Computing IV Sec 201: Project Portfolio

James Walsh

Spring 2023

Contents

1	PS1 :	Linear Feedback Shift Register and Image Encoding	3
2	PS2 :	Sokoban	9
3	PS3:	Pythagorean Tree	21
4	PS4 :	Checkers	2 5
5	PS5 :	DNA Alignment	39
6	PS6 :	$\operatorname{RandWriter}$	44
7	PS7 :	Kronos Log Parsing	51

0.1 Discussion

What I accomplished

In this project, I created a Sprite Object that is a class from the Simple and Fast Multimedia Library (SFML). The Sprite has a number of features. One feature is allowing the Sprite to move across the display window in response to keystrokes. Also, it can increase or decrease in size. Lastly, the sprite has been given an image to represent the object on the display window.

Design and Features

For my project, I decided to use the arrow keys to move the sprite on the screen. To do this I used a switch statement that responded to event.type.code which is a keystroke. If any arrow (up, down, right, left) was pressed, the Sprite moved 10 units in the corresponding direction. For this assignment, I had to come up with a feature to implement on my own. The feature I decided to add was increasing and decreasing the size of the Sprite. I approached the implementation the same way as before, with a switch statement responding to keystrokes. If the 'I' key is pressed the image will increase by a scale of 2 units by using the Sprite's setScale() function. Similarly, if the 'D' key is pressed the image will decrease by a scale of .2 units by using the same function as before. The reason I used switch statements to implement the features of the Sprite was for simplicity. Instead of having multiple if statements to check the key pressed, having one switch statement with multiple cases makes the codes' functionality and appearance much simpler.

What I already knew and what I learned

Going into this project I had a good understanding of what classes are and how to use them. However, I have never used or heard of the SFML library, so getting comfortable with its classes was a learning curve.

When completing this assignment I learned how to use the simple objects/classes in the SFML library. Specifically Shapes, Colors, Sprites, Textures, Events, and RenderWindows. Each of these objects have a number of member functions that come along with them. I was able to get a good understanding of how each work by reading the documentation found at sfml-dev.org.

1 PS1: Linear Feedback Shift Register and Image Encoding

1.1 Discussion

What I accomplished

For this project, I created a Linear Feedback Shift Register (LFSR) class. My implementation of the LFSR, given a given bit register, shifts each bit one space to the left and replaces the vacated bit (the far right bit) with a bit computed by the Boolean exclusive-or function (XOR). The function computes the replacement bit by XOR'ing 3 tap bits and the bit cut-off at the far left of the register after the shift. In this project, the tap bits used were 10, 12, and 13.

In this project, I also created an algorithm that encrypts and decrypts images by using a linear feedback shift register. The Algorithm is described below in the \mathbf{D} esign and Features section.

Design and Features

When implementing the LFSR I used a std::vector of characters to represent the register of bits. I found implementing the register as a vector was a good representation because adding to vector was simple by using push back(). The std::vector class also has random access memory, which makes implementation easy. Within the class, I created two member functions, step() and generate(). The step() function is a void function that takes no parameters, and simulates one LFSR shift, and updates the bit register accordingly. The generate() function simulates takes an integer k as a parameter, and simulates k steps of the register and returns a k-bit integer that the bit-register represents.

I used the LFSR to encrypt and decrypt images by extracting each pixel in the image's red, green, and blue components. For each pixel, the integer value that represents the red/blue/green components are XOR'ed with a newly-generated 8-bit integer that is generated by the LFSR's generate() function. Lastly, the resulting integer from each component (r/b/g) is used to create a new color which is saved to the current pixel. This same process can be used for encrypting and decrypting an image. However, to decrypt an image the original bit-register must be identical to the original bit-register used to encrypt the image. This algorithm is implemented in the transform() function which can be found in the photoMagic.cpp file.

What I already knew and what I learned

Before completing this assignment I knew what I knew what I bit-register was and I was also familiar with the Boolean exclusive-or function. However, I have never seen a programming implementation of an LFSR.

When writing the program I learned how to use what I already knew and apply it to the LFSR object. Once I understood all aspects of the assignment, the implementation was not difficult. When using my newly created LFSR object to encypt and decrypt an image there was definitely a learning curve. I was not familiar with image encryption, but I quickly realized how I could use the generate() function to XOR the result with each rbg integer to create a new color to use in the encrypted image.

Unit Testing

For this program, I wrote six different unit tests using the Boost Unit Test Framework. All test created a FibLFSR with "1011011000110110" as the given register. I created two tests using BOOST_REQUIRE_EQUAL() testing if the step() and generate() function returns the propper integers. Then I used BOOST_REQUIRE_NO_THROW() to test is any for loops go out of bounds. Lastly, I created a test BOOST_CHECK_GE and BOOST_CHECK_LE to test if generate(8) returns a integer between 0 and 255 inclusively. The reason I created this test is to ensure the return value would be a valid rbg color. My program passed all tests.

Below is a screenshot of the Original Image. See Figure 1. Along with a screenshot of the Encoded Image. See Figure 2.



Figure 1: Original/Decoded Image.



Figure 2: Encoded Image.

1.2 Codebase

```
CC = g++
 1
   CFLAGS = -Wall -Werror -pedantic -std=c++17 -g
 2
3
   LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
       lboost_unit_test_framework
   DEPS = FibLFSR.hpp
 4
   OBJS = FibLFSR.o
 5
6
7
   %.o: %.cpp $(DEPS)
8
     $(CC) $(CFLAGS) -c $<
9
10
   .PHONY: all clean
11
12
   all: PhotoMagic test
13
14
   PhotoMagic: PhotoMagic.o $(OBJS)
     $(CC) $(CFLAGS) -o $0 $^ $(LIBS)
15
16
   test: $(OBJS) test.o
17
     $(CC) $(FLAGS) -o test $^ $(LIBS)
18
19
20
   clean:
21
     rm *.o PhotoMagic test
```

```
1
   #include "FibLFSR.hpp"
2
   #include <iostream>
   #include <SFML/System.hpp>
3
   #include <SFML/Window.hpp>
4
   #include <SFML/Graphics.hpp>
5
6
7
   void transform(sf::Image&, FibLFSR*);
8
9
   int main(int argc, char* argv[])
10
     char* inputFile = argv[1];
11
     char* outputFile = argv[2];
12
```

```
13
     char* LFSRseed = argv[3];
14
15
     sf::Image image;
     if (!image.loadFromFile(inputFile))
16
17
       return -1;
18
     FibLFSR myLFSR(LFSRseed);
19
20
     transform(image, &myLFSR);
21
22
     sf::Vector2u size = image.getSize();
23
     sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Image");
24
     sf::Texture texture;
25
26
     texture.loadFromImage(image);
27
28
     sf::Sprite sprite;
29
     sprite.setTexture(texture);
30
31
     while (window.isOpen())
32
33
        sf::Event event;
34
        while (window.pollEvent(event))
35
36
          if (event.type == sf::Event::Closed)
37
            window.close();
38
        }
39
40
       window.clear(sf::Color::White);
41
        window.draw(sprite);
42
        window.display();
43
     }
44
     // fredm: saving a PNG segfaults for me, though it does properly
45
46
     // write the file
     if (!image.saveToFile(outputFile))
47
48
       return -1;
49
50
     return 0;
   }
51
52
53
   // Transforms image using FibLFSR
   void transform(sf::Image& aImage, FibLFSR* aFibLFSR) {
54
     sf::Vector2u size = aImage.getSize();
55
56
     sf::Color p;
     unsigned int x;
57
     unsigned int y;
58
     int newInt;
59
60
61
     for (x = 0; x < size.x; x++) {
62
       for (y = 0; y < size.y; y++) {
63
         p = aImage.getPixel(x, y);
64
         newInt = aFibLFSR->generate(8);
         p.r = p.r ^ newInt;
65
66
         newInt = aFibLFSR->generate(8);
67
         p.g = p.g ^ newInt;
68
         newInt = aFibLFSR->generate(8);
69
         p.b = p.b ^ newInt;
70
         aImage.setPixel(x, y, p);
71
```

```
72
73
74
75 // Display an encrypted copy of the picture, using LFSR to do the encryption
   #pragma once
 1
 3
   #include <string>
 4
   #include <vector>
 5
 6
   class FibLFSR{
 7
   public:
 8
        // Constructor to create LFSR with the given initial seed
 9
       FibLFSR(std::string seed);
10
11
        // Simulate one step and return the new bit 0 or 1
12
       int step();
13
        // Simulate k steps and return a k-bit integer
14
        int generate(int k);
15
16
       friend std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr);
17
18
   private:
19
        // Representation of the seed as an integer
20
        std::vector<char> seedVector;
21
        // Taps
22
        int t1, t2, t3;
23 | };
24 | std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr);
 1 // Copyright 2023 James Walsh
 2 | #include "FibLFSR.hpp"
   #include <iostream>
 3
   #include <vector>
 4
   #include <string>
 5
 6
   #include <cmath>
 7
 8
   int const LENGTH_OF_SEED = 16;
 9
10
   FibLFSR::FibLFSR(std::string seed) {
11
       // convert the seed to a vector of chars
12
       int i;
13
14
        for (i = 0; i < LENGTH_OF_SEED; i++) {</pre>
15
            seedVector.push_back(seed[i]);
16
       }
17
18
        // init taps indexies
19
       t1 = 2;
20
       t2 = 3;
21
        t3 = 5;
22
   }
23
24
   int FibLFSR::step() {
25
       int newBit;
26
       int i;
27
28
        // XOR the last bit and tap bits
       newBit = seedVector[0] ^ seedVector[t3];
29
       newBit = newBit ^ seedVector[t2];
30
```

```
32
33
        for (i = 0; i < LENGTH_OF_SEED - 1; i++) {</pre>
            seedVector[i] = seedVector[i + 1];
34
35
        if (newBit == 1) {
36
37
            seedVector[LENGTH_OF_SEED - 1] = '1';
38
        } else {
39
            seedVector[LENGTH_OF_SEED - 1] = '0';
40
        }
41
42
       return newBit;
   }
43
44
    int FibLFSR::generate(int k) {
45
46
        int currentStep = 0;
        int kBit = 0;
47
48
        int i;
49
        while (currentStep < k) {</pre>
50
51
            this->step();
52
            currentStep++;
53
        }
54
        for (i = LENGTH_OF_SEED - 1; i > (LENGTH_OF_SEED - k - 1); i--) {
            if (seedVector[i] == '1') {
55
                kBit += pow(2, ((LENGTH_OF_SEED - 1) - i));
56
57
58
        }
        return kBit;
59
60
   }
61
62
   std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {</pre>
63
        int i;
64
        for (i = 0; i < LENGTH_OF_SEED; i++) {</pre>
65
            out << lfsr.seedVector[i];</pre>
66
67
        }
68
69
       return out;
   }
70
   #include <iostream>
 1
   #include <string>
 2
 3
   #include "FibLFSR.hpp"
 4
 5
   #define BOOST_TEST_DYN_LINK
 6
 7
   #define BOOST_TEST_MODULE Main
 8
   #include <boost/test/unit_test.hpp>
 9
   BOOST_AUTO_TEST_CASE(testStepInstr1) {
10
      FibLFSR 1("1011011000110110");
11
      BOOST_REQUIRE_EQUAL(1.step(), 0);
12
13
      BOOST_REQUIRE_EQUAL(1.step(), 0);
      BOOST_REQUIRE_EQUAL(1.step(), 0);
14
      BOOST_REQUIRE_EQUAL(1.step(), 1);
15
16
      BOOST_REQUIRE_EQUAL(1.step(), 1);
17
      BOOST_REQUIRE_EQUAL(1.step(), 0);
18
      BOOST_REQUIRE_EQUAL(1.step(), 0);
      BOOST_REQUIRE_EQUAL(1.step(), 1);
19
```

31

newBit = newBit ^ seedVector[t1];

```
20 |}
21
22
   BOOST_AUTO_TEST_CASE(testStepInstr2) {
     FibLFSR 12("1011011000110110");
23
24
     BOOST_REQUIRE_EQUAL(12.generate(9), 51);
25 }
26
27
   BOOST_AUTO_TEST_CASE(testConstructorOutOfBounds) {
28
     FibLFSR 13("1011011000110110");
29
     BOOST_REQUIRE_NO_THROW(13.step());
   }
30
31
32 BOOST_AUTO_TEST_CASE(testStepOutOfBounds) {
     FibLFSR 14("1011011000110110");
33
34
     BOOST_REQUIRE_NO_THROW(14.step());
35 | }
36
37
   BOOST_AUTO_TEST_CASE(testGenerateOutOfBounds) {
     FibLFSR 15("1011011000110110");
38
     BOOST_REQUIRE_NO_THROW(15.step());
39
40
41
42
   BOOST_AUTO_TEST_CASE(testTransformAlgorithm) {
43
      FibLFSR 16("1011011000110110");
44
     int eightBitNum = 16.generate(8);
45
     BOOST_CHECK_GE(eightBitNum, 0);
     BOOST_CHECK_LE(eightBitNum, 255);
46
47
```

2 PS2: Sokoban

2.1 Discussion

What I accomplished

This program creates a fully functional Sokoban Game. Sokoban is a tile-based video game where the player controls a warehouse worker. The goal of the game is to push boxes into designated storage locations. The program uses a level file, which is a text file consisting of a width and height as well as a series of symbols ('.' - empty space, '#' - wall, 'a' - empty storage space, 'A' - box, '@' -player) that represent a level for the Sokoban game. The program then creates a Sokoban object and sets up the game. It establishes a width and height for the display window and stores the series of symbols in a std::vector of characters which represents a Matrix. Each matrix element (x, y) corresponds to a coordinate on the display window where an image block will be drawn. After the Sokoban object is initialized, the program draws the level to the display window, it does this by iterating through the matrix. The image corresponding to the current element is drawn to the screen for the entire matrix. After this is complete, the Sokoban level is properly displayed using a SFML render window.

Once the UI is set up, the user can use the WASD keys to move the player up, left, down, and up to play the game. The player is allowed to move in any direction as long as the player is not moving into a wall, the end of the window, or a moveable box. A box is unmovable if the space where the player is trying to move it has another box, if there is a wall, or if the box is already in a storage location. Once the Player has moved all the boxes into storage, or if all the storages contain boxes, you win! Also, the user can use the 'R' key to reset the game at any time.

Design and Features

To program the game, I created a Sokoban class that is inherited from SFML's Drawable class. The class consists of two constructors, a default constructor that prompts the user to enter a level and then uses the extraction to initialize the object, a value constructor that takes a character array that represents a level file, and a movePlayer() function. The class also has accessor and mutator functions for each of its private member variables, besides a mutator function for the GRID SCALE variable because it is const. The class has a private draw function which is a virtual function inherited from the sf::Drawable class. The draw function iterates through the matrix (vector of characters) and draws the corresponding image to the display window. Along with these member functions, the class contains a list of private member variables. A virtual draw function which is overridden from the Drawable class, height and width variables that store the height/width of the game, an isStorage boolean that is used to keep track if the user has entered a storage location, a const GRID_SCALE that is set to 64 because each tile image is 64 pixels, a SFML Texture player that is used for the current player image being displayed, two counter variables moveable_boxes and empty_boxes, and lastly the gameGrid. I chose to implement the game grid as an std::vector of characters because using I found using std iterators is useful if, for some reason, I changed the type of the game grid to a list the iterators would carry over somewhat easily. The Sokoban class also overloads the insertion and extraction operators. The extraction operator initializes the Sokoban object using the level file, and the insertion operator outputs the level to the terminal in the format given in the level file.

The movePlayer function takes an Enumerated type Movement (W, A, S, D), and the function checks if the move is possible. It does this by calling a series of helper functions that check if the movement is in range, if there is a box in the way, and if so can the box be moved. If the movement is possible, the movePlayer function updates the gameGrid vector. Once the player has moved all the boxes into storage, or if all the storages contain boxes, you win, and a message is displayed. It does this by calling a isWon function which checks if the player has won. If so, it returns true. Main uses this to display the message and pause the game until it is reset.

What I already knew and what I learned

Going into this project, I was not familiar with making any sort of video game, and I have not had much experience with visual display windows either. However, I was used to working with vectors and how to manipulate the elements such that they represent a 2D-Array, so that is why I decided to implement the game grid as a vector of characters.

When completing this project, I learned a lot about how implementing a low-level video game works. Going into this project, I did not realize how intricate a single move can be. I learned how to being aware of every how every decision you make in your code is crutial. This project improved my advanced programming design skills drastically. As the project went on, I got better and better at implementing conditionals and often found myself going back to previous functions and reworking some of the conditionals because I had found a better/simpler algorithm. Lastly, I learned a lot about how delegating work to other functions can make code a lot cleaner and more understandable.

Below is a screenshot of a Completed Level: Figure 3.

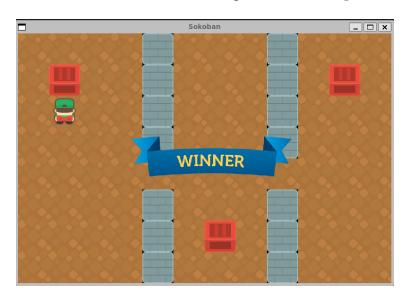


Figure 3: Game Display.

2.2 Codebase

```
1
   CC = g++
 2
   {\tt CFLAGS = -Wall - Werror - pedantic - std = c + +17 - g}
 3
   LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
   DEPS = Sokoban.hpp
 4
 5
   OBJS = Sokoban.o
6
 7
   %.o: %.cpp $(DEPS)
8
      $(CC) $(CFLAGS) -c $<
9
10
    .PHONY: all clean lint
11
12
   all: Sokoban
13
14
   Sokoban: main.o $(OBJS)
      $(CC) $(CFLAGS) -o $0 $^ $(LIBS)
15
16
17
   lint:
18
      cpplint *.cpp *.hpp
19
20
   clean:
21
      rm *.o Sokoban
```

```
1 // Copyright 2023 James Walsh
2 #include <iostream>
```

```
3 | #include "Sokoban.hpp"
 4
   #include <SFML/System.hpp>
   #include <SFML/Window.hpp>
 5
   #include <SFML/Graphics.hpp>
 7
 8
   int main(int argc, char* argv[]) {
 9
        char* level = argv[1];
10
        Sokoban newGame(level);
11
12
        sf::RenderWindow window(sf::VideoMode(
13
                newGame.get_height() * newGame.get_scale(),
14
                newGame.get_width() * newGame.get_scale()), "Sokoban");
15
16
        sf::Texture winnerImage;
        winnerImage.loadFromFile("winner.png");
17
18
        sf::Sprite won(winnerImage);
19
        won.scale(.7, .7);
20
        won.setPosition((newGame.get_height() * newGame.get_scale()) / 3.5,
21
             (newGame.get_width() * newGame.get_scale())/ 7);
22
23
        bool end = false;
24
        while (window.isOpen()) {
25
            sf::Event event;
26
            while (window.pollEvent(event)) {
27
                if (event.type == sf::Event::Closed) {
28
                    window.close();
                }
29
30
                if (event.type == sf::Event::KeyPressed) {
                    if (!end) {
31
32
                        switch (event.key.code) {
33
                             case sf::Keyboard::W:
                                                      // move player up
34
                                 newGame.movePlayer(UP); break;
                                                    // move player left
35
                             case sf::Keyboard::A:
36
                                 newGame.movePlayer(LEFT); break;
37
                             case sf::Keyboard::S:
                                                     // move player down
38
                                 newGame.movePlayer(DOWN); break;
39
                             case sf::Keyboard::D:
                                                     // move player right
40
                                 newGame.movePlayer(RIGHT); break;
41
                            default: break;
                        }
42
                    }
43
44
                    if (event.key.code == sf::Keyboard::R) {
45
                            newGame.reset_level(level);
46
                             end = false;
47
                             break:
48
                    }
                }
49
50
                window.clear();
51
                window.draw(newGame);
52
                if (newGame.isWon()) {
53
                    end = true;
54
                    window.draw(won);
55
                    window.display();
56
57
                window.display();
58
            }
59
        }
60
61
        return 0;
```

```
// Copyright 2023 James Walsh
 1
 2 | #pragma once
   #include <iostream>
 4 #include <vector>
   #include <map>
 5
   #include <SFML/System.hpp>
 6
 7
   #include <SFML/Window.hpp>
 8
   #include <SFML/Graphics.hpp>
 9
10
   enum movement { UP, LEFT, DOWN, RIGHT };
   typedef enum movement Movement;
11
12
13
   class Sokoban : public sf::Drawable {
14
   public:
15
     Sokoban();
16
     explicit Sokoban(char* iLevelFile);
17
     void movePlayer(const Movement m);
18
19
     bool isWon() const;
20
     bool check_and_set_box(size_t x, size_t y, Movement m);
21
     void reset_level(char* iLevelFile);
22
     bool within_bounds(size_t x, size_t y) const;
23
24
     void set_width(size_t w) { width = w; }
25
     void set_height(size_t h) { height = h; }
26
     void set_gameGrid(size_t size) { gameGrid = std::vector<char>(size, ' ');
      }
27
     size_t get_width() const { return width; }
28
     size_t get_height() const { return height; }
29
     std::vector<char> get_gameGrid() const { return gameGrid; }
     int get_scale() const { return GRID_SCALE; }
30
31
     void set_player_pos(size_t x, size_t y, Movement m);
     void set_movable_boxes(size_t increment) { movable_boxes += increment; }
32
33
     void set_empty_locations(size_t increment) { empty_locations += increment;
        }
34
    private:
35
     void draw(sf::RenderTarget& target, sf::RenderStates states) const
36
      override;
37
     size_t width;
     size_t height;
38
39
     std::vector<char> gameGrid;
40
     bool isStorage = false;
     const int GRID_SCALE = 64;
41
42
     sf::Texture player;
43
     size_t movable_boxes = 0;
44
     size_t empty_locations = 0;
45
   };
46
47
   |std::istream& operator>>(std::istream& inStream, Sokoban& object);
48 | std::ostream& operator<<(std::ostream& outStream, const Sokoban& object);
   // Copyright 2023 James Walsh
 1
   #include "Sokoban.hpp"
 2
 3
   #include <iostream>
 4 #include <fstream>
 5 | #include <string>
 6 | #include <algorithm>
```

62 |}

```
7
 8
   Sokoban::Sokoban() {
 9
        std::cout << "Enter a level to play: ";</pre>
10
        std::cin >> *this;
11
        std::cout << *this << std::endl;</pre>
        player.loadFromFile("player_05.png");
12
   }
13
14
15
   Sokoban::Sokoban(char* iLevelFile) {
        std::ifstream levelFile;
16
17
        levelFile.open(iLevelFile);
18
        levelFile >> width;
19
20
        levelFile.get();
21
        levelFile >> height;
22
        gameGrid = std::vector<char>((width * height), '');
23
24
        char content;
25
        size_t x;
26
        size_t y;
27
        for (x = 0; x < width; x++) {
28
            for (y = 0; y < height; y++) {
29
                levelFile >> content;
30
                gameGrid[x + y * width] = content;
            }
31
32
        }
33
34
        empty_locations = std::count(gameGrid.begin(), gameGrid.end(), 'a');
35
        movable_boxes = std::count(gameGrid.begin(), gameGrid.end(), 'A');
36
37
        // DISPLAY THE LEVEL TO THE TERMINAL
38
        std::cout << *this << std::endl;</pre>
        std::cout << movable_boxes << " " << empty_locations << std::endl;</pre>
39
40
        player.loadFromFile("player_05.png");
41
42
43
   void Sokoban::reset_level(char* iLevelFile) {
44
        empty_locations = 0;
        movable_boxes = 0;
45
        std::ifstream levelFile;
46
47
        levelFile.open(iLevelFile);
48
49
        levelFile >> width;
50
        levelFile.get();
51
        levelFile >> height;
52
        gameGrid = std::vector<char>((width * height), '');
53
54
        char content;
55
        size_t x;
56
        size_t y;
57
        for (x = 0; x < width; x++) {
58
            for (y = 0; y < height; y++) {
                levelFile >> content;
59
                gameGrid[x + y * width] = content;
60
61
            }
62
        }
63
64
        empty_locations = std::count(gameGrid.begin(), gameGrid.end(), 'a');
        movable_boxes = std::count(gameGrid.begin(), gameGrid.end(), 'A');
65
```

```
66
67
         // DISPLAY THE LEVEL TO THE TERMINAL
68
        std::cout << *this << std::endl;</pre>
69
        player.loadFromFile("player_05.png");
70
    }
71
 72
    void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const
73
        size_t x;
74
        size_t y;
75
        sf::Texture wall, empty, box, storage;
76
        char current_block;
77
 78
        wall.loadFromFile("block_06.png");
79
        empty.loadFromFile("ground_01.png");
80
        box.loadFromFile("crate_03.png");
81
        storage.loadFromFile("ground_04.png");
        for (x = 0; x < width; x++) {
82
            for (y = 0; y < height; y++) {
83
                 current_block = gameGrid[x + y * width];
84
85
                 if (current_block == '#') {
86
                     sf::Sprite block(wall);
                     block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
87
88
                     target.draw(block);
                 }
89
                 if (current_block == '.') {
90
91
                     sf::Sprite block(empty);
                     block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
92
93
                     target.draw(block);
                 }
94
95
                 if (current_block == 'a') {
96
                     sf::Sprite back(empty);
                     back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
97
98
                     target.draw(back);
99
                     sf::Sprite block(storage);
100
                     block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
101
                     target.draw(block);
102
                 }
103
                 if (current_block == 'A') {
104
                     sf::Sprite back(empty);
105
                     back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
106
                     target.draw(back);
107
                     sf::Sprite block(box);
108
                     block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
109
                     target.draw(block);
                 }
110
                 if (current_block == '1') {
111
112
                     sf::Sprite back(empty);
113
                     back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
114
                     target.draw(back);
115
                     sf::Sprite block(box);
116
                     block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
117
                     target.draw(block);
                 }
118
119
                 if (current_block == '0') {
                     if (isStorage) {
120
121
                         sf::Sprite back(storage);
122
                         back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
123
                         target.draw(back);
```

```
124
                     } else {
125
                          sf::Sprite back(empty);
                          back.setPosition(y * GRID_SCALE, x * GRID_SCALE);
126
127
                          target.draw(back);
128
129
                     sf::Sprite block(player);
                     block.setPosition(y * GRID_SCALE, x * GRID_SCALE);
130
131
                     target.draw(block);
132
                 }
133
            }
134
        }
135
    }
136
137
    bool Sokoban::isWon() const {
        if (empty_locations == 0 || movable_boxes == 0) {
138
139
             return true;
140
        }
141
        return false;
142
    }
143
144
    void Sokoban::movePlayer(const Movement m) {
145
        size_t x = 0;
146
        size_t y = 0;
147
        auto get_player_pos = [this, &x, &y] () -> void {
148
             char currentBlock;
149
150
151
             for (x = 0; x < width; x++) {
152
                 for (y = 0; y < height; y++) {
                     currentBlock = gameGrid[x + y * width];
153
154
                     if (currentBlock == '@') {
155
                         return;
                     }
156
                 }
157
158
             }
159
        };
160
        get_player_pos();
161
162
        switch (m) {
163
             case UP:
164
                 if (within_bounds(x - 1, y) && (gameGrid[(x - 1) + y * width])
165
                       != '#' \&\& gameGrid[(x - 1) + y * width] != '1')) {
166
                     set_player_pos(x, y, UP);
167
                     player.loadFromFile("player_08.png");
168
                 break;
169
             case LEFT:
170
171
                 if (within_bounds(x, y - 1) && (gameGrid[x + (y - 1) * width]
172
                       != '#' && gameGrid[x + (y - 1) * width] != '1')) {
173
                     set_player_pos(x, y, LEFT);
174
                     player.loadFromFile("player_20.png");
175
                 }
176
                 break;
177
             case DOWN:
                 if (within_bounds(x + 1, y) && (gameGrid[(x + 1) + y * width])
178
                       != '#' && gameGrid[(x + 1) + y * width] != '1')) {
179
180
                     set_player_pos(x, y, DOWN);
                     player.loadFromFile("player_05.png");
181
182
                 }
```

```
183
                 break;
184
             case RIGHT:
185
                 if (within_bounds(x, y + 1) && (gameGrid[x + (y + 1) * width])
                       != '#' && gameGrid[x + (y + 1) * width] != '1')) {
186
187
                     set_player_pos(x, y, RIGHT);
                     player.loadFromFile("player_17.png");
188
                 }
189
190
                 break;
191
             default: break;
192
         }
193
    }
194
195
    void Sokoban::set_player_pos(size_t x, size_t y, Movement m) {
196
         switch (m) {
197
             case UP:
198
                 if (!check_and_set_box(x, y, UP)) break;
199
                 if (isStorage) {
200
                     gameGrid[(x - 1) + y * width] = '0';
201
                     gameGrid[x + y * width] = 'a';
202
                     isStorage = false;
203
                 } else {
204
                     if (gameGrid[(x - 1) + y * width] == 'a') {
205
                          isStorage = true;
206
                     }
                     gameGrid[(x - 1) + y * width] = '0';
207
208
                     gameGrid[x + y * width] = '.';
                 }
209
210
                 break;
211
             case LEFT:
212
                 if (!check_and_set_box(x, y, LEFT)) break;
213
                 if (isStorage) {
                     gameGrid[x + (y - 1) * width] = '0';
214
                     gameGrid[x + y * width] = 'a';
215
216
                     isStorage = false;
217
                 } else {
218
                     if (gameGrid[x + (y - 1) * width] == 'a') {
219
                          isStorage = true;
220
221
                     gameGrid[x + (y - 1) * width] = '0';
                     gameGrid[x + y * width] = '.';
222
                 }
223
224
                 break:
225
             case DOWN:
226
                 if (!check_and_set_box(x, y, DOWN)) break;
227
                 if (isStorage) {
228
                     gameGrid[(x + 1) + y * width] = '0';
229
                     gameGrid[x + y * width] = 'a';
                     isStorage = false;
230
231
                 } else {
232
                     if (gameGrid[(x + 1) + y * width] == 'a') {
233
                          isStorage = true;
234
235
                     gameGrid[(x + 1) + y * width] = '0';
236
                     gameGrid[x + y * width] = '.';
237
                 }
238
                 break;
239
             case RIGHT:
                 if (!check_and_set_box(x, y, RIGHT)) break;
240
241
                 if (isStorage) {
```

```
242
                     gameGrid[x + (y + 1) * width] = '0';
243
                     gameGrid[x + y * width] = 'a';
244
                     isStorage = false;
245
                 } else {
                     if (gameGrid[x + (y + 1) * width] == 'a') {
246
247
                          isStorage = true;
248
                     gameGrid[x + (y + 1) * width] = '0';
249
250
                     gameGrid[x + y * width] = '.';
251
                 }
252
                 break;
253
             default: break;
        }
254
    }
255
256
257
    bool Sokoban::check_and_set_box(size_t x, size_t y, Movement m) {
258
        switch (m) {
        case UP:
259
             if (gameGrid[(x - 1) + y * width] == 'A') {
260
261
                 if (within_bounds(x - 2, y) && gameGrid[(x - 2) + y * width] !=
        '.') {
262
                     if (gameGrid[(x - 2) + y * width] == 'a') {
263
                          gameGrid[(x - 2) + y * width] = '1';
264
                         gameGrid[(x - 1) + y * width] = '0';
265
                         gameGrid[x + y * width] = '.';
266
                          empty_locations--;
267
                         movable_boxes--;
268
                         return true;
269
                         break:
270
                     }
271
                 }
272
                 if (within_bounds(x - 2, y) \&\& gameGrid[(x - 2) + y * width]
                      != '#' && gameGrid[(x - 2) + y * width] != 'A') {
273
                     gameGrid[(x - 2) + y * width] = 'A';
274
                     return true;
275
276
                 } else {
277
                     return false;
278
279
             } else {
280
                 return true;
             }
281
282
             break:
283
        case LEFT:
284
             if (gameGrid[x + (y - 1) * width] == 'A') {
285
                 if (within_bounds(x, y - 2) \&\& gameGrid[x + (y - 2) * width] !=
        '.') {
286
                     if (gameGrid[x + (y - 2) * width] == 'a') {
287
                          gameGrid[x + (y - 2) * width] = '1';
288
                          gameGrid[x + (y - 1) * width] = '0';
                          gameGrid[x + y * width] = '.';
289
290
                         empty_locations--;
291
                         movable_boxes--;
292
                         return true;
293
                         break;
294
                     }
295
                 }
296
                 if (within_bounds(x, y - 2) && gameGrid[x + (y - 2) * width]
297
                      != '#' \&\& gameGrid[x + (y - 2) * width] != 'A') {
298
                     gameGrid[x + (y - 2) * width] = 'A';
```

```
299
                     return true;
300
                 } else {
301
                     return false;
                 }
302
303
             } else {
304
                 return true;
             }
305
306
             break;
307
         case DOWN:
308
             if (gameGrid[(x + 1) + y * width] == 'A') {
309
                 if (within_bounds(x + 2, y) && gameGrid[(x + 2) + y * width] !=
        ·.') {
310
                     if (gameGrid[(x + 2) + y * width] == 'a') {
                          gameGrid[(x + 2) + y * width] = '1';
311
312
                          gameGrid[(x + 1) + y * width] = '0';
313
                          gameGrid[x + y * width] = '.';
314
                          empty_locations--;
315
                         movable_boxes--;
316
                         return true;
317
                          break;
                     }
318
319
                 }
320
                 if (within_bounds(x + 2, y) && gameGrid[(x + 2) + y * width]
321
                       != '#' && gameGrid[(x + 2) + y * width] != 'A') {
322
                     gameGrid[(x + 2) + y * width] = 'A';
323
                     return true;
324
                 } else {
325
                     return false;
326
                 }
327
             } else {
328
                 return true;
             }
329
330
             break;
331
         case RIGHT:
332
             if (gameGrid[x + (y + 1) * width] == 'A') {
333
                 if (within_bounds(x, y + 2) && gameGrid[x + (y + 2) * width] !=
        '.') {
334
                     if (gameGrid[x + (y + 2) * width] == 'a') {
335
                          gameGrid[x + (y + 2) * width] = '1';
                          gameGrid[x + (y + 1) * width] = '0';
336
337
                          gameGrid[x + y * width] = '.';
338
                          empty_locations--;
339
                         movable_boxes--;
340
                         return true;
341
                          break:
342
                     }
                 }
343
344
                 if (within_bounds(x, y + 2) \&\& gameGrid[x + (y + 2) * width]
345
                       != '#' && gameGrid[x + (y + 2) * width] != 'A') {
346
                     gameGrid[x + (y + 2) * width] = 'A';
347
                     return true;
348
                 } else {
349
                     return false;
350
                 }
351
             } else {
352
                 return true;
             }
353
354
             break;
355
             default: break;
```

```
356
357
358
         return false;
359
    }
360
    bool Sokoban::within_bounds(size_t x, size_t y) const {
361
362
         if (x >= 0 && x < width) {
363
             if (y >= 0 && y < height)</pre>
364
                 return true;
365
         }
366
         return false;
367
    }
368
369
    std::istream& operator>>(std::istream& inStream, Sokoban& object) {
370
         char* file = new char[10];
371
         char c;
372
         int i;
373
         size_t w;
374
         size_t h;
375
376
         for (i = 0; i < 10; i++) {
377
             inStream >> c;
378
             file[i] = c;
379
         }
380
381
         std::ifstream levelFile;
382
         levelFile.open(file);
383
384
         levelFile >> w;
385
         levelFile.get();
386
         levelFile >> h;
387
388
         object.set_width(w);
389
         object.set_height(h);
390
         object.set_gameGrid(w * h);
391
392
         char content;
393
         size_t x;
394
         size_t y;
395
         for (x = 0; x < object.get_width(); x++) {
396
             for (y = 0; y < object.get_height(); y++) {</pre>
397
                 levelFile >> content;
398
                  object.get_gameGrid()[x + y * object.get_width()] = content;
399
                  if (content == 'a') object.set_empty_locations(1);
                  if (content == 'A') object.set_movable_boxes(1);
400
401
             }
402
         }
403
404
         return inStream;
405
    }
406
407
    std::ostream& operator<<(std::ostream& outStream, const Sokoban& object) {</pre>
408
         size_t x;
409
         size_t y;
410
         for (x = 0; x < object.get_width(); x++) {
411
412
             for (y = 0; y < object.get_height(); y++) {</pre>
413
                  outStream << object.get_gameGrid()[x + y * object.get_width()];</pre>
414
             }
```

3 PS3: Pythagorean Tree

3.1 Discussion

What I accomplished

This project creates a PTree class with the goal of drawing a Pythagorean Tree to an SFML display window. I implemented a structure called Branch to use in my PTree class. My Branch structure is similar to a node that would be used in a tree. It contains a self, left, and right which represents the base square and the two squares on top respectively. The PTree contains a function pTree() that initializes the first square (base square), and within the pTree function a helper function extend is called. This function recursively calls itself until the number of recursions specified by the user is met. Each time the function is called, a new Branch is drawn onto the tree, creating the visualization of Pythagoras Tree. However, my program does not display the tree correctly. The implementation of the recursive function does not properly position the next square relative to the previous one.

3.2 Design and Features

The PTree class contains a constructor that initializes the member variables L, N, recursion_count, root and also sets up the base of the tree by initializing a Branch structure. The class contains a get_length() function which returns the value of L, and it also contains a set_length() function that allows L to be updated as more levels of the tree are added. I implemented pTree function as a friend function so it has the ability to access the member variables directly. The extend function is a member function of PTree. I implemented it this way to make it easy to mutate and access the member functions and variables directly.

I decided to create the Branch structure because I thought one variable containing a root, right, and left would be useful when attempting to extend the tree. Having used a similar structure before gave me familiarity that was helpful when implementing.

What I already knew and what I learned

Going into this assignment, I used SFML render windows and draw functions before, so I knew how to display simple shapes such as a square. I also knew how to create a tree-like structure, which is how I chose to implement Branch.

When working on this assignment, I learned how to use math skills that I already have to find different distances from a relative point. In order to find the top right and left of the current square, I had to use the current square's origin as a starting point, then find use an equation to find the top corners position.

My program does not produce a proper output.

3.3 Codebase

```
CC = g++
 1
 2
   CFLAGS = -g -Wall -Werror -pedantic -std=c++17 -g
3
   LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
   DEPS = PTree.hpp
 4
   OBJS = PTree.o
5
6
 7
   %.o: %.cpp $(DEPS)
     $(CC) $(CFLAGS) -c $<
8
9
10
   .PHONY: all clean lint
11
12
   all: PTree
13
14
   PTree: main.o $(OBJS)
     $(CC) $(CFLAGS) -0 $0 $^ $(LIBS)
15
16
```

```
17 | lint:

18 | cpplint *.cpp *.hpp

19 | 20 | clean:

21 | rm *.o PTree
```

```
// Copyright 2023 James Walsh
   #include <iostream>
3
   #include "PTree.hpp"
   #include <SFML/System.hpp>
   #include <SFML/Window.hpp>
6
   #include <SFML/Graphics.hpp>
7
8
   int main(int argc, char* argv[]) {
9
       PTree myPTree(40, 3);
10
11
       sf::RenderWindow window(sf::VideoMode(6 * myPTree.get_length(),
12
        4 * myPTree.get_length()), "Pythagoras Tree");
13
       while (window.isOpen()) {
14
15
            sf::Event event;
16
            while (window.pollEvent(event)) {
17
                if (event.type == sf::Event::Closed) {
18
                    window.close();
19
20
                window.clear();
21
                pTree(myPTree, window);
22
            }
23
       }
24
25
       return 0;
   }
26
```

```
// Copyright 2023 James Walsh
 1
   #pragma once
 2
3
   #include <iostream>
   #include <SFML/System.hpp>
 4
   #include <SFML/Window.hpp>
   #include <SFML/Graphics.hpp>
6
 7
8
   struct branch;
9
   typedef struct branch Branch;
10
11
  struct branch {
     sf::RectangleShape self;
12
     sf::Vector2f topLeft;
13
     sf::Vector2f topRight;
14
15
     Branch* left;
16
     Branch* right;
17
   };
18
19
   class PTree: public sf::Drawable {
    public:
20
       PTree(double length, int iterations): L(length),
21
22
        N(iterations), recursion_count(0), root(new Branch) {
23
         root->self.setSize(sf::Vector2f(L, L));
24
         root->topLeft = root->self.getOrigin();
25
         float x = (root->self.getOrigin()).x + L * 6;
         float y = (root->self.getOrigin()).y * 4;
26
         root->topRight = sf::Vector2f(x, y);
27
```

```
28
          root->self.setOrigin((L / 2), (L / 2));
29
         root->self.setPosition(((6 * L) / 2), (4 * L) - (L / 2));
30
        ~PTree() {
31
32
         delete root;
        }
33
34
        double get_length() const { return L; }
        void set_length(double length) { L = length; }
35
36
        friend void pTree(PTree& _PTree, sf::RenderWindow& window);
37
        void extend(Branch* next, sf::RenderWindow& window);
38
39
    private:
40
       void draw(sf::RenderTarget& target, sf::RenderStates states) const
       override;
41
       double L;
42
        int N;
43
        int recursion_count;
44
        Branch* root;
45
   |};
46
47
   void pTree(PTree& _PTree, sf::RenderWindow& window);
```

```
// Copyright 2023 James Walsh
 1
   #include "PTree.hpp"
 2
 3
   void pTree(PTree& _PTree, sf::RenderWindow& window) {
 4
 5
        if (_PTree.recursion_count == 0) {
 6
            window.draw(_PTree.root->self);
 7
            window.display();
 8
            _PTree.set_length(_PTree.L / 2);
 9
        }
10
11
        _PTree.extend(_PTree.root, window);
12
13
   void PTree::extend(Branch* next, sf::RenderWindow& window) {
14
15
        next->left = new Branch;
16
        next->right = new Branch;
17
18
        set_length(L / 2);
19
        next->left->self.setSize(sf::Vector2f(L, L));
20
        next->left->self.setFillColor(sf::Color::Red);
        next->left->self.setOrigin(L / 2, L / 2);
21
22
        next->left->self.setPosition(next->topLeft);
23
        window.draw(next->left->self);
24
        window.display();
25
26
        next->right->self.setSize(sf::Vector2f(L, L));
27
        next->right->self.setFillColor(sf::Color::Red);
28
        next->right->self.setOrigin(L / 2, L / 2);
29
        next->right->self.setPosition(next->topRight);
30
        window.draw(next->right->self);
31
        window.display();
32
        recursion_count++;
33
34
        if (recursion_count != N) {
35
            extend(next->left, window);
36
            extend(next->right, window);
37
        }
38 }
```

4 PS4: Checkers

4.1 Discussion

What I accomplished

For this project, I created a complete checkers game. The board is an 8 x 8 square made up of red and black tiles, starting with a red tile in the top left. Red and Black game pieces are placed on the top 3 rows and the bottom 3 of black tiles, with black game pieces at the top and red at the bottom. The game is displayed using an SFML render window. The game starts by allowing the player to select a piece with the mouse. Black gets to go first, so selecting a piece will turn the tile below blue to indicate the current selection. If you click the same tile, it will remove the selection and allow the same player to select a new piece. Otherwise, selecting any other piece will cause the program to check if it is a valid move. It is a valid move if the tile the player is trying to move to is black, is empty, and one space diagonal. Or the player can jump an oposing piece onto an empty diagonal black tile. Jumping a piece will remove the piece that is jumped. If the move is valid, the piece will be moved to the selected tile, and now the red player can now select a piece. The game will continue until one player has no pieces left. Once one player has won, a red winner icon appears, or a black winner icon appears.

Design and Features

To implement the game, I created a Checkers class. In my Checkers class, there is a default constructor which initializes the member variables and sets up a blank board. To store the game information, I used a std::vector of character arrays that are length 8 (the size of each row). I decided to do this because if I changed the vector to be a std::list, the iterators in my functions are easy to transfer to the corresponding iterators. After the game is board calls a member function start_board(), this function places the game pieces in their initial positions. The class has 3 functions that are used for selecting pieces, piece_select(), move_piece(), and can_move(). The piece select function is called the main function in main.cpp, and highlights the selected piece and returns true. Then main calls move_piece, which waits for the user to select a destination position, and after that, calls can move to see if the move is allowed. The can move function checks to see if the move is allowed (a valid move is described in the previous section). If the move is allowed, the function returns true. However, there is an exception to this, if the player is attempting to "jump" a tile then, the remove_piece function is called. The remove_piece function examines the game grid to see if the tile the player is trying to jump contains an opposing player's game piece. If so, the piece is removed from the board and the remove_piece function returns true. Next, the can_move function updates the game board properly. If the move is not allowed, if can move returns false, the piece is not moved and unhighlights the current tile, and the player is allowed to select a new destination tile. The class also contains two functions red_won() and black_won(). These functions return true if the opposing player has no pieces remaining and false otherwise. Lastly, the class contains accessor functions for the dimensions and scale of the board along with an accessor function for the gameGrid vector. Also, the insertion operator is overloaded for the class which outputs the components of the gameGrid to the given ostream.

When implementing, I created a lambda expression <code>is_even()</code> which checks if a size_t number is even or odd. It returns true if even and false if odd. Also, I used the <code><algorithm></code> function <code>count_if()</code> in the red_won and black_won functions. I used the algorithm to iterate through the gameGrid vector. At each character array I passed in a lambda function to return true if an oposing teams piece was found.

What I already knew and what I learned

Going into this assignment I was aware of how low-level game movement works due to my experience programming the Sokoban game. Programming the Sokoban game also taught me how important delegation to other functions is very important, so I incorporated that into this program as well.

When completing this assignment, I learned a lot about using std classes to my advantage. Such as std::pair<> was very useful for referring to the current position and the destination position. I have not used pairs to the extent I did in this program, so I learned a lot using them. and they will be something I incorporate into my code for future projects.

Below is a screenshot of a New Game: Figure 4.

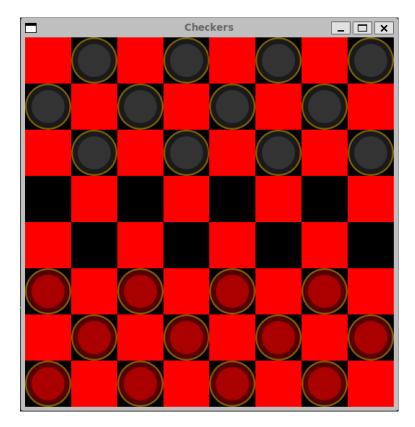


Figure 4: Start of Game.

4.2 Codebase

```
CC = g++
 1
 2
   CFLAGS = -g -Wall -Werror -pedantic -std=c++17
3
   LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
   DEPS = Checkers.hpp
 4
   OBJS = Checkers.o
 5
6
7
   %.o: %.cpp $(DEPS)
     $(CC) $(CFLAGS) -c $<
8
9
10
   .PHONY: all clean lint
11
   all: Checkers
12
13
14
   Checkers: main.o $(OBJS)
     $(CC) $(CFLAGS) -o $0 $^ $(LIBS)
15
16
17
   lint:
18
     cpplint *.cpp *.hpp
19
20
   clean:
21
     rm *.o Checkers
```

```
// Copyright 2023 James Walsh
#include <iostream>
#include "Checkers.hpp"
#include <SFML/System.hpp>
```

```
#include <SFML/Window.hpp>
 6
   #include <SFML/Graphics.hpp>
 7
 8
   int main(int argc, char* argv[]) {
 9
        Checkers newGame;
10
        sf::RenderWindow window(sf::VideoMode(
                newGame.get_dimension() * newGame.get_scale(),
11
                newGame.get_dimension() * newGame.get_scale()), "Checkers");
12
13
        sf::Texture winnerImage;
        sf::Sprite redWon, blackWon;
14
15
        redWon.scale(.7, .7);
16
        blackWon.scale(.3, .3);
17
18
        char p;
19
20
        while (window.isOpen()) {
21
            sf::Event event;
22
            while (window.pollEvent(event)) {
23
                if (event.type == sf::Event::Closed) {
24
                    window.close();
25
26
                if (!newGame.red_Won() && !newGame.black_Won()) {
27
                    if (event.type == sf::Event::MouseButtonPressed) {
28
                         size_t x = event.mouseButton.x / 64;
29
                         size_t y = event.mouseButton.y / 64;
                         std::pair<size_t, size_t> pos(y, x);
30
31
                         if (newGame.piece_select(pos, p, window)) {
32
                             window.draw(newGame);
33
                             window.display();
                             newGame.move_piece(pos, p, window);
34
35
                        }
                    }
36
                    window.clear();
37
38
                    window.draw(newGame);
39
                    window.display();
40
                } else {
41
                    if (newGame.red_Won()) {
42
                         winnerImage.loadFromFile("redwin.png");
                        redWon.setTexture(winnerImage);
43
                        redWon.setPosition(2 * 64, 2 * 46);
44
45
                         window.clear();
                         window.draw(newGame);
46
47
                         window.draw(redWon);
48
                         window.display();
49
                    } else if (newGame.black_Won()) {
50
                         winnerImage.loadFromFile("blackwin.png");
51
52
                         blackWon.setTexture(winnerImage);
53
                         blackWon.setPosition(2 * 64, 2 * 46);
54
                         window.clear();
55
                         window.draw(newGame);
56
                         window.draw(blackWon);
                         window.display();
57
                    }
58
                }
59
           }
60
        }
61
62
63
        return 0;
```

```
// Copyright 2023 James Walsh
 1
   #pragma once
 3
   #include <iostream>
   #include <vector>
 4
   #include <utility>
 5
   #include <SFML/System.hpp>
 6
 7
   #include <SFML/Window.hpp>
 8
   #include <SFML/Graphics.hpp>
 9
10
   class Checkers: public sf::Drawable {
11
   public:
12
       Checkers();
13
14
       void start_board();
15
       bool piece_select(std::pair<size_t, size_t> position, char& p, const sf
       ::RenderWindow& w);
16
       void move_piece(std::pair<size_t, size_t> position, char piece, sf::
       RenderWindow& w);
17
       bool can_move(char piece, std::pair<size_t, size_t> position,
18
                        std::pair<size_t, size_t> destPosition);
19
       bool remove_piece(char piece, std::pair<size_t, size_t> position,
20
                        std::pair<size_t, size_t> destPosition);
21
       bool red_Won();
22
       bool black_Won();
23
       size_t get_dimension() const { return BOARD_DIMENSION; }
24
       size_t get_scale() const { return GRID_SCALE; }
25
       char* get_gameGrid_at(std::vector<char[]>::iterator i) const { return *i
26
       friend std::ostream& operator<<(std::ostream& outStream, const Checkers&
        object);
27
28
    private:
29
       void draw(sf::RenderTarget& target, sf::RenderStates states) const
       override;
30
       const size_t BOARD_DIMENSION;
31
       std::vector<char[8]> gameGrid;
32
       const int GRID_SCALE;
       sf::Texture redPiece, blackPiece, redKing, blackKing;
33
34
       int turn;
   };
35
36
37
   std::ostream& operator<<(std::ostream& outStream, const Checkers& object);
   // Copyright 2023 James Walsh
 1
   #include <algorithm>
 2
 3
   #include "Checkers.hpp"
 4
   Checkers::Checkers():BOARD_DIMENSION(8), gameGrid(std::vector<char[8]>(
 5
       BOARD_DIMENSION)),
 6
            GRID_SCALE(64), turn(0) {
       redPiece.loadFromFile("redpawn.png");
 7
       blackPiece.loadFromFile("blackpawn.png");
 8
 9
       redKing.loadFromFile("redking.png");
10
       blackKing.loadFromFile("blackking.png");
11
       std::vector<char[8]>::iterator pos;
12
       size_t i;
13
       size_t start_color = 0;
14
```

64 |}

```
15
        auto is_even = [] (size_t x) -> bool {
16
            return (x \% 2 == 0);
17
        };
18
        // start_color is even = R odd = B
19
20
        for (pos = gameGrid.begin(); pos != gameGrid.end(); pos++, start_color
       ++) {
21
            if (is_even(start_color)) {
22
                for (i = 0; i < BOARD_DIMENSION; i++) {</pre>
23
                     if (is_even(i)) {
24
                         (*pos)[i] = 'T';
25
                     } else {
                         (*pos)[i] = 't';
26
27
28
                }
29
            } else {
30
                for (i = 0; i < BOARD_DIMENSION; i++) {</pre>
                     if (is_even(i)) {
31
32
                         (*pos)[i] = 't';
33
                     } else {
34
                         (*pos)[i] = 'T';
                     }
35
36
                }
37
            }
38
        }
39
        start_board();
40
41
        std::cout << "Game Board:\n" << *this << std::endl;</pre>
42
   }
43
44
   void Checkers::start_board() {
45
        std::vector<char[8]>::iterator pos;
46
        size_t row;
47
        size_t i;
48
49
        for (pos = gameGrid.begin(), row= 0; pos != gameGrid.end(); pos++, row
       ++) {
50
            for (i = 0; i < BOARD_DIMENSION; i++) {</pre>
                if (row < 3) {
51
                     if ((*pos)[i] == 't') {
52
53
                         (*pos)[i] = 'b';
                     }
54
                } else if (row > 4) {
55
56
                     if ((*pos)[i] == 't') {
                         (*pos)[i] = 'r';
57
58
                     }
                }
59
60
            }
61
        }
62
   }
63
64
   bool Checkers::piece_select(std::pair<size_t, size_t> position, char &p,
65
                                      const sf::RenderWindow& w) {
66
        char piece;
        piece = gameGrid.at(position.first)[position.second];
67
68
        // turn = even: Black turn; turn = odd: Red turn
69
        if (turn % 2 == 0) {
70
            if (piece == 'b') {
71
                gameGrid.at(position.first)[position.second] = 'p';
```

```
72
                 p = 'p';
73
                 return true;
74
            } else if (piece == 'B') {
75
                 gameGrid.at(position.first)[position.second] = 'm';
 76
                 p = m';
 77
                 return true;
            }
 78
 79
        } else if (turn % 2 != 0) {
             if (piece == 'r') {
80
                 gameGrid.at(position.first)[position.second] = 'P';
81
82
                 p = 'P';
83
                 return true;
            } else if (piece == 'R') {
84
85
                 gameGrid.at(position.first)[position.second] = 'M';
86
                 p = 'M';
87
                 return true;
88
            }
89
90
        return false;
91
92
    void Checkers::move_piece(std::pair<size_t, size_t> position, char piece, sf
93
        ::RenderWindow& w) {
94
        bool pick = false;
95
96
        while (!pick) {
97
            sf::Event moveTo;
98
            while (w.pollEvent(moveTo)) {
99
                 if (moveTo.type == sf::Event::MouseButtonPressed) {
100
                     moveTo.type = sf::Event::MouseButtonPressed;
101
                     size_t destX = moveTo.mouseButton.x / 64;
102
                     size_t destY = moveTo.mouseButton.y / 64;
103
                     std::pair<size_t, size_t> destPos(destY, destX);
104
105
                     if (destPos.first == position.first && destPos.second ==
        position.second) {
106
                         if (piece == 'P') {
107
                             gameGrid.at(position.first)[position.second] = 'r';
108
                         } else if (piece == 'p') {
109
                             gameGrid.at(position.first)[position.second] = 'b';
110
                         } else if (piece == 'M') {
111
                             gameGrid.at(position.first)[position.second] = 'R';
112
                         } else if (piece == 'm') {
113
                             gameGrid.at(position.first)[position.second] = 'B';
114
115
                         return;
116
117
                     if (piece == 'P') {
118
                         if (can_move(piece, position, destPos)) {
119
                             if (destPos.first == 0) {
120
                                 gameGrid.at(destPos.first)[destPos.second] = 'R'
121
                             } else {
122
                                  gameGrid.at(destPos.first)[destPos.second] = 'r'
123
124
                             gameGrid.at(position.first)[position.second] = 't';
125
                             turn++;
126
                             pick = true;
```

```
127
128
                     } else if (piece == 'p') {
129
                         if (can_move(piece, position, destPos)) {
130
                              if (destPos.first == 7) {
131
                                  gameGrid.at(destPos.first)[destPos.second] = 'B'
132
                             } else {
133
                                  gameGrid.at(destPos.first)[destPos.second] = 'b'
134
135
                             gameGrid.at(position.first)[position.second] = 't';
136
                             turn++;
137
                             pick = true;
                         }
138
139
                     } else if (piece == 'M') {
140
                         if (can_move(piece, position, destPos)) {
141
                             gameGrid.at(destPos.first)[destPos.second] = 'R';
142
                             gameGrid.at(position.first)[position.second] = 't';
143
                             turn++;
                             pick = true;
144
145
146
                     } else if (piece == 'm') {
147
                         if (can_move(piece, position, destPos)) {
148
                              gameGrid.at(destPos.first)[destPos.second] = 'B';
149
                             gameGrid.at(position.first)[position.second] = 't';
150
                             turn++;
                             pick = true;
151
152
                         }
153
                     } else {
                         if (piece == 'P') {
154
155
                             gameGrid.at(position.first)[position.second] = 'r';
156
                         } else if (piece == 'p') {
157
                              gameGrid.at(position.first)[position.second] = 'b';
                         }
158
                     }
159
160
                 }
            }
161
162
        }
163
    }
164
165
    bool Checkers::can_move(char piece, std::pair<size_t, size_t> position,
166
                                  std::pair<size_t, size_t> destPosition) {
        if (gameGrid.at(destPosition.first)[destPosition.second] == 't') {
167
168
             if (piece == 'P') {
169
                 if (destPosition.first == position.first - 1) {
                     if (destPosition.second == position.second + 1) {
170
171
                         return true;
172
                     } else if (destPosition.second == position.second - 1) {
173
                         return true;
                     }
174
175
                 } else if (destPosition.first == position.first - 2) {
176
                     if (destPosition.second == position.second + 2) {
177
                         if (remove_piece(piece, position, destPosition)) {
                             return true;
178
                         }
179
180
                         return false;
181
                     } else if (destPosition.second == position.second - 2) {
182
                         if (remove_piece(piece, position, destPosition)) {
183
                             return true;
```

```
184
185
                         return false;
                     }
186
                 }
187
             } else if (piece == 'p') {
188
                 if (destPosition.first == position.first + 1) {
189
190
                     if (destPosition.second == position.second + 1) {
191
                         return true;
192
                     } else if (destPosition.second == position.second - 1) {
193
                         return true;
194
                     }
195
                 } else if (destPosition.first == position.first + 2) {
196
                     if (destPosition.second == position.second + 2) {
197
                         if (remove_piece(piece, position, destPosition)) {
198
                             return true;
199
                         }
200
                         return false;
                     } else if (destPosition.second == position.second - 2) {
201
202
                         if (remove_piece(piece, position, destPosition)) {
203
                              return true;
                         }
204
205
                         return false;
206
                     }
207
                 }
             } else if (piece == 'M') {
208
209
                 if (destPosition.first == position.first - 1) {
210
                     if (destPosition.second == position.second + 1) {
211
                         return true;
212
                     } else if (destPosition.second == position.second - 1) {
213
                         return true;
214
215
                 } else if (destPosition.first == position.first + 1) {
216
                     if (destPosition.second == position.second + 1) {
217
                         return true;
218
                     } else if (destPosition.second == position.second - 1) {
219
                         return true;
                     }
220
221
                 } else if (destPosition.first == position.first - 2) {
222
                     if (destPosition.second == position.second + 2) {
223
                         if (remove_piece(piece, position, destPosition)) {
224
                             return true;
                         }
225
226
                         return false;
227
                     } else if (destPosition.second == position.second - 2) {
228
                         if (remove_piece(piece, position, destPosition)) {
229
                             return true;
230
                         }
231
                         return false;
232
233
                 } else if (destPosition.first == position.first + 2) {
234
                     if (destPosition.second == position.second + 2) {
235
                         if (remove_piece(piece, position, destPosition)) {
236
                             return true;
                         }
237
238
                         return false;
239
                     } else if (destPosition.second == position.second - 2) {
240
                         if (remove_piece(piece, position, destPosition)) {
241
                             return true;
242
                         }
```

```
243
                         return false;
244
                     }
245
                 }
246
            } else if (piece == 'm') {
247
                 if (destPosition.first == position.first - 1) {
                     if (destPosition.second == position.second + 1) {
248
249
                         return true;
250
                     } else if (destPosition.second == position.second - 1) {
251
                         return true;
252
253
                 } else if (destPosition.first == position.first + 1) {
254
                     if (destPosition.second == position.second + 1) {
255
                         return true;
256
                     } else if (destPosition.second == position.second - 1) {
257
                         return true;
258
                     }
259
                 } else if (destPosition.first == position.first - 2) {
                     if (destPosition.second == position.second + 2) {
260
261
                         if (remove_piece(piece, position, destPosition)) {
262
                             return true;
                         }
263
264
                         return false;
265
                     } else if (destPosition.second == position.second - 2) {
266
                         if (remove_piece(piece, position, destPosition)) {
267
                             return true;
                         }
268
269
                         return false;
270
                     }
271
                 } else if (destPosition.first == position.first + 2) {
272
                     if (destPosition.second == position.second + 2) {
273
                         if (remove_piece(piece, position, destPosition)) {
274
                             return true:
                         }
275
276
                         return false;
277
                     } else if (destPosition.second == position.second - 2) {
278
                         if (remove_piece(piece, position, destPosition)) {
279
                             return true;
280
                         }
281
                         return false;
                     }
282
                 }
283
            }
284
285
        }
286
        return false;
287
    }
288
289
    bool Checkers::remove_piece(char piece, std::pair<size_t, size_t> position,
290
                         std::pair<size_t, size_t> destPosition) {
291
        if (piece == 'P') {
292
             if (destPosition.second == position.second + 2) {
293
                 if (gameGrid.at(destPosition.first + 1)[destPosition.second - 1]
         == 'b' ||
294
                         gameGrid.at(destPosition.first + 1)[destPosition.second
        - 1] == 'B') {
295
                     gameGrid.at(destPosition.first + 1)[destPosition.second - 1]
         = 't';
296
                     return true;
297
298
            } else if (destPosition.second == position.second - 2) {
```

```
299
                 if (gameGrid.at(destPosition.first + 1)[destPosition.second + 1]
         == 'b' ||
300
                         gameGrid.at(destPosition.first + 1)[destPosition.second
        + 1] == 'B') {
301
                     gameGrid.at(destPosition.first + 1)[destPosition.second + 1]
         = 't';
302
                     return true;
303
                 }
304
            }
305
        } else if (piece == 'p') {
306
             if (destPosition.second == position.second + 2) {
307
                 if (gameGrid.at(destPosition.first - 1)[destPosition.second - 1]
         == 'r' ||
308
                         gameGrid.at(destPosition.first - 1)[destPosition.second
        - 1] == 'R') {
309
                     gameGrid.at(destPosition.first - 1)[destPosition.second - 1]
         = 't';
310
                     return true;
                 }
311
312
             } else if (destPosition.second == position.second - 2) {
313
                 if (gameGrid.at(destPosition.first - 1)[destPosition.second + 1]
         == 'r' ||
314
                         gameGrid.at(destPosition.first - 1)[destPosition.second
        + 1] == 'R') {
315
                     gameGrid.at(destPosition.first - 1)[destPosition.second + 1]
         = 't';
316
                     return true;
317
                 }
318
            }
319
        } else if (piece == 'M') {
320
             if (destPosition.second == position.second + 2) {
321
                 if (destPosition.first == position.first + 2) {
                     if (gameGrid.at(destPosition.first - 1)[destPosition.second
322
        - 1] == 'b' ||
323
                             gameGrid.at(destPosition.first - 1)[destPosition.
       second - 1] == 'B') {
324
                         gameGrid.at(destPosition.first - 1)[destPosition.second
        -1] = 't';
325
                         return true;
                     }
326
327
                 } else if (destPosition.first == position.first - 2) {
328
                     if (gameGrid.at(destPosition.first + 1)[destPosition.second
        - 1] == 'b' ||
329
                                 gameGrid.at(destPosition.first + 1)[destPosition
        .second - 1] == 'B') {
330
                         gameGrid.at(destPosition.first + 1)[destPosition.second
        -1] = 't';
331
                         return true;
332
                     }
333
                 }
334
            } else if (destPosition.second == position.second - 2) {
335
                 if (destPosition.first == position.first + 2) {
336
                     if (gameGrid.at(destPosition.first - 1)[destPosition.second
        + 1] == 'b' ||
                             gameGrid.at(destPosition.first - 1)[destPosition.
337
       second + 1] == 'B') {
338
                         gameGrid.at(destPosition.first - 1)[destPosition.second
        + 1] = 't';
339
                         return true;
```

```
340
341
                 } else if (destPosition.first == position.first - 2) {
342
                    if (gameGrid.at(destPosition.first + 1)[destPosition.second
        + 1] == 'b' ||
                                 gameGrid.at(destPosition.first + 1)[destPosition
343
        .second + 1] == 'B') {
344
                         gameGrid.at(destPosition.first + 1)[destPosition.second
        + 1] = 't';
345
                         return true;
346
                     }
                 }
347
348
            }
349
        } else if (piece == 'm') {
350
            if (destPosition.second == position.second + 2) {
351
                 if (destPosition.first == position.first + 2) {
352
                     if (gameGrid.at(destPosition.first - 1)[destPosition.second
        - 1] == 'r' ||
                             gameGrid.at(destPosition.first - 1)[destPosition.
353
        second - 1] == 'R') {
                         gameGrid.at(destPosition.first - 1)[destPosition.second
354
        -1] = 't';
355
                         return true;
356
357
                 } else if (destPosition.first == position.first - 2) {
                     if (gameGrid.at(destPosition.first + 1)[destPosition.second
358
        - 1] == 'r' ||
                                 gameGrid.at(destPosition.first + 1)[destPosition
359
        .second - 1] == 'R') {
360
                         gameGrid.at(destPosition.first + 1)[destPosition.second
        -1] = 't';
361
                         return true;
                     }
362
363
364
            } else if (destPosition.second == position.second - 2) {
365
                 if (destPosition.first == position.first + 2) {
366
                     if (gameGrid.at(destPosition.first - 1)[destPosition.second
        + 1] == 'r' ||
367
                             gameGrid.at(destPosition.first - 1)[destPosition.
        second + 1] == 'R') {
368
                         gameGrid.at(destPosition.first - 1)[destPosition.second
        + 1] = 't';
369
                         return true;
370
                     }
371
                 } else if (destPosition.first == position.first - 2) {
372
                     if (gameGrid.at(destPosition.first + 1)[destPosition.second
       + 1] == 'r' ||
373
                                 gameGrid.at(destPosition.first + 1)[destPosition
        .second + 1] == 'R') {
374
                         gameGrid.at(destPosition.first + 1)[destPosition.second
        + 1] = 't';
375
                         return true;
376
                     }
377
                 }
            }
378
379
380
        return false;
    }
381
382
383 | bool Checkers::red_Won() {
```

```
384
        int black = std::count_if(gameGrid.begin(), gameGrid.end(), [](char* arr
        ) -> bool {
385
             int found = 0;
386
             for (int i = 0; i < 8; i++) {
                 if (arr[i] == 'b' || arr[i] == 'B') {
387
388
                     found++;
389
                 }
390
             }
391
             return found;
392
         });
393
         return !black;
394
    }
395
396
    bool Checkers::black_Won() {
397
         int red = std::count_if(gameGrid.begin(), gameGrid.end(), [](char* arr)
        -> bool {
398
             int found = 0;
             for (int i = 0; i < 8; i++) {
399
400
                 if (arr[i] == 'r' || arr[i] == 'R') {
401
                     found++;
402
403
             }
404
             return found;
405
         });
406
        return !red;
407
    }
408
409
    void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) const
410
         std::vector<char[8]>::const_iterator pos;
411
         size_t x;
412
         size_t y;
413
414
         sf::RectangleShape tile(sf::Vector2f(64, 64));
415
         sf::Sprite piece;
416
         for (pos = gameGrid.begin(), x = 0; pos != gameGrid.end(); pos++, x++) {
417
418
             for (y = 0; y < BOARD_DIMENSION; y++) {</pre>
419
                 switch ((*pos)[y]) {
                     case 'T':
420
421
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
422
                          tile.setFillColor(sf::Color::Red);
423
                         target.draw(tile);
424
                         break;
425
                     case 'r':
426
                          tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
427
                          tile.setFillColor(sf::Color::Black);
428
                         target.draw(tile);
429
                         piece.setTexture(redPiece);
430
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
431
                         target.draw(piece);
432
                         break;
                     case 'R':
433
434
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
435
                         tile.setFillColor(sf::Color::Black);
436
                         target.draw(tile);
437
                         piece.setTexture(redKing);
438
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
439
                          target.draw(piece);
```

```
440
                         break;
441
                     case 't':
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
442
443
                         tile.setFillColor(sf::Color::Black);
444
                         target.draw(tile);
445
                         break:
                     case 'b':
446
447
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
448
                         tile.setFillColor(sf::Color::Black);
449
                         target.draw(tile);
450
                         piece.setTexture(blackPiece);
451
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
452
                         target.draw(piece);
453
                         break;
                     case 'B':
454
455
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
                         tile.setFillColor(sf::Color::Black);
456
457
                         target.draw(tile);
458
                         piece.setTexture(blackKing);
459
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
460
                         target.draw(piece);
461
                         break;
                     case 'P':
462
463
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
464
                         tile.setFillColor(sf::Color::Blue);
465
                         target.draw(tile);
466
                         piece.setTexture(redPiece);
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
467
468
                         target.draw(piece);
469
                         break;
470
                     case 'p':
471
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
472
                         tile.setFillColor(sf::Color::Blue);
473
                         target.draw(tile);
474
                         piece.setTexture(blackPiece);
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
475
476
                         target.draw(piece);
477
                         break;
                     case 'M':
478
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
479
480
                         tile.setFillColor(sf::Color::Blue);
481
                         target.draw(tile);
482
                         piece.setTexture(redKing);
483
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
484
                         target.draw(piece);
485
                         break;
486
                     case 'm':
487
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
488
                         tile.setFillColor(sf::Color::Blue);
489
                         target.draw(tile);
490
                         piece.setTexture(blackKing);
491
                         piece.setPosition(y * GRID_SCALE, x * GRID_SCALE);
492
                         target.draw(piece);
493
                         break;
494
                     case 'a':
495
                         tile.setPosition(y * GRID_SCALE, x * GRID_SCALE);
496
                         tile.setFillColor(sf::Color::Blue);
497
                         target.draw(tile);
498
                         break;
```

```
499
                      default: break;
500
                 }
            }
501
         }
502
503
    }
504
    std::ostream& operator<<(std::ostream& outStream, const Checkers& object) {</pre>
505
506
         std::vector<char[8]>::const_iterator pos;
507
         size_t i;
508
509
         for (pos = object.gameGrid.begin(); pos != object.gameGrid.end(); pos++)
             for (i = 0; i < object.BOARD_DIMENSION; i++) {</pre>
510
511
                 outStream << (*pos)[i];</pre>
512
             }
513
             outStream << std::endl;</pre>
514
         }
515
         return outStream;
516 }
```

5 PS5: DNA Alignment

5.1 Discussion

What I accomplished

The goal of this project was to create an algorithm that finds the optimal alignment of two DNA strands using dynamic programming. The program will measure the similarity of two genetic sequences by finding the "Edit-distance." The edit distance is calculated by finding the sum of all costs. There are three different cost operations, inserting a gap cost 2, aligning two unidentical characters costs 1, and aligning two identical characters has a cost of 0. To implement this, I created an EDistance class. The class contains a constructor that takes two std::strings representing two strands of DNA. Also, member functions penalty() that returns the cost of aligning the two characters given, min() that finds the minimum value of three integers, optDistance() which computes the optimization matrix and returns the value at [0][0] which is the optimal distance, alignment() that traces the matrix and returns a string that is the actual alignment of the two strands, and display_matirx() that outputs each element in the matrix. Along with four private variables opt, which is a 2D integer array that represents the matrix, sampleA and sampleB which are two std::string, M and N which are lengths sampleA and sampleB but also the dimensions of the matrix. When running the program, the user enters a text file that contains the two DNA strands. The program then outputs the edit distance, a table displaying the optimal alignment with the cost of the alignment, and the execution time of the program.

Design and Features

To find the alignment of the given pair of DNA samples, I created an MxN matrix (opt) with each element computed by finding the minimum value of three numbers. The first value is opt[i+1][j+1] divided by the penalty of aligning the two characters, found by passing the two characters into the penalty() function. Second is opt[i+1][j] + 2, and the third is [i][j+1] + 2. After the matrix is completed, the program iterates through the matrix and calculates the optimal alignment. It does this starting at opt[0][0] and moving diagonally, down, or right depending on the comparison of particular elements in the matrix, it does this unit [M][N] is reached. There are two main comparasions, pt[i][j] == opt[i + 1][j + 1]and opt[i][j] == opt[i + 1][j + 1] + 1. If thefirst comparison is true, there are three sub-comparisons checked, sampleA[i] == sampleB[j] means the characters are equal, opt[i][j] == opt[i + 1][j] + 2 and opt[i][j] == opt[i][j + 1] + 2 both mean they are not equal, and there is a gap inserted. If the second main comparison is true, then the characters are not equal, and no gap is inserted. During the described process, the 2 aligned strings are output to the screen along with the cost of the alignment (0 if equal, 1 if unequal characters, 2 if a character is aligned with a gap).

What I already knew and what I learned

Before completing this project, I used matrices and implemented them in multiple different ways, so iterating through the matrix and using different elements to solve a problem is something I am used to doing. I also knew how DNA alignment is performed, but I have never implemented it into a program.

I learned how to use dynamic programming to implement an algorithm. This project took a concept I am familiar with in the real world, so I found it interesting to create a program for it. It taught me that using programming skills and algorithms I am familiar with makes implementing real word problems simpler than I initially thought.

Below is a screenshot of a Sample DNA Alignment of the two DNA Strands:

Strand 1: AACAGTTACC Strand 2: TAAGGTCA Figure 5.

```
Edit Distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1
Execution time is 0.024209 seconds
```

Figure 5: Sample Alignment.

5.2 Codebase

```
CC = g++
 1
   CFLAGS = -g -Wall -Werror -pedantic -std=c++17
   LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
   DEPS = EDistance.hpp
 4
   OBJS = EDistance.o
 5
6
7
   %.o: %.cpp $(DEPS)
8
     $(CC) $(CFLAGS) -c $<
9
10
   .PHONY: all clean lint
11
   all: EDistance
12
13
14 EDistance: main.o $(OBJS)
     $(CC) $(CFLAGS) -0 $@ $^ $(LIBS)
15
16
17
   lint:
18
     cpplint *.cpp *.hpp
19
20
  clean:
21
     rm *.o EDistance
```

```
1
  // Copyright 2023 James Walsh
2
  #include <iostream>
3
  #include <fstream>
  #include <SFML/System.hpp>
4
  #include <SFML/Window.hpp>
5
   #include <SFML/Graphics.hpp>
6
7
   #include "EDistance.hpp"
8
9
   int main(int argc, char* argv[]) {
10
       std::ifstream file;
11
       std::string s1, s2;
       char* filename = argv[1];
12
```

```
13
        sf::Clock clock;
14
        sf::Time t;
15
16
        file.open(filename);
        std::getline(file, s1);
17
18
        std::getline(file, s2);
19
        EDistance myEDistance(s1, s2);
20
21
        std::cout << "Edit Distance = " <<myEDistance.optDistance() << std::endl</pre>
22
        std::cout << myEDistance.alignment();</pre>
23
        t = clock.getElapsedTime();
        std::cout << "Execution time is " << t.asSeconds() << " seconds\n";</pre>
24
25
26
        return 0;
27
   }
```

```
// Copyright 2023 James Walsh
   #pragma once
   #include <string>
3
 4
   #include <vector>
 5
   #include <algorithm>
6
 7
   class EDistance {
8
   public:
9
        EDistance(std::string stringA, std::string stringB);
10
        ~EDistance();
        static int penalty(char a, char b);
11
12
        static int min(int a, int b, int c);
13
        int optDistance();
14
        std::string alignment();
15
        void display_maxtrix();
16
    private:
17
        int** opt;
        std::string sampleA;
18
19
        std::string sampleB;
20
        int M;
21
        int N;
22 | };
```

```
// Copyright 2023 James Walsh
 2
   #include <algorithm>
3
   #include <iostream>
 4
   #include "EDistance.hpp"
   EDistance::EDistance(std::string stringA, std::string stringB):
6
 7
       sampleA(stringA), sampleB(stringB), M(stringA.size()), N(stringB.size())
        {
8
       opt = new int*[M + 2];
       for (int i = 0; i <= M; i++) {
9
10
           opt[i] = new int[N + 2];
11
12
   }
13
   int EDistance::penalty(char a, char b) {
14
15
       auto penalty_point = [a, b]() -> int {
16
           if (a == b) {
17
                return 0;
           } else {
18
19
                return 1;
```

```
20
21
        };
22
        return penalty_point();
   }
23
24
25
   int EDistance::min(int a, int b, int c) {
        if (a < b) {
26
            if (a < c) {
27
28
                return a;
29
            }
30
        } else {
31
            if (b < c) {
32
                return b;
33
34
        }
35
        return c;
36
   }
37
38
    int EDistance::optDistance() {
39
        int i, j;
40
41
        opt[M][N] = 0;
42
        for (i = 0; i < M; i++) {
43
            opt[i][N] = 2 * (M - i);
44
45
        for (j = 0; j < N; j++) {
            opt[M][j] = 2 * (N-j);
46
47
        for (i = M - 1; i >= 0; i--) {
48
49
            for (j = N - 1; j \ge 0; j--) {
50
                opt[i][j] = min(opt[i+1][j+1] + penalty(sampleA[i], sampleB[j]),
                    opt[i+1][j] + 2, opt[i][j+1] + 2);
51
52
            }
        }
53
54
55
        return opt[0][0];
   }
56
57
58
   std::string EDistance::alignment() {
59
        std::string optimal;
        int i = 0, j = 0;
60
61
        while (i <= M - 1 && j <= N - 1) {
62
63
            if (opt[i][j] == opt[i + 1][j + 1]) {
                if (sampleA[i] == sampleB[j]) {
64
                    optimal.push_back(sampleA[i]);
65
                    optimal.push_back(' ');
66
67
                    optimal.push_back(sampleB[j]);
                    optimal.push_back(' ');
68
69
                    optimal.push_back('0');
70
                    i++;
71
                    j++;
72
                    optimal.push_back('\n');
73
                else if (opt[i][j] == opt[i + 1][j] + 2) {
74
                    optimal.push_back(sampleA[i]);
                    optimal.push_back(' ');
75
76
                    optimal.push_back('-');
77
                    optimal.push_back(' ');
78
                    optimal.push_back('2');
```

```
79
                      i++;
80
                      optimal.push_back('\n');
                 } else if (opt[i][j] == opt[i][j + 1] + 2) {
81
82
                      optimal.push_back(sampleA[i]);
83
                      optimal.push_back(' ');
84
                      optimal.push_back('-');
                      optimal.push_back(' ');
85
86
                      optimal.push_back('2');
87
                      i++;
88
                      optimal.push_back('\n');
89
                 } else {
90
                      i++;
91
                      j++;
92
                      optimal.push_back('\n');
93
94
             } else if (opt[i][j] == opt[i + 1][j + 1] + 1) {
95
                 optimal.push_back(sampleA[i]);
96
                 optimal.push_back(' ');
97
                 optimal.push_back(sampleB[j]);
                 optimal.push_back(' ');
98
99
                 optimal.push_back('1');
100
                 i++;
101
                 j++;
102
                 optimal.push_back('\n');
103
             } else {
104
                 i++;
105
                 j++;
106
             }
107
         }
108
         return optimal;
109
    }
110
    EDistance::~EDistance() {
111
112
         int i;
113
         for (i = 0; i <= M; i++) {</pre>
114
             delete[] opt[i];
115
         }
116
         delete[] opt;
117
    }
118
119
    void EDistance::display_maxtrix() {
120
         int i, j;
121
         for (i = 0; i <= M; i++) {</pre>
122
             for (j = 0; j \le N; j++) {
123
                 std::cout.width(3); std::cout << opt[i][j] << " ";
124
             }
125
             std::cout << std::endl;</pre>
126
         }
127
    }
```

6 PS6: RandWriter

6.1 Discussion

What I accomplished

This project turns a given text into a random text that is somewhat reasonable and readable using a Markov model symbol table. To implement this, I created a RandWriter class. The object contains a constructor, five public member functions, and four private member variables. The constructor takes a std::string "text" and an int "k" as parameters, "text" is used as the original text and "k" is the length of each kgram. The orderK() is an accessor function that returns the value of the variable "K". The freq() function takes a std::string "kgram" as a parameter and returns the frequency of the given kgram in the symbol table. The freq() function is overloaded to take a std::string "kgram" and a char "c" and returns the frequency that c follows the kgram in the text. The kRand() function takes a std::string "kgram" and returns a random character that follows the given kgram. The random character is based on the frequency the given character follows the kgram in the original text. Lastly, the class contains a generate() function that takes a std::string "kgram" and an int "L". The function generates and returns a string of length "L" by simulating a trajectory through the symbol table or kgrams and kgrams+1. The class also overloads the insertion operator which prints out the symbol table to the given ostream.

Design and Features

When implementing this class, I decided to represent the symbol table as a std::map that contains a std::string and anotherstd::map which contains a char and int. The string represents each kgram, the map represents the next character and its frequency, and the frequency of the kgram is found by finding the total of the next character frequencies. I decided to implement the symbol table this way because I found it easy to navigate as well as able to store the kgram and kgram + 1 frequency well and accessible.

I implemented the insertion operator as a friend function, so I was able to access the object's private member variables directly, however, the function takes a const RandWriter object ensuring the function cannot modify the member variables.

When producing random numbers for the kRand function, I used the c++ library <random>. Then I stored a std::minstd_rand0 as a class member function. In the constructor, I created a seed and set it to the time. Lastly, in the kRand function, I declare a std::uniform_int_distribution<int> then pass the std::minstd_rand0 as a parameter to calculate the random number.

What I already knew and what I learned

Before programming this assignment, I had experience using std::map, so that is why I decided to implement the symbol table as so. I also have used std::string often, so I was aware of its member functions which were helpful for this assignment. I learned how to use the <random> library when completing this project. I have used rand() before so I am aware of the usage of seeds, but I have not used <random> to generate random numbers. I also learned a lot about how using frequencies of the next character of kgrams makes a random text drastically more reasonable. If I programmed a random text generator that did not generate the next character by using frequencies from the original text, the text generated would be unreadable. However, implementing frequencies into the random character generator makes the test readable.

Unit Testing

For this program, I wrote four different unit tests using the Boost Unit Test Framework. All tests created a RandWriter object with "gagggagagagagagagaaa" as the text and 1 as k. I tested all member functions of the calls, starting with orderK(). For this function, I simply checked if it returned the proper 'K' value, 1. I did this using BOOST_REQUIRE_EQUAL(). The purpose of the next three tests was to see if the functions

throw and did not throw a std::runtime_error when appropriate. These test tested both versions of freq(), kRand(), and generate(). To do this I used BOOST_REQUIRE_THROW() and BOOST_REQUIRE_NO_THROW(). My program passed all tests.

Below is a result of the generating function, using the first paragraph as the original text, 3 as "K" and 1000 as "L".

This, and reate() is a std::string "kgrams+1. The symbol table. The constring "k" as privated to a character function that constring "kgrameter is overloaded on geream. The variable used to this paracter class a given kgram in the freq() is based a param" and returns a Markov model symbol take and kgram. The frequency that follows the kRand returns a random character function the symbol text. Lastly, that take a trajector follows the the kgram, the kRand readable "K". The rand returns the given kgram" as a in table. The frequency of the object cons, an accessor functionstreate() functions, "text" and as the in the to a std::string a strints overloads the text. Lastly, the symbol text into table. The text and returns a std::structor takes a constring of the function generated to a string "text the to a std::string "kgram" and returns a std::structor which kgram in the ins that the rand readable and reated to a cons, "text" is out takes a std::structor follows that returns the class cont

6.2 Codebase

```
CC = g++
 1
   CFLAGS = -Wall -Werror -pedantic -std=c++17 -g
 2
   |LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
       lboost_unit_test_framework
   DEPS = RandWriter.hpp
   OBJS = RandWriter.o
 5
6
 7
   %.o: %.cpp $(DEPS)
     $(CC) $(CFLAGS) -c $<
8
9
10
   .PHONY: all clean lint
11
   all: TextWriter test
12
13
14
   TextWriter: TextWriter.o $(OBJS)
     $(CC) $(CFLAGS) -o $0 $^ $(LIBS)
15
16
17
   test: $(OBJS) test.o
18
     $(CC) $(FLAGS) -o test $^ $(LIBS)
19
20
21
     cpplint *.cpp *.hpp
22
23
  clean:
24
    rm *.o TextWriter test
```

```
1 // Copyright 2023 James Walsh
   |#include <iostream>
   #include <fstream>
   #include "RandWriter.hpp"
4
5
6
   int main(int argc, char* argv[]) {
7
       std::string text;
8
       std::string current_str;
9
       std::ifstream textFile;
10
       char* k_arr = argv[1];
       char* 1_arr = argv[2];
11
```

```
12
        char* filename = argv[3];
13
        int k = atoi(k_arr);
        int l = atoi(l_arr);
14
15
16
        textFile.open(filename);
        while (!textFile.eof()) {
17
            textFile >> current_str;
18
19
            text.append(current_str);
20
            text.push_back(' ');
21
            current_str.clear();
22
        }
23
        text.pop_back();
24
25
        RandWriter myWriter(text, k);
26
        std::string resulting_text;
27
        std::string first_gram;
28
        for (auto i = 0; i < k; i++) {
29
            first_gram.push_back(text.at(i));
30
        }
31
32
        resulting_text = myWriter.generate(first_gram, 1);
33
34
        std::cout << resulting_text << std::endl;</pre>
35
36
        return 0;
   }
37
```

```
// Copyright 2023 James Walsh
 2
   #include <string>
 3
   #include <map>
 4
   #include <random>
   class RandWriter {
 6
 7
    public:
 8
       // Create a Markov model of order k from given text
 9
       // Assume that text has length at least k.
10
       RandWriter(std::string text, int k);
11
12
       // Order k of Markov model
13
       int orderK() const { return K; }
14
15
       // Number of occurences of kgram in text
16
        // Throw an exception if kgram is not length k
17
       int freq(std::string kgram) const;
18
19
       // Number of times that character c follows kgram
        // if order=0, return num of times that char c appears
20
21
       // (throw an exception if kgram is not of length k)
22
       int freq(std::string kgram, char c) const;
23
24
       // Random character following given kgram
25
       // (throw an exception if kgram is not of length k)
26
        // (throw an exception if no such kgram)
27
       char kRand(std::string kgram);
28
29
       // Generate a string of length L characters by simulating a trajectory
30
       // through the corresponding Markov chain. The first k characters of
31
       // the newly generated string should be the argument kgram.
32
       // Throw an excpetion if kgram is not of length k.
       // Assume that L is at least k
33
```

```
35
36
       friend std::ostream& operator<<(std::ostream& Out, const RandWriter&
       object);
37
38
    private:
       std::map<std::string, std::map<char, int>> table;
39
40
       std::minstd_rand0 gen;
41
       int textLen;
42
       int K;
43
   };
44
   // Overload the stream insertion operator << and display the internal state
   // of the Markov model. Print out the order, alphabet, and the frequencies
45
46
   // of the k-grams and k+1-grams
47
48
   std::ostream& operator<<(std::ostream& out, const RandWriter& object);
 1 // Copyright 2023 James Walsh
   #include <iostream>
   #include <ios>
 3
 4
   #include <algorithm>
 5
   #include <numeric>
   #include <utility>
 6
 7
   #include <vector>
   #include <chrono>
 8
   #include <stdexcept>
 9
   #include "RandWriter.hpp"
10
11
12
   RandWriter::RandWriter(std::string text, int k):textLen(text.size()), K(k) {
13
       int i, j, c;
14
       std::string current_gram;
15
       std::map<std::string, std::map<char, int>>::iterator kgram;
16
       std::map<char, int>::iterator next_char;
17
       for (i = 0, j = 0; i < textLen - k; i++) {
18
19
            current_gram.clear();
20
            while (j < k) {
                c = text.at(i + j);
21
22
                if (c >= 0 \&\& c < 127) {
23
                    current_gram.push_back(text.at(i + j));
                }
24
                j++;
25
26
            }
27
            if (i == 0) {
28
                text.append(current_gram);
29
            }
30
            j--;
31
            kgram = table.find(current_gram);
32
            // if the table already contains the kgram
33
            if (kgram != table.end()) {
                next_char = (*kgram).second.find(text.at(i + j + 1));
34
35
                // if the kgram already contains the next possible char
36
                if (next_char != (*kgram).second.end()) {
                    (*next_char).second++;
37
38
                } else {
39
                // if the kgram doesnt contain the next possible char
40
                    (*kgram).second.insert(std::pair<char, int>(text.at(i + j +
       1), 1));
41
                }
            } else {
42
```

std::string generate(std::string kgram, int L);

34

```
43
                // if the table doesnt contain the kgram
44
                table.insert(std::pair<std::string,
45
                                 std::map<char, int>>(current_gram, std::map<char</pre>
       , int>()));
46
                kgram = table.find(current_gram);
                (*kgram).second.insert(std::pair<char, int>(text.at(i + j + 1),
47
       1));
48
49
            j = 0;
50
        }
51
        unsigned int seed = std::chrono::system_clock::now().time_since_epoch().
       count();
52
        gen = std::minstd_rand0(seed);
   }
53
54
   int RandWriter::freq(std::string kgram) const {
55
56
        std::map<std::string, std::map<char, int>>::const_iterator kgram_iter;
57
        std::map<char, int>::const_iterator next_char;
        int totalFreq = 0;
58
59
60
        if (static_cast<int>(kgram.size()) != K) {
61
            throw std::runtime_error("Invalid kgram length.");
62
63
        kgram_iter = table.find(kgram);
64
        if (kgram_iter != table.end()) {
65
            totalFreq = std::accumulate((*kgram_iter).second.begin(),
                (*kgram_iter).second.end(), 0, [](int &totalFreq, auto next_char
66
       ) {
67
                return totalFreq += next_char.second;
68
            });
69
        }
70
        return totalFreq;
   }
71
72
   int RandWriter::freq(std::string kgram, char c) const {
73
74
        std::map<std::string, std::map<char, int>>::const_iterator kgram_iter;
75
        std::map<char, int>::const_iterator next_char;
76
        int totalFreq = 0;
77
78
        if (static_cast<int>(kgram.size()) != K) {
79
            throw std::runtime_error("Invalid kgram length.");
80
81
        kgram_iter = table.find(kgram);
82
        if (kgram_iter != table.end()) {
83
            for (next_char = (*kgram_iter).second.begin();
                next_char != (*kgram_iter).second.end(); next_char++) {
84
                if ((*next_char).first == c) {
85
86
                    totalFreq = (*next_char).second;
87
                    return totalFreq;
88
                }
89
            }
90
        }
91
        return totalFreq;
   }
92
93
94
   char RandWriter::kRand(std::string kgram) {
95
        std::map<std::string, std::map<char, int>>::const_iterator kgram_iter;
96
        std::map<char, int>::const_iterator next_char;
        std::vector<char> char_list;
97
```

```
98
         int i;
99
         int krandNext = 0;
100
101
        kgram_iter = table.find(kgram);
102
         if (static_cast<int>(kgram.size()) != K) {
103
             throw std::runtime_error("Invalid kgram length.");
        }
104
105
         if (kgram_iter != table.end()) {
106
             for (next_char = (*kgram_iter).second.begin();
                     next_char != (*kgram_iter).second.end(); next_char++) {
107
108
                 for (i = 0; i < (*next_char).second; i++) {</pre>
109
                     char_list.push_back((*next_char).first);
110
                     krandNext++;
                 }
111
             }
112
113
             std::uniform_int_distribution<int> dist(0, krandNext - 1);
114
             krandNext = dist(gen);
115
             return char_list.at(krandNext);
116
        } else {
             throw std::runtime_error("No such kgram found.");
117
118
119
        return char();
120
    }
121
122
    std::string RandWriter::generate(std::string kgram, int L) {
123
         std::string result;
124
        std::string current_gram;
         int i, j;
125
126
        char c;
         if (static_cast<int>(kgram.size()) != K) {
127
128
             throw std::runtime_error("Invalid kgram length.");
129
        } else {
130
             result.append(kgram);
131
             c = kRand(kgram);
132
             result.push_back(c);
133
             for (i = 1; i < L - K; i++) {
134
                 for (j = 0; j < static_cast<int>(K); j++) {
135
                     current_gram.push_back(result.at(i + j));
136
                 }
137
                 c = kRand(current_gram);
138
                 current_gram.clear();
139
                 result.push_back(c);
             }
140
141
        }
142
        return result;
    }
143
144
145
    std::ostream& operator<<(std::ostream& out, const RandWriter& object) {
146
         std::map<std::string, std::map<char, int>>::const_iterator kgram;
147
        std::map<char, int>::const_iterator next_char;
148
        kgram = object.table.begin();
149
        for (kgram = object.table.begin(); kgram != object.table.end(); kgram++)
150
151
             out << (*kgram).first << std::endl;</pre>
152
             for (next_char = (*kgram).second.begin(); next_char != (*kgram).
        second.end(); next_char++) {
153
                 out << "\t" << (*next_char).first << " feq: " << (*next_char).
        second << std:: endl;</pre>
```

```
1
   // Copyright 2123 James Walsh
 2
3
   #include <iostream>
  #include <string>
 4
  #include <stdexcept>
 5
   #include "RandWriter.hpp"
6
 7
8
   #define BOOST_TEST_DYN_LINK
9
   #define BOOST_TEST_MODULE Main
   #include <boost/test/unit_test.hpp>
10
11
   BOOST_AUTO_TEST_CASE(test_orderK) {
12
13
       14
       BOOST_REQUIRE_EQUAL(test.orderK(), 1);
15
   }
16
17
   BOOST_AUTO_TEST_CASE(test_freq) {
       RandWriter test("gagggagagagagagagagaaa", 1);
18
       BOOST_REQUIRE_THROW(test.freq("ga"), std::runtime_error);
19
20
       BOOST_REQUIRE_NO_THROW(test.freq("g"));
21
       BOOST_REQUIRE_EQUAL(test.freq("g"), 9);
22
       BOOST_REQUIRE_THROW(test.freq("ga", 'g'), std::runtime_error);
       BOOST_REQUIRE_NO_THROW(test.freq("g", 'g'));
23
24
       BOOST_REQUIRE_EQUAL(test.freq("g", 'a'), 5);
   }
25
26
27
   BOOST_AUTO_TEST_CASE(test_kRand) {
28
       29
       BOOST_REQUIRE_THROW(test.kRand("ga"), std::runtime_error);
30
       BOOST_REQUIRE_THROW(test.kRand("b"), std::runtime_error);
31
       BOOST_REQUIRE_NO_THROW(test.kRand("g"));
   }
32
33
34
   BOOST_AUTO_TEST_CASE(test_generate) {
35
       RandWriter test("gagggagagggagagaaa", 1);
36
       BOOST_REQUIRE_THROW(test.generate("ga", 10), std::runtime_error);
37
       BOOST_REQUIRE_NO_THROW(test.generate("g", 10));
38
       BOOST_REQUIRE_EQUAL(test.generate("g", 10).size(), 10);
   }
39
```

7 PS7: Kronos Log Parsing

7.1 Discussion

What I accomplished

The purpose of this project review InTouch log(s) of a Kronos InTouch device. A log file contains information about the operations of the device. The program scans the files looking for lines that indicate a boot-up, which looks like (log.c.166) server started. After the program looks for a line that marks the completion of the boot-up sequence, the line will include oejs.AbstractConnector:Started SelectChannelConnector. Both of these lines will also include the log file line number and the date and time of the particular boot-up or completion. A successful boot will start with a start message and will be followed by a completion message. However, this is not always the case. If a start message is followed by another start without a completion, this indicates an "incomplete boot".

Design and Features

My program scans the file searching for these indicators and creates a report file tracking each successful and unsuccessful boot. To do this I used regex expressions. I had two regex expressions, one for start and one for complete. While the program had not reached the end of the file, it reads the line, then used the regex_search() function to see if the line was a start or completion. If it was, the report is updated with the date and time of the start or completion. To indicate if the boot was complete or incomplete, I used a boolean in_progress. The boolean was set to true when a start line is found. Then is set to false when a completion line is found. However, if a start line is found while in_progress is true, the report file marks the current boot as incomplete. After a complete boot, the report file marks the time difference from the start line to the completion line as the "boot time", but my program does not implement the boot time algorithm properly. The report file uses the name of the log file but apends".rpt" to the end.

```
The regex expressions I used were:
Start boot-up: std::regex start("(log.c.d{1,3}) server started")
Completion of boot: std::regex testC("oejs.AbstractConnector:Started
SelectChannelConnector")
```

What I already knew and what I learned

Going into this project I was familiar with input and output files. I have created programs that opened files, read until the end of the file, and done a variety of things with the data acquired many times. Also, I have outputted data to different files in various formats as well.

When completing this project I learned and gained a lot of knowledge on using regex expressions. Regex is something brand knew to me so it took me time to get used to how an expression works, and how I could then use that expression to perform some time of a task. I found the library relatively easy to use, and I found implementing them into my code to perform a task straightforward. I can see regex expressions being very useful for me in the future, and will definitely be something I find myself using when programming.

Below is an example result file of a Complete and Incomplete boot: Figure 6.

```
=== Device boot ===

4(device5_intouch.log): 2013-05-04 05:28:13 Boot Start

**** Incomplete boot ****

=== Device boot ===

31063(device5_intouch.log): 2014-01-26 09:55:07 Boot Start

31176(device5_intouch.log): 2014-01-26 09:58:04 Boot Complete

Boot time: 0
```

Figure 6: Sample.

7.2 Codebase

```
1 | CC = g++
 2
  CFLAGS = --std=c++17 -Wall -Werror -pedantic
3
   LIB = -lboost_unit_test_framework -lboost_regex
5
   .PHONY: all clean lint
6
7
   all: ps7
8
9
   %.o: %.cpp $(DEPS)
10
    $(CC) $(CFLAGS) -c $<
11
12
   ps7: main.o
13
     $(CC) $(CFLAGS) -0 $@ $^ $(LIB)
14
   clean:
15
16
    rm *.o ps7
17
18 | lint:
19
     cpplint *.cpp *.hpp
```

```
// Copyright 2023 James Walsh
 1
   #include <iostream>
   #include <fstream>
 3
 4 #include <string>
   #include <boost/regex.hpp>
 5
 6
   #include "boost/date_time/gregorian/gregorian.hpp"
 7
 8
   char* create_output_file(char* input);
 9
10
   int main(int argc, char* argv[]) {
     std::ifstream inputFile;
11
12
     std::ofstream outputFile;
     char* filename = argv[1];
13
14
     char* output = create_output_file(filename);
15
16
     inputFile.open(filename);
17
     outputFile.open(output);
18
19
     boost::regex start("\\(log\\.c\\.\\d\{1,3\}\\) server started");
20
     boost::regex complete(
21
        "\\oejs\\.AbstractConnector:Started SelectChannelConnector");
22
23
     std::string date = "";
24
     std::string time = "";
25
     std::string startTime;
26
     std::string curretLog = "";
27
     std::string log;
28
     int lineNum = 1;
29
     int bootTime = 0;
30
     bool in_progress;
     bool first = true;
31
32
     int i;
33
     char c;
34
     while (!inputFile.eof()) {
35
36
        char line[1000];
37
        inputFile.getline(line, 999);
        i = 28;
38
```

```
39
        if (regex_search(line, start) && first) {
40
          c = line[i];
          while (c != ')') {
41
42
            curretLog.push_back(c);
43
            i++;
            c = line[i];
44
          }
45
46
          i = 0;
47
          c = line[i];
48
          while (c != ' ') {
49
            date.push_back(c);
50
            i++;
            c = line[i];
51
          }
52
          i++;
53
          c = line[i];
54
          while (c != ' ') {
55
56
            time.push_back(c);
57
            i++;
            c = line[i];
58
          }
59
60
          time.pop_back();
61
          startTime = time;
62
          outputFile << "=== Device boot ===" << std::endl;</pre>
63
          outputFile << lineNum << "(" << filename << "): ";</pre>
          outputFile << date << " " << time << " Boot Start" << std::endl;</pre>
64
          std::cout << line << std::endl;</pre>
65
66
          first = false;
67
          in_progress = true;
68
        } else if (regex_search(line, start) && !in_progress) {
69
          c = line[i];
          while (c != ')') {
70
71
            curretLog.push_back(c);
            i++;
72
73
            c = line[i];
          }
74
          i = 0;
75
          c = line[i];
76
          while (c != ' ') {
77
78
            date.push_back(c);
79
            i++;
80
            c = line[i];
          }
81
82
          i++;
83
          c = line[i];
          while (c != ' ') {
84
85
            time.push_back(c);
            i++;
86
87
            c = line[i];
88
          }
89
          time.pop_back();
90
          startTime = time;
          outputFile << "=== Device boot ===" << std::endl;</pre>
91
          outputFile << lineNum << "(" << filename << "): ";</pre>
92
          outputFile << date << " " << time << " Boot Start" << std::endl;</pre>
93
94
          std::cout << line << std::endl;</pre>
95
          in_progress = true;
        } else if (regex_search(line, start) && in_progress) {
96
97
          c = line[i];
```

```
98
           while (c != ')') {
99
             log.push_back(c);
100
             i++;
101
             c = line[i];
           }
102
           if (curretLog != log) {
103
104
             i = 0;
             c = line[i];
105
106
             while (c != ' ') {
107
               date.push_back(c);
108
               i++;
109
               c = line[i];
             }
110
111
             i++;
             c = line[i];
112
113
             while (c != ' ') {
114
               time.push_back(c);
115
               i++;
               c = line[i];
116
             }
117
118
             time.pop_back();
119
             startTime = time;
120
             outputFile << "**** Incomplete boot ****" << std::endl << std::endl;</pre>
121
             outputFile << "=== Device boot ===" << std::endl;</pre>
122
             outputFile << lineNum << "(" << filename << "): ";</pre>
             outputFile << date << " " << time << " Boot Start" << std::endl;</pre>
123
124
             std::cout << line << std::endl;</pre>
125
             in_progress = true;
126
             curretLog = log;
127
           }
128
         }
129
         if (regex_search(line, complete) && in_progress) {
130
           i = 0;
           c = line[i];
131
           while (c != ' ') {
132
133
             date.push_back(c);
134
             i++;
135
             c = line[i];
136
           }
137
           i++;
138
           c = line[i];
           while (c != '.') {
139
140
             time.push_back(c);
141
             i++;
142
             c = line[i];
           }
143
           outputFile << lineNum << "(" << filename << "): ";</pre>
144
           outputFile << date << " " << time << " Boot Complete" << std::endl;</pre>
145
           outputFile << "\tBoot time: " << bootTime << std::endl << std::endl;</pre>
146
147
           std::cout << line << std::endl;</pre>
148
           in_progress = false;
149
           curretLog.clear();
         }
150
151
         lineNum++;
152
         date.clear();
153
         time.clear();
154
         log.clear();
155
156
       return 0;
```

```
157
158
159
    char* create_output_file(char* input) {
160
    std::string temp;
      temp.append(input);
161
162
      temp.append(".rpt");
      char* out = new char[temp.size()];
163
164
      int i = 0;
      std::for_each(temp.begin(), temp.end(), [&out, &i] (char c) {
165
166
        out[i] = c;
167
        i++;
168
      });
169
170
      return out;
171
   }
```