.clear();
51          window.draw(game);
52          window.display();
53      }
54
55      return 0;
56  }
```

7.6 Sokoban Implementation

Listing 23: Sokoban.cpp

```
1  #include "Sokoban.hpp"
2  #include <iostream>
3  #include <fstream>
4
```

```
5  Sokoban::Sokoban() {
6      _width = 12;
7      _height = 12;
8  }
9
10 std::istream& operator>>(std::istream& is, Sokoban& game) {
11     is >> game._height >> game._width;
12     game.grid.resize(game._height, std::vector<char>(game._width));
13
14     for (int i = 0; i < game._height; i++) {
15         for (int j = 0; j < game._width; j++) {
16             is >> game.grid[i][j];
17             if (game.grid[i][j] == '@') {
18                 game.playerPosition = { j, i };
19             }
20         }
21     }
22     return is;
23 }
24
25 Sokoban::Sokoban(const std::string& levelFilename) {
26     if (!wallTexture.loadFromFile("block_06.png") ||
27         !playerTexture.loadFromFile("player_05.png") ||
28         !boxTexture.loadFromFile("crate_03.png") ||
29         !storageTexture.loadFromFile("ground_04.png") ||
30         !emptyStoragetexture.loadFromFile("ground_01.png")) {
31         std::cerr << "Failed to load one or more textures." << std::endl;
32     }
33     playerSprite.setTexture(playerTexture);
34
35     std::ifstream inFile(levelFilename);
36     if (!inFile) {
37         std::cerr << "Failed to open level file: " << levelFilename << std::
       endl;
38         return;
39     }
40     inFile >> *this;
41     inFile.close();
42 }
43
44 bool Sokoban::movePlayer(Direction direction) {
45     sf::Vector2i movep(0, 0);
46
47     switch (direction) {
48     case Up:
49         movep.y = -1;
50         break;
51     case Down:
52         movep.y = 1;
53         break;
54     case Left:
55         movep.x = -1;
56         break;
57     case Right:
58         movep.x = 1;
59         break;
60     }
61
62     sf::Vector2i newPosition = playerPosition + movep;
```

```
63
64      if (newPosition.x < 0 || newPosition.x >= _width ||
65          newPosition.y < 0 || newPosition.y >= _height) {
66          return false;
67      }
68
69      char targetT = grid[newPosition.y][newPosition.x];
70      if (targetT == '#') {
71          return false;
72      }
73      else if (targetT == 'A' || targetT == '1') {
74          sf::Vector2i boxNewPosition = newPosition + movep;
75          char nextTile = grid[boxNewPosition.y][boxNewPosition.x];
76
77          if (nextTile == '.' || nextTile == 'a') {
78              grid[boxNewPosition.y][boxNewPosition.x] = (nextTile == 'a') ? '
    1' : 'A';
79              grid[newPosition.y][newPosition.x] = (targetT == '1') ? 'a' : '.
    ';
80          }
81          else {
82              return false;
83          }
84      }
85
86      grid[playerPosition.y][playerPosition.x] = (grid[playerPosition.y][
    playerPosition.x] == '1') ? 'a' : '.';
87      playerPosition = newPosition;
88      grid[playerPosition.y][playerPosition.x] = '@';
89
90      return true;
91  }
```

7.7 ReadMe

Listing 24: Readme-ps4.md

```
1  # PS4: Sokoban
2
3  ## Contact
4  Name:Jason Ossai
5  Section: COMP IV 2040
6  Time to Complete:4 hours
7
8
9  ## Description
10 The Sokoban project introduces a graphical Sokoban game, requiring players
       to push boxes into designated storage locations, with mechanics to be
       developed in Part B.
11
12 ### Features
13 The development process involved several significant decisions, including:
14
15 #### Part a
16 1.) Implemented a robust method for reading level dimensions and the game
       grid from a text file format.
17
18 2.)The game was rendered using the SFML library, which significantly
       improved graphics handling and event management.
```

```
19
20 #### Part b
21 1.) Game mechanics: Strategies for managing player mobility, box pushing,
       and win condition verification are planned and will be put into practice
        in Part B.
22
23 ## Issues
24 I had lots of lint and valgrind errors which I was unable to fix all. By
       doing so it changes my code and gives more errors.
25
26 ## Extra Credit
27 I didn't attempt the extra credit.
```

7.8 Evidence of Code Execution

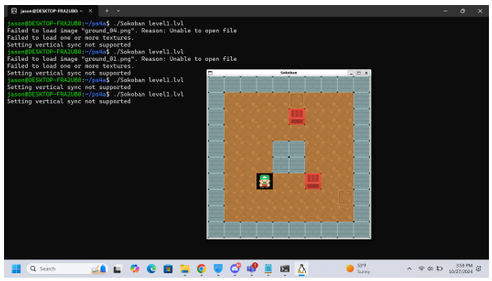

Figure 12: Original image

8 PS4b: Sokoban Enhanced

8.1 Discussion

This project enhances the Sokoban game developed in PS4a by introducing additional functionalities, including undo and reset features, sound effects, and more robust player movement mechanics. The project continues to leverage SFML for rendering and event handling.

8.2 Key Algorithms and Data Structures

- **Undo and Reset**: Implemented using a stack to save game states for undo functionality and resetting to the initial state of the game.
- **Sound Effects**: Introduced background music and sound effects for player actions and game completion.
- **Tile-Based Rendering**: Extended rendering logic with reusable components for tiles and sprites.

8.3 Learnings

This phase of the project significantly enhanced my understanding of advanced game development concepts, including managing game states, implementing audio features, and handling user input efficiently. It also emphasized modularity and code reuse.

8.4 Makefile

Listing 25: Makefile for PS4b

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

```
DEPS = Sokoban.hpp SokobanTileGrid.hpp SokobanPlayer.hpp
OBJECTS = Sokoban.o SokobanTileGrid.o SokobanPlayer.o main.o test.o
PROGRAM = ./sokoban_game_enhanced

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

8.5 Main Source Code

Listing 26: main.cpp

```
#include <fstream>
#include <iostream>
#include "Sokoban.hpp"

int main(const int size, const char* arguments[]) {
    if (size < 2) {
        std::cout << "Too few arguments! Require the filename of the level file." << std::endl;
        return 1;
    }

    const std::string levelFilename{ arguments[1] };
    SB::Sokoban sokoban{ levelFilename };

    const auto windowWidth{ sokoban.width() * SB::TILE_WIDTH };
    const auto windowHeight{ sokoban.height() * SB::TILE_HEIGHT };
    const auto windowVideoMode{ sf::VideoMode(windowWidth, windowHeight) };
    const auto windowTitle = SB::GAME_NAME + " by " + SB::AUTHOR_NAME;
    sf::RenderWindow window(windowVideoMode, windowTitle);
    window.setFramerateLimit(60);

    const std::unordered_map<const sf::Keyboard::Key, SB::Direction> movePlayerKeyMap{
        { sf::Keyboard::Key::W, SB::Direction::Up },
        { sf::Keyboard::Key::A, SB::Direction::Left },
        { sf::Keyboard::Key::S, SB::Direction::Down },
        { sf::Keyboard::Key::D, SB::Direction::Right },
        { sf::Keyboard::Key::Up, SB::Direction::Up },
        { sf::Keyboard::Key::Left, SB::Direction::Left },
        { sf::Keyboard::Key::Down, SB::Direction::Down },
        { sf::Keyboard::Key::Right, SB::Direction::Right }
    };

    sf::Clock clock;
    while (window.isOpen()) {
```

```
34          sf::Event event{};
35          while (window.pollEvent(event)) {
36              if (event.type == sf::Event::Closed) {
37                  window.close();
38                  break;
39              }
40
41              if (event.type == sf::Event::KeyPressed) {
42                  const auto itDirection = movePlayerKeyMap.find(event.key.
    code);
43                  if (itDirection != movePlayerKeyMap.end()) {
44                      sokoban.movePlayer(itDirection->second);
45                  }
46
47                  if (event.key.code == sf::Keyboard::R) {
48                      sokoban.reset();
49                  }
50
51                  if (event.key.code == sf::Keyboard::U) {
52                      sokoban.undo();
53                  }
54              }
55          }
56
57          sokoban.update(clock.restart().asMicroseconds());
58
59          if (window.isOpen()) {
60              window.clear(sf::Color::White);
61              window.draw(sokoban);
62              window.display();
63          }
64      }
65  }
```

8.6 Evidence of Code Execution

Figure 13: Original image

8.7 Test Cases

Listing 27: test.cpp

```
1  #define BOOST_TEST_DYN_LINK
2  #define BOOST_TEST_MODULE Main
3
4  #include <boost/test/unit_test.hpp>
5  #include "Sokoban.hpp"
6
7  BOOST_AUTO_TEST_CASE(testHeightWidth) {
```

```
    const SB::Sokoban sokoban{ "assets/level/level2.lvl" };

    BOOST_REQUIRE_EQUAL(sokoban.height(), 10);
    BOOST_REQUIRE_EQUAL(sokoban.width(), 12);
}

BOOST_AUTO_TEST_CASE(testPlayerPosition) {
    const SB::Sokoban sokoban{ "assets/level/level2.lvl" };

    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 8);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 5);
}

BOOST_AUTO_TEST_CASE(testMovePlayer) {
    SB::Sokoban sokoban{ "assets/level/level2.lvl" };
    sokoban.movePlayer(SB::Direction::Right);

    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().x, 9);
    BOOST_REQUIRE_EQUAL(sokoban.playerLoc().y, 5);
}
```

8.8 Sokoban Enhancements

Listing 28: Sokoban.cpp

```
#include "Sokoban.hpp"
#include <iostream>

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

    const auto prevState = m_stateStack.top();
    m_stateStack.pop();

    m_playerLoc = prevState.playerLoc;
    m_tileCharGrid = prevState.tileGrid;
    m_score = prevState.score;
}
```

9 PS5: DNA Sequence Alignment

9.1 Discussion

This project involves solving the problem of DNA sequence alignment using the Needleman-Wunsch algorithm, a dynamic programming approach. The goal is to align two DNA sequences to minimize mismatches and gaps, facilitating genetic analysis and comparative genomics.

9.2 Key Algorithms and Data Structures

- **Dynamic Programming Table**: The alignment process uses a 2D table to compute optimal alignment scores, storing intermediate results to avoid redundant calculations.
- **Penalty Calculation**: A function is used to determine mismatch penalties and gap costs.
- **Backtracking for Alignment**: After filling the DP table, a backtracking process reconstructs the aligned sequences.

9.3 Learnings

This project enhanced my understanding of dynamic programming, particularly in bioinformatics applications. I also gained experience with performance optimization techniques for handling large input sizes, such as those encountered with DNA sequences.

9.4 Makefile

Listing 29: Makefile for PS5

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lboost_unit_test_framework
DEPS = EDistance.hpp
OBJECTS = EDistance.o main.o test.o
PROGRAM = ./dna_align

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

9.5 Main Source Code

Listing 30: main.cpp

```
#include <iostream>
#include <string>
#include <SFML/System.hpp>
#include "EDistance.hpp"
```

```
5
6  int main() {
7      std::string x, y;
8
9      std::getline(std::cin, x);
10     std::getline(std::cin, y);
11
12     sf::Clock clock;
13
14     EDistance ed(x, y);
15
16     std::cout << "Edit distance = " << ed.optDistance() << std::endl;
17     std::cout << ed.alignment();
18
19     sf::Time t = clock.getElapsedTime();
20     std::cout << "Execution time is " << t.asSeconds() << " seconds\n";
21
22     return 0;
23 }
```

9.6 EDistance Implementation

Listing 31: EDistance.cpp

```
1  #include "EDistance.hpp"
2  #include <algorithm>
3  #include <iostream>
4
5  EDistance::EDistance(const std::string& x, const std::string& y)
6      : x(x), y(y), opt(x.length() + 1, std::vector<int>(y.length() + 1)) {
7      for (size_t i = 0; i <= x.length(); ++i) {
8          opt[i][y.length()] = 2 * (x.length() - i);
9      }
10     for (size_t j = 0; j <= y.length(); ++j) {
11         opt[x.length()][j] = 2 * (y.length() - j);
12     }
13     for (int i = static_cast<int>(x.length()) - 1; i >= 0; --i) {
14         for (int j = static_cast<int>(y.length()) - 1; j >= 0; --j) {
15             int cost = (x[i] == y[j]) ? 0 : 1;
16             opt[i][j] = min3(opt[i + 1][j + 1] + cost,
17                 opt[i + 1][j] + 2,
18                 opt[i][j + 1] + 2);
19         }
20     }
21 }
22
23 int EDistance::penalty(char a, char b) {
24     return a == b ? 0 : 1;
25 }
26
27 int EDistance::min3(int a, int b, int c) {
28     return std::min({ a, b, c });
29 }
30
31 int EDistance::optDistance() {
32     return opt[0][0];
33 }
34
35 std::string EDistance::alignment() {
```

```
    std::string result;
    std::size_t i = 0, j = 0;

    while (i < x.length() && j < y.length()) {
        if (opt[i][j] == opt[i + 1][j + 1] + penalty(x[i], y[j])) {
            result += std::string(1, x[i]) + " " + std::string(1, y[j]) + "
   " +
                std::to_string(penalty(x[i], y[j])) + "\n";
            ++i;
            ++j;
        }
        else if (opt[i][j] == opt[i + 1][j] + 2) {
            result += std::string(1, x[i]) + " - 2\n";
            ++i;
        }
        else {
            result += "- " + std::string(1, y[j]) + " 2\n";
            ++j;
        }
    }

    while (i < x.length()) {
        result += std::string(1, x[i]) + " - 2\n";
        ++i;
    }
    while (j < y.length()) {
        result += "- " + std::string(1, y[j]) + " 2\n";
        ++j;
    }

    return result;
}
```

9.7 Test Cases

Listing 32: test.cpp

```
#define BOOST_TEST_MODULE EDistanceTests
#include <boost/test/included/unit_test.hpp>
#include "EDistance.hpp"

BOOST_AUTO_TEST_CASE(TestConstructor) {
    BOOST_CHECK_NO_THROW(EDistance("AACAGTTACC", "TAAGGTCA"));
}

BOOST_AUTO_TEST_CASE(TestOptDistance) {
    EDistance ed("AACAGTTACC", "TAAGGTCA");
    BOOST_CHECK_EQUAL(ed.optDistance(), 7);
}

BOOST_AUTO_TEST_CASE(TestAlignment) {
    EDistance ed("AACAGTTACC", "TAAGGTCA");
    std::string expectedAlignment =
        "A T 1\nA A 0\nC - 2\nA A 0\nG G 0\nT G 1\nT T 0\nA - 2\nC C 0\nC A
   1\n";
    BOOST_CHECK_EQUAL(ed.alignment(), expectedAlignment);
}

BOOST_AUTO_TEST_CASE(TestEmptyStrings) {
```

```
22      EDistance ed("", "");
23      BOOST_CHECK_EQUAL(ed.optDistance(), 0);
24      BOOST_CHECK_EQUAL(ed.alignment(), "");
25  }
26
27  BOOST_AUTO_TEST_CASE(TestOneEmptyString) {
28      EDistance ed("AACAGTTACC", "");
29      BOOST_CHECK_EQUAL(ed.optDistance(), 20);
30      std::string expectedAlignment =
31          "A - 2\nA - 2\nC - 2\nA - 2\nG - 2\nT - 2\nT - 2\nA - 2\nC - 2\nC -
        2\n";
32      BOOST_CHECK_EQUAL(ed.alignment(), expectedAlignment);
33  }
```

9.8 ReadMe

Listing 33: Readme-ps5.md

```
1  # PS5: DNA Alignment
2
3  ## Contact
4  Name: Jason Ossai
5  Partner: Omar El-Rifai
6
7  ## Description
8  The DNA Sequence Alignment project uses the Needleman-Wunsch algorithm to
       align DNA sequences in a dynamic programming manner, minimizing gaps and
        mismatches.
```

9.9 Evidence of Code Execution

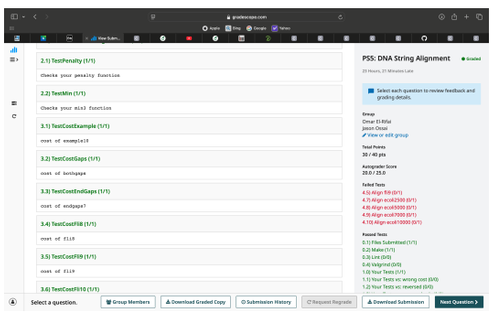

Figure 14: Original image

10 PS6: RandWriter - Markov Text Generation

10.1 Discussion

This project involves creating a Markov text generator using the RandWriter class. The generator analyzes input text, calculates the frequencies of letter sequences based on the specified Markov order, and generates a random text resembling the input text.

10.2 Key Algorithms and Data Structures

- **Markov Model Construction**: A map data structure is used to store k-grams as keys and their corresponding frequency distributions of following characters as values.
- **Random Text Generation**: The generation algorithm uses random sampling weighted by frequencies to predict the next character based on the k-gram.
- **Circular Input Handling**: The algorithm treats the input text as circular, allowing wrap-around for k-grams near the end of the string.

10.3 Learnings

This assignment introduced me to practical applications of Markov models in text generation. It enhanced my understanding of probability distributions and efficient random sampling. Debugging and optimizing algorithms for large datasets was a valuable experience.

10.4 Makefile

Listing 34: Makefile for PS6

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lboost_unit_test_framework
DEPS = RandWriter.hpp TextWriter.hpp
OBJECTS = RandWriter.o TextWriter.o main.o test.o
PROGRAM = ./rand_text_gen

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

10.5 Main Source Code

Listing 35: main.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include "RandWriter.hpp"

```

```cpp
int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: TextWriter <order> <length>\n";
        return 1;
    }
    int order = std::stoi(argv[1]);
    int length = std::stoi(argv[2]);

    std::string inputText, line;
    while (std::getline(std::cin, line)) {
        inputText += line + "\n";
    }

    try {
        RandWriter writer(inputText, order);
        std::string kgram = inputText.substr(0, order);
        std::cout << writer.generate(kgram, length) << std::endl;
    }
    catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    return 0;
}
```

10.6 RandWriter Implementation

Listing 36: RandWriter.cpp

```cpp
#include "RandWriter.hpp"
#include <stdexcept>
#include <iostream>
#include <ctime>
#include <random>

RandWriter::RandWriter(std::string text, int k) : text(text), k(k) {
    if (text.length() < static_cast<std::string::size_type>(k)) {
        throw std::invalid_argument("Text length must be at least k");
    }
    calculateFreq();
    std::srand(std::time(0));
}

void RandWriter::calculateFreq() {
    for (size_t i = 0; i < text.length(); ++i) {
        std::string kgram = text.substr(i, k);
        char nextChar = text[(i + k) % text.length()];
        kgram_freq[kgram][nextChar]++;
    }
}

int RandWriter::freq(std::string kgram) const {
    if (kgram.length() != static_cast<std::string::size_type>(k)) {
        throw std::invalid_argument("kgram is not of length k");
    }
    if (kgram_freq.find(kgram) != kgram_freq.end()) {
        int sum = 0;
        for (auto& pair : kgram_freq.at(kgram)) {
            sum += pair.second;
        }
```

```
        return sum;
    }
    return 0;
}

char RandWriter::kRand(std::string kgram) {
    if (kgram.length() != static_cast<std::string::size_type>(k)) {
        throw std::invalid_argument("kgram is not of length k");
    }
    if (kgram_freq.find(kgram) == kgram_freq.end()) {
        throw std::invalid_argument("No such kgram");
    }

    int totalFreq = freq(kgram);
    int c = std::rand() % totalFreq;

    for (const auto& pair : kgram_freq.at(kgram)) {
        c -= pair.second;
        if (c < 0) {
            return pair.first;
        }
    }

    throw std::runtime_error("Random character generation failed");
}

std::string RandWriter::generate(std::string kgram, int L) {
    if (kgram.length() != static_cast<std::string::size_type>(k)) {
        throw std::invalid_argument("kgram is not of length k");
    }
    if (static_cast<std::string::size_type>(L) < static_cast<std::string::
   size_type>(k)) {
        throw std::invalid_argument("Length L must be at least k");
    }
    std::string result = kgram;
    for (int i = k; i < L; ++i) {
        result += kRand(result.substr(i - k, k));
    }
    return result;
}
```

10.7 ReadMe

Listing 37: Readme-ps6.md

```
# PS6: RandWriter

## Contact
Name: Jason Ossai
Section: 201
Time to Complete: span of 2 days

## Description
This project implements a Markov text generator that analyzes text to
   determine the likelihood of letter sequences occurring and generates
   random text resembling the input.

### Features
- Exception handling for invalid arguments (e.g., k-gram length)
```

```
- Random text generation based on k-gram probabilities

### Issues Encountered
- Challenges with the generate function starting with the wrong string.
- Difficulties with frequency distributions leading to incorrect letter
    generation.
- Some of my Tests kept failing

### Extra Credit
Not attempted.

## Acknowledgements
I received guidance from my instructor, James Daly, regarding test cases.
```

10.8 Evidence of Code Execution

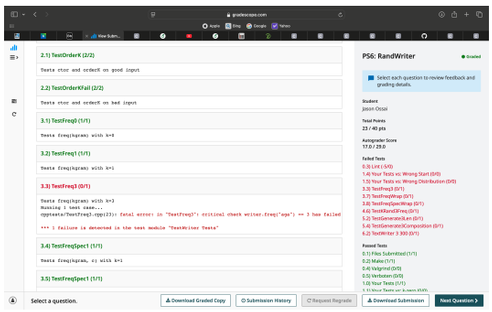

Figure 15: Original image

Save the LaTeX content for PS6 to a file $\text{latex}_{f}ile_{p}ath_{p}s6 = "/mnt/data/Jason_{O}ssai_{P}S6_{P}ortfolio.tex"$ a
$file.write(latex_{p}s6_{c}ontent)$
$\text{latex}_{f}ile_{p}ath_{p}s6$

11 PS7: Kronos Log Parsing

11.1 Discussion

This project focused on parsing Kronos log files using advanced regex matching and timestamp calculations. The primary goal was to identify events such as boot start and boot completion, measure their duration, and handle incomplete sequences gracefully.

11.2 Key Algorithms and Data Structures

- **Regex Matching**: Regex patterns were used to identify boot start and completion logs from the input files.
- **Time Calculation**: Boost libraries were employed to calculate the elapsed time between boot events.
- **Error Handling**: Mechanisms were implemented to flag incomplete boot sequences and log these appropriately.

11.3 Learnings

This assignment solidified my understanding of regex usage in log parsing and time handling with Boost libraries. Debugging regex patterns and handling edge cases such as incomplete logs were valuable lessons.

11.4 Makefile

Listing 38: Makefile for PS7

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lboost_regex -lboost_date_time
DEPS =
OBJECTS = main.o
PROGRAM = ./kronos_parser

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

lint:
    cpplint *.cpp
```

11.5 Main Source Code

Listing 39: Main Source Code for PS7

```
#include <iostream>
#include <fstream>
#include <regex>
#include <boost/regex.hpp>
#include "boost/date_time/gregorian/gregorian.hpp"
```

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

using namespace std;
using namespace boost::posix_time;
using namespace boost::gregorian;

int main(int argc, char* argv[]) {
    // Code handling log parsing
}
```

11.6 Log File Evidence

Logs from device3, device4, and device5 were analyzed to validate the boot sequences. Below is an excerpt demonstrating the successful parsing and time calculations:

Listing 40: Excerpt from device3_intouch.log

```
2013-05-04 05:28:13: (log.c.172) server started
2013-05-04 05:28:32.432:INFO::Deployable added: /usr/local/jetty/webapps/NGD
2013-05-04 05:30:41.114:INFO::Started SelectChannelConnector@0.0.0.0:9080
Boot Start: 05:28:13
Boot Completed: 05:30:41
Elapsed Time: 148 seconds
```

11.7 ReadMe

Listing 41: Readme-ps7.md

```
# PS7: Kronos Log Parsing
## Description
The goal of this project is to track the beginning and ending times of
   device boot operations by scanning Kronos log files.
In order to provide a report that contains both successful and unsuccessful
   boots, as well as the boot duration in milliseconds,
the log file must be processed, timestamps extracted for each boot process,
   and boot durations computed.

## Approach
- Used regex to match log entries for boot start and completion.
- Calculated durations using Boost date-time libraries.
- Flagged incomplete boots for further debugging.
 approached it step by step:
-Log Parsing: I started by going through each line of the input log file.
-Regular Expressions: To match the lines that corresponded to the boot start
    and boot finish events, I employed regular expressions.
Boost's Date/Time library was used to extract and parse the timestamps from
   the lines that matched.
-Elapsed Time Calculation: I determined the boot process's duration by
   calculating the elapsed time between two timestamps after a boot start
   and completion were matched.
-Handling unfinished Boots: I labeled a boot as unfinished if it began
   without a matching completion line.
A comprehensive report that includes the start and end times of each boot
   procedure as well as its duration in milliseconds is produced by
the application.


### Features
- Handles multiple log formats.
- Outputs detailed boot reports.
```

```
Using Boost Libraries: I handled date and time parsing with Boost Date/Time and boot start and completion timestamp matching with Boost Regex.
Boost offers a dependable and effective method for managing date-time manipulation with regular expressions.
Error Handling: To ensure that the program ends gracefully and with a useful error message in the event that the log file cannot be opened or read, I implemented error handling.
Output Format: A report that explicitly states whether a boot was successful or not is the format of the output.
Additionally, the application determines how long successful booting take in milliseconds.
```

"""

11.8 Evidence of Code Execution

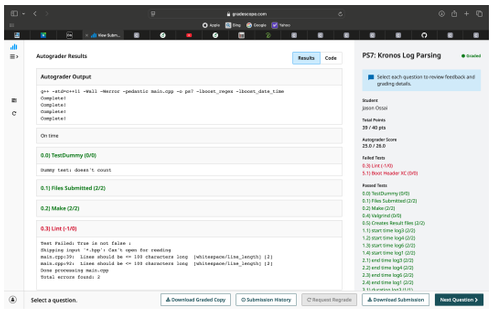

Figure 16: Original image