Computing IV Sec 011: Project Portfolio

Yashwanth Reddy Surapareddy

Summer 2023

Contents

TIME TO COMPLETE PORTFOLIO : 13 Hours

1 PS0: Hello SFML

1.1 Discussion:

Hello World using SFML is CompIV's first project. We initially prepared our environment for this task by installing the SFML library. We also look for the most recent C++ version. As a result, we test the provided code to see if it functions properly. In the final section of the assignment, we include an image and make use of the SFML sprite attribute to move the picture around using the keyboard's keys. The output is in **Figure: 4**

1.2 Key algorithms, Data structures and OO Designs used in this Assignment:

As the project itself is a basic project using the code provided, this assignment did not call for any of the important algorithms, data structures, or OO designs. Instead, we only needed to add SFML sprites and a few keyboard events to finish the project.

1.3 Images used:

Figure 1: Sprite Image-1

Figure 2: Sprite Image-2

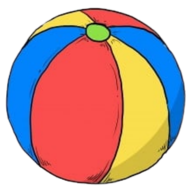


Figure 3: Sprite Image-3

1.4 What I accomplished :

I inserted 3 images where the two images are dogs which are placed in the bottom two corners and a ball which can be moved using the keyboard clicks. Use A, D, W, X to move the ball from one corner to other between the dogs.

1.5 What I already knew :

I was aware of how to move the image using keyboard keys and also mouse pointer events as I already learnt Javascript. So, It was much easy to do. I know SFML is different than JavaScript but the concept of it is similar to me.

1.6 What I learned :

For the first time, I learned how to utilize SFML, as well as how to create events and work with them in the SFML field. Overall, I enjoyed working on this assignment because it was full of amazing and novel things for me.

1.7 Challenges :

Using the SFML was difficult in the beginning, but tried to understand it for some point.

1.8 Codebase

Makefile:
This Makefile was provided in portal.

```
CC = g++
CFLAGS = --std=c++14 -Wall -Werror -pedantic
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system

all: sfml-app

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

sfml-app: main.o
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o sfml-app
```

main.cpp:
The main file where the code executes and provides the output as displayed in figure 2.

```
#include<stdio.h>
#include <SFML/Graphics.hpp>
int movement_x(sf::Sprite sprite2)
{       sf::Vector2f pos = sprite2.getPosition();
    float offX=0;
    float a;
if(sf::Keyboard::isKeyPressed(sf::Keyboard::A) && pos.x!=100)
        offX=-1;
        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::D) && pos.x!=700)
        offX=1;
 a = pos.x + offX;
 return a;
}
int movement_y(sf::Sprite sprite2)
{
 sf::Vector2f pos = sprite2.getPosition();
 float offY=0;
 float b;
if(sf::Keyboard::isKeyPressed(sf::Keyboard::W) && pos.y!=100)
        offY=-1;
        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::X) && pos.y!=700)
        offY=1;
```

```cpp
 b = pos.y + offY;
 return b;
}

int main()
{
    sf::RenderWindow window(sf::VideoMode(1000, 1000), "Yashwanth Reddy");

   sf::Texture t;
    if(!t.loadFromFile("sprite.png"))
    return EXIT_FAILURE;
    sf::Sprite sprite(t);

    sf::Texture t1;
    if(!t1.loadFromFile("sprite1.png")){
    return EXIT_FAILURE;}
    sf::Sprite sprite1(t1);

    //setting dog images in bottom corners
    sf::Vector2f Size(200.0f,200.0f);
    sprite.setPosition(0.0f, window.getSize().y -Size.y);
    sprite1.setPosition(window.getSize().x -Size.x, window.getSize().y -
   Size.y);
    //making dog images to be in same size
    sprite.setScale(Size.x/ sprite.getLocalBounds().width, Size.y/ sprite.
   getLocalBounds().height);
    sprite1.setScale(Size.x/ sprite1.getLocalBounds().width, Size.y/ sprite1
   .getLocalBounds().height);
    sf::Texture t2;
    if(!t2.loadFromFile("sprite2.png"))
    return EXIT_FAILURE;
    sf::Sprite sprite2(t2);
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Yellow);
    shape.setPosition(0,0);
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        float move_x , move_y;
    move_x = movement_x(sprite2);
    move_y = movement_y(sprite2);

        sprite2.setPosition(move_x, move_y);
        window.clear(sf::Color::Cyan);
        window.draw(shape);
        window.draw(sprite);
        window.draw(sprite1);
        window.draw(sprite2);
        window.display();
    }

    return 0;
}
```

1.9 Output:

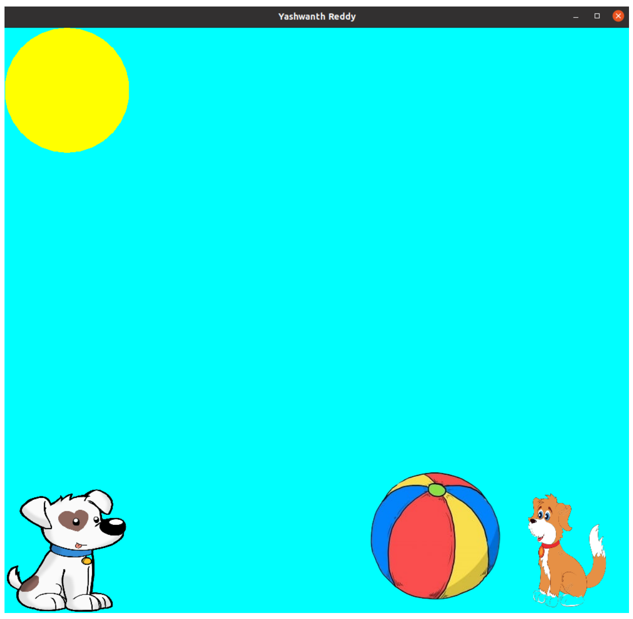


Figure 4: Output of PS0 Assignment

2 PS1a: Linear Feedback Shift Register

2.1 Discussion:

The LFSR (Linear Feedback shift Register), which is the Fibonacci LFSR, is implemented in the ps1a assignment. This homework assignment pertains to PhotoMagic, in PS1B. There are two primary functions in this project: step() and generate(). The result of the tap places is the lsb together with the one bit of the provided seed, which is left shifted using the step() function. These tap positions employ XOR processes, which ultimately produce a lsb outcome. The generate() function creates the states based on the k inputs that are provided.
XOR Truth Table:

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

2.2 Key algorithms, Data structures and OO Designs used in this Assignment:

I have used string for the seed. I used simple string functions to implement the step function and generate function where the generate function takes the 'k' value.

The Tap position algorithm is as follows:

```
int b10 = static_cast<int>(s.at(s.length() - 11)) - 48;
int b12 = static_cast<int>(s.at(s.length() - 13)) - 48;
int b13 = static_cast<int>(s.at(s.length() - 14)) - 48;
int bi = static_cast<int>(s.at(0)) - 48;
int i = 0;
while (i < static_cast<int>(s.length()) - 1) {
s[i] = s[i + 1];
i++;
}
s[s.length() - 1] = static_cast<char>(((bi ^ b13) ^ b12) ^ (b10 + 48));
return static_cast<int>(s[s.length() - 1]) - 48;

```

2.3 What I accomplished :

I accomplished the full work of the LFSR and also both the important functions step() as well as generate() works completely fine. I learnt how to lint the program by installing the linting tool in my ubuntu.

2.4 What I already knew :

Due to my completion of Digital Logic Design and Computer Architecture, I was familiar with the XOR idea, as well as how to shift bits and the meaning of the MSB and LSB.

2.5 What I learned :

I learned that how this LFSR is going to be used for the Image encoding and decoding of the Image in further part of the PS1 code. Where we utilize this part of assignment.

2.6 Challenges :

Because I was unfamiliar with the idea of the Boost Library, it was difficult for me to produce the LSB bit from the tap places and reattach it to the seed. It was also difficult for me to write the new test case.

2.7 Codebase

Makefile:

This Makefile is created by the referrence of the Version2 Makefile from the notes.

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

OBJECTS= FibLFSR.o

all: ps1 test

ps1: main.o $(OBJECTS)
    $(CC) main.o $(OBJECTS) -o ps1 $(LIB)

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

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


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

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


clean:
    rm *.o
    rm ps1
    rm test
```

FibLFSR.hpp:

This file serves as the header for the "FibLFSR.cpp" file that was previously mentioned. The initialization of the functions, libraries, and variables is contained in this file.

```
// copyright 2023 YashwanthReddy
#pragma once
#include <iostream>
#include <string>
class FibLFSR {
 public:
    explicit FibLFSR(std::string seeds);
    int step();
    int generate(int k);
    friend std::ostream& operator<< (std::ostream &out, FibLFSR &cFibLFSR);
 private:
    std::string s;
};
```

FibLFSR.cpp:
This file contains the important methods such as step() and generate().

```cpp
// copyright 2023 YashwanthReddy
#include "FibLFSR.hpp"
#include <iostream>
#include <cmath>
#include <sstream>
FibLFSR::FibLFSR(std::string seeds) {
s = seeds;
}
int FibLFSR::step() {
int b10 = static_cast<int>(s.at(s.length() - 11)) - 48;
int b12 = static_cast<int>(s.at(s.length() - 13)) - 48;
int b13 = static_cast<int>(s.at(s.length() - 14)) - 48;
int bi = static_cast<int>(s.at(0)) - 48;
int i = 0;
while (i < static_cast<int>(s.length()) - 1) {
s[i] = s[i + 1];
i++;
}
s[s.length() - 1] = static_cast<char>(((bi ^ b13) ^ b12) ^ (b10 + 48));
return static_cast<int>(s[s.length() - 1]) - 48;
}
int FibLFSR::generate(int k) {
int ex = 0;
for (int i = k - 1; i >= 0; i--) {
int st = step();
if (st == 1) {
ex = ex + static_cast<int>(pow(2, i));
}
}
return ex;
}
std::ostream& operator<<(std::ostream& out, FibLFSR& cFibLFSR) {
out << cFibLFSR.s;
return out;
}
```

test.cpp:
This file is the test file which utilizes the <Boost Library>
my 4 test cases are as follows:

- **At first case seed is 1011011000110110**
- **At second case seed is 0111001000110111**
- **At third case seed is 1100011011000011**
- **At fourth case seed is 0011011000110111**

```cpp
// copyright 2023 YashwanthReddy
#include <iostream>
#include <string>
#include "FibLFSR.hpp"
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
BOOST_AUTO_TEST_CASE(case1) {
FibLFSR l("1011011000110110");
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
FibLFSR l2("1011011000110110");
BOOST_REQUIRE(l2.generate(9) == 51);
}
BOOST_AUTO_TEST_CASE(case2) {
FibLFSR l("0111001000110111");
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
FibLFSR l2("1011011000110111");
BOOST_REQUIRE(l2.generate(5) == 3);
}
BOOST_AUTO_TEST_CASE(case3) {
FibLFSR l("1100011011000011");
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
FibLFSR l2("1100011011000011");
BOOST_REQUIRE(l2.generate(5) == 6);
}
BOOST_AUTO_TEST_CASE(case4) {
FibLFSR l("0011011000110111");
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
FibLFSR l2("0011011000110111");
BOOST_REQUIRE(l2.generate(7) == 76);
}
```

3 PS1b: PhotoMagic

3.1 Discussion:

The PS1B makes use of the PS1A's LFSR: LINEAR FEEDBACK SHIFT REGISTER to facilitate the encoding and decoding of the input image.This project encrypts each and every pixel of the input image, input-file.png, using the LFSR Randomizer, then saves the result to the output-file.png file. For Encryption I used Common Example i.e;

./PhotoMagic input-file.png output-file.png 0011001100000
Output is in: Figure:16

If we want to decrypt the encrypted image we write it as follows.
./PhotoMagic output-file.png input-file.png 0011001100000
Output is in: Figure: 7

3.2 Key algorithms, Data structures and OO Designs used in this Assignment:

I used void transform function which takes the image file and the bit generator, it converts all the pixels one by one.

```
void transform( sf::Image& image, FibLFSR* bit_generator) {
sf::Vector2u imagesize = image.getSize();
sf::Color p;
int x = 0;
while(x < (signed)imagesize.x) {
for(int y = 0; y < (signed)imagesize.y; y++) {
p = image.getPixel(x, y);
p.r = p.r ^ bit_generator -> generate(8);
p.g = p.g ^ bit_generator -> generate(8);
p.b = p.b ^ bit_generator -> generate(8);
image.setPixel(x, y, p);
}
x++;
}
}

```

3.3 Images used:

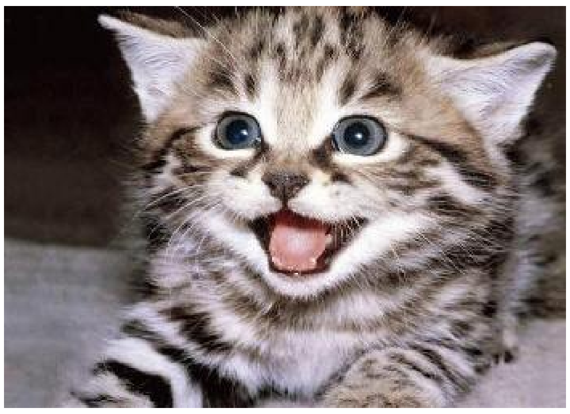


Figure 5: Image Considered as Input

3.4 What I accomplished :

I used the C++ programming language to encode and decode the image with success. I was able to accomplish something spectacular by using many windows.

3.5 What I already knew :

I already knew how to use images and textures to make sprites. I was able to make the project's Makefile and display the windows.

3.6 What I learned :

I discovered how to use two windows for various outputs. I was able to understand the idea of encoding and decoding utilizing the image's pixels and the LFSR Randomizer. Additionally, I learned about the mathematical processes used to create the pixels.

3.7 Challenges :

To find out the image pixels and the also to identify how to tranform them to the normal to encoded and then decode the image.

3.8 Codebase

Makefile:
This Makefile is extension of the ps1a Makefile.

```
CC= g++
CFLAGS= -Wall -Werror -ansi -pedantic
SFMLFLAGS= -lsfml-graphics -lsfml-window -lsfml-system

all:    PhotoMagic test

PhotoMagic: PhotoMagic.o FibLFSR.o
    $(CC) PhotoMagic.o FibLFSR.o -o PhotoMagic $(SFMLFLAGS)

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

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

clean:
    rm *.o
    rm PhotoMagic
    rm *.hpp.gch
```

FibLFSR.hpp:
This file serves as the header for the "FibLFSR.cpp" file.

```cpp
// copyright 2023 YashwanthReddy
#pragma once
#include <iostream>
#include <string>
class FibLFSR {
 public:
    explicit FibLFSR(std::string seeds);
    int step();
    int generate(int k);
    friend std::ostream& operator<< (std::ostream &out, FibLFSR &cFibLFSR);
 private:
    std::string s;
};
```

FibLFSR.cpp:
This file contains the function declarations such as step() and generate().

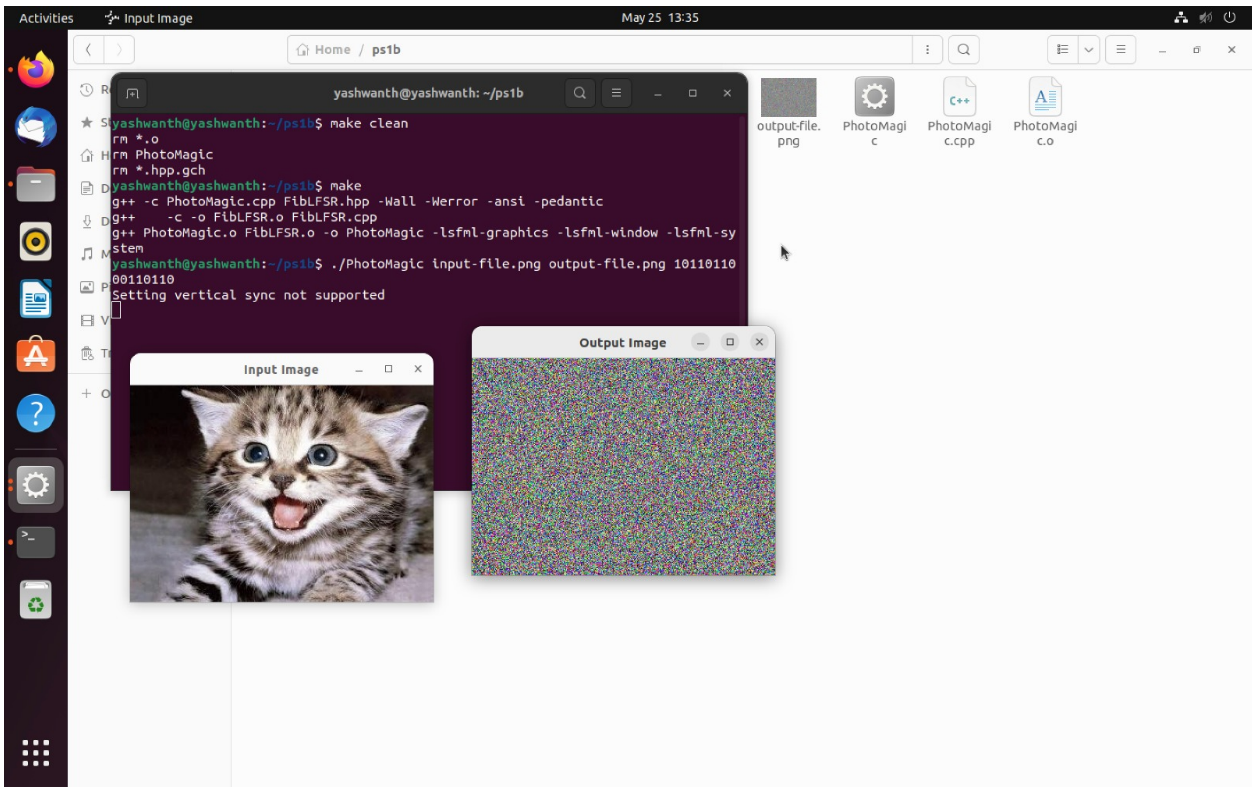
```
// copyright 2023 YashwanthReddy
#include "FibLFSR.hpp"
#include <iostream>
#include <cmath>
#include <sstream>
FibLFSR::FibLFSR(std::string seeds) {
s = seeds;
}
int FibLFSR::step() {
int b10 = static_cast<int>(s.at(s.length() - 11)) - 48;
int b12 = static_cast<int>(s.at(s.length() - 13)) - 48;
int b13 = static_cast<int>(s.at(s.length() - 14)) - 48;
int bi = static_cast<int>(s.at(0)) - 48;
int i = 0;
while (i < static_cast<int>(s.length()) - 1) {
s[i] = s[i + 1];
i++;
}
s[s.length() - 1] = static_cast<char>(((bi ^ b13) ^ b12) ^ (b10 + 48));
return static_cast<int>(s[s.length() - 1]) - 48;
}
int FibLFSR::generate(int k) {
int ex = 0;
for (int i = k - 1; i >= 0; i--) {
int st = step();
if (st == 1) {
ex = ex + static_cast<int>(pow(2, i));
}
}
return ex;
}
std::ostream& operator<<(std::ostream& out, FibLFSR& cFibLFSR) {
out << cFibLFSR.s;
return out;
}
```

PhotoMagic.cpp:
The primary file in which the reading, writing, encoding, and decoding of the image occur is this one. The output is presented in two windows by this file. the file's input and output windows.

```
#include <iostream>
#include <string>
#include <sstream>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"
void transform( sf::Image& image, FibLFSR* bit_generator) {
sf::Vector2u imagesize = image.getSize();
sf::Color p;
int x = 0;
while(x < (signed)imagesize.x) {
for(int y = 0; y < (signed)imagesize.y; y++) {
p = image.getPixel(x, y);
p.r = p.r ^ bit_generator -> generate(8);
p.g = p.g ^ bit_generator -> generate(8);
p.b = p.b ^ bit_generator -> generate(8);
image.setPixel(x, y, p);
}
x++;
}
}
std::string h_password(std::string stx) {
int h = 33;
int slen = stx.length();
int i = 0;
while(i < slen){
h = h ^ stx[i];
h*= h;
i++;
}
std::string b;
while(h != 0) {
b = (h % 2 == 0 ? "0" : "1") + b;
h /= 2;
}
return b;
}
int main(int argc, char* argv[]) {
if(argc != 4) {
std::cout << "Input: ./PhotoMagic <inputfilename> <outputfilename> <seed>\n"
    ;
return -1;
}
std::string input_fname(argv[1]);
std::string output_fname(argv[2]);
std::string password(argv[3]);
std::string seed = h_password(password);
FibLFSR bit_generator(seed);
sf::Image input_image;
if (!input_image.loadFromFile(input_fname)) {
return -1;
}
sf::Image output_image;
if (!output_image.loadFromFile(input_fname)) {
return -1;
}
sf::Vector2u imagesize = input_image.getSize();
sf::RenderWindow input_window(sf::VideoMode(imagesize.x, imagesize.y), "
    Input Image");
sf::RenderWindow output_window(sf::VideoMode(imagesize.x, imagesize.y), "
    Output Image");
sf::Texture in_texture, out_texture;
in_texture.loadFromImage(input_image);
transform(input_image, &bit_generator);
out_texture.loadFromImage(input_image);
sf::Sprite in_sprite, out_sprite;
in_sprite.setTexture(in_texture);
out_sprite.setTexture(out_texture);
while (input_window.isOpen() && output_window.isOpen()) {
sf::Event event;
while (input_window.pollEvent(event)) {
if(event.type == sf::Event::Closed) {
input_window.close();
}
}
```

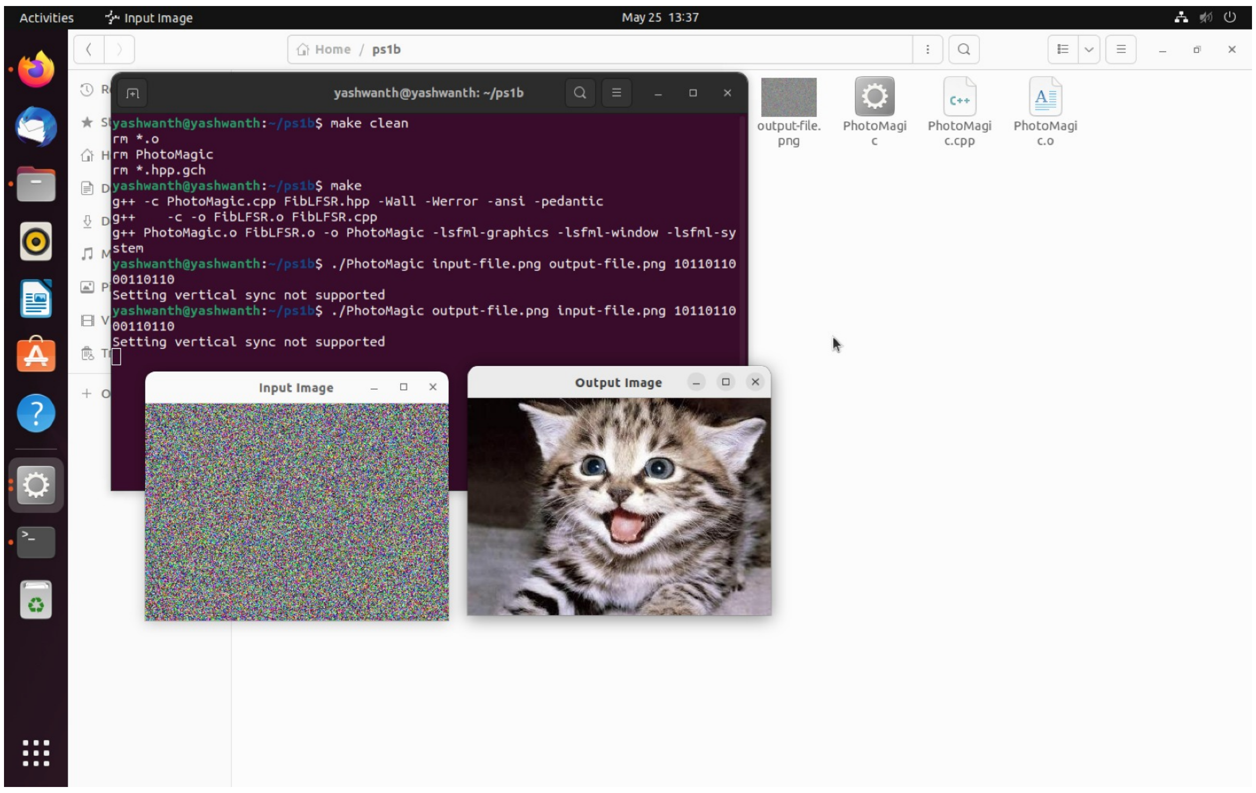
```
while (output_window.pollEvent(event)) {
if (event.type == sf::Event::Closed) {
output_window.close();
}
}
input_window.clear();
input_window.draw(in_sprite);
input_window.display();
output_window.clear();
output_window.draw(out_sprite);
output_window.display();
}
if (!input_image.saveToFile(output_fname)) {
return -1;
}
return 0;
}
```

3.9 Output:



```
yashwanth@yashwanth:~/ps1b$ make clean
rm *.o
rm PhotoMagic
rm *.hpp.gch
yashwanth@yashwanth:~/ps1b$ make
g++ -c PhotoMagic.cpp FibLFSR.hpp -Wall -Werror -ansi -pedantic
g++    -c -o FibLFSR.o FibLFSR.cpp
g++ PhotoMagic.o FibLFSR.o -o PhotoMagic -lsfml-graphics -lsfml-window -lsfml-sy
stem
yashwanth@yashwanth:~/ps1b$ ./PhotoMagic input-file.png output-file.png 10110110
00110110
Setting vertical sync not supported
```

Figure 6: Encoded Image



```
yashwanth@yashwanth:~/ps1b$ make clean
rm *.o
rm PhotoMagic
rm *.hpp.gch
yashwanth@yashwanth:~/ps1b$ make
g++ -c PhotoMagic.cpp FibLFSR.hpp -Wall -Werror -ansi -pedantic
g++    -c -o FibLFSR.o FibLFSR.cpp
g++ PhotoMagic.o FibLFSR.o -o PhotoMagic -lsfml-graphics -lsfml-window -lsfml-sy
stem
yashwanth@yashwanth:~/ps1b$ ./PhotoMagic input-file.png output-file.png 10110110
00110110
Setting vertical sync not supported
yashwanth@yashwanth:~/ps1b$ ./PhotoMagic output-file.png input-file.png 10110110
00110110
Setting vertical sync not supported
```

Figure 7: Decoded Image

4 PS2: Recursive Graphics (Pythagoras Tree)

4.1 Discussion:

Because each triple of the contacting squares encloses a right triangle in a configuration that is typically used to show the Pythagorean theorem, the Pythagoras tree is named after the Greek mathematician Pythagoras. The square-based plane fractal was created by Dutch math instructor Albert E. Bosman. In this Assignment, I created a constructor of the PTree class. It takes a reference to an

sf::RenderWindow object and initializes the window member variable with it. And the draw function of the PTree class is the recursive function that draws a Pythagoras tree fractal. A sf::RectangleShape object named square is created with dimensions l x l and positioned at pos.

The square is styled with a black fill color, a 3-pixel green outline, and drawn on the window. The draw function is called recursively twice to draw the sub-trees. The angles are adjusted by adding or subtracting 45 degrees, and the depth is decreased by 1. This recursive process continues until the depth reaches 0, and the function returns, completing the drawing of the Pythagoras tree fractal. **The command Line is as Follows: ./PTree**

4.2 Key algorithms, Data structures and OO Designs used in this Assignment:

The code utilizes the sf::RectangleShape class to create and draw rectangles representing the segments of the Pythagoras tree. The square object is an instance of sf::RectangleShape and is styled and drawn using its member functions (setFillColor, setOutlineThickness, setOutlineColor, and setPosition). The draw function is recursively called twice with updated parameters: one call with an angle of ang - 45.0f and another call with an angle of ang + 45.0f. Both calls have a decreased depth value of depth - 1 and use the new length newLen and position newPos.

Recursive method used in PTree:

```
draw(newPos, newLen, ang - 45.0f, depth - 1, scale);
draw(newPos, newLen, ang + 45.0f, depth - 1, scale);

```

4.3 What I accomplished :

Using recursive approaches, I was able to create the PFractal. The use of recursion to construct patterns was truly amazing and efficient. In order to get extra credit, I added colors.

4.4 What I already knew :

I already knew how to create a window and few events. I also knew how to add background to the window and set the window size. I also knew how to take command line arguments.

4.5 What I learned :

I discovered a more effective way to implement the sf::Drawable class. I also learned how to create stunning patterns using recursive techniques and mathematical formulas.

4.6 Challenges :

To find out the exact math formula and align the triangle to the center was the challenging part and I was unable to sort it.

4.7 Codebase

Makefile:

This file combines and executes after checking the errors

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

.PHONY: all clean lint

all: $(PROGRAM)

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

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

clean:
	rm *.o $(PROGRAM)

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

main.cpp

This file is important as the Recursion takes place as well as the window is drawn. It calls the draw for creating the squares until unless the depth becomes zero.

```
// copyright 2023 YashwanthReddy
#include <SFML/Graphics.hpp>
#include "PTree.hpp"
const int w = 1000;
const int h = 800;
int main() {
sf::RenderWindow window(sf::VideoMode(w, h), "Yashwanth's Square Fractal");
window.setFramerateLimit(100);
PTree g(window);
g.generate(w / 2, h, 200.0f, -90.0f, 10, 0.7f);
return 0;
}
void PTree::generate(int x, int y, float length, float angle, int depth,
    float scale) {
while (window_.isOpen()) {
sf::Event event;
while (window_.pollEvent(event)) {
if(event.type == sf::Event::Closed) {
window_.close();
}
}
window_.clear(sf::Color::White);
draw(sf::Vector2f(x, y), length, angle, depth, scale);
window_.display();
}
}
```

PTree.hpp

It is a header file where the initialization of libraries as well as variables and methods takes place.

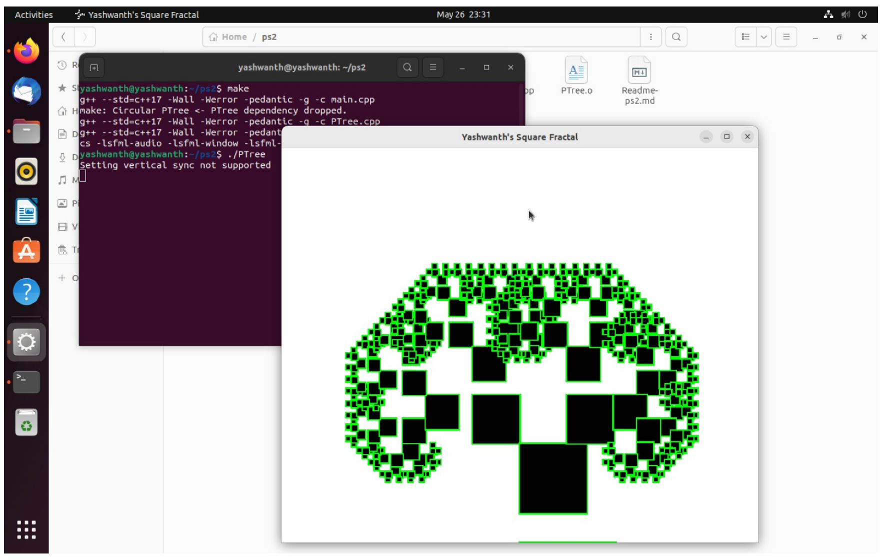
```cpp
// copyright 2023 YashwanthReddy
#ifndef PTREE_HPP
#define PTREE_HPP
#include <SFML/Graphics.hpp>
class PTree {
 public:
    PTree(sf::RenderWindow& window);
    void generate(int x, int y, float length, float angle, int depth, float
   scale);

 private:
    void draw(sf::Vector2f pos, float l, float ang, int depth, float scale);
    sf::RenderWindow& window_;
};
#endif
```

PTree.cpp

This file uses the SFML library to create the PTree.

```cpp
// copyright 2023 YashwanthReddy
#include "PTree.hpp"
#include <cmath>
#include <SFML/Graphics.hpp>
PTree::PTree(sf::RenderWindow& window) : window_(window) {}
void PTree::draw(sf::Vector2f pos, float l, float ang, int depth, float
   scale) {
if (depth == 0) {
return;
}
float angle_ = ang * (M_PI / 180.0f);
float endX = pos.x + l * std::cos(angle_);
float endY = pos.y + l * std::sin(angle_);
sf::RectangleShape square(sf::Vector2f(l, l));
square.setPosition(pos);
square.setFillColor(sf::Color::Black);
square.setOutlineThickness(3.0f);
square.setOutlineColor(sf::Color::Green);
window_.draw(square);
float newLen = l * scale;
sf::Vector2f newPos = sf::Vector2f(endX, endY);
draw(newPos, newLen, ang - 45.0f, depth - 1, scale);
draw(newPos, newLen, ang + 45.0f, depth - 1, scale);
}
```

4.8 Output:


Activities
Yashwanth's Square Fractal
May 26 23:31
Home / ps2
yashwanth@yashwanth: ~/ps2
PTree.o
Readme-ps2.md
Yashwanth's Square Fractal

```
yashwanth@yashwanth:~/ps2$ make
g++ --std=c++17 -Wall -Werror -pedantic -g -c main.cpp
make: Circular PTree <- PTree dependency dropped.
g++ --std=c++17 -Wall -Werror -pedantic -g -c PTree.cpp
g++ --std=c++17 -Wall -Werror -pedant
cs -lsfml-audio -lsfml-window -lsfml-
yashwanth@yashwanth:~/ps2$ ./PTree
Setting vertical sync not supported
```

Figure 8: PTree Output

5 PS3a: N-Body Simulation (Static)

5.1 Discussion:

We have completed our formal schooling, which included lessons on solar systems. We questioned how these planets were animated to appear on the screen. These tasks are designed to teach students how to produce animation, so that should help with your questions. In the first section, we will make a static animation of the solar system, and in the second, we will use physics to simulate the motion of the static bodies.

The **PS3a** is to load and display images of the planets. CelestialBody and Universe are the two classes we are employing. Any planet depicted in the cosmos can be seen using the class CelestialBody, for instance our planet Earth. The CelestialBody vector is part of the Universe class, and using the vector, a function to generate the complete universe is available.

We use the provided planets.txt file for this assignment:
Inputs: 5
Radius: 2.50e+11

x-axis	y-axis	velocity of x-axis	velocity of y-axis	Mass	filename
1.4960e+1	0.0000e+00	0.0000e+00	2.9800e+04	5.9740e+24	earth.gif
2.2790e+11	0.0000e+00	0.0000e+00	2.4100e+04	6.4190e+2	mars.gif
5.7900e+10	0.0000e+00	0.0000e+00	4.7900e+04	3.3020e+23	mercury.gif
0.0000e+00	0.0000e+00	0.0000e+00	0.0000e+00	1.9890e+30	sun.gif
1.0820e+11	0.0000e+00	0.0000e+00	3.5000e+04	4.8690e+24	venus.gif

5.2 Key algorithms, Data structures and OO Designs used in this Assignment:

I read the data from planets.txt using istream and used ostream for output.puts the planets on display in an SFML window. file input is read using the operator. I did not employ any intelligent pointers. Only General Pointers were used. utilized the Address-operator to manage variables' and objects' addresses. I did not employ any intelligent pointers.

istream code:

```
std::istream& operator>>(std::istream& data, CelestialBody& celestialBody) {
    data >> celestialBody.x >> celestialBody.y >> celestialBody.vx
    >> celestialBody.vy >> celestialBody.mass >> celestialBody.fn;
    celestialBody.calp();
    if (!celestialBody.image.loadFromFile(celestialBody.fn))
        return data;
    celestialBody.texture.loadFromImage(celestialBody.image);
    celestialBody.sprite.setTexture(celestialBody.texture);
    celestialBody.sprite.setPosition(celestialBody.rx, celestialBody.ry);
    return data;
}
```

ostream code:

```
std::ostream& operator<<(std::ostream& os, const CelestialBody&
   celestialBody) {
    os << "CelestialBody: x: " << celestialBody.x << " y: "
    << celestialBody.y << std::endl;
    os << " velx: " << celestialBody.vx << " vely: "
    << celestialBody.vy << std::endl;
    os << " mass: " << celestialBody.mass << " " << std::endl;
    os << "x: " << celestialBody.rx << "     " << celestialBody.ry
    << "     " << celestialBody.fn << std::endl;
    return os;
}
```

5.3 Images used:

Figure 9: Background Image

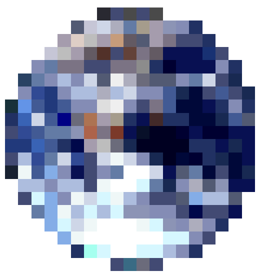


Figure 10: Earth Image

Figure 11: Mars Image

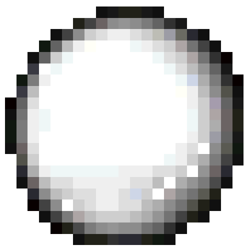


Figure 12: Venus Image

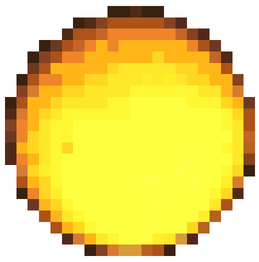


Figure 13: Sun Image

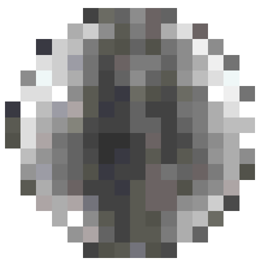


Figure 14: Mercury Image

5.4 What I accomplished :

I accomplished creating a solar system by the given input data. I created my first animated solar system. I am amazed that how physics can be implemented in the code.

5.5 What I already knew :

I knew how to add the background and use of the draw. I also knew how to take input from the file and display. I knew how to create header files and implement them into the cpp files.

5.6 What I learned :

I gained physics knowledge while developing the solar model. I realized how extensively the istream and ostream were used in this assignment. I was aware of the mathematics needed to position the Celestial Bodies. I made good progress in my ability to learn how to use vectors and unitary approaches. Sincerity be damned, I was bad at operator overloading, but after working on this project, I have got it down.

5.7 Challenges :

I was unable to use smart pointers and few algorithm classes, so I kind of felt a difficulty in that.

5.8 Codebase

Makefile

This Makefile has no Linting as the program does not have any lints.

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

OBJECTS = Universe.o CelestialBody.o

all: NBody test

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

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

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

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

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

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

clean:
    rm -f *.o NBody test

lint:
    cpplint *.cpp
```

main.cpp

The main file is important as it is base for creating a window and displaying the CelestialBodies on it.

```cpp
// Copyright 2023 Yashwanthreddy
#include <iostream>
#include <string>
#include <vector>
#include "CelestialBody.hpp"
#include "Universe.hpp"
#include <SFML/Graphics.hpp>
int main() {
unsigned int w = 625;
unsigned int h = 420;
sf::RenderWindow window(sf::VideoMode(w, h), "The Solar System");
window.setFramerateLimit(60);
    std::string n, r;
std::cout << "Enter the number of celestial bodies: ";
std::cin >> n;
std::cout << "Enter the radius of the universe: ";
std::cin >> r;
Universe u(std::stoi(n), std::stof(r));
std::cout << "There are " << n << " celestialBodies" << std::endl;
for (int i = 0; i < u.nop(); i++) {
```

```cpp
CelestialBody* b = new CelestialBody();
b -> width_(w);
b -> height_(h);
b -> setRadius(u.Radius());
std::cout << "Enter details for celestial body " << i + 1 << std::endl;
std::cin >> *b;
u.Body(b);
std::cout << *b;
}
sf::Texture tex;
if (!tex.loadFromFile("image.png")) {
std::cout << "Failed to load background image." << std::endl;
return -1;
}
sf::Sprite sprite;
sprite.setTexture(tex);
while (window.isOpen()) {
sf::Event event;
while (window.pollEvent(event)) {
if (event.type == sf::Event::Closed)
window.close();
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
window.close();
}
window.clear();
window.draw(sprite);
std::vector<CelestialBody*> bs = u.bodies();
for (const auto& b : bs)
window.draw(*b);
window.display();
}
return 0;
}
```

CelestialBody.hpp

The CelestialBody.hpp contains the initializations of istream,ostream and variable and methods for the creation of the Nbodies.

```cpp
// Copyright 2023 Yashwanthreddy
#pragma once
#include <iostream>
#include <string>
#include <SFML/Graphics.hpp>

class CelestialBody : public sf::Drawable {
 public:
    CelestialBody();
    CelestialBody(double x, double y, double vx,
    double vy, double mass, double radius, std::string fn);
    double x, y, rx, ry, vx, vy, mass, radius;
    void calp();
    std::string fn;
    void width_(int w);
    void height_(int h);
    void setRadius(float ur);
    friend std::istream& operator>>(std::istream& input,
    CelestialBody& celestialBody);
    friend std::ostream& operator<<(std::ostream& output,
    const CelestialBody& celestialBody);
 private:
    void draw(sf::RenderTarget& target, sf::RenderStates st) const override;
    int sWidth, sHeight;
    float uRadius;
    sf::Image image;
    sf::Sprite sprite;
    sf::Texture texture;
};
```

CelestialBody.cpp

This is the file where the Celestial body data are taken by the istream and provide an accurate calculation for the position of them as well as provide the data of each CelestialBody on the terminal.

```cpp
// Copyright 2023 Yashwanthreddy
#include "CelestialBody.hpp"
CelestialBody::CelestialBody(double x, double y, double vx, double vy,
               double mass, double radius, std::string fn) {
    this->x = x;
    this->y = y;
    this->vx = vx;
    this->vy = vy;
    this->mass = mass;
    this->radius = radius;
    this->fn = fn;
}

CelestialBody::CelestialBody() {
    return;
}

void CelestialBody::width_(int w) {
    sWidth = w;
}

void CelestialBody::height_(int h) {
    sHeight = h;
}

void CelestialBody::setRadius(float ur) {
    uRadius = ur;
}

std::istream& operator>>(std::istream& data, CelestialBody& celestialBody) {
    data >> celestialBody.x >> celestialBody.y >> celestialBody.vx
    >> celestialBody.vy >> celestialBody.mass >> celestialBody.fn;
    celestialBody.calp();
    if (!celestialBody.image.loadFromFile(celestialBody.fn))
        return data;
    celestialBody.texture.loadFromImage(celestialBody.image);
    celestialBody.sprite.setTexture(celestialBody.texture);
    celestialBody.sprite.setPosition(celestialBody.rx, celestialBody.ry);
    return data;
}

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

void CelestialBody::calp() {
    rx = ((x / uRadius) * (sWidth / 2)) + (sWidth / 2);
    ry = ((y / uRadius) * (sHeight / 2)) + (sHeight / 2);
}

std::ostream& operator<<(std::ostream& os, const CelestialBody&
   celestialBody) {
    os << "CelestialBody: x: " << celestialBody.x << " y: "
    << celestialBody.y << std::endl;
    os << " velx: " << celestialBody.vx << " vely: "
    << celestialBody.vy << std::endl;
    os << " mass: " << celestialBody.mass << " " << std::endl;
    os << "x: " << celestialBody.rx << "     " << celestialBody.ry
    << "     " << celestialBody.fn << std::endl;
    return os;
}
```

Universe.hpp

This header file contains the declarations of few important variables such as numberOFplanets,radius and also methods.

```cpp
// Copyright 2023 Yashwanthreddy
#pragma once
#include <iostream>
#include <vector>
#include "CelestialBody.hpp"
class Universe {
 public:
    Universe();
    Universe(int n, float r);
    int nop() const;
    float Radius() const;
    void setRadius(float radius);
    void Body(CelestialBody* b);
    std::vector<CelestialBody*> bodies() const;

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

 private:
    int numberOfPlanets;
    float radius;
    std::vector<CelestialBody*> bs;
};
```

Universe.cpp

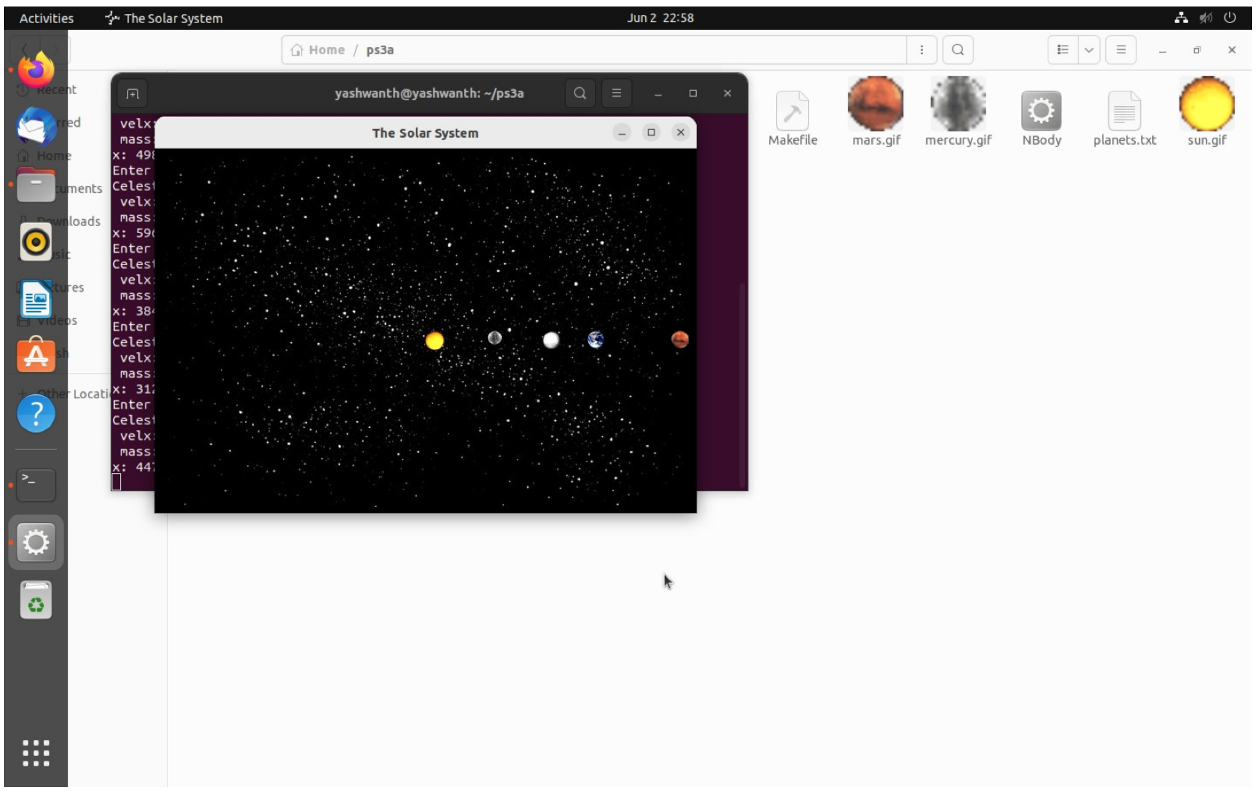
This file sets radius and adds Body to window

```cpp
// Copyright 2023 Yashwanthreddy
#include "Universe.hpp"
Universe::Universe() : numberOfPlanets(0), radius(0.0) {}
Universe::Universe(int n, float r) : numberOfPlanets(n), radius(r) {}
int Universe::nop() const {
    return numberOfPlanets;
}
float Universe::Radius() const {
    return radius;
}
void Universe::setRadius(float radius) {
    this->radius = radius;
}
void Universe::Body(CelestialBody* b) {
    bs.push_back(b);
}
std::vector<CelestialBody*> Universe::bodies() const {
    return bs;
}
std::ostream& operator<<(std::ostream& os, const Universe& uni) {
    os << uni.numberOfPlanets << " " << uni.radius;
    return os;
}
std::istream& operator>>(std::istream& is, Universe& uni) {
    is >> uni.numberOfPlanets >> uni.radius;
    return is;
}
```

test.cpp
The test cases for checking the correctness of CelestialBodies.

```cpp
// copyright 2023 yashwanthreddy
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <boost/test/unit_test.hpp>
#include "Universe.hpp"
#include "CelestialBody.hpp"

BOOST_AUTO_TEST_CASE(test1) {
    Universe universe(5, 2.50e+11);
    BOOST_CHECK_EQUAL(universe.nop(), 5);
    BOOST_CHECK_EQUAL(universe.Radius(), 249999999000);
}

BOOST_AUTO_TEST_CASE(test2) {
CelestialBody b(1.4960e+11, 0.0000e+00, 0.0000e+00,
  2.9800e+04, 5.9740e+24, 6.371e+06, "earth.gif");
    b.width_(625);
    b.height_(420);
    b.setRadius(2.50e+11);
    b.calp();
    BOOST_CHECK_EQUAL(b.vx, 0);
    BOOST_CHECK_EQUAL(b.vy, 29800);
}

BOOST_AUTO_TEST_CASE(test3) {
    CelestialBody b(2.2790e+11, 0.0000e+00, 0.0000e+00,
     2.4100e+04, 6.4190e+23, 3.397e+06, "mars.gif");
    BOOST_CHECK_EQUAL(b.mass, 6.4190e+23);
}

BOOST_AUTO_TEST_CASE(test4) {
    CelestialBody b(5.7900e+10, 0.0000e+00, 0.0000e+00,
     4.7900e+04, 3.3020e+23, 2.4397e+06, "mercury.gif");
    BOOST_CHECK_EQUAL(b.fn, "mercury.gif");
}
```

5.9 Output:



```
 velx
 mass
x: 498
Enter
Celest
 velx
 mass
x: 59
Enter
Celest
 velx
 mass
x: 384
Enter
Celest
 velx
 mass
x: 31
Enter
Celest
 velx
 mass
x: 44
```

Figure 15: NBody Simulation (static)

6 PS3b: N-Body Simulation

6.1 Discussion:

In this project, we are utilizing physics to simulate the solar system that was developed in PS3A. Basic physics formulas for forces such net force, acceleration, and pair-wise forces. The planets revolve around the sun according to the specified simulated time and time step. the detail code by default.
colorboxyellow./NBody 157788000.0 25000.0 planets.txt

6.2 Key algorithms, Data structures and OO Designs used in this Assignment:

The smart pointers I used in this project helped the earlier PS3A code run more smoothly. To really get the right physics formula to run, I reviewed the lecture notes from the class as well as a few physics websites. To manage the memory locations of variables and objects in the CelestialBody and Universe classes, I used smart pointers. For the project's memory management, I also employed the traditional vector. The physical laws were used extensively in the modeling of this project.

6.3 Images used:

The images are already shown in PS3a Sections, To browse them you can click on the following **Figure Numbers:**

- **The Background is Figure 9**
- **The Earth is Figure 10**
- **The Mars is Figure 11**
- **The Venus is Figure 12**
- **The Sun is Figure 13**
- **The Mercury is Figure 14**

6.4 What I accomplished :

I used the C++ SFML package to create a fully animated solar system. Additionally, I added sounds to the window for (extra point).

6.5 What I already knew :

I was previously proficient at reading data from the provided input file. I was well aware of the SFML library's ability to create the background and draw bodies.

6.6 What I learned :

I discovered how physics may be utilized in the realm of computers. It taught me how to apply several physics formulas to various areas of the tasks in this assignment. I learned how to design several solar systems and how to incorporate audio into the SFMl's window by completing the extra credit for this assignment.

6.7 Challenges :

I felt difficult while converting the music from one type to other and later managed to insert the sound correctly.

6.8 Codebase

Makefile

This Makefile has no Linting as the program does not have any lints.

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

OBJECTS = Universe.o CelestialBody.o

all: NBody test

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

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

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

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

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

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

clean:
	rm -f *.o NBody test

lint:
	cpplint *.cpp
```

main.cpp

The main file is crucial since it serves as the foundation for building a window and putting the CelestialBodies on it.Additionally, I have added more functions to this part-b to create a simulation of these celestial bodies. I also added Time Elapse and audio to the SFML window.

```cpp
// copyright 2023 YashwanthReddy
#include <iostream>
#include <fstream>
#include <string>
#include <exception>
#include <vector>
#include "Universe.hpp"
#include "CelestialBody.hpp"
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#define widthwin 512
#define fpswin 30
#define heightwin 512
int main(int argc, char* argv[]) {
if(argc < 2) {
cout << "wrong syntax use the code provided in readme" << endl;
return 1;
}
```

```cpp
float mtime, ctime;
float time_change;
ctime = 0;
mtime = stod(argv[1]);
time_change = stod(argv[2]);
sf::Vector2f centerUniverse{widthwin / 2, heightwin / 2};
Universe solarSystem(centerUniverse);
cin >> solarSystem;
sf::RenderWindow window(sf::VideoMode(widthwin, heightwin), "The Solar
    System");
sf::Texture spaceTextures;
sf::Font sFont;
sf::SoundBuffer buf;
window.setFramerateLimit(fpswin);
if(!spaceTextures.loadFromFile("image.png")) {
throw FileNotFoundException();
cout << "No background image selected" << endl;
}
sf::Sprite spaceBackground(spaceTextures);
spaceBackground.setScale(static_cast<float>
(widthwin) / spaceTextures.getSize().x
, static_cast<float>(heightwin) / spaceTextures.getSize().y);
if(!sFont.loadFromFile("font.ttf")) {
throw FileNotFoundException();       }
if(!buf.loadFromFile("2001.wav")) {
throw FileNotFoundException();
}
sf::Sound sound(buf);
sf::Text timeElapsed{"Time Elapsed: " + to_string(ctime), sFont};
timeElapsed.setPosition(0 , 0);
timeElapsed.setCharacterSize(20);
timeElapsed.setOutlineColor(sf::Color::White);
while(window.isOpen()) {
sf::Event event;
while(window.pollEvent(event)) {
sound.play();
if (event.type == sf::Event::Closed ||
sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
ofstream result;
result.open("output.txt");
result << solarSystem;
result.close();
window.close();
}
}
if (ctime < mtime) {
window.clear();
window.draw(spaceBackground);
for( const auto &p : solarSystem.getBodies() )
window.draw(*p);
window.draw(timeElapsed);
window.display();
solarSystem.step(time_change);
ctime += time_change;
timeElapsed.setString("Time Elapsed: " + to_string(ctime)
+ "Seconds"); }
else
window.setFramerateLimit(60);
}
```

```
return 0;
}
```

CelestialBody.hpp

Initializations for istream, ostream, variables, and methods for creating Nbodies are included in the CelestialBody.hpp file. To create the simulation of the planets, I have also incorporated File eXceptions and more functions in this part-b.

```
// copyright 2023 YashwanthReddy
#pragma once
#include <iostream>
#include <string>
#include <exception>
#include <iomanip>
#include <SFML/Graphics.hpp>
using namespace std;
struct FileNotFoundException : public exception {
const char * what() const noexcept {
return "Can't find file!";
}
};
class CelestialBody : public sf::Drawable {
 public:
    CelestialBody() {}
    CelestialBody(sf::Vector2f iPosition, sf::Vector2f iVelocity,
    float iMass, std::string iImageRef);
    inline float getMass() {
    return mass;
    }
    inline sf::Vector2f getPosition() const {
    return position;
    }
    inline void setPosition(sf::Vector2f nPosition) {
    position = nPosition;
    }
    inline sf::Vector2f getVelocity() const {
    return velocity;
    }
    inline void setVelocity(sf::Vector2f nVelocity) {
    velocity = nVelocity;
    }
    static void createUniverse(sf::Vector2f iCenterUniverse,
    float iRadiusUniverse);
    void spriteUpdate();
    ~CelestialBody() {
    delete sprite_textures;
    }
    friend ostream& operator<<(ostream &out, const CelestialBody& cb);

 private:
    sf::Vector2f position;
    sf::Vector2f velocity;
    float mass;
    string imageRef;
    sf::Sprite sprite;
    sf::Texture* sprite_textures;
    virtual void draw(sf::RenderTarget& rendTarget,
    sf::RenderStates rendStates) const;
    static sf::Vector2f centerUniverse;
    static float radiusUniverse;
};
```

CelestialBody.cpp

This file contains the information about each celestial body on the terminal as well as the data that the istream uses to calculate each celestial body's position accurately.

```
// copyright 2023 PranayaLaxmi
#include "CelestialBody.hpp"
sf::Vector2f CelestialBody::centerUniverse{0, 0};
float CelestialBody::radiusUniverse = 0;
CelestialBody::CelestialBody(sf::Vector2f iPosition,
sf::Vector2f iVelocity, float iMass, string iImageRef) {
mass = iMass;
position = iPosition;
velocity = iVelocity;
imageRef = iImageRef;
sprite_textures = new sf::Texture;
if(!sprite_textures->loadFromFile(imageRef)) {
throw FileNotFoundException();
}
sprite = sf::Sprite(*sprite_textures);
sprite.setOrigin(sprite_textures->getSize().x / 2,
sprite_textures->getSize().y / 2);
spriteUpdate();
}
void CelestialBody::createUniverse(sf::Vector2f iCenterUniverse,
float iRadiusUniverse) {
centerUniverse = iCenterUniverse;
radiusUniverse = iRadiusUniverse;
}
void CelestialBody::spriteUpdate() {
sf::Vector2f sprite_position{position.x / radiusUniverse
*centerUniverse.x + centerUniverse.x,
position.y / radiusUniverse * centerUniverse.y + centerUniverse.y};
sprite.setPosition(sprite_position);
}
void CelestialBody::draw(sf::RenderTarget& rendTarget,
 sf::RenderStates rendStates) const {
rendTarget.draw(sprite, rendStates);
}
ostream& operator<<(ostream &out, const CelestialBody& cb) {\
out.setf(ios_base::scientific);
out << setprecision(4) << left;
out << setw(12) << cb.position.x << setw(12) << cb.position.y
<< setw(12)<< cb.velocity.x <<
setw(12) << cb.velocity.y << setw(12)<< cb.mass << right
<< setw(12) << cb.imageRef;
out.unsetf(ios_base::scientific);
return out;
}
```

Universe.hpp

This header file includes the declarations for a few key variables, including methods, numberOFplanets, and radius. We added istream and ostream to it in part-b.

```
// copyright 2023 YashwanthReddy
#pragma once
#include <iostream>
#include <cmath>
#include <string>
```

```cpp
#include <vector>
#include <memory>
#include "CelestialBody.hpp"
#include <SFML/Graphics.hpp>
using namespace std;
class Universe {
 public:
    Universe() {}
    Universe(sf::Vector2f iCenter) : center(iCenter) {}
    inline const vector<unique_ptr<CelestialBody>>&getBodies()
    const { return celBodies; }
    friend istream& operator>>(istream& in, Universe& u);
    friend ostream& operator<<(ostream& out, const Universe& u);
    void step(float seconds);
 private:
    sf::Vector2f center;
    float radius;
    vector<unique_ptr<CelestialBody>> celBodies;
};
```

Universe.cpp

In order to organize the Bodies when they are rotating along the supplied orbit and calculate scaleForce, Netforce, and other forces, the universe file in part-b has also been extremely important.

```cpp
// copyright 2023 YashwanthReddy
#include "Universe.hpp"
std::istream& operator>>(std::istream& in, Universe& u) {
int numBodies;
in >> numBodies >> u.radius;
CelestialBody::createUniverse(u.center, u.radius);
for(int i = 0; i < numBodies; i++) {
float xPosition, yPosition, xVelocity, yVelocity, mass;
    string imageRef;
in >> xPosition >> yPosition >> xVelocity >> yVelocity >> mass >> imageRef;
u.celBodies.push_back(
make_unique<CelestialBody>(sf::Vector2f(xPosition, yPosition),
sf::Vector2f(xVelocity, yVelocity), mass, imageRef));
}
return in;
}
ostream& operator<<(ostream& out, const Universe& u) {
out << u.celBodies.size() << endl;
out << u.radius << endl;
for(const auto &b : u.celBodies) out << (*b) << endl;
return out;
}
void Universe::step(float seconds) {
auto getNetForce = [&](size_t planetIndex) -> sf::Vector2f {
sf::Vector2f netForce;
for(size_t i = 0; i < celBodies.size(); i++) {
if(i != planetIndex) {
sf::Vector2f position_change = celBodies[i]->getPosition()
- celBodies[planetIndex]->getPosition();
float planetDistance = hypot(position_change.x,
 position_change.y);
float scaleForce =(6.67430e-11 * celBodies[planetIndex]->getMass() *
celBodies[i]->getMass()) /pow(planetDistance, 2);
sf::Vector2f force_xy = {
scaleForce * (position_change.x /planetDistance),
scaleForce * (position_change.y /planetDistance)
};
netForce += force_xy;
}
}
return netForce;
};
vector<sf::Vector2f> rPosition, rVelocity;
for(size_t i = 0; i < celBodies.size(); i++) {
sf::Vector2f netForce = getNetForce(i);
sf::Vector2f planetAccel = {
netForce.x / celBodies[i]->getMass(),
netForce.y / celBodies[i]->getMass()
};
sf::Vector2f velocity = {
celBodies[i]->getVelocity().x + seconds * planetAccel.x,
celBodies[i]->getVelocity().y + seconds * planetAccel.y
};
sf::Vector2f position = {
celBodies[i]->getPosition().x + seconds * velocity.x,
celBodies[i]->getPosition().y + seconds * velocity.y
};
rVelocity.push_back(velocity);
rPosition.push_back(position);
}
for(size_t i = 0; i < celBodies.size(); i++) {
celBodies[i]->setVelocity(rVelocity[i]);
celBodies[i]->setPosition(rPosition[i]);
celBodies[i]->spriteUpdate();
}
}
```

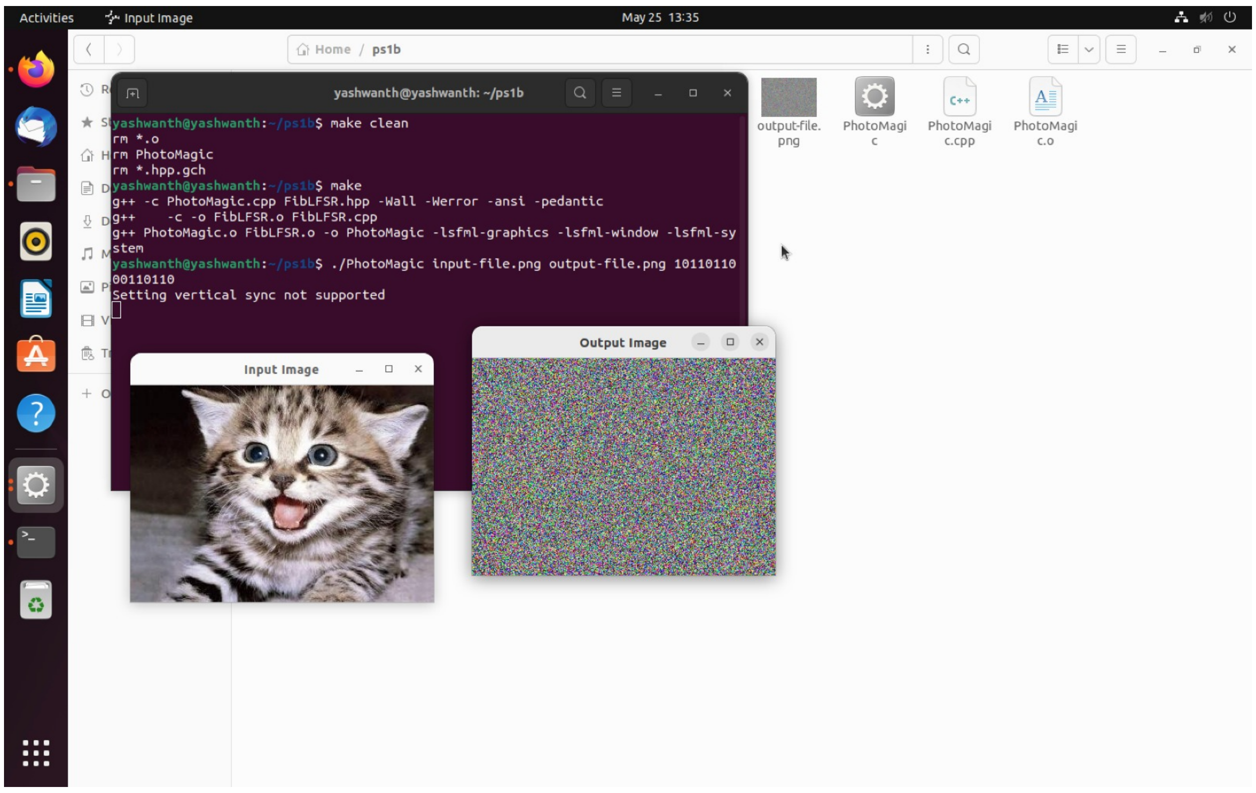
test.cpp
The test cases for checking the correctness of CelestialBodies.

```
// copyrights 2023 YashwanthReddy
#include <iostream>
#include <stdexcept>
#include "Universe.hpp"
#include "CelestialBody.hpp"
#define BOOST_TEST_MODULE MyTest
#include <boost/test/included/unit_test.hpp>
BOOST_AUTO_TEST_CASE(case1) {
std::ifstream file("planets.txt");
Universe u;
file >> u;
const std::vector<std::unique_ptr<CelestialBody>>& bodies = u.getBodies();
BOOST_CHECK_EQUAL(bodies.size(), 5);
}
BOOST_AUTO_TEST_CASE(case2) {
std::ifstream file("planets.txt");
Universe u;
file >> u;
const std::vector<std::unique_ptr<CelestialBody>>& bodies = u.getBodies();
const sf::Vector2f& position = bodies[0]->getPosition();
const sf::Vector2f& velocity = bodies[0]->getVelocity();
BOOST_CHECK_EQUAL(position.x, 149600000000);
BOOST_CHECK_EQUAL(position.y, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.x, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.y, 2.9800e+04);
}
BOOST_AUTO_TEST_CASE(case3) {
```

```
std::ifstream file("planets.txt");
Universe u;
file >> u;
const std::vector<std::unique_ptr<CelestialBody>>& bodies = u.getBodies();
const sf::Vector2f& velocity = bodies[1]->getVelocity();
BOOST_CHECK_EQUAL(velocity.x, 0.0000e+00);
}

BOOST_AUTO_TEST_CASE(case4) {
std::ifstream file("planets.txt");
Universe u;
file >> u;
const std::vector<std::unique_ptr<CelestialBody>>& bodies = u.getBodies();
const sf::Vector2f& position = bodies[2]->getPosition();
const sf::Vector2f& velocity = bodies[2]->getVelocity();
BOOST_CHECK_EQUAL(position.x, 57900000000);
BOOST_CHECK_EQUAL(position.y, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.x, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.y, 4.7900e+04);
}
BOOST_AUTO_TEST_CASE(case5) {
std::ifstream file("planets.txt");
Universe u;
file >> u;
const std::vector<std::unique_ptr<CelestialBody>>& bodies = u.getBodies();
const sf::Vector2f& position = bodies[3]->getPosition();
const sf::Vector2f& velocity = bodies[3]->getVelocity();
BOOST_CHECK_EQUAL(position.x, 0.0000e+00);
BOOST_CHECK_EQUAL(position.y, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.x, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.y, 0.0000e+00);
}
BOOST_AUTO_TEST_CASE(case6) {
std::ifstream file("planets.txt");
Universe u;
file >> u;
const std::vector<std::unique_ptr<CelestialBody>>& bodies = u.getBodies();
const sf::Vector2f& position = bodies[4]->getPosition();
const sf::Vector2f& velocity = bodies[4]->getVelocity();
BOOST_CHECK_EQUAL(position.x, 108200000000);
BOOST_CHECK_EQUAL(position.y, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.x, 0.0000e+00);
BOOST_CHECK_EQUAL(velocity.y, 3.5000e+04);
}
```

6.9 Output:

To see the static Output got to Figure: 15 To see the output taken when planets are moving is given below:



```
yashwanth@yashwanth:~/ps1b$ make clean
rm *.o
rm PhotoMagic
rm *.hpp.gch
yashwanth@yashwanth:~/ps1b$ make
g++ -c PhotoMagic.cpp FibLFSR.hpp -Wall -Werror -ansi -pedantic
g++    -c -o FibLFSR.o FibLFSR.cpp
g++ PhotoMagic.o FibLFSR.o -o PhotoMagic -lsfml-graphics -lsfml-window -lsfml-sy
stem
yashwanth@yashwanth:~/ps1b$ ./PhotoMagic input-file.png output-file.png 10110110
00110110
Setting vertical sync not supported
```

Figure 16: Non Static Screenshoted Image

7 PS4a: Sokoban UI

7.1 Discussion:

Sokoban is a puzzle game that involves moving boxes or crates within a confined space. The objective of the game is to push all the boxes onto designated target locations, represented by dots or squares, while navigating through a maze-like level. The player controls a character that can move horizontally and vertically, and they can push the boxes but cannot pull them. The game is typically presented in a top-down view, with the level layout displayed on a grid. The player needs to plan their moves strategically to avoid getting boxes trapped in corners or against walls, as this could lead to an unsolvable situation. The challenge increases as the levels progress, with more complex arrangements of boxes and obstacles. Sokoban requires logical thinking and problem-solving skills to find the optimal moves and solve each puzzle. It emphasizes spatial reasoning and pattern recognition, as players need to visualize the consequences of their actions and plan ahead to avoid getting stuck.

7.2 Key algorithms, Data structures and OO Designs used in this Assignment:

The operator» overload allows reading Sokoban objects from an input stream. It reads a filename and calls the loadLevel function to load the corresponding level. The setWindowDimensions function calculates the required window dimensions based on the size of the grid and the tile size. The game loads levels from input files using the loadLevel function. The function reads the dimensions of the grid and the characters representing each tile from the input file and populates the grid accordingly.

7.3 Images used:

Figure 17: Player Image

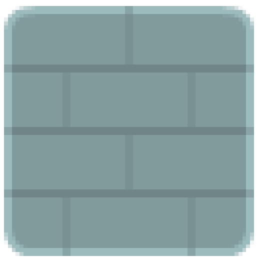


Figure 18: Wall Image

Figure 19: Box Image

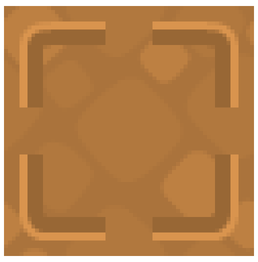


Figure 20: Target Image

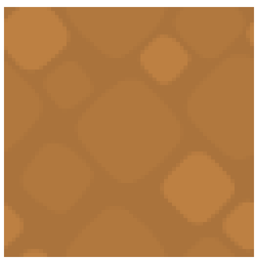


Figure 21: Empty/ground Image

7.4 What I accomplished :

My code includes a loadLevel function that reads level data from a file and populates the grid accordingly. It extracts the dimensions of the grid and the characters representing each tile, such as walls, boxes, targets, and the player. This allows for loading different levels for the Sokoban game.

7.5 What I already knew :

I knew the gameplay and SFML library and these made coding of this program easier.

7.6 What I learned :

I learnt how to place the tiles perfectly inside the window.

7.7 Challenges :

there are no issues regarding the code, and there is problem when including the empty.png background to player but changed the player picture. faced issues during the grid values, but tried to resolve them and now, it displays everything.

7.8 Codebase

Makefile:
This Makefile has no lint included

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

OBJECTS = Sokoban.o

all: Sokoban

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

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

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


clean:
    rm -f *.o Sokoban

lint:
    cpplint *.cpp
```

Sokoban.hpp:

This file consists of variables and functions used in the Sokoban.cpp file.

```cpp
// copyright 2023 YashwanthReddy
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string>
#include <SFML/Graphics.hpp>
enum Direction { Up, Down, Left, Right };
struct Coordinate {
int x;
int y;
};
class Sokoban : public sf::Drawable {
 private:
    std::vector<std::vector<char>> grid;
    Coordinate playerLocation;
    std::vector<Coordinate> boxLocations;
    std::vector<Coordinate> storageLocations;
    sf::Texture emptyTexture;
    sf::Texture wallTexture;
    sf::Texture boxTexture;
    sf::Texture targetTexture;
    sf::Texture playerTexture;
    std::vector<sf::Sprite> emptySprites;
    std::vector<sf::Sprite> wallSprites;
    std::vector<sf::Sprite> boxSprites;
    std::vector<sf::Sprite> storageSprites;
    sf::Sprite playerSprite;
    sf::Vector2u windowSize;
    unsigned int tileSize;
    sf::Clock clock;
    void setWindowDimensions();
    void setTileSize();

 public:
    Sokoban();
    sf::Time elapsedTime;
    std::string formatTime(sf::Time time) const;
    bool loadLevel(const std::string& filename);
    void movePlayer(Direction direction);
    void draw(sf::RenderTarget& target, sf::RenderStates states) const
   override;
    unsigned int getWindowWidth() const;
    unsigned int getWindowHeight() const;
    bool isBox(int x, int y) const;
    bool isWon() const;
    unsigned int width;
    unsigned int height;
    friend std::istream& operator>>(std::istream& is, Sokoban& sokoban);
};
```

Sokoban.cpp:
This file is representation of static window with tiles

```
// copyright 2023 YashwanthReddy
#include "Sokoban.hpp"
#include <string>
Sokoban::Sokoban() : width(0), height(0) {
emptyTexture.loadFromFile("empty.png");
wallTexture.loadFromFile("wall.png");
boxTexture.loadFromFile("box.png");
targetTexture.loadFromFile("target.png");
playerTexture.loadFromFile("player.png");
tileSize = 64;
clock.restart();
}
bool Sokoban::loadLevel(const std::string& filename) {
std::ifstream inputFile(filename);
if (!inputFile) {
std::cout << "Failed to open the input file." << std::endl;
return false;
}
int numRows, numColumns;
inputFile >> numRows >> numColumns;
grid.resize(numRows, std::vector<char>(numColumns));
for (int i = 0; i < numRows; ++i) {
for (int j = 0; j < numColumns; ++j) {
inputFile >> grid[i][j];
}
}
inputFile.close();
setWindowDimensions();
for (int i = 0; i < numRows; ++i) {
for (int j = 0; j < numColumns; ++j) {
char tile = grid[i][j];
if (tile == '@') {
playerLocation = {j, i};
playerSprite.setTexture(playerTexture);
playerSprite.setPosition(j * tileSize, i * tileSize);
} else if (tile == '#') {
sf::Sprite wallSprite;
wallSprite.setTexture(wallTexture);
wallSprite.setPosition(j * tileSize, i * tileSize);
wallSprites.push_back(wallSprite);
} else if (tile == 'A') {
sf::Sprite boxSprite;
boxSprite.setTexture(boxTexture);
boxSprite.setPosition(j * tileSize, i * tileSize);
boxSprites.push_back(boxSprite);
boxLocations.push_back({j, i});
} else if (tile == 'a') {
sf::Sprite storageSprite;
storageSprite.setTexture(targetTexture);
storageSprite.setPosition(j * tileSize, i * tileSize);
storageSprites.push_back(storageSprite);
storageLocations.push_back({j, i});
} else if (tile == '.') {
sf::Sprite emptySprite;
emptySprite.setTexture(emptyTexture);
emptySprite.setPosition(j * tileSize, i * tileSize);
emptySprites.push_back(emptySprite);
}
}
}
return true;
}
void Sokoban::movePlayer(Direction direction) {
// Will be done in ps4(b)
}
void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
for (const sf::Sprite& wallSprite : wallSprites)
target.draw(wallSprite, states);
for (const sf::Sprite& boxSprite : boxSprites)
target.draw(boxSprite, states);
for (const sf::Sprite& storageSprite : storageSprites)
target.draw(storageSprite, states);
for (const sf::Sprite& emptySprite : emptySprites)
target.draw(emptySprite, states);
target.draw(playerSprite, states);
}
unsigned int Sokoban::getWindowWidth() const {
return windowSize.x;
}
unsigned int Sokoban::getWindowHeight() const {
return windowSize.y;
}
bool Sokoban::isBox(int x, int y) const {
for (const Coordinate& boxLocation : boxLocations) {
if (boxLocation.x == x && boxLocation.y == y)
return true;
}
return false;
}
bool Sokoban::isWon() const {
// Will be done in ps(b)
return false;
}
void Sokoban::setWindowDimensions() {
unsigned int numColumns = grid[0].size();
unsigned int numRows = grid.size();
windowSize.x = numColumns * tileSize;
windowSize.y = numRows * tileSize;
}
std::istream& operator>>(std::istream& is, Sokoban& sokoban) {
    std::string filename;
is >> filename;
if (!sokoban.loadLevel(filename)) {
// levels will be read
}
return is;
}
```

main.cpp:

This adjusts the window size according to levels and moreover creates the window.

```
// copyright 2023 YashwanthReddy
#include <SFML/Graphics.hpp>
#include "Sokoban.hpp"
std::string formatElapsedTime(const sf::Time& elapsedTime) {
int minutes = static_cast<int>(elapsedTime.asSeconds()) / 60;
int seconds = static_cast<int>(elapsedTime.asSeconds()) % 60;
return std::to_string(minutes) + ":" +
(seconds < 10 ? "0" : "") + std::to_string(seconds);
}
int main(int argc, char* argv[]) {
if (argc < 2) {
std::cout << "Usage: ./Sokoban <level_file.txt>" << std::endl;
return -1;
}
Sokoban game;
if (!game.loadLevel(argv[1])) {
std::cout << "Failed to load the level." << std::endl;
return -1;
}
sf::RenderWindow window(sf::VideoMode
(game.getWindowWidth(), game.getWindowHeight()), "Sokoban");
sf::Clock clock;
sf::Time elapsedTime;
sf::Font font;
if (!font.loadFromFile("arial.ttf")) {
std::cout << "Failed to load font." << std::endl;
return -1;
}
while (window.isOpen()) {
sf::Event event;
while (window.pollEvent(event)) {
if (event.type == sf::Event::Closed) {
window.close();
} else if (event.type == sf::Event::KeyPressed) {
if (event.key.code == sf::Keyboard::Up) {
game.movePlayer(Direction::Up);
} else if (event.key.code == sf::Keyboard::Down) {
game.movePlayer(Direction::Down);
} else if (event.key.code == sf::Keyboard::Left) {
game.movePlayer(Direction::Left);
} else if (event.key.code == sf::Keyboard::Right) {
game.movePlayer(Direction::Right);
}
}
}
elapsedTime += clock.restart();
window.clear();
window.draw(game);
sf::Text timeText;
timeText.setFont(font);
timeText.setString(formatElapsedTime(elapsedTime));
timeText.setCharacterSize(24);
timeText.setFillColor(sf::Color::White);
timeText.setPosition(10.f, 10.f);
window.draw(timeText);
window.display();
}
return 0;
}
```

7.9 Output :

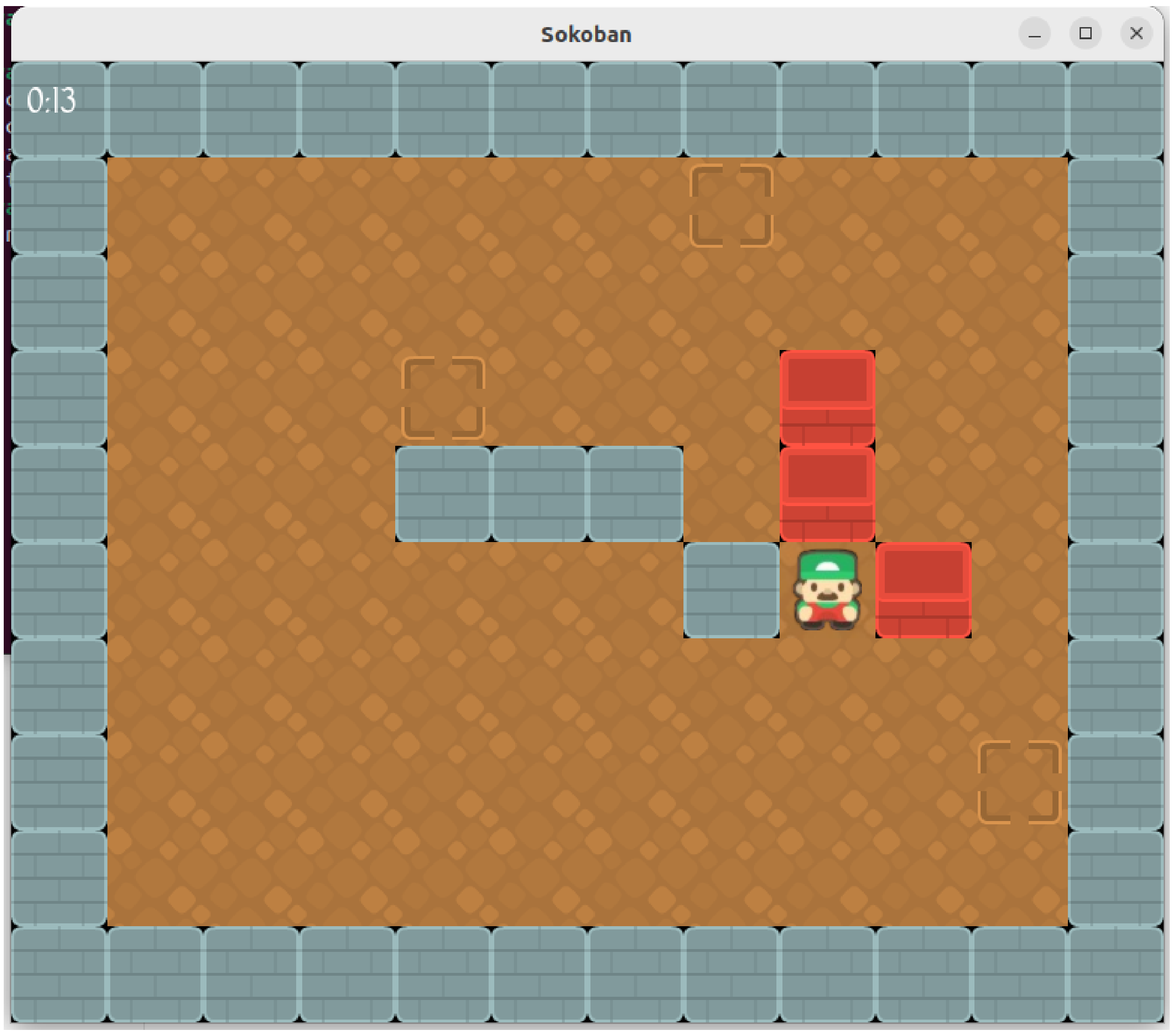


Figure 22: Output-1

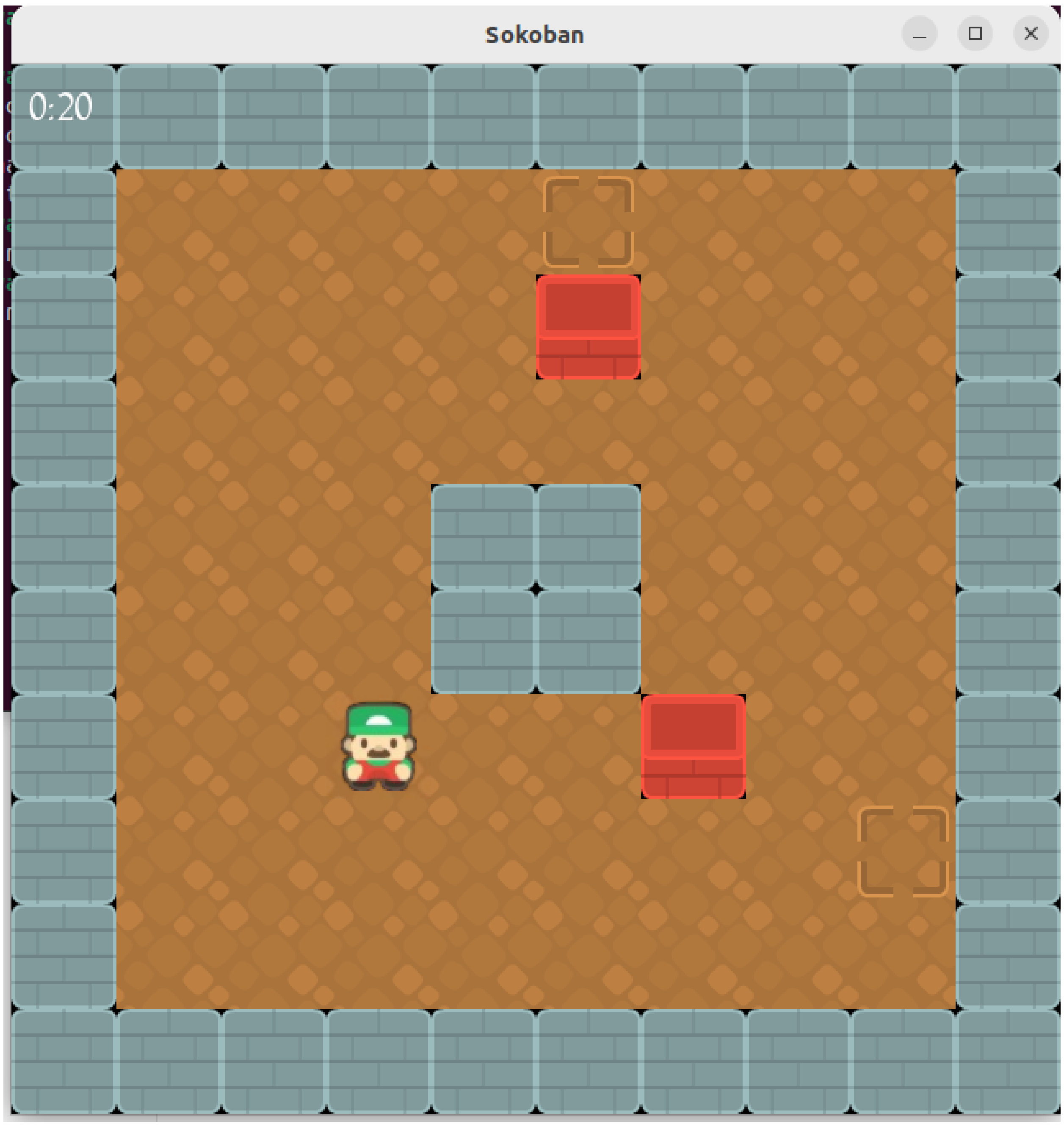


Figure 23: Output-2

8 PS4b: Sokoban

8.1 Discussion:

In the previous Sokoban UI, we just displayed the tiles on the window and not the gameplay. In this particular project, we implement the gameplay and make the player to move around in his designated paths. The player tries to push the boxes into the targets, if all the targets are covered by boxes, then the game is over and the level gets upgraded automatically. My code displays the 6 levels and after its completion, the player gets to know the time taken to finish all the levels. For this, movePlayer() and isWon() functions were implemented to give the perfect gameplay.

8.2 Key algorithms, Data structures and OO Designs used in this Assignment:

The movePlayer function handles the movement of the player based on the specified direction. It calculates the new player location and checks for collisions with walls and boxes, ensuring valid moves. The moveBox function is responsible for moving a box in a given direction. It updates the box's position and sprite, as well as checks for collisions with other boxes or walls. The isWon function checks if all the boxes in the storage locations, indicating a win condition. It uses the $(std :: all_)$ of algorithm to iterate over the storage locations and verifies if each location contains a box. The operator» is overloaded to enable loading level data from an input stream into the Sokoban object. It reads the dimensions and characters from the stream and populates the grid accordingly.

8.3 Images used:

The images are already shown in PS4a Sections, To browse them you can click on the following *Figure Numbers:*

- **The Player is Figure 17**
- **The Target is Figure 20**
- **The Empty/Ground is Figure 21**
- **The box is Figure 19**
- **The Wall is Figure 18**

8.4 What I accomplished :

I accomplished the gameplay of the Sokoban and incremented the levels from level1 to level6. Also displayed the music and 'You Won' picture when the player finishes all the levels.

8.5 What I already knew :

I knew the SFML library for setting the tileSize and tiles correctly.

8.6 What I learned :

I learnt how to place the tiles perfectly inside the window and make player moves in his designated places.

8.7 Challenges :

Changing the window size after every level and moreover making the box and player.png coordination was difficult.

8.8 Codebase

Makefile

This Makefile does not have linting, as there are no lints in my code

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

OBJECTS = Sokoban.o

all: Sokoban

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

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

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


clean:
    rm -f *.o Sokoban

lint:
    cpplint *.cpp
```

main.cpp

This is the main file where the window is drawn and level incrementation occurs.

```
// copyright 2023 YashwanthReddy
#include <iostream>
#include <stack>
#include "Sokoban.hpp"
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
std::string formatElapsedTime(const sf::Time& elapsedTime) {
int minutes = static_cast<int>(elapsedTime.asSeconds()) / 60;
int seconds = static_cast<int>(elapsedTime.asSeconds()) % 60;
return std::to_string(minutes) + ":" +
(seconds < 10 ? "0" : "") + std::to_string(seconds);
}
int main(int argc, char* argv[]) {
if (argc < 2) {
std::cout << "Usage: ./Sokoban <level_file.txt>" << std::endl;
return -1;
}
Sokoban game;
int c = 1;
    std::string fn = argv[1];
if (!game.loadLevel(fn)) {
std::cout << "Failed to load the level." << std::endl;
return -1;
}
sf::RenderWindow window(sf::VideoMode(game.
getWindowWidth(), game.getWindowHeight()), "Sokoban");
sf::Clock clock;
sf::Time elapsedTime;
sf::Font font;
```

```
if (!font.loadFromFile("arial.ttf")) {
std::cout << "Failed to load font." << std::endl;
return -1;
}
std::stack<Sokoban> undoStack;
while (window.isOpen()) {
sf::Event event;
while (window.pollEvent(event)) {
if (event.type == sf::Event::Closed) {
window.close();
} else if (event.type == sf::Event::KeyPressed) {
if (event.key.code == sf::Keyboard::Up) {
undoStack.push(game);
game.movePlayer(Direction::Up);
} else if (event.key.code == sf::Keyboard::Down) {
undoStack.push(game);
game.movePlayer(Direction::Down);
} else if (event.key.code == sf::Keyboard::Left) {
undoStack.push(game);
game.movePlayer(Direction::Left);
} else if (event.key.code == sf::Keyboard::Right) {
undoStack.push(game);
game.movePlayer(Direction::Right);
} else if (event.key.code == sf::Keyboard::R) {
game = Sokoban();
undoStack = std::stack<Sokoban>();
if (!game.loadLevel(fn)) {
std::cout << "Failed to load the level." << std::endl;
return -1;
}
} else if (event.key.code == sf::Keyboard::D && !undoStack.empty()) {
game = undoStack.top();
undoStack.pop();
}
}
}
elapsedTime += clock.restart();
window.clear();
window.draw(game);
sf::Text timeText;
timeText.setFont(font);
timeText.setString(formatElapsedTime(elapsedTime));
timeText.setCharacterSize(24);
timeText.setFillColor(sf::Color::Black);
timeText.setPosition(10.f, 10.f);
window.draw(timeText);
window.display();
if (game.isWon()) {
std::cout << "You won level " << c << std::endl;
c++;
if (c <= 6) {
fn = "level" + std::to_string(c) + ".txt";
if (!game.loadLevel(fn)) {
std::cout << "Failed to load" << std::endl;
break;
}
window.create(sf::VideoMode(game.
getWindowWidth(), game.getWindowHeight()), "Sokoban");
} else {
```

```cpp
std::cout << "You Done It" << std::endl;
std::cout << "Time You Took: " << formatElapsedTime(elapsedTime) << std::
    endl;
window.close();
sf::RenderWindow cWindow(sf::VideoMode(740, 493), "Congratulations!");
sf::Texture texture;
if (!texture.loadFromFile("image.png")) {
std::cout << "Failed to load image." << std::endl;
return -1;
}
sf::Sprite sprite(texture);
cWindow.draw(sprite);
cWindow.display();
sf::SoundBuffer soundBuffer;
if (!soundBuffer.loadFromFile("sound.ogg")) {
std::cout << "Failed to load sound." << std::endl;
return -1;
}
sf::Sound sound(soundBuffer);
sound.play();
while (cWindow.isOpen()) {
sf::Event event;
while (cWindow.pollEvent(event)) {
if (event.type == sf::Event::Closed) {
cWindow.close();
}
}
}
return 0;
}
}
}
return 0;
}
```

Sokoban.hpp

The initialization of the Sokoban methods and variables.

```cpp
// copyright 2023 YashwanthReddy
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <string>
#include <SFML/Graphics.hpp>
enum Direction { Up, Down, Left, Right };
struct Coordinate {
    int x;
    int y;
};
class Sokoban : public sf::Drawable {
 private:
    std::vector<std::vector<char>> grid;
    Coordinate playerLocation;
    std::vector<Coordinate> boxLocations;
    std::vector<Coordinate> storageLocations;
    sf::Texture emptyTexture;
    sf::Texture wallTexture;
    sf::Texture boxTexture;
    sf::Texture targetTexture;
    sf::Texture playerTexture;
    std::vector<sf::Sprite> emptySprites;
    std::vector<sf::Sprite> wallSprites;
    std::vector<sf::Sprite> boxSprites;
    std::vector<sf::Sprite> storageSprites;
    sf::Sprite playerSprite;
    sf::Vector2u windowSize;
    unsigned int tileSize;
    sf::Clock clock;
    void setWindowDimensions();
    void moveBox(int boxIndex, Direction direction);

 public:
    Sokoban();
    bool loadLevel(const std::string& filename);
    void movePlayer(Direction direction);
    void draw(sf::RenderTarget& target, sf::RenderStates states) const
   override;
    unsigned int getWindowWidth() const;
    unsigned int getWindowHeight() const;
    bool isBox(int x, int y) const;
    bool isWon() const;
    void clearLevel();
};
std::istream& operator>>(std::istream& is, Sokoban& sokoban);
```

Sokoban.cpp

The Sokoban tiles display and entire gameplay is in this file.

```cpp
// copyright 2023 yashwanthReddy
#include <algorithm>
#include <string>
#include "Sokoban.hpp"
Sokoban::Sokoban() : tileSize(64) {
emptyTexture.loadFromFile("empty.png");
wallTexture.loadFromFile("wall.png");
boxTexture.loadFromFile("box.png");
targetTexture.loadFromFile("target.png");
playerTexture.loadFromFile("player.png");
}
bool Sokoban::loadLevel(const std::string& filename) {
std::ifstream inputFile(filename);
if (!inputFile.is_open()) {
std::cerr << "Failed to open the input file." << std::endl;
return false;
}
int numRows, numColumns;
inputFile >> numRows >> numColumns;
grid.resize(numRows, std::vector<char>(numColumns));
for (int i = 0; i < numRows; ++i) {
for (int j = 0; j < numColumns; ++j) {
inputFile >> grid[i][j];
}
}
inputFile.close();
setWindowDimensions();
clearLevel();
for (int i = 0; i < numRows; ++i) {
for (int j = 0; j < numColumns; ++j) {
char tile = grid[i][j];
if (tile == '@') {
playerLocation = { j, i };
}
if (tile == '#') {
sf::Sprite wallSprite;
wallSprite.setTexture(wallTexture);
wallSprite.setPosition(j * tileSize, i * tileSize);
wallSprites.push_back(wallSprite);
} else if (tile == 'A') {
sf::Sprite boxSprite;
boxSprite.setTexture(boxTexture);
boxSprite.setPosition(j * tileSize, i * tileSize);
boxSprites.push_back(boxSprite);
boxLocations.push_back({ j, i });
} else if (tile == 'a') {
sf::Sprite storageSprite;
storageSprite.setTexture(targetTexture);
storageSprite.setPosition(j * tileSize, i * tileSize);
storageSprites.push_back(storageSprite);
storageLocations.push_back({ j, i });
} else if (tile == '.' || tile == '@') {
sf::Sprite emptySprite;
emptySprite.setTexture(emptyTexture);
emptySprite.setPosition(j * tileSize, i * tileSize);
emptySprites.push_back(emptySprite);
}
}
}
playerSprite.setTexture(playerTexture);
playerSprite.setPosition(playerLocation.
x * tileSize, playerLocation.y * tileSize);
return true;
}
void Sokoban::movePlayer(Direction direction) {
int dx = 0;
int dy = 0;
if (direction == Direction::Up) {
dy = -1;
} else if (direction == Direction::Down) {
        dy = 1;
} else if (direction == Direction::Left) {
        dx = -1;
} else if (direction == Direction::Right) {
        dx = 1;
}
Coordinate nextLocation = { playerLocation.x + dx, playerLocation.y + dy };
if (nextLocation.x < 0 || nextLocation.x >= static_cast<int>(grid[0].size())
    ||
nextLocation.y < 0 || nextLocation.y >= static_cast<int>(grid.size()))
return;
char nextTile = grid[nextLocation.y][nextLocation.x];
if (nextTile == '#')
return;
if (isBox(nextLocation.x, nextLocation.y)) {
int boxIndex = -1;
for (std::vector<Coordinate>::size_type i = 0; i < boxLocations.size(); ++i)
    {
if (boxLocations[i].x == nextLocation
.x && boxLocations[i].y == nextLocation.y) {
boxIndex = i;
break;
}
}
if (boxIndex == -1)
return;
Coordinate boxNextLocation = { nextLocation.x + dx, nextLocation.y + dy };
if (boxNextLocation.x < 0 || boxNextLocation
.x >= static_cast<int>(grid[0].size()) ||
boxNextLocation.y < 0 || boxNextLocation.y >= static_cast<int>(grid.size()))
return;
char boxNextTile = grid[boxNextLocation.y][boxNextLocation.x];
if (boxNextTile == '#' || isBox(boxNextLocation.x, boxNextLocation.y))
return;
moveBox(boxIndex, direction);
}
playerLocation = nextLocation;
playerSprite.setPosition(playerLocation.
x * tileSize, playerLocation.y * tileSize);
}
void Sokoban::moveBox(int boxIndex, Direction direction) {
int dx = 0;
int dy = 0;
if (direction == Direction::Up) {
dy = -1;
} else if (direction == Direction::Down) {
```

```cpp
dy = 1;
} else if (direction == Direction::Left) {
dx = -1;
} else if (direction == Direction::Right) {
dx = 1;
}
Coordinate boxLocation = boxLocations[boxIndex];
Coordinate boxNextLocation = { boxLocation.x + dx, boxLocation.y + dy };
boxLocations[boxIndex] = boxNextLocation;
boxSprites[boxIndex].setPosition(boxNextLocation.
x * tileSize, boxNextLocation.y * tileSize);
sf::Sprite emptySprite;
emptySprite.setTexture(emptyTexture);
emptySprite.setPosition(boxLocation.x * tileSize, boxLocation.y * tileSize);
emptySprites.push_back(emptySprite);
}
void Sokoban::setWindowDimensions() {
unsigned int numColumns = grid[0].size();
unsigned int numRows = grid.size();
windowSize.x = numColumns * tileSize;
windowSize.y = numRows * tileSize;
}
void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
for (const auto& sprite : emptySprites)
target.draw(sprite, states);
for (const auto& sprite : wallSprites)
target.draw(sprite, states);
for (const auto& sprite : boxSprites)
target.draw(sprite, states);
for (const auto& sprite : storageSprites)
target.draw(sprite, states);
target.draw(playerSprite, states);
for (const auto& sprite : boxSprites)
target.draw(sprite, states);
}
unsigned int Sokoban::getWindowWidth() const {
return windowSize.x;
}
unsigned int Sokoban::getWindowHeight() const {
return windowSize.y;
}
bool Sokoban::isBox(int x, int y) const {
Coordinate location{x, y};
auto isBoxLocation = [location](const Coordinate& box) {
return box.x == location.x && box.y == location.y;
};
return std::find_if(boxLocations
.begin(), boxLocations.end(), isBoxLocation) != boxLocations.end();
}
bool Sokoban::isWon() const {
return std::all_of(storageLocations
.begin(), storageLocations.end(), [this](const Coordinate& location) {
return isBox(location.x, location.y);
});
}
void Sokoban::clearLevel() {
wallSprites.clear();
boxSprites.clear();
```

```
storageSprites.clear();
emptySprites.clear();
boxLocations.clear();
storageLocations.clear();
}
std::istream& operator>>(std::istream& is, Sokoban& sokoban) {
    std::string filename;
is >> filename;
if (!sokoban.loadLevel(filename)) {
std::cerr << "Failed to load level from file: " << filename << std::endl;
}
return is;
}
```

8.9 Output:

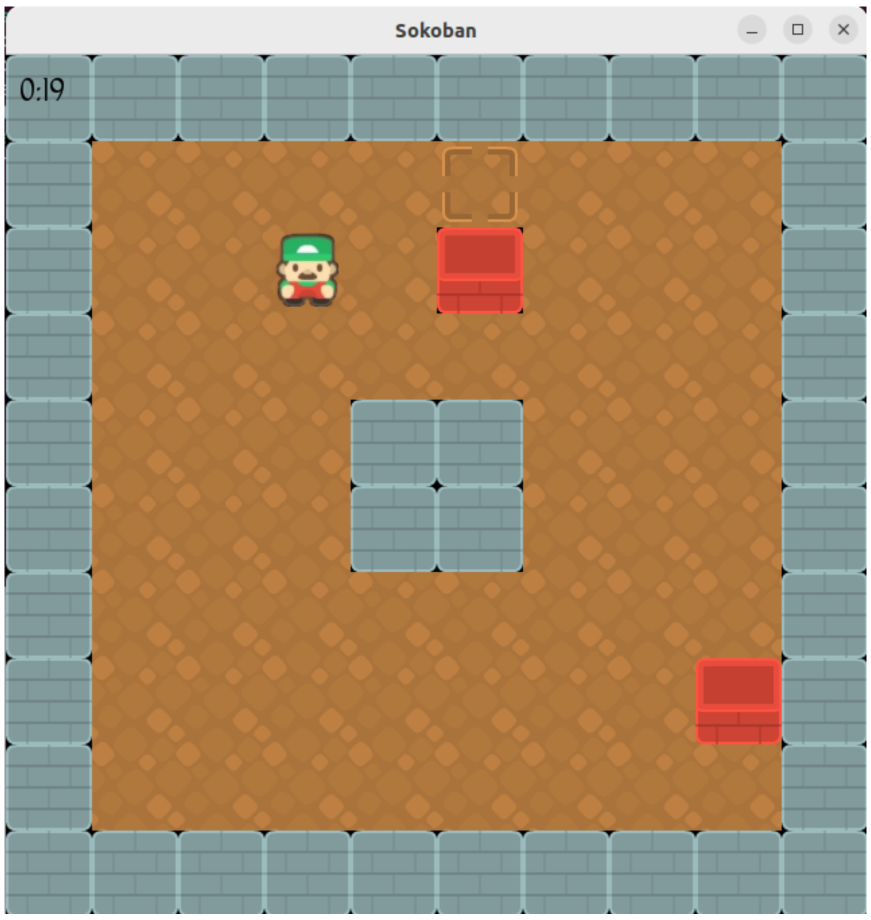


Figure 24: Output screenshot-1

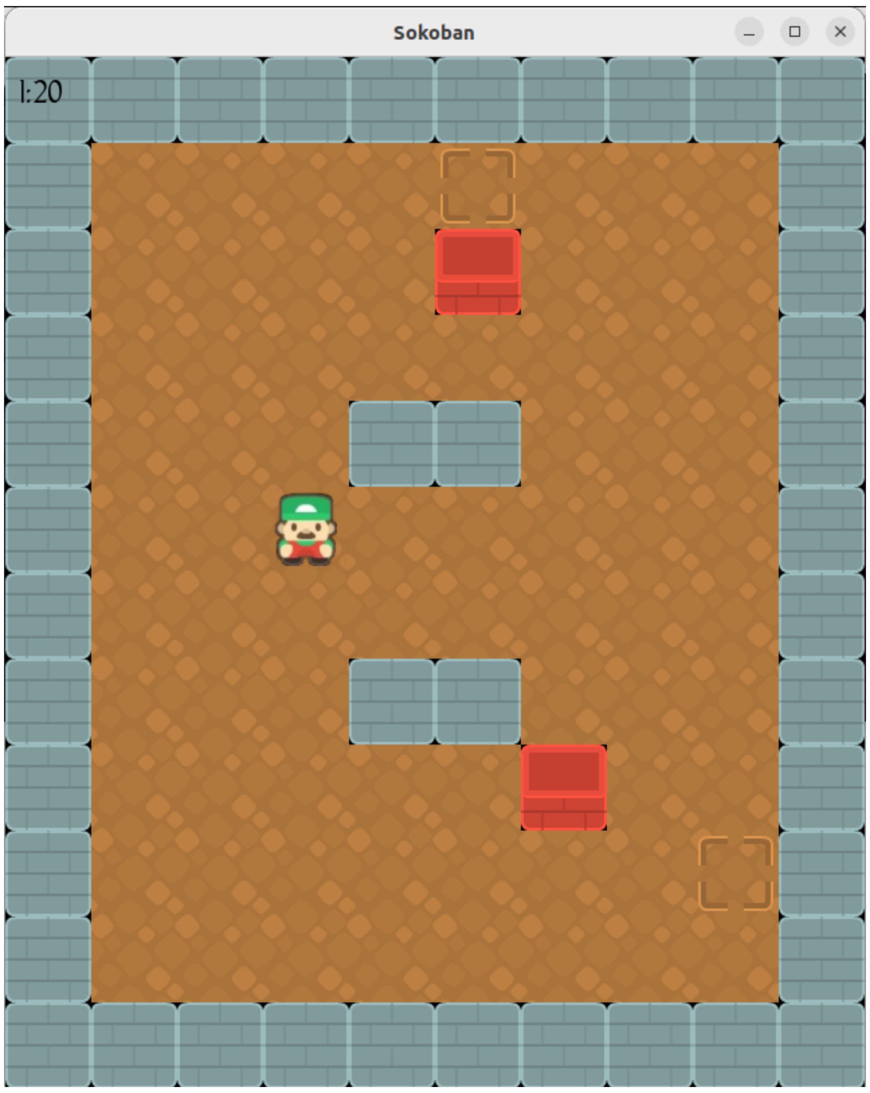


Figure 25: Output screenshot-2

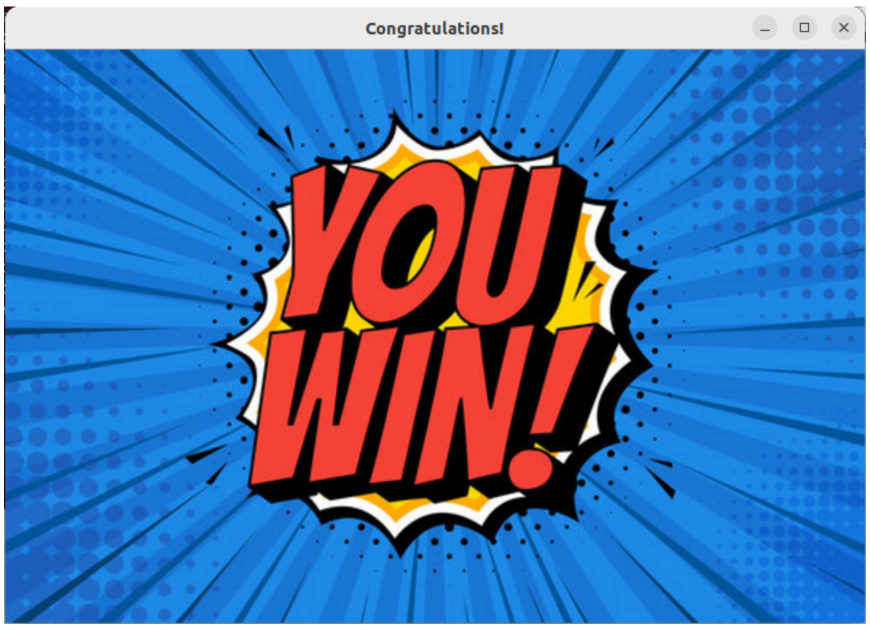


Figure 26: You Won Output

9 PS5: DNA Sequence Alignment

9.1 Discussion:

Finding similarities and differences between two or more DNA sequences is the aim of DNA sequence alignment. The genetic information that is encoded in an organism's DNA is represented as a string of nucleotides (A, T, C, and G) as a result of DNA sequencing. To determine the best sequence alignment of two DNA strings, I created a program. To do this, I developed a class that could compute the edit distance to determine how close the two sequences are to one another and weigh such distance to model mutations that might occur when aligning DNA sequences.

9.2 Key algorithms, Data structures and OO Designs used in this Assignment:

The code uses lambda functions ($lambda_penaltyandlambda_minimum$) to calculate the penalty for different characters and find the minimum value among three integers, respectively. These lambda functions provide a concise way to define and use small, inline functions within the code. The code utilizes a 2D array (opt) to store the intermediate results and calculate the edit distance. The dimensions of the array are based on the lengths of the input strings. The code implements the Wagner-Fischer algorithm for calculating the edit distance between two strings. The algorithm uses dynamic programming to determine the minimum number of operations (insertions, deletions, and substitutions) required to transform one string into another.

9.3 What I accomplished :

I have implemented the edit distance algorithm using dynamic programming to calculate the minimum number of operations required to transform one string into another. Additionally, you have designed a class structure with appropriate data structures and utilized lambda functions for penalty calculation and finding the minimum value.

9.4 What I learned :

I learnt about lambda functions. I also learnt how to read DNA sequences from files or input streams, store them in memory, and manipulate them during the alignment process.

9.5 Challenges :

I faced problem while solving the test cases and tried to clear them.

9.6 Codebase

Makefile
This Makefile is used to make the output file by checking the errors.

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
OBJECTS= EDistance.o
all: EDistance test
EDistance: main.o $(OBJECTS)
	$(CC) main.o $(OBJECTS) -o EDistance $(LIB)
test: test.o $(OBJECTS)
	$(CC) test.o $(OBJECTS) -o test $(LIB)
main.o: main.cpp EDistance.hpp
	$(CC) -c main.cpp  $(CFLAGS)
EDistance.o: EDistance.cpp EDistance.hpp
	$(CC) -c EDistance.cpp $(CFLAGS)
test.o: test.cpp EDistance.hpp
	$(CC) -c test.cpp $(CFLAGS)
clean:
	rm *.o
	rm EDistance
	rm test
```

main.cpp
The edit distance and total distance are shown in this file.

```cpp
// Copyright 2023 YashwanthReddy & PranayaLaxmi
#include <iostream>
#include <string>
#include "EDistance.hpp"
#include <SFML/System.hpp>
int main(int argc, const char* argv[]) {
    sf::Clock clock;
    sf::Time t;

    std::string string1, string2;
    std::cin >> string1 >> string2;

    EDistance ed(string1, string2);

    int distance = ed.optDistance();

    std::string alignment = ed.alignment();

    std::cout << "Edit distance = " << distance << std::endl;
    std::cout << alignment << std::endl;

    t = clock.getElapsedTime();
    std::cout << "Execution time is " << t.asSeconds()
    << " seconds \n" << std::endl;
    return 0;
}
```

EDistance.hpp

This file serves as the "EDistance.cpp" header. The EDistance class, which declares the private and public variables and functions, is contained in this file.

```cpp
// Copyright 2023 YashwanthReddy & PranayaLaxmi
#pragma once
#include <string>
#include <algorithm>
class EDistance {
 public:
        EDistance(const std::string &a, const std::string &b);
        ~EDistance();
        static int penalty(char a, char b);
        static int min3(int a, int b, int c);
        int optDistance();
        std::string alignment();

 private:
        int **opt;
        std::string x;
        std::string y;
        int M, N;
};
```

EDistance.cpp

This file computes the DNA sequence alignment cost matrix.

```cpp
// Copyright 2023 YashwanthReddy & PranayaLaxmi
#include <iostream>
#include <string>
#include <algorithm>
#include "EDistance.hpp"
EDistance::EDistance(const std::string &a, const std::string &b) {
    this->x = a;
    this->y = b;
    if (a.empty() || b.empty()) {
        throw std::exception();
    } else {
    M = x.size();
    N = y.size();
    opt = new int *[M + 1]();
    if (!opt) {
       std::cout << "Not enough memory" << std::endl;
        exit(1);
    }
    for (int i = 0; i < M + 1; i++)
        opt[i] = new int[N + 1]();
    for (int i = 0; i <= M; i++) {
        opt[i][N] = 2 * (M - i);
    }
    for (int j = 0; j <= N; j++) {
        opt[M][j] = 2 * (N - j);
    }
}}
int EDistance::optDistance() {
    for (int i = M - 1; i >= 0; i--)
        for (int j = N - 1; j >= 0; j--)
opt[i][j] = min3(opt[i + 1][j + 1] + penalty(x[i],
y[j]), opt[i + 1][j] + 2, opt[i][j + 1] + 2);
    return opt[0][0];
}
```

```
std::string EDistance::alignment() {
    std::string result;
    int i = 0, j = 0;
    while (i < M || j < N) {
if (i < M && j < N && x[i] == y[j] && opt[i][j] == opt[i + 1][j + 1]) {
            result += x[i];
            result += " ";
            result += y[j];
            result += " 0";
            i++;
            j++;
} else if (i < M && j < N && x[i] != y[j] &&
opt[i][j] == opt[i + 1][j + 1] + 1) {
            result += x[i];
            result += " ";
            result += y[j];
            result += " 1";
            i++;
            j++;
        } else if (i < M && opt[i][j] == opt[i + 1][j] + 2) {
            result += x[i];
            result += " - 2";
            i++;
        } else if (j < N && opt[i][j] == opt[i][j + 1] + 2) {
            result += "- ";
            result += y[j];
            result += " 2";
            j++;
        }
        if (i < M || j < N) result += "\n";
    }
    return result;
}
int EDistance::penalty(char a, char b) {
     char A = a, B = b;
    auto lambda_penalty = [ = ]()->int {
        if (A == B) {  // returns 0 if the number of characters are equal
            return 0;
        } else if (A != B) {  // Else, it returns 1
            return 1;
        }
        return -1;
    };
    return lambda_penalty();
}
int EDistance::min3(int a, int b, int c) {
    int A = a, B = b, C = c;
    auto lambda_minimum = [ = ]()->int {
        if (A < B && A < C) {
            return A;
        } else if (B < A && B < C) {
            return B;
        } else if (C < A && C < B) {
            return C;
        }
        return A;
    };
    return lambda_minimum();
}
```

```cpp
EDistance::~EDistance() {
    for (int i = 0; i <= M; i++) {
        delete[] opt[i];
    }
    delete[] opt;
}
```

test.cpp

This file is used to test the public functions declared.

```cpp
// Copyright 2023 YashwanthReddy & PranayaLaxmi
#define BOOST_TEST_MODULE MyTest
#include <boost/test/included/unit_test.hpp>
#include "EDistance.hpp"
BOOST_AUTO_TEST_CASE(test1) {
    EDistance ed1("hello", "hello");
    BOOST_REQUIRE_EQUAL(ed1.optDistance(), 0);

    EDistance ed2("eatting", "setting");
    BOOST_REQUIRE_EQUAL(ed2.optDistance(), 2);
}
BOOST_AUTO_TEST_CASE(test2) {
BOOST_REQUIRE_THROW(EDistance("hello", ""), std::exception);
BOOST_REQUIRE_THROW(EDistance("", ""), std::exception);
}
BOOST_AUTO_TEST_CASE(test3) {
    int result1 = EDistance::min3(4, 3, 2);
    BOOST_REQUIRE_EQUAL(result1, 2);

    int result2 = EDistance::min3(5, 3, 5);
    BOOST_REQUIRE_EQUAL(result2, 3);

    int result3 = EDistance::min3(7, 7, 7);
    BOOST_REQUIRE_EQUAL(result3, 7);
}
BOOST_AUTO_TEST_CASE(test4) {
    std::string a = "a";
    std::string b = "a";
    EDistance ed(a, b);

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

    BOOST_REQUIRE_EQUAL(result, 0);
}
```

9.7 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
Execution time is 0.01087 seconds
```

10 PS7: Kronos Log Parsing

10.1 Discussion:

We analyze the Kronos Intouch time clock log by using regular expressions to parse the file. Also we verify device boot up timing. here we take the given device[1-6]*underscore*intouch.log files and give a resultant file which contains the time, date, status and line number of the boot to the .rpt file.

10.2 Key algorithms, Data structures and OO Designs used in this Assignment:

Struct: Defines the structure Service with attributes name, startTime, and completionTime to store information about each boot event.

```
 1. start_rgx: This regex is used to match and capture the start of a server in the log file. It has the following pattern:
     (.*): (\(log.c.166\) server started.*)
     Explanation:
         (.*): : Matches and captures any characters followed by a colon.
         (\(log.c.166\) server started.*): Matches and captures the substring "(log.c.166) server started" followed by any characters.

 2. succ_rgx: This regex is used to match and capture the successful completion of the server boot in the log file. It has the following pattern:
     (.*)\.\d*:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080.*
     Explanation:
         (.*)\.\d*: Matches and captures any characters followed by a dot and any number of digits.
         :INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080.*: Matches the remaining substring ":INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:9080" followed by any characters.
```

These regular expressions are used to identify and extract relevant information from the log file, such as the start time and completion time of the server boot events.

10.3 What I accomplished :

I developed a program that reads a log file containing device boot information. I implemented regular expressions to extract relevant information from the log file, such as the start time and completion time of the server boot events and stored the extracted information in a data structure called Service, which holds the service name, start time, and completion time. Generated a device boot report by writing the extracted information to an output file. Overall, I accomplished the task of parsing and analyzing device boot information from a log file and generating a report based on that data.

10.4 What I learned :

I learnt the implementation of <regex> and also learnt the <regex> library.

10.5 Challenges :

Felt difficult in understanding the regex expressions and the way to use them correctly.

10.6 Codebase

Makefile:

This Makefile has no linting, as there are no lints in the code.

```
CC = g++
CFLAGS = -std=c++17 -Wall -Werror -pedantic
LFLAGS = -lboost_regex -lboost_date_time

all: ps7

ps7: main.o
    $(CC) -o $@ main.o $(LFLAGS)

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

clean:
    rm -f ps7 *.o
```

main.cpp:

This file contains the regex library and time library and functions required for parsing.

```cpp
// copyright 2023 YashwanthReddy
#include <iostream>
#include <string>
#include <fstream>
#include <boost/regex.hpp>
#include "boost/date_time/posix_time/posix_time.hpp"
using std::cout;
using std::cin;
using std::endl;
using std::string;
using boost::regex;
using boost::smatch;
using boost::regex_error;
using boost::gregorian::date;
using boost::gregorian::from_simple_string;
using boost::gregorian::date_period;
using boost::gregorian::date_duration;
using boost::posix_time::ptime;
using boost::posix_time::time_duration;
struct Service {
    string name;
ptime startTime;
ptime completionTime;
};
int main(int argc, char** args) {
if (argc != 2) {
cout << "usage: ./ps7 [logfile]" << endl;
exit(1);
}
    string sgx, rs;
regex start_rgx;
regex succ_rgx;
bool flag = false;
string filename(args[1]);
std::ifstream infile(filename);
std::ofstream outfile(filename + ".rpt");
if (!infile || !outfile) {
cout << "open file error" << endl;
exit(1);
}
try {
start_rgx = regex(R"((.*): (\(log.c.166\) server started.*))");
succ_rgx = regex("(.*)\\.\\d*:INFO:oejs.AbstractConnector:Started "
"SelectChannelConnector@0.0.0.0:9080.*");
}
catch (regex_error& exc) {
[&]() {
cout << "regex init failed" << exc.code() << endl;
}();
exit(1);
}
std::vector<Service> services;
int ln = 1;
    string stx;
outfile << "The Device Boot Report" << endl;
outfile << "InTouch log file: " << filename << endl;
outfile << "Summary:" << endl<< endl;
while (getline(infile, sgx)) {
if (regex_match(sgx, start_rgx)) {
smatch sm;
regex_match(sgx, sm, start_rgx);
if (flag) {
outfile << ln - 1 << "(" << filename << "): Incomplete Booting";
outfile << "\t   Time: N/A" << endl;
outfile << endl;
}
flag = true;
Service service;
service.name = sm[2];
service.startTime = ptime(boost::posix_time::time_from_string(sm[1]));
services.push_back(service);
stx = sm[2];
outfile << "Device Boot - " << endl;
outfile << ln << "(" << filename << "): " << sm[1] << " Boot Start" << endl;
}
if (regex_match(sgx, succ_rgx)) {
smatch sm;
regex_match(sgx, sm, succ_rgx);
if (!services.empty()) {
Service& service = services.back();
service.completionTime = ptime(boost::posix_time::time_from_string(sm[1]));
outfile << ln << "(" << filename << "): " << sm[1] << " Boot Completed";
outfile << "\t   Time: " << (service.completionTime - service
.startTime).total_milliseconds() << "ms" << endl;
outfile << endl;
}
flag = false;
}
ln++;
}
if (flag) {
outfile << ln - 1 << "(" << filename << "): Incomplete Booting";
outfile << "\t   Time: N/A" << endl;
outfile << endl;
}
infile.close();
outfile.close();
return 0;
}
```

10.7 Output:

These are outputs of Devices 1 and 2.
The Output-1:

```
The 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
```

The Output-2:

```
The Device Boot Report
InTouch log file: device2_intouch.log
Summary:

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