Computing IV Sec 202: Project Portfolio

William Susi

Spring 2023

Contents

Time to Complete Portfolio: 16 Hours

1 PS0: Hello SFML

1.1 Overview

In this assignment, we were introduced to SFML. Our goal was to simultaneously run two different windows. One containing the SFML demo of a green circle, and the other a sprite of our choice that was movable using the arrow keys.

1.2 End Product

Below is the output. On the left is the sample green circle. On the right is the custom movable lightsaber that makes a lightsaber swinging noise when moved.

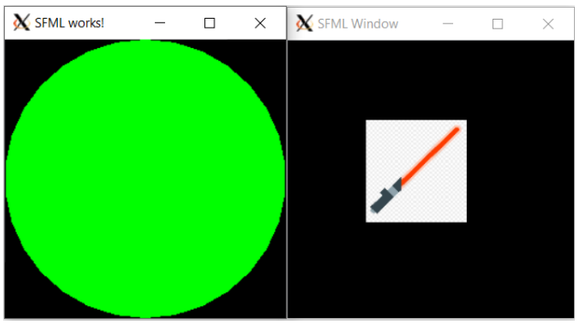

Figure 1: PS0 Output

1.3 What I Already Knew

Since this was an intro project, I only had my past C++ coding experience.

1.4 Design Decisions and Implementations

There were not many major decisions for this project as it was a demo to learn SFML. However, since I chose a lightsaber, I thought it was best that a noise accompany the movement, so I chose to do such.

1.5 What I Learned

I learned how to run a basic SFML program. This included generating a window and drawing a sprite to that window. I learned how to move a sprite using keystrokes which in turn could cause an action, in this cause, make a noise.

1.6 Challenges

There were were no noticeable challenges along the way.

1.7 Codebase

Makefile:

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

.PHONY: all clean lint

all: sfml-app lint

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

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

clean:
    rm *.o sfml-app

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

main.cpp:

```
// Copyright 2023 William Susi

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

int main() {
    // Set up green circle window
    sf::RenderWindow circleWindow(sf::VideoMode(200, 200), "SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

    // Set up extended code demo window
    sf::RenderWindow window(sf::VideoMode(500, 500), "SFML Window");
    sf::Texture texture;
    if (!texture.loadFromFile("sprite.png")) {
        return EXIT_FAILURE;
    }
    sf::Sprite sprite(texture);
    sprite.setScale(.2, .2);

    // Create lightsaber audio
    sf::SoundBuffer buffer;
    if (!buffer.loadFromFile("lightsaber_swing.ogg")) {
        return EXIT_FAILURE;
    }
    sf::Sound sound;
    sound.setBuffer(buffer);

    // Keep window open until closed
    while (circleWindow.isOpen() && window.isOpen()) {
        sf::Event event;
        while (circleWindow.pollEvent(event) || window.pollEvent(event)) {
```

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

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {  // move sprite
    up
            sprite.move(0, -1);
            sound.play();
        } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {  //
    move sprite down
            sprite.move(0, 1);
            sound.play();
        } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {  //
    move sprite left
            sprite.move(-1, 0);
            sound.play();
        } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {  //
    move sprite right
            sprite.move(1, 0);
            sound.play();
        }

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

        circleWindow.clear();
        circleWindow.draw(shape);
        circleWindow.display();
    }

    return 0;
}
```

2 PS1: Photomagic with LFSR

2.1 Overview

In this project, we were tasked with making a program that could encrypt and decrypt an image given a binary seed. To do this, a class FibLFSR was made with similarities to a linear feedback shift register. Using this class, an image's pixels could be randomized and the image could be 'encrypted'. Based off the design of the FibLFSR, an image could be encrypted and decrypted using the same seed.

2.2 End Product

Below is the output. On the left is the encryption, from the Darth Vader image into seeming random 'static'. On the right is the decryption, from 'static' back into the original Darth Vader image.

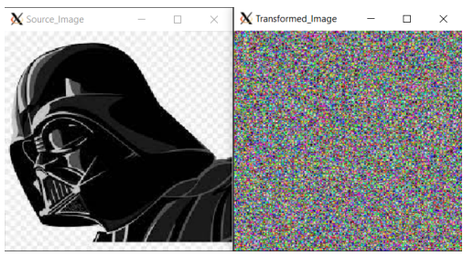

Figure 2: Encryption

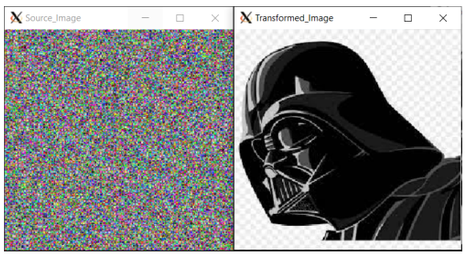

Figure 3: Decryption

2.3 What I Already Knew

I knew how to shift bits and how shift registers worked for reference when creating the LibLFSR class. I also knew how to make a window and draw to it in SFML from ps0.

2.4 Design Decisions and Implementations

I chose to represent the FibLFSR class as a string. This made it so accessing certain elements and appending or taking a character from the front to represent a shift wasn't too difficult. Inside the class, the 'step' function was made to represent a shift. This function XORed specific bit positions, with the final bit result being appended to the string, and the front bit being removed. This was then used by the 'generate' function to make a binary number of a given size. The function looped for a given size and generated a bit until it had a string of bits and had created a number. To encrypt/decrypt the image, the 'transform' function used this idea to randomize a given image's pixels. Each pixel's rgb value could be represented as an 8-bit number so 'generate' was used with an argument of 8 to seemingly generate a completely random red, green, or blue value. This made each pixel completely different from it's original rgb values.

As extra credit I made it so not only a binary seed could be used to encrypt the image, but a seed of any numbers or characters could be used. For this, I accounted for any character located in the ASCII table. What I did was make any odd characters in the ASCII table a 1 and any even a 0. This made it so 0 and 1 still remained the same while other characters could now be made into 0s and 1s. Any password over length 16 had those trailing bits ignored and anything shorter had it's remaining bits alternating between '0's and '1's. These together make it seem as random as possible while also making it so the same alphanumeric seed made the same binary seed every time.

2.5 What I Learned

I learned how to run two windows simultaneously in SFML. For an image, I learned how access and edit the rgb values of it's pixels.

2.6 Challenges

There were were no major challenges along the way.

2.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
DEPS = FibLFSR.hpp
OBJ = FibLFSR.o
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_unit_test_framework

.PHONY: all clean lint

all: PhotoMagic test lint

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

PhotoMagic: PhotoMagic.o $(OBJ)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

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

clean:
    rm *.o PhotoMagic test

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

Photomagic.cpp:

```
// Copyright 2023 William Susi

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

/*
Encrypt: "./PhotoMagic image.png output.png 1010101010101010"
         "./PhotoMagic image.png output.png banana"
Decrypt: "./PhotoMagic output.png output2.png 1010101010101010"
         "./PhotoMagic output.png output2.png banana"
*/

int main(int argc, char** argv) {
    sf::Image sourceImage;
    sf::Image transformedImage;
    if (!sourceImage.loadFromFile(argv[1]) || !transformedImage.loadFromFile(argv[1])) {
        return -1;
    }

    // Convert string seed to binary seed
    FibLFSR seed = FibLFSR(generateBinarySeed(argv[3]));
    transform(transformedImage, &seed);

    sf::Vector2u size = sourceImage.getSize();
    sf::RenderWindow sourceWindow(sf::VideoMode(size.x, size.y), "Source_Image");
```

```
    sf::RenderWindow transformedWindow(sf::VideoMode(size.x, size.y), "
    Transformed_Image");

    sf::Texture sourceTexture;
    sourceTexture.loadFromImage(sourceImage);
    sf::Sprite sourceSprite;
    sourceSprite.setTexture(sourceTexture);

    sf::Texture transformedTexture;
    transformedTexture.loadFromImage(transformedImage);
    sf::Sprite transformedSprite;
    transformedSprite.setTexture(transformedTexture);

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

        sourceWindow.clear();
        sourceWindow.draw(sourceSprite);
        sourceWindow.display();

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

    if (!transformedImage.saveToFile(argv[2])) {
        return -1;
    }

    return 0;
}
```

FibLFSR.cpp:

```
// Copyright 2023 William Susi

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

FibLFSR::FibLFSR(string seed) {
    _register = seed;
}

string FibLFSR::getRegister() const {
    return _register;
}

// Makes one shift on a FibLFSR object
int FibLFSR::step() {
```

```
    int resultBit = _register[5] ^ (_register[3] ^ (_register[2] ^ _register[0]));

    _register = _register.substr(1, 16);
    _register.append(to_string(resultBit));

    return resultBit;
}

// Makes k shifts on a FibLFSR object with a returned number dictated by the fall off bits
int FibLFSR::generate(int k) {
    int result = 0;
    for (int i = 0; i < k; i++) {
        result = (result * 2) + step();
    }
    return result;
}

ostream& operator<<(ostream& outStream, const FibLFSR& lfsr) {
    outStream << lfsr.getRegister();
    return outStream;
}

// Encrypts/decrypts a image given a seed.
void transform(sf::Image& img, FibLFSR* reg) {
    sf::Color pixel;
    for (unsigned int x = 0; x < img.getSize().x; x++) {
        for (unsigned int y = 0; y < img.getSize().y; y++) {
            pixel = img.getPixel(x, y);

            pixel.r = pixel.r ^ reg->generate(8);
            pixel.g = pixel.g ^ reg->generate(8);
            pixel.b = pixel.b ^ reg->generate(8);

            img.setPixel(x, y, pixel);
        }
    }
}

// Generates a pseudo-random binary seed of length 16
string generateBinarySeed(string seed) {
    string binarySeed = "";
    int size = seed.size();
    int i;
    for (i = 0; i < size && i < 16; i++) {
        if (seed[i] % 2 == 1) {
            binarySeed.append("1");
        } else {
            binarySeed.append("0");
        }
    }

    for (i = size; i < 16; i++) {
        if (i % 2 == 1) {
            binarySeed.append("1");
        } else {
            binarySeed.append("0");
        }
```

```
    }

    return binarySeed;
}
```

FibLFSR.hpp:

```
// Copyright 2023 William Susi

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

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

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

    string getRegister() const;

    int step();
    int generate(int k);
 private:
    string _register;
};

void transform(sf::Image&, FibLFSR*);

string generateBinarySeed(string seed);

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

test.cpp:

```
// Copyright 2023 William Susi

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

BOOST_AUTO_TEST_CASE(testStep) {
    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_REQUIRE_EQUAL(l.step(), 1);
```

```
19      BOOST_REQUIRE_EQUAL(l.step(), 0);
20  }
21
22  BOOST_AUTO_TEST_CASE(testGenerate1) {
23      FibLFSR l2("1011011000110110");
24      BOOST_REQUIRE_EQUAL(l2.generate(9), 51);
25  }
26
27  BOOST_AUTO_TEST_CASE(testGenerate2) {
28      FibLFSR l3("1011011000110110");
29
30      BOOST_CHECK_EQUAL(l3.generate(5), 3);
31      BOOST_CHECK_EQUAL(l3.getRegister(), "1100011011000011");
32      BOOST_CHECK_EQUAL(l3.generate(5), 6);
33      BOOST_CHECK_EQUAL(l3.getRegister(), "1101100001100110");
34      BOOST_CHECK_EQUAL(l3.generate(5), 14);
35      BOOST_CHECK_EQUAL(l3.getRegister(), "0000110011001110");
36      BOOST_CHECK_EQUAL(l3.generate(5), 24);
37      BOOST_CHECK_EQUAL(l3.getRegister(), "1001100111011000");
38      BOOST_CHECK_EQUAL(l3.generate(5), 1);
39      BOOST_CHECK_EQUAL(l3.getRegister(), "0011101100000001");
40      BOOST_CHECK_EQUAL(l3.generate(5), 13);
41      BOOST_CHECK_EQUAL(l3.getRegister(), "0110000000101101");
42      BOOST_CHECK_EQUAL(l3.generate(5), 28);
43      BOOST_CHECK_EQUAL(l3.getRegister(), "0000010110111100");
44  }
45
46  BOOST_AUTO_TEST_CASE(testGetReg) {
47      FibLFSR l4("1011011000110110");
48      BOOST_CHECK_EQUAL(l4.getRegister(), "1011011000110110");
49  }
50
51  BOOST_AUTO_TEST_CASE(testOutputOverload) {
52      FibLFSR l5("1011011000110110");
53      std::stringstream ss;
54      ss << l5;
55      BOOST_CHECK_EQUAL(ss.str(), "1011011000110110");
56  }
57
58  BOOST_AUTO_TEST_CASE(testSeedGenDefault) {
59      BOOST_CHECK_EQUAL(generateBinarySeed("0000000000000000"), "
        0000000000000000");
60      BOOST_CHECK_EQUAL(generateBinarySeed("1111111111111111"), "
        1111111111111111");
61      BOOST_CHECK_EQUAL(generateBinarySeed("0101001000111100"), "
        0101001000111100");
62  }
63
64  BOOST_AUTO_TEST_CASE(testSeedGenWords) {
65      BOOST_CHECK_EQUAL(generateBinarySeed("RevengeOfTheSith").size(), 16);
66      BOOST_CHECK_EQUAL(generateBinarySeed("RevengeOfTheSith"), "
        0101011100011100");
67      BOOST_CHECK_EQUAL(generateBinarySeed("RevengeOfTheSith"), "
        0101011100011100");
68  }
69
70  BOOST_AUTO_TEST_CASE(testSeedGenOverflow) {
71      BOOST_CHECK_EQUAL(generateBinarySeed("a7sdybj';.,234fd@$#sdaf").size(),
        16);
```

```
72      BOOST_CHECK_EQUAL(generateBinarySeed("a7sdybj';.,234fd@$#sdaf"), "
    1110100110001000");
73      BOOST_CHECK_EQUAL(generateBinarySeed("a7sdybj';.,234fd"), "
    1110100110001000");
74      BOOST_CHECK_EQUAL(generateBinarySeed("a7sdybj';.,234fd"),
75      generateBinarySeed("a7sdybj';.,234fd@$#sdaf"));
76  }
77
78  BOOST_AUTO_TEST_CASE(testSeedGenShort) {
79      BOOST_CHECK_EQUAL(generateBinarySeed("123abc").size(), 16);
80      BOOST_CHECK_EQUAL(generateBinarySeed("123abc"), "1011010101010101");
81      BOOST_CHECK_EQUAL(generateBinarySeed("123abc"), "1011010101010101");
82  }
83
84  BOOST_AUTO_TEST_CASE(testTransform) {
85      string stringSeed = "1010101010101010";
86      string img = "vader.png";
87
88      FibLFSR seed = FibLFSR(stringSeed);
89      sf::Image sourceImage, transformedImage;
90      sourceImage.loadFromFile(img);
91      transformedImage.loadFromFile(img);
92
93      transform(transformedImage, &seed);
94      for (unsigned int x = 0; x < sourceImage.getSize().x; x++) {
95          for (unsigned int y = 0; y < sourceImage.getSize().y; y++) {
96              int count = 0;
97              if (sourceImage.getPixel(x, y).r == transformedImage.getPixel(x,
    y).r) {
98                  count++;
99              }
100             if (sourceImage.getPixel(x, y).g == transformedImage.getPixel(x,
    y).g) {
101                 count++;
102             }
103             if (sourceImage.getPixel(x, y).b == transformedImage.getPixel(x,
    y).b) {
104                 count++;
105             }
106             BOOST_CHECK_NE(count, 3);
107         }
108     }
109
110     seed = FibLFSR(stringSeed);
111     transform(transformedImage, &seed);
112     for (unsigned int x = 0; x < sourceImage.getSize().x; x++) {
113         for (unsigned int y = 0; y < sourceImage.getSize().y; y++) {
114             BOOST_REQUIRE_EQUAL(sourceImage.getPixel(x, y).r,
    transformedImage.getPixel(x, y).r);
115             BOOST_REQUIRE_EQUAL(sourceImage.getPixel(x, y).g,
    transformedImage.getPixel(x, y).g);
116             BOOST_REQUIRE_EQUAL(sourceImage.getPixel(x, y).b,
    transformedImage.getPixel(x, y).b);
117         }
118     }
119 }
```

3 PS2: Sokoban

3.1 Overview

In this assignment, we were tasked with making a simple version of the game Sokoban which is a simple 2d block-pushing game. The program was to be run with a given level file that contained a grid of characters representing the starting game state. The game includes the ability of being able to move the player around using WASD and reset the level using 'R'. The boxes/storage containers are able to be pushed around the game area with the goal of putting them into the storage spaces. Once all the boxes are in the storage spaces or all storage spaces are filled with boxes then the game is won.

3.2 End Product

Below is the output. Level one is being played. On the left is the starting game state. On the right is the game being won.

Figure 4: Level 1 Start

Figure 5: Level 1 Won

3.3 What I Already Knew

I knew some basics of SFML for sprite movement and drawing to the screen. As for C++, I knew a variety of containers that could be used to develop the class. I also had past experience of implementing game mechanics/boundary conditions from previous school and personal projects.

3.4 Design Decisions and Implementations

For the game internals I created a Sokoban class. To display the game, I separated the game field into many different object containers. I first made a vector of textures that were loaded in once. These were to be shared between multiple sprites because sprites take up far less memory than textures, so this improved load speeds and reduced memory usage. I then made a 2d vector of characters to hold an exact copy of level file. This was for ease of access later on for game mechanics. As for displaying the game, I separated the environment(walls and floors), storage containers/boxes, storage spots, and the player into different sprite containers. This made it so that certain game elements could be drawn over each other/have same positions, such as the player standing on the floor. To display these, the class inherited from SFML draw class and the 'draw' function was overloaded so if you called 'draw' with a Sokoban object it would draw the whole game out.

As for the core game mechanics, I controlled most of them through the 'movePlayer' and 'isWon' functions. In the 'movePlayer' function I made a vector of x,y pairs that represented the displacement after a key press. This made it so I could generalize the game mechanics across keystroke presses to reduce the code four fold. For each movement I checked that it wouldn't be an illegal movement and would act appropriately with the box. For the player,

that meant they could not go out of bounds, through a wall, or into a box. However, the player should be allowed to push a box, unless another box was in front of it, or by pushing the box it would go out of bounds or through a wall. If any such illegal movement was attempted, the 'movePlayer' made sure no such movement was made. If the move was legal the respective sprite's positions were updated accordingly.

There were three scenarios where the player could win and the 'isWon' function handled them. The first two were if there were equal boxes to spaces, or more boxes than spaces, then the player won once all the storage spaces were filled. The third way was if there were more spaces than boxes, and then the player won once all the boxes were in a storage location. To account for all of them the lower number between the number of boxes and the number of storage spaces could be used to determine when the game was won. To check if boxes were in storage spaces I looped through the vectors of both. For each box/storage combination I compared their position using the 'find_if' algorithm paired with a lambda function. If by the end of looping the number of boxes in storage spots met that determined lower number, then the player had won.

For extra credit, I added a few different things. I first added a timer in the top left that counted the time passed after the level began. Some small features I added were a victory sound when the game was won, and a player model change when the player's direction changed. As a custom feature I added the ability for the player to play the next level if 'N' is pressed or play the previous level if 'P' is pressed after completing the current level. To go to the next/previous level that level must exist or the game will just remain on the winning screen.

3.5 What I Learned

I continued to learn more about SFML, especially sprites and the manipulation of them. I learned how to properly overload the draw function for user made classes/objects. I learned/used lambda functions for the first time.

3.6 Challenges

Everything turned out how I wanted it, but this was the first project with some more challenging problems. The first big trouble I had was overloading the draw function; it was simple after I completed it, but it was hard to find how to overload it properly using SFML API, so it became somewhat of a trial and error. It also took a bit of time trying to center the victory text, but that too felt like was because of SFML API. In general it was tough to figure out how I wanted my game to be represented at the beginning, so I was constantly making more containers to hold the game objects.

3.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
DEPS = Sokoban.hpp
OBJ = Sokoban.o
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework

.PHONY: all clean lint

all: Sokoban lint

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

Sokoban: main.o $(OBJ)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o Sokoban

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

main.cpp:

```
// Copyright 2023 William Susi

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

// Call ./Sokoban level1.txt

int main(int argc, char** argv) {
    string level = argv[1];
    START: Sokoban game(level);
    // std::cout << game << std::endl;

    sf::RenderWindow window(sf::VideoMode(game.getWidth() * 64, game.
   getHeight() * 64), "Sokoban");

    // Set up clock
    sf::Clock clock;
    sf::Time time;
    time = clock.getElapsedTime();

    // Set up default font
    sf::Font font;
    font.loadFromFile("SpaceMono-Regular.ttf");

    // Set up intital timer display
    sf::Text timerText("00:00", font);
    timerText.setFillColor(sf::Color::Green);
    timerText.setCharacterSize(18);
    timerText.setPosition(5, 12);

    // Pre-load victory display text
```

```
31      sf::Text victoryText("You Win!", font);
32      victoryText.setFillColor(sf::Color::Green);
33      int size = 12;
34      victoryText.setCharacterSize((game.getWidth() < game.getHeight()) ?
35      game.getWidth() * size : game.getHeight() * size);
36      victoryText.setPosition(window.getView().getCenter().x - (victoryText.
        getLocalBounds().width/2),
37      window.getView().getCenter().y - 64);
38
39      // Set up victory sound
40      sf::SoundBuffer buffer;
41      if (!buffer.loadFromFile("victory.ogg")) {
42          return EXIT_FAILURE;
43      }
44      sf::Sound sound;
45      sound.setBuffer(buffer);
46      bool soundPlayed = false;
47
48      while (window.isOpen()) {
49          sf::Event event;
50          while (window.pollEvent(event)) {
51              if (event.type == sf::Event::Closed) {
52                  window.close();
53              }
54
55              // Check for different keyboard input events
56              if (event.type == sf::Event::KeyPressed) {
57                  // Check for movement
58                  if (event.key.code == sf::Keyboard::W|| event.key.code == sf
        ::Keyboard::Up) {
59                      game.movePlayer(UP);
60                  } else if (event.key.code == sf::Keyboard::S||
61                  event.key.code == sf::Keyboard::Down) {
62                      game.movePlayer(DOWN);
63                  } else if (event.key.code == sf::Keyboard::A ||
64                  event.key.code == sf::Keyboard::Left) {
65                      game.movePlayer(LEFT);
66                  } else if (event.key.code == sf::Keyboard::D ||
67                  event.key.code == sf::Keyboard::Right) {
68                      game.movePlayer(RIGHT);
69                  }
70
71                  // Check for restart
72                  if (event.key.code == sf::Keyboard::R) {
73                      goto START;
74                  }
75
76                  // Check for previous or next levels
77                  int levelNum = stoi(level.substr(5, (level.size() - 9)));
78                  if (game.isWon() && event.key.code == sf::Keyboard::N) {
79                      string tempLevel = "level" + to_string(levelNum + 1) + "
        .lvl";
80                      std::fstream test;
81                      test.open(tempLevel, std::fstream::in);
82                      if (!test.good()) {
83                          break;
84                      }
85                      level = tempLevel;
86                      goto START;
```

```
                } else if (game.isWon() && event.key.code == sf::Keyboard::P
    ) {
                    string tempLevel = "level" + to_string(levelNum - 1) + "
    .lvl";
                    std::fstream test;
                    test.open(tempLevel, std::fstream::in);
                    if (!test.good()) {
                        break;
                    }
                    level = tempLevel;
                    goto START;
                }
            }
        }

        // Update timer
        if (!game.isWon()) {
            time = clock.getElapsedTime();
        }

        // Set timer
        int seconds = time.asSeconds();
        if (seconds / 60 < 10 && seconds % 60 < 10) {
            timerText.setString("0" + to_string(seconds/60) + ":0" +
    to_string(seconds%60));
        } else if (seconds / 60 < 10) {
            timerText.setString("0" + to_string(seconds/60) + ":" +
    to_string(seconds%60));
        } else if (seconds % 60 < 10) {
            timerText.setString(to_string(seconds/60) + ":0" + to_string(
    seconds%60));
        } else {
            timerText.setString(to_string(seconds/60) + ":" + to_string(
    seconds%60));
        }

        // Display everything
        window.clear();
        window.draw(game);
        window.draw(timerText);
        if (game.isWon()) {
            window.draw(victoryText);
            if (!soundPlayed) {
                sound.play();
                soundPlayed = true;
            }
        }
        window.display();
    }

    return 0;
}
```

Sokoban.cpp:

```
// Copyright 2023 William Susi

#include "Sokoban.hpp"

Sokoban::Sokoban(std::string lvl) {
```

```
6     sf::Image img;
7     sf::Texture text;
8
9     if (!img.loadFromFile("block_06.png")) {  // Wall
10        exit(-1);
11    }
12    text.loadFromImage(img);
13    textures.push_back(text);
14
15    if (!img.loadFromFile("ground_01.png")) {  // Floor
16        exit(-1);
17    }
18    text.loadFromImage(img);
19    textures.push_back(text);
20
21    if (!img.loadFromFile("ground_04.png")) {  // Storage
22        exit(-1);
23    }
24    text.loadFromImage(img);
25    textures.push_back(text);
26
27    if (!img.loadFromFile("crate_03.png")) {  // Box
28        exit(-1);
29    }
30    text.loadFromImage(img);
31    textures.push_back(text);
32
33    if (!img.loadFromFile("player_17.png")) {  // Player_Right
34        exit(-1);
35    }
36    text.loadFromImage(img);
37    textures.push_back(text);
38
39    if (!img.loadFromFile("player_05.png")) {  // Player_Down
40        exit(-1);
41    }
42    text.loadFromImage(img);
43    textures.push_back(text);
44
45    if (!img.loadFromFile("player_20.png")) {  // Player_Left
46        exit(-1);
47    }
48    text.loadFromImage(img);
49    textures.push_back(text);
50
51    if (!img.loadFromFile("player_08.png")) {  // Player_Up
52        exit(-1);
53    }
54    text.loadFromImage(img);
55    textures.push_back(text);
56
57    std::fstream level;
58    level.open(lvl, std::fstream::in);  // Load level file
59    if (!level.good()) {
60        cout << "Input file unable to open." << endl;
61        exit(-1);
62    }
63
64    level >> *this;  // Create game from level
```

```
65 }
66
67 int Sokoban::getWidth() const {
68     return mapChar.at(0).size();
69 }
70
71 int Sokoban::getHeight() const {
72     return mapChar.size();
73 }
74
75 void Sokoban::movePlayer(Direction dir) {
76     if (isWon()) {  // Prevent movement after game ends
77         return;
78     }
79
80     vector<pair<int, int>> xyDisp = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
81     int xP = playerSprite.getPosition().x/64;
82     int yP = playerSprite.getPosition().y/64;
83     pair<int, int> disp = xyDisp[dir];
84
85     // Prevent player from moving outside bounds
86     if (xP + disp.first < 0 || xP + disp.first >= static_cast<int>(mapChar.
    at(0).size()) ||
87         yP + disp.second < 0 || yP + disp.second >= static_cast<int>(mapChar
    .size())) {
88         return;
89     }
90
91     // Prevent player from moving through walls
92     if (mapChar.at(yP + disp.second).at(xP + disp.first) == '#') {
93         return;
94     }
95
96     // Implementation of proper box/player physics
97     for (sf::Sprite &box : boxSprites) {
98         int xB = box.getPosition().x/64;
99         int yB = box.getPosition().y/64;
100        // If box is in front of player
101        if (xB == xP + disp.first && yB == yP + disp.second) {
102            // Does not allow box to move if next space is out of bounds
103            if (xB + disp.first < 0 || xB + disp.first >= static_cast<int>(
    mapChar.at(0).size()) ||
104                yB + disp.second < 0 || yB + disp.second >= static_cast<int
    >(mapChar.size())) {
105                return;
106            }
107            // If the space after the box is a wall or another box do now
    allow the player to move
108            if (mapChar.at(yB + disp.second).at(xB + disp.first) != '#' &&
109                mapChar.at(yB + disp.second).at(xB + disp.first) != 'A' ) {
110                mapChar.at(yB + disp.second).at(xB + disp.first) = 'A';
111                box.move(disp.first * 64, disp.second * 64);
112            } else {
113                return;
114            }
115        }
116    }
117
118    // Move player and update changed sprites
```

```
119     mapChar.at(yP).at(xP) = '.';
120     mapChar.at(yP + disp.second).at(xP + disp.first) = '@';
121     playerSprite.move(disp.first * 64, disp.second * 64);
122     playerSprite.setTexture(textures[dir + 4]);
123 }
124 
125 bool Sokoban::isWon() const {
126     int size = (boxSprites.size() < storageLocations.size()) ?
127     boxSprites.size() : storageLocations.size();
128     int inPlace = 0;
129 
130     // Check for different game winning scenarios
131     for (pair<int, int> const& storage : storageLocations) {
132         vector<sf::Sprite>::const_iterator result = find_if(boxSprites.
    cbegin(), boxSprites.cend(),
133         [storage](const sf::Sprite& box) {
134             return (box.getPosition().x == storage.first && box.getPosition
    ().y == storage.second);
135         });
136         if (result != boxSprites.cend()) {
137             inPlace++;
138         }
139     }
140 
141     return (inPlace == size);
142 }
143 
144 std::istream& operator>>(std::istream& inStream, Sokoban& game) {
145     string line;
146     int width, height;
147     inStream >> height >> width;
148 
149     getline(inStream, line);
150     game.terrainSprites.clear();
151     game.mapChar.clear();
152 
153     for (int h = 0; h < height; h++) {
154         getline(inStream, line);
155         game.terrainSprites.emplace_back();
156         game.mapChar.emplace_back();
157 
158         for (int w = 0; w < width; w++) {
159             char space = line[w];
160             sf::Sprite terrain;
161 
162             if (space == '#') {  // Wall
163                 terrain.setTexture(game.textures[0]);
164             } else if (space == '.') {  // Floor
165                 terrain.setTexture(game.textures[1]);
166             } else if (space == 'a') {  // Storage
167                 terrain.setTexture(game.textures[2]);
168                 game.storageLocations.push_back(std::make_pair(w * 64, h *
    64));
169             } else if (space == 'A') {  // Box
170                 terrain.setTexture(game.textures[1]);
171                 sf::Sprite box;
172                 box.setTexture(game.textures[3]);
173                 box.setPosition(w * 64, h * 64);
174                 game.boxSprites.push_back(box);
```

```
            } else if (space == '@') {  // Player
                terrain.setTexture(game.textures[1]);
                sf::Sprite player;
                player.setTexture(game.textures[5]);
                player.setPosition(w * 64, h * 64);
                game.playerSprite = player;
            } else {
                cout << "Character not recognized from file." << endl;
                exit(-1);
            }
            terrain.setPosition(w * 64, h * 64);
            game.terrainSprites.at(h).push_back(terrain);
            game.mapChar.at(h).push_back(space);
        }
    }

    return inStream;
}

std::ostream& operator<<(std::ostream& outStream, const Sokoban& game) {
    for (size_t h = 0; h < game.mapChar.size(); h++) {
        for (size_t w = 0; w < game.mapChar.at(0).size(); w++) {
            outStream << game.mapChar.at(h).at(w);
        }
        outStream << endl;
    }
    return outStream;
}

void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
    for (size_t h = 0; h < terrainSprites.size(); h++) {
        for (size_t w = 0; w < terrainSprites.at(0).size(); w++) {
            target.draw(terrainSprites.at(h).at(w), states);
        }
    }

    for (sf::Sprite const& box : boxSprites) {
        target.draw(box, states);
    }

    target.draw(playerSprite, states);
}
```

Sokoban.hpp:

```
// Copyright 2023 William Susi

#include <iostream>
#include <utility>
#include <fstream>
#include <algorithm>
#include <string>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

using std::vector;
```

```
15 using std::pair;
16 using std::string;
17 using std::to_string;
18 using std::cout;
19 using std::endl;
20
21 typedef enum {
22     RIGHT,
23     DOWN,
24     LEFT,
25     UP
26 } Direction;
27
28 class Sokoban: public sf::Drawable{
29  public:
30     explicit Sokoban(string lvl);
31
32     int getWidth() const;
33     int getHeight() const;
34
35     void movePlayer(Direction dir);
36     bool isWon() const;
37
38     friend std::istream& operator>>(std::istream& inStream, Sokoban& level);
39     friend std::ostream& operator<<(std::ostream& outStream, const Sokoban&
       level);
40
41  private:
42     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
       const;
43
44     vector<vector<char>> mapChar;
45     vector<vector<sf::Sprite>> terrainSprites;
46     vector<sf::Sprite> boxSprites;
47     vector<pair<int, int>> storageLocations;
48     sf::Sprite playerSprite;
49     vector<sf::Texture> textures;
50 };
```

4 PS3: Pythagorean Tree

4.1 Overview

This project creates a Pythagorean Tree from a given side length, size, and angle measure. A Pythagorean Tree starts as a square, and generates two more squares (of possible different sizes depending on the given angle measure) in opposing directions. Each new square will generate two new squares in the same manner as many times as the given recursion depth. Depending on the depth, it starts to make a tree-like structure.

4.2 End Product

Below is the output. This is the simplest version of the tree using a 45 degree angle.

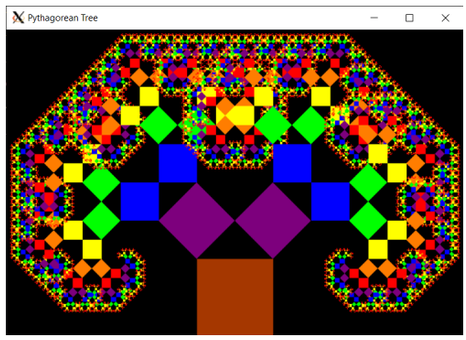

Figure 6: Pythagorean Tree: 45 Degrees

4.3 What I Already Knew

I had SFML experience from past projects and basic manipulation fo rectangular shapes. As for C++, I knew a how to create a basic tree and node class and use recursion within them.

4.4 Design Decisions and Implementations

I chose to make two classes, a Tree class and a Node class. A tree object has a root Node which represents the starting square in this case. Each Node is a rectangular sprite with the addition of a left and right node member to generate future squares. The Tree calls upon a recursive 'grow' function which creates two children from a given parent Node with the correct new position, angle, size, etc. This recursion loops for the given depth.

For extra credit, I added two things. The first feature I added was color changing depending on the depth of the tree. The starting color is always brown (like a stump) and then any following levels loop through the rainbow. The second feature I added was the ability to change the angle of the new generated squares. However, in doing so, a problem arose which is covered in the challenges section.

4.5 What I Learned

I learned what a Pythagorean tree is and how it's made. I learned more about SFML shapes and how translations effect them.

4.6 Challenges

I came across a few different challenges. The first was setting the origin of every new object. SFML API is pretty bad at documenting this, along with it being confusing in general, so it took me a while to get each new rotation/position correct. The second challenge, which I was unable to fix, was the problem resulting from doing the extra credit mentioned above which involved changing the angle of recursion. With a changed angle, the tree no longer fit in the output window, and I had trouble finding a way to adjust the window size appropriately. I could not figure out a proper way to make a window that would perfectly fit any given generated tree to screen.

4.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
DEPS = PTree.hpp
OBJ = PTree.o
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework

.PHONY: all clean lint

all: PTree lint

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

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

clean:
    rm *.o PTree

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

main.cpp:

```
// Copyright 2023 William Susi

#include "PTree.hpp"

int main(int argc, char** argv) {
    double side = stod(argv[1]);
    int depth = stoi(argv[2]);
    double angle = 45;
    if (argc == 4) {
        angle = stod(argv[3]);
    }

    // double side1 = side * cos(angle * M_PI / 180);
    // double side2 = side * sin(angle * M_PI / 180);

    // double width = depth * abs(side1-side2) * 2;
    // double height = depth * abs(side1-side2) * 1.5 - side1;

    // sf::Vector2f center(width/2 - side/2 + (side1-side2) * 3 , (height -
   side * 1.5));

    // Tree tree(side, depth, angle, center);
    // tree.pTree();
    // sf::RenderWindow window(sf::VideoMode(width, height), "Pythagorean
   Tree");

    double width = side * 6;
    double height = side * 4;
    sf::Vector2f center(width/2 - side/2, height - side);
    Tree tree(side, depth, angle, center);
    tree.pTree();
```

```
    sf::RenderWindow window(sf::VideoMode(width, height), "Pythagorean Tree");

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

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

    return 0;
}
```

Sokoban.cpp:

```
// Copyright 2023 William Susi

#include "PTree.hpp"

Tree::Tree(double length, int depth, double angle, sf::Vector2f center) {
    _sideLength = length;
    _recDepth = depth;
    _angle = angle;
    sf::Color color = sf::Color(165, 55, 0);
    _base = new Node(_sideLength, sf::Vector2f(0, 0), center, 0, color);
}

double Tree::getLength() const {
    return _sideLength;
}

// Starts the grow of the tree at the root
void Tree::pTree() {
    _base->grow(_angle, _recDepth);
}

// Calls draw on the root Node
void Tree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    target.draw(*_base, states);
}

Node::Node(double sideLength, sf::Vector2f origin,
           sf::Vector2f position, double rotation, sf::Color color) {
    _square.setSize(sf::Vector2f(sideLength, sideLength));
    _square.setOrigin(origin.x, origin.y);
    _square.setPosition(position.x, position.y);
    _square.setRotation(rotation);
    _square.setFillColor(color);

    _leftSquare = nullptr;
    _rightSquare = nullptr;
}

// Recursion function that generates branches of the tree
```

```
40 void Node::grow(double angle, int repeat) {
41     // End recursion after depth is reached
42     if (repeat == 0) {
43         return;
44     }
45     repeat--;
46 
47     // Calculate left and right square lengths
48     double side = _square.getLocalBounds().width;
49     double leftSide = (side * cos(angle * M_PI / 180));
50     double rightSide = (side * sin(angle * M_PI / 180));
51 
52     // Set left square's origin to the bottom left and right square's origin
       to the bottom right
53     sf::Vector2f leftOri(0, leftSide);
54     sf::Vector2f rightOri(rightSide, rightSide);
55 
56     auto bounds = _square.getLocalBounds();
57     sf::Vector2f leftPos(_square.getTransform().
58     transformPoint(bounds.left, bounds.top));
59     sf::Vector2f rightPos(_square.getTransform().
60     transformPoint(bounds.left + bounds.width, bounds.top));
61 
62     // Make new rotation
63     double leftRotate = _square.getRotation() - angle;
64     double rightRotate = _square.getRotation() - (angle - 90);
65 
66     // Get depth color
67     sf::Color color = generateColor(repeat);
68 
69     _leftSquare = new Node(leftSide, leftOri, leftPos, leftRotate, color);
70     _rightSquare = new Node(rightSide, rightOri, rightPos, rightRotate,
       color);
71 
72     // Call recursive 'grow' on each new square
73     _leftSquare->grow(angle, repeat);
74     _rightSquare->grow(angle, repeat);
75 }
76 
77 // Overload of SFML draw to draw each child node of a given parent
78 void Node::draw(sf::RenderTarget& target, sf::RenderStates states) const {
79     target.draw(_square, states);
80     if (_leftSquare == nullptr || _rightSquare == nullptr) {
81         return;
82     }
83     target.draw(*_leftSquare, states);
84     target.draw(*_rightSquare, states);
85 }
86 
87 // Generates the next color of the rainbow given a depth
88 sf::Color generateColor(int repeat) {
89     sf::Color colors[6] = {sf::Color(255, 0, 0), sf::Color(255, 125, 0), sf
       ::Color(255, 255, 0),
90                            sf::Color(0, 255, 0), sf::Color(0, 0, 255), sf::
       Color(125, 0, 125)};
91     return (colors[repeat % 6]);
92 }
```

Sokoban.hpp:

```cpp
// Copyright 2023 William Susi
#pragma once

#include <math.h>
#include <iostream>
#include <string>

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

using std::string;
using std::cout;
using std::endl;
using std::stod;
using std::stoi;

class Node: public sf::Drawable {
 public:
  Node(double sideLength, sf::Vector2f origin,
      sf::Vector2f position, double rotation, sf::Color color);
  void grow(double angle, int repeat);

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

  sf::RectangleShape _square;
  Node* _leftSquare;
  Node* _rightSquare;
};

class Tree: public sf::Drawable {
 public:
  Tree(double length, int depth, double angle, sf::Vector2f center);
  double getLength() const;
  void pTree();

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

  Node* _base;
  double _sideLength;
  int _recDepth;
  double _angle;
};

sf::Color generateColor(int repeat);
```

5 PS4: Checkers

5.1 Overview

This project builds the widely known game of Checkers. In this game, two players, black and red, look to remove each other's pieces by jumping over them. When a player no longer has any piece on the game board the other player wins. During a player's turn, a selected piece can move in two diagonal directions depending on the piece's color. If such a move is blocked by one of there own pieces, it can not move in that direction. If such a move is open they are free to move there. If such a move is blocked by an opposing piece, and the next space is free, that piece can be jumped, and removed from the game board. If a piece arrives at the opposing players side that piece is 'kinged', and can now move in all four diagonal directions. The game rules that were not yet implemented were multi-jumping, and forcing a player to make a jump if they can.

5.2 End Product

Below is the output. This is the standard setup for a Checkers game.

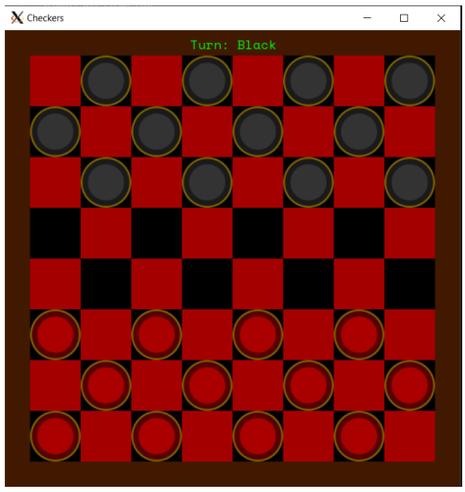

Figure 7: Checkers: Start

5.3 What I Already Knew

Going into this project I had already completed Sokoban, so I had a good foundation for creating a game UI in SFML and implementing its game mechanics using C++.

5.4 Design Decisions and Implementations

The code for Checkers was comprised of two major classes called 'Checkers' and 'Pawn'. The 'Pawn' class was meant to represent a piece and contained the sprite for a game piece along with a boolean that indicated if it was a king or not. In the 'Checkers' class I made a vector containing the game textures with similar reasoning to the textures vector in Sokoban. The game tiles were represented using a 2d-vector of rectangles. Each player's pieces were separated into their own respective vectors of 'Pawns', one representing the red player and the other the black. The Checkers class also had a boolean variable named '_blackTurn' to represent whose turn it was.

The game starts by creating a 'Checkers' object which does the following:

- Creates a base layer of brown which represents the board's border.
- Fills the 2d-vector of tiles with alternated red and black squares.
- Adds a black piece to its respective vector for every black tile that was in the first three rows and adds a red piece to its respective vector for every black tile that was in the final three rows.

Once the game was made a number of smaller helper functions were made to reduce repeat code and code length. Here are some of the more important ones:

- 'isBlackTurn' - Returns whose turn it is, black or red.
- 'toTile' - Converts a pixel's coordinates to a tile location.
- 'inBounds' - Checks if a piece's move is in bound.
- 'getCurrPieces' & 'getOppPieces' - Returns the current players pieces and the opposing players pieces depending on whose turn it is.
- 'clearHighlights' - Clears any previously highlighted pieces/tiles.

As for the larger functions, these include many of the core game mechanics as follows:

- 'takeTurn' - Switches the players turn.
- 'checkKing' - Checks if a pawn needs to be kinged by checking if any piece has reached the opponents side, and if so, makes that piece a king.
- 'onTile' - Checks if a piece can be found on a given tile by looping through a players' pieces and comparing their locations to the given tile.
- 'clickPieces' - Highlights a tile if the current player clicked on their piece at that tile. It gets the current players pieces and loops through, checking to see if the click was on a tile with their piece. If one is found, any past piece selection highlights are cleared, the new tile is highlighted, and the 'showMoves' function is called.
- 'showMoves' - Highlights the tiles where the current selected piece can possibly move. It first checks what type of piece it is, and if it is a king, to know what direction it can move in. It then loops through those directions to highlight the tiles it can actually move to. It checks the two moves of jumping into a adjacent open space and jumping an opposing piece where no other piece is on the next tile. Finally, the 'makeMove' function is called.
- 'makeMove' - Moves a player's selected piece into a highlighted move area. It first checks the game board for a selected/highlighted piece. It then makes sure the player's mouse click is on a highlighted move location by comparing the two. If the mouse click matches a spot where the piece can move, the piece is set to that spot, the current turn is switched to the next player, and any highlights are cleared. Finally, the 'removePieceJumped' function is called.
- 'removePieceJumped' - Removes a piece from the game board if it was jumped using the remove_if algorithm. The function starts out by finding the displacement between the starting spot of a piece and the move it just made. It then uses a lambda to check if an opposing piece can be found at that displacement. If found, remove_if returns a new end iterator that the 'erase' member function of vector takes to remove the associated piece from the game board.

5.5 What I Learned

I learned a few small things, but what I was able to put into practice/learn was refactoring. The internals of the game were originally comprised of many larger functions. Once I got the code working, refactoring took place to clean everything up and reduce code. This consisted of condensing lines and creating helper functions. Once in place, the code flowed much nicer, was easier to follow, and was more efficient.

5.6 Challenges

Overall this wasn't that different from Sokoban, so there weren't many new big challenges, but there were some smaller challenges. One problem that arose was the implementation of the 'remove_if' algorithm. A piece that had jumped another would occasionally duplicate itself because I wasn't properly implementing the 'remove_if' algorithm. Once I re-read the documentation on 'remove_if' I was able to properly use it and reduce the duplication problem.

5.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
DEPS = Checkers.hpp
OBJ = Checkers.o
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework

.PHONY: all clean lint

all: Checkers lint

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

Checkers: main.o $(OBJ)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o Checkers

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

main.cpp:

```
// Copyright 2023 William Susi

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

int main(int argc, char** argv) {
    Checkers game = Checkers();

    sf::RenderWindow window(sf::VideoMode(TILE_LENGTH * BOARD_SIZE + 2 *
   BORDER_WIDTH,
                                          TILE_LENGTH * BOARD_SIZE + 2 *
   BORDER_WIDTH), "Checkers");

    sf::Font font;
    font.loadFromFile("SpaceMono-Regular.ttf");

    sf::Text turnText("Turn: Black", font);
    turnText.setFillColor(sf::Color::Green);
    turnText.setCharacterSize(BORDER_WIDTH/2);

    sf::Text victoryText("", font);
    victoryText.setFillColor(sf::Color::Green);
    victoryText.setCharacterSize(TILE_LENGTH);

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

```

```
30              // Checks for player trying to select a piece
31              if (event.type == sf::Event::MouseButtonPressed && !game.isWon()
    ) {
32                  if (event.mouseButton.button == sf::Mouse::Left &&
33                      event.mouseButton.x >= BORDER_WIDTH &&
34                      event.mouseButton.x < TILE_LENGTH * BOARD_SIZE +
    BOARD_SIZE &&
35                      event.mouseButton.y >= BORDER_WIDTH &&
36                      event.mouseButton.y < TILE_LENGTH * BOARD_SIZE +
    BOARD_SIZE) {
37                      game.makeMove(sf::Vector2f(event.mouseButton.x, event.
    mouseButton.y));
38                      if (!game.clickPiece(sf::Vector2f(event.mouseButton.x,
    event.mouseButton.y))) {
39                          game.clearHighlights();
40                      }
41                  }
42              }
43          }
44
45          // Displays the winner if someone has won
46          if (game.isWon()) {
47              if (game.isBlackTurn()) {
48                  victoryText.setString("Red Wins!");
49              } else {
50                  victoryText.setString("Black Wins!");
51              }
52              victoryText.setPosition(window.getView().getCenter().x -
53                                      (victoryText.getLocalBounds().width/2),
54                                       window.getView().getCenter().y -
55                                      (victoryText.getLocalBounds().height/2));
56          }
57
58          // Sets whose turn it is
59          if (game.isBlackTurn()) {
60              turnText.setString("Turn: Black");
61          } else {
62              turnText.setString("Turn: Red");
63          }
64          turnText.setPosition(window.getView().getCenter().x - (turnText.
    getLocalBounds().width/2),
65                               BORDER_WIDTH/4);
66
67          // Creates border
68          sf::RectangleShape border(sf::Vector2f(TILE_LENGTH * BOARD_SIZE + 2
    * BORDER_WIDTH,
69                                                 TILE_LENGTH * BOARD_SIZE + 2
    * BORDER_WIDTH));
70          border.setFillColor(sf::Color(65, 25, 0));
71
72          // Display everything
73          window.clear();
74          window.draw(border);
75          window.draw(game);
76          window.draw(turnText);
77          window.draw(victoryText);
78          window.display();
79      }
80
```

```
    return 0;
}
```

Checkers.cpp:

```
// Copyright 2023 William Susi

#include "Checkers.hpp"

const sf::Color moveHighlight(255, 245, 195);
const sf::Color pieceHighlight(185, 255, 185);

Pawn::Pawn(sf::Sprite piece): _piece(piece), _isKing(false) {}

bool Pawn::isKing() const {
    return _isKing;
}

sf::Sprite Pawn::getPiece() const {
    return _piece;
}

void Pawn::movePiece(sf::Vector2f pieceXY) {
    _piece.setPosition(pieceXY.x, pieceXY.y);
}

void Pawn::makeKing(sf::Texture& texture) {
    _piece.setTexture(texture);
    _isKing = true;
}

Checkers::Checkers() {
    sf::Image img;
    sf::Texture text;

    // Load textures
    if (!img.loadFromFile("piece_images/blackpawn.png")) {
        exit(-1);
    }
    text.loadFromImage(img);
    _textures.push_back(text);

    if (!img.loadFromFile("piece_images/blackking.png")) {
        exit(-1);
    }
    text.loadFromImage(img);
    _textures.push_back(text);

    if (!img.loadFromFile("piece_images/redpawn.png")) {
        exit(-1);
    }
    text.loadFromImage(img);
    _textures.push_back(text);

    if (!img.loadFromFile("piece_images/redking.png")) {
        exit(-1);
    }
    text.loadFromImage(img);
    _textures.push_back(text);

```

```
56      // Set up initial game board state
57      for (int h = 0; h < BOARD_SIZE; h++) {
58          _tiles.emplace_back();
59          for (int w = 0; w < BOARD_SIZE; w++) {
60              sf::RectangleShape tile(sf::Vector2f(TILE_LENGTH, TILE_LENGTH));
61              sf::Sprite piece;
62              if ((h + w) % 2 == 0) {  // Set red tiles
63                  tile.setFillColor(sf::Color(165, 0, 0));
64              } else {  // Set black tiles
65                  tile.setFillColor(sf::Color::Black);
66                  if (h < 3) {  // Set black pawns
67                      piece.setTexture(_textures[0]);
68                      piece.setPosition(w * TILE_LENGTH + BORDER_WIDTH,
69                                        h * TILE_LENGTH + BORDER_WIDTH);
70                      _blackPieces.push_back(Pawn(piece));
71                  } else if (h > 4) {  // Set red pawns
72                      piece.setTexture(_textures[2]);
73                      piece.setPosition(w * TILE_LENGTH + BORDER_WIDTH,
74                                        h * TILE_LENGTH + BORDER_WIDTH);
75                      _redPieces.push_back(Pawn(piece));
76                  }
77              }
78              tile.setPosition(w * TILE_LENGTH + BORDER_WIDTH, h * TILE_LENGTH
    + BORDER_WIDTH);
79              _tiles.at(h).push_back(tile);
80          }
81      }
82
83      // Black starts
84      _blackTurn = true;
85  }
86
87  bool Checkers::isBlackTurn() const {
88      return _blackTurn;
89  }
90
91  // Switches player turn
92  void Checkers::takeTurn() {
93      if (isBlackTurn()) {
94          _blackTurn = false;
95      } else {
96          _blackTurn = true;
97      }
98  }
99
100 // Clears any previously highlighted tiles
101 void Checkers::clearHighlights() {
102     for (int h = 0; h < BOARD_SIZE; h++) {
103         for (int w = 0; w < BOARD_SIZE; w++) {
104             if ((h + w) % 2 == 0) {  // Tile red
105                 _tiles.at(h).at(w).setFillColor(sf::Color(165, 0, 0));
106             } else {
107                 _tiles.at(h).at(w).setFillColor(sf::Color::Black);
108             }
109         }
110     }
111 }
112
113 // Coverts a given pixel coordinate to a tile coordinate
```

```
114 int Checkers::toTile(double pixel) {
115     return (pixel - BORDER_WIDTH) / TILE_LENGTH;
116 }
117 
118 // Checks if a given move is inbounds
119 bool Checkers::inBounds(int move) {
120     return move >= 0 && move < BOARD_SIZE;
121 }
122 
123 // Gets the current player's pieces
124 vector<Pawn>& Checkers::getCurrPieces() {
125     if (isBlackTurn()) {
126         return _blackPieces;
127     }
128     return _redPieces;
129 }
130 
131 // Get the opposing player's pieces
132 vector<Pawn>& Checkers::getOppPieces() {
133     if (isBlackTurn()) {
134         return _redPieces;
135     }
136     return _blackPieces;
137 }
138 
139 void Checkers::checkKing() {
140     for (Pawn& piece : _blackPieces) {
141         if (((piece.getPiece().getPosition().y - BORDER_WIDTH) / TILE_LENGTH
    ) == 7) {
142             piece.makeKing(_textures[1]);
143         }
144     }
145 
146     for (Pawn& piece : _redPieces) {
147         if (((piece.getPiece().getPosition().y - BORDER_WIDTH) / TILE_LENGTH
    ) == 0) {
148             piece.makeKing(_textures[3]);
149         }
150     }
151 }
152 
153 // Check if a player has won
154 bool Checkers::isWon() {
155     return _blackPieces.size() == 0 || _redPieces.size() == 0;
156 }
157 
158 // Checks if a piece could be found on a given tile
159 bool Checkers::onTile(int tileX, int tileY, vector<Pawn>& pieces) {
160     if (tileX >= 0 && tileX < BOARD_SIZE && tileY >= 0 && tileY < BOARD_SIZE
    ) {
161        for (Pawn& piece : pieces) {
162             int pieceX = toTile(piece.getPiece().getPosition().x);
163             int pieceY = toTile(piece.getPiece().getPosition().y);
164             if (tileX == pieceX && tileY == pieceY) {
165                 return true;
166             }
167         }
168     }
169     return false;
```

```
}

// Given a mouse click, highlight a tile if the click was on a piece
bool Checkers::clickPiece(sf::Vector2f mouseXY) {
    vector<Pawn>& pieces = getCurrPieces();
    int mouseX = toTile(mouseXY.x);
    int mouseY = toTile(mouseXY.y);
    for (Pawn& piece : pieces) {
        int pieceX = toTile(piece.getPiece().getPosition().x);
        int pieceY = toTile(piece.getPiece().getPosition().y);
        if (mouseX == pieceX && mouseY == pieceY) {
            clearHighlights();
            _tiles.at(pieceY).at(pieceX).setFillColor(pieceHighlight);
            showMoves(sf::Vector2f(pieceX, pieceY), piece);
            return true;
        }
    }
    return false;
}

// Highlight the possible tiles a current piece can move to
void Checkers::showMoves(sf::Vector2f pieceXY, Pawn& piece) {
    vector<Pawn> yourPieces = getCurrPieces();
    vector<Pawn> oppPieces = getOppPieces();
    vector<sf::Vector2f> moves;

    // Choose possible moveable directions
    if (isBlackTurn()) {
        if (piece.isKing()) {
            moves = {{1, 1}, {-1, 1}, {-1, -1}, {1, -1}};
        } else {
            moves = {{1, 1}, {-1, 1}};
        }
    } else {
        if (piece.isKing()) {
            moves = {{1, 1}, {-1, 1}, {-1, -1}, {1, -1}};
        } else {
            moves = {{-1, -1}, {1, -1}};
        }
    }

    // Highlight possible moves
    for (auto& disp : moves) {
        if (!onTile(pieceXY.x + disp.x, pieceXY.y + disp.y, yourPieces) &&
            !onTile(pieceXY.x + disp.x, pieceXY.y + disp.y, oppPieces) &&
            inBounds(pieceXY.x + disp.x) && inBounds(pieceXY.y + disp.y)) {
            _tiles.at(pieceXY.y + disp.y).at(pieceXY.x + disp.x).
    setFillColor(moveHighlight);
        } else if (onTile(pieceXY.x + disp.x, pieceXY.y + disp.y, oppPieces)
     &&
            !(onTile(pieceXY.x + 2 * disp.x, pieceXY.y + 2 * disp.y,
    oppPieces)) &&
            !(onTile(pieceXY.x + 2 * disp.x, pieceXY.y + 2 * disp.y,
    yourPieces)) &&
            inBounds(pieceXY.x + 2 * disp.x) && inBounds(pieceXY.y + 2 *
    disp.y)) {
            _tiles.at(pieceXY.y + 2 * disp.y).at(pieceXY.x + 2 * disp.x)
            .setFillColor(moveHighlight);
        }
```

```
224     }
225 }
226 
227 // Moves a players pawn if selected to a highlighted spot
228 void Checkers::makeMove(sf::Vector2f mouseXY) {
229     int tileX = toTile(mouseXY.x);
230     int tileY = toTile(mouseXY.y);
231     vector<Pawn>& pieces = getCurrPieces();
232     vector<Pawn>& oppPieces = getOppPieces();
233     if (_tiles.at(tileY).at(tileX).getFillColor() == moveHighlight) {
234         for (size_t h = 0; h < _tiles.size(); h++) {
235             for (size_t w = 0; w < _tiles.at(0).size(); w++) {
236                 if (_tiles.at(h).at(w).getFillColor() == pieceHighlight) {
237                     for (Pawn& piece : pieces) {
238                         int pieceX = toTile(piece.getPiece().getPosition().x
    );
239                         int pieceY = toTile(piece.getPiece().getPosition().y
    );
240                         if (static_cast<int>(w) == pieceX && static_cast<int
    >(h) == pieceY) {
241                             piece.movePiece(sf::Vector2f(tileX * TILE_LENGTH
     + BORDER_WIDTH,
242                                                          tileY * TILE_LENGTH
     + BORDER_WIDTH));
243                             removePieceJumped(oppPieces, w, h, tileX, tileY)
    ;
244                             checkKing();
245                             takeTurn();
246                             clearHighlights();
247                             return;
248                         }
249                     }
250                 }
251             }
252         }
253     }
254 }
255 
256 // Removes a opponents piece if a jump was detected
257 void Checkers::removePieceJumped(vector<Pawn>& pieces, int srcX, int srcY,
    int destX, int destY) {
258     int pieceX = srcX + (destX - srcX)/2;
259     int pieceY = srcY + (destY - srcY)/2;
260 
261     auto end = pieces.end();
262     pieces.erase(remove_if(pieces.begin(), end,
263     [pieceX, pieceY](Pawn& piece) {
264         return pieceX == (piece.getPiece().getPosition().x - BORDER_WIDTH) /
     TILE_LENGTH &&
265                pieceY == (piece.getPiece().getPosition().y - BORDER_WIDTH) /
     TILE_LENGTH;
266     }), end);
267 }
268 
269 // Overloaded draw function for Checkers class
270 void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) const
     {
271     for (size_t h = 0; h < _tiles.size(); h++) {
272         for (size_t w = 0; w < _tiles.at(0).size(); w++) {
```

```
            target.draw(_tiles.at(h).at(w), states);
        }
    }

    for (Pawn const& piece : _blackPieces) {
        target.draw(piece.getPiece(), states);
    }

    for (Pawn const& piece : _redPieces) {
        target.draw(piece.getPiece(), states);
    }
}
```

Checkers.hpp:

```
// Copyright 2023 William Susi

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

using std::vector;
using std::string;
using std::cout;
using std::endl;

#define TILE_LENGTH 64
#define BORDER_WIDTH 32
#define BOARD_SIZE 8

class Pawn {
 public:
  explicit Pawn(sf::Sprite piece);

  sf::Sprite getPiece() const;
  bool isKing() const;
  void movePiece(sf::Vector2f pieceXY);
  void makeKing(sf::Texture& texture);

 private:
  sf::Sprite _piece;
  bool _isKing;
};

class Checkers: public sf::Drawable {
 public:
  Checkers();

  bool isBlackTurn() const;
  void takeTurn();
  int toTile(double pixel);
  bool inBounds(int move);
  vector<Pawn>& getCurrPieces();
  vector<Pawn>& getOppPieces();

  void checkKing();
```

```
46    bool isWon();
47
48    void clearHighlights();
49    bool onTile(int tileX, int tileY, vector<Pawn>& pieces);
50    bool clickPiece(sf::Vector2f mouseXY);
51    void showMoves(sf::Vector2f pieceXY, Pawn& piece);
52
53
54    void makeMove(sf::Vector2f mouseXY);
55    void removePieceJumped(vector<Pawn>& pieces, int srcX, int srcY, int destX
        , int destY);
56
57   private:
58    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
        ;
59
60    vector<sf::Texture> _textures;
61    vector<vector<sf::RectangleShape>> _tiles;
62    vector<Pawn> _blackPieces;
63    vector<Pawn> _redPieces;
64    bool _blackTurn;
65  };
```

6 PS5: DNA Alignment

6.1 Overview

This project computes the optimal sequence alignment for two DNA strings. It aligns two DNA sequences using a least cost method.

6.2 End Product

Below is the output. This aligns two DNA strings with the cost of the total alignment at the top with the cost of each alignment as the last column.

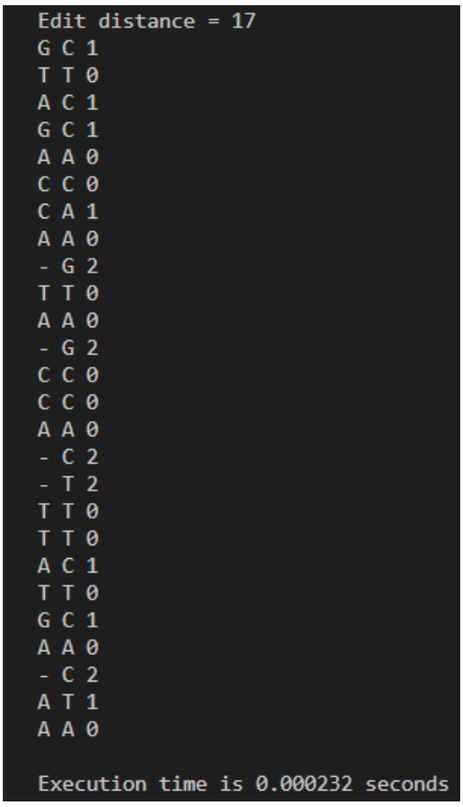

Figure 8: "stx26" DNA Alignment

6.3 What I Already Knew

I knew how to create a matrix of integers to store the cumulative penalties.

6.4 Design Decisions and Implementations

The project uses the Needleman-Wunsch method of dynamic programming. It was implemented through the EDistance class, which takes two strings of length N and M, and creates an NxM matrix of accumulated alignment penalties. To compute the value of each value in the matrix it was broken into sub-problems. The final row and final column can be computed from the lengths of the given strings. For filling everything else, we start at [N][M], and fill out the matrix backwards by filling each cell with the minimum of three values. These three values are the three possible cases: A letter of a string aligning with the same letter, a different letter, or a gap and finds the best comparison. Once the matrix is filled, the penalty of alignment can be found at [0][0]. The actual alignment can be traced backwards using a similar way we populated the matrix.

6.5 What I Learned

I learned what the Needleman-Wunsch dynamic programming method is and how it can be beneficial to reduce time complexity and already done computations.

6.6 Challenges

There were no major challenges.

6.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
DEPS = EDistance.hpp
OBJ = EDistance.o
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework

.PHONY: all clean lint

all: EDistance lint

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

EDistance: main.o $(OBJ)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o EDistance

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

main.cpp:

```
// Copyright 2023 William Susi

#include "EDistance.hpp"

int main(int argc, char** argv) {
    sf::Clock clock;
    sf::Time time;

    string s1, s2;
    cin >> s1;
    cin >> s2;

    EDistance dna(s1, s2);

    cout << "Edit distance = " << dna.optDistance() << endl;

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

    // Print elapsed time
    time = clock.getElapsedTime();
    cout << "Execution time is " << time.asSeconds() << " seconds \n";

    return 0;
}
```

Checkers.cpp:

```
// Copyright 2023 William Susi

#include "EDistance.hpp"

```

```
5  // EDistance constructor that accepts the two strings to be compared and
       allocates a NxM matrix.
6  EDistance::EDistance(string& s1, string& s2) : _s1(s1), _s2(s2) {
7      // Set up and fill matrix with placeholder value
8      for (size_t i = 0; i <= s1.size(); i++) {
9          _matrix.emplace_back(s2.size() + 1, -1);
10     }
11     // Fill in last row
12     int cost = 0;
13     for (auto i = _matrix.rbegin(); i != _matrix.rend(); i++) {
14         i->back() = cost;
15         cost += 2;
16     }
17     // Fill in last column
18     cost = 0;
19     for (auto i = _matrix.back().rbegin(); i != _matrix.back().rend(); i++)
       {
20         *i = cost;
21         cost += 2;
22     }
23 }
24
25 // Returns the penalty between two characters
26 int EDistance::penalty(char a, char b) {
27     return a != b;
28 }
29
30 // Returns the minimum of the three arguments.
31 int EDistance::min(int a, int b, int c) {
32     if (a <= b && a <= c) {
33         return a;
34     } else if (b <= a && b <= c)  {
35         return b;
36     }
37     return c;
38     // return std::min(a, std::min(b, c));
39 }
40
41 // Populates the matrix with the distance between two strings and returns
       the optimal distance.
42 int EDistance::optDistance() {
43     for (size_t i = _s1.size() - 1; i < _s1.size(); i--) {
44         for (size_t j = _s2.size() - 1; j < _s2.size(); j--) {
45             _matrix.at(i).at(j) = min(_matrix.at(i + 1).at(j + 1) + penalty(
       _s1[i], _s2[j]),
46                                       _matrix.at(i).at(j + 1) + 2,
47                                            _matrix.at(i + 1).at(j) + 2);
48         }
49     }
50
51     return _matrix.at(0).at(0);
52 }
53
54 // Traces the matrix to the optimal distance and returns that alignment.
55 string EDistance::alignment() {
56     string path = "";
57     size_t i = 0, j = 0;
58
59     while (i < _s1.size() && j < _s2.size()) {
```

```
60          if (_matrix.at(i).at(j) == _matrix.at(i + 1).at(j) + 2) {  // Gap in
    s2
61              path = path + _s1[i] + " - 2\n";
62              i++;
63          } else if (_matrix.at(i).at(j) == _matrix.at(i).at(j + 1) + 2) {  //
    Gap in s1
64              path = path + "- " + _s2[j] + " 2\n";
65              j++;
66          } else if (_matrix.at(i).at(j) == _matrix.at(i + 1).at(j + 1)) {  //
    If equal
67              path = path + _s1[i] + " " + _s2[j] + " 0\n";
68              i++;
69              j++;
70          } else if (_matrix.at(i).at(j) == _matrix.at(i + 1).at(j + 1) + 1) {
     // Else
71              path = path + _s1[i] + " " + _s2[j] + " 1\n";
72              i++;
73              j++;
74          }
75      }
76      return path;
77  }
```

Checkers.hpp:

```
1   // Copyright 2023 William Susi
2
3   #include <iostream>
4   #include <utility>
5   #include <fstream>
6   #include <algorithm>
7   #include <string>
8   #include <vector>
9   #include <iomanip>
10
11  #include <SFML/System.hpp>
12  #include <SFML/Window.hpp>
13  #include <SFML/Graphics.hpp>
14  #include <SFML/Audio.hpp>
15
16  using std::cin;
17  using std::cout;
18  using std::endl;
19  using std::string;
20  using std::vector;
21
22  class EDistance {
23   public:
24      EDistance(string& s1, string& s2);
25
26      static int penalty(char a, char b);
27      static int min(int a, int b, int c);
28
29      int optDistance();
30      string alignment();
31
32   private:
33      string _s1;
34      string _s2;
35      vector<vector<int>> _matrix;
```

```
36 };
```

7 ps6: RandWriter

7.1 Overview

This project uses a given text and generates new random text based off of it. It takes the text, a kgram length, and a new text length. A kgram is a substring of the text of the given length. Using these many many kgrams, and their occurrences, new text of any length can be generating using the past kgrams probabilities.

7.2 End Product

Below is the output. This is a string of random text generated from the text provided by Tom Sawyer. The kgram length was set to 7 and 150 characters were generated.

> CHAPTER XXIIIAT last time, but it don't pull it out:"Tom.""Oh, I shoulders threw themselves hanging'll come before."A trifle out of the predestinely.

7.3 What I Already Knew

I knew how to parse through text in C++..

7.4 Design Decisions and Implementations

This project created a RandWriter class which was used to hold the kgrams and k+1grams generated from a text. I chose to store the kgrams and k+1grams separately as unordered maps, where the key was a string representing the kgram, and the value was an int representing the number of times the kgram occurred. The constructor parsed through the text and filled the two maps, while also generating a seed that would be used later for RNG. As for generating new text it goes as follows:

- 'generate' - This function returns a string of new text given a starting kgram and length of generation. It loops for the desired length of generation, calls 'kRand' to get a pseudo-random character, and returns a string once all those characters have been generated.
- 'kRand' - This function returns a character based off a given kgram. The kgram is first checked to actually exist in the kgram map using the 'freq' function, and if it is, it finds and stores all its associated k+1grams. It then counts the number of total occurrences of all k+1grams with kgram in them so that a k+1gram with more occurrences has a higher probability of being picked. A number is then randomly generated based off the RandWriter seed and a distribution is applied to it so that the random number is between 1 and the total occurrences. That random number is then used to pick one of the k+1grams. That character is then taken and returned as the new random character.
- 'freq' - Determines if a given kgram or a k+1gram can be found in their respective maps. The overload for a k+1gram wasn't implemented because of how the internal structure of my RandWriter was made (two separate maps).

7.5 What I Learned

I learned what kgrams are. I learned C++'s version of random number generation and how to implement that. I learned more about maps and how to implement them.

7.6 Challenges

There were small challenges along the way, but nothing that became a major issue.

7.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
DEPS = RandWriter.hpp
OBJ = RandWriter.o
LIB = -lboost_unit_test_framework

.PHONY: all clean lint

all: TextWriter test lint

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

TextWriter: main.o $(OBJ)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

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

clean:
    rm *.o TextWriter test

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

main.cpp:

```
// Copyright 2023 William Susi

#include "RandWriter.hpp"

int main(int argc, char** argv) {
    int kGramLen = std::stod(argv[1]);
    int numChar = std::stod(argv[2]);

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

    RandWriter randText(text, kGramLen);

    cout << randText.generate(text.substr(0, randText.orderK()), numChar) <<
    endl;

    return 0;
}
```

Checkers.cpp:

```
// Copyright 2023 William Susi

#include "RandWriter.hpp"

RandWriter::RandWriter(string text, int k) : _orderK(k), _alphabet("
   abcdefghijklmnopqrstuvwxyz") {
```

```
6       text += text.substr(0, _orderK);
7       for (size_t i = 0; i < text.size() - _orderK; i++) {
8           string kgram1 = text.substr(i, _orderK);
9           string kgram2 = text.substr(i, _orderK + 1);
10          _kGrams[kgram1]++;
11          _kPlusGrams[kgram2]++;
12      }
13
14      unsigned int seed = std::chrono::system_clock::now().time_since_epoch().
        count();
15      std::minstd_rand0 num(seed);
16      _randNum = num;
17  }
18
19  // Returns the order
20  int RandWriter::orderK() const {
21      return _orderK;
22  }
23
24  // Checks if a kgram matches the order size
25  void RandWriter::isRightSize(string kgram) const {
26      if (static_cast<int>(kgram.size()) != _orderK) {
27          throw std::runtime_error("kgram is not the correct length");
28      }
29  }
30
31  // Finds the number of occurrences of a given kgram
32  int RandWriter::freq(string kgram) const {
33      isRightSize(kgram);
34      auto gram = _kGrams.find(kgram);
35      if (gram == _kGrams.end()) {
36          return 0;
37      } else {
38          return gram->second;
39      }
40  }
41
42  // Finds the number of occurrences of a given k+1gram
43  int RandWriter::freq(string kgram, char c) const {
44      isRightSize(kgram);
45      string newGram = kgram + c;
46      auto gram = _kPlusGrams.find(newGram);
47      if (gram == _kPlusGrams.end()) {
48          return 0;
49      } else {
50          return gram->second;
51      }
52  }
53
54  // Generates a pseudo-random character based on the occurrences
55  // of given k+1grams in the RandWriter text.
56  char RandWriter::kRand(string kgram) {
57      if (freq(kgram) == 0) {
58          throw std::runtime_error("kgram: '" + kgram + "' does not exist");
59      }
60
61      // Create map of k+1grams that start with kgram
62      unordered_map<string, int> tempGrams;
63      for (auto kp1gram : _kPlusGrams) {
```

```
64          auto found = (kp1gram.first).find(kgram);
65          if (found == 0) {
66              tempGrams.insert(kp1gram);
67          }
68      }
69
70      // Count the number of total occurrences of k+1grams with kgram in them
71      int totalOccurrences = 0;
72      std::for_each(tempGrams.begin(), tempGrams.end(),
73          [&totalOccurrences](auto& key) {
74              totalOccurrences += key.second;
75      });
76
77      // Generate a random number between 0 and the number of total
        occurrences
78      std::uniform_int_distribution<int> dist(1, totalOccurrences);
79      int ranNum = dist(_randNum);
80
81      // Use that random number to pick a random k+1gram adn grab its last
        letter
82      int currSum = 0;
83      for (auto key : tempGrams) {
84          currSum += key.second;
85          if (currSum >= ranNum) {
86              return key.first[_orderK];
87          }
88      }
89
90      throw std::runtime_error("Could not generate new char");
91  }
92
93  // Generates a string of length L given a starting kgram based of the
        RandWriter text.
94  string RandWriter::generate(string kgram, int L) {
95      isRightSize(kgram);
96      string randomText = kgram;
97      string gram = kgram;
98      for (int i = 0; i < L - _orderK; i++) {
99          char c = kRand(gram);
100         randomText += c;
101         if (gram.size() != 0) {
102             gram = gram.substr(1, _orderK - 1) + c;
103         }
104     }
105     return randomText;
106 }
107
108 // Displays the internal state of a RandWriter object by printing out the
        order, alphabet,
109 // and the frequencies of the k-grams and k+1-grams.
110 ostream& operator<<(ostream& outStream, const RandWriter& randWriter) {
111     outStream << "Order: " << randWriter._orderK << endl;
112
113     outStream << "Alphabet: ";
114     for (char letter : randWriter._alphabet) {
115         outStream << letter << " ";
116     }
117
118     outStream << endl << "k-grams:" << endl;
```

```
    for (auto kgram : randWriter._kGrams) {
        outStream << " - " << kgram.first << ": " << kgram.second << endl;
    }

    outStream << endl << "k+1-grams:" << endl;
    for (auto kgram : randWriter._kPlusGrams) {
        outStream << " - " << kgram.first << ": " << kgram.second << endl;
    }

    return outStream;
}
```

Checkers.hpp:

```
// Copyright 2023 William Susi

#include <iostream>
#include <fstream>
#include <algorithm>
#include <exception>
#include <string>
#include <chrono>
#include <random>
#include <unordered_map>

using std::cin;
using std::cout;
using std::endl;
using std::ostream;
using std::string;
using std::unordered_map;

class RandWriter {
 public:
    RandWriter(string text, int k);

    int orderK() const;

    void isRightSize(string kgram) const;

    int freq(string kgram) const;
    int freq(string kgram, char c) const;
    char kRand(string kgram);
    string generate(string kgram, int L);

    friend ostream& operator<<(ostream& outStream, const RandWriter&
   randWriter);
 private:
    int _orderK;
    unordered_map<string, int> _kGrams;
    unordered_map<string, int> _kPlusGrams;
    string _alphabet;
    std::minstd_rand0 _randNum;
};
```

8 ps7: Kronos Log Parsing

8.1 Overview

This project parses a log file using regular expressions and outputs a report file.

8.2 End Product

Below is the output. This is the end portion of a report file. It shows the final started and completed boots of a log file.

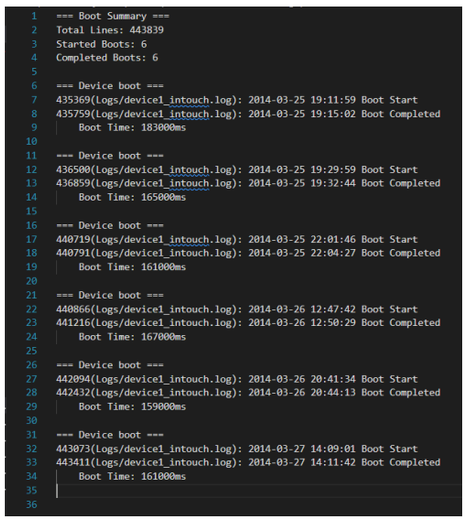

Figure 9: Device 1 Boot Report

8.3 What I Already Knew

I knew how to parse through an input file in C++.

8.4 Design Decisions and Implementations

There wasn't a need for a class or an outside file so everything was done in main. First two regex expressions were made to properly detect start and complete boots. Each line from the log file was checked against these expressions. When a boot started it was written to the report file with a date and time. If a boot had already started, and another boot was encountered before a complete boot, an incomplete boot was written to the report file following the previous boot. The new boot was then written to the file with a date and time. If a completion was encountered following a boot it was written to the file with a date and time. The boot time between the start and completion was computed via the 'convertTime' function and was also added to the report. This function took two date/time strings, made them ptime objects, subtracted them, and converted the computed time to milliseconds. Once the whole log file was parsed, a summary was added to the beginning of the report indicating the total lines read, the total of boots started, and the total of completed boots.

8.5 What I Learned

I learned what regular expression were and how useful they are for parsing important information.

8.6 Challenges

I had a little trouble at the start of where to start because it felt a bit daunting, but once I started it was pretty simple. I also had some general trouble with the regex and time/date libraries because the API for the boost library is hard to follow, so some functions used from them took a second to get right.

8.7 Codebase

Makefile:

```
CC = g++
CFLAGS = --std=c++17 -Wall -Werror -pedantic -g
DEPS = main.hpp
OBJ = main.o
LIB = -lboost_unit_test_framework -lboost_date_time

.PHONY: all clean lint

all: ps7 lint

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

ps7: main.o $(OBJ)
    $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
    rm *.o ps7

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

main.cpp:

```
// Copyright 2023 William Susi

#include "main.hpp"

int main(int argc, char** argv) {
    // Regexs for start and completion of log
    regex start{R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}): \(log\.c\.\d+\) server started )"};
    regex completion{R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})+\.\d{3}:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:\d{4})"};

    // Load log file
    std::fstream log;
    string fileName = argv[1];
    log.open(fileName, std::fstream::in);

    // Load report file
    std::fstream report;
    report.open("Reports/" + fileName.substr(5, fileName.size()-4) + ".rpt", std::fstream::out);

    // Create boot stream
    std::stringstream boots;

    // Get line from log
    int lineNum = 1, started = 0, completed = 0;
    bool isBegun = false;
    string line, startTime, endTime;
    while (getline(log, line)) {
        std::match_results<string::const_iterator> match;
        if (regex_match(line, match, start)) {  // Check if a start boot is found
```

```
            if (isBegun) {  // Check if a boot already started
                boots << "**** Incomplete boot **** " << endl << endl;
            }
            startTime = match[1];
            boots << "=== Device boot ===" << endl;
            boots << lineNum << "(" << fileName << "): " << startTime << "
   Boot Start" << endl;
            isBegun = true;
            started++;
        } else if (regex_match(line, match, completion)) {  // Check if boot
    completed
            endTime = match[1];
            boots << lineNum << "(" << fileName << "): " << endTime << "
   Boot Completed" << endl;
            boots << "\tBoot Time: " << convertTime(startTime, endTime) << "
   ms" << endl << endl;
            isBegun = false;
            completed++;
        }

        lineNum++;
    }

    // Write Summary
    report << "=== Boot Summary ===" << endl;
    report << "Total Lines: " << lineNum << endl;
    report << "Started Boots: " << started << endl;
    report << "Completed Boots: " << completed << endl << endl;

    // Add boots to report
    report << boots.rdbuf();

    // Close files
    log.close();
    report.close();

    return 0;
}

// Function to convert the difference between two date/times into
   milliseconds
boost::date_time::time_resolution_traits_adapted64_impl::int_type
convertTime(string start, string end) {
    return (time_from_string(end) - time_from_string(start)).
   total_milliseconds();
}
```

Checkers.cpp:

```
// Copyright 2023 William Susi

#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <regex>
#include "boost/date_time/posix_time/posix_time.hpp"

using std::cin;
using std::cout;
```

```
12 using std::endl;
13 using std::ostream;
14 using std::string;
15 using std::regex;
16 using std::regex_match;
17 using boost::posix_time::time_from_string;
18
19 boost::date_time::time_resolution_traits_adapted64_impl::int_type
20 convertTime(string start, string end);
```