Computing IV Sec 203: Project Portfolio

Christopher Lambert

Spring 2024

Contents

Time to Complete Portfolio: 8 hours

1 PS0: Hello SFML

1.1 Discussion

This project was intended to display introductory experience working with the C++ Simple and Fast Multimedia Library(SFML). Since the library is only compatible within a linux environment I was tasked with seting up a linux environment and virtual desktop via Windows Subsystem for Linux and an application called Xserver. To aid code development I was also introduced to compiler tools such as make and cpplint for compilation and formatting. The current formatting of this project's code follows Google developer standards. The goal of this project was to take the tutorial code provided and extend it to draw a sprite while having it move in accordance with the arrow keys. The image out the output can be seen in Figure: 1

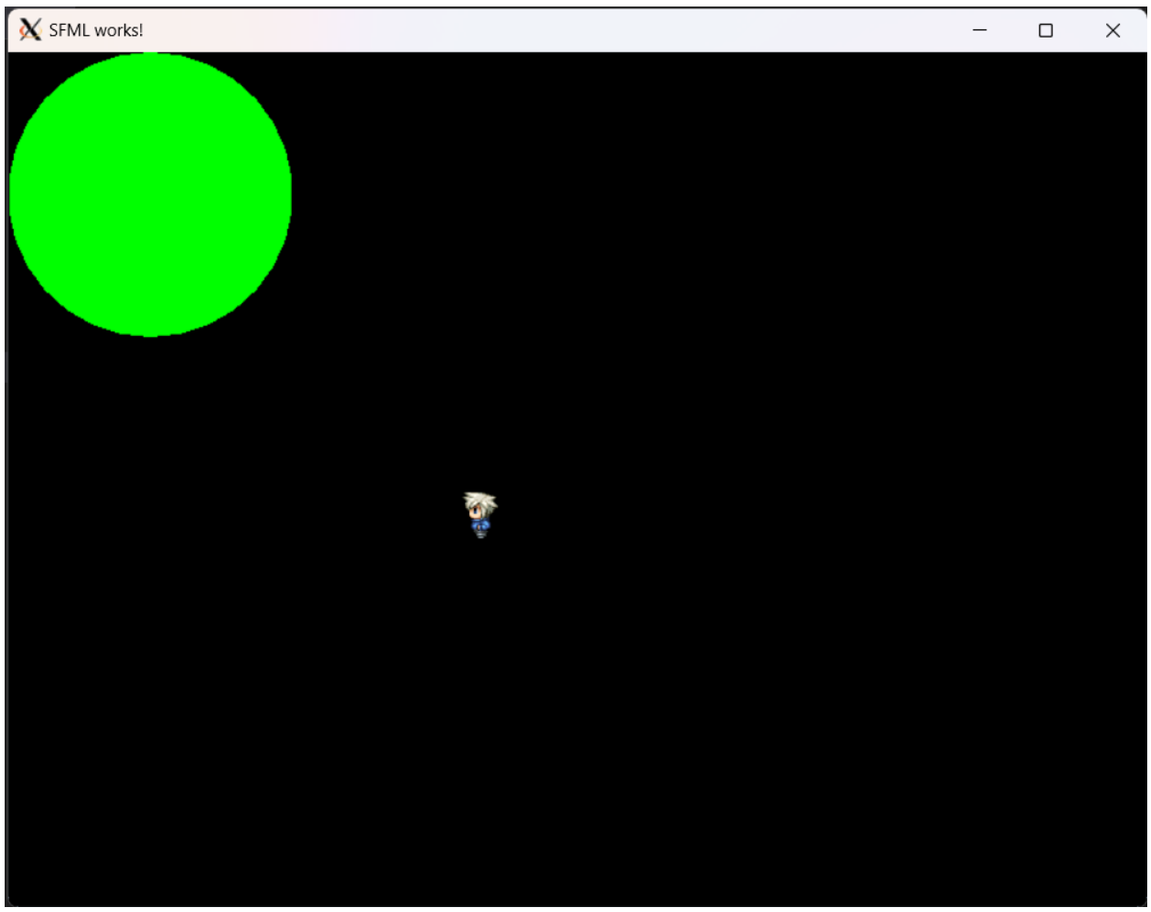

Figure 1: Window produced from running main.cpp

1.2 What I accomplished

- Setup a linux enviroment and virtual desktop
- Utilizing compiler tools such as make
- Followed Google developer standards fro code formatting
- Extended tutorial code to draw a sprite and implement arrow key movement

1.3 What I already knew

- Basic linux command line nagivation
- General linux file permissions, I/O, and executables

1.4 What I learned

- Improved understanding of SMFL library
- Using compiler tools such as make
- Code Formatting
- How to navigate through documentation

1.5 Challenges

- Configuration of the Makefile

1.6 Codebase

Listing 1: Makefile

```
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 = AnimatedSprite.hpp Animation.cpp
# Your compiled .o files
OBJECTS = Animation.o AnimatedSprite.o
# The name of your program
PROGRAM = sfml-app

.PHONY: all clean lint


all: $(PROGRAM)

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

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

clean:
    rm *.o $(PROGRAM)

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

Listing 2: main.cpp

```
// Copyright 2024 Chris Lambert
#include <SFML/Graphics.hpp>
#include <SFML/Window/Keyboard.hpp>
#include "AnimatedSprite.hpp"

int main() {
    sf::Vector2i screenDimensions(800, 600);
    sf::RenderWindow window(sf::VideoMode(screenDimensions.x,
   screenDimensions.y), "SFML works!");
    window.setFramerateLimit(60);

    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);
```

```
13
14      sf::Texture texture;
15      if (!texture.loadFromFile("./sprite.png"))
16          return EXIT_FAILURE;
17
18      sf::Sprite sprite(texture);
19      Animation walkingAnimationDown;
20      walkingAnimationDown.setSpriteSheet(texture);
21      walkingAnimationDown.addFrame(sf::IntRect(32, 0, 32, 32));
22      walkingAnimationDown.addFrame(sf::IntRect(64, 0, 32, 32));
23      walkingAnimationDown.addFrame(sf::IntRect(32, 0, 32, 32));
24      walkingAnimationDown.addFrame(sf::IntRect(0, 0, 32, 32));
25
26      Animation walkingAnimationLeft;
27      walkingAnimationLeft.setSpriteSheet(texture);
28      walkingAnimationLeft.addFrame(sf::IntRect(32, 32, 32, 32));
29      walkingAnimationLeft.addFrame(sf::IntRect(64, 32, 32, 32));
30      walkingAnimationLeft.addFrame(sf::IntRect(32, 32, 32, 32));
31      walkingAnimationLeft.addFrame(sf::IntRect(0, 32, 32, 32));
32
33      Animation walkingAnimationRight;
34      walkingAnimationRight.setSpriteSheet(texture);
35      walkingAnimationRight.addFrame(sf::IntRect(32, 64, 32, 32));
36      walkingAnimationRight.addFrame(sf::IntRect(64, 64, 32, 32));
37      walkingAnimationRight.addFrame(sf::IntRect(32, 64, 32, 32));
38      walkingAnimationRight.addFrame(sf::IntRect(0, 64, 32, 32));
39
40      Animation walkingAnimationUp;
41      walkingAnimationUp.setSpriteSheet(texture);
42      walkingAnimationUp.addFrame(sf::IntRect(32, 96, 32, 32));
43      walkingAnimationUp.addFrame(sf::IntRect(64, 96, 32, 32));
44      walkingAnimationUp.addFrame(sf::IntRect(32, 96, 32, 32));
45      walkingAnimationUp.addFrame(sf::IntRect(0, 96, 32, 32));
46
47      Animation* currentAnimation = &walkingAnimationDown;
48
49      // set up AnimatedSprite
50      AnimatedSprite animatedSprite(sf::seconds(0.2), true, false);
51      animatedSprite.setPosition(sf::Vector2f(screenDimensions / 2));
52
53      sf::Clock frameClock;
54
55      float speed = 80.f;
56      bool noKeyWasPressed = true;
57
58      while (window.isOpen()) {
59          sf::Event event;
60          while (window.pollEvent(event)) {
61              if (event.type == sf::Event::Closed)
62                  window.close();
63          }
64          sf::Time frameTime = frameClock.restart();
65
66          // if a key was pressed set the correct animation and move correctly
67          sf::Vector2f movement(0.f, 0.f);
68          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
69              currentAnimation = &walkingAnimationUp;
70              movement.y -= speed;
71              noKeyWasPressed = false;
```

```
72          }
73          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
74              currentAnimation = &walkingAnimationDown;
75              movement.y += speed;
76              noKeyWasPressed = false;
77          }
78          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
79              currentAnimation = &walkingAnimationLeft;
80              movement.x -= speed;
81              noKeyWasPressed = false;
82          }
83          if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
84              currentAnimation = &walkingAnimationRight;
85              movement.x += speed;
86              noKeyWasPressed = false;
87          }
88          animatedSprite.play(*currentAnimation);
89          animatedSprite.move(movement * frameTime.asSeconds());
90
91          // if no key was pressed stop the animation
92          if (noKeyWasPressed) {
93              animatedSprite.stop();
94          }
95          noKeyWasPressed = true;
96
97          // update AnimatedSprite
98          animatedSprite.update(frameTime);
99
100         window.clear();
101         window.draw(shape);
102         window.draw(animatedSprite);
103         window.display();
104     }
105     return 0;
106 }
```

Listing 3: Animation.cpp

```
1  ////////////////////////////////////////////////////////////
2  //
3  // Copyright (C) 2014 Maximilian Wagenbach (aka. Foaly) (foaly.f@web.de)
4  //
5  // This software is provided 'as-is', without any express or implied
       warranty.
6  // In no event will the authors be held liable for any damages
7  // arising from the use of this software.
8  //
9  // Permission is granted to anyone to use this software for any purpose,
10 // including commercial applications, and to alter it and redistribute it
       freely,
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 // you must not claim that you wrote the original software.
15 // If you use this software in a product, an acknowledgment
16 // in the product documentation would be appreciated but is not required.
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 // and must not be misrepresented as being the original software.
20 //
```

```
21 // 3. This notice may not be removed or altered from any source distribution
   .
22 //
23 ////////////////////////////////////////////////////////////
24
25 #include "Animation.hpp"
26
27 Animation::Animation() : m_texture(NULL) {
28 }
29
30 void Animation::addFrame(sf::IntRect rect) {
31     m_frames.push_back(rect);
32 }
33
34 void Animation::setSpriteSheet(const sf::Texture& texture) {
35     m_texture = &texture;
36 }
37
38 const sf::Texture* Animation::getSpriteSheet() const {
39     return m_texture;
40 }
41
42 std::size_t Animation::getSize() const {
43     return m_frames.size();
44 }
45
46 const sf::IntRect& Animation::getFrame(std::size_t n) const {
47     return m_frames[n];
48 }
```

Listing 4: Animation.hpp

```
1  ////////////////////////////////////////////////////////////
2  //
3  // Copyright (C) 2014 Maximilian Wagenbach (aka. Foaly) (foaly.f@web.de)
4  //
5  // This software is provided 'as-is', without any express or implied
   warranty.
6  // In no event will the authors be held liable for any damages
7  // arising from the use of this software.
8  //
9  // Permission is granted to anyone to use this software for any purpose,
10 // including commercial applications, and to alter it and redistribute it
   freely,
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 // you must not claim that you wrote the original software.
15 // If you use this software in a product, an acknowledgment
16 // in the product documentation would be appreciated but is not required.
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 // and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source distribution
   .
22 //
23 ////////////////////////////////////////////////////////////
24
25 #ifndef ANIMATION_INCLUDE
```

```
26 #define ANIMATION_INCLUDE
27
28 #include <vector>
29 #include <SFML/Graphics/Rect.hpp>
30 #include <SFML/Graphics/Texture.hpp>
31
32 class Animation {
33  public:
34     Animation();
35
36     void addFrame(sf::IntRect rect);
37     void setSpriteSheet(const sf::Texture& texture);
38     const sf::Texture* getSpriteSheet() const;
39     std::size_t getSize() const;
40     const sf::IntRect& getFrame(std::size_t n) const;
41
42  private:
43     std::vector<sf::IntRect> m_frames;
44     const sf::Texture* m_texture;
45 };
46
47 #endif
48 // ANIMATION_INCLUDE
```

Listing 5: AnimatedSprite.cpp

```
1  ////////////////////////////////////////////////////////////
2  //
3  // Copyright (C) 2014 Maximilian Wagenbach (aka. Foaly) (foaly.f@web.de)
4  //
5  // This software is provided 'as-is', without any express or implied
       warranty.
6  // In no event will the authors be held liable for any damages arising
7  // from the use of this software.
8  //
9  // Permission is granted to anyone to use this software for any purpose,
10 // including commercial applications, and to alter it and redistribute it
       freely,
11 // subject to the following restrictions:
12 //
13 // 1. The origin of this software must not be misrepresented;
14 // you must not claim that you wrote the original software.
15 // If you use this software in a product, an acknowledgment
16 // in the product documentation would be appreciated but is not required.
17 //
18 // 2. Altered source versions must be plainly marked as such,
19 // and must not be misrepresented as being the original software.
20 //
21 // 3. This notice may not be removed or altered from any source distribution
       .
22 //
23 ////////////////////////////////////////////////////////////
24
25 #include "AnimatedSprite.hpp"
26
27 AnimatedSprite::AnimatedSprite(sf::Time frameTime, bool paused, bool looped)
       :
28     m_animation(NULL), m_frameTime(frameTime), m_currentFrame(0),
29     m_isPaused(paused), m_isLooped(looped), m_texture(NULL) {
30 }
```

```
31
32 void AnimatedSprite::setAnimation(const Animation& animation) {
33     m_animation = &animation;
34     m_texture = m_animation->getSpriteSheet();
35     m_currentFrame = 0;
36     setFrame(m_currentFrame);
37 }
38
39 void AnimatedSprite::setFrameTime(sf::Time time) {
40     m_frameTime = time;
41 }
42
43 void AnimatedSprite::play() {
44     m_isPaused = false;
45 }
46
47 void AnimatedSprite::play(const Animation& animation) {
48     if (getAnimation() != &animation)
49         setAnimation(animation);
50     play();
51 }
52
53 void AnimatedSprite::pause() {
54     m_isPaused = true;
55 }
56
57 void AnimatedSprite::stop() {
58     m_isPaused = true;
59     m_currentFrame = 0;
60     setFrame(m_currentFrame);
61 }
62
63 void AnimatedSprite::setLooped(bool looped) {
64     m_isLooped = looped;
65 }
66
67 void AnimatedSprite::setColor(const sf::Color& color) {
68     // Update the vertices' color
69     m_vertices[0].color = color;
70     m_vertices[1].color = color;
71     m_vertices[2].color = color;
72     m_vertices[3].color = color;
73 }
74
75 const Animation* AnimatedSprite::getAnimation() const {
76     return m_animation;
77 }
78
79 sf::FloatRect AnimatedSprite::getLocalBounds() const {
80     sf::IntRect rect = m_animation->getFrame(m_currentFrame);
81
82     float width = static_cast<float>(std::abs(rect.width));
83     float height = static_cast<float>(std::abs(rect.height));
84
85     return sf::FloatRect(0.f, 0.f, width, height);
86 }
87
88 sf::FloatRect AnimatedSprite::getGlobalBounds() const {
89     return getTransform().transformRect(getLocalBounds());
```

```
90  }
91
92  bool AnimatedSprite::isLooped() const {
93      return m_isLooped;
94  }
95
96  bool AnimatedSprite::isPlaying() const {
97      return !m_isPaused;
98  }
99
100 sf::Time AnimatedSprite::getFrameTime() const {
101     return m_frameTime;
102 }
103
104 void AnimatedSprite::setFrame(std::size_t newFrame, bool resetTime) {
105     if (m_animation) {
106         // calculate new vertex positions and texture coordiantes
107         sf::IntRect rect = m_animation->getFrame(newFrame);
108
109         m_vertices[0].position = sf::Vector2f(0.f, 0.f);
110         m_vertices[1].position = sf::Vector2f(0.f, static_cast<float>(rect.
    height));
111         m_vertices[2].position = sf::Vector2f(static_cast<float>(rect.width)
    ,
112                                               static_cast<float>(rect.height))
    ;
113         m_vertices[3].position = sf::Vector2f(static_cast<float>(rect.width)
    , 0.f);
114
115         float left = static_cast<float>(rect.left) + 0.0001f;
116         float right = left + static_cast<float>(rect.width);
117         float top = static_cast<float>(rect.top);
118         float bottom = top + static_cast<float>(rect.height);
119
120         m_vertices[0].texCoords = sf::Vector2f(left, top);
121         m_vertices[1].texCoords = sf::Vector2f(left, bottom);
122         m_vertices[2].texCoords = sf::Vector2f(right, bottom);
123         m_vertices[3].texCoords = sf::Vector2f(right, top);
124     }
125
126     if (resetTime)
127         m_currentTime = sf::Time::Zero;
128 }
129
130 void AnimatedSprite::update(sf::Time deltaTime) {
131     // if not paused and we have a valid animation
132     if (!m_isPaused && m_animation) {
133         // add delta time
134         m_currentTime += deltaTime;
135
136         // if current time is bigger then the frame time advance one frame
137         if (m_currentTime >= m_frameTime) {
138             // reset time, but keep the remainder
139          m_currentTime = sf::microseconds(m_currentTime.asMicroseconds() %
140                                           m_frameTime.asMicroseconds());
141
142             // get next Frame index
143             if (m_currentFrame + 1 < m_animation->getSize()) {
144                 m_currentFrame++;
```

```
            } else {
                // animation has ended
                if (!m_isLooped) {
                    m_isPaused = true;
                } else {
                    m_currentFrame = 0;
                    // reset to start
                }
            }

            // set the current frame, not reseting the time
            setFrame(m_currentFrame, false);
        }
    }
}

void AnimatedSprite::draw(sf::RenderTarget& target, sf::RenderStates states)
    const {
    if (m_animation && m_texture) {
        states.transform *= getTransform();
        states.texture = m_texture;
        target.draw(m_vertices, 4, sf::Quads, states);
    }
}
```

Listing 6: AnimatedSprite.hpp

```
////////////////////////////////////////////////////////////
//
// Copyright (C) 2014 Maximilian Wagenbach (aka. Foaly) (foaly.f@web.de)
//
// This software is provided 'as-is', without any express or implied
   warranty.
// In no event will the authors be held liable for any damages
// arising from the use of this software.
//
// Permission is granted to anyone to use this software for any purpose,
// including commercial applications, and to alter it and redistribute it
   freely,
// subject to the following restrictions:
//
// 1. The origin of this software must not be misrepresented;
// you must not claim that you wrote the original software.
// If you use this software in a product, an acknowledgment
// in the product documentation would be appreciated but is not required.
//
// 2. Altered source versions must be plainly marked as such,
// and must not be misrepresented as being the original software.
//
// 3. This notice may not be removed or altered from any source distribution
   .
//
////////////////////////////////////////////////////////////

#ifndef ANIMATEDSPRITE_INCLUDE
#define ANIMATEDSPRITE_INCLUDE

#include <SFML/Graphics/RenderTarget.hpp>
#include <SFML/System/Time.hpp>
#include <SFML/Graphics/Drawable.hpp>
```

```cpp
#include <SFML/Graphics/Transformable.hpp>
#include <SFML/System/Vector2.hpp>

#include "Animation.hpp"

class AnimatedSprite : public sf::Drawable, public sf::Transformable {
 public:
    explicit AnimatedSprite(sf::Time frameTime = sf::seconds(0.2f),
                            bool paused = false, bool looped = true);

    void update(sf::Time deltaTime);
    void setAnimation(const Animation& animation);
    void setFrameTime(sf::Time time);
    void play();
    void play(const Animation& animation);
    void pause();
    void stop();
    void setLooped(bool looped);
    void setColor(const sf::Color& color);
    const Animation* getAnimation() const;
    sf::FloatRect getLocalBounds() const;
    sf::FloatRect getGlobalBounds() const;
    bool isLooped() const;
    bool isPlaying() const;
    sf::Time getFrameTime() const;
    void setFrame(std::size_t newFrame, bool resetTime = true);

 private:
    const Animation* m_animation;
    sf::Time m_frameTime;
    sf::Time m_currentTime;
    std::size_t m_currentFrame;
    bool m_isPaused;
    bool m_isLooped;
    const sf::Texture* m_texture;
    sf::Vertex m_vertices[4];

    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
   const;
};

#endif
// ANIMATEDSPRITE_INCLUDE
```

2 PS1: LFSR / PhotoMagic

2.1 Discussion

For this assignment, I implemented a program that produces pseudo-random bits by simulating a linear feedback shift register (LFSR), and used it to implement a simple form of encryption for digital pictures. The first half of the assignment involved implementing the FibLFSR class and writing unit tests using the Boost test framework to ensure the class and its methods work properly. The second half will utilize the implemented algorithm to "encrypt" images and write it to a file.

To execute the program you need 3 command line arguments; the output filename, the input filename, and the 16 bit binary seed for the LSFR algorithm. If you need to decrypt the image you can simply re-run the program on the encrypted image with the same seed to decrpyt it. An example of the pre-encryption and post-encryption image can been viewd in Figure 2 and Figure 3.

Figure 2: Pre-encryption Image

2.2 What I accomplished

- Implementing a linear shift feedback register
- Encoding the algorithm to encrypt/decrypt
- Constructed a command line application
- Extended tutorial code to draw a sprite and implement arrow key movement

2.3 What I already knew

- How digital images are interpreted via pixels with rgb settings
- How to accept command line arguments in c++ executable

Figure 3: Post-encryption Image

2.4 What I learned

- The linear shift feedback register algorithm
- More depth on C++ bitwise operations

2.5 Challenges

- Implementing the LSFR algorithm

2.6 Codebase

Listing 7: Makefile

```
# Compiler variables
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
# ar command variables
AR = ar
ARFLAGS = -rcs
# Your .hpp files
DEPS = FibLFSR.hpp
# Your compiled .o files
OBJECTS = FibLFSR.o PhotoMagic.o
# The name of your program
PROGRAM = test
PROGRAM2 = PhotoMagic
PHOTO_PROGRAM = static
# Static library name
STATIC_LIB = PhotoMagic.a

.PHONY: all clean lint

```

```
all: $(PROGRAM) $(PROGRAM2) $(PHOTO_PROGRAM)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<
# test:
$(PROGRAM): test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)
#Photomagic:
$(PROGRAM2): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)
# PhotoMagic static library:
$(PHOTO_PROGRAM) : $(OBJECTS)
    $(AR) $(ARFLAGS) $(STATIC_LIB) $^

clean:
    rm *.o $(PROGRAM) $(PROGRAM2) $(STATIC_LIB)

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

Listing 8: main.cpp

```
// Copyright 2024 Chris Lambert
//  using SFML to load a file, manipulate its pixels, write it to disk

// g++ -o Photomagic main.cpp PhotoMagic.cpp FibLFSR.cpp -lsfml-graphics -lsfml-window -lsfml-system
#include <iostream>
#include <string>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

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

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

int main(int argc, char* argv[]) {
  // read in command line arguments and construct the seed
  string input = argv[1];
  string output = argv[2];
  string seed = argv[3];

  FibLFSR algorithm(seed);

  sf::Image image;
  if (!image.loadFromFile(input))
    return -1;
  // encrypt the first image
  transform(image, &algorithm);
  // save the encrypted image

  sf::Vector2u size = image.getSize();
  sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "Encrypted Image");

  // Encrytped image sprite
```

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

  sf::Sprite sprite;
  sprite.setTexture(texture);

  while (window1.isOpen()) {
    sf::Event event;
    while (window1.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
        window1.close();
    }
    window1.clear();
    window1.draw(sprite);
    window1.display();
  }

  //   write the file
  if (!image.saveToFile(output))
    return -1;

  return 0;
}
```

Listing 9: FibLFSR.cpp

```
// Copyright 2024 Chris Lambert

#include "FibLFSR.hpp"

#include <cmath>
#include <exception>
#include <iostream>
#include <string>

using PhotoMagic::FibLFSR;

// defines your seed
FibLFSR::FibLFSR(string seed) : seed(seed), isValid(true) {
  // checking for a valid seed
  if (seed.length() != 16) {
    isValid = false;
  }
  for (auto i : seed) {
    if (i != '0' && i != '1') {
      isValid = false;
      break;
    }
  }
}
// helper function for char to int conversion
int FibLFSR::check(int index) const { return seed[index] == '0' ? 0 : 1; }

int FibLFSR::step() {
  // all invalid seeds get stored but not manipulated
  if (!this->isValid)
    return 0;

  tap15 = check(0);
  tap13 = check(2);
```

```
  tap12 = check(3);
  tap10 = check(5);
  // automatic type conversion with 1 and 0
  bool step = tap15;
  step = (step != tap13);
  step = (step != tap12);
  step = (step != tap10);

  string newSeed = this->seed.substr(1, this->seed.length() - 1);
  newSeed += (step) ? '1' : '0';
  this->seed = newSeed;

  return step;
}

int FibLFSR::generate(int k) {
  // invalid parameters wont be manipulated and return 0
  if (k > 16 || k < 0) {
    return 0;
  }
  if (!this->isValid) {
    return 0;
  }
  // perform the step k times
  for (int i = 0; i < k; i++) {
    step();
  }
  // splices the string to the k bit sequence and converts it using a
   reverse iterator/counter
  string binary = seed.substr(seed.length() - k, k);
  int count = 0;
  int finalValue = 0;

  for (auto i = binary.rbegin(); i != binary.rend(); ++i) {
    if (*i == '1')
      finalValue += pow(2, count);
    count++;
  }
  return finalValue;
}
// ostream operator overload
ostream &PhotoMagic::operator<<(ostream &out, const FibLFSR &lfsr) {
  out << lfsr.seed;
  return out;
}
```

Listing 10: FibLFSR.hpp

```
// Copyright 2024 Chris Lambert
#pragma once
#include <iostream>

using std::ostream;
using std::string;

namespace PhotoMagic {
class FibLFSR {
 public:
  friend ostream &operator<<(ostream &out, const FibLFSR &lfsr);
  // constructor to create LFSR with the given initial seed and tap
```

```
  explicit FibLFSR(string seed);
  // simulate one step and return the new bit as 0 or 1
  int step();
  // simulate k steps and return k-bit integer
  int generate(int k);

 private:
  // helper functions
  int check(int index) const;
  string seed;
  int tap15, tap13, tap12, tap10;
  bool isValid;
};

ostream &operator<<(ostream &out, const FibLFSR &lfsr);

}  // namespace PhotoMagic
```

Listing 11: PhotoMagic.cpp

```
// Copyright 2023 Chris Lambert
#include "PhotoMagic.hpp"

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

#include "FibLFSR.hpp"

void PhotoMagic::transform(sf::Image& image, FibLFSR* seed) {
  auto size = image.getSize();

  for (unsigned int i = 0; i < size.x; ++i) {
    for (unsigned int j = 0; j < size.y; ++j) {
      sf::Color p = image.getPixel(i, j);

      auto p_red = p.r;
      auto p_green = p.g;
      auto p_blue = p.b;

      int r_seed = seed->generate(15);
      auto new_red = p_red ^ r_seed;

      int g_seed = seed->generate(15);
      auto new_green = p_green ^ g_seed;

      int b_seed = seed->generate(15);
      auto new_blue = p_blue ^ b_seed;

      sf::Color color(new_red, new_green, new_blue);

      image.setPixel(i, j, color);
    }
  }
}
```

Listing 12: PhotoMagic.hpp

```
// Copyright 2024 Chris Lambert
#pragma once
#include <SFML/Graphics.hpp>
```

```cpp
#include <SFML/System.hpp>
#include <SFML/Window.hpp>

#include "FibLFSR.hpp"

namespace PhotoMagic {
// Transforms image using FibLFSR
void transform(sf::Image&, FibLFSR*);
// Display an encrypted copy of the picture, using the LFSR to do the
   encryption
}  // namespace PhotoMagic
```

Listing 13: test.cpp

```cpp
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly
// test.cpp for PS1a
// updated 1/8/2024

#include <iostream>
#include <sstream>
#include <string>

#include "FibLFSR.hpp"

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

using PhotoMagic::FibLFSR;

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

BOOST_AUTO_TEST_CASE(testGenerateInstr) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_EQUAL(l.generate(9), 51);
}
BOOST_AUTO_TEST_CASE(testGenerateAfterStep) {
  FibLFSR l("1011011000000111");
  l.step();
  l.step();
  BOOST_REQUIRE_EQUAL(l.generate(8), 123);
}
BOOST_AUTO_TEST_CASE(testGenerateMaxBits) {
  FibLFSR l("1011011000000111");
  BOOST_REQUIRE_EQUAL(l.generate(16), 7872);
}
BOOST_AUTO_TEST_CASE(testOutputStream) {
  FibLFSR l("1011011000000111");
```

```
47    std::stringstream output;
48    std::streambuf* buffer = std::cout.rdbuf();  // Save cout buffer
49    std::cout.rdbuf(output.rdbuf());             // redirect cout to output
       object
50    std::cout << l;
51    std::cout.rdbuf(buffer);  // restore cout
52    BOOST_CHECK_EQUAL(output.str(), "1011011000000111");
53  }
54  BOOST_AUTO_TEST_CASE(testOutputAfterInstr) {
55    FibLFSR l("1011011000000111");
56    l.step();
57    l.step();
58    l.generate(13);
59    std::stringstream output;
60    std::streambuf* buffer = std::cout.rdbuf();
61    std::cout.rdbuf(output.rdbuf());
62    std::cout << l;
63    std::cout.rdbuf(buffer);
64    BOOST_CHECK_EQUAL(output.str(), "1000111011100000");
65  }
```

3 PS2: Pythagoras Tree

3.1 Discussion

For PS2, I completed a program that draws a Pythagoras tree, a fractal made of squares that enclose right triangles, often used to illustrate the Pythagorean theorem. The program uses recursion to draw smaller trees within each square. By specifying the size of the base square and the depth of recursion, the program creates an image that fits within a defined window size. Understanding the mathematical principles behind the Pythagorean tree, such as the relationships between the side lengths of the squares and the angles involved, was crucial in implementing the recursive algorithm correctly. The program accepts 2 arguments passed in from stdin; the first being initial length of the square base and the other being the depth of the recursion. Based on the 2 arguments the program will autimatically size the window to display the full tree.

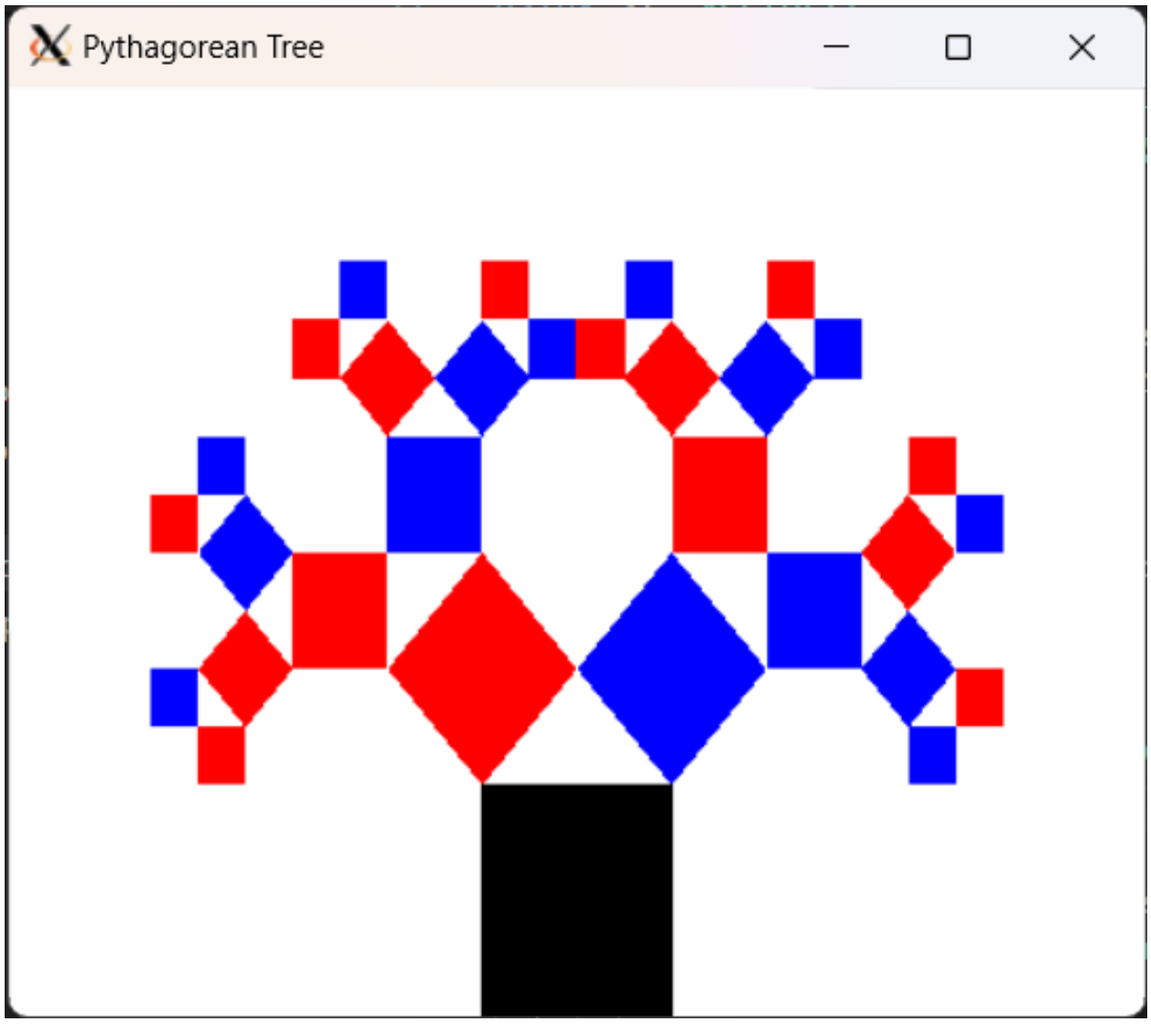

Figure 4: Pytharean Tree with base length 50 and depth 4

3.2 What I accomplished

- Implemented the Pythagoras tree fractal generation using recursion
- Managed to correctly calculate the sizes and angles for each level of recursion.
- Developed a program that automatically sizes the window to fit the entire tree.

3.3 What I already knew

- Recursion programming concepts
- Basic geopmetry and trigonometry

3.4 What I learned

- Experience on the graphical representation of geometric shapes
- How to rotate a shape with respect to the window dimensions

3.5 Challenges

- Implementing the trigonometric calculations with respect to the shapes current rotation
- Returning the proper cordinates with respect to how the windows origin point is in the top left corner whereas when one woudld traditionally learn from a centered origin point posed a difficulty so I do not thin the implementation is fully correct

3.6 Codebase

Listing 14: main.cpp

```
1  // Copyright 2024 Chris Lambert
2  //  using SFML to load a file, manipulate its pixels, write it to disk
3
4  // g++ -o ps2 main.cpp -lsfml-graphics -lsfml-window -lsfml-system
5  #include <SFML/Graphics.hpp>
6  #include <SFML/System.hpp>
7  #include <SFML/Window.hpp>
8  #include <cmath>
9  #include <iostream>
10 #include <string>
11
12 using namespace std;
13 using sf::Vector2f;
14
15 Vector2f CalcOffset(sf::RectangleShape square, Vector2f offset);
16
17 int main(int argc, char* argv[]) {
18   // read in command line arguments and construct the seed (start with #1)
19
20   // Initial square
21   sf::RectangleShape square;
22   square.setOrigin(square.getSize().x / 2, square.getSize().y / 2);
23   square.setSize(Vector2f(100, 100));
24   square.setFillColor(sf::Color::Black);
25   square.setPosition(250, 300);
26
27   auto point1 = square.getTransform().transformPoint(square.getPoint(0));
28   auto point2 = square.getTransform().transformPoint(square.getPoint(1));
29   auto point3 = square.getTransform().transformPoint(square.getPoint(2));
30   auto point4 = square.getTransform().transformPoint(square.getPoint(3));
31
32   Vector2f offset(square.getSize().x / 2.0, square.getSize().y / 2.0);
33
34   auto newSize = [](Vector2f vector) { return Vector2f(vector.x * sin(45 *
      M_PI / 180.0f), vector.y * sin(45 * M_PI / 180.0f)); };
35
36   auto getCorner = [](sf::RectangleShape x) { return x.getTransform().
      transformPoint(x.getPoint(1)); };
37   auto newOffset = [](sf::RectangleShape x) { return Vector2f(x.getSize().x
      / 2.0, x.getSize().y / 2.0); };
38   auto getCorner1 = [](sf::RectangleShape x) { return x.getTransform().
      transformPoint(x.getPoint(0)); };
39
```

```
40  // cout << dot1.getRotation() << endl;
41  auto offset1 = newOffset(square);
42
43  sf::RectangleShape ltest(newSize(square.getSize()));
44  ltest.setOrigin(ltest.getSize().x / 2, ltest.getSize().y / 2);
45  ltest.setFillColor(sf::Color::Blue);
46  ltest.rotate(-45);
47  cout << " should be 315 " << ltest.getRotation() << endl;
48  ltest.setPosition(getCorner1(square) + CalcOffset(ltest, offset));  // 315
49
50  auto loffset = newOffset(ltest);
51
52  sf::RectangleShape ltest2(newSize(ltest.getSize()));
53  ltest2.setOrigin(ltest2.getSize().x / 2, ltest2.getSize().y / 2);
54  ltest2.setFillColor(sf::Color::Blue);
55  ltest2.rotate(ltest.getRotation() - 45);
56  cout << ltest2.getRotation() << endl;
57  ltest2.setPosition(getCorner1(ltest) + CalcOffset(ltest2, loffset));  //
     270
58
59  auto loffset1 = newOffset(ltest2);
60
61  sf::RectangleShape ltest3(newSize(ltest2.getSize()));
62  ltest3.setOrigin(ltest3.getSize().x / 2, ltest3.getSize().y / 2);
63  ltest3.setFillColor(sf::Color::Blue);
64  ltest3.rotate(ltest2.getRotation() - 45);
65  cout << ltest3.getRotation() << endl;
66  ltest3.setPosition(getCorner1(ltest2) + Vector2f(-loffset1.x, 0));  // 225
67
68  auto loffset2 = newOffset(ltest3);
69
70  sf::RectangleShape ltest4(newSize(ltest3.getSize()));
71  ltest4.setOrigin(ltest4.getSize().x / 2, ltest4.getSize().y / 2);
72  ltest4.setFillColor(sf::Color::Blue);
73  ltest4.rotate(ltest3.getRotation() - 45);
74  cout << ltest4.getRotation() << endl;
75  ltest4.setPosition(getCorner1(ltest3) + Vector2f(-loffset2.x / sqrt(2),
     loffset2.y / sqrt(2)));  // 180
76
77  auto loffset3 = newOffset(ltest4);
78
79  sf::RectangleShape ltest5(newSize(ltest4.getSize()));
80  ltest5.setOrigin(ltest5.getSize().x / 2, ltest5.getSize().y / 2);
81  ltest5.setFillColor(sf::Color::Blue);
82  ltest5.rotate(ltest4.getRotation() - 45);
83  cout << ltest5.getRotation() << endl;
84  ltest5.setPosition(getCorner1(ltest4) + Vector2f(0, loffset3.y));  // 135
85
86  auto loffset4 = newOffset(ltest5);
87
88  sf::RectangleShape ltest6(newSize(ltest5.getSize()));
89  ltest6.setOrigin(ltest6.getSize().x / 2, ltest6.getSize().y / 2);
90  ltest6.setFillColor(sf::Color::Blue);
91  ltest6.rotate(ltest5.getRotation() - 45);
92  cout << ltest6.getRotation() << endl;
93  ltest6.setPosition(getCorner1(ltest5) + CalcOffset(ltest6, loffset4));  //
      90
94
95  auto loffset5 = newOffset(ltest6);
```

```
96
97    sf::RectangleShape ltest7(newSize(ltest6.getSize()));
98    ltest7.setOrigin(ltest7.getSize().x / 2, ltest7.getSize().y / 2);
99    ltest7.setFillColor(sf::Color::Blue);
100   ltest7.rotate(ltest6.getRotation() - 45);
101   cout << ltest7.getRotation() << endl;
102   ltest7.setPosition(getCorner1(ltest6) + Vector2f(loffset5.x, 0));  // 45
103
104   auto loffset6 = newOffset(ltest7);
105
106   sf::RectangleShape ltest8(newSize(ltest7.getSize()));
107   ltest8.setOrigin(ltest8.getSize().x / 2, ltest8.getSize().y / 2);
108   ltest8.setFillColor(sf::Color::Blue);
109   ltest8.rotate(ltest7.getRotation() - 45);
110   cout << ltest8.getRotation() << endl;
111   ltest8.setPosition(getCorner1(ltest7) + Vector2f(loffset6.x / sqrt(2), -
       loffset6.y / sqrt(2)));  // 0
112
113   sf::RenderWindow window1(sf::VideoMode(600, 400), "Pythagorean Tree");
114   while (window1.isOpen()) {
115     sf::Event event;
116     while (window1.pollEvent(event)) {
117       if (event.type == sf::Event::Closed)
118         window1.close();
119     }
120     window1.clear(sf::Color::White);
121     window1.draw(square);
122
123     window1.draw(ltest);
124     window1.draw(ltest2);
125     window1.draw(ltest3);
126     window1.draw(ltest4);
127     window1.draw(ltest5);
128     window1.draw(ltest6);
129     window1.draw(ltest7);
130     window1.draw(ltest8);
131     window1.display();
132   }
133
134   return 0;
135 }
136
137 Vector2f CalcOffset(sf::RectangleShape square, Vector2f offset) {
138   auto rotation = square.getRotation();
139
140   if (rotation == 0) {
141     return Vector2f(offset.x / sqrt(2), -offset.y / sqrt(2));
142   } else if (rotation == 45) {
143     return Vector2f(offset.x, 0);
144   } else if (rotation == 90) {
145     return Vector2f(offset.x / sqrt(2), offset.y / sqrt(2));
146   } else if (rotation == 135) {
147     return Vector2f(0, offset.y);
148   } else if (rotation == 180) {
149     return Vector2f(-offset.x / sqrt(2), offset.y / sqrt(2));
150   } else if (rotation == 225) {
151     return Vector2f(-offset.x, 0);
152   } else if (rotation == 270) {
153     return Vector2f(-offset.x / sqrt(2), -offset.y / sqrt(2));
```

```
  } else if (rotation == 315) {
    return Vector2f(0, -offset.y);
  }
}
```

Listing 15: PTree.cpp

```
#include "PTree.hpp"

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <cmath>
#include <iostream>
#include <string>

using namespace std;
using sf::Vector2f;

Vector2f CalcOffset(sf::RectangleShape square, Vector2f offset);
Vector2f CalcOffset2(sf::RectangleShape square, Vector2f offset);

PTree::PTree(double length, int depth) : baseLength(length), depth(depth) {
  sf::RectangleShape square;
  square.setSize(Vector2f(baseLength, baseLength));
  square.setOrigin(square.getSize().x / 2, square.getSize().y / 2);
  square.setFillColor(sf::Color::Black);
  square.setPosition(6 * length / 2, 4 * length - (baseLength / 2));

  cout << square.getRotation() << endl;

  pTree(square, depth);

  sf::RenderWindow window1(sf::VideoMode(6 * length, 4 * length), "
   Pythagorean Tree");
  while (window1.isOpen()) {
    sf::Event event;
    while (window1.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
        window1.close();
    }
    window1.clear(sf::Color::White);
    window1.draw(square);
    for (auto i = shapes.begin(); i != shapes.end(); ++i) {
      window1.draw(*i);
    }
    window1.display();
  }
}

void PTree::pTree(sf::RectangleShape parent, int depth) {
  if (depth == 0)
    return;

  auto newSize = [](Vector2f vector) { return Vector2f(vector.x * sin(45 *
   M_PI / 180.0f), vector.y * sin(45 * M_PI / 180.0f)); };
  auto getCorner = [](sf::RectangleShape x) { return x.getTransform().
   transformPoint(x.getPoint(1)); };
  auto newOffset = [](sf::RectangleShape x) { return Vector2f(x.getSize().x
   / 2.0, x.getSize().y / 2.0); };
```

```
  auto getCorner1 = [](sf::RectangleShape x) { return x.getTransform().
   transformPoint(x.getPoint(0)); };
  auto offset = newOffset(parent);

  sf::RectangleShape childShape1;
  childShape1.setSize(newSize(parent.getSize()));
  childShape1.setOrigin(childShape1.getSize().x / 2.0, childShape1.getSize()
   .y / 2.0);
  childShape1.setFillColor(sf::Color::Blue);
  childShape1.rotate(parent.getRotation() + 45);
  childShape1.setPosition(getCorner(parent) + CalcOffset(childShape1, offset
   ));
  // cout << childShape1.getRotation() << endl;

  sf::RectangleShape childShape2;
  childShape2.setSize(newSize(parent.getSize()));
  childShape2.setOrigin(childShape2.getSize().x / 2.0, childShape2.getSize()
   .y / 2.0);
  childShape2.setFillColor(sf::Color::Red);
  childShape2.rotate(parent.getRotation() - 45);
  childShape2.setPosition(getCorner1(parent) + CalcOffset2(childShape2,
   offset));
  cout << "child shape 2 roto: " << childShape2.getRotation() << endl;

  this->shapes.push_back(childShape1);
  this->shapes.push_back(childShape2);

  // Recursively call pTree for each child shape
  pTree(childShape1, depth - 1);
  pTree(childShape2, depth - 1);
}

Vector2f CalcOffset(sf::RectangleShape square, Vector2f offset) {
  auto rotation = square.getRotation();

  if (rotation == 0) {
    return Vector2f(-offset.x / sqrt(2), -offset.y / sqrt(2));
  } else if (rotation == 45) {
    return Vector2f(0, -offset.y);
  } else if (rotation == 90) {
    return Vector2f(offset.x / sqrt(2), -offset.y / sqrt(2));
  } else if (rotation == 135) {
    return Vector2f(offset.x, 0);
  } else if (rotation == 180) {
    return Vector2f(offset.x / sqrt(2), offset.y / sqrt(2));
  } else if (rotation == 225) {
    return Vector2f(0, offset.y);
  } else if (rotation == 270) {
    return Vector2f(-offset.x / sqrt(2), offset.y / sqrt(2));
  } else if (rotation == 315) {
    return Vector2f(-offset.x, 0);
  }
}

Vector2f CalcOffset2(sf::RectangleShape square, Vector2f offset) {
  auto rotation = square.getRotation();

  if (rotation == 0) {
    return Vector2f(offset.x / sqrt(2), -offset.y / sqrt(2));
```

```
  } else if (rotation == 45) {
    return Vector2f(offset.x, 0);
  } else if (rotation == 90) {
    return Vector2f(offset.x / sqrt(2), offset.y / sqrt(2));
  } else if (rotation == 135) {
    return Vector2f(0, offset.y);
  } else if (rotation == 180) {
    return Vector2f(-offset.x / sqrt(2), offset.y / sqrt(2));
  } else if (rotation == 225) {
    return Vector2f(-offset.x, 0);
  } else if (rotation == 270) {
    return Vector2f(-offset.x / sqrt(2), -offset.y / sqrt(2));
  } else if (rotation == 315) {
    return Vector2f(0, -offset.y);
  }
}
```

Listing 16: PTree.hpp

```
#pragma once

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

class PTree {
 public:
  PTree(double length, int depth);

  ~PTree() {}

 private:
  void pTree(sf::RectangleShape parentSquare, int depth);
  std::vector<sf::RectangleShape> shapes;
  double baseLength;
  int depth;
};
```

Listing 17: test.cpp

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

using namespace std;

void drawTree(sf::RenderWindow &window, sf::RectangleShape rect, float angle
    , int depth) {
  if (depth == 0) {
    return;
  }

  // Draw the current rectangle
  window.draw(rect);

  // Calculate the next rectangle's position and size
  sf::Vector2f size = rect.getSize();
  sf::Vector2f position = rect.getPosition();
  float newWidth = size.x * cos(angle);
  float newHeight = size.y * sin(angle);

  // Create the next rectangle
```

```
  sf::RectangleShape nextRect(sf::Vector2f(newWidth, newHeight));
  nextRect.setFillColor(sf::Color::White);
  nextRect.setOutlineColor(sf::Color::Black);
  nextRect.setOutlineThickness(1);
  nextRect.setPosition(position.x + size.x - newWidth, position.y -
   newHeight);
  nextRect.setRotation(-45);

  // Recursively draw the next branches
  drawTree(window, nextRect, angle, depth - 1);
  nextRect.setRotation(45);
  drawTree(window, nextRect, angle, depth - 1);
}

int main() {
  int depth = 5;                      // Depth of the tree
  float angle = 45 * M_PI / 180.0f;  // 45 degrees in radians

  sf::RenderWindow window(sf::VideoMode(800, 600), "Pythagorean Tree");
  window.clear(sf::Color::Black);

  // Create the trunk of the tree
  sf::RectangleShape trunk(sf::Vector2f(20, 100));
  trunk.setFillColor(sf::Color::White);
  trunk.setPosition(390, 500);
  trunk.setRotation(-90);

  drawTree(window, trunk, angle, depth);

  window.display();

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

  return 0;
}
```

4 PS3: Sokoban

4.1 Discussion

For this assignment, I implemented a program that mimic the popular Japanese Tile game, Sokoban. The goal as the player is to push all boxes into teh designated storage areas on the map. The mechanics behind this project is to have the program accept the filename of the map in the argument, then search for the text file of the map layout which is denoted in ASCII characters. The characters are mapped the image that tile should be. Data structures utilized in this project were a vector of strings to represent the map, a vector of sprites and their respective images to load into the map, a pair to respresent player cordinates, and a map match the arrowkeys to the players orientation.

An example of one of the levels can be seen in Figure: 5

Figure 5: Game map with level1.lvl file

4.2 What I accomplished

- Implemented Sokoban game with play movement and box pushing mechanics
- Created a graphical respesentation using SFML
- Implemented level loading from text files and rendering of the game map

4.3 What I already knew

- SFML basics and I/O

- Application of vectors and maps

4.4 What I learned

- A better understanding of video game concept such as game loops and rendering
- Managing game states and user inputs in application programming

4.5 Challenges

- Implementing the box pushing mechanics with respect to the obstacles

4.6 Codebase

Listing 18: Makefile

```
# Compiler variables
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
# ar command variables
AR = ar
ARFLAGS = -rcs
# Your .hpp files
DEPS = Sokoban.hpp
# Your compiled .o files
OBJECTS = Sokoban.o
# The name of your program
PROGRAM = Sokoban
TEST = test
# Static library name
STATIC_LIB = Sokoban.a

.PHONY: all clean lint

all: $(PROGRAM) $(STATIC_LIB) $(TEST)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<
# Sokoban:
$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

# static library:
$(STATIC_LIB) : $(OBJECTS)
    $(AR) $(ARFLAGS) $@ $^

#boost test:
$(TEST): test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) $(STATIC_LIB)

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

Listing 19: main.cpp

```cpp
// Copyright 2024 Chris Lambert
#include <iostream>
#include "Sokoban.hpp"

int main(int argc, char* argv[]) {
  SB::Sokoban sokoban;

  return 0;
}
```

Listing 20: Sokoban.cpp

```cpp
// Copyright 2024 Chris Lambert
#include "Sokoban.hpp"

#include <fstream>
#include <iostream>
#include <sstream>
#include <string>

using sf::Image;
using sf::Sprite;
using sf::Texture;
using std::cin;
using std::getline;
using std::pair;
using std::string;

using std::cout;
using std::endl;

SB::Sokoban::Sokoban() {
  played = false;
  currentD = Direction::Down;

  cin >> filename;
  std::ifstream file(filename);

  if (!file.is_open()) {
    std::cerr << "Failed to open file." << std::endl;
    exit(1);
  }

  string line;
  getline(file, line);
  std::istringstream iss(line);
  iss >> this->height >> this->width;

  // initialize vector size
  this->board.resize(height);

  for (int i = 0; i < height; i++) {
    std::getline(file, line);
    board[i] = line;
  }

  // up down left right
  Image image1, image2, image3, image4, image5, image6, image7, image8;
  image1.loadFromFile("player_08.png");
```

```
48    image2.loadFromFile("player_05.png");
49    image3.loadFromFile("player_20.png");
50    image4.loadFromFile("player_17.png");
51    image5.loadFromFile("block_06.png");
52    image6.loadFromFile("crate_03.png");
53    image7.loadFromFile("ground_01.png");
54    image8.loadFromFile("ground_04.png");
55
56    textures = new Texture[8];
57    text = new sf::Text[1];
58
59    // up down left right
60    textures[0].loadFromImage(image1);
61    textures[1].loadFromImage(image2);
62    textures[2].loadFromImage(image3);
63    textures[3].loadFromImage(image4);
64    // wall box empty storage
65    textures[4].loadFromImage(image5);
66    textures[5].loadFromImage(image6);
67    textures[6].loadFromImage(image7);
68    textures[7].loadFromImage(image8);
69
70    playerMap[Direction::Up] = textures[0];
71    playerMap[Direction::Down] = textures[1];
72    playerMap[Direction::Left] = textures[2];
73    playerMap[Direction::Right] = textures[3];
74
75    setSprites();
76
77    sf::Font font;
78    if (!font.loadFromFile("Noto Mono for Powerline.ttf")) {
79      std::cerr << "Could not Open Font" << endl;
80    }
81
82    // Create a text which uses our font
83    sf::Text text1;
84    text1.setFont(font);
85    text1.setCharacterSize(50);
86    text1.setStyle(sf::Text::Regular);
87    text1.setString("You Win!");
88    sf::FloatRect textRect = text1.getLocalBounds();
89    text1.setOrigin(textRect.left + textRect.width / 2.0f, textRect.top +
       textRect.height / 2.0f);
90    text1.setPosition(width * 64 / 2, height * 64 / 2);
91    this->text[0] = text1;
92
93    sf::RenderWindow window(sf::VideoMode(width * 64, height * 64), "Sokoban")
       ;
94    while (window.isOpen()) {
95      sf::Event event;
96      while (window.pollEvent(event)) {
97        if (event.type == sf::Event::Closed)
98          window.close();
99      }
100     window.clear(sf::Color::Black);
101
102     for (auto i = sprites.begin(); i != sprites.end(); ++i) {
103       window.draw(*i);
104     }
```

```
105     setSprites();
106     window.display();
107 
108     if (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {
109       restart();
110     }
111     if (isWon()) {
112       window.draw(text[0]);
113       window.display();
114       continue;
115     }
116     if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
117       movePlayer(Direction::Up);
118     }
119     if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
120       movePlayer(Direction::Left);
121     }
122     if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
123       movePlayer(Direction::Down);
124     }
125     if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
126       movePlayer(Direction::Right);
127     }
128   }
129 }
130 SB::Sokoban::Sokoban(std::string filename) : filename(filename) {
131   std::ifstream file(filename);
132 
133   if (!file.is_open()) {
134     std::cerr << "Failed to open file." << std::endl;
135     exit(1);
136   }
137 
138   string line;
139   getline(file, line);
140   std::istringstream iss(line);
141   iss >> this->width >> this->height;
142 
143   // initialize vector size
144   this->board.resize(height);
145 
146   for (int i = 0; i < height; i++) {
147     std::getline(file, line);
148     board[i] = line;
149   }
150 
151   // up down left right
152   Image image1, image2, image3, image4, image5, image6, image7, image8;
153   image1.loadFromFile("player_08.png");
154   image2.loadFromFile("player_05.png");
155   image3.loadFromFile("player_20.png");
156   image4.loadFromFile("player_17.png");
157   image5.loadFromFile("block_06.png");
158   image6.loadFromFile("crate_03.png");
159   image7.loadFromFile("ground_01.png");
160   image8.loadFromFile("ground_04.png");
161 
162   textures = new Texture[8];
163   text = new sf::Text[1];
```

```
164
165   // up down left right
166   textures[0].loadFromImage(image1);
167   textures[1].loadFromImage(image2);
168   textures[2].loadFromImage(image3);
169   textures[3].loadFromImage(image4);
170   // wall box empty storage
171   textures[4].loadFromImage(image5);
172   textures[5].loadFromImage(image6);
173   textures[6].loadFromImage(image7);
174   textures[7].loadFromImage(image8);
175
176   playerMap[Direction::Up] = textures[0];
177   playerMap[Direction::Down] = textures[1];
178   playerMap[Direction::Left] = textures[2];
179   playerMap[Direction::Right] = textures[3];
180
181   setSprites();
182
183   sf::Font font;
184   if (!font.loadFromFile("Noto Mono for Powerline.ttf")) {
185     std::cerr << "Could not Open Font" << endl;
186   }
187
188   // Create a text which uses our font
189   sf::Text text1;
190   text1.setFont(font);
191   text1.setCharacterSize(50);
192   text1.setStyle(sf::Text::Regular);
193   text1.setString("You Win!");
194   sf::FloatRect textRect = text1.getLocalBounds();
195   text1.setOrigin(textRect.left + textRect.width / 2.0f, textRect.top +
      textRect.height / 2.0f);
196   text1.setPosition(width * 64 / 2, height * 64 / 2);
197   this->text[0] = text1;
198 }
199 // destructor
200 SB::Sokoban::~Sokoban() {
201   delete[] textures;
202   delete[] text;
203 }
204 //
205 void SB::Sokoban::movePlayer(Direction d) {
206   currentD = d;
207   auto swap = [](auto& x, auto& y) {
208     auto temp = x;
209     x = y;
210     y = temp;
211   };
212   // first is down, second is across
213   auto& location = board[playerLoc().first][playerLoc().second];
214   auto& above = board[playerLoc().first - 1][playerLoc().second];
215   auto& below = board[playerLoc().first + 1][playerLoc().second];
216   auto& left = board[playerLoc().first][playerLoc().second - 1];
217   auto& right = board[playerLoc().first][playerLoc().second + 1];
218
219   auto& above2 = board[playerLoc().first - 2][playerLoc().second];
220   auto& below2 = board[playerLoc().first + 2][playerLoc().second];
221   auto& left2 = board[playerLoc().first][playerLoc().second - 2];
```

```
222   auto& right2 = board[playerLoc().first][playerLoc().second + 2];
223   // based on direction, look to blocks ahead and determine proper mvoement
224   switch (d) {
225     case Direction::Up:
226
227       if (above == '.') {
228         swap(location, above);
229         position.first -= 1;
230       } else if (above == 'A') {
231         if (above2 == 'a') {
232           above2 = '1';
233           above = '.';
234           swap(location, above);
235           position.first += 1;
236         } else if (above2 == '.') {
237           swap(above2, above);
238           swap(location, above);
239         }
240       }
241       while ((sf::Keyboard::isKeyPressed(sf::Keyboard::W))) {
242       }
243       break;
244     case Direction::Down:
245
246       if (below == '.') {
247         swap(location, below);
248         position.first += 1;
249       } else if (below == 'A') {
250         if (below2 == 'a') {
251           below2 = '1';
252           below = '.';
253           swap(location, below);
254           position.first += 1;
255         } else if (below2 == '.') {
256           swap(below2, below);
257           swap(location, below);
258         }
259       }
260       while ((sf::Keyboard::isKeyPressed(sf::Keyboard::S))) {
261       }
262       break;
263     case Direction::Left:
264
265       if (left == '.') {
266         swap(location, left);
267         position.second -= 1;
268       } else if (left == 'A') {
269         if (left2 == 'a') {
270           left2 = '1';
271           left = '.';
272           swap(location, left);
273           position.second -= 1;
274         } else if (left2 == '.') {
275           swap(left2, left);
276           swap(location, left);
277           position.second -= 1;
278         }
279       }
280
```

```
      while ((sf::Keyboard::isKeyPressed(sf::Keyboard::A))) {
      }
      break;
    case Direction::Right:

      if (right == '.') {
        swap(location, right);
        position.second += 1;
      } else if (right == 'A') {
        if (right2 == 'a') {
          right2 = '1';
          right = '.';
          swap(location, right);
          position.second += 1;
        } else if (right2 == '.') {
          swap(right2, right);
          swap(location, right);
          position.second += 1;
        }
      }
      while ((sf::Keyboard::isKeyPressed(sf::Keyboard::D))) {
      }
      break;
    default:
      break;
  }
}

void SB::Sokoban::restart() {
  played = false;
  currentD = Direction::Down;
  sprites.clear();

  std::ifstream file(filename);

  if (!file.is_open()) {
    std::cerr << "Failed to open file." << std::endl;
    exit(1);
  }

  string line;
  getline(file, line);
  std::istringstream iss(line);
  iss >> this->width >> this->height;

  // initialize vector size
  this->board.clear();
  this->board.resize(height);

  for (int i = 0; i < height; i++) {
    std::getline(file, line);
    board[i] = line;
  }
  setSprites();
}

bool SB::Sokoban::isWon() {
  // look for an 'a' on the board
  for (auto i : board) {
```

```
    for (auto j : i) {
      if (j == 'a')
        return false;
    }
  }
  return true;
}

/*
texture mappings:
0  - up
1 - down
2 - left
3 - right

4 - wall
5 - box
6 - empty (tile)
7 - storage (border tile)
*/

void SB::Sokoban::setSprites() {
  double cordinates[] = {0.0, 0.0};

  Sprite block;

  for (int i = 0; i < height; i++) {
    std::string row = board[i];
    for (int j = 0; j < width; j++) {
      char icon = row[j];
      block.setPosition(cordinates[0], cordinates[1]);

      if (icon == '#') {  // a wall
        block.setTexture(textures[4]);
      } else if (icon == '@') {  // player position, draw tile then player
        block.setTexture(textures[6]);
        this->sprites.push_back(block);
        block.setTexture(playerMap[currentD]);
        position.first = i;
        position.second = j;
      } else if (icon == '.') {  // empty space, draw a wile
        block.setTexture(textures[6]);
      } else if (icon == 'A') {  // A box
        block.setTexture(textures[5]);
      } else if (icon == 'a') {  // storage location
        block.setTexture(textures[7]);
      } else if (icon == '1') {  // box already in storage location
        block.setTexture(textures[5]);
      }

      this->sprites.push_back(block);
      cordinates[0] += 64;
    }
    cordinates[1] += 64;
    cordinates[0] = 0;
  }
}

namespace SB {
```

```
std::istream& operator>>(std::istream& input, Sokoban& sokoban) {
  input >> sokoban.filename;
  return input;
}
}  // namespace SB
```

Listing 21: Sokoban.hpp

```
// Copyright 2024 Chris Lambert
#pragma once

#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>

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

class Sokoban : public sf::Drawable {
 public:
  Sokoban();
  // for test purposes
  explicit Sokoban(std::string filename);

  int getWidth() const { return width; }
  int getHeight() const { return height; }
  std::pair<int, int> playerLoc() const { return position; }
  void movePlayer(Direction d);
  bool isWon();

  friend std::istream& operator>>(std::istream& input, Sokoban& sokoban);

  ~Sokoban();

 protected:
  virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
    {}

 private:
  std::string icons;
  std::string filename;
  int width;
  int height;
  std::vector<std::string> board;
  std::vector<sf::Sprite> sprites;
  std::pair<int, int> position;
  void setSprites();
  void restart();
  void playWin();
  sf::Texture* textures;
  sf::Text* text;
  std::map<Direction, sf::Texture> playerMap;
```

```
  Direction currentD;
  bool played;
  sf::Sound sound;
  sf::SoundBuffer soundBuffer;
};

}  // namespace SB
```

Listing 22: test.cpp

```
// Copyright 2024 Chris Lambert
#include <iostream>
#include <sstream>
#include <string>

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

#include "Sokoban.hpp"

BOOST_AUTO_TEST_CASE(sokoban_initialization_test) {
  SB::Sokoban sokoban("level1.lvl");

  // Check if width and height are initialized correctly
  BOOST_CHECK_EQUAL(sokoban.getWidth(), 10);
  BOOST_CHECK_EQUAL(sokoban.getHeight(), 10);
  // Add more checks for other initial conditions
}

BOOST_AUTO_TEST_CASE(sokoban_move_test) {
  SB::Sokoban sokoban("level1.lvl");

  // Move player up
  sokoban.movePlayer(SB::Direction::Up);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().first, 5);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().second, 3);

  // Move player left
  sokoban.movePlayer(SB::Direction::Left);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().first, 5);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().second, 2);

  // Move player down
  sokoban.movePlayer(SB::Direction::Down);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().first, 6);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().second, 2);

  // Move player right
  sokoban.movePlayer(SB::Direction::Right);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().first, 6);
  BOOST_CHECK_EQUAL(sokoban.playerLoc().second, 3);
}

BOOST_AUTO_TEST_CASE(sokoban_win_test) {
  SB::Sokoban sokoban("level1.lvl");

  // Check if the game is won
  BOOST_CHECK_EQUAL(sokoban.isWon(), false);
}
```

5 PS4: N-Body Simulation

5.1 Discussion

For this project, I implemented a celestial body simulation program that reads and processes planet data from either stdin or a specified file. The program parses the data, creating CelestialBody objects that store each planet's coordinates, velocity, and image filename, as well as a Universe object that tracks the number of planets and the scaling factor. Using a vector of unique pointers to CelestialBody objects, the program maps the planets to their respective coordinates and simulates orbital patterns by transforming each frame of a GIF. The animation continues until the specified stop and step times provided as command line arguments. At the end, the program outputs the current state of the universe in the same format as the input file.

An example of one of the levels can be seen in Figure: 6

Figure 6: Screen capture of the running N-Body program

5.2 What I accomplished

- Successfully parsed and processed planet data
- Implemented orbital simulation with CelestialBody objects
- Created a GIF animation of the orbital patterns

5.3 What I already knew

- Object-oriented programming principles in C++
- Basic physics concepts related to planetary motion

5.4 What I learned

- Advanced usage of vectors and unique pointers in C++
- Techniques for processing and transforming image frames

5.5 Challenges

- Ensuring smooth and accurate orbital simulations

5.6 Codebase

Listing 23: Makefile

```
# Compiler variables
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
# ar command variables
AR = ar
ARFLAGS = -rcs
# Your .hpp files
DEPS = Universe.hpp CelestialBody.hpp
# Your compiled .o files
OBJECTS = Universe.o CelestialBody.o
# The name of your program
PROGRAM = NBody
TEST = test
# Static library name
STATIC_LIB = NBody.a

.PHONY: all clean lint

all: $(PROGRAM) $(STATIC_LIB) $(TEST)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<
# Nbody:
$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

# Nbody static library:
$(STATIC_LIB) : $(OBJECTS)
    $(AR) $(ARFLAGS) $@ $^

#boost test:
 $(TEST): test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) $(STATIC_LIB) $(TEST)

lint:
```

```
    cpplint *.cpp *.hpp
```

Listing 24: main.cpp

```
// Copyright 2024 Chris Lambert
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>
#include "Universe.hpp"
#include "CelestialBody.hpp"

using std::cin;
using std::string;

int main(int argc, char* argv[]) {
  double elapsedTime = 0.0;
  double T = std::stod(argv[1]);
  double deltaT = std::stod(argv[2]);

  NB::Universe universe;
  cin >> universe;

  sf::RenderWindow window(sf::VideoMode(500, 500), "The Solar System!");
  // Extra credit background image
  sf::Texture backgroundTexture;
  if (!backgroundTexture.loadFromFile("background.jpg")) {
    std::cerr << "Error loading background image" << std::endl;
  }
  // scale the background up to fill the screen
  sf::Vector2u windowSize = window.getSize();
  sf::Vector2u textureSize = backgroundTexture.getSize();
  float scaleX = static_cast<float>(windowSize.x) / textureSize.x;
  float scaleY = static_cast<float>(windowSize.y) / textureSize.y;
  sf::Sprite backgroundSprite(backgroundTexture);
  backgroundSprite.setScale(scaleX, scaleY);

  // loads the universe with respect to the window size.
  universe.load(window.getSize().x);
  sf::Clock clock;
  while (window.isOpen()) {
    sf::Event event;
    while (window.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
        window.close();
    }

    if (elapsedTime < T) {
      universe.step(deltaT);
      elapsedTime += deltaT;
    } else {
      window.close();
    }
    window.setTitle(" The Solar System! - Elapsed Time: " +
                    std::to_string(clock.getElapsedTime().asSeconds()) + "
   seconds");

    window.clear(sf::Color::Black);
    window.draw(backgroundSprite);
    window.draw(universe);
```

```
    window.display();
  }
  std::cout << universe << std::endl;
}
```

Listing 25: CelestialBody.cpp

```
// Copyright 2024 Chris Lambert
#include "CelestialBody.hpp"
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>

NB::CelestialBody::CelestialBody() = default;

void NB::CelestialBody::loadImage() {
  if (!this->image.loadFromFile(this->planet)) {
    std::cerr << " Error loading an image " << std::endl;
  }
  if (!this->texture.loadFromImage(image)) {
    std::cerr << " Error loading a texture " << std::endl;
  }
  this->sprite.setTexture(texture);

  sf::FloatRect rect = sprite.getLocalBounds();
  sprite.setOrigin(rect.left + rect.width / 2.0f, rect.top + rect.height /
   2.0f);
}

// iostream overloads
std::istream& NB::operator>>(std::istream& input, CelestialBody& body) {
  input >> body.xpos >> body.ypos >> body.xvel >> body.yvel >> body.mass >>
   body.planet;
  return input;
}
std::ostream& NB::operator<<(std::ostream& out, const CelestialBody& body) {
  out << std::scientific << body.xpos << " " << body.ypos << " "
      << body.xvel << " " << body.yvel << " " << body.mass << " " << body.
   planet;
  return out;
}
```

Listing 26: CelestialBody.hpp

```
// Copyright 2024 Chris Lambert
#pragma once
#include <iostream>
#include <string>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>

namespace NB {

class CelestialBody : public sf::Drawable {
 public:
  friend std::istream& operator>>(std::istream& input, CelestialBody& body);
```

```
15    friend std::ostream& operator<<(std::ostream& out, const CelestialBody&
       body);
16
17    CelestialBody();
18    ~CelestialBody() {}
19
20    void setLocation(double scale, float windowSize) {
21      this->sprite.setPosition(xpos * scale + (windowSize / 2), ypos * scale +
       (windowSize / 2));
22    }
23    sf::Vector2f position() { return sf::Vector2f(xpos, ypos); }
24    sf::Vector2f velocity() { return sf::Vector2f(xvel, yvel); }
25    void setVelocity(double x, double y) {
26      this->xvel = x;
27      this->yvel = y;
28    }
29    void setPosition(double x, double y) {
30      this->xpos = x;
31      this->ypos = y;
32    }
33    double getMass() { return mass; }
34    void loadImage();
35
36   protected:
37    void draw(sf::RenderTarget& target, sf::RenderStates states) const
       override {
38      target.draw(sprite, states);
39    }
40
41   private:
42    double xpos;
43    double ypos;
44    double xvel;
45    double yvel;
46    double mass;
47    std::string planet;
48    sf::Texture texture;
49    sf::Sprite sprite;
50    sf::Image image;
51  };
52  std::istream& operator>>(std::istream& input, CelestialBody& body);
53  std::ostream& operator<<(std::ostream& os, const CelestialBody& body);
54  }  // namespace NB
```

Listing 27: Universe.cpp

```
1   // Copyright 2024 Chris Lambert
2   #include <string>
3   #include <fstream>
4   #include <iostream>
5   #include <sstream>
6   #include <cmath>
7   #include "Universe.hpp"
8   #include "CelestialBody.hpp"
9   #include <SFML/Graphics.hpp>
10  #include <SFML/System.hpp>
11  #include <SFML/Window.hpp>
12  #include <SFML/Audio.hpp>
13
14  using std::cout;
```

```
15 using std::endl;
16 using std::string;
17 NB::Universe::Universe(string filename) {
18   std::ifstream file(filename);
19   if (!file.is_open()) {
20     std::cerr << "Error opening file." << std::endl;
21   }
22 
23   file >> *this;
24 
25   file.close();
26 }
27 void NB::Universe::load(float windowSize) {
28   double scale = (windowSize / 2) / this->rad;
29   this->winScale = scale;
30   this->winSize = windowSize;
31   for (auto& body : bodies) {
32     // load image/texture/sprite and set location
33     body->loadImage();
34     body->setLocation(scale, windowSize);
35   }
36 }
37 void NB::Universe::step(double seconds) {
38   const double g = 6.67e-11;
39 
40   CelestialBody* sun = nullptr;
41   // look for the sun
42   for (auto& body : bodies) {
43     if (body->position() == sf::Vector2f(0.0, 0.0)) {
44       sun = body.get();
45     }
46   }
47 
48   for (auto& body : bodies) {
49     // ignore the sun
50     if (body.get() == sun)
51       continue;
52     // get net force
53     double netF, netX, netY, dX, dY, r;
54 
55     dX = sun->position().x - body->position().x;
56     dY = sun->position().y - body->position().y;
57 
58     r = sqrt(pow(dX, 2) + pow(dY, 2));
59 
60     netF = g * body->getMass() * sun->getMass() / pow(r, 2);
61 
62     netX = netF * dX / r;
63     netY = netF * dY / r;
64     // get acceleration
65     double aX, aY;
66 
67     aX = netX / body->getMass();
68     aY = netY / body->getMass();
69 
70     double newVelx = body->velocity().x + seconds * aX;
71     double newVely = body->velocity().y + seconds * aY;
72 
73     body->setVelocity(newVelx, newVely);
```

```
74
75      double newPosx = body->position().x + seconds * newVelx;
76      double newPosy = body->position().y + seconds * newVely;
77
78      body->setPosition(newPosx, newPosy);
79      body->setLocation(winScale, winSize);
80    }
81  }
82  // iostream overloads
83  std::istream& NB::operator>>(std::istream& input, Universe& universe) {
84    int newNumBodies;
85    double newRadius;
86
87    input >> newNumBodies >> newRadius;
88
89    universe.numBodies = newNumBodies;
90    universe.rad = newRadius;
91    universe.bodies.clear();
92
93    for (int i = 0; i < universe.numBodies; ++i) {
94      std::unique_ptr<CelestialBody> body = std::make_unique<CelestialBody>();
95      input >> *body;
96      body->loadImage();
97      universe.bodies.push_back(std::move(body));
98    }
99    return input;
100 }
101 std::ostream& NB::operator<<(std::ostream& out, const Universe& universe) {
102   out << universe.numBodies << endl;
103   out << universe.rad << endl;
104   for (const auto& body : universe.bodies) {
105     out << *body << endl;
106   }
107   return out;
108 }
```

Listing 28: Universe.hpp

```
1  // Copyright 2024 Chris Lambert
2  #pragma once
3  #include <string>
4  #include <vector>
5  #include <memory>
6  #include "CelestialBody.hpp"
7
8  namespace NB {
9  class Universe : public CelestialBody {
10  public:
11    friend std::istream& operator>>(std::istream& input, Universe& universe);
12    friend std::ostream& operator<<(std::ostream& out, const Universe&
       universe);
13    Universe() = default;
14    explicit Universe(std::string);
15    void load(float);
16    int numPlanets() { return numBodies; }
17    double radius() { return rad; }
18    void step(double seconds);
19
20    ~Universe() {}
21    CelestialBody& operator[](size_t index) {
```

```
    if (index >= bodies.size()) {
      throw std::out_of_range("Index out of range");
    }
    return *bodies[index];
  }

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

 private:
  int numBodies;
  double rad;
  double winScale;
  double winSize;
  std::vector<std::unique_ptr<CelestialBody>> bodies;
};
std::istream& operator>>(std::istream& input, Universe& universe);
std::ostream& operator<<(std::ostream& os, const Universe& universe);
}  // namespace NB
```

Listing 29: test.cpp

```
// Copyright 2024 Chris Lambert
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <fstream>
#include <sstream>
#include <boost/test/unit_test.hpp>
#include "CelestialBody.hpp"
#include "Universe.hpp"
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>

BOOST_AUTO_TEST_CASE(celestial_body_IO_test) {
  std::string line("1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e
   +24    earth.gif");
  NB::CelestialBody body;
  std::istringstream(line) >> body;

  std::ostringstream oss;
  oss << body;

  BOOST_CHECK_EQUAL(oss.str(),
                    "1.496000e+11 0.000000e+00 0.000000e+00 2.980000e+04
   5.974000e+24 earth.gif");
}

BOOST_AUTO_TEST_CASE(universe_IO_test) {
  NB::Universe universe("planets.txt");
  std::ostringstream oss;
  oss << universe;
  BOOST_CHECK(oss);
}
```

```
32
33 BOOST_AUTO_TEST_CASE(universe_getter_test) {
34   NB::Universe universe("planets.txt");
35
36   // Check if the universe is loaded with the correct number of bodies and
     radius
37   BOOST_CHECK_EQUAL(universe.numPlanets(), 5);
38   BOOST_CHECK_EQUAL(universe.radius(), 2.50e+11);
39 }
40 BOOST_AUTO_TEST_CASE(celestial_body_getter_test) {
41   std::string line("1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e
     +24    earth.gif");
42   NB::CelestialBody body;
43   std::istringstream(line) >> body;
44
45   // Check if the universe is loaded with the correct number of bodies and
     radius
46   BOOST_CHECK_EQUAL(body.velocity().x, 0);
47   double a = 1.49599994e+11;
48   BOOST_CHECK_CLOSE(body.position().x, a, 0.0001);
49 }
50 BOOST_AUTO_TEST_CASE(universe_step__test) {
51   NB::Universe universe("planets.txt");
52   double x = 1.4960e+11;
53   double y = 1.19750003e+09;
54
55   universe.step(25000);
56
57   BOOST_CHECK_CLOSE(universe[0].position().x, x, 0.1);
58   BOOST_CHECK_CLOSE(universe[2].position().y, y, 0.1);
59 }
```

6 PS5: DNA Alignment

6.1 Discussion

For the PS5 assignment, I implemented a program that calculates the edit distance between two strings using the Levenshtein algorithm. The program reads two strings from stdin, constructs a 2D matrix to represent the edit distance computation, and fills in the matrix recursively. The edit distance is then printed along with the alignment sequence, showing the gaps and costs per index. The Levenshtein algorithm was implemented using recursion to compute the edit distance, and a vector of vectors was used to mimic matrix operations for storing the strings and the matrix.

An example :output of one of the files are seen below:

```
./EDistance < example10.txt
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

Elapsed time: 3e-05
```

6.2 What I accomplished

- Successfully implemented the Levenshtein algorithm for calculating edit distance
- Printed the alignment sequence with gaps and costs
- Efficiently used recursion and dynamic programming to fill the matrix

6.3 What I already knew

- Recursion and dynamic programming

6.4 What I learned

- Improved understanding of dynamic programming concepts
- Practiced implementing algorithms with complex matrix operations

6.5 Challenges

- Ensuring the correct edit distance out of the many possible cases

6.6 Codebase

Listing 30: Makefile

```
1 # Compiler variables
2 CC = g++
3 CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
4 LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
      lboost_unit_test_framework -O3
5 # ar command variables
6 AR = ar
```

```
ARFLAGS = -rcs
# Your .hpp files
DEPS = EDistance.hpp
# Your compiled .o files
OBJECTS = EDistance.o
# The name of your program
PROGRAM = EDistance
TEST = test
# Static library name
STATIC_LIB = EDistance.a

.PHONY: all clean lint

all: $(PROGRAM) $(STATIC_LIB) $(TEST)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<
# EDistance:
$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

# EDistance.a static library:
$(STATIC_LIB) : $(OBJECTS)
    $(AR) $(ARFLAGS) $@ $^

#boost test:
 $(TEST): test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) $(STATIC_LIB) $(TEST)

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

Listing 31: main.cpp

```
// Copyright 2024 Chris Lambert
#include <iostream>
#include <string>
#include "EDistance.hpp"
#include <SFML/System.hpp>

using std::cin;
using std::cout;
using std::endl;

int main() {
  sf::Clock clock;
  EDistance test;
  cin >> test;

  sf::Time t = clock.getElapsedTime();

  cout << test.alignment() << endl;

  cout << "Elapsed time: " << t.asSeconds() << endl;

  return 0;
```

```
}
```

Listing 32: CelestialBody.cpp

```
// Copyright 2024 Chris Lambert
#include "EDistance.hpp"

EDistance::EDistance(string a, string b) {
  if (b.length() > a.length()) {
    std::swap(a, b);
  }
  matrix.resize(a.size() + 2, std::vector<string>(b.size() + 2, "-"));

  for (size_t i = 1; i <= a.size(); ++i) {
    matrix[i][0] = a[i - 1];
  }
  for (size_t j = 1; j <= b.size(); ++j) {
    matrix[0][j] = b[j - 1];
  }
}
int EDistance::optDistance() {
  // 12 and 10 -> 10 , 8
  int count = 0;

  for (size_t i = matrix[0].size() - 1; i > 0; --i) {
    matrix[matrix.size() - 1][i] = std::to_string(count);
    count += 2;
  }
  count = 0;

  for (size_t i = matrix.size() - 1; i > 0; --i) {
    matrix[i][matrix[0].size() - 1] = std::to_string(count);
    count += 2;
  }

  for (size_t j = matrix.size() - 2; j > 0; --j) {
    for (size_t i = matrix[0].size() - 2; i > 0; --i) {
      auto& entry = matrix[j][i];

      auto a = matrix[j][0];
      auto b = matrix[0][i];

      auto x = std::stoi(matrix[j][i + 1]) + 2;
      auto y = std::stoi(matrix[j + 1][i]) + 2;
      auto z = std::stoi(matrix[j + 1][i + 1]) + 1;

      if (penalty(a[0], b[0])) {
        entry = std::to_string(min3(z, y, x));
      } else {
        entry = std::to_string(std::stoi(matrix[j + 1][i + 1]));
      }
    }
  }

  return stoi(matrix[1][1]);
}

int EDistance::calcDistance(size_t top, size_t left) {
  auto& entry = matrix[top][left];

```

```
  auto a = matrix[top][0];
  auto b = matrix[0][left];

  int x = calcDistance(top, left + 1) + 2;
  int y = calcDistance(top + 1, left) + 2;
  int z = calcDistance(top + 1, left + 1) + 1;

  if (penalty(a[0], b[0])) {
    entry = std::to_string(min3(z, y, x));
  } else {
    entry = matrix[top + 1][left + 1];
  }

  return stoi(entry);
}

string EDistance::alignment() {
  string alignment;

  alignment += "Edit Distance = " + std::to_string(optDistance()) + "\n";
  size_t i = 1, j = 1;
  while (i < matrix.size() - 1 && j < matrix[0].size() - 1) {
    auto a = matrix[i][0];
    auto b = matrix[0][j];
    int opt_current = std::stoi(matrix[i][j]);
    int opt_diagonal = std::stoi(matrix[i + 1][j + 1]);
    int opt_down = std::stoi(matrix[i + 1][j]);
    int opt_right = std::stoi(matrix[i][j + 1]);

    if (opt_current == opt_diagonal + (penalty(a[0], b[0]) ? 1 : 0)) {
      alignment += a + " " + b + " " + std::to_string(penalty(a[0], b[0])) +
    "\n";
      ++i;
      ++j;
    } else if (opt_current == opt_down + 2) {
      alignment += a + "   2\n";
      ++i;
    } else if (opt_current == opt_right + 2) {
      alignment += "   " + b + " 2\n";
      ++j;
    }
  }
  return alignment;
}
std::istream& operator>>(std::istream& input, EDistance& ed) {
  string a, b;
  getline(input, a);
  getline(input, b);
  ed = EDistance(a, b);

  return input;
}
```

Listing 33: CelestialBody.hpp

```
// Copyright 2024 Chris Lambert
#pragma once
#include <string>
#include <vector>
#include <iostream>
```

```
using std::string;
using std::vector;

class EDistance {
 public:
  friend std::istream& operator>>(std::istream& input, EDistance& ed);

  auto& operator[](size_t index) {
    if (index >= 0 && index <= matrix.size())
      return matrix[index];
    else
      throw std::out_of_range("Out of Range Index");
  }

  EDistance() = default;
  EDistance(string, string);

  static int penalty(char a, char b) { return (a == b) ? 0 : 1; }
  static int min3(int a, int b, int c) { return std::min(std::min(a, b), c);
    }
  int optDistance();
  string alignment();

 private:
  vector<vector<string>> matrix;

  int calcDistance(size_t top, size_t left);
};
```

Listing 34: test.cpp

```
// Copyright 2024 Chris Lambert
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <fstream>
#include <sstream>
#include <boost/test/unit_test.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>
#include "EDistance.hpp"

BOOST_AUTO_TEST_CASE(testConstructor) {
  string a = "peter";
  string b = "pan";

  EDistance ed(a, b);
  BOOST_CHECK_EQUAL(ed[0].size(), 5);
  BOOST_CHECK_EQUAL(ed[1][0], "p");
  BOOST_CHECK_EQUAL(ed[0][1], "p");
}

BOOST_AUTO_TEST_CASE(testOptDistance) {
  EDistance ed("peterpan", "neverland");
  BOOST_CHECK_EQUAL(ed.optDistance(), 5);
}

BOOST_AUTO_TEST_CASE(testStreamOperator) {
  std::stringstream ss("AGGTAGCAGAAC\nCGGTCAGTCA\n");
```

```
30   EDistance ed;
31   ss >> ed;
32   BOOST_CHECK_EQUAL(ed[1][0], "A");
33   BOOST_CHECK_EQUAL(ed[0][1], "C");
34 }
35 BOOST_AUTO_TEST_CASE(testIndex) {
36   std::stringstream ss("AGGTAGCAGAAC\nCGGTCAGTCA\n");
37   EDistance ed;
38   ss >> ed;
39   BOOST_REQUIRE_THROW(ed[15], std::out_of_range);
40 }
```

7 PS6: RandWriter

RandWriter, is a text generation program based on Markov chains. It is inspired by Claude Shannon's work in information theory and serves as a practical application of probabilistic modeling in text generation. The program takes two command line arguments, the order k and the length of the sentence to be generated L. It reads input data from stdin and uses this data to build a model of text. The model is constructed by analyzing the frequency of k-grams (sequences of k characters) in the input text. This information is stored in two map data structures, one mapping the frequency of k-grams and the other mapping the frequency of the next character following each k-gram.

To generate text, the program uses the constructed model to probabilistically select the next character based on the current k-gram. This process continues iteratively until the desired sentence length is reached. The program utilizes a lambda expression to efficiently handle circular substrings of the generated text, ensuring that the text generation process is seamless.

7.1 Discussion

An example :output of this model can be seen in below, the data for the model is a book by Tom Sawyer:

```
k = 2; L = 85
./TextWriter 2 85 < tomsawyer.txt
THE And throt?" then to-wis mand a toccout, The grounhaverging ever leake vis of wed.
```

7.2 What I accomplished

- Implemented a text generation program using Markov chains
- Demonstrated the application of probabilistic modeling in text generation
- Used maps for storing frequency data

7.3 What I already knew

- C++ I/O basics
- Using nested map data structures

7.4 What I learned

- Practical application of Markov chains in text generation
- Efficient handling of circular substrings in text generation

7.5 Challenges

- Calculating the probability based on frequency accurately

7.6 Codebase

Listing 35: Makefile

```
# Compiler variables
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lboost_unit_test_framework -O3
# ar command variables
AR = ar
ARFLAGS = -rcs
# Your .hpp files
```

```
DEPS = RandWriter.hpp
# Your compiled .o files
OBJECTS = RandWriter.o
# The name of your program
PROGRAM = TextWriter
TEST = test
# Static library name
STATIC_LIB = TextWriter.a

.PHONY: all clean lint

all: $(PROGRAM) $(STATIC_LIB) $(TEST)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<
# TextWriter:
$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

# TextWriter.a static library:
$(STATIC_LIB) : $(OBJECTS)
    $(AR) $(ARFLAGS) $@ $^

#boost test:
 $(TEST): test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) $(STATIC_LIB) $(TEST)

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

Listing 36: main.cpp

```
// Copyright 2024 Chris Lambert
#include <iostream>
#include <string>
#include "RandWriter.hpp"

using std::stoi;

int main(int argc, char* argv[]) {
  string text;

  string line;

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

  int order = stoi(argv[1]);
  int length = stoi(argv[2]);

  RandWriter rw(text, order);

  std::string initial_kgram = text.substr(0, order);
  std::string genText = rw.generate(initial_kgram, length);

```

```
  std::cout << genText << std::endl;

  // std::cout << rw << std::endl;

  return 0;
}
```

Listing 37: CelestialBody.cpp

```
// Copyright 2024 Chris Lambert
#include "RandWriter.hpp"
#include <algorithm>
#include <vector>
#include <random>
using std::cout;
using std::endl;

RandWriter::RandWriter(const string& text, size_t k) : _k(k), text(text) {
  for (size_t i = 0; i <= text.length() - k; ++i) {
    string kgram = text.substr(i, _k);
    char c = text[i + _k];
    kgramCount[kgram]++;
    charCount[kgram][c]++;
  }
}

int RandWriter::freq(const string& kgram) const {
  if (kgram.length() != _k)
    throw std::invalid_argument("Invalid kgram length");

  auto entry = kgramCount.find(kgram);

  if (entry != kgramCount.end())
    return entry->second;
  else
    return 0;
}

int RandWriter::freq(const string& kgram, char c) const {
  if (kgram.length() != _k)
    throw std::invalid_argument("Invalid kgram length");

  if (_k == 0) {
    return std::count(text.begin(), text.end(), c);
  }

  auto entry = charCount.find(kgram);

  if (entry != charCount.end()) {
    auto charEntry = entry->second.find(c);
    if (charEntry != entry->second.end())
      return charEntry->second;
  }
  return 0;
}

char RandWriter::kRand(const string& kgram) {
  if (kgram.length() != _k) {
    throw std::invalid_argument("Invalid kgram");
  }
```

```
52
53    // Calculate the total frequency of characters following the given kgram
54    int totalFreq = freq(kgram);
55
56    if (totalFreq == 0) {
57      throw std::invalid_argument("No characters following kgram");
58    }
59
60    std::random_device rd;
61    std::mt19937 gen(rd());
62    std::uniform_int_distribution<int> dis(1, totalFreq);
63
64    int count = dis(gen);  // Randomly selected count
65
66    int charFreq = 0;
67
68    for (const auto& submap : charCount[kgram]) {
69      charFreq += submap.second;
70      if (charFreq >= count) {
71        return submap.first;
72      }
73    }
74
75    throw std::runtime_error("Failed to generate character");
76  }
77
78  string RandWriter::generate(const string& kgram, size_t L) {
79    if (kgram.length() != _k) {
80      throw std::invalid_argument("kgram is not of length k");
81    }
82    string generatedText = kgram;
83    for (size_t i = _k; i < L; ++i) {
84      try {
85        char nextChar = kRand(generatedText.substr(generatedText.length() - _k
      ));
86        generatedText.push_back(nextChar);
87      } catch (std::invalid_argument& e) {
88        auto circularString = [](string text, size_t order) {
89          return text.substr(text.length() - order) + text.substr(0, order);
90        };
91        string subCircString = circularString(generatedText, _k);
92        char nextChar = kRand(subCircString.substr(subCircString.length() - _k
      ));
93        generatedText.push_back(nextChar);
94      }
95    }
96
97    return generatedText;
98  }
99
100 std::istream& operator>>(std::istream& is, RandWriter& rw) {
101   std::string inputText;
102   is >> inputText;
103   rw.text = inputText;
104   return is;
105 }
106
107 std::ostream& operator<<(std::ostream& os, const RandWriter& rw) {
108   os << " Order: " << rw.orderK() << "\n"
```

```
109       << "Alphabet: ";
110   for (char c : rw.alphabet) {
111     os << c << " ";
112   }
113   os << "\n";
114 
115   os << "K-gram Frequencies:\n";
116   for (const auto& entry : rw.kgramCount) {
117     os << entry.first << ": " << entry.second << "\n";
118   }
119 
120   os << "K+1-gram Frequencies:\n";
121   for (const auto& entry : rw.charCount) {
122     for (const auto& subentry : entry.second) {
123       os << entry.first << subentry.first << ": " << subentry.second << "\n"
      ;
124     }
125   }
126 
127   return os;
128 }
```

Listing 38: CelestialBody.hpp

```
1  // Copyright 2024 Chris Lambert
2  #pragma once
3  #include <iostream>
4  #include <string>
5  #include <map>
6 
7  using std::map;
8  using std::string;
9 
10 class RandWriter {
11  public:
12   friend std::istream& operator>>(std::istream& is, RandWriter& rw);
13   friend std::ostream& operator<<(std::ostream& os, const RandWriter& rw);
14   // Create a Markov model of order k from given text
15   // Assume that text has length at least k.
16   RandWriter(const string& text, size_t k);
17   size_t orderK() const { return this->_k; }  // Order k of Markov model
18 
19   // Number of occurences of kgram in text
20   // Throw an exception if kgram is not length k
21   int freq(const string& kgram) const;
22   // Number of times that character c follows kgram
23   // if order=0, return num of times that char c appears
24   // (throw an exception if kgram is not of length k)
25   int freq(const string& kgram, char c) const;
26   // Random character following given kgram
27   // (throw an exception if kgram is not of length k)
28   // (throw an exception if no such kgram)
29   char kRand(const string& kgram);
30   // Generate a string of length L characters by simulating a trajectory
31   // through the corresponding Markov chain. The first k characters of
32   // the newly generated string should be the argument kgram.
33   // Throw an exception if kgram is not of length k.
34   // Assume that L is at least k
35   string generate(const string& kgram, size_t L);
36 
```

```
 private:
  size_t _k;
  string text;
  const string alphabet = "
   ABCDEFGHIJHLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz ";
  map<string, int> kgramCount;
  map<string, map<char, int>> charCount;
};
// Overload the stream insertion operator << and display the internal state
// of the Markov model. Print out the order, alphabet, and the frequencies
// of the k-grams and k+1-grams
```

Listing 39: test.cpp

```
// Copyright 2024 Chris Lambert
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <fstream>
#include <sstream>
#include <boost/test/unit_test.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>
#include "RandWriter.hpp"

BOOST_AUTO_TEST_CASE(testOrder) {
  RandWriter rw("hello world", 2);
  BOOST_CHECK_EQUAL(rw.orderK(), 2);
}
BOOST_AUTO_TEST_CASE(testOrderZero) {
  RandWriter rw("hello world hello world", 0);
  BOOST_CHECK_EQUAL(rw.orderK(), 0);
}

BOOST_AUTO_TEST_CASE(testFreq) {
  RandWriter rw("hello world", 1);
  BOOST_CHECK_EQUAL(rw.freq("h"), 1);
  BOOST_CHECK_EQUAL(rw.freq("l"), 3);
  BOOST_CHECK_EQUAL(rw.freq("u"), 0);

  BOOST_REQUIRE_THROW(rw.freq("fdgf"), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(testFreqChar) {
  RandWriter rw("hello world", 2);
  BOOST_CHECK_EQUAL(rw.freq("he", 'l'), 1);
  BOOST_CHECK_EQUAL(rw.freq("wo", 'l'), 0);

  BOOST_REQUIRE_THROW(rw.freq("hel", 'a'), std::invalid_argument);
}

BOOST_AUTO_TEST_CASE(testKRand) {
  RandWriter rw("hello world", 2);
  BOOST_CHECK(rw.kRand("he") == 'l');
  BOOST_REQUIRE_THROW(rw.kRand("xyz"), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(testKRandZero) {
  RandWriter rw("hello world", 0);
  BOOST_REQUIRE_THROW(rw.kRand("he"), std::invalid_argument);
```

```
47   BOOST_REQUIRE_THROW(rw.kRand("xyz"), std::invalid_argument);
48 }
49
50 BOOST_AUTO_TEST_CASE(testGenerate) {
51   RandWriter rw("hello world", 2);
52   BOOST_CHECK_EQUAL(rw.generate("he", 16).length(), 16);
53   BOOST_CHECK_EQUAL(rw.generate("he", 16).substr(0, 2), "he");
54
55   BOOST_REQUIRE_THROW(rw.generate("xyz", 5), std::invalid_argument);
56 }
```

8 PS7: Kronos Log Parsing

PS7: Kronos Log Parsing parses Kronos InTouch time clock log files to analyze device startup times. It utilizes regex expressions to extract relevant information such as startup messages and timestamps. The program identifies each device startup, determines its success status, and calculates the elapsed time if applicable. It saves the output to a report file.

The program's key features include the use of regular expressions to extract data and the implementation of algorithms to analyze and process log files efficiently. Additionally, the program employs data structures to store and manipulate the extracted information, ensuring accurate and structured output.

8.1 Discussion

An example :output of this program can be seen below:

```
Device Boot Report
435369 2014-03-25 19:11:59 success
435759 2014-03-25 19:15:02 stopped
Time elapsed: 183000ms
436500 2014-03-25 19:29:59 success
436859 2014-03-25 19:32:44 stopped
Time elapsed: 165000ms
440719 2014-03-25 22:01:46 success
440791 2014-03-25 22:04:27 stopped
Time elapsed: 161000ms
440866 2014-03-26 12:47:42 success
441216 2014-03-26 12:50:29 stopped
Time elapsed: 167000ms
442094 2014-03-26 20:41:34 success
442432 2014-03-26 20:44:13 stopped
Time elapsed: 159000ms
443073 2014-03-27 14:09:01 success
443411 2014-03-27 14:11:42 stopped
Time elapsed: 161000ms
```

8.2 What I accomplished

- Successfully parsed Kronos InTouch time clock log files
- Identified device startup times and success statuses
- Calculated elapsed time for each device startup

8.3 What I already knew

- C++ file I/O

8.4 What I learned

- How to use regex expressions for data extraction
- Scripting

8.5 Challenges

- Making sure I had the correct regex patterns

8.6 Codebase

Listing 40: Makefile

```
# Compiler variables
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
    lboost_unit_test_framework
# ar command variables
AR = ar
ARFLAGS = -rcs
# Your .hpp files
DEPS = Universe.hpp CelestialBody.hpp
# Your compiled .o files
OBJECTS = Universe.o CelestialBody.o
# The name of your program
PROGRAM = NBody
TEST = test
# Static library name
STATIC_LIB = NBody.a

.PHONY: all clean lint

all: $(PROGRAM) $(STATIC_LIB) $(TEST)

# Wildcard recipe to make .o files from corresponding .cpp file
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<
# Nbody:
$(PROGRAM): main.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

# Nbody static library:
$(STATIC_LIB) : $(OBJECTS)
    $(AR) $(ARFLAGS) $@ $^

#boost test:
 $(TEST): test.o $(OBJECTS)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o $(PROGRAM) $(STATIC_LIB) $(TEST)

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

Listing 41: main.cpp

```
// Copyright 2024 Chris Lambert
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>
#include "Universe.hpp"
#include "CelestialBody.hpp"

using std::cin;
using std::string;

int main(int argc, char* argv[]) {
```

```
  double elapsedTime = 0.0;
  double T = std::stod(argv[1]);
  double deltaT = std::stod(argv[2]);

  NB::Universe universe;
  cin >> universe;

  sf::RenderWindow window(sf::VideoMode(500, 500), "The Solar System!");
  // Extra credit background image
  sf::Texture backgroundTexture;
  if (!backgroundTexture.loadFromFile("background.jpg")) {
    std::cerr << "Error loading background image" << std::endl;
  }
  // scale the background up to fill the screen
  sf::Vector2u windowSize = window.getSize();
  sf::Vector2u textureSize = backgroundTexture.getSize();
  float scaleX = static_cast<float>(windowSize.x) / textureSize.x;
  float scaleY = static_cast<float>(windowSize.y) / textureSize.y;
  sf::Sprite backgroundSprite(backgroundTexture);
  backgroundSprite.setScale(scaleX, scaleY);

  // loads the universe with respect to the window size.
  universe.load(window.getSize().x);
  sf::Clock clock;
  while (window.isOpen()) {
    sf::Event event;
    while (window.pollEvent(event)) {
      if (event.type == sf::Event::Closed)
        window.close();
    }

    if (elapsedTime < T) {
      universe.step(deltaT);
      elapsedTime += deltaT;
    } else {
      window.close();
    }
    window.setTitle(" The Solar System! - Elapsed Time: " +
                    std::to_string(clock.getElapsedTime().asSeconds()) + "
   seconds");

    window.clear(sf::Color::Black);
    window.draw(backgroundSprite);
    window.draw(universe);
    window.display();
  }
  std::cout << universe << std::endl;
}
```

Listing 42: CelestialBody.cpp

```
// Copyright 2024 Chris Lambert
#include "CelestialBody.hpp"
#include <iostream>
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>

NB::CelestialBody::CelestialBody() = default;
```

```
10
11 void NB::CelestialBody::loadImage() {
12   if (!this->image.loadFromFile(this->planet)) {
13     std::cerr << " Error loading an image " << std::endl;
14   }
15   if (!this->texture.loadFromImage(image)) {
16     std::cerr << " Error loading a texture " << std::endl;
17   }
18   this->sprite.setTexture(texture);
19
20   sf::FloatRect rect = sprite.getLocalBounds();
21   sprite.setOrigin(rect.left + rect.width / 2.0f, rect.top + rect.height /
      2.0f);
22 }
23
24 // iostream overloads
25 std::istream& NB::operator>>(std::istream& input, CelestialBody& body) {
26   input >> body.xpos >> body.ypos >> body.xvel >> body.yvel >> body.mass >>
      body.planet;
27   return input;
28 }
29 std::ostream& NB::operator<<(std::ostream& out, const CelestialBody& body) {
30   out << std::scientific << body.xpos << " " << body.ypos << " "
31       << body.xvel << " " << body.yvel << " " << body.mass << " " << body.
      planet;
32   return out;
33 }
```

Listing 43: CelestialBody.hpp

```
1  // Copyright 2024 Chris Lambert
2  #pragma once
3  #include <iostream>
4  #include <string>
5  #include <SFML/Graphics.hpp>
6  #include <SFML/System.hpp>
7  #include <SFML/Window.hpp>
8  #include <SFML/Audio.hpp>
9
10 namespace NB {
11
12 class CelestialBody : public sf::Drawable {
13  public:
14   friend std::istream& operator>>(std::istream& input, CelestialBody& body);
15   friend std::ostream& operator<<(std::ostream& out, const CelestialBody&
      body);
16
17   CelestialBody();
18   ~CelestialBody() {}
19
20   void setLocation(double scale, float windowSize) {
21     this->sprite.setPosition(xpos * scale + (windowSize / 2), ypos * scale +
      (windowSize / 2));
22   }
23   sf::Vector2f position() { return sf::Vector2f(xpos, ypos); }
24   sf::Vector2f velocity() { return sf::Vector2f(xvel, yvel); }
25   void setVelocity(double x, double y) {
26     this->xvel = x;
27     this->yvel = y;
28   }
```

```
  void setPosition(double x, double y) {
    this->xpos = x;
    this->ypos = y;
  }
  double getMass() { return mass; }
  void loadImage();

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

 private:
  double xpos;
  double ypos;
  double xvel;
  double yvel;
  double mass;
  std::string planet;
  sf::Texture texture;
  sf::Sprite sprite;
  sf::Image image;
};
std::istream& operator>>(std::istream& input, CelestialBody& body);
std::ostream& operator<<(std::ostream& os, const CelestialBody& body);
}  // namespace NB
```

Listing 44: Universe.cpp

```
// Copyright 2024 Chris Lambert
#include <string>
#include <fstream>
#include <iostream>
#include <sstream>
#include <cmath>
#include "Universe.hpp"
#include "CelestialBody.hpp"
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>

using std::cout;
using std::endl;
using std::string;
NB::Universe::Universe(string filename) {
  std::ifstream file(filename);
  if (!file.is_open()) {
    std::cerr << "Error opening file." << std::endl;
  }

  file >> *this;

  file.close();
}
void NB::Universe::load(float windowSize) {
  double scale = (windowSize / 2) / this->rad;
  this->winScale = scale;
  this->winSize = windowSize;
```

```
31    for (auto& body : bodies) {
32      // load image/texture/sprite and set location
33      body->loadImage();
34      body->setLocation(scale, windowSize);
35    }
36  }
37  void NB::Universe::step(double seconds) {
38    const double g = 6.67e-11;
39
40    CelestialBody* sun = nullptr;
41    // look for the sun
42    for (auto& body : bodies) {
43      if (body->position() == sf::Vector2f(0.0, 0.0)) {
44        sun = body.get();
45      }
46    }
47
48    for (auto& body : bodies) {
49      // ignore the sun
50      if (body.get() == sun)
51        continue;
52      // get net force
53      double netF, netX, netY, dX, dY, r;
54
55      dX = sun->position().x - body->position().x;
56      dY = sun->position().y - body->position().y;
57
58      r = sqrt(pow(dX, 2) + pow(dY, 2));
59
60      netF = g * body->getMass() * sun->getMass() / pow(r, 2);
61
62      netX = netF * dX / r;
63      netY = netF * dY / r;
64      // get acceleration
65      double aX, aY;
66
67      aX = netX / body->getMass();
68      aY = netY / body->getMass();
69
70      double newVelx = body->velocity().x + seconds * aX;
71      double newVely = body->velocity().y + seconds * aY;
72
73      body->setVelocity(newVelx, newVely);
74
75      double newPosx = body->position().x + seconds * newVelx;
76      double newPosy = body->position().y + seconds * newVely;
77
78      body->setPosition(newPosx, newPosy);
79      body->setLocation(winScale, winSize);
80    }
81  }
82  // iostream overloads
83  std::istream& NB::operator>>(std::istream& input, Universe& universe) {
84    int newNumBodies;
85    double newRadius;
86
87    input >> newNumBodies >> newRadius;
88
89    universe.numBodies = newNumBodies;
```

```
  universe.rad = newRadius;
  universe.bodies.clear();

  for (int i = 0; i < universe.numBodies; ++i) {
    std::unique_ptr<CelestialBody> body = std::make_unique<CelestialBody>();
    input >> *body;
    body->loadImage();
    universe.bodies.push_back(std::move(body));
  }
  return input;
}
std::ostream& NB::operator<<(std::ostream& out, const Universe& universe) {
  out << universe.numBodies << endl;
  out << universe.rad << endl;
  for (const auto& body : universe.bodies) {
    out << *body << endl;
  }
  return out;
}
```

Listing 45: Universe.hpp

```
// Copyright 2024 Chris Lambert
#pragma once
#include <string>
#include <vector>
#include <memory>
#include "CelestialBody.hpp"

namespace NB {
class Universe : public CelestialBody {
 public:
  friend std::istream& operator>>(std::istream& input, Universe& universe);
  friend std::ostream& operator<<(std::ostream& out, const Universe&
   universe);
  Universe() = default;
  explicit Universe(std::string);
  void load(float);
  int numPlanets() { return numBodies; }
  double radius() { return rad; }
  void step(double seconds);

  ~Universe() {}
  CelestialBody& operator[](size_t index) {
    if (index >= bodies.size()) {
      throw std::out_of_range("Index out of range");
    }
    return *bodies[index];
  }

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

 private:
  int numBodies;
```

```
  double rad;
  double winScale;
  double winSize;
  std::vector<std::unique_ptr<CelestialBody>> bodies;
};
std::istream& operator>>(std::istream& input, Universe& universe);
std::ostream& operator<<(std::ostream& os, const Universe& universe);
}  // namespace NB
```

Listing 46: test.cpp

```
// Copyright 2024 Chris Lambert
#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main
#include <fstream>
#include <sstream>
#include <boost/test/unit_test.hpp>
#include "CelestialBody.hpp"
#include "Universe.hpp"
#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Audio.hpp>

BOOST_AUTO_TEST_CASE(celestial_body_IO_test) {
  std::string line("1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24    earth.gif");
  NB::CelestialBody body;
  std::istringstream(line) >> body;

  std::ostringstream oss;
  oss << body;

  BOOST_CHECK_EQUAL(oss.str(),
                    "1.496000e+11 0.000000e+00 0.000000e+00 2.980000e+04 5.974000e+24 earth.gif");
}

BOOST_AUTO_TEST_CASE(universe_IO_test) {
  NB::Universe universe("planets.txt");
  std::ostringstream oss;
  oss << universe;
  BOOST_CHECK(oss);
}

BOOST_AUTO_TEST_CASE(universe_getter_test) {
  NB::Universe universe("planets.txt");

  // Check if the universe is loaded with the correct number of bodies and radius
  BOOST_CHECK_EQUAL(universe.numPlanets(), 5);
  BOOST_CHECK_EQUAL(universe.radius(), 2.50e+11);
}
BOOST_AUTO_TEST_CASE(celestial_body_getter_test) {
  std::string line("1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24    earth.gif");
  NB::CelestialBody body;
  std::istringstream(line) >> body;

  // Check if the universe is loaded with the correct number of bodies and
```

```
    radius
  BOOST_CHECK_EQUAL(body.velocity().x, 0);
  double a = 1.49599994e+11;
  BOOST_CHECK_CLOSE(body.position().x, a, 0.0001);
}
BOOST_AUTO_TEST_CASE(universe_step__test) {
  NB::Universe universe("planets.txt");
  double x = 1.4960e+11;
  double y = 1.19750003e+09;

  universe.step(25000);

  BOOST_CHECK_CLOSE(universe[0].position().x, x, 0.1);
  BOOST_CHECK_CLOSE(universe[2].position().y, y, 0.1);
}
```