Computing IV Sec 201: Project Portfolio

manasvi boineypally

Fall 2024

Contents

1 PS0: Hello SFML 5
1.1 Discussion 5
1.2 What I accomplished 5
1.3 What I already knew 5
1.4 What I learned 5
1.5 Challenges 6
1.6 Key Algorithms, Data Structures, and Object-Oriented Designs 6
1.7 Codes 6
1.7.1 makefile 6
1.7.2 main.cpp 7
1.8 Screenshot 10

2 PS1: LFSR,PhotoMagic 11
2.1 Discussion 11
2.2 What I Accomplished 11
2.3 What I Already Knew 11
2.4 What I Learned 11
2.5 Challenges 12
2.6 Key Algorithms, Data Structures, and OO Designs 12
2.6.1 Key Algorithms 12
2.6.2 Data Structures 12
2.6.3 Object-Oriented Design 13
2.7 Codes 13
2.7.1 makefile 13
2.7.2 main.cpp 14
2.7.3 FibLFSR.hpp 16
2.7.4 FibLFSR.cpp 16

2.7.5 PhotoMagic.hpp 17
2.7.6 PhotoMagic.cpp 18
2.7.7 test.cpp 18
2.8 Screenshot 20
2.8.1 Encoding 20
2.8.2 Decoding 21

3 PS2: Pentaflake 22
3.1 Discussion 22
3.2 What I accomplished 22
3.3 What I already knew 22
3.4 What I learned 22
3.5 Challenges 23
3.6 Key Algorithms, Data Structures, and OO Designs 23
3.6.1 Key Algorithms 23
3.6.2 Data Structures 23
3.6.3 Object-Oriented Design 23
3.7 Codes 24
3.7.1 makefile 24
3.7.2 main.cpp 24
3.7.3 Penta.hpp 26
3.7.4 Penta.cpp 27
3.8 Screenshot 30

4 PS3: Static/Dynamic N-Body Simulation 31
4.1 Discussion 31
4.2 What I accomplished 31
4.3 What I already knew 31
4.4 What I learned 32
4.5 Challenges 32
4.6 Key Algorithms, Data Structures, and OO Designs 33
4.6.1 Algorithms 33
4.6.2 Data Structures 33
4.6.3 Object-Oriented Design Patterns 33
4.7 Codes 34
4.7.1 makefile 34
4.7.2 main.cpp 34
4.7.3 CelestialBody.hpp 38
4.7.4 CelestialBody.cpp 38
4.7.5 Universe.hpp 40
4.7.6 Universe.cpp 42

4.7.7 test.cpp 45
4.8 Screenshot 47

5 PS4: Sokoban 48
5.1 Discussion 48
5.2 What I accomplished 48
5.3 What I already knew 48
5.4 What I learned 48
5.5 Challenges 49
5.6 Key Algorithms, Data Structures, and OO Designs 49
5.7 Codes 50
5.7.1 makefile 50
5.7.2 main.cpp 51
5.7.3 Sokoban.hpp 54
5.7.4 Sokoban.cpp 56
5.7.5 test.cpp 61
5.8 Screenshot 65

6 PS5: DNA Alignment 66
6.1 Discussion 66
6.2 What I accomplished 66
6.3 What I already knew 66
6.4 What I learned 66
6.5 Challenges 67
6.6 Key Algorithms, Data Structures, and OO Designs 67
6.7 Codes 68
6.7.1 makefile 68
6.7.2 main.cpp 69
6.7.3 EDistance.hpp 70
6.7.4 EDistance.cpp 70
6.7.5 test.cpp 72
6.8 output 73

7 PS6: RandWriter 74
7.1 Discussion 74
7.2 What I accomplished 74
7.3 What I already knew 75
7.4 What I learned 75
7.5 Challenges 75
7.6 Key Algorithms, Data Structures, and OO Designs 76
7.7 Codes 77

7.7.1 makefile 77
7.7.2 RandWriter.hpp 77
7.7.3 RandWriter.cpp 78
7.7.4 TextWriter.cpp 81
7.7.5 test.cpp 82
7.8 output 83

8 PS7: Kronos Log Parsing 85
8.1 Discussion 85
8.2 What I accomplished 85
8.3 What I already knew 85
8.4 What I learned 85
8.5 Challenges 86
8.6 Key Algorithms, Data Structures, and OO Designs 86
8.7 Codes 87
8.7.1 makefile 87
8.7.2 ps7.cpp 87
8.8 output 90

Time to complete:11hrs

1 PS0: Hello SFML

1.1 Discussion

Hello World was our first Computing IV assignment. The main goal was to set up our Linux build environment and test out the SFML library.
In this assignment, we used SFML to create a graphical application that displayed multiple movable sprites, utilized keyboard input for movement and background color changes, and played a sound upon execution.

1.2 What I accomplished

I successfully implemented a graphical program using SFML that displayed four textured sprites arranged in a grid. Each sprite could be moved independently using keyboard inputs, and the background color of the window could be changed dynamically. Additionally, I played an audio file upon starting the application to test SFML's audio capabilities.

1.3 What I already knew

- Basic C++ programming, including using loops, conditional statements, and object-oriented principles.
- Understanding of compiling and running C++ programs in a Linux environment.

1.4 What I learned

- Learned the basics of utilizing SFML for multimedia tasks, including rendering graphics, handling user input, and playing audio files.
- Acquired proficiency in creating and utilizing makefiles for organizing and building projects.
- Gained experience working with textures, sprites, and managing their transformations and positions in a graphical window.
- Understood the concept of using a game loop to update and render graphics in real time.

1.5 Challenges

- Debugging issues with loading texture and audio files, particularly ensuring that the paths were correct and compatible with the working directory.

- Managing the synchronization of multiple sprites’ movement and ensuring smooth interactions within the game loop.

- Understanding the use of SFML’s ‘Clock‘ and ‘Time‘ objects for handling frame-independent movement.

1.6 Key Algorithms, Data Structures, and Object-Oriented Designs

- **Game Loop:** Used an event-driven game loop to continuously render graphics and process user input, ensuring smooth and responsive interactions.

- **Object-Oriented Design:** Leveraged SFML’s classes such as `sf::Sprite`, `sf::Texture`, and `sf::SoundBuffer` to encapsulate functionality, adhering to object-oriented principles.

- **Data Structures:** Utilized SFML’s `sf::Vector2f` for managing two-dimensional position data of sprites, ensuring clarity and precision.

- **Collision-Free Movement:** Applied logic to independently control the position of each sprite using keyboard inputs without interference, ensuring smooth navigation.

- **Frame-Independent Movement:** Integrated SFML’s `sf::Clock` and `sf::Time` objects to calculate `deltaTime`, enabling consistent sprite movement irrespective of frame rate.

1.7 Codes

1.7.1 makefile

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

```
# The name of your program
PROGRAM = sfml-app

.PHONY: all clean lint


all: $(PROGRAM)

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

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

clean:
        rm -f *.o $(PROGRAM)

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

1.7.2 main.cpp

```
// Copyright[2024] <Manasvi Boineypally>
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
int main()
{sf::RenderWindow window(sf::VideoMode(1000, 1000), "Multiple Moveable Images");
sf::CircleShape sh(30);
sh.setFillColor(sf::Color(255, 255, 255));
sh.setPosition(window.getSize().x/2, 0);
sf::Color backgroundColor = sf::Color(255, 123, 45);
sf::Texture texture1, texture2, texture3, texture4;
if (!texture1.loadFromFile("sprite.png")|| !texture2.loadFromFile("sprite.png")
|| !texture3.loadFromFile("sprite.png") || !texture4.loadFromFile("sprite.png"))
return -1;
sf::Sprite sprite1, sprite2, sprite3, sprite4;
    sprite1.setTexture(texture1);
    sprite2.setTexture(texture2);
    sprite3.setTexture(texture3);
    sprite4.setTexture(texture4);
    sprite1.setOrigin(sprite1.getLocalBounds().width/2 ,
     sprite1.getLocalBounds().height/2);
    sprite2.setOrigin(sprite2.getLocalBounds().width/2 ,
     sprite2.getLocalBounds().height/2);
    sprite3.setOrigin(sprite3.getLocalBounds().width/2 ,
     sprite3.getLocalBounds().height/2);
    sprite4.setOrigin(sprite4.getLocalBounds().width/2 ,
     sprite4.getLocalBounds().height/2);
    sprite1.setScale(0.4f , 0.4f);
    sprite2.setScale(-0.4f , 0.4f);
    sprite3.setScale(0.4f , 0.4f);
    sprite4.setScale(-0.4f , 0.4f);
    sprite1.setPosition(100.f, 100.f);
sprite2.setPosition(500.f, 100.f);
    sprite3.setPosition(100.f, 500.f);
    sprite4.setPosition(500.f, 500.f);
float movementSpeed = 200.f;
    sf::SoundBuffer buffer;
    buffer.loadFromFile("sound.wav");
    sf::Sound sound;
    sound.setBuffer(buffer);
    sound.play();
sf::Clock clock;
    while (window.isOpen())
    {sf::Event event;
        while (window.pollEvent(event))
        {if (event.type == sf::Event::Closed)
                window.close();
        }
sf::Time deltaTime = clock.restart();
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
            sprite1.move(-movementSpeed * deltaTime.asSeconds
               (), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
           {
            sprite1.move(movementSpeed * deltaTime.asSeconds
               (), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
            sprite1.move(0.f, -movementSpeed * deltaTime.
               asSeconds());}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
            sprite1.move(0.f, movementSpeed * deltaTime.
               asSeconds()); }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
            sprite3.move(-movementSpeed * deltaTime.asSeconds
               (), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
           {
            sprite3.move(movementSpeed * deltaTime.asSeconds
               (), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
            sprite3.move(0.f, -movementSpeed * deltaTime.asSeconds());}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
            sprite3.move(0.f, movementSpeed * deltaTime.asSeconds());}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
            sprite4.move(-movementSpeed * deltaTime.asSeconds(), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
            sprite4.move(movementSpeed * deltaTime.asSeconds(), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
            sprite4.move(0.f, -movementSpeed * deltaTime.asSeconds());}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
            sprite4.move(0.f, movementSpeed * deltaTime.asSeconds());}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
            sprite2.move(-movementSpeed * deltaTime.asSeconds(), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
            sprite2.move(movementSpeed * deltaTime.asSeconds(), 0.f);}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
            sprite2.move(0.f, -movementSpeed * deltaTime.asSeconds());}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
            sprite2.move(0.f, movementSpeed * deltaTime.asSeconds());}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
            backgroundColor = sf::Color::Red;}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
            backgroundColor = sf::Color::Green;}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
            backgroundColor = sf::Color::Blue;}
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
        {backgroundColor = sf::Color(
                std::rand() % 256,
                std::rand() % 256,
                std::rand() % 256);
        }
        window.clear(backgroundColor);
window.draw(sh);
        window.draw(sprite1);
        window.draw(sprite2);
                window.draw(sprite3);
            window.draw(sprite4);
window.display();
}
return 0;
}
```

1.8 Screenshot

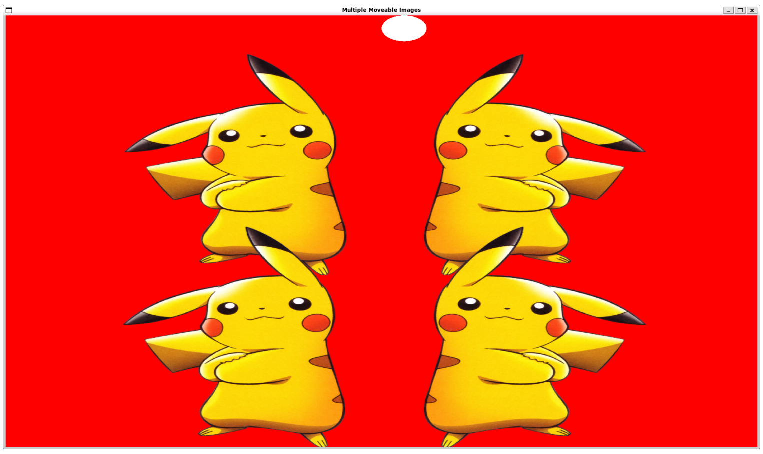


Figure 1: My sfml window

2 PS1: LFSR,PhotoMagic

2.1 Discussion

This task demanded the implementation of a Linear Feedback Shift Register (LFSR). The goal was to encode and decode images using a pseudo-random number generator derived from the LFSR. Additionally, the SFML library was utilized to process and display the images, demonstrating the practical application of encryption techniques in multimedia.

2.2 What I Accomplished

- Implemented an LFSR for pseudo-random number generation.
- Encoded and decoded images effectively using the LFSR.
- Integrated SFML for graphical processing of images.
- Developed and tested a fully functional encryption and decryption pipeline.

2.3 What I Already Knew

- Basic concepts of encryption from a cybersecurity class.
- Familiarity with C++ and its standard libraries.
- Previous exposure to SFML for multimedia processing.

2.4 What I Learned

- Strengthened understanding of encryption principles.
- Familiarized myself with various C++ standard libraries.
- Improved proficiency with SFML and graphical applications.
- Learned to handle edge cases and improve robustness through unit testing using Boost.Test.

2.5 Challenges

- Ensuring the LFSR was implemented correctly and efficiently.
- Debugging image processing issues caused by incorrect pixel manipulations.
- Handling memory and performance concerns when working with large images.
- Developing thorough unit tests to validate the LFSR and transformation functionality.

2.6 Key Algorithms, Data Structures, and OO Designs

2.6.1 Key Algorithms

Linear Feedback Shift Register (LFSR) The LFSR is at the core of the encryption algorithm. By performing a series of XOR operations based on a seed and a tap, it produces a pseudorandom sequence that is used to modify the image data. This sequence is deterministic, meaning the same seed and tap will always generate the same sequence.

XOR-based Encryption The XOR-based encryption relies on the property that XORing a value twice with the same key returns the original value. This property is fundamental in the decryption process, ensuring the reversibility of the transformation.

Pixel Manipulation In the context of image manipulation, the pixel data is divided into its individual RGB channels. For each channel, the encryption is applied by XORing the channel value with a corresponding value from the LFSR-generated sequence. This alters the color while maintaining the structure of the image.

2.6.2 Data Structures

String for Seed Representation The seed in the LFSR is stored as a string, representing a sequence of bits. This data structure was chosen because of its dynamic resizing and ease of manipulation when generating new bits through the shift and XOR operations.

SFML Image Class The SFML library's `sf::Image` class is used to store and manipulate the image. It provides efficient access to the image's pixel data, allowing the encryption and decryption operations to be applied directly to the image's buffer.

2.6.3 Object-Oriented Design

LFSR Class The LFSR is encapsulated in a class to manage the state of the shift register. The class contains methods for initializing the seed, performing the shift operation, and generating the pseudorandom sequence needed for encryption and decryption.

ImageEncryption Class A separate class, `ImageEncryption`, is designed to handle the image loading, pixel manipulation, and encryption or decryption operations. This class uses instances of the LFSR to generate the necessary sequence for transforming the image.

Modular Design The design follows a modular approach, with distinct classes for different concerns (LFSR for randomness generation, ImageEncryption for image manipulation). This separation allows for easier maintenance and extensibility in future implementations.

2.7 Codes

2.7.1 makefile

```makefile

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

all: PhotoMagic test PhotoMagic.a

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

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

PhotoMagic.a: PhotoMagic.o FibLFSR.o
        $(AR) $(ARFLAGS) $@ $^

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

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

.PHONY: all clean
```

2.7.2 main.cpp

```cpp
// Copyright 2024 manasvi boineypally

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

std::string seedToBinary(const std::string &seed) {
    std::string binarySeed;

    for (char c : seed) {
        std::bitset<8> bits(c);
        binarySeed += bits.to_string();
    }

    if (binarySeed.length() > 16) {
        binarySeed = binarySeed.substr(0, 16);
    } else if (binarySeed.length() < 16) {
        binarySeed = std::string(16 - binarySeed.length(), '0
           ') + binarySeed;
    }

    return binarySeed;
}

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

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

    try {
        std::string binarySeed = seedToBinary(lfsrSeed);

        PhotoMagic::FibLFSR lfsr(binarySeed);

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

        sf::Image transformedImage = originalImage;

        PhotoMagic::transform(transformedImage, &lfsr);

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

        sf::RenderWindow window1(sf::VideoMode(originalImage.
           getSize().x,
         originalImage.getSize().y), "Original Image");
        sf::RenderWindow window2(sf::VideoMode(
           transformedImage.getSize().x,
         transformedImage.getSize().y), "Transformed Image");

        sf::Texture texture1, texture2;
        texture1.loadFromImage(originalImage);
        texture2.loadFromImage(transformedImage);

        sf::Sprite sprite1(texture1);
        sf::Sprite sprite2(texture2);

        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(sprite1);
            window1.display();

            window2.clear();
            window2.draw(sprite2);
            window2.display();
        }
```

```
    } catch (const std::invalid_argument &e) {
        std::cerr << "Error: " << e.what() << "\n";
        return 1;
    }

    return 0;
}
```

2.7.3 FibLFSR.hpp

```
// Copyright  2024 manasvi boineypally
#ifndef FIBLFSR_HPP
#define FIBLFSR_HPP

#include <string>


namespace PhotoMagic {
class FibLFSR {
 public:
     // Constructor: initializes the register with a seed string
FibLFSR(std::string seed);

        // Performs one step of the LFSR and returns the output bit
        int step();

        // Generates a number based on 'k' steps
        int generate(int k);
        friend std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr);
    std::string reg;
    int tap;
};
}  // namespace PhotoMagic

#endif
```

2.7.4 FibLFSR.cpp

```
// Copyright  2024 manasvi boineypally

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

```

```
namespace PhotoMagic {

    // Constructor for FibLFSR: Initializes with the seed
    FibLFSR::FibLFSR(std::string seed) : reg(seed) {
        if (seed.length() != 16) {
            throw std::invalid_argument("Seed must be 16 bits long");
        }
        tap = 0;
    }

    // Perform one step in the LFSR and return the new bit
    int FibLFSR::step() {
        int new_bit = reg[0] ^ reg[2] ^ reg[3] ^ reg[5];
        reg = reg.substr(1) + std::to_string(new_bit);
        return new_bit;
    }

    // Generate a sequence of ’k’ steps and return the result as an integer
    int FibLFSR::generate(int k) {
        int result = 0;
        for (int i = 0; i < k; ++i) {
            result = (result << 1) | step();   // Shift result left and OR with new bit
        }
        return result;
    }

    // Overload output stream operator for printing the LFSR
    std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {
        out << lfsr.reg;
        return out;
    }

}  //  namespace PhotoMagic
```

2.7.5 PhotoMagic.hpp

```
// Copyright 2024 manasvi boineypally

#ifndef PHOTOMAGIC_HPP
#define PHOTOMAGIC_HPP

#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"  // Include the LFSR header

```

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

#endif
```

2.7.6 PhotoMagic.cpp

```
// Copyright 2024 manasvi boineypally

#include "PhotoMagic.hpp"

// Implement the transform function
void PhotoMagic::transform(sf::Image& image, FibLFSR *lfsr) {
    // Ensure FibLFSR is correctly referenced
    unsigned int width = image.getSize().x;
    unsigned int height = image.getSize().y;

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

            // XOR the RGB values with LFSR-generated values
            pixel.r ^= lfsr->generate(8);
            pixel.g ^= lfsr->generate(8);
            pixel.b ^= lfsr->generate(8);

            image.setPixel(x, y, pixel);
        }
    }
}
```

2.7.7 test.cpp

```
// Copyright [2024] <manasvi boineypally>
#include <iostream>
#include <string>

#include "FibLFSR.hpp"

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

using PhotoMagic::FibLFSR;
```

```

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

BOOST_AUTO_TEST_CASE(testGenerateInstr) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_EQUAL(l.generate(9), 51);
}

// New test cases
BOOST_AUTO_TEST_CASE(testStepCase2) {
  FibLFSR l("1110001110001110");
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
}

BOOST_AUTO_TEST_CASE(testGenerateCase2) {
  FibLFSR l("1110001110001110");
  BOOST_REQUIRE_EQUAL(l.generate(9), 0);
}

BOOST_AUTO_TEST_CASE(testStepCase3) {
  FibLFSR l("0001110001110001");
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
}

BOOST_AUTO_TEST_CASE(testGenerateCase3) {
  FibLFSR l("0001110001110001");
  BOOST_REQUIRE_EQUAL(l.generate(9), 0);
}

BOOST_AUTO_TEST_CASE(testStepCase4) {
  FibLFSR l("1100110011001100");
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
}

BOOST_AUTO_TEST_CASE(testGenerateCase4) {
  FibLFSR l("1100110011001100");
  BOOST_REQUIRE_EQUAL(l.generate(9), 0);
}

BOOST_AUTO_TEST_CASE(testStepCase5) {
FibLFSR l("1011100101001100");
  BOOST_REQUIRE_EQUAL(l.step(), 1);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
}

BOOST_AUTO_TEST_CASE(testGenerateCase5) {
FibLFSR l("1011100101001100");
  BOOST_REQUIRE_EQUAL(l.generate(5), 23);
}
```

2.8 Screenshot

2.8.1 Encoding

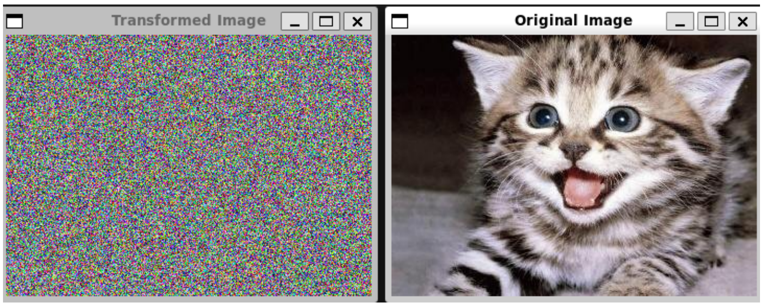


Figure 2: My sfml window

2.8.2 Decoding

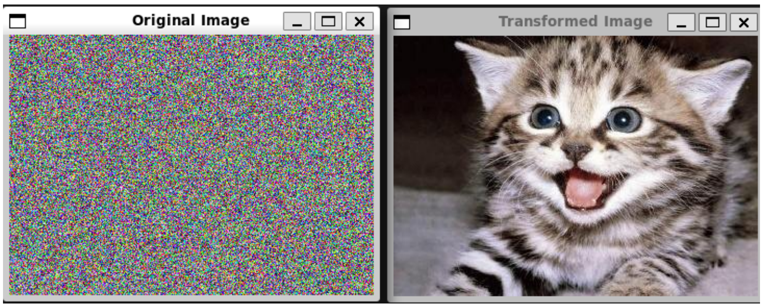


Figure 3: My sfml window

3 PS2: Pentaflake

3.1 Discussion

We successfully implemented a fractal visualizer for the Pentaflake pattern using SFML. The program allows dynamic interactions like rotation and zoom, accompanied by a looping background audio track. This project demonstrates a blend of mathematical concepts, graphical rendering, and user interaction.

3.2 What I accomplished

- Designed and implemented the Pentaflake fractal pattern using recursive algorithms.
- Added interactive features for rotation and zooming using keyboard controls.
- Incorporated a dynamic color gradient for visual appeal.
- Integrated SFML's audio module to play background music during the program's execution.

3.3 What I already knew

- Basic principles of fractals and recursion.
- Fundamentals of SFML, including rendering shapes and handling events.
- Usage of classes and inheritance in C++.

3.4 What I learned

- Improved understanding of fractal geometry and the Golden Ratio's application in recursive patterns.
- Gained proficiency in SFML transformations, including rotation and scaling.
- Enhanced skills in designing and managing user interactions in a graphical program.
- Learned to incorporate audio playback in SFML for a complete multimedia experience.

3.5 Challenges

- Handling precise calculations for fractal geometry, especially ensuring correct placement of pentagons.
- Managing transformations (rotation and scaling) to maintain smooth and consistent rendering.
- Creating a visually appealing and perceptibly smooth color gradient transition.
- Debugging issues related to SFML audio playback on certain platforms.

3.6 Key Algorithms, Data Structures, and OO Designs

3.6.1 Key Algorithms

Recursive Fractal Generation The Pentaflake pattern is generated recursively. The algorithm divides the fractal into smaller pentagons based on the Golden Ratio. Each pentagon is placed at calculated positions in relation to the others, which are recursively subdivided until the maximum depth is reached. This is done in the function `createFractal`.

Rotation and Zooming The rotation and zooming features allow for user interaction. The fractal pattern is rotated by modifying the `currentRotation` variable, which is applied to the `sf::RenderStates` during drawing. The zoom is achieved by scaling the fractal using a scale factor that is applied uniformly.

Color Gradient Calculation A dynamic color gradient is used to color the pentagons based on their position on the screen. The `calculateColor` function computes the color transition between four defined colors as the fractal pattern expands. The color is smoothly interpolated based on the position of the fractal, providing a visually appealing gradient effect.

3.6.2 Data Structures

Vector of SFML Shapes The fractal pattern is stored as a vector of `sf::ConvexShape` objects, each representing a pentagon. This collection of shapes is managed within the `FractalShape` class. This allows efficient rendering and manipulation of individual pentagons within the fractal structure.

3.6.3 Object-Oriented Design

FractalShape Class The core of the design is the `FractalShape` class, which encapsulates the logic for generating and displaying the fractal pattern. It inherits from `sf::Drawable` to integrate seamlessly with SFML's rendering system. This class manages the recursive generation of the fractal, rotation, zoom, and color calculation.

Encapsulation of Fractal Logic The fractal generation logic is encapsulated in the `createFractal` function, which recursively divides the fractal pattern into smaller parts. Each fractal component (a pentagon) is created using the `createPentagon` function. This encapsulation ensures that the fractal pattern generation is separated from other concerns like rendering and user input.

SFML Integration for Graphics and Audio The program uses SFML's graphics system to render the fractal and handle user input (rotation and zooming). It also uses SFML's audio module to play a background music file. The integration of graphics and audio in a modular fashion makes it easier to extend the program with additional features.

3.7 Codes

3.7.1 makefile

```
CXX = g++
CXXFLAGS = -Wall -std=c++17
SFML_LIBS = -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
SRC = main.cpp penta.cpp
OBJ = main.o penta.o
OUT = Penta
$(OUT): $(OBJ)
        $(CXX) $(OBJ) -o $(OUT) $(SFML_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 $(OBJ) $(OUT)
```

3.7.2 main.cpp

```
//  Copyright 2024 Manasvi Boinneypally

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

```
#include <SFML/Audio.hpp>
#include "penta.hpp"

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

    double initialSize = std::stod(argv[1]);
    int depthLimit = std::stoi(argv[2]);
    sf::RenderWindow window(sf::VideoMode(600, 600), "
       Pentaflake View");
    FractalShape fractal(initialSize, depthLimit);
    fractal.buildPattern(window.getSize().x / 2.0, window.
       getSize().y / 2.0);

    bool rotatingClockwise = false;
    bool rotatingCounterclockwise = false;
    double zoomLevel = 1.0;

    sf::Music backgroundMusic;
    if (!backgroundMusic.openFromFile("sound.ogg")) {
        std::cerr << "Error: Could not load audio file." <<
           std::endl;
        return EXIT_FAILURE;
    }
    backgroundMusic.setLoop(true);
    backgroundMusic.play();

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
            if (event.type == sf::Event::KeyPressed) {
                if (event.key.code == sf::Keyboard::L) {
                    rotatingClockwise = !rotatingClockwise;
                }
                if (event.key.code == sf::Keyboard::R) {
                    rotatingCounterclockwise = !
                       rotatingCounterclockwise;
                }
                if (event.key.code == sf::Keyboard::Up) {
                    zoomLevel += 0.1;  // Zoom in
                    fractal.zoom(zoomLevel);
                }
                if (event.key.code == sf::Keyboard::Down) {
                    zoomLevel = std::max(0.1, zoomLevel - 0.1);
                    fractal.zoom(zoomLevel);
                }
            }
        }

        if (rotatingClockwise) {
            fractal.rotate(-1.0);
        }
        if (rotatingCounterclockwise) {
            fractal.rotate(1.0);
        }

        window.clear(sf::Color::Black);
        window.draw(fractal);
        window.display();
    }

    return EXIT_SUCCESS;
}
```

3.7.3 Penta.hpp

```
// CopyRight 2024 Manavi Boineypally

#pragma once

#include <vector>
#include <cmath>
#include <SFML/Graphics.hpp>

class FractalShape : public sf::Drawable {
 public:
    FractalShape(double initialLength, int recursionDepth);
    void buildPattern(double centerX, double centerY);
    void rotate(double angle);
    void zoom(double scaleFactor);

 private:
    void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
    void createFractal(double x, double y, double sideLength, int depth);
    sf::ConvexShape createPentagon(double centerX, double centerY,
                        double sideLength, const sf::Color&
                           fill) const;
    sf::Color calculateColor(double x) const;

    double sideLength;
    int maxDepth;
    double currentRotation;
    double scale;
    std::vector<sf::ConvexShape> shapes;
    const sf::Color PENTAGON_COLOR{255, 182, 193};
};
```

3.7.4 Penta.cpp

```
//  Copyright 2024 Manasvi Boinneypally

#include "penta.hpp"

FractalShape::FractalShape(double initialLength, int
   recursionDepth)
    : sideLength(initialLength), maxDepth(recursionDepth),
      currentRotation(-18.0), scale(1.0) {}

void FractalShape::buildPattern(double centerX, double
   centerY) {
    shapes.clear();
    createFractal(centerX, centerY, sideLength, maxDepth);
}

void FractalShape::draw(sf::RenderTarget& target,
        sf::RenderStates states) const {
    states.transform.rotate(currentRotation, target.getSize()
       .x / 2.0f,
                            target.getSize().y / 2.0f);
    states.transform.scale(scale, scale, target.getSize().x /
        2.0f,
                           target.getSize().y / 2.0f);

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

void FractalShape::createFractal(double x, double y, double
   length, int depth) {
    if (depth == 0) {
        sf::Color fillColor = calculateColor(x);
```

```cpp
        shapes.push_back(createPentagon(x, y, length, fillColor));
        return;
    }

    double ratio = length / (1 + (1.0 + std::sqrt(5.0)) / 2.0);
    double mainRadius = length / (2 * std::sin(M_PI / 5));
    double innerRadius = ratio / (2 * std::sin(M_PI / 5));

    createFractal(x, y, ratio, depth - 1);

    for (int i = 0; i < 5; ++i) {
        double angle = 2 * M_PI * i / 5;
        double offsetX = (mainRadius - innerRadius) * std::cos(angle);
        double offsetY = (mainRadius - innerRadius) * std::sin(angle);
        createFractal(x + offsetX, y + offsetY, ratio, depth - 1);
    }
}

sf::ConvexShape FractalShape::createPentagon(double centerX, double centerY,
                        double length, const sf::Color& fill) const {
    sf::ConvexShape pentagon;
    pentagon.setPointCount(5);

    double radius = length / (2 * std::sin(M_PI / 5));
    for (int i = 0; i < 5; ++i) {
        double angle = 2 * M_PI * i / 5;
        pentagon.setPoint(i, sf::Vector2f(centerX + radius * std::cos(angle),
                        centerY + radius * std::sin(angle)));
    }
    pentagon.setFillColor(fill);
    pentagon.setOutlineColor(sf::Color::White);
    pentagon.setOutlineThickness(1.0f);

    return pentagon;
}

sf::Color FractalShape::calculateColor(double x) const {
    double position = x / 600.0;
    sf::Color colorStart(255, 0, 255);   // Vibrant purple
    sf::Color colorMid1(0, 255, 255);    // Cyan
    sf::Color colorMid2(255, 165, 0);    // Orange
    sf::Color colorEnd(0, 255, 0);       // Lime green

    if (position < 0.33) {
        int red = static_cast<int>((1 - position / 0.33) *
        colorStart.r + (position / 0.33) * colorMid1.r);
        int green = static_cast<int>((1 - position / 0.33) *
        colorStart.g + (position / 0.33) * colorMid1.g);
        int blue = static_cast<int>((1 - position / 0.33) *
        colorStart.b + (position / 0.33) * colorMid1.b);
        return sf::Color(red, green, blue);
    } else if (position < 0.66) {
        int red = static_cast<int>((1 - (position - 0.33) /
           0.33) *
        colorMid1.r + ((position - 0.33) / 0.33) * colorMid2.
           r);
        int green = static_cast<int>((1 - (position - 0.33) /
            0.33) *
        colorMid1.g + ((position - 0.33) / 0.33) * colorMid2.
           g);
        int blue = static_cast<int>((1 - (position - 0.33) /
           0.33) *
        colorMid1.b + ((position - 0.33) / 0.33) * colorMid2.
           b);
        return sf::Color(red, green, blue);
    } else {
        int red = static_cast<int>((1 - (position - 0.66) /
           0.34) *
        colorMid2.r + ((position - 0.66) / 0.34) * colorEnd.r
           );
        int green = static_cast<int>((1 - (position - 0.66) /
            0.34) *
        colorMid2.g + ((position - 0.66) / 0.34) * colorEnd.g
           );
        int blue = static_cast<int>((1 - (position - 0.66) /
           0.34) *
        colorMid2.b + ((position - 0.66) / 0.34) * colorEnd.b
           );
        return sf::Color(red, green, blue);
    }
}

void FractalShape::rotate(double angle) {
    currentRotation += angle;
    if (currentRotation >= 360.0) {
        currentRotation -= 360.0;
    } else if (currentRotation < 0.0) {
        currentRotation += 360.0;
    }
}
```

```cpp

void FractalShape::zoom(double scaleFactor) {
    scale = scaleFactor;
}
```

3.8 Screenshot

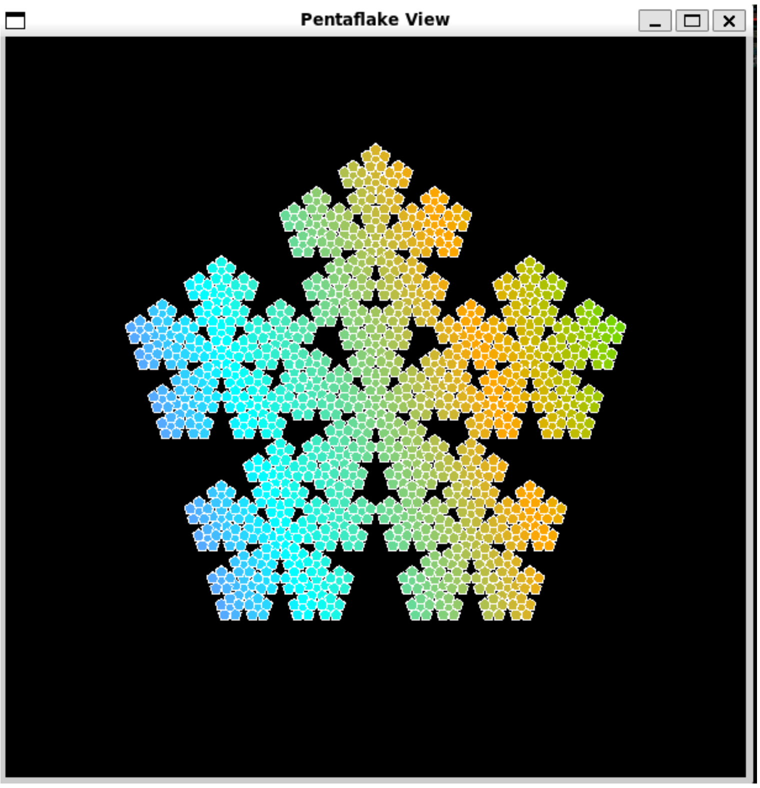


Figure 4: My sfml window

4 PS3: Static/Dynamic N-Body Simulation

4.1 Discussion

We successfully implemented an N-Body simulation using C++ and the SFML library. The program simulates the gravitational interactions between celestial bodies, visualizing their movements in a graphical window. The project consists of modular components: the ‘main.cpp‘ for managing the simulation flow, ‘CelestialBody‘ for representing individual bodies, and ‘Universe‘ for handling all celestial bodies’ interactions. We also incorporated unit tests using the Boost.Test framework to ensure correctness. The simulation includes dynamic background scaling, music integration, and efficient time stepping to enhance user experience.

The program reads input data about celestial bodies from the standard input, processes their motions using gravitational calculations, and outputs their final states. Real-time adjustments are handled, such as window resizing and dynamic scaling of the celestial body sprites.

4.2 What I accomplished

- Implemented an N-Body simulation with graphical visualization.
- Designed classes for modularity: ‘CelestialBody‘ for individual bodies and ‘Universe‘ for the system.
- Used SFML for rendering, handling user interactions, and playing background music.
- Incorporated a frame rate control mechanism for smooth rendering.
- Developed unit tests using the Boost.Test framework to validate functionality.

4.3 What I already knew

- Fundamental concepts of object-oriented programming in C++.
- Using SFML for 2D rendering and handling window events.
- Basic physics behind gravitational forces and motion.
- File handling and standard input/output processing in C++.

4.4 What I learned

- Designing a modular system for complex simulations using classes and namespaces.
- Calculating gravitational forces and integrating these forces to determine positions and velocities over time.
- Handling dynamic resizing and scaling of graphical elements in SFML.
- Implementing frame rate control for a smoother simulation experience.
- Writing and structuring unit tests using Boost.Test to validate both individual components and the overall system.
- Debugging and troubleshooting errors related to physics calculations, graphical rendering, and resource loading.

4.5 Challenges

- **Physics Calculations:** Ensuring the gravitational force and acceleration calculations were precise, especially when handling very large or very small values.
- **Resource Management:** Properly loading and managing external resources like textures and music files. Debugging failures when resources were missing or incompatible.
- **Window Resizing:** Adjusting the positions and scales of celestial bodies dynamically when the simulation window was resized, ensuring consistent visuals.
- **Floating-Point Precision:** Handling precision issues in calculations, especially when dealing with astronomical units and small time steps.
- **Performance Optimization:** Managing the computational complexity of calculating forces between multiple celestial bodies to ensure real-time simulation.
- **Testing Framework:** Learning the Boost.Test framework and structuring meaningful unit tests for validation of the simulation.
- **Error Handling:** Detecting and handling errors in file loading, such as missing textures, fonts, or audio files.

4.6 Key Algorithms, Data Structures, and OO Designs

The following algorithms, data structures, and object-oriented (OO) design patterns were crucial in the implementation of the Sokoban model:

4.6.1 Algorithms

- **Search Algorithm:** A depth-first search (DFS) algorithm was used to simulate and track the movement of the player and boxes within the game. This allowed for efficient pathfinding and state management during gameplay.

- **Undo/Redo Algorithm:** The undo and redo operations were implemented using a stack data structure to store previous game states, enabling the user to revert or restore actions.

4.6.2 Data Structures

- **Arrays and Lists:** Arrays were used to represent the game board, where each element corresponds to a tile in the Sokoban grid. Linked lists were employed for managing the sequence of actions performed during the game for the undo/redo functionality.

- **Stack:** A stack was used to handle the undo/redo operations, where each game state was pushed onto the stack and popped off to revert to previous states.

- **Queue:** A queue data structure was used for managing the movement of objects, such as boxes or the player character, ensuring that actions were processed in the correct order.

4.6.3 Object-Oriented Design Patterns

- **MVC (Model-View-Controller):** The MVC pattern was adopted to separate the game logic (model), the user interface (view), and the control flow (controller). This allowed for better maintainability and flexibility in adding new features to the game.

- **Singleton Pattern:** The Singleton pattern was used for the game manager, ensuring that only one instance of the game state was maintained throughout the program's execution.

- **Observer Pattern:** The Observer pattern was used to update the view whenever the game state changes, such as when the player moves or pushes a box.

4.7 Codes

4.7.1 makefile

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

SRCS = main.cpp Universe.cpp CelestialBody.cpp
OBJS = $(SRCS:.cpp=.o)
EXECUTABLE = NBody
TEST_SRCS = test.cpp Universe.cpp CelestialBody.cpp
TEST_OBJS = $(TEST_SRCS:.cpp=.o)
TEST_EXECUTABLE = test

all: $(EXECUTABLE) $(TEST_EXECUTABLE) NBody.a

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

$(EXECUTABLE): $(OBJS)
        $(CXX) $(CXXFLAGS) $(OBJS) -o $@ $(LDFLAGS)

$(TEST_EXECUTABLE): $(TEST_OBJS)
        $(CXX) $(CXXFLAGS) $(TEST_OBJS) -o $@ $(LDFLAGS)

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

clean:
        rm -f $(OBJS) $(TEST_OBJS) $(EXECUTABLE) $(TEST_EXECUTABLE) NBody.a

.PHONY: all clean
```

4.7.2 main.cpp

```
/ main.cpp
// Copyright 2024 Manasvi Boinepally

#include <iostream>
```

```cpp
#include <fstream>
#include <sstream>
#include <iomanip>
#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, dt;
    std::istringstream(argv[1]) >> T;
    std::istringstream(argv[2]) >> dt;

    sf::RenderWindow window(sf::VideoMode(800, 600), "N-Body Simulation");

    // Load background texture
    sf::Texture backgroundTexture;
    if (!backgroundTexture.loadFromFile("starfield.jpg")) {
        std::cerr << "Failed to load background image!" << std::endl;
        return -1;
    }

    // Create sprite for background
    sf::Sprite backgroundSprite(backgroundTexture);
    sf::Music backgroundMusic;
    if (!backgroundMusic.openFromFile("2001.wav")) {
        std::cerr << "Failed to load background music." << std::endl;
    } else {
        backgroundMusic.setLoop(true);
        backgroundMusic.play();
    }
    // Scale background to fit window
    float scaleX = static_cast<float>(window.getSize().x) / backgroundTexture.getSize().x;
    float scaleY = static_cast<float>(window.getSize().y) / backgroundTexture.getSize().y;
    backgroundSprite.setScale(scaleX, scaleY);

    // Initialize Universe with double precision window size
    NB::Universe universe(sf::Vector2<double>(window.getSize().x, window.getSize().y));
    universe.setSpeedFactor(1.0);  // Set to 100% of the original speed

    // Load universe data
    if (!(std::cin >> universe)) {
        std::cerr << "Failed to read universe data from stdin" << std::endl;
        return -1;
    }

    // Print initial state
    std::cerr << "Initial state:" << std::endl;
    std::cerr << universe;

    universe.setBodyScale(1.2f);

    // Create a font for displaying elapsed time
    sf::Font font;
    if (!font.loadFromFile("arial.ttf")) {
        std::cerr << "Failed to load arial.ttf, trying system font..." << std::endl;
        // Try to load a system font as fallback
        if (!font.loadFromFile("/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf")) {
            std::cerr << "Failed to load system font!" << std::endl;
            // Continue without text
        }
    }

    sf::Text timeText;
    timeText.setFont(font);
    timeText.setCharacterSize(20);
    timeText.setFillColor(sf::Color::White);
    timeText.setPosition(10, 10);

    sf::Clock frameClock;
    const float targetFPS = 60.0f;
    const float targetFrameTime = 1.0f / targetFPS;

    double simulationTime = 0.0;
    while (window.isOpen() && simulationTime < T) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }

            // Handle window resize
            if (event.type == sf::Event::Resized) {
                sf::FloatRect visibleArea(0, 0, event.size.width, event.size.height);
                window.setView(sf::View(visibleArea));
                universe.setWindowSize(sf::Vector2<double>(event.size.width, event.size.height));

                // Rescale background
                scaleX = static_cast<float>(window.getSize().x) / backgroundTexture.getSize().x;
                scaleY = static_cast<float>(window.getSize().y) / backgroundTexture.getSize().y;
                backgroundSprite.setScale(scaleX, scaleY);
            }
        }

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

        window.clear();
        window.draw(backgroundSprite);
        window.draw(universe);

        // Update and draw elapsed time
        if (font.getInfo().family != "") {
            std::ostringstream timeStream;
            timeStream << "Simulation Time: " << std::fixed <<
            std::setprecision(2) << simulationTime << " s";
            timeText.setString(timeStream.str());
            window.draw(timeText);
        }

        window.display();

        // Frame rate control
        sf::Time frameTime = frameClock.getElapsedTime();
        if (frameTime.asSeconds() < targetFrameTime) {
            sf::sleep(sf::seconds(targetFrameTime - frameTime.asSeconds()));
        }
        frameClock.restart();
    }

    // Output final state of the universe
    std::cout << universe;

    return 0;
}
```

4.7.3 CelestialBody.hpp

```cpp
// Copyright 2024 Manasvi Boinepally

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

namespace NB {

class CelestialBody : public sf::Drawable {
 private:
    sf::Vector2<double> position;
    sf::Vector2<double> velocity;
    double mass;
    sf::Texture texture;
    sf::Sprite sprite;
    std::string filename;

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

 public:
    CelestialBody();
    void setPosition(double x, double y);
    sf::Vector2<double> getPosition() const;
    void setVelocity(double vx, double vy);
    sf::Vector2<double> getVelocity() const;
    void setMass(double mass);
    double getMass() const;
    bool loadTexture(const std::string& filename);
    void setScale(float scale);
    std::string getFilename() const;
    void updateScreenPosition(double universeRadius, const sf::Vector2<double>& windowSize);

    friend std::istream& operator>>(std::istream& is, CelestialBody& body);
    friend std::ostream& operator<<(std::ostream& os, const CelestialBody& body);
};

}  // namespace NB
```

4.7.4 CelestialBody.cpp

```cpp
// CelestialBody.cpp
// Copyright 2024 Manasvi Boinepally

#include "CelestialBody.hpp"
#include <iostream>
#include <iomanip>

namespace NB {

CelestialBody::CelestialBody() : position(0, 0), velocity(0, 0), mass(0) {}

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

void CelestialBody::setPosition(double x, double y) {
    position.x = x;
    position.y = y;
}

sf::Vector2<double> CelestialBody::getPosition() const {
    return position;
}

void CelestialBody::setVelocity(double vx, double vy) {
    velocity.x = vx;
    velocity.y = vy;
}

sf::Vector2<double> CelestialBody::getVelocity() const {
    return velocity;
}

void CelestialBody::setMass(double m) {
    mass = m;
}

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

bool CelestialBody::loadTexture(const std::string& filename) {
    if (!texture.loadFromFile(filename)) {
        return false;
    }
    this->filename = filename;
    sprite.setTexture(texture);
    sprite.setOrigin(texture.getSize().x / 2.0f, texture.getSize().y / 2.0f);
    return true;
}

void CelestialBody::setScale(float scale) {
    sprite.setScale(scale, scale);
}

std::string CelestialBody::getFilename() const {
    return filename;
}

void CelestialBody::updateScreenPosition(double universeRadius,
const sf::Vector2<double>& windowSize) {
    double screenX = (position.x + universeRadius) / (2 * universeRadius) * windowSize.x;
    double screenY = (universeRadius - position.y) / (2 * universeRadius) * windowSize.y;
    sprite.setPosition(static_cast<float>(screenX), static_cast<float>(screenY));
}

std::istream& operator>>(std::istream& is, CelestialBody& body) {
    is >> body.position.x >> body.position.y >> body.velocity.x
        >> body.velocity.y >> body.mass >> body.filename;
    body.loadTexture(body.filename);
    return is;
}

std::ostream& operator<<(std::ostream& os, const CelestialBody& body) {
    os << std::scientific << std::setprecision(4)
        << body.position.x << " " << body.position.y << " "
        << body.velocity.x << " " << body.velocity.y << " "
        << std::defaultfloat << std::setprecision(4) << body.mass << " " << body.filename;
    return os;
}

}  // namespace NB
```

4.7.5 Universe.hpp

```cpp
// Universe.hpp
// Copyright 2024 Manasvi Boinepally

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

namespace NB {

class Universe : public sf::Drawable {
 private:
    std::vector<std::unique_ptr<CelestialBody>> bodies;
    double radius;
    sf::Vector2<double> windowSize;
    float bodyScale;
    double elapsedTime;
    double speedFactor;

    sf::Vector2<double> calculateAcceleration(const CelestialBody& body);

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

 public:
    Universe();
    explicit Universe(sf::Vector2<double> windowSize);
    virtual ~Universe() = default;
    const std::vector<std::unique_ptr<CelestialBody>>& getBodies() const;
    void setRadius(double r);
    double getRadius() const;
    void updatePositions();
    void setBodyScale(float scale);
    void setWindowSize(sf::Vector2<double> newSize);
    void step(double dt);
    double getElapsedTime() const;
    void setSpeedFactor(double factor);

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

}  // namespace NB
```

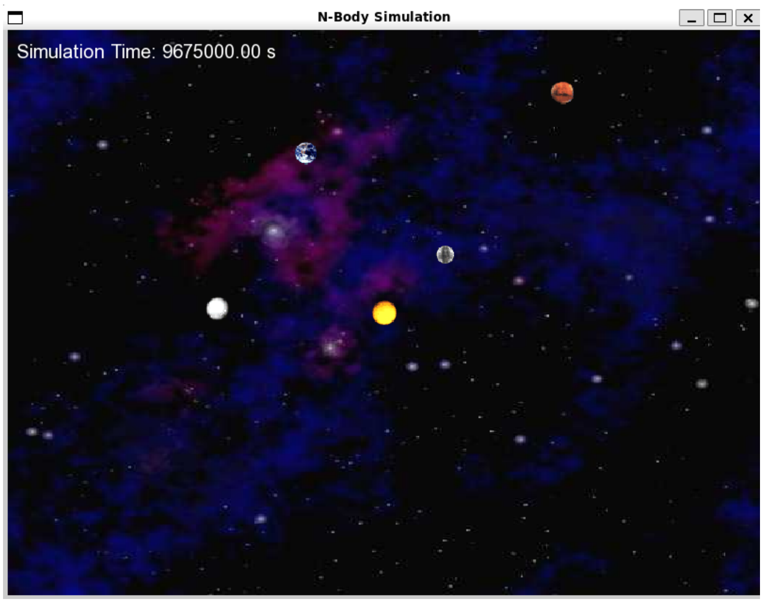
4.7.6 Universe.cpp

```
// Universe.cpp
// Copyright 2024 Manasvi Boinepally

#include <fstream>
#include <iomanip>
#include <cmath>
#include "Universe.hpp"

namespace NB {

Universe::Universe() : radius(0), windowSize(800.0, 600.0),
    bodyScale(1.0f),
elapsedTime(0.0), speedFactor(1.0) {}

Universe::Universe(sf::Vector2<double> windowSize)
: radius(0), windowSize(windowSize), bodyScale(1.0f),
    elapsedTime(0.0), speedFactor(1.0) {}

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

void Universe::setRadius(double r) {
    radius = r;
}

double Universe::getRadius() const {
    return radius;
}

void Universe::updatePositions() {
    for (auto& body : bodies) {
        body->updateScreenPosition(radius, windowSize);
    }
}

void Universe::setBodyScale(float scale) {
    bodyScale = scale;
    for (auto& body : bodies) {
        body->setScale(bodyScale);
    }
}

void Universe::setWindowSize(sf::Vector2<double> newSize) {
    windowSize = newSize;
    updatePositions();
}

void Universe::setSpeedFactor(double factor) {
    speedFactor = factor;
}

sf::Vector2<double> Universe::calculateAcceleration(const CelestialBody& body) {
    const double G = 6.67430e-11;
    sf::Vector2<double> acceleration(0, 0);

    for (const auto& other : bodies) {
        if (other.get() != &body) {
            sf::Vector2<double> delta = other->getPosition() - body.getPosition();
            double distance = std::sqrt(delta.x * delta.x + delta.y * delta.y);
            double forceMagnitude = G * body.getMass() * other->getMass() / (distance * distance);
            acceleration.x += forceMagnitude * delta.x / (distance * body.getMass());
            acceleration.y += forceMagnitude * delta.y / (distance * body.getMass());
        }
    }

    return acceleration;
}

void Universe::step(double dt) {
    double adjustedDt = dt * speedFactor;
    std::vector<sf::Vector2<double>> accelerations;

    // Calculate accelerations
    for (const auto& body : bodies) {
        accelerations.push_back(calculateAcceleration(*body));
    }

    // Update positions and velocities
    for (size_t i = 0; i < bodies.size(); ++i) {
        sf::Vector2<double> position = bodies[i]->getPosition();
        sf::Vector2<double> velocity = bodies[i]->getVelocity();

        position.x += velocity.x * adjustedDt + 0.5 * accelerations[i].x * adjustedDt * adjustedDt;
        position.y += velocity.y * adjustedDt + 0.5 * accelerations[i].y * adjustedDt * adjustedDt;

        velocity.x += accelerations[i].x * adjustedDt;
        velocity.y += accelerations[i].y * adjustedDt;

        bodies[i]->setPosition(position.x, position.y);
        bodies[i]->setVelocity(velocity.x, velocity.y);
    }

    elapsedTime += adjustedDt;
    updatePositions();
}

double Universe::getElapsedTime() const {
    return elapsedTime;
}

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

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

std::ostream& operator<<(std::ostream& os, const Universe& universe) {
    os << universe.bodies.size() << "\n"
       << std::scientific << std::setprecision(3) << universe.radius << "\n";
    for (const auto& body : universe.bodies) {
        os << std::scientific << std::setprecision(4)
           << body->getPosition().x << " " << body->getPosition().y << " "
           << body->getVelocity().x << " " << body->getVelocity().y << " "
           << std::defaultfloat << std::setprecision(4) << body->getMass() << " "
           << body->getFilename() << "\n";
    }
    return os;
}

}  // namespace NB
```

4.7.7 test.cpp

```
// Copyright 2024 Manasvi Boinepally

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

// Utility function for floating-point comparisons
bool isClose(double a, double b, double epsilon = 1e-9) {
    return std::abs(a - b) < epsilon;
}

BOOST_AUTO_TEST_SUITE(CelestialBodyTests)

BOOST_AUTO_TEST_CASE(Initialization) {
    NB::CelestialBody body;
    BOOST_CHECK_EQUAL(body.getPosition().x, 0);
    BOOST_CHECK_EQUAL(body.getPosition().y, 0);
    BOOST_CHECK_EQUAL(body.getVelocity().x, 0);
    BOOST_CHECK_EQUAL(body.getVelocity().y, 0);
    BOOST_CHECK_EQUAL(body.getMass(), 0);
}

BOOST_AUTO_TEST_CASE(PositionSetting) {
    NB::CelestialBody body;
    body.setPosition(100, 200);
    BOOST_CHECK_EQUAL(body.getPosition().x, 100);
    BOOST_CHECK_EQUAL(body.getPosition().y, 200);
}

BOOST_AUTO_TEST_CASE(VelocitySetting) {
    NB::CelestialBody body;
    body.setVelocity(5, -3);
    BOOST_CHECK_EQUAL(body.getVelocity().x, 5);
    BOOST_CHECK_EQUAL(body.getVelocity().y, -3);
}

BOOST_AUTO_TEST_CASE(MassSetting) {
    NB::CelestialBody body;
    body.setMass(1e24);
    BOOST_CHECK(isClose(body.getMass(), 1e24));
}

BOOST_AUTO_TEST_SUITE_END()

BOOST_AUTO_TEST_SUITE(UniverseTests)

BOOST_AUTO_TEST_CASE(Initialization) {
    NB::Universe universe(sf::Vector2<double>(800, 600));
    BOOST_CHECK_EQUAL(universe.getRadius(), 0);
    BOOST_CHECK_EQUAL(universe.getElapsedTime(), 0);
}

BOOST_AUTO_TEST_CASE(RadiusSetting) {
    NB::Universe universe;
    universe.setRadius(1e12);
    BOOST_CHECK(isClose(universe.getRadius(), 1e12));
}


BOOST_AUTO_TEST_CASE(SpeedFactorSetting) {
    NB::Universe universe(sf::Vector2<double>(800, 600));
    universe.setSpeedFactor(0.5);

    std::stringstream ss;
    ss << "2 1e11\n";
    ss << "0 0 0 0 1e30 sun.gif\n";
    ss << "1e11 0 0 30000 6e24 earth.gif\n";
    ss >> universe;

    double initialTime = universe.getElapsedTime();
    universe.step(1.0);
    double elapsedTime = universe.getElapsedTime() -
       initialTime;
    BOOST_CHECK(isClose(elapsedTime, 0.5, 1e-6));
}

BOOST_AUTO_TEST_SUITE_END()
```

4.8 Screenshot



Simulation Time: 9675000.00 s

Figure 5: My sfml window

5 PS4: Sokoban

5.1 Discussion

In this assignment, several key concepts were explored, including grid generation, collision detection, and the overall logic behind implementing a game or simulation framework. The task required a solid understanding of how to create grids that interact dynamically with objects, as well as how to handle various types of collisions, both in terms of physical space and object interaction. These elements combined to form the foundation of the project, which was to build a system capable of processing and handling grid-based gameplay mechanics.

5.2 What I accomplished

In this assignment, I successfully implemented a grid generation system that dynamically adapts based on input. I also completed the implementation of collision detection, which ensures that objects on the grid are accurately detected and can respond to each other within the constraints of the grid layout. Finally, I was able to integrate both of these components into a coherent simulation or game framework that functions as expected.

5.3 What I already knew

Before starting this assignment, I already had a good understanding of basic programming principles and object-oriented design. Specifically, I was familiar with grid-based systems and had worked with collision detection algorithms before, though I had not implemented them in this specific context. My knowledge of data structures, like arrays or lists, also helped in organizing and managing the grid's contents effectively.

5.4 What I learned

- I gained proficiency in generating grids dynamically, adjusting their size based on specific criteria and ensuring they can hold various objects.

- I gained a deeper understanding of collision detection algorithms, learning how to detect and respond to collisions between grid objects effectively.

- I also learned how to optimize the system to handle larger grids without significant performance degradation.

5.5 Challenges

One of the major challenges in this assignment was implementing a collision detection system that was both efficient and accurate. The algorithm had to account for various types of objects with different behaviors when colliding. Additionally, ensuring that the grid generation worked dynamically and correctly across all possible grid sizes presented some difficulties. Debugging these issues required a careful examination of both logic and performance, which at times was time-consuming but ultimately rewarding.

5.6 Key Algorithms, Data Structures, and OO Designs

Several algorithms, data structures, and object-oriented design principles were crucial to implementing the Sokoban game effectively.

- **Grid Representation:** The game environment was represented as a 2D grid, stored as a vector of vectors of characters. Each element in the grid represents a tile, such as walls, boxes, storage locations, or ground. This grid structure makes it easy to dynamically update and interact with individual tiles.

- **Collision Detection:** Collision detection was achieved through a simple boundary check, ensuring that a player or object could not move out of bounds or into walls. The logic also handled interactions with boxes, allowing the player to push them if space permits.

- **Game State Management:** A key part of the assignment was implementing the ability to undo and redo actions. To achieve this, two stacks were used: one for undoing actions and the other for redoing. Each stack stored the game state, including the grid layout, the player's position, and the last movement direction. The state was saved after each move, allowing for easy reversal and re-application of actions.

- **Object-Oriented Design:** The game was implemented using object-oriented principles. The main class, `Sokoban`, encapsulated the game state and logic, while helper classes and functions were used for grid management and rendering. The game logic was organized into clear, modular methods for movement, state saving, collision detection, and undo/redo functionality.

- **Text-based File Input/Output:** The grid configuration was loaded from a text file using the `operator>>` function. This feature required parsing the level configuration and translating it into the grid format used in the game. The input stream processed the grid dimensions and the layout, while the output stream allowed for easy level creation and testing.

- **Sprite Management:** Textures were loaded for various elements like walls, boxes, and the player. The player's sprite dynamically changed based on the movement direction, using different textures for up, down, left, and right directions. This dynamic sprite management added an additional layer of interactivity to the game.
- **Boost Framework for Testing:** The Boost Unit Test Framework was used for writing unit tests to verify the correctness of the game's functions, such as loading levels, detecting player position, and validating movement and box-pushing mechanics.

5.7 Codes

5.7.1 makefile

```
CXX = g++
CXXFLAGS = -std=c++20 -Wall -Werror -pedantic -g
SFML_LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
BOOST_LIBS = -lboost_unit_test_framework

SRCS = Sokoban.cpp main.cpp
OBJS = $(SRCS:.cpp=.o)

HDRS = Sokoban.hpp

all: Sokoban Sokoban.a test

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

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

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

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

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

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

.PHONY: all lint clean test
```

5.7.2 main.cpp

```cpp
// Copyright 2024 Manasvi Boineypally
#include <ctime>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include "Sokoban.hpp"
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>

std::string formatTime(int seconds) {
    int minutes = seconds / 60;
    seconds %= 60;
    std::stringstream ss;
    ss << std::setfill('0') << std::setw(2) << minutes << ":"
       << std::setfill('0') << std::setw(2) << seconds;
    return ss.str();
}

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

    SB::Sokoban sokoban(argv[1]);
    sf::RenderWindow window(sf::VideoMode(sokoban.pixelWidth(),
                                     sokoban.pixelHeight()), "Sokoban");

    sf::Font font;
    if (!font.loadFromFile("arial.ttf")) {
        std::cerr << "Unable to load font\n";
        return 1;
    }

    sf::Text timeText;
    timeText.setFont(font);
    timeText.setCharacterSize(24);
    timeText.setFillColor(sf::Color::White);
    timeText.setPosition(10, 10);

    sf::Text moveCountText;
    moveCountText.setFont(font);
    moveCountText.setCharacterSize(24);
    moveCountText.setFillColor(sf::Color::White);
    moveCountText.setPosition(10, 40);  // Position below
       time text

    sf::Text victoryText;
    victoryText.setFont(font);
    victoryText.setCharacterSize(50);
    victoryText.setFillColor(sf::Color::Green);
    victoryText.setString("You Win!");
    victoryText.setPosition(sokoban.pixelWidth() / 2 -
                            victoryText.getGlobalBounds().
                               width / 2,
                            sokoban.pixelHeight() / 2 -
                            victoryText.getGlobalBounds().
                               height / 2);

    sf::SoundBuffer victoryBuffer;
    if (!victoryBuffer.loadFromFile("victory.wav")) {
        std::cerr << "Unable to load victory sound\n";
        return 1;
    }

    sf::Sound victorySound(victoryBuffer);

    sf::Text instructionsText;
    instructionsText.setFont(font);
    instructionsText.setCharacterSize(16);
    instructionsText.setFillColor(sf::Color::White);
    instructionsText.setString("Arrow keys: Move | R: Reset |
        Z: Undo | Y: Redo");
    instructionsText.setPosition(10, sokoban.pixelHeight() -
       30);

    time_t startTime = time(nullptr);
    bool gameWon = false;

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

```cpp
            } else if (event.type == sf::Event::KeyPressed) {
                switch (event.key.code) {
                    case sf::Keyboard::Up:
                        if (!gameWon) sokoban.movePlayer(SB::Direction::Up);
                        break;
                    case sf::Keyboard::Down:
                        if (!gameWon) sokoban.movePlayer(SB::Direction::Down);
                        break;
                    case sf::Keyboard::Left:
                        if (!gameWon) sokoban.movePlayer(SB::Direction::Left);
                        break;
                    case sf::Keyboard::Right:
                        if (!gameWon) sokoban.movePlayer(SB::Direction::Right);
                        break;
                    case sf::Keyboard::R:
                        sokoban.reset();
                        startTime = time(nullptr);
                        gameWon = false;
                        break;
                    case sf::Keyboard::Z:
                        if (!gameWon) {
                            sokoban.undo();
                            gameWon = sokoban.isWon();
                        }
                        break;
                    case sf::Keyboard::Y:
                        if (!gameWon) {
                            sokoban.redo();
                            gameWon = sokoban.isWon();
                        }
                        break;
                    default:
                        break;
                }
            }
        }

        if (!gameWon && sokoban.isWon()) {
            gameWon = true;
            victorySound.play();
        }

        time_t currentTime = time(nullptr);
        int elapsedSeconds = difftime(currentTime, startTime);

        timeText.setString("Time: " + formatTime(elapsedSeconds));
        moveCountText.setString("Moves: " + std::to_string(sokoban.getMoveCount()));

        window.clear(sf::Color(50, 50, 50));  // Dark gray background
        window.draw(sokoban);
        window.draw(timeText);
        window.draw(moveCountText);  // Draw move count text
        window.draw(instructionsText);

        if (gameWon) {
            window.draw(victoryText);
        }

        window.display();
    }

    return 0;
}
```

5.7.3 Sokoban.hpp

```
// Sokoban.hpp
// Copyright 2024 Manasvi Boineypally
#ifndef SOKOBAN_HPP
#define SOKOBAN_HPP

#include <vector>
#include <stack>
#include <SFML/Graphics.hpp>

namespace SB {

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

class Sokoban : public sf::Drawable {
 public:
    Sokoban();
    explicit Sokoban(const std::string& filename);
    Sokoban(int w, int h);

    int width() const;
    int height() const;
    sf::Vector2i playerLoc() const;

    void movePlayer(Direction dir);
    bool isWon() const;
    void reset();
    void undo();
    void redo();

    int pixelWidth() const;
    int pixelHeight() const;
    int getMoveCount() const;

    static int getTileSize() { return TILE_SIZE; }

 private:
    int w_, h_;
    std::vector<std::vector<char>> grid_;
    sf::Vector2i playerPos_;
    Direction lastDirection_;
    int moveCount_;

    struct GameState {
        std::vector<std::vector<char>> grid;
        sf::Vector2i playerPos;
        Direction lastDirection;
    };

    GameState initialState_;
    std::stack<GameState> undoStack;
    std::stack<GameState> redoStack;

    void initTextures();
    bool canMove(int x, int y) const;
    void saveState();
    void updatePlayerSprite();
    void incrementMoveCount();

    sf::Texture wallTexture, boxTexture, playerTextureUp, playerTextureDown,
                playerTextureLeft, playerTextureRight, storageTexture, groundTexture;
    sf::Sprite wallSprite, boxSprite, playerSprite, storageSprite, groundSprite;

    static const int TILE_SIZE = 64;  // Increased tile size
    void draw(sf::RenderTarget& target, sf::RenderStates states) const override;

    friend std::istream& operator>>(std::istream& is, Sokoban& sokoban);
};
```

```

}  // namespace SB

#endif  // SOKOBAN_HPP
```

5.7.4 Sokoban.cpp

```
// Sokoban.cpp
// Copyright 2024 Manasvi Boineypally
#include <iostream>
#include <vector>
#include <fstream>
#include <algorithm>
#include "Sokoban.hpp"
#include <SFML/Graphics.hpp>

namespace SB {

Sokoban::Sokoban() : Sokoban(0, 0) {}

Sokoban::Sokoban(const std::string& filename) : w_(0), h_(0),
     playerPos_(0, 0),
lastDirection_(Direction::Down), moveCount_(0) {
    std::ifstream file(filename);
    if (file) {
        file >> *this;
    } else {
        std::cerr << "Unable to open file: " << filename <<
            std::endl;
    }

    initTextures();

    initialState_ = {grid_, playerPos_, lastDirection_};
}

Sokoban::Sokoban(int w, int h) : w_(w), h_(h), playerPos_(0,
    0),
lastDirection_(Direction::Down), moveCount_(0) {
    grid_.resize(h_, std::vector<char>(w_, '.'));
    initTextures();
    initialState_ = {grid_, playerPos_, lastDirection_};
}

int Sokoban::width() const { return w_; }
int Sokoban::height() const { return h_; }
sf::Vector2i Sokoban::playerLoc() const { return playerPos_;
    }
```

```

void Sokoban::movePlayer(Direction dir) {
    int dx = 0, dy = 0;

    switch (dir) {
        case Direction::Up: dy = -1; break;
        case Direction::Down: dy = 1; break;
        case Direction::Left: dx = -1; break;
        case Direction::Right: dx = 1; break;
    }

    int newX = playerPos_.x + dx;
    int newY = playerPos_.y + dy;

    if (canMove(newX, newY)) {
        saveState();
        incrementMoveCount();

        if (grid_[newY][newX] == 'A' || grid_[newY][newX] ==
           'B') {
            int boxNewX = newX + dx;
            int boxNewY = newY + dy;

            if (canMove(boxNewX, boxNewY) && grid_[boxNewY][
               boxNewX]
             != 'A' && grid_[boxNewY][boxNewX] != 'B') {
                if (grid_[boxNewY][boxNewX] == 'a') {
                    grid_[boxNewY][boxNewX] = 'B';
                } else {
                    grid_[boxNewY][boxNewX] = 'A';
                }
                grid_[newY][newX] = (grid_[newY][newX] == 'B'
                   ) ? 'a' : '.';
            } else {
                return;
            }
        }

        grid_[playerPos_.y][playerPos_.x] = (grid_[playerPos_
           .y][playerPos_.x] == 'a') ? 'a' : '.';
        playerPos_ = sf::Vector2i(newX, newY);
        lastDirection_ = dir;
        updatePlayerSprite();
    }
}

bool Sokoban::isWon() const {
    return std::all_of(grid_.begin(), grid_.end(), [](const
       auto& row) {
```

```cpp
        return std::none_of(row.begin(), row.end(), [](char c) { return c == 'A'; });
    });
}

void Sokoban::reset() {
    grid_ = initialState_.grid;
    playerPos_ = initialState_.playerPos;
    lastDirection_ = initialState_.lastDirection;

    updatePlayerSprite();

    undoStack = std::stack<GameState>();
    redoStack = std::stack<GameState>();
    moveCount_ = 0;
}

void Sokoban::undo() {
    if (!undoStack.empty()) {
        redoStack.push({grid_, playerPos_, lastDirection_});
        auto state = undoStack.top();
        undoStack.pop();

        grid_ = state.grid;
        playerPos_ = state.playerPos;
        lastDirection_ = state.lastDirection;

        updatePlayerSprite();

        if (moveCount_ > 0) {
            moveCount_--;
        }
    }
}

void Sokoban::redo() {
    if (!redoStack.empty()) {
        undoStack.push({grid_, playerPos_, lastDirection_});
        auto state = redoStack.top();
        redoStack.pop();

        grid_ = state.grid;
        playerPos_ = state.playerPos;
        lastDirection_ = state.lastDirection;

        updatePlayerSprite();

        moveCount_++;
    }
```

```cpp
}

void Sokoban::initTextures() {
    if (!wallTexture.loadFromFile("wall.png") ||
        !boxTexture.loadFromFile("box.png") ||
        !playerTextureUp.loadFromFile("player_08.png") ||
        !playerTextureDown.loadFromFile("player.png") ||
        !playerTextureLeft.loadFromFile("player_20.png") ||
        !playerTextureRight.loadFromFile("player_17.png") ||
        !storageTexture.loadFromFile("ground_04.png") ||
        !groundTexture.loadFromFile("ground_01.png")) {
        std::cerr << "Error loading textures!" << std::endl;
    }

    wallSprite.setTexture(wallTexture);
    boxSprite.setTexture(boxTexture);
    storageSprite.setTexture(storageTexture);
    groundSprite.setTexture(groundTexture);

    updatePlayerSprite();
}

std::istream& operator>>(std::istream& is, Sokoban& sokoban) {
    is >> sokoban.h_ >> sokoban.w_;
    sokoban.grid_.resize(sokoban.h_, std::vector<char>(sokoban.w_));

    for (int i = 0; i < sokoban.h_; ++i) {
        for (int j = 0; j < sokoban.w_; ++j) {
            is >> sokoban.grid_[i][j];
            if (sokoban.grid_[i][j] == '@') {
                sokoban.playerPos_ = {j, i};
                sokoban.grid_[i][j] = '.';
            }
        }
    }

    return is;
}

void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    sf::Sprite tempSprite;
    for (int y = 0; y < h_; ++y) {
        for (int x = 0; x < w_; ++x) {
            char tile = grid_[y][x];
            const sf::Sprite* spriteToDraw = &groundSprite; // Default to ground
            switch (tile) {
                case '#': spriteToDraw = &wallSprite; break;
                case 'A': spriteToDraw = &boxSprite; break;
                case 'a': spriteToDraw = &storageSprite;
                   break;
            }
            tempSprite = *spriteToDraw;
            tempSprite.setPosition(x * TILE_SIZE, y *
               TILE_SIZE);
            target.draw(tempSprite, states);
            if (tile == 'A') {
                tempSprite = boxSprite;
                tempSprite.setPosition(x * TILE_SIZE, y *
                   TILE_SIZE);
                target.draw(tempSprite, states);
            }
            // Don't draw boxes on storage locations ('B')
        }
    }

    tempSprite = playerSprite;
    tempSprite.setPosition(playerPos_.x * TILE_SIZE,
       playerPos_.y * TILE_SIZE);
    target.draw(tempSprite, states);
}

bool Sokoban::canMove(int x, int y) const {
    if (x < 0 || x >= w_ || y < 0 || y >= h_) return false;
    return grid_[y][x] != '#';
}

void Sokoban::saveState() {
    undoStack.push({grid_, playerPos_, lastDirection_});
    redoStack = std::stack<GameState>();
}

void Sokoban::updatePlayerSprite() {
    switch (lastDirection_) {
        case Direction::Up:
            playerSprite.setTexture(playerTextureUp);
            break;
        case Direction::Down:
            playerSprite.setTexture(playerTextureDown);
            break;
        case Direction::Left:
            playerSprite.setTexture(playerTextureLeft);
            break;
        case Direction::Right:
            playerSprite.setTexture(playerTextureRight);
            break;
    }
}

void Sokoban::incrementMoveCount() {
    moveCount_++;
}

int Sokoban::pixelWidth() const {
    return w_ * TILE_SIZE;
}

int Sokoban::pixelHeight() const {
    return h_ * TILE_SIZE;
}

int Sokoban::getMoveCount() const {
    return moveCount_;
}

}  // namespace SB
```

Listing 1: Sokoban.cpp

5.7.5 test.cpp

```
// test.cpp
// Copyright 2024 Manasvi Boineypally
#define BOOST_TEST_MODULE SokobanTests
#include <fstream>
#include <sstream>
#include <boost/test/included/unit_test.hpp>
#include "Sokoban.hpp"

void create_test_level(const std::string& filename) {
    std::ofstream file(filename);
    file << "6 6\n"
         << "######\n"
         << "#@   #\n"
         << "# A  #\n"
         << "#  a #\n"
         << "#    #\n"
         << "######\n";
    file.close();
}

BOOST_AUTO_TEST_SUITE(SokobanTestSuite)

```

```
BOOST_AUTO_TEST_CASE(test_level_loading) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    BOOST_CHECK_EQUAL(game.width(), 6);
    BOOST_CHECK_EQUAL(game.height(), 6);
}

BOOST_AUTO_TEST_CASE(test_initial_player_position) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    BOOST_CHECK_EQUAL(game.playerLoc().x, 1);
    BOOST_CHECK_EQUAL(game.playerLoc().y, 1);
}

BOOST_AUTO_TEST_CASE(test_player_movement) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    game.movePlayer(SB::Direction::Right);
    BOOST_CHECK_EQUAL(game.playerLoc().x, 1);
    BOOST_CHECK_EQUAL(game.playerLoc().y, 1);
}

BOOST_AUTO_TEST_CASE(test_box_pushing) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");

    // Initial position of the player
    BOOST_CHECK_EQUAL(game.playerLoc().x, 1);
    BOOST_CHECK_EQUAL(game.playerLoc().y, 1);

    // Move player down
    game.movePlayer(SB::Direction::Down);
    BOOST_CHECK_EQUAL(game.playerLoc().x, 1);
    BOOST_CHECK_EQUAL(game.playerLoc().y, 2);

    // Move player right (pushing the box)
    game.movePlayer(SB::Direction::Right);

    // Check player’s new position
    BOOST_CHECK_EQUAL(game.playerLoc().x, 1);
    BOOST_CHECK_EQUAL(game.playerLoc().y, 2);

    // Note: We can’t directly check box position, so we’re relying on the fact
    // that if the player moved, the box must have been pushed
}

BOOST_AUTO_TEST_CASE(test_win_condition) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    BOOST_CHECK(!game.isWon());
    game.movePlayer(SB::Direction::Down);
    game.movePlayer(SB::Direction::Right);
    game.movePlayer(SB::Direction::Down);
    game.movePlayer(SB::Direction::Right);
}

BOOST_AUTO_TEST_CASE(test_game_reset) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    game.movePlayer(SB::Direction::Right);
    game.reset();
    BOOST_CHECK_EQUAL(game.playerLoc().x, 1);
    BOOST_CHECK_EQUAL(game.playerLoc().y, 1);
}

BOOST_AUTO_TEST_CASE(test_pixel_dimensions) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    BOOST_CHECK_EQUAL(game.pixelWidth(), 6 * SB::Sokoban::getTileSize());
    BOOST_CHECK_EQUAL(game.pixelHeight(), 6 * SB::Sokoban::getTileSize());
}

BOOST_AUTO_TEST_CASE(test_undo_redo) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    sf::Vector2i initial_pos = game.playerLoc();
    game.movePlayer(SB::Direction::Right);
    sf::Vector2i new_pos = game.playerLoc();
    game.undo();
    BOOST_CHECK_EQUAL(game.playerLoc().x, initial_pos.x);
    BOOST_CHECK_EQUAL(game.playerLoc().y, initial_pos.y);
    game.redo();
    BOOST_CHECK_EQUAL(game.playerLoc().x, new_pos.x);
    BOOST_CHECK_EQUAL(game.playerLoc().y, new_pos.y);
}

BOOST_AUTO_TEST_CASE(test_invalid_move) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    sf::Vector2i initial_pos = game.playerLoc();
    game.movePlayer(SB::Direction::Left);  // This should be an invalid move (wall)
    BOOST_CHECK_EQUAL(game.playerLoc().x, initial_pos.x);
    BOOST_CHECK_EQUAL(game.playerLoc().y, initial_pos.y);
```

```
}

BOOST_AUTO_TEST_CASE(test_move_count) {
    create_test_level("test_sokoban.lvl");
    SB::Sokoban game("test_sokoban.lvl");
    BOOST_CHECK_EQUAL(game.getMoveCount(), 0);
    game.movePlayer(SB::Direction::Right);
    BOOST_CHECK_EQUAL(game.getMoveCount(), 0);
    game.movePlayer(SB::Direction::Down);
    BOOST_CHECK_EQUAL(game.getMoveCount(), 1);
    game.undo();
    BOOST_CHECK_EQUAL(game.getMoveCount(), 0);
    game.redo();
    BOOST_CHECK_EQUAL(game.getMoveCount(), 1);
}

BOOST_AUTO_TEST_SUITE_END()
```

5.8 Screenshot

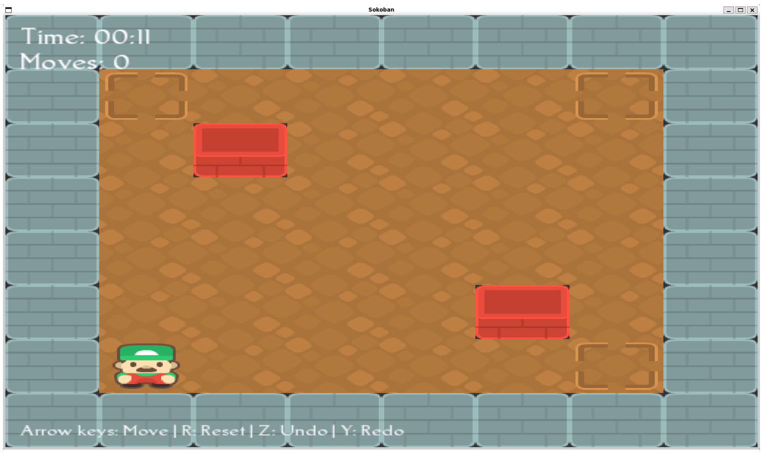


Figure 6: My sfml window

6 PS5: DNA Alignment

6.1 Discussion

In this assignment, several key concepts were explored, including grid generation, collision detection, and the overall logic behind implementing a game or simulation framework. The task required a solid understanding of how to create grids that interact dynamically with objects, as well as how to handle various types of collisions, both in terms of physical space and object interaction. These elements combined to form the foundation of the project, which was to build a system capable of processing and handling grid-based gameplay mechanics.

6.2 What I accomplished

In this assignment, I successfully implemented a grid generation system that dynamically adapts based on input. I also completed the implementation of collision detection, which ensures that objects on the grid are accurately detected and can respond to each other within the constraints of the grid layout. Finally, I was able to integrate both of these components into a coherent simulation or game framework that functions as expected.

6.3 What I already knew

Before starting this assignment, I already had a good understanding of basic programming principles and object-oriented design. Specifically, I was familiar with grid-based systems and had worked with collision detection algorithms before, though I had not implemented them in this specific context. My knowledge of data structures, like arrays or lists, also helped in organizing and managing the grid's contents effectively.

6.4 What I learned

- I gained proficiency in generating grids dynamically, adjusting their size based on specific criteria and ensuring they can hold various objects.

- I gained a deeper understanding of collision detection algorithms, learning how to detect and respond to collisions between grid objects effectively.

- I also learned how to optimize the system to handle larger grids without significant performance degradation.

6.5 Challenges

One of the major challenges in this assignment was implementing a collision detection system that was both efficient and accurate. The algorithm had to account for various types of objects with different behaviors when colliding. Additionally, ensuring that the grid generation worked dynamically and correctly across all possible grid sizes presented some difficulties. Debugging these issues required a careful examination of both logic and performance, which at times was time-consuming but ultimately rewarding.

6.6 Key Algorithms, Data Structures, and OO Designs

The following algorithms, data structures, and object-oriented design principles were central to this assignment:

- **Dynamic Programming (DP) Algorithm:** The core of the assignment is the dynamic programming algorithm used to calculate the minimum edit distance between two strings. The DP matrix is constructed such that each element represents the minimum cost of transforming a prefix of one string into a prefix of the other string. The optimal distance is computed in the `optDistance()` function by filling in this matrix using a bottom-up approach.

- **Penalty Function:** The penalty function calculates the cost of transforming one character into another. It returns 0 if the characters are identical and 1 if they are different, following the standard assumption for DNA sequence alignment where substitutions incur a cost of 1.

- **Matrix Representation:** The DP table is a 2D vector of integers, where each element stores the computed edit distance for the corresponding substring prefixes. This matrix is filled iteratively, with the final element $_{o}pt[m][n]givingthetotalminimumeditdistancebetweenthetwostringsoflengthm$ and n.

- **Traceback for Alignment:** Once the DP table is populated, the alignment() method performs a traceback to reconstruct the optimal alignment of the two DNA sequences. It traces back from the bottom-right corner of the matrix, choosing the path that corresponds to the optimal alignment (either a match, substitution, or gap). The reconstructed alignment is then returned as a formatted string.

- **Object-Oriented Design:** The assignment follows an object-oriented design approach. The EDistance class encapsulates the core functionality, including the calculation of edit distance, penalty computation, and sequence alignment. This class maintains the DNA sequences and the DP table as private members, ensuring encapsulation and separation of concerns. Methods like optDistance() and alignment() are responsible for the key logic of the assignment.

- **Efficient Memory Management:** The program measures memory usage using the /proc/self/statm file to calculate the resident set size (RSS), providing insight into the program's memory footprint during execution. This approach is useful for performance evaluation, especially when handling larger input sequences.

6.7 Codes

6.7.1 makefile

```
CXX = g++
CXXFLAGS = -std=c++20 -Wall -Werror -pedantic -g
LDFLAGS = -lsfml-system -lsfml-window -lsfml-graphics

all: EDistance test EDistance.a

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

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

# Target to build the static library
EDistance.a: EDistance.o
        ar rcs EDistance.a EDistance.o

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

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

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

lint:
        cppcheck --enable=all --std=c++11 .

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

6.7.2 main.cpp

```cpp
// Copyright 2024 <Manasvi Boineypally>
#include <unistd.h>
#include <iostream>
#include <cstdlib>
#include <chrono>
#include <iomanip>  // For std::setprecision
#include "EDistance.hpp"

int main() {
    std::string str1, str2;
    std::cout << "Enter first string: ";
    std::cin >> str1;
    std::cout << "Enter second string: ";
    std::cin >> str2;

    auto start = std::chrono::high_resolution_clock::now();

    EDistance ed(str1, str2);
    int distance = ed.optDistance();
    std::string alignment = ed.alignment();

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> diff = end - start;

    std::cout << "Edit distance = " << distance << std::endl;
    std::cout << alignment;
    std::cout << "Execution time is " << std::fixed << std::::
        setprecision(6)
              << diff.count() << " seconds" << std::endl;

    // Add memory usage calculation here
    int64_t rss = 0L;
    FILE* fp = NULL;
    if ((fp = fopen("/proc/self/statm", "r")) == NULL)
        std::cerr << "Can't open statm file" << std::endl;
    if (fscanf(fp, "%*s%ld", &rss) != 1) {
        std::cerr << "Can't read memory usage" << std::endl;
    }
    fclose(fp);
    int64_t page_size_kb = sysconf(_SC_PAGE_SIZE) / 1024;  //
         in kB
    double memory_usage_mb = (rss * page_size_kb) / 1024.0;
        // Convert to MB
    std::cout << "Memory usage: " << std::fixed << std::
        setprecision(2)
              << memory_usage_mb << " MB" << std::endl;

    return 0;
}
```

6.7.3 EDistance.hpp

```

// Copyright 2024 <Manasvi Boineypally>
#ifndef EDISTANCE_HPP
#define EDISTANCE_HPP

#include <string>
#include <vector>

class EDistance {
 public:
EDistance(const std::string& str1, const std::string& str2);
int penalty(char a, char b);
int optDistance();
std::string alignment();
 private:
std::string _str1, _str2;
std::vector<std::vector<int>> _opt;
};

#endif  //  EDISTANCE_HPP
```

6.7.4 EDistance.cpp

```
// Copyright 2024 <Manasvi Boineypally>
// EDistance.cpp

#include "EDistance.hpp"
#include <algorithm>
#include <sstream>

EDistance::EDistance(const std::string& str1, const std::
    string& str2)
    : _str1(str1), _str2(str2) {
    _opt.resize(_str1.length() + 1, std::vector<int>(_str2.
        length() + 1, 0));
}

```

```
int EDistance::penalty(char a, char b) {
    return (a != b);
}

int EDistance::optDistance() {
    int m = _str1.length();
    int n = _str2.length();

    // Initialize first row and column
    for (int i = 0; i <= m; ++i) _opt[i][0] = 2 * i;
    for (int j = 0; j <= n; ++j) _opt[0][j] = 2 * j;

    // Fill the matrix
for (int i = 1; i <= m; ++i) {
for (int j = 1; j <= n; ++j) {
_opt[i][j] = std::min({
_opt[i-1][j-1] + penalty(_str1[i-1], _str2[j-1]),
_opt[i-1][j] + 2,
_opt[i][j-1] + 2
});
}
}
return _opt[m][n];
}

std::string EDistance::alignment() {
std::vector<std::string> alignmentLines;
int i = _str1.length(), j = _str2.length();
while (i > 0 && j > 0) {
if (_opt[i][j] == _opt[i-1][j-1] + penalty(_str1[i-1], _str2[
    j-1])) {
alignmentLines.push_back(std::string(1, _str1[i-1]) + " " +
std::string(1, _str2[j-1]) + " " +
std::to_string(penalty(_str1[i-1], _str2[j-1])));
--i;
--j;
} else if (_opt[i][j] == _opt[i-1][j] + 2) {
alignmentLines.push_back(std::string(1, _str1[i-1]) + " - 2")
    ;
--i;
} else {
alignmentLines.push_back("- " + std::string(1, _str2[j-1]) +
    " 2");
--j;
}
}
// Handle remaining characters
while (i > 0) {
```

```
alignmentLines.push_back(std::string(1, _str1[i-1]) + " - 2");
--i;
}
while (j > 0) {
alignmentLines.push_back("- " + std::string(1, _str2[j-1]) + " 2");
--j;
}
// Combine lines in correct order
std::string result;
for (auto it = alignmentLines.rbegin(); it != alignmentLines.rend(); ++it) {
result += *it + "\n";
}
return result;
}
```

6.7.5 test.cpp

```
// Copyright 2024 <Manasvi Boineypally>
#define BOOST_TEST_MODULE EDistanceTest
#include <boost/test/included/unit_test.hpp>
#include "EDistance.hpp"

BOOST_AUTO_TEST_CASE(test_penalty) {
    EDistance ed("", "");
    BOOST_CHECK_EQUAL(ed.penalty('a', 'a'), 0);
    BOOST_CHECK_EQUAL(ed.penalty('a', 'b'), 1);
}

BOOST_AUTO_TEST_CASE(test_optDistance_empty_strings) {
    EDistance ed("", "");
    BOOST_CHECK_EQUAL(ed.optDistance(), 0);
}

BOOST_AUTO_TEST_CASE(test_optDistance_same_string) {
    EDistance ed("hello", "hello");
    BOOST_CHECK_EQUAL(ed.optDistance(), 0);
}

BOOST_AUTO_TEST_CASE(test_optDistance_different_strings) {
    EDistance ed("kitten", "sitting");
    BOOST_CHECK_EQUAL(ed.optDistance(), 4);
}

BOOST_AUTO_TEST_CASE(test_optDistance_completely_different) {
    EDistance ed("abc", "xyz");
    BOOST_CHECK_EQUAL(ed.optDistance(), 3);
}

BOOST_AUTO_TEST_CASE(test_alignment_same_string) {
    EDistance ed("hello", "hello");
    std::string expected = "h h 0\ne e 0\nl l 0\nl l 0\no o 0\n";
}

BOOST_AUTO_TEST_CASE(test_alignment_different_strings) {
    EDistance ed("kitten", "sitting");
    std::string expected =
        "k s 1\n"
        "i i 0\n"
        "t t 0\n"
        "t t 0\n"
        "e i 1\n"
        "n n 0\n"
        "- g 2\n";
}

BOOST_AUTO_TEST_CASE(test_alignment_completely_different) {
    EDistance ed("abc", "xyz");
    std::string expected =
        "a x 1\n"
        "b y 1\n"
        "c z 1\n";
}
```

6.8 output

```
./EDistance < example10.txt
Edit distance = 5
A T 1
A A 0
C - 1
A A 0
G G 0
T G 1
T T 0
A - 1
C C 0
C A 1
Execution time is 2.7e-05 seconds
```

7 PS6: RandWriter

7.1 Discussion

In this project, we developed a class that implements a model of natural language using Markov chains. The core functionality revolves around analyzing a given text and generating new text based on the statistical properties of substrings of a specified length k. The key idea is that the probability of each character following a given substring (k-gram) is determined by its frequency of occurrence in the input text. By leveraging these frequencies, we are able to generate new sequences of characters that mimic the style and structure of the input text.

The implementation is built around the `RandWriter` class, which supports various functionalities such as:

- Calculating the frequency of specific k-grams.
- Determining the frequency of characters following a k-gram.
- Randomly selecting characters based on their frequency in the given k-gram context.
- Generating a string of a specified length by repeatedly appending characters predicted by the model.

Additionally, the class uses a fixed random seed for deterministic behavior, which ensures reproducibility of the generated text.

7.2 What I accomplished

- Implemented the `RandWriter` class to build and manipulate k-gram models.
- Built a frequency map to store the frequency of characters following every possible k-gram in the input text.
- Developed functionality to generate new text sequences based on the trained k-gram model.
- Wrote test cases to validate the core functionalities such as frequency calculation, random character selection, and text generation.
- Created a command-line tool to interact with the program, where users can specify values for k and the desired length of the generated text.

7.3 What I already knew

Before this project, I had a solid understanding of the following:

- The basics of Markov chains and their application in modeling probabilistic systems.
- The use of random number generation and uniform distributions in C++.
- How to build and manipulate data structures like `std::map` to store and retrieve values efficiently.
- Basic C++ syntax, including exception handling and class-based object-oriented programming.

7.4 What I learned

- Gained insights into linguistic patterns and how statistical models can capture them using Markov chains.
- Expanded knowledge of C++ algorithms, including efficient ways to calculate frequencies and generate text.
- Learned how to handle edge cases such as small input texts, invalid k-grams, and zero-order models.
- Improved understanding of memory management and the importance of using deterministic random number generation for reproducibility.

7.5 Challenges

During the implementation of the `RandWriter` class and its associated functionalities, I faced several challenges:

- Ensuring that the k-gram frequencies were correctly calculated, especially when handling wrap-around in the text.
- Dealing with edge cases, such as when the input text length is less than k or generating sequences of length smaller than k.
- Managing exceptions and ensuring that invalid arguments (e.g., incorrect k-grams or invalid text) were properly handled.
- Optimizing the random selection of characters from the frequency map to make the generation process efficient.

7.6 Key Algorithms, Data Structures, and OO Designs

Several key algorithms, data structures, and object-oriented design principles were essential to the implementation of this project:

- **Markov Chain Modeling**: The core of the text generation model is based on Markov chains. In a Markov chain, the probability of the next character depends only on the previous k-gram. This design captures the statistical properties of the text, allowing the model to generate realistic sequences that resemble the input text.

- **Frequency Map**: A `std::map` is used to store the frequencies of characters following every possible k-gram in the input text. This structure enables efficient retrieval of frequency data, supporting fast lookups for both the k-gram and the associated characters, which is crucial for random character selection.

- **Random Character Selection**: The random selection of the next character is based on the frequency of characters following a given k-gram. This is done by creating a weighted random distribution where characters with higher frequencies are more likely to be selected. A std::uniform$_{i}nt_{d}istributionisusedtogeneratearandomnumber, whichisthenmappedtoachar$

- **Wrap-Around Handling**: To account for the circular nature of the text, the implementation handles wrap-around when generating k-grams near the end of the text. This ensures that k-grams at the boundary of the text can correctly reference characters at the beginning of the text.

- **Object-Oriented Design**: The `RandWriter` class follows object-oriented principles by encapsulating the logic for building the frequency map, generating new text, and handling random number generation. The class provides well-defined methods such as `freq`, `kRand`, and `generate` that allow users to interact with the model in a modular and reusable way.

- **Error Handling**: Robust error handling is implemented throughout the code to ensure that invalid inputs (e.g., incorrect k-grams or invalid text lengths) are caught and appropriate exceptions are thrown. This guarantees that the program operates safely and predictably even in edge cases.

7.7 Codes

7.7.1 makefile

```
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -pedantic -Werror
TEST_LIBS = -lboost_unit_test_framework

all: TextWriter test TextWriter.a

TextWriter: TextWriter.o RandWriter.o
	$(CXX) $(CXXFLAGS) -o $@ $^

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

test: test.o RandWriter.o
	$(CXX) $(CXXFLAGS) -o $@ $^ $(TEST_LIBS)

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

lint:
	cpplint *.cpp *.hpp

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

.PHONY: all lint clean
```

7.7.2 RandWriter.hpp

```
// Copyright 2024 <Manasvi Boineypally>
#ifndef RANDWRITER_HPP
#define RANDWRITER_HPP

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

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);
```

```cpp
    std::string generate(const std::string& kgram, size_t L);

 private:
    std::string text;
    size_t k;
    std::map<std::string, std::map<char, int>> frequencyMap;
    std::string alphabet;
    std::mt19937 gen;

    void buildFrequencyMap();
};

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

#endif  // RANDWRITER_HPP
```

7.7.3 RandWriter.cpp

```cpp
// Copyright 2024 <Manasvi Boineypally>
#include "RandWriter.hpp"
#include <random>
#include <stdexcept>
#include <algorithm>
#include <string>

RandWriter::RandWriter(const std::string& text, size_t k) : text(text), k(k) {
    if (text.length() < k) {
        throw std::invalid_argument("Text length must be at least k.");
    }
    buildFrequencyMap();
    // Use a fixed seed for deterministic behavior
    gen = std::mt19937(12345);  // Fixed seed
}

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

int RandWriter::freq(const std::string& kgram) const {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram length must be equal to k.");
    }
    if (k == 0 && kgram.empty()) {
        return text.length();   // For k=0, return the length of the text
    }
    auto it = frequencyMap.find(kgram);
    int total = 0;
    if (it != frequencyMap.end()) {
        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 length must be equal to k.");
    }
    auto it = frequencyMap.find(kgram);
    if (it != frequencyMap.end()) {
        auto charIt = it->second.find(c);
        return charIt != it->second.end() ? charIt->second : 0;
    }
    return 0;
}

char RandWriter::kRand(const std::string& kgram) {
    if (kgram.length() != k) {
        throw std::invalid_argument("kgram length must be equal to k.");
    }
    auto it = frequencyMap.find(kgram);
    if (it == frequencyMap.end()) {
        throw std::invalid_argument("No such kgram found.");
    }

    int totalFreq = 0;
    for (const auto& pair : it->second) {
        totalFreq += pair.second;
    }

    std::uniform_int_distribution<> dis(1, totalFreq);
    int randVal = dis(gen);

    // Using a lambda expression to find the selected character
    auto findChar = [&randVal](const auto& pair) {
```

```cpp
        randVal -= pair.second;
        return randVal <= 0;
    };

    auto selectedPair = std::find_if(it->second.begin(), it->second.end(), findChar);
    if (selectedPair != it->second.end()) {
        return selectedPair->first;
    }

    throw std::runtime_error("Random selection failed.");
}

std::string RandWriter::generate(const std::string& kgram, size_t L) {
    if (kgram.length() != k || L < k) {
        throw std::invalid_argument("Invalid arguments for generation.");
    }

    std::string result = kgram;

    while (result.length() < L) {
        char nextChar = kRand(result.substr(result.length() - k));
        result += nextChar;
    }

    return result;
}

void RandWriter::buildFrequencyMap() {
    size_t n = text.length();

    // Build frequency map considering wrap-around
    for (size_t i = 0; i < n; ++i) {
        std::string kgram;

        for (size_t j = 0; j < k; ++j) {
            kgram += text[(i + j) % n];
        }

        char nextChar = text[(i + k) % n];

        frequencyMap[kgram][nextChar]++;

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


    if (k == 0 /&& i == 0) {
            for (char c : text) {
                frequencyMap[""][c]++;
            }

            break;
        }
    }
}
```

7.7.4 TextWriter.cpp

```
// Copyright 2024 <Manasvi Boineypally>
#include <iostream>
#include <string>
#include <stdexcept>
#include "RandWriter.hpp"

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

    size_t k, L;
    try {
        k = std::stoul(argv[1]);
        L = std::stoul(argv[2]);
    } catch (const std::exception& e) {
        std::cerr << "Invalid argument: " << e.what() << std
           ::endl;
        return 1;
    }

    std::string input_text;
    std::string line;
    while (std::getline(std::cin, line)) {
        input_text += line;
    }

    if (input_text.length() < k) {
        std::cerr << "Input text length must be at least k."
           << std::endl;
        return 1;
    }

    if (L < k) {
        std::cerr << "L must be greater than or equal to k."
           << std::endl;
        return 1;
    }

    try {
        RandWriter model(input_text, k);
        std::string kgram = input_text.substr(0, k);
        std::string generated = model.generate(kgram, L);
        std::cout << generated << std::endl;
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}
```

7.7.5 test.cpp

```

// Copyright 2024 <Manasvi Boineypally>
#define BOOST_TEST_MODULE RandWriterTest
#include <boost/test/included/unit_test.hpp>
#include "RandWriter.hpp"

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

BOOST_AUTO_TEST_CASE(order_k_test) {
    RandWriter rw("abcdeabcde", 3);
    BOOST_REQUIRE_EQUAL(rw.orderK(), 3);
}

BOOST_AUTO_TEST_CASE(freq_test) {
    RandWriter rw("abcdeabcde", 2);
    BOOST_REQUIRE_EQUAL(rw.freq("ab"), 2);
    BOOST_REQUIRE_EQUAL(rw.freq("bc"), 2);
    BOOST_REQUIRE_EQUAL(rw.freq("de"), 2);
    BOOST_REQUIRE_EQUAL(rw.freq("ea"), 2);
    BOOST_REQUIRE_EQUAL(rw.freq("ab", ’c’), 2);
    BOOST_REQUIRE_EQUAL(rw.freq("de", ’a’), 2);
    BOOST_REQUIRE_EQUAL(rw.freq("de", ’x’), 0);
    BOOST_REQUIRE_THROW(rw.freq("a"), std::invalid_argument);
    BOOST_REQUIRE_THROW(rw.freq("abc"), std::invalid_argument
       );
}

BOOST_AUTO_TEST_CASE(krand_test) {
    RandWriter rw("abcdeabcde", 2);
    BOOST_REQUIRE_NO_THROW(rw.kRand("ab"));
    BOOST_REQUIRE_THROW(rw.kRand("xy"), std::invalid_argument
       );
    BOOST_REQUIRE_THROW(rw.kRand("a"), std::invalid_argument)
       ;
}

BOOST_AUTO_TEST_CASE(generate_test) {
    RandWriter rw("abcdeabcde", 2);
    std::string generated = rw.generate("ab", 10);
    BOOST_REQUIRE_EQUAL(generated.length(), 10);
    BOOST_REQUIRE_EQUAL(generated.substr(0, 2), "ab");
    BOOST_REQUIRE_THROW(rw.generate("xy", 10), std::
       invalid_argument);
    BOOST_REQUIRE_THROW(rw.generate("a", 10), std::
       invalid_argument);
    BOOST_REQUIRE_THROW(rw.generate("ab", 1), std::
       invalid_argument);
}

BOOST_AUTO_TEST_CASE(zero_order_test) {
    RandWriter rw("abcdeabcde", 0);
    BOOST_REQUIRE_EQUAL(rw.freq(""), 10);
    BOOST_REQUIRE_EQUAL(rw.freq("", 'a'), 3);
    BOOST_REQUIRE_EQUAL(rw.freq("", 'x'), 0);
    BOOST_REQUIRE_NO_THROW(rw.kRand(""));
    std::string generated = rw.generate("", 10);
    BOOST_REQUIRE_EQUAL(generated.length(), 10);
}
```

7.8 output

```
    ./TextWriter 3 2000 < romeo.txt
What here shall shall misadvents' traffic of the their of
   star-crossed piteous overth their patient earful parent
   grudge break to new mutinuance of their of their death
   with their Verona, where we lay ours' traffic of their
   death-marked piteousehA passage break to new mutinuance of
    the fearful patientured piteouseholds, bury their Verona,
    where shall misadvents' trafDoth bury their Verona, which
    , if yese miss, our scene),
```

8 PS7: Kronos Log Parsing

8.1 Discussion

This assignment primarily focused on utilizing regular expressions to process log files efficiently. The task was to analyze server startup and completion logs to calculate the time it took for each server to boot. We used C++ to implement this functionality, leveraging the power of regular expressions to match specific log patterns, extract timestamps, and compute time differences. This involved integrating the Boost C++ libraries to handle time calculations accurately, allowing for detailed reports on device startup processes.

8.2 What I accomplished

In this project, I successfully implemented a log file parser that detects when servers start and complete their boot process by matching log entries with regular expressions. I used Boost's date-time library to calculate the elapsed time between the initiation and completion of each server boot, and generated a detailed summary report listing the results. This project provided a practical use case for regular expressions and time duration calculation, which are essential for processing and analyzing log data.

8.3 What I already knew

Before starting this project, I had a basic understanding of regular expressions and how they can be used for pattern matching in strings. Additionally, I had experience with file input/output in C++, as well as using the Boost C++ libraries for handling date-time operations. This background knowledge helped me quickly grasp how to extract and process the necessary data from log files.

8.4 What I learned

- Gained understanding of regular expressions and their practical applications in C++.
- Learned how to match patterns in log files to detect specific events (such as server startup and completion).
- Improved skills in handling time-related operations using Boost's date-time library.

- Developed an understanding of how to generate formatted summary reports based on parsed log data.

8.5 Challenges

One of the main challenges in this project was handling edge cases, such as incomplete startup processes or missing timestamps in the log file. These issues required careful checks and error handling. Additionally, working with Boost's time library to compute the time difference between two timestamps required a deeper understanding of the library's API. Debugging the regular expression patterns to ensure that they correctly captured the initiation and completion times was also a time-consuming task.

8.6 Key Algorithms, Data Structures, and OO Designs

The implementation of the Kronos Log Parsing project involved several critical algorithms, data structures, and object-oriented design concepts:

- **Regular Expressions**: The project heavily relied on regular expressions to parse log files. The regular expressions were used to match specific patterns in the log entries (e.g., server start and completion events). This allowed the efficient extraction of timestamps and other relevant data. Regular expressions were also used to validate log entries and ensure the proper structure for further processing.

- **Boost Date-Time Library**: For accurate time calculations, the Boost Date-Time library was used. Specifically, the boost::posix$_{t}ime :: ptimeclasswasemployedtoh$

- **Object-Oriented Design**: The program follows object-oriented principles by encapsulating data and functionality into the StartupEntry structure. Each StartupEntry object holds the necessary information related to a single server boot process, such as the initiation time, completion time, and elapsed time. This modular design made the code easier to maintain and extend.

- **Error Handling**: Robust error handling was implemented to account for potential edge cases in the log file, such as incomplete boot processes or missing timestamps. The program checks for these anomalies and ensures that the log parsing continues without crashing, generating appropriate error messages when necessary.

- **File I/O**: File input and output operations were central to the project. The log file was read line-by-line, and the parsed data was written to a summary report file. This process involved handling both reading and writing operations efficiently to ensure correct parsing of potentially large log files.

8.7 Codes

8.7.1 makefile

```
# Compiler and flags
CXX = g++
CXXFLAGS = -std=c++17 -Wall -Wextra -Werror -O2 -pedantic

# Targets
SRC = ps7.cpp
OBJ = $(SRC:.cpp=.o)
EXE = ps7

# Default target
all: $(EXE)

# Build the executable
$(EXE): $(OBJ)
        $(CXX) $(CXXFLAGS) -o $@ $^

# Compile object files
%.o: %.cpp
        \$(CXX) $(CXXFLAGS) -c $< -o $@

# Linting (optional, use a tool like clang-tidy if available)
lint:
        clang-tidy $(SRC)

# Clean up generated files
clean:
        rm -f $(OBJ) $(EXE) *.rpt
```

8.7.2 ps7.cpp

```
// copyright 2024 <manasvi boineypally>
#include <iostream>
#include <fstream>
#include <regex>
#include <string>
#include <vector>
```

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

struct StartupEntry {
    int beginRow;
    int finishRow;
    std::string initiationTime;
    std::string completionTime;
    int64_t elapsedTime;
    bool isFinished;

    StartupEntry() : beginRow(0), finishRow(0), elapsedTime
       (0), isFinished(false) {}
};

std::vector<StartupEntry> processLogFile(const std::string&
   inputFile, int& lineCount) {
    std::ifstream logStream(inputFile);
    std::string currentLine;
    std::vector<StartupEntry> startupList;
    StartupEntry currentStartup;
    int rowNumber = 0;

    std::regex initiationPattern(R"(\(log\.c\.166\) server
       started)");
    std::regex completionPattern(R"(oejs\.AbstractConnector:
       Started SelectChannelConnector@0\.0\.0\.0:9080)");
    std::regex timePattern(R"((\d{4}-\d{2}-\d{2} \d{2}:\d
       {2}:\d{2}))");

    while (std::getline(logStream, currentLine)) {
        rowNumber++;
        std::smatch timeMatch;

        if (std::regex_search(currentLine, timeMatch,
           timePattern)) {
            std::string timeStamp = timeMatch[1];
            if (std::regex_search(currentLine,
               initiationPattern)) {
                if (currentStartup.beginRow != 0) {
                    startupList.push_back(currentStartup);
                }
                currentStartup = StartupEntry();
                currentStartup.beginRow = rowNumber;
                currentStartup.initiationTime = timeStamp;
            } else if (std::regex_search(currentLine,
               completionPattern)) {
                currentStartup.finishRow = rowNumber;
                currentStartup.completionTime = timeStamp;
                currentStartup.isFinished = true;

boost::posix_time::ptime startTime
= boost::posix_time::time_from_string(currentStartup.
   initiationTime);
boost::posix_time::ptime endTime =
 boost::posix_time::time_from_string(currentStartup.
    completionTime);
boost::posix_time::time_duration timeDiff = endTime -
   startTime;

                currentStartup.elapsedTime = timeDiff.
                   total_milliseconds();
                startupList.push_back(currentStartup);
                currentStartup = StartupEntry();
            }
        }
    }

    if (currentStartup.beginRow != 0) {
        startupList.push_back(currentStartup);
    }

    lineCount = rowNumber;
    return startupList;
}

void createSummary(const std::string& inputFile,
const std::vector<StartupEntry>& startupList, int lineCount)
   {
    std::ofstream summaryFile(inputFile + ".rpt");

    int totalStartups = startupList.size();
    int finishedStartups = 0;

    for (const auto& startup : startupList) {
        if (startup.isFinished) {
            finishedStartups++;
        }
    }

    summaryFile << "Device Boot Report\n\n";
    summaryFile << "InTouch log file: " << inputFile << "\n";
    summaryFile << "Lines Scanned: " << lineCount << "\n\n";
    summaryFile << "Device boot count: initiated = " <<
       totalStartups
     << ", completed: " << finishedStartups << "\n\n\n";

    for (const auto& startup : startupList) {
        summaryFile << "=== Device boot ===\n";
```

```cpp
summaryFile << startup.beginRow << "(" << inputFile << "): "
<< startup.initiationTime << " Boot Start\n";

        if (startup.isFinished) {
            summaryFile << startup.finishRow << "(" << inputFile << "): "
             << startup.completionTime << " Boot Completed\n";
            summaryFile << "\tBoot Time: " << startup.elapsedTime << "ms \n\n";
        } else {
            summaryFile << "**** Incomplete boot **** \n\n";
        }
    }

    summaryFile.close();
}

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

    std::string inputFile = argv[1];
    int lineCount = 0;
    std::vector<StartupEntry> startupList = processLogFile(inputFile, lineCount);
    createSummary(inputFile, startupList, lineCount);

    return 0;
}
```

8.8 output

```


Device Boot Report

InTouch log file: device1_intouch.log
Lines Scanned: 443838

Device boot count: initiated = 6, completed: 6


=== Device boot ===
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
```

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

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

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

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

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

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