COMPUTING IV Sec 202:Project Portfolio

Ashish Kosana

Fall 2024

Contents

1 PS0: Hello SFML **4**
- 1.1 Overview . . . 4
- 1.2 What I Accomplished . . . 4
- 1.3 What I Learned . . . 4
- 1.4 Discussion of Key Algorithms & Data structures . . . 4
- 1.5 What I Already Knew . . . 5
- 1.6 Challenges . . . 5
- 1.7 Codes . . . 5
  - 1.7.1 Makefile . . . 5
  - 1.7.2 main.cpp . . . 5
  - 1.7.3 screenshot . . . 7

2 PS1: LFSR & PhotoMagic **8**
- 2.1 Overview . . . 8
- 2.2 What I Accomplished . . . 8
- 2.3 What I Learned . . . 8
- 2.4 Discussion of Key Algorithms . . . 8
- 2.5 What I Already Knew . . . 9
- 2.6 Challenges . . . 9
- 2.7 Codes . . . 9
  - 2.7.1 Makefile . . . 9
  - 2.7.2 main.cpp . . . 10
  - 2.7.3 FibLFSR.hpp . . . 12
  - 2.7.4 FibLFSR.cpp . . . 12
  - 2.7.5 PhotoMagic.hpp . . . 13
  - 2.7.6 PhotoMagic.cpp . . . 14
  - 2.7.7 test.cpp . . . 14
  - 2.7.8 screenshot . . . 16

3 PS2: Pentaflake **17**
- 3.1 Overview . . . 17
- 3.2 What I Accomplished . . . 17
- 3.3 What I Learned . . . 17
- 3.4 Discussion of Key Algorithms . . . 17
- 3.5 What I Already Knew . . . 18

3.6 Challenges . . . 18
3.7 Codes . . . 18
3.7.1 Makefile . . . 18
3.7.2 main.cpp . . . 19
3.7.3 penta.hpp . . . 19
3.7.4 penta.cpp . . . 20
3.7.5 Screenshot . . . 22

4 PS3 :Static N-body Simulation & Dynamic N-body Simulation 23
4.1 Overview . . . 23
4.2 What I Accomplished . . . 23
4.3 What I Learned . . . 23
4.4 Discussion of Key Algorithms . . . 23
4.5 What I Already Knew . . . 24
4.6 Challenges . . . 24
4.7 Codes . . . 24
4.7.1 Makefile . . . 24
4.7.2 main.cpp . . . 25
4.7.3 CelestialBody.hpp . . . 30
4.7.4 CelestialBody.cpp . . . 31
4.7.5 Universe.hpp . . . 33
4.7.6 Universe.cpp . . . 34
4.7.7 test.cpp . . . 36
4.7.8 Screenshot . . . 40

5 PS4: Sokoban UI & Sokoban 41
5.1 Overview . . . 41
5.2 What I Accomplished . . . 41
5.3 What I Learned . . . 41
5.4 Discussion of Key Algorithms . . . 41
5.5 What I Already Knew . . . 41
5.6 Challenges . . . 42
5.7 Codes . . . 42
5.7.1 Makefile . . . 42
5.7.2 main.cpp . . . 43
5.7.3 Sokoban.hpp . . . 47
5.7.4 Sokoban.cpp . . . 49
5.7.5 test.cpp . . . 54
5.7.6 Screenshot . . . 56

6 PS5: DNA Alignment 57
6.1 Overview . . . 57
6.2 What I Accomplished . . . 57
6.3 What I Learned . . . 57
6.4 Discussion of Key Algorithms . . . 57

6.5 What I Already Knew . . . 58
6.6 Challenges . . . 58
6.7 Codes . . . 58
6.7.1 Makefile . . . 58
6.7.2 main.cpp . . . 59
6.7.3 EDistance.hpp . . . 60
6.7.4 EDistance.cpp . . . 61
6.7.5 test.cpp . . . 62
6.7.6 Input: . . . 64
6.7.7 Output: . . . 64

7 PS6:RandWriter 65
7.1 Overview . . . 65
7.2 What I Accomplished . . . 65
7.3 What I Learned . . . 65
7.4 Discussion of Key Algorithms & Data Structures . . . 65
7.5 What I Already knew . . . 65
7.6 Challenges . . . 66
7.7 Codes . . . 66
7.7.1 Makefile . . . 66
7.7.2 RandWriter.hpp . . . 66
7.7.3 RandWriter.cpp . . . 67
7.7.4 TextWriter.cpp . . . 71
7.7.5 test.cpp . . . 72
7.7.6 Input: . . . 73
7.7.7 Output: . . . 73

8 PS7: Kronos Log Parsing 74
8.1 Overview . . . 74
8.2 What I Accomplished . . . 74
8.3 What I Learned . . . 74
8.4 Discussion of Key Algorithms . . . 74
8.5 What I Already Knew . . . 75
8.6 Challenges . . . 75
8.7 Codes . . . 75
8.7.1 Makefile . . . 75
8.7.2 ps7.cpp . . . 76
8.7.3 Output: . . . 79

Time to Complete: 13 Hours

1 PS0: Hello SFML

1.1 Overview

I have made a simple program using SFML that shows a moving sprite and a rectangle on the sfml window. The sprite moves when I press keyboard keys, and the background color changes to four different colors for four different keys in the keyboard. This project helped me understand how to create basic graphics and handle user input.

1.2 What I Accomplished

1. I have created a program where the sprite moves smoothly and responds to key presses.
2. I have implemented the sprite movement using delta time calculations, where the sprite moves different sides for different keys.
3. I have added background colour changing feature such that it creates fun for the user.

1.3 What I Learned

1. I have learned how to create a window using sfml and load shapes and sprites into it.
2. I have understood how to detect key presses and make the program respond to them.
3. I have learned how to implement graphics in c++ using sfml

1.4 Discussion of Key Algorithms & Data structures

1. **Data structures:**
==> sf::Sprite for the image that moves.
==> sf::RectangleShape for a rectangle that stays on the screen.
==> sf::Vector2f to handle positions and movement.
==> sf::Clock and sf::Time to measure time and control movement.
2. **Key Algorithms:**
==> Movement: The sprite moves in different directions when I press W, A, S, D, or the arrow keys.
==> Delta Time: I have used time to make sure the sprite moves smoothly and with the same speed.
==> Color Change: The background color changes depending on which key is pressed.
==>prite Movement:Ensure smooth movement using a frame rate-independent formula: distance=movementSpeed×deltaTime.
Object Oriented Designs:
==>Abstraction:
I have used abstractions to focus on implementing higher-level functionality like controlling the sprite's movement and changing the background color.
==>Interaction between objects:
I have learned that objects can interact with each other. For example, sf::RenderWindow interacts with sf::Sprite to display the sprite, while sf::Keyboard interacts with sf::Sprite to move it

1.5 What I Already Knew

==>I am familier with c++ language.

1.6 Challenges

==>Learning how to use SFML for the first time, such as creating a window, drawing shapes, and loading images, was a bit challenging.
==>As i was learning sfml for the first time Combining graphics programming with C++ was tricky.

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
DEPS =
OBJECTS = main.o
PROGRAM = sfml-app

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

1.7.2 main.cpp

```
#include<iostream>
#include <SFML/Graphics.hpp>
int main() {
    sf::RenderWindow window(sf::VideoMode(1000, 800), "Ashish's Moving
   Sprite");
```

```
    sf::RectangleShape rec(sf::Vector2f(100, 120));
    rec.setFillColor(sf::Color::Red);
    rec.setPosition(50.f, 55.f);
    window.setFramerateLimit(60);
    sf::Texture spriteTexture;
    if (!spriteTexture.loadFromFile("Sprite.png")) {
        std::cerr << "image loading failed!" << std::endl;
        return -1;
    }
    sf::Sprite sprite;
    sprite.setTexture(spriteTexture);
    sprite.setScale(0.5f, 0.5f);
    sf::Vector2u windowSize = window.getSize();
    sf::FloatRect spriteBounds = sprite.getGlobalBounds();
    float posX = (windowSize.x - spriteBounds.width) / 3.0f;
    float posY = (windowSize.y - spriteBounds.height) / -20.0f;
    sprite.setPosition(posX, posY);
    sprite.setScale(0.2f, 0.2f);
    float movementSpeed = 200.0f;
    sf::Clock frameClock;
    sf::Color backgroundColor = sf::Color::White;
    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if(event.type == sf::Event::Closed) {
                window.close();
            }
        }
        sf::Time elapsedTime = frameClock.restart();
        float deltaTime = elapsedTime.asSeconds();
        backgroundColor = sf::Color::White;
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
            sprite.move(-movementSpeed * deltaTime, 0);
            backgroundColor = sf::Color::Magenta;
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
            sprite.move(movementSpeed * deltaTime, 0);
            backgroundColor = sf::Color::Green;
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up) ||
            sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
            sprite.move(0, -movementSpeed * deltaTime);
            backgroundColor = sf::Color::Cyan;
```

```
50          }
51          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down) ||
52              sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
53              sprite.move(0, movementSpeed * deltaTime);
54              backgroundColor = sf::Color::Yellow;
55          }
56          window.clear(backgroundColor);
57          window.draw(sprite);
58          window.draw(rec);
59          window.display();
60      }
61      return 0;
62  }
```

1.7.3 screenshot

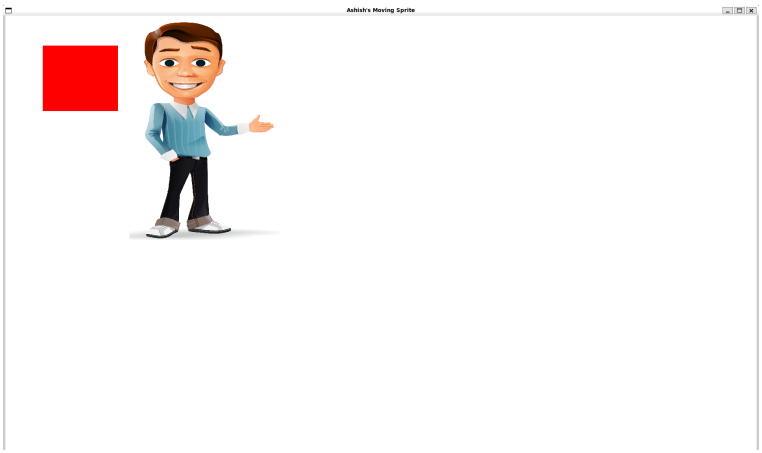

Figure 1: My sfml window

2 PS1: LFSR & PhotoMagic

2.1 Overview

In this project, I have implemented a Fibonacci Linear Feedback Shift Register (LFSR) that simulates pseudo-random bit generation. This LFSR uses specific tap positions to calculate new bits using XOR operations.
I have Extended the use of the LFSR to encrypt and decrypt images by changing the pixel values using the pseudo-random sequence generated by the LFSR.
==>The encryption process uses the pseudo-random bit sequence generated by the LFSR to alter the red, green, and blue (RGB) values of each pixel in the input image.
==>The Decryption process also uses the same LFSR configuration to regenerate the identical pseudo-random sequence used in encryption process.

2.2 What I Accomplished

1. I have implemented FibLFSR class for simulating the behaviour of LFSR.
2. I have implemented the step function which which moves the bit position one to left and adds new bit.
3. I have successfully implemented encryption and decryption for images using LFSR.
4. I have ensured that the decrypted image matches the original image.

2.3 What I Learned

==> I have learned that an LFSR works by shifting the bits to the left and replacing the empty bit with the result of XORing specific tap bits.
==>I have learned how to write and run unit tests using the Boost framework
==>I have understood LFSR and how it generates pseudo-random sequences.
==>I have learned that how to change the image pixel values using C++ and SFML.

2.4 Discussion of Key Algorithms

1.Data structures:
i.) std::string:
==>I have used a string to represent the LFSR state. Each character in the string represents a bit either '0' or '1'.
ii.) sf::Image:
==>To handle the input and output images.
iii.)std::vector:
==>For processing pixel data.
2.Algorithms:
1. Step Function (step()):
==> It calculates the new bit by XORing the leftmost bit with the bits at the tap positions.
2.Image Encryption:
==>Pixel values were altered by performing XOR operations between pixel data and LFSR output.

2.5 What I Already Knew

==> I know basic XOR operations.
==>Basic C++ syntax, including classes, constructors, and functions.
Object Oriented Designs:
==>Classes and Objects: FibLFSR is the main class, and objects of this class are created to represent LFSRs.
==>Abstraction: The FibLFSR class abstracts the LFSR functionality.

2.6 Challenges

==>For the first time i have faced difficulty in understanding the FIBLFSR algorithm.
==>I have faced difficulty in Encrypting and Decrypting the image using the 16 bit seed.
==>I felt little bit difficult in xor'ing the pixels.

2.7 Codes

2.7.1 Makefile

```
CXX = g++
CXXFLAGS = -std=c++11 -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

```
1
2  // CopyRight 2024 Ashish kosana
3  #include <iostream>
4  #include <bitset>
5  #include "PhotoMagic.hpp"
6  #include "FibLFSR.hpp"
7  #include <SFML/Graphics.hpp>
8
9  // Converts alphanumeric key to a binary seed
10 std::string convertKeyToBinary(const std::string &key) {
11     std::string binarySeed;
12     for (char c : key) {
13         binarySeed += std::bitset<8>(c).to_string();
14     }
15     // Ensure the binary seed is exactly 16 bits long
16     if (binarySeed.size() > 16) {
17         binarySeed = binarySeed.substr(0, 16);
18     } else if (binarySeed.size() < 16) {
19         binarySeed.append(16 - binarySeed.size(), '0');
20     }
21     return binarySeed;
22 }
23
24 int main(int argc, char* argv[]) {
25     if (argc != 4) {
26         std::cerr << "Usage: " << argv[0]
27                   << " <input-file> <output-file> <LFSR-seed>" << std::endl;
28         return 1;
29     }
30
31     std::string inputFile = argv[1];
32     std::string outputFile = argv[2];
33     std::string seed = argv[3];
34
35     // Convert the alphanumeric seed to binary
36     std::string binarySeed = convertKeyToBinary(seed);
37
38     sf::Image inputImage;
39     if (!inputImage.loadFromFile(inputFile)) {
40         std::cerr << "Failed to load input image." << std::endl;
41         return 1;
42     }
43
```

```
    PhotoMagic::FibLFSR lfsr(binarySeed);
    sf::Image outputImage = inputImage;
    PhotoMagic::transform(outputImage, &lfsr);

    if (!outputImage.saveToFile(outputFile)) {
        std::cerr << "Failed to save output image." << std::endl;
        return 1;
    }

    sf::Texture inputTexture, outputTexture;
    inputTexture.loadFromImage(inputImage);
    outputTexture.loadFromImage(outputImage);

    sf::Sprite inputSprite(inputTexture), outputSprite(outputTexture);

    sf::Vector2u size = inputImage.getSize();
    sf::RenderWindow inputWindow(sf::VideoMode(size.x, size.y), "Original
   Image");
    sf::RenderWindow outputWindow(sf::VideoMode(size.x, size.y), "
   Transformed Image");

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

        inputWindow.clear();
        inputWindow.draw(inputSprite);
        inputWindow.display();

        outputWindow.clear();
        outputWindow.draw(outputSprite);
        outputWindow.display();
    }
    return 0;
}
```

2.7.3 FibLFSR.hpp

```cpp
// CopyRight 2024 Ashish kosana
#ifndef FIBLFSR_HPP
#define FIBLFSR_HPP

#include <string>
#include <iostream>

namespace PhotoMagic {

class FibLFSR {
 public:
explicit FibLFSR(std::string seed);
    int step();
    int generate(int k);
    std::string toString() const;

    friend std::ostream& operator<<(std::ostream& os, const FibLFSR& lfsr);
    friend std::istream& operator>>(std::istream& is, FibLFSR& lfsr);

 private:
    std::string register_state;
};

}  // namespace PhotoMagic

#endif  // FIBLFSR_HPP
```

2.7.4 FibLFSR.cpp

```cpp
#include <stdexcept>
#include "FibLFSR.hpp"

namespace PhotoMagic {

FibLFSR::FibLFSR(std::string seed) : register_state(seed) {
    if (seed.empty() || seed.find_first_not_of("01") != std::string::npos) {
        throw std::invalid_argument("Seed must be a non-empty binary string"
   );
    }
}

int FibLFSR::step() {
```

```
    int new_bit = (register_state[0] - '0') ^ (register_state[2] - '0') ^
                  (register_state[3] - '0') ^ (register_state[5] - '0');
    register_state = register_state.substr(1) + std::to_string(new_bit);
    return new_bit;
}

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

std::string FibLFSR::toString() const {
    return register_state;
}

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

std::istream& operator>>(std::istream& is, FibLFSR& lfsr) {
    std::string seed;
    is >> seed;
    lfsr = FibLFSR(seed);
    return is;
}

}  // namespace PhotoMagic
```

2.7.5 PhotoMagic.hpp

```
// CopyRight 2024 Ashish kosana
#ifndef PHOTOMAGIC_HPP
#define PHOTOMAGIC_HPP

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

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

```

```
#endif  // PHOTOMAGIC_HPP
```

2.7.6 PhotoMagic.cpp

```
// CopyRight 2024 Ashish kosana
#include "PhotoMagic.hpp"
namespace PhotoMagic {
void transform(sf::Image& image, FibLFSR* lfsr) {
        sf::Vector2u size = image.getSize();
        for (unsigned int y = 0; y < size.y; ++y) {
            for (unsigned int x = 0; x < size.x; ++x) {
                sf::Color pixel = image.getPixel(x, y);
                uint8_t r = pixel.r ^ lfsr->generate(8);
                uint8_t g = pixel.g ^ lfsr->generate(8);
                uint8_t b = pixel.b ^ lfsr->generate(8);
                image.setPixel(x, y, sf::Color(r, g, b, pixel.a));
}
}
}
}  // namespace PhotoMagic
```

2.7.7 test.cpp

```
// CopyRight [2024] Ashish Kosana
#include <iostream>
#include <string>
#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"
#include "PhotoMagic.hpp"

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

using PhotoMagic::FibLFSR;
using PhotoMagic::transform;

BOOST_AUTO_TEST_CASE(testInitialization) {
    FibLFSR l("1010101010101010");
    BOOST_REQUIRE_EQUAL(l.toString(), "1010101010101010");
}

BOOST_AUTO_TEST_CASE(testEmptySeed) {
```

```
21      BOOST_CHECK_THROW(FibLFSR l(""), std::invalid_argument);
22  }
23
24  BOOST_AUTO_TEST_CASE(testInvalidSeedCharacters) {
25      BOOST_CHECK_THROW(FibLFSR l("1010A01010101010"), std::invalid_argument);
26  }
27
28  BOOST_AUTO_TEST_CASE(testImageTransform) {
29      sf::Image image;
30      if (!image.loadFromFile("input.png")) {
31          BOOST_FAIL("Failed to load test image file 'input.png'");
32      }
33
34      sf::Image originalImage = image;
35
36      // Initialize LFSR with a seed and transform the image (encrypt)
37      FibLFSR lfsrEncrypt("1010101010101010");
38      transform(image, &lfsrEncrypt);
39
40      // Reinitialize LFSR with the same seed and transform the image again (
        decrypt)
41      FibLFSR lfsrDecrypt("1010101010101010");
42      transform(image, &lfsrDecrypt);
43
44      // Verify if each pixel in the decrypted image matches the original
45      sf::Vector2u size = image.getSize();
46      for (unsigned int x = 0; x < size.x; ++x) {
47          for (unsigned int y = 0; y < size.y; ++y) {
48              BOOST_REQUIRE(image.getPixel(x, y) == originalImage.getPixel(x,
        y));
49          }
50      }
51  }
```

2.7.8 screenshot

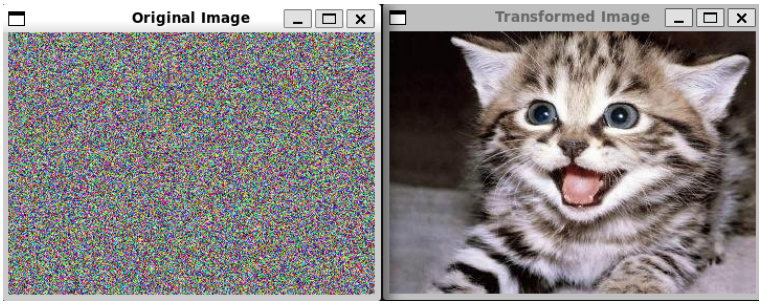

Figure 2: Encrypted Image

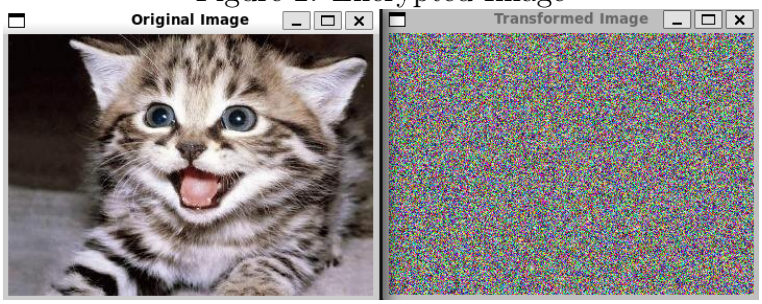

Figure 3: Decrypted Image

3 PS2: Pentaflake

3.1 Overview

I have created a program using SFML that generates a visual structure called the Pentaflake. This structure starts with a pentagon and adds smaller pentagons around it through a recursive process. Users can specify the size of the base pentagon and how many times to repeat this process. The Pentaflake rotates and changes colors based on its position and depth, making it visually engaging

3.2 What I Accomplished

==>I have implemented a recursive function that draws smaller pentagons around a central pentagon, creating the fractal effect.
==>Used SFML to create a window where the Pentaflake is displayed and updated in real-time.
==>I have included rotation functionality so that the entire pentaflake can spin, making it more engaging for viewers.

3.3 What I Learned

==>I have learned the concept of recursion and how it can be applied to generate complex shapes like pentagons.

3.4 Discussion of Key Algorithms

Data Structures:
1. std::vector<sf::ConvexShape>:
==>Used to store all the pentagons that make up the Pentaflake.
2. sf::Vector2f:
==>Used for handling positions of the pentagons in 2D space.
Algorithms:
1. Recursive Drawing:
==>The createPentaflake function calls itself to draw smaller pentagons around each existing one until reaching the specified depth.
2. Rotation Logic:
==>The entire Pentaflake is rotated around its center using SFML's transformation functions, adding movement to the display.
Object Oriented Designs:
==>Recurssion:
The method createPentaflake calls itself to generate smaller pentagons within the main pentagon.
==>Inheritance:
The class Pentaflake is inherited from sf::Drawable.

3.5 What I Already Knew

==>I was familiar with C++ programming basics, including syntax and data types.

3.6 Challenges

==>I have faced challenge in writing the recurssion logic for generating the small pentagons around the base pentagon.
==>I have faced little difficulty while linking the sfml and boost libraries.
==>I have faced difficult over calculating the dynamic color for each pentagon.

3.7 Codes

3.7.1 Makefile

```
CC = g++
CFLAGS = --std=c++20 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework
DEPS = penta.hpp
SOURCES = penta.cpp
PROGRAM = Penta

.PHONY: all clean lint

all: $(PROGRAM)

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

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

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

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

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

3.7.2 main.cpp

```
// copyright [2024] Ashish Kosana
#include <iostream>
#include <SFML/Graphics.hpp>
#include "penta.hpp"
int main(int argc, char* argv[]) {
if (argc != 3) {
std::cerr << "Usage: " << argv[0] <<
" <side_length> <recursion_depth>" << std::endl;
return 1;
}
double L = std::stod(argv[1]);
int N = std::stoi(argv[2]);
sf::RenderWindow window(sf::VideoMode(800, 800), "Pentaflake");
window.setFramerateLimit(60);
Pentaflake pentaflake(L, N);
while (window.isOpen()) {
sf::Event event;
while (window.pollEvent(event)) {
if (event.type == sf::Event::Closed)
window.close();
}
window.clear(sf::Color::Cyan);
window.draw(pentaflake);
window.display();
}
return 0;
}
```

3.7.3 penta.hpp

```
// copyright [2024] Ashish Kosana
#pragma once
#include <cmath>
#include <SFML/Graphics.hpp>

class Pentaflake : public sf::Drawable {
 public:
    Pentaflake(double sideLength, int depth);

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

```
    void createPentaflake(double sideLength, int depth, sf::Vector2f
    position, float rotation);

    std::vector<sf::ConvexShape> pentagons;
    const double PHI = (1 + std::sqrt(5)) / 2;
};
```

3.7.4 penta.cpp

```
#include "penta.hpp"
#include <ctime>

Pentaflake::Pentaflake(double sideLength, int depth) {
    sf::Vector2f center(400, 400);
    createPentaflake(sideLength, depth, center, 0);
}

void Pentaflake::draw(sf::RenderTarget& target, sf::RenderStates states)
    const {
    static float rotationAngle = 0.0f;
    rotationAngle += 100.5f;  // Adjust this value to change rotation speed

    sf::Transform rotation;
    rotation.rotate(rotationAngle, 400, 400);  // Rotate around the center
    (400, 400)

    for (const auto& pentagon : pentagons) {
        sf::RenderStates rotatedStates = states;
        rotatedStates.transform *= rotation;
        target.draw(pentagon, rotatedStates);
    }
}

void Pentaflake::createPentaflake(double sideLength,
 int depth, sf::Vector2f position, float rotation) {
    if (depth < 0) return;

    sf::ConvexShape pentagon;
    pentagon.setPointCount(5);
    int r = static_cast<int>((position.x / 800.0) * 255);
    int g = static_cast<int>((position.y / 800.0) * 255);
    int b = static_cast<int>((depth / 5.0) * 255);
    sf::Color fillColor(r, g, b, 128);
    pentagon.setFillColor(fillColor);
```

```
    pentagon.setOutlineColor(sf::Color::Black);
    pentagon.setOutlineThickness(1);

    double angle = 2 * M_PI / 5;
    double radius = sideLength / (2 * std::sin(M_PI / 5));

    for (int i = 0; i < 5; ++i) {
        double x = radius * std::cos(i * angle - M_PI / 2 + rotation);
        double y = radius * std::sin(i * angle - M_PI / 2 + rotation);
        pentagon.setPoint(i, sf::Vector2f(x, y));
    }

    pentagon.setPosition(position);
    pentagons.push_back(pentagon);

    if (depth > 0) {
        double newSideLength = sideLength / (1 + PHI);
        double newRadius = newSideLength / (2 * std::sin(M_PI / 5));
        double offset = radius + newRadius;

        for (int i = 0; i < 5; ++i) {
            double x = offset * std::cos(i * angle - M_PI / 2 + rotation);
            double y = offset * std::sin(i * angle - M_PI / 2 + rotation);
            sf::Vector2f newPosition(position.x + x, position.y + y);
            createPentaflake(newSideLength, depth - 1, newPosition, rotation
    + i * angle);
        }

        createPentaflake(newSideLength, depth - 1, position, rotation + M_PI
   );
    }
}
```

3.7.5 Screenshot

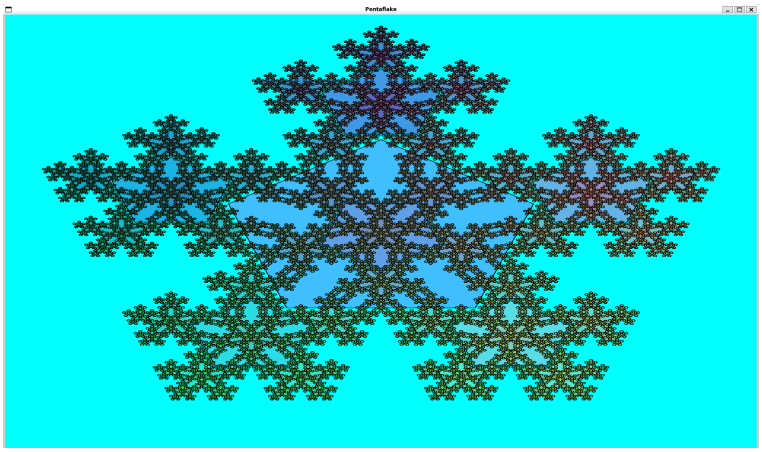

Figure 4: My Pentaflake

4 PS3 :Static N-body Simulation & Dynamic N-body Simulation

4.1 Overview

I developed an N-Body Simulation program using SFML that models the motion of celestial bodies in a universe. The project simulates gravitational interactions between particles, allowing users to visualize planetary movements based on Newton's laws of motion and gravitation

4.2 What I Accomplished

1.I have used SFML for graphical rendering, allowing dynamic visualization of planetary movements.
2. I have Developed a Universe class to manage multiple celestial bodies and their interactions.
3. I have Implemented a CelestialBody class to represent individual celestial objects with position, velocity, and mass properties.

4.3 What I Learned

==>I have learned how to implement graphics and animations using sfml.
==>I have learned some concepts of physics

4.4 Discussion of Key Algorithms

Data Structures:
1.sf::Vector2f:
for position and velocity representation
2.sf::Sprite and sf::Texture
for graphical rendering
Algorithms:
1. Gravitational Force Calculation:
The gravitational force between two celestial bodies is calculated using Newton's law of universal gravitation:

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

Where:

- F is the gravitational force between the two bodies,
- G is the gravitational constant ($6.67430 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$),
- m_1 and m_2 are the masses of the two bodies,
- r is the distance between the centers of the two bodies.

2. Position and Velocity Update:
==>By Applying Newton's laws of motion i have calculated the position and velocity.
Object Oriented Designs:
==>Inheritance:
The Universe class is inherited from sf::Drawable (SFML class), which is used to draw it onto an SFML window.
==>Friend Functions:
The operator» and operator« for Universe are implemented as friend functions.

4.5 What I Already Knew

==>I was familiar with basic physics concepts.
==>I have learned Use of sfml from previous projects.

4.6 Challenges

==>I have faced difficulty in implementing the correct gravitational force between each planets.
==>I have faced difficulty in Drawing and updating all the celestial bodies.
==>I have faced difficulty in setting the initial velocity and positions.

4.7 Codes

4.7.1 Makefile

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

all: NBody NBody.a test

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

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

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

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

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

```

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

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

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

.PHONY: all clean lint test
```

4.7.2 main.cpp

```
// CopyRight [2024] Ashish Kosana
#include <iostream>
#include <iomanip>
#include <sstream>
#include "Universe.hpp"
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

// Function to draw a digit using rectangles
void drawDigit(sf::RenderWindow& window, int digit, float x, float y, float
   size) {
    const bool segments[10][7] = {
        {1, 1, 1, 0, 1, 1, 1},  // 0
        {0, 0, 1, 0, 0, 1, 0},  // 1
        {1, 0, 1, 1, 1, 0, 1},  // 2
        {1, 0, 1, 1, 0, 1, 1},  // 3
        {0, 1, 1, 1, 0, 1, 0},  // 4
        {1, 1, 0, 1, 0, 1, 1},  // 5
        {1, 1, 0, 1, 1, 1, 1},  // 6
        {1, 0, 1, 0, 0, 1, 0},  // 7
        {1, 1, 1, 1, 1, 1, 1},  // 8
        {1, 1, 1, 1, 0, 1, 1}   // 9
    };

    sf::RectangleShape rect;
    rect.setFillColor(sf::Color::White);

    // Draw the segments
    if (segments[digit][0]) {
        rect.setSize(sf::Vector2f(size, size/5));
        rect.setPosition(x, y);
```

```
        window.draw(rect);
    }
    if (segments[digit][1]) {
        rect.setSize(sf::Vector2f(size/5, size));
        rect.setPosition(x, y);
        window.draw(rect);
    }
    if (segments[digit][2]) {
        rect.setSize(sf::Vector2f(size/5, size));
        rect.setPosition(x+size-size/5, y);
        window.draw(rect);
    }
    if (segments[digit][3]) {
        rect.setSize(sf::Vector2f(size, size/5));
        rect.setPosition(x, y+size-size/5);
        window.draw(rect);
    }
    if (segments[digit][4]) {
        rect.setSize(sf::Vector2f(size/5, size));
        rect.setPosition(x, y+size);
        window.draw(rect);
    }
    if (segments[digit][5]) {
        rect.setSize(sf::Vector2f(size/5, size));
        rect.setPosition(x+size-size/5, y+size);
        window.draw(rect);
    }
    if (segments[digit][6]) {
        rect.setSize(sf::Vector2f(size, size/5));
        rect.setPosition(x, y+size*2-size/5);
        window.draw(rect);
    }
}

// Function to draw "sec" using rectangles
void drawSec(sf::RenderWindow& window, float x, float y, float size) {
    sf::RectangleShape rect;
    rect.setFillColor(sf::Color::White);

    // Draw 's'
    rect.setSize(sf::Vector2f(size, size/5));
    rect.setPosition(x, y);
    window.draw(rect);
    rect.setPosition(x, y + size/2);
    window.draw(rect);
```

```
76      rect.setPosition(x, y + size - size/5);
77      window.draw(rect);
78      rect.setSize(sf::Vector2f(size/5, size/2));
79      rect.setPosition(x, y);
80      window.draw(rect);
81      rect.setPosition(x + size - size/5, y + size/2);
82      window.draw(rect);
83
84      // Draw 'e'
85      x += size * 1.2;
86      rect.setSize(sf::Vector2f(size, size/5));
87      rect.setPosition(x, y);
88      window.draw(rect);
89      rect.setPosition(x, y + size/2);
90      window.draw(rect);
91      rect.setPosition(x, y + size - size/5);
92      window.draw(rect);
93      rect.setSize(sf::Vector2f(size/5, size/2 + size/5));
94      rect.setPosition(x, y);
95      window.draw(rect);
96      rect.setSize(sf::Vector2f(size, size/5));
97      rect.setPosition(x, y + size/2);
98      window.draw(rect);
99
100     // Draw 'c'
101     x += size * 1.2;
102     rect.setSize(sf::Vector2f(size, size/5));
103     rect.setPosition(x, y);
104     window.draw(rect);
105     rect.setPosition(x, y + size - size/5);
106     window.draw(rect);
107     rect.setSize(sf::Vector2f(size/5, size));
108     rect.setPosition(x, y);
109     window.draw(rect);
110 }
111
112 int main(int argc, char* argv[]) {
113     if (argc != 3) {
114         std::cerr << "Usage: " << argv[0] << " <T> <dt>" << std::endl;
115         return 1;
116     }
117
118     double T = std::stod(argv[1]);
119     double dt = std::stod(argv[2]);
120
```

```
121     NB::Universe universe;
122     std::cin >> universe;
123 
124     // Create render window
125     sf::RenderWindow window(sf::VideoMode(800, 800), "ASHISH'S SOLAR SYSTEM"
        );
126 
127     // Set vertical sync enabled (no return value check, since it's void)
128     window.setVerticalSyncEnabled(true);
129 
130     double viewSize = universe.getRadius() * 2.5e-9;
131     sf::View universeView(sf::Vector2f(0, 0), sf::Vector2f(static_cast<float
        >(viewSize),
132                           static_cast<float>(viewSize)));
133     sf::View defaultView = window.getDefaultView();
134 
135     // Load background texture
136     sf::Texture backgroundTexture;
137     if (!backgroundTexture.loadFromFile("starfield.jpg")) {
138         std::cerr << "Failed to load background image" << std::endl;
139         return 1;
140     }
141     sf::Sprite background(backgroundTexture);
142 
143     // Load and play music
144     sf::Music music;
145     if (!music.openFromFile("2001.wav")) {
146         std::cerr << "Failed to load music file" << std::endl;
147         return 1;
148     }
149     music.setLoop(true);
150     music.play();
151     std::cout << "Music started playing." << std::endl;
152 
153     // Scale and position the background
154     background.setScale(
155         static_cast<float>(viewSize) / backgroundTexture.getSize().x,
156         static_cast<float>(viewSize) / backgroundTexture.getSize().y);
157     background.setPosition(static_cast<float>(-viewSize/2), static_cast<
        float>(-viewSize/2));
158 
159     // Setup for displaying elapsed time using shapes
160     sf::RectangleShape timeBox(sf::Vector2f(150, 50));
161     timeBox.setFillColor(sf::Color(0, 0, 0, 128));
162     timeBox.setPosition(defaultView.getSize().x - 160, 10);
```

```
163
164     sf::Clock clock;
165     double elapsedTime = 0.0;
166     bool firstDraw = true;
167
168     // Main window loop
169     while (window.isOpen() && elapsedTime < T) {
170         sf::Event event;
171         while (window.pollEvent(event)) {
172             if (event.type == sf::Event::Closed) {
173                 window.close();
174             }
175             if (event.type == sf::Event::KeyPressed && event.key.code == sf
    ::Keyboard::S) {
176                 if (music.getStatus() == sf::Music::Playing) {
177                     music.pause();
178                     std::cout << "Music paused" << std::endl;
179                 } else {
180                     music.play();
181                     std::cout << "Music resumed" << std::endl;
182                 }
183             }
184         }
185
186         // Calculate frame time
187         double frameTime = clock.restart().asSeconds();
188         elapsedTime += frameTime;
189
190         // Update universe state
191         universe.step(dt);
192
193         // Clear and draw everything
194         window.clear();
195
196         // Draw universe
197         window.setView(universeView);
198         window.draw(background);
199         window.draw(universe);
200
201         // Draw time
202         window.setView(defaultView);
203         window.draw(timeBox);
204
205         // Display elapsed time in simulation
206         int seconds = static_cast<int>(elapsedTime);
```

```
        int tens = seconds / 10;
        int ones = seconds % 10;

        drawDigit(window, tens, defaultView.getSize().x - 150, 15, 20);
        drawDigit(window, ones, defaultView.getSize().x - 120, 15, 20);

        // Draw "sec" for seconds
        drawSec(window, defaultView.getSize().x - 90, 15, 20);

        if (firstDraw) {
            std::cout << "Drawing universe for the first time" << std::endl;
            firstDraw = false;
        }

        // Display rendered frame
        window.display();
    }

    // Output universe state at the end
    std::cout << universe;

    return 0;
}
```

4.7.3 CelestialBody.hpp

```
// CopyRight [2024] Ashish Kosana
#pragma once
#include <memory>
#include <string>
#include <SFML/Graphics.hpp>

namespace NB {

class CelestialBody : public sf::Drawable {
 public:
    CelestialBody();
    CelestialBody(double x, double y, double vx, double vy, double mass,
                  const std::string& filename);

    void update(double dt);
    sf::Vector2f getPosition() const;
    sf::Vector2f getVelocity() const;
    double getMass() const;
```

```
19
20      void setVelocity(const sf::Vector2f& newVelocity);
21      void setPosition(const sf::Vector2f& newPosition);
22      void applyForce(const sf::Vector2f& force, double dt);
23
24      friend std::istream& operator>>(std::istream& is, CelestialBody& body);
25      friend std::ostream& operator<<(std::ostream& os, const CelestialBody&
        body);
26
27  protected:
28      void draw(sf::RenderTarget& target, sf::RenderStates states) const
        override;
29
30  private:
31      sf::Vector2f position;
32      sf::Vector2f velocity;
33      double mass;
34      sf::Texture texture;
35      mutable sf::Sprite sprite;
36      static constexpr float SCALE = 1e-9f;
37      static constexpr double G = 6.67430e-11;
38  };
39
40  }  // namespace NB
```

4.7.4 CelestialBody.cpp

```
1  // CopyRight [2024] Ashish Kosana
2  #include "CelestialBody.hpp"
3  #include <iostream>
4
5  namespace NB {
6
7  CelestialBody::CelestialBody() : mass(0) {}
8
9  CelestialBody::CelestialBody(double x, double y, double vx, double vy,
       double m,
10 const std::string& filename)
11 : position(x, y), velocity(vx, vy), mass(m) {
12 if (!texture.loadFromFile(filename)) {
13 std::cerr << "Failed to load texture: " << filename << std::endl;
14 }
15 sprite.setTexture(texture);
```

```
16 sprite.setOrigin(sprite.getLocalBounds().width / 2, sprite.getLocalBounds().
      height / 2);
17 sprite.setPosition(position.x * SCALE, position.y * SCALE);
18 }
19
20 void CelestialBody::update(double dt) {
21 position += velocity * static_cast<float>(dt);
22 sprite.setPosition(position.x * SCALE, position.y * SCALE);
23 }
24
25 sf::Vector2f CelestialBody::getPosition() const {
26 return position;
27 }
28
29 sf::Vector2f CelestialBody::getVelocity() const {
30 return velocity;
31 }
32
33 double CelestialBody::getMass() const {
34 return mass;
35 }
36
37 void CelestialBody::setVelocity(const sf::Vector2f& newVelocity) {
38 velocity = newVelocity;
39 }
40
41 void CelestialBody::setPosition(const sf::Vector2f& newPosition) {
42 position = newPosition;
43 sprite.setPosition(position.x * SCALE, position.y * SCALE);
44 }
45
46 void CelestialBody::applyForce(const sf::Vector2f& force, double dt) {
47 sf::Vector2f acceleration = force / static_cast<float>(mass);
48 velocity += acceleration * static_cast<float>(dt);
49 }
50
51 void CelestialBody::draw(sf::RenderTarget& target, sf::RenderStates states)
      const {
52 target.draw(sprite, states);
53 }
54
55 std::istream& operator>>(std::istream& is, CelestialBody& body) {
56 std::string filename;
57 is >> body.position.x >> body.position.y >> body.velocity.x >> body.velocity
      .y
```

```
>> body.mass >> filename;
if (!body.texture.loadFromFile(filename)) {
std::cerr << "Failed to load texture: " << filename << std::endl;
}
body.sprite.setTexture(body.texture);
body.sprite.setOrigin(body.sprite.getLocalBounds().width / 2,
body.sprite.getLocalBounds().height / 2);
body.sprite.setPosition(body.position.x * body.SCALE, body.position.y * body
    .SCALE);
return is;
}

std::ostream& operator<<(std::ostream& os, const CelestialBody& body) {
os << body.position.x << " " << body.position.y << " "
<< body.velocity.x << " " << body.velocity.y << " "
<< body.mass << " " << "image.png";  // Assuming image filename
return os;
}

}  // namespace NB
```

4.7.5 Universe.hpp

```
// CopyRight [2024] Ashish Kosana
#pragma once

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

namespace NB {

class Universe : public sf::Drawable {
 public:
    Universe();
    explicit Universe(const std::string& filename);

    void update(double dt);
    void step(double seconds);
    size_t size() const;
    double getRadius() const;
    const CelestialBody& operator[](size_t index) const;

```

```
22      friend std::istream& operator>>(std::istream& is, Universe& universe);
23      friend std::ostream& operator<<(std::ostream& os, const Universe&
        universe);
24
25  protected:
26      void draw(sf::RenderTarget& target, sf::RenderStates states) const
        override;
27
28  private:
29      std::vector<std::unique_ptr<CelestialBody>> bodies;
30      double radius;
31      sf::Vector2f calculateForce(const CelestialBody& body1, const
        CelestialBody& body2) const;
32  };
33
34  }  // namespace NB
```

4.7.6 Universe.cpp

```
1   // CopyRight [2024] Ashish Kosana
2   #include "Universe.hpp"
3   #include <fstream>
4   #include <iostream>
5   #include <cmath>
6
7   namespace NB {
8
9   Universe::Universe() : radius(0) {}
10
11  Universe::Universe(const std::string& filename) {
12  std::ifstream file(filename);
13  if (file) {
14  file >> *this;
15  } else {
16  std::cerr << "Failed to open file: " << filename << std::endl;
17  }
18  }
19
20  void Universe::update(double dt) {
21  for (auto& body : bodies) {
22  body->update(dt);
23  }
24  }
25
```

```
26 void Universe::step(double seconds) {
27 for (auto& body : bodies) {
28 sf::Vector2f totalForce(0, 0);
29 for (const auto& otherBody : bodies) {
30 if (body.get() != otherBody.get()) {
31 totalForce += calculateForce(*body, *otherBody);
32 }
33 }
34 body->applyForce(totalForce, seconds);
35 }
36
37 for (auto& body : bodies) {
38 body->update(seconds);
39 }
40 }
41
42 size_t Universe::size() const { return bodies.size(); }
43 double Universe::getRadius() const { return radius; }
44
45 const CelestialBody& Universe::operator[](size_t index) const {
46 return *bodies[index];
47 }
48
49 void Universe::draw(sf::RenderTarget& target, sf::RenderStates states) const
       {
50 for (const auto& body : bodies) {
51 target.draw(*body, states);
52 }
53 }
54
55 sf::Vector2f Universe::calculateForce(const CelestialBody& body1,
56 const CelestialBody& body2) const {
57 sf::Vector2f delta = body2.getPosition() - body1.getPosition();
58 double distance = std::sqrt(delta.x * delta.x + delta.y * delta.y);
59 double force = (6.67430e-11 * body1.getMass() * body2.getMass()) / (distance
       * distance);
60 return (delta / static_cast<float>(distance)) * static_cast<float>(force);
61 }
62
63 std::istream& operator>>(std::istream& is, Universe& universe) {
64 size_t n;
65 is >> n >> universe.radius;
66 universe.bodies.clear();
67 for (size_t i = 0; i < n; ++i) {
68 auto body = std::make_unique<CelestialBody>();
```

```
is >> *body;
universe.bodies.push_back(std::move(body));
}
return is;
}

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

}  // namespace NB
```

4.7.7 test.cpp

```
// CopyRight [2024] Ashish Kosana
#define BOOST_TEST_MODULE NBodyTests
#include <sstream>
#include <boost/test/included/unit_test.hpp>
#include "Universe.hpp"
#include "CelestialBody.hpp"

const float EPSILON = 1e-6f;

BOOST_AUTO_TEST_CASE(CelestialBodyInputOutput) {
    std::stringstream ss("1.496e+11 0 0 29800 5.974e+24 earth.gif");
    NB::CelestialBody body;
    ss >> body;

    BOOST_CHECK_CLOSE(body.getPosition().x, 1.496e+11f, EPSILON);
    BOOST_CHECK_SMALL(body.getPosition().y, EPSILON);
    BOOST_CHECK_SMALL(body.getVelocity().x, EPSILON);
    BOOST_CHECK_CLOSE(body.getVelocity().y, 29800.0f, EPSILON);
    BOOST_CHECK_CLOSE(body.getMass(), 5.974e+24f, EPSILON);

    std::stringstream out;
    out << body;
    BOOST_CHECK_EQUAL(out.str(), "1.496e+11 0 0 29800 5.974e+24 image.png");
}

BOOST_AUTO_TEST_CASE(UniverseInitialization) {
```

```
27      NB::Universe universe;
28      BOOST_CHECK_EQUAL(universe.size(), 0);
29      BOOST_CHECK_SMALL(static_cast<float>(universe.getRadius()), EPSILON);
30  }
31
32  BOOST_AUTO_TEST_CASE(UniverseInputOutput) {
33      std::stringstream ss("1 2.50e+11\n1.496e+11 0 0 29800 5.974e+24 mars.gif
        \n");
34      NB::Universe universe;
35      ss >> universe;
36
37      BOOST_CHECK_EQUAL(universe.size(), 1);
38      BOOST_CHECK_CLOSE(static_cast<float>(universe.getRadius()), 2.50e+11f,
        EPSILON);
39
40      std::stringstream out;
41      out << universe;
42      BOOST_CHECK_EQUAL(out.str(), "1 2.5e+11\n1.496e+11 0 0 29800 5.974e+24
        image.png\n");
43  }
44
45  BOOST_AUTO_TEST_CASE(UniverseAccessOperator) {
46      std::stringstream ss("1 2.50e+11\n1.496e+11 0 0 29800 5.974e+24 mercury.
        gif\n");
47      NB::Universe universe;
48      ss >> universe;
49
50      BOOST_CHECK_CLOSE(universe[0].getPosition().x, 1.496e+11f, EPSILON);
51      BOOST_CHECK_SMALL(universe[0].getPosition().y, EPSILON);
52      BOOST_CHECK_SMALL(universe[0].getVelocity().x, EPSILON);
53      BOOST_CHECK_CLOSE(universe[0].getVelocity().y, 29800.0f, EPSILON);
54      BOOST_CHECK_CLOSE(universe[0].getMass(), 5.974e+24f, EPSILON);
55  }
56
57  BOOST_AUTO_TEST_CASE(UniverseSimulationStep) {
58      std::stringstream ss("1 2.50e+11\n1.496e+11 0 0 29800 5.974e+24 sun.gif\
        n");
59      NB::Universe universe;
60      ss >> universe;
61
62      BOOST_CHECK_CLOSE(universe[0].getPosition().x, 1.496e+11f, EPSILON);
63      BOOST_CHECK_SMALL(universe[0].getPosition().y, EPSILON);
64
65      universe.step(1.0);
66
```

```
67      BOOST_CHECK_CLOSE(universe[0].getPosition().x, 1.496e+11f, EPSILON);
68      BOOST_CHECK_CLOSE(universe[0].getPosition().y, 29800.0f, EPSILON);
69  }
70
71  BOOST_AUTO_TEST_CASE(UniverseMultipleBodies) {
72      std::stringstream ss("3 2.50e+11\n"
73                           "1.496e+11 0 0 29800 5.974e+24 saturn.gif\n"
74                           "2.279e+11 0 0 24100 6.419e+23 pluto.gif\n"
75                           "5.790e+10 0 0 47900 3.302e+23 uranus.gif\n");
76      NB::Universe universe;
77      ss >> universe;
78
79      BOOST_CHECK_EQUAL(universe.size(), 3);
80      BOOST_CHECK_CLOSE(static_cast<float>(universe.getRadius()), 2.50e+11f,
        EPSILON);
81
82      // Check Earth
83      BOOST_CHECK_CLOSE(universe[0].getPosition().x, 1.496e+11f, EPSILON);
84      BOOST_CHECK_CLOSE(universe[0].getVelocity().y, 29800.0f, EPSILON);
85      BOOST_CHECK_CLOSE(universe[0].getMass(), 5.974e+24f, EPSILON);
86
87      // Check Mars
88      BOOST_CHECK_CLOSE(universe[1].getPosition().x, 2.279e+11f, EPSILON);
89      BOOST_CHECK_CLOSE(universe[1].getVelocity().y, 24100.0f, EPSILON);
90      BOOST_CHECK_CLOSE(universe[1].getMass(), 6.419e+23f, EPSILON);
91
92      // Check Mercury
93      BOOST_CHECK_CLOSE(universe[2].getPosition().x, 5.790e+10f, EPSILON);
94      BOOST_CHECK_CLOSE(universe[2].getVelocity().y, 47900.0f, EPSILON);
95      BOOST_CHECK_CLOSE(universe[2].getMass(), 3.302e+23f, EPSILON);
96
97      universe.step(1.0);
98
99      // Check updated positions
100     BOOST_CHECK_CLOSE(universe[0].getPosition().y, 29800.0f, EPSILON);
101     BOOST_CHECK_CLOSE(universe[1].getPosition().y, 24100.0f, EPSILON);
102     BOOST_CHECK_CLOSE(universe[2].getPosition().y, 47900.0f, EPSILON);
103 }
104
105 // New test case for physics simulation
106 BOOST_AUTO_TEST_CASE(PhysicsSimulation) {
107     std::stringstream ss("2 2.50e+11\n"
108                          "0 0 0 0 1.989e+30 sun.gif\n"
109                          "1.496e+11 0 0 29800 5.974e+24 earth.gif\n");
110     NB::Universe universe;
```

```
111     ss >> universe;
112
113     BOOST_CHECK_EQUAL(universe.size(), 2);
114
115     // Initial positions
116     BOOST_CHECK_SMALL(universe[0].getPosition().x, EPSILON);
117     BOOST_CHECK_SMALL(universe[0].getPosition().y, EPSILON);
118     BOOST_CHECK_CLOSE(universe[1].getPosition().x, 1.496e+11f, EPSILON);
119     BOOST_CHECK_SMALL(universe[1].getPosition().y, EPSILON);
120
121     // Simulate for 1 day
122     double dt = 86400;  // 1 day in seconds
123     universe.step(dt);
124
125     // Check that the sun has not moved significantly
126     BOOST_CHECK_SMALL(universe[0].getPosition().x, 1e5f);
127     BOOST_CHECK_SMALL(universe[0].getPosition().y, 1e5f);
128
129     // Check that the Earth has moved
130     BOOST_CHECK_CLOSE(universe[1].getPosition().x, 1.496e+11f, 1.0f);
131     BOOST_CHECK(std::abs(universe[1].getPosition().y) > 2.5e+9f);
132
133     // Check that the Earth's velocity has changed slightly due to the sun's
        gravity
134     BOOST_CHECK(universe[1].getVelocity().x < 0);
135     BOOST_CHECK_CLOSE(universe[1].getVelocity().y, 29800.0f, 1.0f);
136 }
```

4.7.8 Screenshot

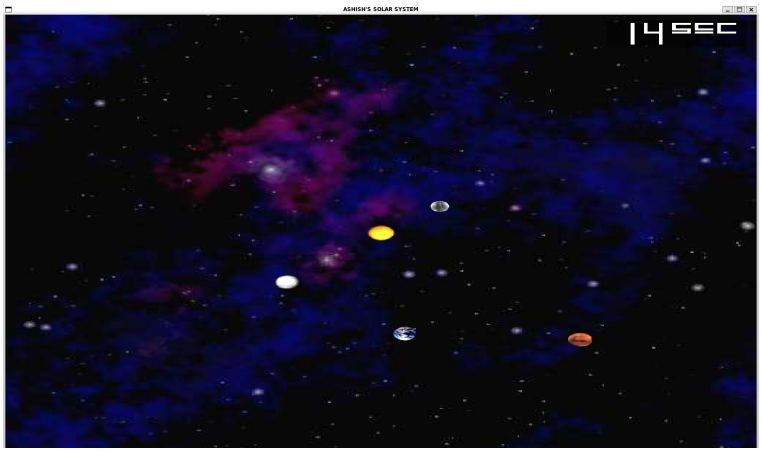

Figure 5: My Solar System

5 PS4: Sokoban UI & Sokoban

5.1 Overview

I implemented a Sokoban game using C++ and SFML, implementing the core mechanics of this classic puzzle game. The project involved creating a game board, handling player movement, and managing box pushing mechanics.

5.2 What I Accomplished

1.Implemented level loading from text files, allowing for easy addition of new puzzles
2.Implemented a win condition check to determine when the player has completed the level
3.Developed a rendering system using SFML to display the game board, player, boxes, and storage locations

5.3 What I Learned

==>I have learned how to handle the user inputs and event processing.
==>Rendering game elements using sprites

5.4 Discussion of Key Algorithms

Data Structures:
1.sf::Vector2i
==>Represents the player's current position on the game board.
2.sf::Sprite Array:
Includes sprites for ground, walls, boxes, storage locations, and player.
3. sf::Texture Array:
==>Stores textures for game elements.
==>Corresponds to the sprite array for efficient texture management.
Algorithms:
1.Player Movement Algorithm:
==>Checks for valid moves based on surrounding cells.
==>Updates player position and box positions if applicable.
2. Win Condition Checking:
==>Iterates through the game board to verify if all boxes are on storage locations.
Object Oriented Designs:
==>Class:
The sokoban class represents the core game logic and contains the member variables.
==>Operator Overloading:
The operator« and operator» for Sokoban and sf::Vector2 are overloaded to handle input and output operations properly.

5.5 What I Already Knew

==>I have idead about c++ file handling.
==>I have grip over sfml ,that how to load the textures etc from previous projects.

5.6 Challenges

==>I have faced difficulty in implementing the logic for pushing the boxes on to the storage areas and preventing the player not to push the boxes on to the blocked area.
==>I have faced some difficulty in implementing the undo logic.
==>I have also faced difficulty in implementing the victory where the user gets congratulations you won the game on the window.
==>I have faced little difficulty in playing the victory music by checking the win condition.

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] Ashish Kosana
#include <iostream>
#include <fstream>
#include <sstream>
#include <iomanip>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>  // Include SFML Audio module
#include "Sokoban.hpp"

// Function to load best time from file
int loadBestTime(const std::string& levelName) {
    std::ifstream file(levelName + "_best_time.txt");
    int bestTime = -1;
    if (file >> bestTime) {
        return bestTime;
    }
    return -1;  // If no time is recorded, return -1
}

// Function to save best time to file
void saveBestTime(const std::string& levelName, int bestTime) {
    std::ofstream file(levelName + "_best_time.txt");
    file << bestTime;
}

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

    SB::Sokoban game(argv[1]);
    sf::RenderWindow window(sf::VideoMode(game.pixelWidth(),
    game.pixelHeight()), "ASHISH'S SOKOBAN");

    // Load font
    sf::Font font;
    if (!font.loadFromFile("arialn.ttf")) {
        std::cerr << "Error loading font" << std::endl;
        return 1;
    }

    // Load victory music
```

```
    sf::Music victoryMusic;
    if (!victoryMusic.openFromFile("victory.wav")) {
        std::cerr << "Error loading victory music" << std::endl;
        return 1;
    }

    // Timer and reset texts
    sf::Text timerText("", font, 20);
    timerText.setFillColor(sf::Color::White);
    timerText.setPosition(10, 10);

    // Move counter text
    sf::Text moveCounterText("Moves: 0", font, 20);
    moveCounterText.setFillColor(sf::Color::White);
    moveCounterText.setPosition(10, 40);

    // Best time display
    sf::Text bestTimeText("Best Time: --:--", font, 20);
    bestTimeText.setFillColor(sf::Color::Yellow);
    bestTimeText.setPosition(10, 70);

    // Load the best time from file
    int bestTime = loadBestTime(argv[1]);
    if (bestTime != -1) {
        int bestMinutes = bestTime / 60;
        int bestSeconds = bestTime % 60;
        std::stringstream bestTimeStream;
        bestTimeStream << "Best Time: " << std::setw(2) << std::setfill('0')
                       << bestMinutes << ":" << std::setw(2) << std::setfill
    ('0')
                       << bestSeconds;
        bestTimeText.setString(bestTimeStream.str());
    }

    sf::Clock clock;
    int moveCount = 0;
    bool gameWon = false;
    bool musicPlayed = false;  // Flag to check if music has been played

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

```
88              if (event.type == sf::Event::KeyPressed) {
89                  if (!gameWon) {
90                      switch (event.key.code) {
91                          case sf::Keyboard::Up:
92                          case sf::Keyboard::W:
93                              game.movePlayer(SB::Direction::Up);
94                              moveCount++;
95                              break;
96                          case sf::Keyboard::Down:
97                          case sf::Keyboard::S:
98                              game.movePlayer(SB::Direction::Down);
99                              moveCount++;
100                             break;
101                         case sf::Keyboard::Left:
102                         case sf::Keyboard::A:
103                             game.movePlayer(SB::Direction::Left);
104                             moveCount++;
105                             break;
106                         case sf::Keyboard::Right:
107                         case sf::Keyboard::D:
108                             game.movePlayer(SB::Direction::Right);
109                             moveCount++;
110                             break;
111                         case sf::Keyboard::U:  // Undo move
112                             game.undo();
113                             if (moveCount > 0) moveCount--;
114                             break;
115                         case sf::Keyboard::R:  // Reset game
116                             game.reset();
117                             clock.restart();
118                             moveCount = 0;
119                             musicPlayed = false;  // Reset music played flag
120                             victoryMusic.stop();  // Stop music if playing
121                             break;
122                         default:
123                             break;
124                     }
125                 }
126                 // Check for reset after winning
127                 if (gameWon && event.key.code == sf::Keyboard::R) {
128                     game.reset();
129                     clock.restart();
130                     moveCount = 0;
131                     gameWon = false;  // Reset gameWon status to allow
    playing again
```

```
132                     musicPlayed = false;  // Reset music played flag
133                     victoryMusic.stop();  // Stop music if playing
134                 }
135             }
136         }
137 
138         window.clear();
139         window.draw(game);
140 
141         // Update timer display
142         int seconds = static_cast<int>(clock.getElapsedTime().asSeconds());
143         int minutes = seconds / 60;
144         seconds %= 60;
145         std::stringstream timeStream;
146         timeStream << std::setw(2) << std::setfill('0') << minutes
147                    << ":" << std::setw(2) << std::setfill('0') << seconds;
148         timerText.setString(timeStream.str());
149         window.draw(timerText);
150 
151         // Update move counter display
152         std::stringstream moveStream;
153         moveStream << "Moves: " << moveCount;
154         moveCounterText.setString(moveStream.str());
155         window.draw(moveCounterText);
156 
157         // Display best time
158         window.draw(bestTimeText);
159 
160         // Check if the game is won and display the win message if true
161         if (game.isWon()) {
162             gameWon = true;
163             sf::Text winText("Congratulations! You won!", font, 30);
164             winText.setFillColor(sf::Color::Green);
165             winText.setPosition(game.pixelWidth() / 2 - winText.
    getLocalBounds().width / 2,
166                                 game.pixelHeight() / 2 - winText.
    getLocalBounds().height / 2);
167             window.draw(winText);
168 
169             // Play victory music if not already played
170             if (!musicPlayed) {
171                 victoryMusic.play();
172                 musicPlayed = true;  // Set flag to indicate music has been
    played
173             }
```

```

            // Calculate elapsed time and update best time if necessary
            int elapsedTime = static_cast<int>(clock.getElapsedTime().
    asSeconds());
            if (bestTime == -1 || elapsedTime < bestTime) {
                bestTime = elapsedTime;
                saveBestTime(argv[1], bestTime);
                bestTimeText.setString("Best Time: "
                + timeStream.str());  // Update best time display
            }
        }

        window.display();
    }

    return 0;
}
```

5.7.3 Sokoban.hpp

```

// Copyright [2024] Ashish Kosana
#ifndef SOKOBAN_HPP
#define SOKOBAN_HPP

#include <ostream>
#include <vector>
#include <string>
#include <stack>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <SFML/System/Vector2.hpp>

template <typename T>
std::ostream& operator<<(std::ostream& os, const sf::Vector2<T>& vec);

namespace SB {

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

struct Move {
    sf::Vector2i playerOldPos;
    sf::Vector2i playerNewPos;
    sf::Vector2i boxOldPos;
```

```
25      sf::Vector2i boxNewPos;
26      bool isBoxMove;
27  };
28
29  class Sokoban : public sf::Drawable {
30   public:
31      Sokoban();
32      explicit Sokoban(const std::string& filename);
33      int width() const;
34      int height() const;
35      sf::Vector2i playerLoc() const;
36      void movePlayer(Direction dir);
37      bool isWon() const;
38      void reset();
39      int pixelWidth() const;
40      int pixelHeight() const;
41      void undo();
42
43      friend std::istream& operator>>(std::istream& is, Sokoban& sokoban);
44      friend std::ostream& operator<<(std::ostream& os, const Sokoban& sokoban
        );
45
46   protected:
47      void draw(sf::RenderTarget& target, sf::RenderStates states) const
        override;
48
49   private:
50      static const int TILE_SIZE = 64;
51      int m_width;
52      int m_height;
53      std::vector<char> m_grid;
54      std::vector<char> m_initialGrid;
55      sf::Vector2i m_playerPos;
56      sf::Vector2i m_initialPlayerPos;
57      sf::Texture m_textures[8];  // Increased for multiple box colors and
        player directions
58      sf::Sprite m_sprites[8];
59      Direction m_lastDirection;
60      std::stack<Move> m_moveHistory;
61      sf::SoundBuffer m_victorySoundBuffer;
62      sf::Sound m_victorySound;
63
64      void loadTextures();
65      void loadFromFile(const std::string& filename);
66      void updatePlayerSprite();
```

```
};

}  // namespace SB

#endif  // SOKOBAN_HPP
```

5.7.4 Sokoban.cpp

```
// Copyright [2024] Ashish Kosana
#include "Sokoban.hpp"
#include <iostream>
#include <fstream>
#include <algorithm>

template <typename T>
std::ostream& operator<<(std::ostream& os, const sf::Vector2<T>& vec) {
    os << "(" << vec.x << ", " << vec.y << ")";
    return os;
}

// Explicit instantiation for int
template std::ostream& operator<<(std::ostream&, const sf::Vector2<int>&);

namespace SB {

Sokoban::Sokoban() : m_width(0), m_height(0), m_lastDirection(Direction::
   Down) {
    loadTextures();
}

Sokoban::Sokoban(const std::string& filename) : m_width(0), m_height(0),
m_lastDirection(Direction::Down) {
    loadTextures();
    loadFromFile(filename);
}

void Sokoban::loadTextures() {
    const std::string textureFiles[] = {
        "ground_01.png", "block_06.png", "crate_03.png", "ground_04.png",
        "player_05.png", "player_20.png", "player_17.png", "player_08.png"
    };

    for (int i = 0; i < 8; ++i) {
        if (!m_textures[i].loadFromFile(textureFiles[i])) {
```

```
36              std::cerr << "Failed to load " << textureFiles[i] << std::endl;
37          }
38          m_sprites[i].setTexture(m_textures[i]);
39      }
40  }
41
42  int Sokoban::width() const { return m_width; }
43  int Sokoban::height() const { return m_height; }
44  sf::Vector2i Sokoban::playerLoc() const { return m_playerPos; }
45
46  void Sokoban::movePlayer(Direction dir) {
47      sf::Vector2i newPos = m_playerPos;
48      switch (dir) {
49          case Direction::Up: newPos.y--; break;
50          case Direction::Down: newPos.y++; break;
51          case Direction::Left: newPos.x--; break;
52          case Direction::Right: newPos.x++; break;
53      }
54
55      m_lastDirection = dir;
56      updatePlayerSprite();
57
58      if (newPos.x >= 0 && newPos.x < m_width &&
59          newPos.y >= 0 && newPos.y < m_height) {
60          char &targetCell = m_grid[newPos.y * m_width + newPos.x];
61          Move move{m_playerPos, newPos, {-1, -1}, {-1, -1}, false};
62
63          if (targetCell == '.' || targetCell == 'a' || targetCell == 'b') {
64              std::swap(m_grid[m_playerPos.y * m_width + m_playerPos.x],
    targetCell);
65              m_playerPos = newPos;
66              m_moveHistory.push(move);
67          } else if (targetCell == 'A' || targetCell == 'B' ||
68                     targetCell == '1' || targetCell == '2') {
69              sf::Vector2i boxNewPos = newPos + (newPos - m_playerPos);
70              if (boxNewPos.x >= 0 && boxNewPos.x < m_width &&
71                  boxNewPos.y >= 0 && boxNewPos.y < m_height) {
72                  char &boxTargetCell = m_grid[boxNewPos.y * m_width +
    boxNewPos.x];
73                  if (boxTargetCell == '.' || boxTargetCell == 'a' ||
74                      boxTargetCell == 'b') {
75                      move.isBoxMove = true;
76                      move.boxOldPos = newPos;
77                      move.boxNewPos = boxNewPos;
78
```

```
                    if (targetCell == 'A' || targetCell == '1') {
                        boxTargetCell = (boxTargetCell == '.') ? 'A' :
                            (boxTargetCell == 'a' ? '1' : 'A');
                    } else {  // 'B' or '2'
                        boxTargetCell = (boxTargetCell == '.') ? 'B' :
                            (boxTargetCell == 'b' ? '2' : 'B');
                    }

                    targetCell = (targetCell == 'A' || targetCell == 'B') ? '.' :
                        (targetCell == '1' ? 'a' : 'b');
                    m_grid[m_playerPos.y * m_width + m_playerPos.x] =
                        (m_grid[m_playerPos.y * m_width + m_playerPos.x] == '@') ?
                        '.' : (m_grid[m_playerPos.y * m_width + m_playerPos.x] == '!' ?
                        'a' : 'b');
                    m_playerPos = newPos;
                    m_grid[m_playerPos.y * m_width + m_playerPos.x] = '@';
                    m_moveHistory.push(move);
                }
            }
        }
    }

    if (isWon()) {
        // Victory condition reached, perform relevant action
    }
}

bool Sokoban::isWon() const {
    return std::none_of(m_grid.begin(), m_grid.end(), [](char c) {
        return c == 'A' || c == 'B'; });
}

void Sokoban::updatePlayerSprite() {
    switch (m_lastDirection) {
        case Direction::Up:
            m_sprites[4] = sf::Sprite(m_textures[7]);  // Up sprite
            break;
        case Direction::Down:
            m_sprites[4] = sf::Sprite(m_textures[4]);  // Down sprite
            break;
        case Direction::Left:
            m_sprites[4] = sf::Sprite(m_textures[5]);  // Left sprite
```

```
121             break;
122         case Direction::Right:
123             m_sprites[4] = sf::Sprite(m_textures[6]);  // Right sprite
124             break;
125     }
126 }
127 
128 void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
129     for (int y = 0; y < m_height; ++y) {
130         for (int x = 0; x < m_width; ++x) {
131             char cell = m_grid[y * m_width + x];
132             sf::Sprite sprite = m_sprites[0];  // Background sprite
133             sprite.setPosition(x * TILE_SIZE, y * TILE_SIZE);
134             target.draw(sprite, states);
135 
136             switch (cell) {
137                 case '#': sprite = m_sprites[1]; break;
138                 case 'A': sprite = m_sprites[2]; break;
139                 case 'B': sprite = m_sprites[7]; break;
140                 case 'a':
141                 case 'b': sprite = m_sprites[3]; break;
142                 case '@':
143                     sprite = m_sprites[4];  // Player sprite based on
    direction
144                     break;
145                 default: continue;  // Skip drawing for empty spaces
146             }
147             sprite.setPosition(x * TILE_SIZE, y * TILE_SIZE);
148             target.draw(sprite, states);
149         }
150     }
151 }
152 
153 void Sokoban::reset() {
154     m_grid = m_initialGrid;
155     m_playerPos = m_initialPlayerPos;
156     m_moveHistory = std::stack<Move>();
157 }
158 
159 int Sokoban::pixelWidth() const { return m_width * TILE_SIZE; }
160 int Sokoban::pixelHeight() const { return m_height * TILE_SIZE; }
161 
162 void Sokoban::loadFromFile(const std::string& filename) {
163     std::ifstream file(filename);
```

```
164     if (file) {
165         file >> *this;
166     } else {
167         std::cerr << "Failed to open file: " << filename << std::endl;
168     }
169 }
170 
171 void Sokoban::undo() {
172     if (!m_moveHistory.empty()) {
173         Move lastMove = m_moveHistory.top();
174         m_moveHistory.pop();
175 
176         // Undo player move
177         char &oldCell = m_grid[lastMove.playerOldPos.y *
178             m_width + lastMove.playerOldPos.x];
179         char &newCell = m_grid[lastMove.playerNewPos.y *
180             m_width + lastMove.playerNewPos.x];
181 
182         newCell = (newCell == '@') ? '.' : (newCell == '!') ? 'a' : 'b';
183         oldCell = '@';
184         m_playerPos = lastMove.playerOldPos;
185 
186         // Undo box move if applicable
187         if (lastMove.isBoxMove) {
188             char &boxOldCell = m_grid[lastMove.boxOldPos.y *
189                 m_width + lastMove.boxOldPos.x];
190             char &boxNewCell = m_grid[lastMove.boxNewPos.y *
191                 m_width + lastMove.boxNewPos.x];
192 
193             if (boxNewCell == 'A' || boxNewCell == '1') {
194                 boxOldCell = 'A';
195                 boxNewCell = (boxNewCell == 'A') ? '.' : 'a';
196             } else if (boxNewCell == 'B' || boxNewCell == '2') {
197                 boxOldCell = 'B';
198                 boxNewCell = (boxNewCell == 'B') ? '.' : 'b';
199             }
200         }
201     }
202 }
203 
204 std::istream& operator>>(std::istream& is, Sokoban& sokoban) {
205     is >> sokoban.m_height >> sokoban.m_width;
206     sokoban.m_grid.resize(sokoban.m_height * sokoban.m_width);
207     sokoban.m_initialGrid.resize(sokoban.m_height * sokoban.m_width);
208     is.ignore();
```

```
    for (int y = 0; y < sokoban.m_height; ++y) {
        for (int x = 0; x < sokoban.m_width; ++x) {
            char cell;
            is.get(cell);
            sokoban.m_grid[y * sokoban.m_width + x] = cell;
            sokoban.m_initialGrid[y * sokoban.m_width + x] = cell;
            if (cell == '@') {
                sokoban.m_playerPos = sf::Vector2i(x, y);
                sokoban.m_initialPlayerPos = sf::Vector2i(x, y);
            }
        }
        is.ignore();
    }
    return is;
}

std::ostream& operator<<(std::ostream& os, const Sokoban& sokoban) {
    os << sokoban.m_height << " " << sokoban.m_width << std::endl;
    for (int y = 0; y < sokoban.m_height; ++y) {
        for (int x = 0; x < sokoban.m_width; ++x) {
            os << sokoban.m_grid[y * sokoban.m_width + x];
        }
        os << std::endl;
    }
    return os;
}

}  // namespace SB
```

5.7.5 test.cpp

```
// Copyright [2024] Ashish Kosana
#define BOOST_TEST_MODULE SokobanTest
#include <fstream>
#include <iostream>
#include <boost/test/included/unit_test.hpp>
#include "Sokoban.hpp"

// Helper function to create and initialize a test level
void setup_test_level(SB::Sokoban& game, const std::string& level) {
    std::ofstream levelFile(level);
    levelFile << "5 5\n"
              << "#####\n"
              << "#.@.#\n"
```

```
                << "#.A.#\n"
                << "#.a.#\n"  // 'a' is the storage location
                << "#####\n";
    levelFile.close();
    game = SB::Sokoban(level);
}

// Updated simulateWin function with additional logging
bool simulateWin(SB::Sokoban& game) {
    game.movePlayer(SB::Direction::Down);  // Move player to box
    game.movePlayer(SB::Direction::Down);  // Push box onto storage
    return game.isWon();
}

BOOST_AUTO_TEST_CASE(test_basic_movement) {
    SB::Sokoban game;
    setup_test_level(game, "test_level_basic.lvl");

    sf::Vector2i initial_pos = game.playerLoc();
    game.movePlayer(SB::Direction::Right);
    BOOST_CHECK_EQUAL(game.playerLoc().x, initial_pos.x + 1);
    BOOST_CHECK_EQUAL(game.playerLoc().y, initial_pos.y);
}

BOOST_AUTO_TEST_CASE(test_wall_collision) {
    SB::Sokoban game;
    setup_test_level(game, "test_level_wall.lvl");

    sf::Vector2i initial_pos = game.playerLoc();
    game.movePlayer(SB::Direction::Up);  // Move towards wall
    BOOST_CHECK_EQUAL(game.playerLoc().x, initial_pos.x);
    BOOST_CHECK_EQUAL(game.playerLoc().y, initial_pos.y);
}

BOOST_AUTO_TEST_CASE(test_box_push) {
    SB::Sokoban game;
    setup_test_level(game, "test_level_push.lvl");

    BOOST_CHECK(simulateWin(game));  // Check if the game registers as won
}

BOOST_AUTO_TEST_CASE(test_box_blocked) {
    SB::Sokoban game;
    setup_test_level(game, "test_level_blocked.lvl");

```

```
59      game.movePlayer(SB::Direction::Down);  // Move to box
60      game.movePlayer(SB::Direction::Right);  // Attempt to push in blocked
        direction
61      BOOST_CHECK_EQUAL(game.playerLoc().x, 3);  // Confirms that the push
        fails
62  }
63
64  BOOST_AUTO_TEST_CASE(test_win_condition) {
65      SB::Sokoban game;
66      setup_test_level(game, "test_level_win.lvl");
67
68      BOOST_CHECK(simulateWin(game));  // Should return true if win is
        registered
69  }
70
71  BOOST_AUTO_TEST_CASE(test_reset) {
72      SB::Sokoban game;
73      setup_test_level(game, "test_level_reset.lvl");
74
75      sf::Vector2i initial_pos = game.playerLoc();
76      game.movePlayer(SB::Direction::Right);
77      game.reset();
78      BOOST_CHECK_EQUAL(game.playerLoc().x, initial_pos.x);
79      BOOST_CHECK_EQUAL(game.playerLoc().y, initial_pos.y);
80  }
```

5.7.6 Screenshot

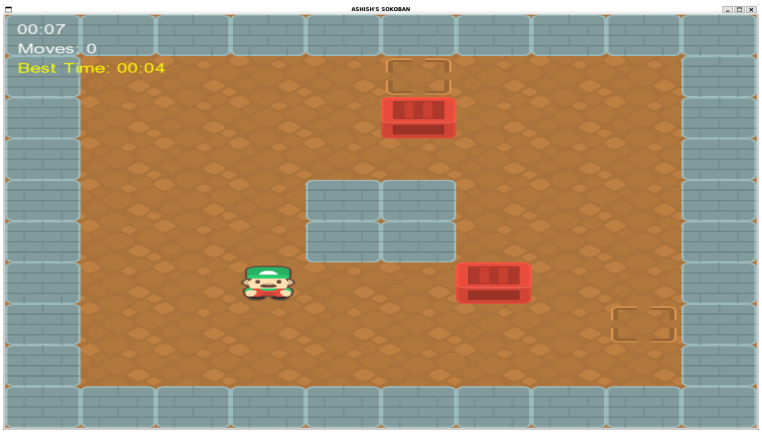

Figure 6: My Sokoban

6 PS5: DNA Alignment

6.1 Overview

I have implemented a program for DNA alignment, which aligns two given DNA sequences using a sequence alignment algorithm. The program computes the optimal alignment and displays the results. This project helped me understand the complexities of sequence alignment algorithms, which are fundamental in bioinformatics for comparing genetic sequences.

6.2 What I Accomplished

1. I have implemented an algorithm for aligning two DNA sequences, ensuring accurate matching of bases.
2. I have incorporated dynamic programming to compute the optimal alignment, including gaps and mismatches.

6.3 What I Learned

==>I understood how DNA alignment is used in real-world applications, such as genetic comparison and evolutionary studies.
==>I learned how sequence alignment algorithms work, including the concepts of scoring matrices, gaps, and penalties.
==>I understood how DNA alignment is used in real-world applications, such as genetic comparison and evolutionary studies.

6.4 Discussion of Key Algorithms

Data Structures:
1.2D Matrix:
==>Used to store the scoring values during the dynamic programming process for sequence alignment.
2. Vectors/Strings:
==>Used to store the input DNA sequences and their aligned counterparts
Algorithms:
1.Edit Distance:
==>Matrix Initialization: The dynamic programming matrix (opt) is initialized with dimensions based on the lengths of the input strings x and y. The matrix is filled with zeros initially.
==>Filling the Matrix: The matrix is filled row by row using a recurrence relation
2.Dynamic Programming:
==>The matrix is filled based on recurrence relations, considering scores for matches, mismatches, and gap penalties.
3. Backtracking:
==>Once the optimal matrix is built, backtracking is used to extract the aligned sequences.
Object Oriented Designs:
==>class:

The EDistance class played a vital role in this project for calculating the distances.
==>Exception Handling:
I have used exception handling to ensure that the input strings are valid or not.

6.5 What I Already Knew

1. String Manipulation:
I had basic idea about string operations, which were crucial for handling and comparing DNA sequences in this project.
2. 2D Arrays/Matrix Handling:
I was familiar with using 2D arrays to store and manipulate data, which is important for the dynamic programming matrix in the alignment algorithm.

6.6 Challenges

==>I have faced difficulty in Ensuring that the algorithm handles various edge cases, such as empty strings, strings with one character, or strings with different lengths.
==>I have faced difficulty in the backtracking process to extract the optimal alignment

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


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

```
// Copyright Ashish Kosana [2024]
#include <iostream>
#include <iomanip>  // For std::setprecision
#include <cstdlib>  // For system calls to get memory usage
#include <cstring>
#include "EDistance.hpp"
#include <SFML/System.hpp>

// Function to get current memory usage (in MB)
double getMemoryUsageMB() {
    double memoryUsage = 0.0;
    FILE* file = fopen("/proc/self/status", "r");
    if (file) {
        char line[128];
        while (fgets(line, 128, file) != NULL) {
            if (strncmp(line, "VmRSS:", 6) == 0) {
                memoryUsage = strtod(line + 6, NULL) / 1024.0;  // Convert
   KB to MB
                break;
            }
        }
        fclose(file);
    }
    return memoryUsage;
}

int main() {
    std::string x, y;
    std::cin >> x >> y;

    double initialMemory = getMemoryUsageMB();

    sf::Clock clock;
```

```
    EDistance ed(x, y);
    int distance = ed.optDistance();
    std::cout << "Edit distance = " << distance << std::endl;
    std::cout << ed.alignment();

    // Print memory usage in MB
    double finalMemory = getMemoryUsageMB();
    double memoryUsedMB = finalMemory - initialMemory;
    std::cout << "Memory used: " << std::fixed << std::setprecision(2)
              << memoryUsedMB << " MB" << std::endl;

    // Print edit distance again after the sequence
    std::cout << "Edit distance after sequence = " << distance << std::endl;

    sf::Time elapsed = clock.getElapsedTime();
    std::cout << "Execution time is " << std::fixed << std::setprecision(6)
              << elapsed.asSeconds() << " seconds" << std::endl;
    return 0;
}
```

6.7.3 EDistance.hpp

```
// Copyright Ashish Kosana [2024]
#ifndef EDISTANCE_HPP
#define EDISTANCE_HPP

#include <string>
#include <vector>

class EDistance {
 public:
    EDistance(const std::string& x, const std::string& y);
    static int penalty(char a, char b);
    static int min3(int a, int b, int c);
    int optDistance();
    std::string alignment();

 private:
    std::string x, y;
    std::vector<std::vector<int>> opt;
    void calculateOptDistance();
};

#endif  // EDISTANCE_HPP
```

6.7.4 EDistance.cpp

```cpp

// Copyright Ashish Kosana [2024]
#include "EDistance.hpp"
#include <algorithm>
#include <stdexcept>

EDistance::EDistance(const std::string& x, const std::string& y) : x(x), y(y) {
    if (x.empty() || y.empty()) {
        throw std::invalid_argument("Input strings must not be empty.");
    }
    opt.resize(x.size() + 1, std::vector<int>(y.size() + 1, 0));
}

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

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

void EDistance::calculateOptDistance() {
    for (size_t i = 0; i <= x.size(); ++i)
        opt[i][y.size()] = 2 * (x.size() - i);

    for (size_t j = 0; j <= y.size(); ++j)
        opt[x.size()][j] = 2 * (y.size() - j);

    for (int i = x.size() - 1; i >= 0; --i) {
        for (int j = y.size() - 1; j >= 0; --j) {
            int matchOrMismatch = opt[i + 1][j + 1] + penalty(x[i], y[j]);
            int gapInX = opt[i + 1][j] + 2;
            int gapInY = opt[i][j + 1] + 2;
            opt[i][j] = min3(matchOrMismatch, gapInX, gapInY);
        }
    }
}

int EDistance::optDistance() {
    calculateOptDistance();
    return opt[0][0];
}
```

```

std::string EDistance::alignment() {
    std::string result;
    std::string::size_type i = 0;
    std::string::size_type j = 0;

    while (i < x.size() || j < y.size()) {
        // Check bounds before accessing opt[i][j]
        if (i < x.size() && j < y.size() &&
            opt[i][j] == opt[i + 1][j + 1] + penalty(x[i], y[j])) {
            // Align characters from both strings
            result += x[i] + std::string(" ") + y[j] + " " +
                      std::to_string(penalty(x[i], y[j])) + "\n";
            i++;
            j++;
        } else if (i < x.size() && (j == y.size() ||
                   opt[i][j] == opt[i + 1][j] + 2)) {
            // Align character from x with a gap in y
            result += x[i] + std::string(" - 2\n");
            i++;
        } else if (j < y.size()) {
            // Align character from y with a gap in x
            result += "- " + std::string(1, y[j]) + " 2\n";
            j++;
        }
    }

    return result;
}
```

6.7.5 test.cpp

```
\texttt{
// Copyright Ashish Kosana [2024]
#include <stdexcept>
#define BOOST_TEST_MODULE EDistanceTest
#include <boost/test/included/unit_test.hpp>
#include "EDistance.hpp"

BOOST_AUTO_TEST_CASE(penalty_test) {
    BOOST_CHECK_EQUAL(EDistance::penalty('A', 'A'), 0);
    BOOST_CHECK_EQUAL(EDistance::penalty('A', 'T'), 1);
    BOOST_CHECK_EQUAL(EDistance::penalty('G', 'G'), 0);
    BOOST_CHECK_EQUAL(EDistance::penalty('C', 'A'), 1);
```

```
13 }
14
15 BOOST_AUTO_TEST_CASE(min3_test) {
16     BOOST_CHECK_EQUAL(EDistance::min3(1, 2, 3), 1);
17     BOOST_CHECK_EQUAL(EDistance::min3(3, 2, 1), 1);
18     BOOST_CHECK_EQUAL(EDistance::min3(3, 3, 3), 3);
19     BOOST_CHECK_EQUAL(EDistance::min3(-1, 0, 1), -1);
20 }
21
22 BOOST_AUTO_TEST_CASE(opt_distance_test) {
23     EDistance ed("AACAGTTACC", "TAAGGTCA");
24     BOOST_CHECK_EQUAL(ed.optDistance(), 7);
25
26     EDistance ed2("ACGT", "ACGT");
27     BOOST_CHECK_EQUAL(ed2.optDistance(), 0);
28 }
29
30 BOOST_AUTO_TEST_CASE(alignment_test) {
31     EDistance ed("AACAGTTACC", "TAAGGTCA");
32     int distance = ed.optDistance();
33     BOOST_CHECK_EQUAL(distance, 7);
34
35     std::string result = ed.alignment();
36     BOOST_CHECK(!result.empty());
37 }
38
39 BOOST_AUTO_TEST_CASE(exception_handling_test) {
40     BOOST_REQUIRE_THROW(EDistance("", "TACG"), std::invalid_argument);
41     BOOST_REQUIRE_THROW(EDistance("ACGT", ""), std::invalid_argument);
42
43     BOOST_REQUIRE_NO_THROW(EDistance("ACGT", "TACG"));
44     BOOST_REQUIRE_NO_THROW(EDistance("A", "G"));
45 }
46
47 BOOST_AUTO_TEST_CASE(complex_alignment_test) {
48     EDistance ed("ACGTACGT", "TACG");
49     int distance = ed.optDistance();
50     BOOST_CHECK_EQUAL(distance, 8);
51
52     std::string result = ed.alignment();
53     BOOST_CHECK(!result.empty());
54     BOOST_CHECK(result.find("2\n") != std::string::npos);
55 }
56 }
```

6.7.6 Input:

```
./EDistance < example10.txt
```

6.7.7 Output:

```
Edit distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
Memory used: 0.00 MB
Edit distance after sequence = 7
Execution time is 0.000202 seconds
```

7 PS6:RandWriter

7.1 Overview

In the RandWriter project, I implemented a program that generates random text files based on user-defined parameters such as the number of words, sentences, or paragraphs. The program uses random word selection to build sentences and paragraphs, and the user can specify the desired output file size

7.2 What I Accomplished

1.I have implemented random word selection from a dictionary to ensure the generated text is readable.
2. I have added functionality for users to specify the desired output file size, making the program versatile for different types of testing or data generation.
3. I have implemented a program that generates random text files based on user-defined parameters

7.3 What I Learned

==>I have learned how to generate random characters based on probabilities.
==>I have learned how to use dictionaries to store data like character frequencies and probabilities.

7.4 Discussion of Key Algorithms & Data Structures

Data Structures:
1. std::string: Used to handle individual words, sentences, and paragraphs that are generated.
2.std::ofstream: Used for writing the generated text to a file.
3. std::vector: Holds the sentences or paragraphs to ensure the program can handle multiple levels of text structure.
Algorithms:
1. Random Word Selection: The program selects random words from the dictionary to form sentences. It ensures the randomness and diversity of the text generated.
2. File Output: The program writes the generated content to a text file with user-specified parameters, such as file size or number of words.
Object Oriented Designs:
==>Exception Handling:
I have used exceptions (e.g: std::invalid argument) to handle errors in a controlled manner.
==>Operator Overloading:
The operator« is overloaded as a friend function for easy printing of RandWriter objects.

7.5 What I Already knew

==> I have girp over c++ programming concepts such as file handling,classes and objects.

7.6 Challenges

==>I have faced difficulty in understanding the k-gram concept and how to use it for text generation.
==>I have faced difficulty in the process of computing the frequency of k-grams and their subsequent characters.
==>I have faced difficulty in handling the invalid inputs, such as when the k-gram does not exist or when the order is zero.

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] Ashish Kosana

#include <string>
#include <unordered_map>
```

```
#include <unordered_set>
#include <vector>
#include <iostream>
#include <stdexcept>
#include <random>

class RandWriter {
 public:
    RandWriter(const std::string& input_text, size_t order);
    size_t orderK() const;

    int freq(const std::string& kgram) const;
    int freq(const std::string& kgram, char c) const;
    char kRand(const std::string& kgram);
    std::string generate(const std::string& kgram, size_t L);

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

 private:
    void populateKgramMap();
    char weightedRandomChar(const std::unordered_map<char, int>& charFreq)
   const;

    size_t k;
    std::string text;
    std::unordered_map<std::string, std::unordered_map<char, int>> kgram_map
   ;
};
```

7.7.3 RandWriter.cpp

```
// Copyright [2024] Ashish Kosana
#include "RandWriter.hpp"
#include <iostream>
#include <stdexcept>
#include <unordered_map>
#include <unordered_set>
#include <vector>
#include <random>


thread_local std::mt19937 gen(std::random_device {}());

RandWriter::RandWriter(const std::string& input_text, size_t order)
```

```
    : k(order), text(input_text) {
    if (text.empty()) {
        throw std::invalid_argument("Input text cannot be empty");
    }
    if (k > text.size()) {
        throw std::invalid_argument("Order k cannot exceed input text length
   ");
    }

    if (k > 0) {
        populateKgramMap();
    }
}

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

void RandWriter::populateKgramMap() {
    size_t n = text.size();
    for (size_t i = 0; i < n; ++i) {
        std::string kgram = text.substr(i, k);

        // Wrap-around handling for k-grams
        if (kgram.size() < k) {
            kgram += text.substr(0, k - kgram.size());
        }

        char next_char = text[(i + k) % n];
        kgram_map[kgram][next_char]++;
    }
}

int RandWriter::freq(const std::string& kgram) const {
    if (k == 0) {
        if (kgram.empty()) {
            return text.size();
        }
        throw std::invalid_argument("kgram must be empty when k = 0");
    }
    if (kgram.size() != k) {
        throw std::invalid_argument("kgram must be of length k");
    }
    auto it = kgram_map.find(kgram);
    if (it == kgram_map.end()) {
```

```
58          return 0;
59      }
60      int total = 0;
61      for (const auto& pair : it->second) {
62          total += pair.second;
63      }
64      return total;
65  }
66
67  int RandWriter::freq(const std::string& kgram, char c) const {
68      if (k == 0) {
69          return 0;  // No k-grams exist for k = 0
70      }
71      if (kgram.size() != k) {
72          throw std::invalid_argument("kgram must be of length k");
73      }
74      auto it = kgram_map.find(kgram);
75      if (it == kgram_map.end()) {
76          return 0;
77      }
78      auto char_it = it->second.find(c);
79      return (char_it != it->second.end()) ? char_it->second : 0;
80  }
81
82  char RandWriter::kRand(const std::string& kgram) {
83      if (k == 0) {
84          throw std::invalid_argument("kRand not valid for k = 0");
85      }
86      if (kgram.size() != k) {
87          throw std::invalid_argument("kgram must be of length k");
88      }
89      auto it = kgram_map.find(kgram);
90      if (it == kgram_map.end()) {
91          throw std::invalid_argument("kgram not found in text");
92      }
93      return weightedRandomChar(it->second);
94  }
95
96  char RandWriter::weightedRandomChar(const std::unordered_map<char, int>&
        charFreq) const {
97      std::vector<char> chars;
98      std::vector<int> weights;
99
100     for (const auto& pair : charFreq) {
101         chars.push_back(pair.first);
```

```
        weights.push_back(pair.second);
    }

    std::discrete_distribution<> dist(weights.begin(), weights.end());
    return chars[dist(gen)];
}

std::string RandWriter::generate(const std::string& kgram, size_t L) {
    if (k == 0) {
        throw std::invalid_argument("Cannot generate text with k = 0");
    }
    if (kgram.size() != k) {
        throw std::invalid_argument("kgram must be of length k");
    }
    if (L < k) {
        throw std::invalid_argument("L must be at least k");
    }

    std::string result = kgram;
    for (size_t i = k; i < L; ++i) {
        char next_char = kRand(result.substr(result.size() - k, k));
        result += next_char;
    }
    return result;
}

std::ostream& operator<<(std::ostream& os, const RandWriter& rw) {
    os << "Order: " << rw.k << "\nAlphabet: ";
    std::unordered_set<char> alphabet;
    for (char c : rw.text) {
        alphabet.insert(c);
    }
    for (char c : alphabet) {
        os << c << " ";
    }
    os << "\nFrequencies:\n";
    for (const auto& pair : rw.kgram_map) {
        os << pair.first << ": ";
        for (const auto& sub_pair : pair.second) {
            os << sub_pair.first << "(" << sub_pair.second << ") ";
        }
        os << "\n";
    }
    return os;
}
```

7.7.4 TextWriter.cpp

```cpp
// Copyright [2024] Ashish Kosana
#include <iostream>
#include <fstream>
#include <stdexcept>
#include "RandWriter.hpp"

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

    size_t k = 0, L = 0;
    try {
        k = std::stoi(argv[1]);
        L = std::stoi(argv[2]);
    } catch (const std::exception& e) {
        std::cerr << "Error parsing arguments: " << e.what() << std::endl;
        return 1;
    }

    std::string text((std::istreambuf_iterator<char>(std::cin)),
                     std::istreambuf_iterator<char>());

    if (text.empty()) {
        std::cerr << "Error: No input text provided.\n";
        return 1;
    }

    try {
        RandWriter writer(text, k);


        auto generateKgram = [&text, k]() { return text.substr(0, k); };

        std::string kgram = generateKgram();
        std::cout << writer.generate(kgram, L) << 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] Ashish Kosana

#define BOOST_TEST_MODULE RandWriterTests
#define BOOST_TEST_DYN_LINK
#include <stdexcept>
#include <boost/test/unit_test.hpp>
#include "RandWriter.hpp"

BOOST_AUTO_TEST_CASE(TestConstructor) {
    // Test valid construction
    BOOST_REQUIRE_NO_THROW(RandWriter("abcde", 2));
    BOOST_REQUIRE_NO_THROW(RandWriter("abc", 0));

    // Test invalid construction
    BOOST_REQUIRE_THROW(RandWriter("", 2), std::invalid_argument);
    BOOST_REQUIRE_THROW(RandWriter("abc", 4), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(TestOrderK) {
    RandWriter rw("abcde", 2);
    BOOST_REQUIRE_EQUAL(rw.orderK(), 2);
}

BOOST_AUTO_TEST_CASE(TestFreq) {
    RandWriter rw("abcabcabc", 2);
    BOOST_REQUIRE_EQUAL(rw.freq("ab"), 3);
    BOOST_REQUIRE_EQUAL(rw.freq("bc"), 3);
    BOOST_REQUIRE_EQUAL(rw.freq("ca"), 3);
    BOOST_REQUIRE_EQUAL(rw.freq("ab", 'c'), 3);
    BOOST_REQUIRE_THROW(rw.freq("a"), std::invalid_argument);
    BOOST_REQUIRE_THROW(rw.freq("abc"), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(TestKRand) {
    RandWriter rw("abcabcabc", 2);
    BOOST_REQUIRE_NO_THROW(rw.kRand("ab"));
    BOOST_REQUIRE_THROW(rw.kRand("xy"), std::invalid_argument);
```

```
38      BOOST_REQUIRE_THROW(rw.kRand("a"), std::invalid_argument);
39  }
40
41  BOOST_AUTO_TEST_CASE(TestGenerate) {
42      RandWriter rw("abcabcabc", 2);
43      std::string generated = rw.generate("ab", 10);
44      BOOST_REQUIRE_EQUAL(generated.length(), 10);
45      BOOST_REQUIRE_EQUAL(generated.substr(0, 2), "ab");
46      BOOST_REQUIRE_THROW(rw.generate("ab", 1), std::invalid_argument);
47      BOOST_REQUIRE_THROW(rw.generate("a", 5), std::invalid_argument);
48  }
```

7.7.6 Input:

```
1  ./TextWriter 2 11 <  input17.txt
```

7.7.7 Output:

```
1  gaggaagagag
```

8 PS7: Kronos Log Parsing

8.1 Overview

In this project, I have implemented a program to parse and analyze log files from the Kronos InTouch device, which is a Linux-based touch-screen time clock. The main aim is to identify device startup sequences, and determine whether they were successful or incomplete, and calculate the duration of each boot process

8.2 What I Accomplished

1. I have implemented a program in C++ to parse Kronos InTouch logs and analyze boot-up data.
2. I have used regular expressions to identify boot start and completion events.
3. I have calculated boot durations using timestamps and the Boost Date-Time library.
4. I have handled edge cases like incomplete or nested boot sequences and ensured these were included in the final report.
5. I have generated structured output reports that summarized key boot details.

8.3 What I Learned

1. I have learned how to analyze and process real-world log files to extract meaningful insights.
2. I have improved my understanding of regular expressions and their application in parsing structured text.
3. I have also gained the experience using the Boost Date-Time library for precise time calculations.

8.4 Discussion of Key Algorithms

Data Structures:
1. `std::vector<std::pair<std::string, std::string»`
To store matched log entries as pairs of timestamps and event types ("Boot Start" or "Boot Complete").
2. `std::unordered_map<int, std::vector<std::string»`
It is used for associating sequence IDs with corresponding log events, facilitating organized and efficient data retrieval.
3. std::tm
It is used for parsing and storing timestamp details extracted from log entries.
4. std::regex
It is used for defining and matching patterns in the log data to identify relevant events
Algorithms:
1. Log Parsing
I have implemented a regular expression-based on the approach to match specific patterns in log entries.
2. Output Formatting

==>I have formatted the final output into a structured report with clear sections:
i.) Boot sequence ID.
ii.) Start and completion timestamps.
iii.) Boot duration in milliseconds.
iv.) Incomplete boots were categorized in a separate section for clarity.
Object Oriented Designs:
==>Class: The BootRecord class is the key part of the project, as it encapsulates all the details about each boot event.
==>Encapsulation:
The BootRecord class encapsulates all the data for each boot event.

8.5 What I Already Knew

==>I was already familiar with some basic file handling operations in C++ for reading and writing data.

8.6 Challenges

==>I have faced difficulty in Processing large log files efficiently.
==>I have faced difficulty in Parsing timestamps and calculating durations accurately .
==>I have faced difficulty in identifying and processing logs for incomplete boots .

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 $@
```

```
21
22 # Linting (optional, use a tool like clang-tidy if available)
23 lint:
24     clang-tidy $(SRC)
25
26 # Clean up generated files
27 clean:
28     rm -f $(OBJ) $(EXE) *.rpt
```

8.7.2 ps7.cpp

```
1  // Copyright [2024] Ashish Kosana
2  #include <iostream>
3  #include <fstream>
4  #include <regex>
5  #include <string>
6  #include <vector>
7  #include <chrono>
8  #include <iomanip>
9  #include <sstream>
10
11 struct BootRecord {
12     int startLine;
13     int endLine;
14     std::string startTime;
15     std::string endTime;
16     bool completed;
17     int durationMs;
18 };
19
20 std::regex startRegex(R"(\(log\.c\.166\) server started)");
21 std::regex endRegex(R"(oejs\.AbstractConnector:Started .+)");
22
23 int calculateDurationMs(const std::string& start, const std::string& end) {
24     std::tm tmStart = {}, tmEnd = {};
25     std::istringstream ssStart(start);
26     std::istringstream ssEnd(end);
27     ssStart >> std::get_time(&tmStart, "%Y-%m-%d %H:%M:%S");
28     ssEnd >> std::get_time(&tmEnd, "%Y-%m-%d %H:%M:%S");
29
30     auto startTime = std::chrono::
31     system_clock::from_time_t(std::mktime(&tmStart));
32     auto endTime = std::chrono::
33     system_clock::from_time_t(std::mktime(&tmEnd));
```

```
    return std::chrono::duration_cast<std::chrono::milliseconds>(
        endTime - startTime).count();
}

std::vector<BootRecord> parseLogFile(const std::string& filename) {
    std::vector<BootRecord> bootRecords;
    std::ifstream file(filename);
    std::string line;
    int lineNumber = 0;
    BootRecord currentBoot;
    bool inBoot = false;

    if (!file.is_open()) {
        std::cerr << "Error: Unable to open file " << filename << std::endl;
        return bootRecords;
    }

    while (std::getline(file, line)) {
        lineNumber++;
        std::smatch match;

        if (std::regex_search(line, match, startRegex)) {
            if (inBoot) {
                currentBoot.completed = false;
                bootRecords.push_back(currentBoot);
            }
            inBoot = true;
            currentBoot = BootRecord{lineNumber, 0,
            line.substr(0, 19), "", false, 0};
        } else if (inBoot && std::regex_search(line, match, endRegex)) {
            currentBoot.endLine = lineNumber;
            currentBoot.endTime = line.substr(0, 19);
            currentBoot.completed = true;
            currentBoot.durationMs = calculateDurationMs(
            currentBoot.startTime, currentBoot.endTime);
            bootRecords.push_back(currentBoot);
            inBoot = false;
        }
    }

    if (inBoot) {
        currentBoot.completed = false;
        bootRecords.push_back(currentBoot);
    }

```

```
    file.close();
    return bootRecords;
}

void generateReport(const std::string&
inputFile, const std::string& outputFile,
                    const std::vector<BootRecord>& bootRecords) {
    std::ofstream out(outputFile);
    if (!out.is_open()) {
        std::cerr << "Error: Unable to create output file "
        << outputFile << std::endl;
        return;
    }

    out << "Device Boot Report" << std::endl;
    out << "InTouch log file: " << inputFile << std::endl;
    out << "Summary:" << std::endl << std::endl;

    for (const auto& record : bootRecords) {
        out << "Device Boot - " << std::endl;
        out << record.startLine << "(" << inputFile << "): " << record.
   startTime
            << " Boot Start" << std::endl;
        if (record.completed) {
            out << record.endLine << "(" << inputFile << "): " << record.
   endTime
                << " Boot Completed\t   Time: " << record.durationMs << "ms"
                << std::endl;
        } else {
            out << "Boot Incomplete" << std::endl;
        }
        out << std::endl;
    }

    out.close();
    std::cout << "Report generated successfully in: "
    << outputFile << std::endl;
}

std::vector<std::string> getEndTimes(
    const std::vector<BootRecord>& records) {
    std::vector<std::string> endTimes;
    for (const auto& record : records) {
        if (!record.completed) {
            endTimes.push_back(record.startTime);
```

```
        }
    }
    return endTimes;
}

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

    std::string inputFile = argv[1];
    std::string outputFile = inputFile + ".rpt";

    auto bootRecords = parseLogFile(inputFile);
    generateReport(inputFile, outputFile, bootRecords);

    auto failedBoots = getEndTimes(bootRecords);
    std::cout << "Failed Boots: " << failedBoots.size() << std::endl;
    for (const auto& time : failedBoots) {
        std::cout << time << std::endl;
    }

    return 0;
}
```

8.7.3 Output:

```
Device Boot Report
InTouch log file: device1_intouch.log
Summary:

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    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    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    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    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    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    Time:
    161000ms
```