COMP 4 SEC 201 PORTFOLIO

Ajay Alamuri

Spring 2023

Contents:

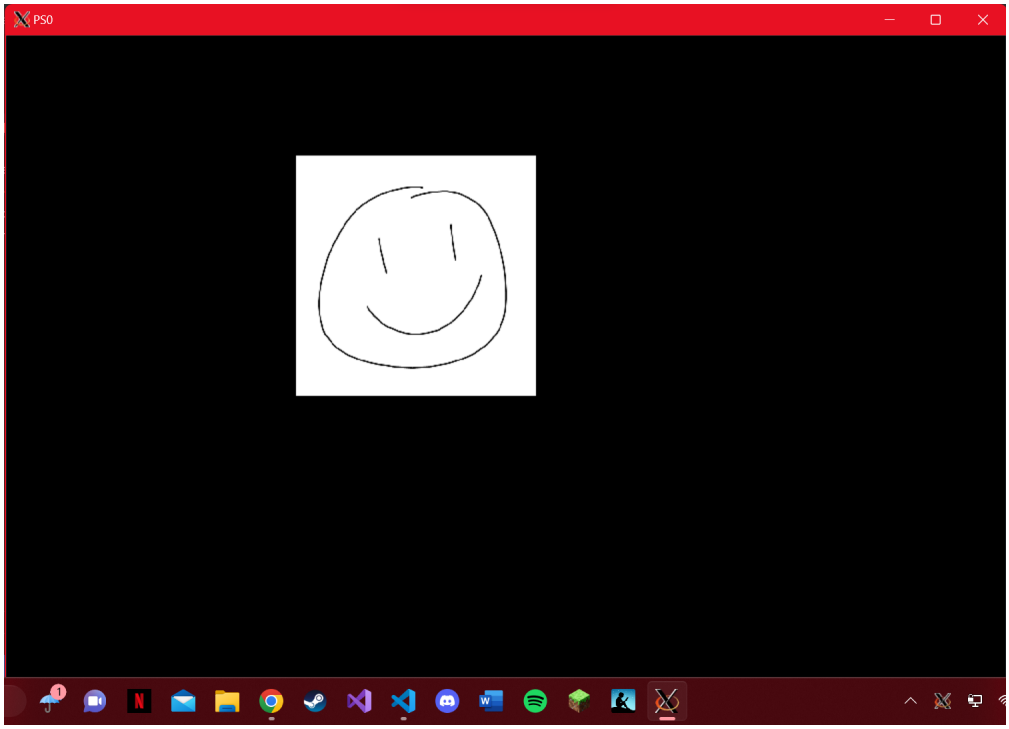
Time To Complete Portfolio: 8 hours

1 PS0 Hello World with SFML

1.1 Discussion:

This program was meant to act as our introduction to the SFML Library. Having not worked with graphics APIs before, SFML was very new to me. Many of the following assignments used SFML, so I spent a lot of time in the documentation trying to learn. I was able to learn how to create my own sprites and utilize them in a window, starting with loading in an image, converting it to a texture, and using that as a sprite. I also learned how to interact with the sprite using user inputs such as keys which became very important later on. Although the assignment was frustrating at first since I couldn't just mess with the object however I wanted. Getting used to using accessors and mutators was a very important skill to learn due to that being extremely important for almost all Object Oriented Programming.

1.2 Output:



1.3 Codebase:

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

.PHONY: all clean lint

all: sfml-app

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

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

clean:
  rm *.o sfml-app

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

```
//AJAY ALAMURI

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

int main()
{
    sf::RenderWindow window(sf::VideoMode(1000, 800), "PS0");

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

    float x_pos = 100;
    float y_pos = 100;
    float cur_scale = 0.3;

    sf::Texture texture;
    texture.loadFromFile("sprite.png");

    sf::Sprite sprite(texture);
    sprite.setScale(cur_scale, cur_scale);
    sprite.setPosition(x_pos, y_pos);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::KeyPressed){
                switch(event.key.code){
```

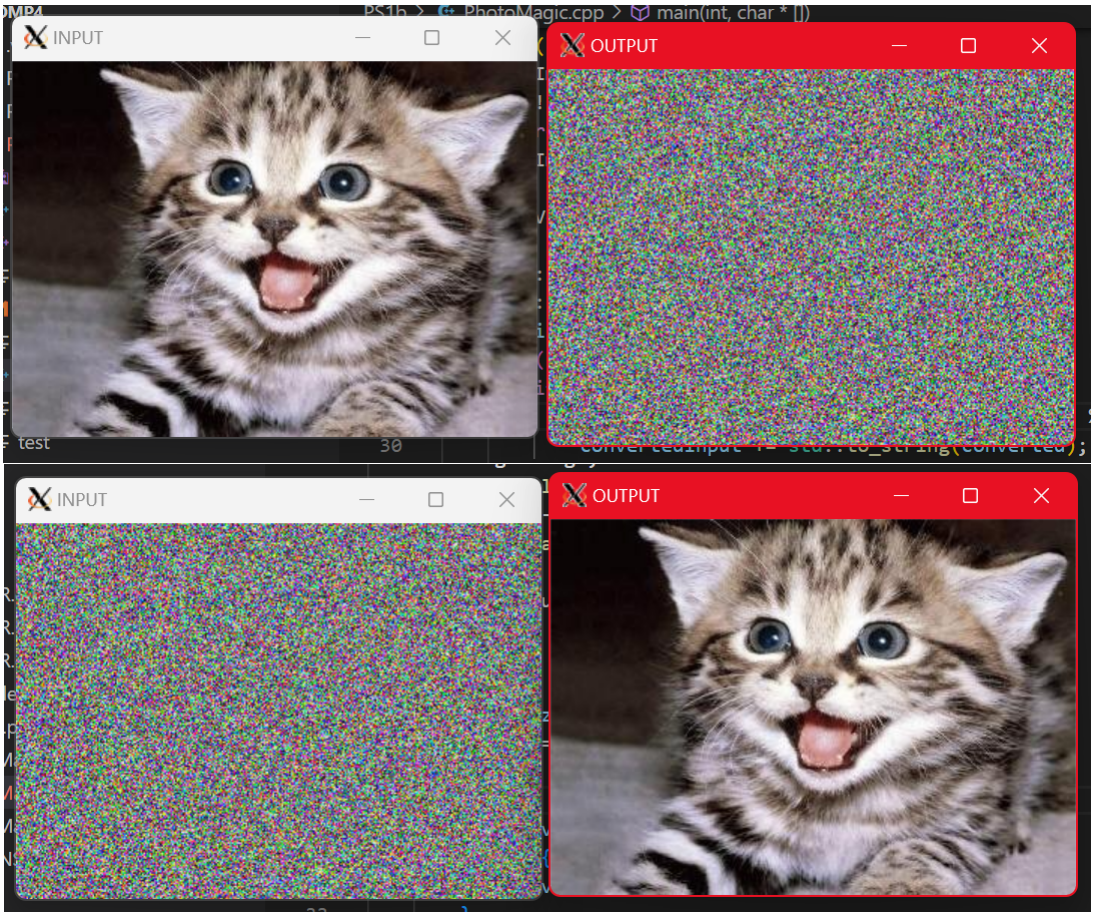
```
                    case sf::Keyboard::W:
                        y_pos -= 10;
                        break;
                    case sf::Keyboard::A:
                        x_pos -= 10;
                        break;
                    case sf::Keyboard::S:
                        y_pos += 10;
                        break;
                    case sf::Keyboard::D:
                        x_pos += 10;
                        break;
                    case sf::Keyboard::Up:
                        cur_scale += 0.01;
                        break;
                    case sf::Keyboard::Down:
                        cur_scale -= 0.01;
                        break;
                    default:
                        break;
                }
            }
        }

        sprite.setPosition(x_pos, y_pos);
        sprite.setScale(cur_scale, cur_scale);

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

    return 0;
}
```

2 PS1 Linear Feedback Shift Register and Image Encoding

2.1 Discussion:

This project focused on combined usage of the SFML library from PS0 and the usage of a Linear Feedback Shift Register to encode an image. The first step of the program was to create a functioning Fibonnaci Linear Feedback Shift Register (abbreviated to FibLFSR). By using a string of 1's and 0's to create a seed, I implemented a function to step which would cycle the stored string and return the new bit that was added to the string. The other function was the generate function which would take the amount of bits that would be used to make a new number, this new number would be created by each of the bits that come out of sequential step functions. The next step was to use the FibLFSR to encode an image. By using the SFML library to take in an image, the transform function would cycle through each pixel of the image and XOR each of the RGB components with a new number that came from the generate function of the FibLFSR. This created an encoded image that could be reversed given the same seed since a double XOR results in the original number. This program also helped me work on my Object Oriented programming skills since I had to use the functions from the SFML library to work with each of the pixels. This was really simple once I read through the documentation more but before that I faced a lot of trouble because I assumed that I would be able to just mess with the pixels, not the subcomponents of each pixel.

2.2 Output:



2.3 Codebase:

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

.PHONY: lint all clean

all: lint test PhotoMagic

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

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

test: test.o FibLFSR.o transform.o
  $(CC) $(CFLAGS) -o $@ $^ $(LIB)
```

```

clean:
  rm *.o PhotoMagic test
lint:
  cpplint *.cpp *.hpp
```

```
// Copyright 2023 Ajay

// pixels.cpp:
// using SFML to load a file, manipulate its pixels, write it to disk


// g++ -o pixels pixels.cpp -lsfml-graphics -lsfml-window

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

#include "FibLFSR.hpp"
#include "transform.hpp"

int main(int argc, char *argv[]) {
    sf::Image image;
    if (!image.loadFromFile(argv[1]))
        return -1;
    sf::Image base(image);

    sf::Vector2u size = image.getSize();

    std::string input(argv[3]);
    std::string convertedInput;
    int inputSize = input.size();
    for (int i = 0; i < 16; i++) {
        if (i < inputSize) {
            int converted = (static_cast<int>(input[i]) % 2);
            convertedInput += std::to_string(converted);
        } else {
            convertedInput += '0';
        }
    }
    FibLFSR seed(convertedInput);
    transform(image, &seed);

    sf::RenderWindow window1(sf::VideoMode(size.x, size.y), "INPUT");
    sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "OUTPUT");

    sf::Texture texture;
    texture.loadFromImage(image);
    sf::Texture org;
    org.loadFromImage(base);

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

```

```
    while (window1.isOpen() && window2.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window1.close();
        }
        while (window2.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window2.close();
        }
        window1.clear();
        window1.draw(originalSprite);
        window1.display();
        window2.clear();
        window2.draw(sprite);
        window2.display();
    }

    if (!image.saveToFile(argv[2]))
        return -1;

    return 0;
}
```

```
// Copyright 2023 Ajay

#include <algorithm>

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

#include "FibLFSR.hpp"

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

```
// Copyright 2023 Ajay

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

#include "FibLFSR.hpp"
#include "transform.hpp"

void transform(sf::Image& image, FibLFSR* seed) {
    // p is a pixelimage.getPixel(x, y);
    sf::Color p;

    sf::Vector2u size = image.getSize();

    for (unsigned x = 0; x < size.x; x++) {
        for (unsigned y = 0; y < size.y; y++) {
            p = image.getPixel(x, y);
            p.r = p.r ^ seed->generate(8);
            p.g = p.g ^ seed->generate(8);
```

```
            p.b = p.b ^ seed->generate(8);
            image.setPixel(x, y, p);
        }
    }
}
```

```
// Copyright 2023 Ajay
#pragma once

#include <iostream>
#include <string>

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

class FibLFSR {
 public:
    // Constructor to create LFSR with the given initial seed
    explicit FibLFSR(std::string seed = "0000000000000001"): bitstring(seed)
     {}
    // Simulate one step and return the new bit as 0 or 1
    int step();
    // Simulate k steps and return a k-bit integer
    int generate(int k);

    friend std::ostream& operator<<(std::ostream&, const FibLFSR& lfsr);
 private:
    std::string bitstring;
};

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

```
// Copyright 2023 Ajay

#include <string>
#include <cmath>

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

#include "FibLFSR.hpp"

int FibLFSR::step() {
    int result = bitstring[0] ^ bitstring[2] ^ bitstring[3] ^ bitstring[5];
    bitstring += std::to_string(result);
    bitstring = bitstring.substr(1);
    return result;
}

int FibLFSR::generate(int k) {
    int num = 0;
    for (int i = 0; i < k; i++) {
        num *= 2;
        int result = step();
```

```
        if (result == 1) {
            num += result;
        }
    }
    return num;
}

std::ostream& operator<<(std::ostream& out, const FibLFSR& lfsr) {
    out << lfsr.bitstring;
    return out;
}
```

```
// Copyright 2022
// By Dr. Rykalova
// Editted by Dr. Daly
// test.cpp for PS1a
// updated 5/12/2022

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

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

#include "FibLFSR.hpp"
#include "transform.hpp"

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

BOOST_AUTO_TEST_CASE(testStepInstr1) {
  FibLFSR l("1011011000110110");
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 0);
  BOOST_REQUIRE_EQUAL(l.step(), 1);
}

BOOST_AUTO_TEST_CASE(testStepInstr2) {
  FibLFSR l2("1011011000110110");
  BOOST_REQUIRE_EQUAL(l2.generate(9), 51);
}
// TEST CASE 1
BOOST_AUTO_TEST_CASE(testStepAndGenerate) {
    FibLFSR first("1001000000000000");
    BOOST_REQUIRE_EQUAL(first.step(), 0);
    BOOST_REQUIRE_EQUAL(first.step(), 1);
    BOOST_REQUIRE_EQUAL(first.step(), 0);
    BOOST_REQUIRE_EQUAL(first.generate(3), 4);
```

```
45 }
46 // TEST CASE 2
47 BOOST_AUTO_TEST_CASE(stepGenerateOut) {
48     FibLFSR second("1000000000000000");
49     BOOST_REQUIRE_EQUAL(second.step(), 1);
50     BOOST_REQUIRE_EQUAL(second.generate(5), 0);
51     std::stringstream s;
52     s << second;
53     BOOST_REQUIRE_EQUAL(s.str(), "0000000000100000");
54 }
```

3 PS2 Sokoban

3.1 Discussion:

Sokoban was the first big project that I did in this class. The objective of this program was to make a block pushing game that took text files which had characters that outlined the starting positions of everything in a level. This project gave me a lot of trouble at first due to trouble working with positioning of sprites. I frequently ended up with all the sprites stacked on one another or none of the textures properly loading. Once I got that figured out, the rest of the program was actually quite easy, since all I needed to do was pull together the stuff I learned in PS0 to enable movement and then create a win screen. One problem that I fixed was that the window would be too big and go off screen bounds so I made the program automatically detect the screen size and decrease its screen size. Although I didn't use a matrix, I did use arrays as a form of pseudo-matrix which made it easier to store each sprite with it's positional values, however since I used arrays, I had to dynamically allocate memory which was very difficult at first but also allowed me to get real experience using destructors. This project would have been a lot easier if I used vectors or the provided matrix class, but now I know how to do it the hard way if necessary.

3.2 Output:

3.3 Codebase:

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

.PHONY: lint all clean

all: lint Sokoban

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

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

clean:
  rm *.o Sokoban
```

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

```
// Copyright 2023 Ajay

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

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

#include "Sokoban.hpp"

int main(int argc, char* argv[]) {
    Sokoban game;
    if (argc > 1) {
        std::string given = argv[1];
        std::stringstream input;
        input << given;
        std::istream in(input.rdbuf());
        in >> game;
    }

    game.loadLevel();

    game.shrink();

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

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::KeyPressed) {
                switch (event.key.code) {
                    case sf::Keyboard::W:
                    case sf::Keyboard::Up:
                        game.tryMove(UP);
                        break;
                    case sf::Keyboard::A:
                    case sf::Keyboard::Left:
                        game.tryMove(LEFT);
                        break;
                    case sf::Keyboard::S:
                    case sf::Keyboard::Down:
                        game.tryMove(DOWN);
                        break;
                    case sf::Keyboard::D:
                    case sf::Keyboard::Right:
                        game.tryMove(RIGHT);
                        break;
                    case sf::Keyboard::R:
```

```
                        game.loadLevel();
                    default:
                        break;
                }
            }
        }
        window.clear();
        window.draw(game);
        window.display();
        if (game.isWon() == 1) {
            sf::Image iWon;
            if (!iWon.loadFromFile("sources/win.png"))
                throw std::runtime_error("Image could not be opened!");
            sf::Texture tWon;
            tWon.loadFromImage(iWon);
            sf::Sprite sWon;
            sWon.setTexture(tWon);
            unsigned h = 1200;
            unsigned w = 800;
            while (window.getSize().x < h || window.getSize().y < w) {
                h /= 2;
                w /= 2;
                sWon.scale(0.5, 0.5);
            }
            sWon.setPosition(0, 0);
            window.clear();
            window.draw(game);
            window.draw(sWon);
            window.display();
            while (window.waitEvent(event)) {
                if (event.type == sf::Event::Closed)
                    window.close();
            }
        }
    }
    return 0;
}
```

```
// Copyright 2023 Ajay

#pragma once

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

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

typedef enum movement {UP, LEFT, DOWN, RIGHT} Movement;

class Sokoban: public sf::Drawable {
 public:
    Sokoban();
    int getWidth() const { return _width; }
```

```
    int getHeight() const { return _height; }
    int getPx() const { return _px; }
    void loadLevel();
    void shrink();
    void tryMove(Movement);
    friend std::istream& operator>>(std::istream& in, Sokoban& obj);
    ~Sokoban();
    const int isWon();

 private:
    std::string _level;
    int _px;
    int _width;
    int _height;
    sf::Sprite* _spriteArr;
    sf::Sprite* _mvArr;
    sf::Texture _wall;
    sf::Texture _ground;
    sf::Texture _goal;
    sf::Texture _box;
    sf::Texture _front;
    sf::Texture _back;
    sf::Texture _right;
    sf::Texture _left;
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
   const;
};
```

```
// Copyright 2023 Ajay

#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <iterator>

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

#include "Sokoban.hpp"

Sokoban::Sokoban() {
    std::string lvl = "sources/level1.lvl";
    _level = lvl;
    _px = 64;
    _spriteArr = new sf::Sprite[1];
    _mvArr = new sf::Sprite[1];
    sf::Image wall, ground, goal, box, front, back, right, left;
    if (!wall.loadFromFile("sources/block_06.png"))
        throw std::runtime_error("Image could not be opened!");
    if (!ground.loadFromFile("sources/ground_01.png"))
        throw std::runtime_error("Image could not be opened!");
    if (!goal.loadFromFile("sources/ground_04.png"))
        throw std::runtime_error("Image could not be opened!");
    if (!box.loadFromFile("sources/crate_03.png"))
```

```
29          throw std::runtime_error("Image could not be opened!");
30      if (!front.loadFromFile("sources/player_05.png"))
31          throw std::runtime_error("Image could not be opened!");
32      if (!back.loadFromFile("sources/player_08.png"))
33          throw std::runtime_error("Image could not be opened!");
34      if (!right.loadFromFile("sources/player_17.png"))
35          throw std::runtime_error("Image could not be opened!");
36      if (!left.loadFromFile("sources/player_20.png"))
37          throw std::runtime_error("Image could not be opened!");
38      sf::Texture tWall, tGround, tGoal, tBox, tFront, tBack, tRight, tLeft;
39      tWall.loadFromImage(wall);
40      tGround.loadFromImage(ground);
41      tGoal.loadFromImage(goal);
42      tBox.loadFromImage(box);
43      tFront.loadFromImage(front);
44      tBack.loadFromImage(back);
45      tRight.loadFromImage(right);
46      tLeft.loadFromImage(left);
47      _wall = tWall;
48      _ground = tGround;
49      _goal = tGoal;
50      _box = tBox;
51      _front = tFront;
52      _back = tBack;
53      _right = tRight;
54      _left = tLeft;
55  }
56
57  std::istream& operator>>(std::istream& in, Sokoban& obj) {
58      std::string lvl = "sources/";
59      std::string input;
60      in >> input;
61      lvl += input;
62      obj._level = lvl;
63      return in;
64  }
65
66  void Sokoban::loadLevel() {
67      delete[] _spriteArr;
68      delete[] _mvArr;
69      std::ifstream fp(_level);
70      if (!fp.is_open())
71          throw std::runtime_error("File could not be opened!");
72      fp >> _height;
73      fp >> _width;
74      char* array = new char[_width * _height];
75      char* curLine = new char[_width];
76      for (int j = 0; j < _height; j++) {
77          fp >> curLine;
78          for (int i = 0; i < _width; i++) {
79              array[i + (j * _width)] = curLine[i];
80          }
81      }
82      delete[] curLine;
83      fp.close();
```

```
    sf::Sprite* sprArr = new sf::Sprite[_width * _height];
    for (int height = 0; height < _height; height++) {
        for (int width = 0; width < _width; width++) {
            char key = array[width + (height * _width)];
            sf::Sprite sprite;
            if (key == '#') {
                sprite.setTexture(_wall);
            } else if (key == 'a' || key == '1') {
                sprite.setTexture(_goal);
            } else {
                sprite.setTexture(_ground);
            }
            sprite.setPosition((width * 64), (height * 64));
            sprArr[width + (height * _width)] = sprite;
        }
    }
    _spriteArr = sprArr;
    sf::Sprite* moveArr = new sf::Sprite[_width * _height];
    for (int height = 0; height < _height; height++) {
        for (int width = 0; width < _width; width++) {
            if (array[width + (height * _width)] == '@') {
                sf::Sprite sprite;
                sprite.setTexture(_front);
                sprite.setPosition((width*64), (height*64));
                moveArr[width + (height * _width)] = sprite;
            } else if (array[width + (height * _width)] == 'A' ||
                array[width + (height * _width)] == '1') {
                sf::Sprite sprite;
                sprite.setTexture(_box);
                sprite.setPosition((width*64), (height*64));
                moveArr[width + (height * _width)] = sprite;
            }
        }
    }
    _mvArr = moveArr;
}

void Sokoban::tryMove(Movement direction) {
    int location;
    for (int i = 0; i < (_width * _height); i++) {
        if (_mvArr[i].getTexture() == &_front ||
        _mvArr[i].getTexture() == &_back ||
        _mvArr[i].getTexture() == &_right ||
        _mvArr[i].getTexture() == &_left) {
            location = i;
        }
    }
    switch (direction) {
        case UP:
            if (_spriteArr[location - _width].getTexture() == &_wall ||
            (_spriteArr[location].getPosition().y - _px) < 0) {
                break;
            }
            if (_mvArr[location - _width].getTexture() == &_box) {
                if (_spriteArr[location - (2 * _width)].getTexture() == &
```

```
    _wall ||
                _mvArr[location - (2 * _width)].getTexture() == &_box ||
                (_spriteArr[location].getPosition().y - (2 * _px)) < 0) {
                    break;
                }
                _mvArr[location - _width].move(0, -_px);
                _mvArr[location - (2 * _width)] = _mvArr[location - _width];
                _mvArr[location].move(0, -_px);
                _mvArr[location].setTexture(_back);
                _mvArr[location - _width] = _mvArr[location];
                sf::Sprite sprite;
                _mvArr[location] = sprite;
            } else {
                _mvArr[location].move(0, -_px);
                _mvArr[location].setTexture(_back);
                _mvArr[location - _width] = _mvArr[location];
                sf::Sprite sprite;
                _mvArr[location] = sprite;
            }
            break;
        case LEFT:
            if (_spriteArr[location - 1].getTexture() == &_wall ||
            (_spriteArr[location].getPosition().x - _px) <= -_px) {
                break;
            }
            if (_mvArr[location - 1].getTexture() == &_box) {
                if (_spriteArr[location - 2].getTexture() == &_wall ||
                _mvArr[location - 2].getTexture() == &_box ||
                (_spriteArr[location].getPosition().x - (2 * _px)) <= -_px)
    {
                    break;
                }
                _mvArr[location - 1].move(-_px, 0);
                _mvArr[location - 2] = _mvArr[location - 1];
                _mvArr[location].move(-_px, 0);
                _mvArr[location].setTexture(_left);
                _mvArr[location - 1] = _mvArr[location];
                sf::Sprite sprite;
                _mvArr[location] = sprite;
            } else {
                _mvArr[location].move(-_px, 0);
                _mvArr[location].setTexture(_left);
                _mvArr[location - 1] = _mvArr[location];
                sf::Sprite sprite;
                _mvArr[location] = sprite;
            }
            break;
        case DOWN:
            if (_spriteArr[location + _width].getTexture() == &_wall ||
            (_spriteArr[location].getPosition().y + _px) >= _height * _px) {
                break;
            }
            if (_mvArr[location + _width].getTexture() == &_box) {
                if (_spriteArr[location + (2 * _width)].getTexture() == &
    _wall ||
```

```
191                 _mvArr[location + (2 * _width)].getTexture() == &_box ||
192                 (_spriteArr[location].getPosition().y + (2 * _px)) >=
    _height * _px) {
193                     break;
194                 }
195                 _mvArr[location + _width].move(0, _px);
196                 _mvArr[location + (2 * _width)] = _mvArr[location + _width];
197                 _mvArr[location].move(0, _px);
198                 _mvArr[location].setTexture(_front);
199                 _mvArr[location + _width] = _mvArr[location];
200                 sf::Sprite sprite;
201                 _mvArr[location] = sprite;
202             } else {
203                 _mvArr[location].move(0, _px);
204                 _mvArr[location].setTexture(_front);
205                 _mvArr[location + _width] = _mvArr[location];
206                 sf::Sprite sprite;
207                 _mvArr[location] = sprite;
208             }
209             break;
210         case RIGHT:
211             if (_spriteArr[location + 1].getTexture() == &_wall ||
212             (_spriteArr[location].getPosition().x + _px) >= _width * _px) {
213                 break;
214             }
215             if (_mvArr[location + 1].getTexture() == &_box) {
216                 if (_spriteArr[location + 2].getTexture() == &_wall ||
217                 _mvArr[location + 2].getTexture() == &_box ||
218                 (_spriteArr[location].getPosition().x + (2 *_px)) >= _width
    * _px) {
219                     break;
220                 }
221                 _mvArr[location + 1].move(_px, 0);
222                 _mvArr[location + 2] = _mvArr[location + 1];
223                 _mvArr[location].move(_px, 0);
224                 _mvArr[location].setTexture(_right);
225                 _mvArr[location + 1] = _mvArr[location];
226                 sf::Sprite sprite;
227                 _mvArr[location] = sprite;
228             } else {
229                 _mvArr[location].move(_px, 0);
230                 _mvArr[location].setTexture(_right);
231                 _mvArr[location + 1] = _mvArr[location];
232                 sf::Sprite sprite;
233                 _mvArr[location] = sprite;
234             }
235             break;
236         default:
237             break;
238     }
239 }
240
241 const int Sokoban::isWon() {
242     sf::Sprite* spr;
243     spr = _spriteArr;
```

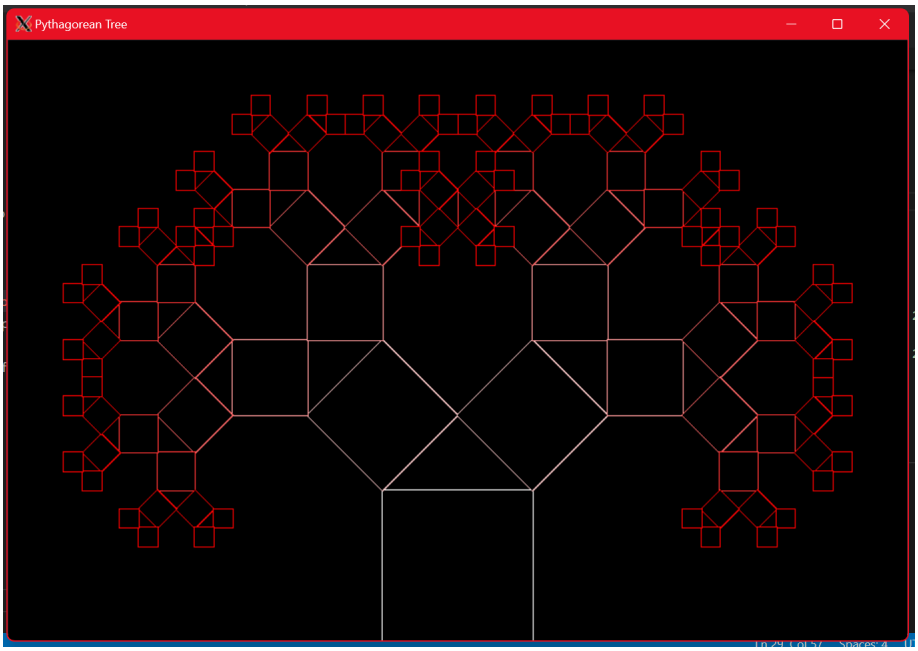
```
244     sf::Texture* goal;
245     goal = &_goal;
246     int num = std::count_if(spr, spr + (_width * _height),
247     [goal] (sf::Sprite obj) {
248         return obj.getTexture() == goal;
249     });
250     int correct = 0;
251     for (int i = 0; i < (_width * _height); i++) {
252         if (_spriteArr[i].getTexture() == &_goal && _mvArr[i].getTexture()
    == &_box) {
253             correct++;
254         }
255     }
256     if (correct == num) {
257         return 1;
258     }
259     return 0;
260 }
261
262 void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
263     for (int i = 0; i < (_width * _height); i++) {
264         target.draw(_spriteArr[i], states);
265         target.draw(_mvArr[i], states);
266     }
267 }
268
269 void Sokoban::shrink() {
270     while ((static_cast<unsigned>(_width * _px) >= sf::VideoMode::
    getDesktopMode().width ) ||
271         (static_cast<unsigned>(_height * _px) >= sf::VideoMode::
    getDesktopMode().height)) {
272         _px /= 2;
273         for (int height = 0; height < _height; height++) {
274             for (int width = 0; width < _width; width++) {
275                 _spriteArr[width + (height * _width)].scale(0.5, 0.5);
276                 _spriteArr[width + (height * _width)].setPosition((width*_px
    ), (height*_px));
277                 _mvArr[width + (height * _width)].scale(0.5, 0.5);
278                 _mvArr[width + (height * _width)].setPosition((width*_px), (
    height*_px));
279             }
280         }
281     }
282 }
283
284 Sokoban::~Sokoban() {
285     delete[] _spriteArr;
286     delete[] _mvArr;
287 }
```

4 PS3 Pythagorean Tree

4.1 Discussion:

This program was a bit of trouble for me, specifically because I didn't understand how to get a recursive function to create branching components without calling it multiple times in each iteration but that led to excessive branching in one direction instead of balanced branching. My next guess was to try to build the left side of the entire tree and then just reflect it to the other side, however that also did not work since I still had excessive branching stemming from the left branch instead of branches balancing. This ultimately led to me having each call of the function take the iteration depth it was in and use the iteration depth to calculate how many squares had to be drawn and drawing all of them from the squares they came from. To get the orientation and positioning, I had an extensive amount of if statements catering to each possible orientation which resulted in extremely long code. Although I made it work, the squares were fractionally offset from the corners where they needed to be, this was due to using $\frac{\sqrt{2}}{2}$ instead of $sin(45)$ or $cos(45)$ in an effort to avoid converting the angles of rotation between radians and degrees constantly.

4.2 Output:



4.3 Codebase:

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

.PHONY: lint all clean

all: lint PTree

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

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

clean:
  rm *.o PTree
lint:
  cpplint *.cpp *.hpp
```

```
// Copyright 2023 Ajay

#include <iostream>
#include <cmath>
#include <string>
#include <vector>
```

```

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

#include "PTree.hpp"

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cout << "Invalid Arguments Entered" << std::endl;
        return 1;
    }

    double l = std::stod(argv[1]);
    int i = std::stoi(argv[2]);

    PTree shape(l, i);

    sf::RenderWindow window(sf::VideoMode(l * 6, l * 4), "Pythagorean Tree")
    ;

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }
        window.clear();
        window.draw(shape);
        window.display();
    }
    return 0;
}
```

```
// Copyright 2023 Ajay

#include <vector>

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

class PTree : public sf::Drawable {
 public:
    PTree(double length, int iterations);
 private:
    void pTree(double length, int iterations);
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
    const;
    int depth;
    std::vector<sf::RectangleShape> _TREE;
};
```

```
// Copyright 2023 Ajay

#include <iostream>
```

```cpp
#include <cmath>
#include <algorithm>
#include <iterator>

#include "PTree.hpp"

void PTree::pTree(double length, int iterations) {
    sf::Vector2f shapesize(static_cast<float>(length), static_cast<float>(length));
    sf::Color redden(0, 10, 10, 0);
    if (_TREE.size() == 0) {
        sf::RectangleShape shape(shapesize);
        shape.setOutlineThickness(1);
        shape.setOutlineColor(sf::Color::White);
        shape.setFillColor(sf::Color::Black);
        shape.setOrigin(length / 2, length / 2);
        shape.setPosition(3 * static_cast<float>(length),
            3 * shapesize.y + (shapesize.y / 2));
        _TREE.push_back(shape);
    } else if (_TREE.size() == 1) {
        sf::RectangleShape shape1(shapesize);
        sf::RectangleShape shape2(shapesize);
        shape1.setOutlineThickness(1);
        shape1.setOutlineColor(sf::Color::White - redden);
        shape1.setFillColor(sf::Color::Black);
        shape2.setOutlineThickness(1);
        shape2.setOutlineColor(sf::Color::White - redden);
        shape2.setFillColor(sf::Color::Black);
        shape1.setOrigin(length / 2, length  / 2);
        shape1.setRotation(_TREE.front().getRotation() + 45);
        shape2.setOrigin(length / 2, length  / 2);
        shape2.setRotation(_TREE.front().getRotation() - 45);
        shape1.setPosition(_TREE.front().getPosition().x - (_TREE.front().getSize().x / 2),
            _TREE.front().getPosition().y - _TREE.front().getSize().y);
        shape2.setPosition(_TREE.front().getPosition().x + (_TREE.front().getSize().x / 2),
            _TREE.front().getPosition().y - _TREE.front().getSize().y);
        _TREE.push_back(shape1);
        _TREE.push_back(shape2);
    } else {
        auto s = _TREE.rbegin();
        std::vector<sf::RectangleShape> vec;
        for (int i = 0; i < pow(2, iterations - 1); i++) {
            sf::RectangleShape shape1(shapesize);
            sf::RectangleShape shape2(shapesize);
            shape1.setOutlineThickness(1);
            shape1.setOutlineColor(s->getOutlineColor() - redden);
            shape1.setFillColor(sf::Color::Black);
            shape2.setOutlineThickness(1);
            shape2.setOutlineColor(s->getOutlineColor() - redden);
            shape2.setFillColor(sf::Color::Black);
            shape1.setOrigin(length / 2, length  / 2);
            shape1.setRotation(s->getRotation() + 45);
            shape2.setOrigin(length / 2, length  / 2);
```

```
            shape2.setRotation(s->getRotation() - 45);
            if (s->getRotation() == 0) {
                shape1.setPosition(s->getPosition().x - (length / sqrt(2)),
                    s->getPosition().y - (length * sqrt(2)));
                shape2.setPosition(s->getPosition().x + (length / sqrt(2)),
                    s->getPosition().y - (length * sqrt(2)));
            } else if (s->getRotation() == 180) {
                shape2.setPosition(s->getPosition().x - (length / sqrt(2)),
                    s->getPosition().y + (length * sqrt(2)));
                shape1.setPosition(s->getPosition().x + (length / sqrt(2)),
                    s->getPosition().y + (length * sqrt(2)));
            } else if (s->getRotation() == 45) {
                shape1.setPosition(s->getPosition().x - (length * 1.5),
                    s->getPosition().y - (length / 2));
                shape2.setPosition(s->getPosition().x - (length / 2),
                    s->getPosition().y - (length * 1.5));
            } else if (s->getRotation() == 315) {
                shape2.setPosition(s->getPosition().x + (length * 1.5),
                    s->getPosition().y - (length / 2));
                shape1.setPosition(s->getPosition().x + (length / 2),
                    s->getPosition().y - (length * 1.5));
            } else if (s->getRotation() == 90) {
                shape1.setPosition(s->getPosition().x - (length * sqrt(2)),
                    s->getPosition().y + (length / sqrt(2)));
                shape2.setPosition(s->getPosition().x - (length * sqrt(2)),
                    s->getPosition().y - (length / sqrt(2)));
            } else if (s->getRotation() == 270) {
                shape2.setPosition(s->getPosition().x + (length * sqrt(2)),
                    s->getPosition().y + (length / sqrt(2)));
                shape1.setPosition(s->getPosition().x + (length * sqrt(2)),
                    s->getPosition().y - (length / sqrt(2)));
            } else if (s->getRotation() == 135) {
                shape1.setPosition(s->getPosition().x - (length / 2),
                    s->getPosition().y + (length * 1.5));
                shape2.setPosition(s->getPosition().x - (length * 1.5),
                    s->getPosition().y + (length / 2));
            } else if (s->getRotation() == 225) {
                shape2.setPosition(s->getPosition().x + (length / 2),
                    s->getPosition().y + (length * 1.5));
                shape1.setPosition(s->getPosition().x + (length * 1.5),
                    s->getPosition().y + (length / 2));
            }
            vec.push_back(shape1);
            vec.push_back(shape2);
            s++;
        }
        for (auto j : vec) {
            _TREE.push_back(j);
        }
    }
    if (iterations != depth) {
        pTree(static_cast<double>(length * (sqrt(2)/2)), (iterations + 1));
    }
    return;
}
```

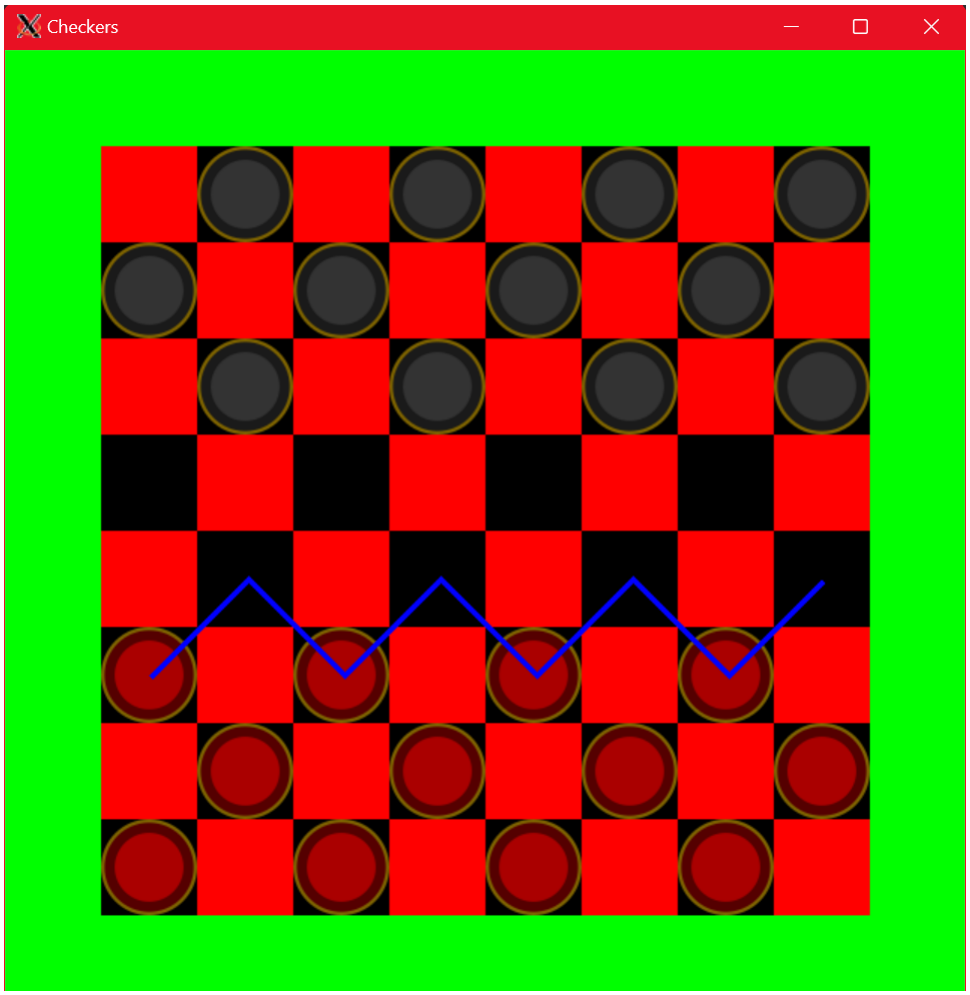
```
111
112 PTree::PTree(double length, int iterations) {
113     depth = iterations;
114     pTree(length, 0);
115 }
116
117 void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
118         for (auto item : _TREE) {
119             target.draw(item, states);
120         }
121         return;
122 }
```

5 PS4 Checkers

5.1 Discussion:

This program focused on creating fully implemented Checkers game. Using what I had learned from Sokoban, I implemented this grid based game using a vector. Since I had just done Sokoban, I was able to get the board and pieces working extremely quickly, however the trouble I had at first was selecting a piece. Since the piece could be selected anywhere in the bounds of the square, the coordinate of the mouse click had to first be checked to see if it was within the boundaries of the tile and then check if the tile was a valid one to select. I also had to make sure there was only one selected tile at a time. The next part was figuring out a way to show valid moves for each piece. This was difficult for multiple reasons. The first of which was because I needed to show the moves for only the current side, this required me to make something that would keep track of which side was current active, which I had the main function take care off, calling functions with the active side as a parameter instead of letting each function decide. The next problem was creating a visual marker for each possible move and also validating the start and end coordinates of jumps. This was solved by making a vector that stored each sprite, along with the start and end coordinates for each move. After this point my project was fully working. Working with vector objects was easier than creating an array, however the matrix still seemed like a better object to use for next time. One important thing that I learned was that it may feel easier to try and adapt a program that isn't working, but sometimes it is better to completely dismantle the project and restart to make a better project.

5.2 Output:



5.3 Codebase:

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

.PHONY: lint all clean

all: lint Checkers

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

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

```

clean:
  rm *.o Checkers
lint:
  cpplint *.cpp *.hpp
```

```
// Copyright 2023 Ajay

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

#include "Checkers.hpp"

int main(int argc, char* argv[]) {
    Checkers game;
    sf::RenderWindow window(sf::VideoMode(640, 640), "Checkers");
    sf::Color curTurn = sf::Color::Red;
    int gameWon = 0;
    while (window.isOpen()) {
        pColor side = (curTurn == sf::Color::Red) ? RED : BLACK;
        sf::Event event;
        while (window.pollEvent(event)) {
            game.showMoves(side);
            if (game.checkWin(curTurn) != NONE) {
                gameWon = 1;
            }
            if (event.type == sf::Event::Closed)
                window.close();
            if (event.type == sf::Event::MouseButtonPressed) {
                if (event.mouseButton.button == sf::Mouse::Button::Left) {
                    if ((curTurn == sf::Color::Red && game.checkPiece(event.
    mouseButton.x,
                        event.mouseButton.y) == RED) || (curTurn == sf::
    Color::Black &&
                            game.checkPiece(event.mouseButton.x, event.
    mouseButton.y) == BLACK)) {
                                game.selectPiece(event.mouseButton.x, event.
    mouseButton.y, side);
                    } else if (game.existSelected()) {
                        if (game.mvPiece(event.mouseButton.x, event.
    mouseButton.y, side)) {
                            if (curTurn == sf::Color::Red) {
                                curTurn = sf::Color::Black;
                            } else {
                                curTurn = sf::Color::Red;
                            }
                        }
                    }
                }
            }
        }
        window.clear();
        window.draw(game);
        window.display();
        if (gameWon == 1) {
```

```
            sf::Image winPic;
            if (game.checkWin(curTurn) == RED) {
                if (!winPic.loadFromFile("sources/rWin.png"))
                    throw std::runtime_error("Image could not be opened!");
            } else if (game.checkWin(curTurn) == BLACK) {
                if (!winPic.loadFromFile("sources/bWin.png"))
                    throw std::runtime_error("Image could not be opened!");
            }
            sf::Texture texture;
            texture.loadFromImage(winPic);
            sf::Sprite sprite;
            sprite.setTexture(texture);
            window.clear();
            window.draw(sprite);
            window.display();
            while (window.waitEvent(event)) {
                if (event.type == sf::Event::Closed) {
                    window.close();
                }
            }
        }
    }
    return 0;
}
```

```
// Copyright 2023 Ajay

#pragma once

#include <iostream>
#include <vector>
#include <utility>

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

typedef enum pcolor {NONE, BORDER, BLACK, RED} pColor;

class Checkers : public sf::Drawable {
 public:
    Checkers();
    void showMoves(pColor side);
    sf::Color selectPiece(int x, int y, pColor side);
    bool mvPiece(int x, int y, pColor side);
    pColor checkPiece(float fx, float fy);
    bool existSelected();
    pColor checkWin(sf::Color curTurn);
 private:
    sf::Texture _redPawn;
    sf::Texture _redKing;
    sf::Texture _blackPawn;
    sf::Texture _blackKing;
    std::vector<sf::Sprite> _pieceArr;
    std::vector<sf::RectangleShape> _board;
    std::vector<std::pair<sf::RectangleShape, std::pair<sf::Vector2f, sf::
```

```
    Vector2f>>> _blackMoves;
32  std::vector<std::pair<sf::RectangleShape, std::pair<sf::Vector2f, sf::
    Vector2f>>> _redMoves;
33  virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
    const;
34 };
```

```
1  // Copyright 2023 Ajay
2
3  #include <cmath>
4  #include <algorithm>
5
6  #include "Checkers.hpp"
7
8  Checkers::Checkers() {
9      // Board Creation
10     sf::Vector2f tileSize(64.f, 64.f);
11     sf::RectangleShape _red(tileSize);
12     sf::RectangleShape _black(tileSize);
13     sf::RectangleShape _border(tileSize);
14     _red.setFillColor(sf::Color::Red);
15     _black.setFillColor(sf::Color::Black);
16     _border.setFillColor(sf::Color::Green);
17     for (int k = 0; k < 10; k++) {
18         if (k == 0 || k == 9) {
19             for (int i = 0; i < 10; i++) {
20                     _border.setPosition(i * 64, k * 64);
21                     _board.push_back(_border);
22             }
23         } else if (k % 2 == 1) {
24             for (int j = 0; j < 10; j++) {
25                 if (j == 0 || j == 9) {
26                     _border.setPosition(j * 64, k * 64);
27                     _board.push_back(_border);
28                 } else if (j % 2 == 1) {
29                     _red.setPosition(j * 64, k * 64);
30                     _board.push_back(_red);
31                 } else {
32                     _black.setPosition(j * 64, k * 64);
33                     _board.push_back(_black);
34                 }
35             }
36         } else if (k % 2 == 0) {
37             for (int l = 0; l < 10; l++) {
38                 if (l == 0 || l == 9) {
39                     _border.setPosition(l * 64, k * 64);
40                     _board.push_back(_border);
41                 } else if (l % 2 == 1) {
42                     _black.setPosition(l * 64, k * 64);
43                     _board.push_back(_black);
44                 } else {
45                     _red.setPosition(l * 64, k * 64);
46                     _board.push_back(_red);
47                 }
48             }
49         }
```

```
50      }
51      // Piece Creation
52      sf::Image rK, rP, bK, bP;
53      if (!rK.loadFromFile("sources/redking.png"))
54          throw std::runtime_error("Image could not be opened!");
55      if (!rP.loadFromFile("sources/redpawn.png"))
56          throw std::runtime_error("Image could not be opened!");
57      if (!bK.loadFromFile("sources/blackking.png"))
58          throw std::runtime_error("Image could not be opened!");
59      if (!bP.loadFromFile("sources/blackpawn.png"))
60          throw std::runtime_error("Image could not be opened!");
61      sf::Texture rKing, rPawn, bKing, bPawn;
62      rKing.loadFromImage(rK);
63      rPawn.loadFromImage(rP);
64      bKing.loadFromImage(bK);
65      bPawn.loadFromImage(bP);
66      _redPawn = rPawn;
67      _redKing = rKing;
68      _blackPawn = bPawn;
69      _blackKing = bKing;
70      // Piece Set
71      for (int i = 10; i < 39; i++) {
72          if (_board[i].getFillColor() == sf::Color::Black) {
73              sf::Sprite sprite;
74              sprite.setTexture(_blackPawn);
75              sprite.setPosition(_board[i].getPosition());
76              _pieceArr.push_back(sprite);
77          }
78      }
79      for (int i = 60; i < 89; i++) {
80          if (_board[i].getFillColor() == sf::Color::Black) {
81              sf::Sprite sprite;
82              sprite.setTexture(_redPawn);
83              sprite.setPosition(_board[i].getPosition());
84              _pieceArr.push_back(sprite);
85          }
86      }
87  }
88
89  void Checkers::showMoves(pColor side) {
90      sf::RectangleShape line1(sf::Vector2f(64 * std::sqrt(2), 3));
91      sf::RectangleShape line2(sf::Vector2f(64 * std::sqrt(2), 3));
92      sf::RectangleShape line3(sf::Vector2f(64 * std::sqrt(2), 3));
93      sf::RectangleShape line4(sf::Vector2f(64 * std::sqrt(2), 3));
94      line1.setRotation(-45);
95      line2.setRotation(-135);
96      line3.setRotation(45);
97      line4.setRotation(135);
98      line1.setFillColor(sf::Color::Blue);
99      line2.setFillColor(sf::Color::Blue);
100     line3.setFillColor(sf::Color::Blue);
101     line4.setFillColor(sf::Color::Blue);
102     if (side == BLACK) {
103         for (auto piece : _pieceArr) {
104             if (piece.getTexture() == &_blackPawn || piece.getTexture() == &
```

```
    _blackKing) {
105                 _redMoves.clear();
106                 if (checkPiece(piece.getPosition().x + 64, piece.getPosition
    ().y + 64) == NONE) {
107                     line3.setPosition(piece.getPosition().x + 32.5, piece.
    getPosition().y + 32.5);
108                     if (line3.getPosition() != sf::Vector2f(0, 0)) {
109                         std::pair<sf::Vector2f, sf::Vector2f> coords(piece.
    getPosition(),
110                             piece.getPosition() + sf::Vector2f(64, 64));
111                         std::pair<sf::RectangleShape, std::pair<sf::Vector2f
    ,
112                             sf::Vector2f>> downRight(line3, coords);
113                         _blackMoves.push_back(downRight);
114                     }
115                 }
116                 if (checkPiece(piece.getPosition().x + 64, piece.getPosition
    ().y + 64) == RED &&
117                     checkPiece(piece.getPosition().x + 128, piece.
    getPosition().y + 128) == NONE) {
118                     line3.setSize(sf::Vector2f(line3.getSize().x * 2, line3.
    getSize().y));
119                     line3.setPosition(piece.getPosition().x + 32.5, piece.
    getPosition().y + 32.5);
120                     if (line3.getPosition() != sf::Vector2f(0, 0)) {
121                         std::pair<sf::Vector2f, sf::Vector2f> coords(piece.
    getPosition(),
122                             piece.getPosition() + sf::Vector2f(128, 128));
123                         std::pair<sf::RectangleShape, std::pair<sf::Vector2f
    ,
124                             sf::Vector2f>> downRight(line3, coords);
125                         _blackMoves.push_back(downRight);
126                     }
127                     line3.setSize(sf::Vector2f(line3.getSize().x / 2, line3.
    getSize().y));
128                 }
129                 if (checkPiece(piece.getPosition().x - 64, piece.getPosition
    ().y + 64) == NONE) {
130                     line4.setPosition(piece.getPosition().x + 32.5, piece.
    getPosition().y + 32.5);
131                     if (line4.getPosition() != sf::Vector2f(0, 0)) {
132                         std::pair<sf::Vector2f, sf::Vector2f> coords(piece.
    getPosition(),
133                             piece.getPosition() + sf::Vector2f(-64, 64));
134                         std::pair<sf::RectangleShape, std::pair<sf::Vector2f
    ,
135                             sf::Vector2f>> downLeft(line4, coords);
136                         _blackMoves.push_back(downLeft);
137                     }
138                 }
139                 if (checkPiece(piece.getPosition().x - 64, piece.getPosition
    ().y + 64) == RED &&
140                     checkPiece(piece.getPosition().x - 128, piece.
    getPosition().y + 128) == NONE) {
141                     line4.setSize(sf::Vector2f(line4.getSize().x * 2, line4.
```

```
    getSize().y));
142                     line4.setPosition(piece.getPosition().x + 32.5, piece.
    getPosition().y + 32.5);
143                     if (line4.getPosition() != sf::Vector2f(0, 0)) {
144                         std::pair<sf::Vector2f, sf::Vector2f> coords(piece.
    getPosition(),
145                             piece.getPosition() + sf::Vector2f(-128, 128));
146                         std::pair<sf::RectangleShape, std::pair<sf::Vector2f
    ,
147                             sf::Vector2f>> downRight(line4, coords);
148                         _blackMoves.push_back(downRight);
149                     }
150                     line4.setSize(sf::Vector2f(line4.getSize().x / 2, line4.
    getSize().y));
151                 }
152                 if (piece.getTexture() == &_blackKing) {
153                     if (checkPiece(piece.getPosition().x + 64,
154                         piece.getPosition().y - 64) == NONE) {
155                         line1.setPosition(piece.getPosition().x + 32.5,
156                             piece.getPosition().y + 32.5);
157                         if (line1.getPosition() != sf::Vector2f(0, 0)) {
158                             std::pair<sf::Vector2f, sf::Vector2f> coords(
    piece.getPosition(),
159                                 piece.getPosition() + sf::Vector2f(64, -64))
    ;
160                             std::pair<sf::RectangleShape, std::pair<sf::
    Vector2f,
161                                 sf::Vector2f>> upRight(line1, coords);
162                             _blackMoves.push_back(upRight);
163                         }
164                     }
165                     if (checkPiece(piece.getPosition().x + 64,
166                         piece.getPosition().y - 64) == RED &&
167                         checkPiece(piece.getPosition().x + 128,
168                             piece.getPosition().y - 128) == NONE) {
169                         line1.setSize(sf::Vector2f(line1.getSize().x * 2,
    line1.getSize().y));
170                         line1.setPosition(piece.getPosition().x + 32.5,
171                             piece.getPosition().y + 32.5);
172                         if (line1.getPosition() != sf::Vector2f(0, 0)) {
173                             std::pair<sf::Vector2f, sf::Vector2f> coords(
    piece.getPosition(),
174                                 piece.getPosition() + sf::Vector2f(128,
    -128));
175                             std::pair<sf::RectangleShape, std::pair<sf::
    Vector2f,
176                                 sf::Vector2f>> downRight(line1, coords);
177                             _blackMoves.push_back(downRight);
178                         }
179                         line1.setSize(sf::Vector2f(line1.getSize().x / 2,
    line1.getSize().y));
180                     }
181                     if (checkPiece(piece.getPosition().x - 64,
182                         piece.getPosition().y - 64) == NONE) {
183                         line2.setPosition(piece.getPosition().x + 32.5,
```

```
                            piece.getPosition().y + 32.5);
                    if (line2.getPosition() != sf::Vector2f(0, 0)) {
                        std::pair<sf::Vector2f, sf::Vector2f>
                            coords(piece.getPosition(),
                                piece.getPosition() + sf::Vector2f(-64, -64));
                        std::pair<sf::RectangleShape, std::pair<sf::Vector2f,
                            sf::Vector2f>> upLeft(line2, coords);
                        _blackMoves.push_back(upLeft);
                    }
                }
                if (checkPiece(piece.getPosition().x - 64,
                    piece.getPosition().y - 64) == RED &&
                    checkPiece(piece.getPosition().x - 128,
                        piece.getPosition().y - 128) == NONE) {
                    line2.setSize(sf::Vector2f(line2.getSize().x * 2, line2.getSize().y));
                    line2.setPosition(piece.getPosition().x + 32.5,
                        piece.getPosition().y + 32.5);
                    if (line2.getPosition() != sf::Vector2f(0, 0)) {
                        std::pair<sf::Vector2f, sf::Vector2f> coords(piece.getPosition(),
                            piece.getPosition() + sf::Vector2f(-128, -128));
                        std::pair<sf::RectangleShape, std::pair<sf::Vector2f,
                            sf::Vector2f>> downRight(line2, coords);
                        _blackMoves.push_back(downRight);
                    }
                    line2.setSize(sf::Vector2f(line2.getSize().x / 2, line2.getSize().y));
                }
            }
        }
    }
} else if (side == RED) {
    for (auto piece : _pieceArr) {
        if (piece.getTexture() == &_redPawn || piece.getTexture() == &_redKing) {
            _blackMoves.clear();
            if (checkPiece(piece.getPosition().x + 64, piece.getPosition().y - 64) == NONE) {
                line1.setPosition(piece.getPosition().x + 32.5, piece.getPosition().y + 32.5);
                if (line1.getPosition() != sf::Vector2f(0, 0)) {
                    std::pair<sf::Vector2f, sf::Vector2f> coords(piece.getPosition(),
                        piece.getPosition() + sf::Vector2f(64, -64));
                    std::pair<sf::RectangleShape, std::pair<sf::Vector2f, sf::Vector2f>> upRight
                        (line1, coords);
                    _redMoves.push_back(upRight);
                }
            }
```

```
227                 if (checkPiece(piece.getPosition().x + 64, piece.getPosition
    ().y - 64) == BLACK &&
228                     checkPiece(piece.getPosition().x + 128, piece.
    getPosition().y - 128) == NONE) {
229                     line1.setSize(sf::Vector2f(line1.getSize().x * 2, line1.
    getSize().y));
230                     line1.setPosition(piece.getPosition().x + 32.5, piece.
    getPosition().y + 32.5);
231                     if (line1.getPosition() != sf::Vector2f(0, 0)) {
232                         std::pair<sf::Vector2f, sf::Vector2f> coords(piece.
    getPosition(),
233                             piece.getPosition() + sf::Vector2f(128, -128));
234                         std::pair<sf::RectangleShape, std::pair<sf::Vector2f
    ,
235                             sf::Vector2f>> downRight(line1, coords);
236                         _redMoves.push_back(downRight);
237                     }
238                     line1.setSize(sf::Vector2f(line1.getSize().x / 2, line1.
    getSize().y));
239                 }
240                 if (checkPiece(piece.getPosition().x - 64, piece.getPosition
    ().y - 64) == NONE) {
241                     line2.setPosition(piece.getPosition().x + 32.5, piece.
    getPosition().y + 32.5);
242                     if (line2.getPosition() != sf::Vector2f(0, 0)) {
243                         std::pair<sf::Vector2f, sf::Vector2f> coords(piece.
    getPosition(),
244                             piece.getPosition() + sf::Vector2f(-64, -64));
245                         std::pair<sf::RectangleShape, std::pair<sf::Vector2f
    , sf::Vector2f>> upLeft
246                             (line2, coords);
247                         _redMoves.push_back(upLeft);
248                     }
249                 }
250                 if (checkPiece(piece.getPosition().x - 64, piece.getPosition
    ().y - 64) == BLACK &&
251                     checkPiece(piece.getPosition().x - 128, piece.
    getPosition().y - 128) == NONE) {
252                     line2.setSize(sf::Vector2f(line2.getSize().x * 2, line2.
    getSize().y));
253                     line2.setPosition(piece.getPosition().x + 32.5, piece.
    getPosition().y + 32.5);
254                     if (line2.getPosition() != sf::Vector2f(0, 0)) {
255                         std::pair<sf::Vector2f, sf::Vector2f> coords(piece.
    getPosition(),
256                             piece.getPosition() + sf::Vector2f(-128, -128));
257                         std::pair<sf::RectangleShape, std::pair<sf::Vector2f
    ,
258                             sf::Vector2f>> downRight(line2, coords);
259                         _redMoves.push_back(downRight);
260                     }
261                     line2.setSize(sf::Vector2f(line2.getSize().x / 2, line2.
    getSize().y));
262                 }
263                 if (piece.getTexture() == &_redKing) {
```

```
264                     if (checkPiece(piece.getPosition().x + 64,
265                         piece.getPosition().y + 64) == NONE) {
266                         line3.setPosition(piece.getPosition().x + 32.5,
267                             piece.getPosition().y + 32.5);
268                         if (line3.getPosition() != sf::Vector2f(0, 0)) {
269                             std::pair<sf::Vector2f, sf::Vector2f> coords(
    piece.getPosition(),
270                                 piece.getPosition() + sf::Vector2f(64, 64));
271                             std::pair<sf::RectangleShape, std::pair<sf::
    Vector2f,
272                                 sf::Vector2f>> downRight(line3, coords);
273                             _redMoves.push_back(downRight);
274                         }
275                     }
276                     if (checkPiece(piece.getPosition().x + 64,
277                         piece.getPosition().y + 64) == BLACK &&
278                         checkPiece(piece.getPosition().x + 128,
279                             piece.getPosition().y + 128) == NONE) {
280                         line3.setSize(sf::Vector2f(line3.getSize().x * 2,
    line3.getSize().y));
281                         line3.setPosition(piece.getPosition().x + 32.5,
282                             piece.getPosition().y + 32.5);
283                         if (line3.getPosition() != sf::Vector2f(0, 0)) {
284                             std::pair<sf::Vector2f, sf::Vector2f> coords(
    piece.getPosition(),
285                                 piece.getPosition() + sf::Vector2f(128, 128)
    );
286                             std::pair<sf::RectangleShape, std::pair<sf::
    Vector2f,
287                                 sf::Vector2f>> downRight(line3, coords);
288                             _redMoves.push_back(downRight);
289                         }
290                         line3.setSize(sf::Vector2f(line3.getSize().x / 2,
    line3.getSize().y));
291                     }
292                     if (checkPiece(piece.getPosition().x - 64,
293                         piece.getPosition().y + 64) == NONE) {
294                         line4.setPosition(piece.getPosition().x + 32.5,
295                             piece.getPosition().y + 32.5);
296                         if (line4.getPosition() != sf::Vector2f(0, 0)) {
297                             std::pair<sf::Vector2f, sf::Vector2f> coords(
    piece.getPosition(),
298                                 piece.getPosition() + sf::Vector2f(-64, 64))
    ;
299                             std::pair<sf::RectangleShape, std::pair<sf::
    Vector2f,
300                                 sf::Vector2f>> downLeft(line4, coords);
301                             _redMoves.push_back(downLeft);
302                         }
303                     }
304                     if (checkPiece(piece.getPosition().x - 64,
305                         piece.getPosition().y + 64) == BLACK &&
306                         checkPiece(piece.getPosition().x - 128,
307                             piece.getPosition().y + 128) == NONE) {
308                         line4.setSize(sf::Vector2f(line4.getSize().x * 2,
```

```
    line4.getSize().y));
309                         line4.setPosition(piece.getPosition().x + 32.5,
310                             piece.getPosition().y + 32.5);
311                         if (line4.getPosition() != sf::Vector2f(0, 0)) {
312                             std::pair<sf::Vector2f, sf::Vector2f> coords(
    piece.getPosition(),
313                                 piece.getPosition() + sf::Vector2f(-128,
    128));
314                             std::pair<sf::RectangleShape, std::pair<sf::
    Vector2f,
315                                 sf::Vector2f>> downRight(line4, coords);
316                             _redMoves.push_back(downRight);
317                         }
318                         line4.setSize(sf::Vector2f(line4.getSize().x / 2,
    line4.getSize().y));
319                     }
320                 }
321             }
322         }
323     }
324 }
325
326 pColor Checkers::checkPiece(float fx, float fy) {
327     if (fx < 64 || fx >= 576 || fy < 64 || fy >= 576) {
328         return BORDER;
329     }
330     for (auto piece : _pieceArr) {
331         sf::Vector2f pos = piece.getPosition();
332         if (fx >= pos.x && fx < (pos.x + 64) && fy >= pos.y && fy < (pos.y +
     64)) {
333             if (piece.getTexture() == &_blackPawn || piece.getTexture() == &
    _blackKing) {
334                 return BLACK;
335             }
336             if (piece.getTexture() == &_redPawn || piece.getTexture() == &
    _redKing) {
337                 return RED;
338             }
339         }
340     }
341     return NONE;
342 }
343
344 sf::Color Checkers::selectPiece(int x, int y, pColor side) {
345     float fx = static_cast<float>(x);
346     float fy = static_cast<float>(y);
347     for (int j = 10; j < 89; j++) {
348         if (_board[j].getFillColor() == sf::Color::Yellow) {
349             _board[j].setFillColor(sf::Color::Black);
350         }
351     }
352     for (auto piece : _pieceArr) {
353         sf::Vector2f pos = piece.getPosition();
354         if (fx >= pos.x && fx < (pos.x + 64) && fy >= pos.y && fy < (pos.y +
     64)) {
```

```
355             if (checkPiece(fx, fy) == BLACK && side == BLACK) {
356                 for (int i = 10; i < 89; i++) {
357                     if (_board[i].getPosition() == pos) {
358                         _board[i].setFillColor(sf::Color::Yellow);
359                     }
360                 }
361                 return sf::Color::Black;
362             } else if (checkPiece(fx, fy) == RED && side == RED) {
363                 for (int i = 10; i < 89; i++) {
364                     if (_board[i].getPosition() == pos) {
365                         _board[i].setFillColor(sf::Color::Yellow);
366                     }
367                 }
368                 return sf::Color::Red;
369             }
370         }
371     }
372     return sf::Color::Green;
373 }
374
375 bool Checkers::existSelected() {
376     for (auto tile : _board) {
377         if (tile.getFillColor() == sf::Color::Yellow) {
378             return true;
379         }
380     }
381     return false;
382 }
383
384 bool Checkers::mvPiece(int x, int y, pColor side) {
385     float fx = static_cast<float>(x);
386     float fy = static_cast<float>(y);
387     sf::Vector2f pos;
388     for (int i = 0; i < 89; i++) {
389         if (_board[i].getFillColor() == sf::Color::Yellow) {
390             pos = _board[i].getPosition();
391             _board[i].setFillColor(sf::Color::Black);
392             break;
393         }
394     }
395     if (side == RED) {
396         for (auto mvPair : _redMoves) {
397             if (mvPair.second.first == pos) {
398                 if (fx >= mvPair.second.second.x && fx < (mvPair.second.
    second.x + 64) &&
399                     fy >= mvPair.second.second.y && fy < (mvPair.second.
    second.y + 64)) {
400                     auto rPiece = std::find_if(_pieceArr.begin(),
    _pieceArr.end(),
401                         [mvPair](sf::Sprite pc) {
402                             if (mvPair.second.first + sf::Vector2f(128,
    128)
403                                 == mvPair.second.second) {
404                                 return (pc.getPosition() ==
405                                     mvPair.second.first + sf::Vector2f
```

```
    (64, 64));
406                                 } else if (mvPair.second.first + sf::
    Vector2f(-128, 128)
407                                     == mvPair.second.second) {
408                                     return (pc.getPosition() ==
409                                         mvPair.second.first + sf::Vector2f
    (-64, 64));
410                                 } else if (mvPair.second.first + sf::
    Vector2f(128, -128)
411                                     == mvPair.second.second) {
412                                     return (pc.getPosition() ==
413                                         mvPair.second.first + sf::Vector2f
    (64, -64));
414                                 } else if (mvPair.second.first + sf::
    Vector2f(-128, -128)
415                                     == mvPair.second.second) {
416                                     return (pc.getPosition() ==
417                                         mvPair.second.first + sf::Vector2f
    (-64, -64));
418                                 }
419                                 return false;
420                         });
421                         if (rPiece != _pieceArr.end())
422                             _pieceArr.erase(rPiece);
423                         auto pPiece = std::find_if(_pieceArr.begin(),
    _pieceArr.end(),
424                             [pos](sf::Sprite pc) {
425                                 return (pc.getPosition() == pos);
426                         });
427                         pPiece->setPosition(mvPair.second.second);
428                         if (pPiece->getPosition().y == 64) {
429                             pPiece->setTexture(_redKing);
430                         }
431                         return true;
432                 }
433             }
434         }
435     } else if (side == BLACK) {
436         for (auto mvPair : _blackMoves) {
437             if (mvPair.second.first == pos) {
438                 if (fx >= mvPair.second.second.x && fx < (mvPair.second.
    second.x + 64) &&
439                     fy >= mvPair.second.second.y && fy < (mvPair.second.
    second.y + 64)) {
440                         auto rPiece = std::find_if(_pieceArr.begin(),
    _pieceArr.end(),
441                             [mvPair](sf::Sprite pc) {
442                                 if (mvPair.second.first + sf::Vector2f(128,
    128)
443                                     == mvPair.second.second) {
444                                     return (pc.getPosition() ==
445                                         mvPair.second.first + sf::Vector2f
    (64, 64));
446                                 } else if (mvPair.second.first + sf::
    Vector2f(-128, 128)
```

```
447                                     == mvPair.second.second) {
448                                     return (pc.getPosition() ==
449                                         mvPair.second.first + sf::Vector2f
    (-64, 64));
450                                 } else if (mvPair.second.first + sf::
    Vector2f(128, -128)
451                                     == mvPair.second.second) {
452                                     return (pc.getPosition() ==
453                                         mvPair.second.first + sf::Vector2f
    (64, -64));
454                                 } else if (mvPair.second.first + sf::
    Vector2f(-128, -128)
455                                     == mvPair.second.second) {
456                                     return (pc.getPosition() ==
457                                         mvPair.second.first + sf::Vector2f
    (-64, -64));
458                                 }
459                                 return false;
460                         });
461                         if (rPiece != _pieceArr.end())
462                             _pieceArr.erase(rPiece);
463                         auto pPiece = std::find_if(_pieceArr.begin(),
    _pieceArr.end(),
464                         [pos](sf::Sprite pc) {
465                             return (pc.getPosition() == pos);
466                         });
467                         pPiece->setPosition(mvPair.second.second);
468                         if (pPiece->getPosition().y == 512) {
469                             pPiece->setTexture(_blackKing);
470                         }
471                         return true;
472                 }
473             }
474         }
475     }
476     return false;
477 }
478
479 pColor Checkers::checkWin(sf::Color curTurn) {
480     if (curTurn == sf::Color::Black) {
481         if (_blackMoves.size() == 0) {
482             return RED;
483         }
484     } else if (curTurn == sf::Color::Red) {
485         if (_redMoves.size() == 0) {
486             return BLACK;
487         }
488     }
489     return NONE;
490 }
491
492 void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) const
     {
493         for (auto tile : _board) {
494             target.draw(tile, states);
```

```
        }
        for (auto piece : _pieceArr) {
            target.draw(piece, states);
        }
        for (auto bMove : _blackMoves) {
            target.draw(bMove.first, states);
        }
        for (auto rMove : _redMoves) {
            target.draw(rMove.first, states);
        }
}
```

6 PS5 DNA Alignment

6.1 Discussion:

This project focused on creating the smallest edit distance for two strings of DNA so that the the strings would match up as close as possible. This project was extremely confusing for me at first, however I learned two very important things from it. The first was that using matrices was a lot easier to deal with compared to using arrays or vectors with matrix math. By using a matrix, it was a lot easier to index each item and figure out item relations. The other thing I learned was that when working in a group with someone, it is important to work with someone who has the same frame of mind on a project. Me and my partner worked together but while my partner only wanted to work at certain times, I was focused on completing it as soon as I could. This resulted in a little more trouble than it would have if we both had the same attitude to the project. However we were able to compromise and work together which led me to realize that 2 heads are better than 1 in most cases. Part of the problem with working with others is the clash of understanding styles which results in one person understanding the current problem, while the other person is incredibly lost from their understanding. Overall, this experience was interesting, however I would prefer to code my small scale projects on my own.

6.2 Output:

```
ajay@DESKTOP-PCRG7HC:/mnt/c/Users/Ajay/Downloads/COMP4/PS5$ ./EDistance < sequence/example10.txt
Edit Distance = 7
A T 1
A A 0
C - 2
A A 0
G G 0
T G 1
T T 0
A - 2
C C 0
C A 1

Execution time is 0.000874 seconds
ajay@DESKTOP-PCRG7HC:/mnt/c/Users/Ajay/Downloads/COMP4/PS5$
```

6.3 Codebase:

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

.PHONY: lint all clean

all: lint EDistance

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

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

clean:
  rm *.o EDistance
lint:
  cpplint *.cpp *.hpp
```

```
// Copyright 2023 Alex Ford & Ajay Alamuri
#include <fstream>

#include <SFML/System.hpp>

#include "EDistance.hpp"

int main(int argc, char *argv[]) {
    sf::Clock clock;
    sf::Time t;
    std::string i1;
    std::string i2;

    std::cin >> i1;
    std::cin >> i2;

    i1 += '-';
    i2 += '-';

    EDistance table(i1, i2);
    std::cout << "Edit Distance = " << table.optDistance() <<
        '\n' << table.alignment() << std::endl;
    t = clock.getElapsedTime();
    std::cout << "Execution time is " << t.asSeconds() << " seconds \n";
    return 0;
}
```

```
// Copyright 2023 Alex Ford & Ajay Alamuri

#pragma once

#include <algorithm>
```

```hpp
#include <iostream>
#include <vector>
#include <string>

#include "Matrix.hpp"

class EDistance : public Matrix {
 public:
    EDistance(std::string, std::string);
    static int penalty(char, char);
    static int min(int, int, int);
    int optDistance();
    std::string alignment();
    void printData();
    void printFirst();
    void printInd(int x, int y);
    void printLast();
 private:
    std::string _input1;
    std::string _input2;
    Matrix _data = Matrix(0, 0);
};
```

```cpp
// Copyright 2023 Alex Ford & Ajay Alamuri

#include <stdexcept>

#include "EDistance.hpp"
#include "Matrix.hpp"

EDistance::EDistance(std::string s1, std::string s2) {
    _data = Matrix(s1.size(), s2.size());
    _input1 = s1;
    _input2 = s2;
}

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

int EDistance::min(int a, int b, int c) {
    int z = (a < b) ? a : b;
    return (z < c) ? z : c;
}

int EDistance::optDistance() {
    for (size_t i = 1; i < _data.height(); i++) {
        _data[i - 1][_data.width() - 1] = 2 * (_data.height() - i);
    }
    for (size_t j = 1; j < _data.width(); j++) {
        _data[_data.height() - 1][j - 1] = 2 * (_data.width() - j);
    }
    for (size_t j = _data.width() - 1; j > 0; j--) {
        for (size_t i = _data.height() - 1; i > 0; i--) {
            _data[i - 1][j - 1] = min(
                _data[i][j] + penalty(_input1[i - 1], _input2[j - 1]),
```

```
                _data[i][j - 1] + 2, _data[i - 1][j] + 2);
        }
    }
    return _data[0][0];
}

std::string EDistance::alignment() {
    std::string output;
    size_t i = 0, j = 0;
    while (i < _data.height() - 1 && j < _data.width() - 1) {
        if (_data[i][j] == (_data[i+1][j+1] + penalty(_input1[i], _input2[j
    ]))) {
            output += _input1[i];
            output += " ";
            output += _input2[j];
            output += " ";
            (penalty(_input1[i], _input2[j]) == 1) ? (output += '1') : (
    output += '0');
            output += "\n";
            i++, j++;
        } else if (_data[i][j] == (_data[i+1][j] + 2)) {
            output += _input1[i];
            output += " - 2\n";
            i++;
        } else if (_data[i][j] == (_data[i][j+1] + 2)) {
            output += "- ";
            output += _input2[j];
            output += " 2\n";
            j++;
        }
    }
    return output;
}
```

```
// Copyright 2023 Dr. Daly

#pragma once

#include <iostream>

class Matrix {
 public:
    Matrix() { matrix = new int[0]; }
    Matrix(size_t h, size_t w);
    Matrix(const Matrix& m);
    Matrix(Matrix&& m) noexcept;
    ~Matrix() { delete [] matrix; }

    Matrix& operator=(const Matrix& m);
    Matrix& operator=(Matrix&& m) noexcept;

    size_t height() const { return rows; }
    size_t width() const { return cols; }
    size_t size() const { return height() * width(); }

    int& operator()(size_t r, size_t c);
```

```
    int& at(size_t r, size_t c);
    int* operator[](size_t r);
    const int& operator()(size_t r, size_t c) const;
    const int& at(size_t r, size_t c) const;
    const int* operator[](size_t r) const;

    Matrix& operator+=(const Matrix& rhs);

 private:
    size_t rows;
    size_t cols;
    int * matrix;
};

Matrix operator+(const Matrix& lhs, const Matrix& rhs);
```

```
// Copyright 2023 Dr. Daly

#include "Matrix.hpp"

#include <algorithm>
#include <stdexcept>

Matrix::Matrix(size_t h, size_t w):
        rows(h), cols(w), matrix(new int[h * w]) {}

Matrix::Matrix(const Matrix& m):
        rows(m.rows), cols(m.cols), matrix(new int[m.size()]) {
    std::copy(m.matrix, m.matrix + m.size(), this->matrix);
}

Matrix::Matrix(Matrix&& m) noexcept:
        rows(m.rows), cols(m.cols), matrix(m.matrix) {
    m.matrix = nullptr;
}

Matrix& Matrix::operator=(const Matrix& m) {
    if (this != &m) {
        int* data = new int[m.size()];
        delete [] this->matrix;
        this->matrix = data;
        this->rows = m.rows;
        this->cols = m.cols;
        std::copy(m.matrix, m.matrix + m.size(), this->matrix);
    }
    return *this;
}

Matrix& Matrix::operator=(Matrix&& m) noexcept {
    if (this != &m) {
        this->rows = m.rows;
        this->cols = m.cols;
        delete [] this->matrix;
        this->matrix = m.matrix;
        m.matrix = nullptr;
    }
```

```
    return *this;
}

int& Matrix::operator()(size_t r, size_t c) {
    return matrix[r * cols + c];
}

int& Matrix::at(size_t r, size_t c) {
    if (r >= rows || c >= cols) {
        throw std::out_of_range("Parameters out of bounds!");
    }
    return matrix[r * cols + c];
}

int* Matrix::operator[](size_t r) {
    return &matrix[r * cols];
}

const int& Matrix::operator()(size_t r, size_t c) const {
    return matrix[r * cols + c];
}

const int& Matrix::at(size_t r, size_t c) const {
    if (r >= rows || c >= cols) {
        throw std::out_of_range("Parameters out of bounds!");
    }
    return matrix[r * cols + c];
}

const int* Matrix::operator[](size_t r) const {
    return &matrix[r * cols];
}

Matrix operator+(const Matrix& lhs, const Matrix& rhs) {
    if (lhs.height() != rhs.height() || lhs.width() != rhs.width()) {
        throw std::invalid_argument("Matrix dimensions don't match");
    }
    Matrix result(lhs.height(), lhs.width());
    for (size_t r = 0; r < lhs.height(); r++) {
        for (size_t c = 0; c < lhs.width(); c++) {
            result(r, c) = lhs(r, c) + rhs(r, c);
        }
    }
    return result;
}
Matrix& Matrix::operator+=(const Matrix& rhs) {
    if (height() != rhs.height() || width() != rhs.width()) {
        throw std::invalid_argument("Matrix dimensions don't match");
    }
    for (size_t r = 0; r < height(); r++) {
        for (size_t c = 0; c < width(); c++) {
            (*this)(r, c) += rhs(r, c);
        }
    }
    return *this;
```

```
96 }
```

7 PS6 RandWriter

7.1 Discussion:

This project gave me a lot of trouble at first. This was due to my lack of understanding of the central concept of the program. Once my friends helped explain it to me, I understood what it meant. We had to create Markov models of the order provided by the user. The model would store strings of characters based on each appearance and would be able to use those strings to create new strings of text or phrases. While the idea to the project was simple enough, I had to relearn how to use Boost Tests, which I struggled with. However, once I got it working, the usage of test based programming was easier to work with since I would not have to create a new input every time that I worked on the program, and it would automatically tell me whenever the function was working properly. The downside was that I had to know the output which was difficult when creating functions that were meant to be randomized, this resulted in some false negatives and false positives. By using test based programming, I ensured that the work I was putting in was actually doing something since I would know if the program was returning something incorrect or not. This sped up testing time significantly and allowed me to crack out this problem in a few hours. Using the algorithm library was also really helpful since it took care of a lot of the work when going through the data models.

7.2 Output:

```
ajay@DESKTOP-PCRG7HC:/mnt/c/Users/Ajay/Downloads/COMP4/PS6$ ./TextWriter 2 11 < input17.txt
Order: 2
Freq1: 5
Freq2: 1
kRand: g
Generate: gagagaggagg
ajay@DESKTOP-PCRG7HC:/mnt/c/Users/Ajay/Downloads/COMP4/PS6$
```

7.3 Codebase:

```
1 CC = g++
2 DEPS = RandWriter.hpp
```

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

.PHONY: lint all clean

all: lint Test TextWriter

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

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

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

clean:
  rm *.o TextWriter Test
lint:
  cpplint *.cpp *.hpp
```

```
// Copyright 2023 Ajay

#include <iostream>

#include "RandWriter.hpp"

int main(int c, char **argv) {
    int k, L;
    std::string orgText;
    k = std::stoi(argv[1]);
    L = std::stoi(argv[2]);

    std::string line;
    while (std::getline(std::cin, line)) {
        orgText += line;
    }
    RandWriter write(orgText, k);
    std::cout << "Order: " << write.orderK() << std::endl;
    std::cout << "Freq1: " << write.freq("ga") << std::endl;
    std::cout << "Freq2: " << write.freq("gg", 'c') << std::endl;
    std::cout << "kRand: " << write.kRand("ga") << std::endl;
    std::cout << "Generate: " << write.generate("ga", L) << std::endl;
    return 0;
}
```

```
// Copyright 2023 Ajay

#pragma once

#include <string>
#include <map>
#include <random>
#include <utility>

```

```
class RandWriter {
 public:
     // Create a Markov model of order k from given text
     // Assume that text has length at least k.
     RandWriter(std::string text, int k);
     // Order k of Markov model
     int orderK() const;
     // Number of occurences of kgram in text
     // Throw an exception if kgram is not length k
     int freq(std::string kgram) const;
     // Number of times that character c follows kgram
     // if order=0, return num of times that char c appears
     // (throw an exception if kgram is not of length k)
     int freq(std::string kgram, char c) const;
     // Random character following given kgram
     // (throw an exception if kgram is not of length k)
     // (throw an exception if no such kgram)
     char kRand(std::string kgram);
     // Generate a std::string of length L characters by simulating a
    trajectory
     // through the corresponding Markov chain. The first k characters of
     // the newly generated std::string should be the argument kgram.
     // Throw an excpetion if kgram is not of length k.
     // Assume that L is at least k
     std::string generate(std::string kgram, int L);

 private:
     std::mt19937 _randGen;
     std::string _original;
     size_t _order;
     std::map<std::string, int> _markov;
};
// Overload the stream insertion operator << and display the internal state
// of the Markov model. Print out the order, alphabet, and the frequencies
// of the k-grams and k+1-grams
```

```
// Copyright 2023 Ajay

#include <iostream>
#include <algorithm>
#include <numeric>
#include <stdexcept>
#include <chrono>
#include <vector>

#include "RandWriter.hpp"

// Create a Markov model of order k from given text
// Assume that text has length at least k.
RandWriter::RandWriter(std::string text, int k) {
    _original = text;
    _order = static_cast<size_t>(k);
    std::mt19937 rgen(std::chrono::system_clock::
        now().time_since_epoch().count());
    _randGen = rgen;
    for (size_t i = 0; i < text.size(); i++) {
```

```
21          // KGRAM
22          std::string kgram;
23          for (size_t j = 0; j < _order; j++) {
24              kgram += text[(i + j) % text.size()];
25          }
26          // FIND OCCURENCE
27          auto location = std::find_if(_markov.begin(), _markov.end(),
28              [kgram](auto item) {
29                  return (kgram == item.first);
30              });
31          // ADD TO MODEL, OR INCREASE OCCURENCE COUNT
32          if (location == _markov.end()) {
33              _markov[kgram] = 1;
34          } else {
35              location->second++;
36          }
37      }
38  }
39
40  // Order k of Markov model
41  int RandWriter::orderK() const { return _order; }
42
43  // Number of occurences of kgram in text
44  // Throw an exception if kgram is not length k
45  int RandWriter::freq(std::string kgram) const {
46      if (kgram.size() != _order)
47          throw std::runtime_error("Provided k-gram is not of correct length")
        ;
48      auto location = std::find_if(_markov.begin(), _markov.end(),
49          [kgram](auto item) {
50              return (kgram == item.first);
51          });
52      if (location != _markov.end())
53          return location->second;
54      return 0;
55  }
56
57  // Number of times that character c follows kgram
58  // if order=0, return num of times that char c appears
59  // (throw an exception if kgram is not of length k)
60  int RandWriter::freq(std::string kgram, char c) const {
61      if (_order != 0) {
62          if (kgram.size() != _order)
63              throw std::runtime_error("Provided k-gram is not of correct
        length");
64          kgram += c;
65          int times = 0;
66          for (size_t i = 0; i < _original.size(); i++) {
67              std::string text;
68              for (size_t j = 0; j < kgram.size(); j++) {
69                  text += _original[(i + j) % _original.size()];
70              }
71              if (kgram == text)
72                  times++;
73          }
```

```
        return times;
    }
    int num = std::count(_original.begin(), _original.end(), c);
    return num;
}

// Random character following given kgram
// (throw an exception if kgram is not of length k)
// (throw an exception if no such kgram)
char RandWriter::kRand(std::string kgram) {
    if (kgram.size() != _order)
        throw std::runtime_error("Provided k-gram is not of correct length")
    ;
    auto loc = std::find_if(_markov.begin(), _markov.end(),
        [kgram](auto item) {
            return (kgram == item.first);
        });
    if (loc == _markov.end()) {
        throw std::runtime_error("Provided k-gram does not exist in model");
    }
    std::vector<char> charvec;
    for (int i = 0; i < 127; i++) {
        int l = freq(kgram, static_cast<char>(i));
        for (; l > 0; l--) {
            charvec.push_back(static_cast<char>(i));
        }
    }
    std::uniform_int_distribution<int> range(0, charvec.size() - 1);
    int index = range(_randGen);
    char c = charvec[index];
    return c;
}

// Generate a std::string of length L characters by simulating a trajectory
// through the corresponding Markov chain. The first k characters of
// the newly generated std::string should be the argument kgram.
// Throw an excpetion if kgram is not of length k.
// Assume that L is at least k
std::string RandWriter::generate(std::string kgram, int L) {
    if (kgram.size() != _order)
        throw std::runtime_error("Provided k-gram is not of correct length")
    ;
    std::string out = kgram;
    for (int i = kgram.size(); i < L; i++) {
        std::string ngram = out.substr(out.size() - kgram.size(), kgram.size
    ());
        out += kRand(ngram);
    }
    return out;
}
```

```
// Copyright 2023 Ajay

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

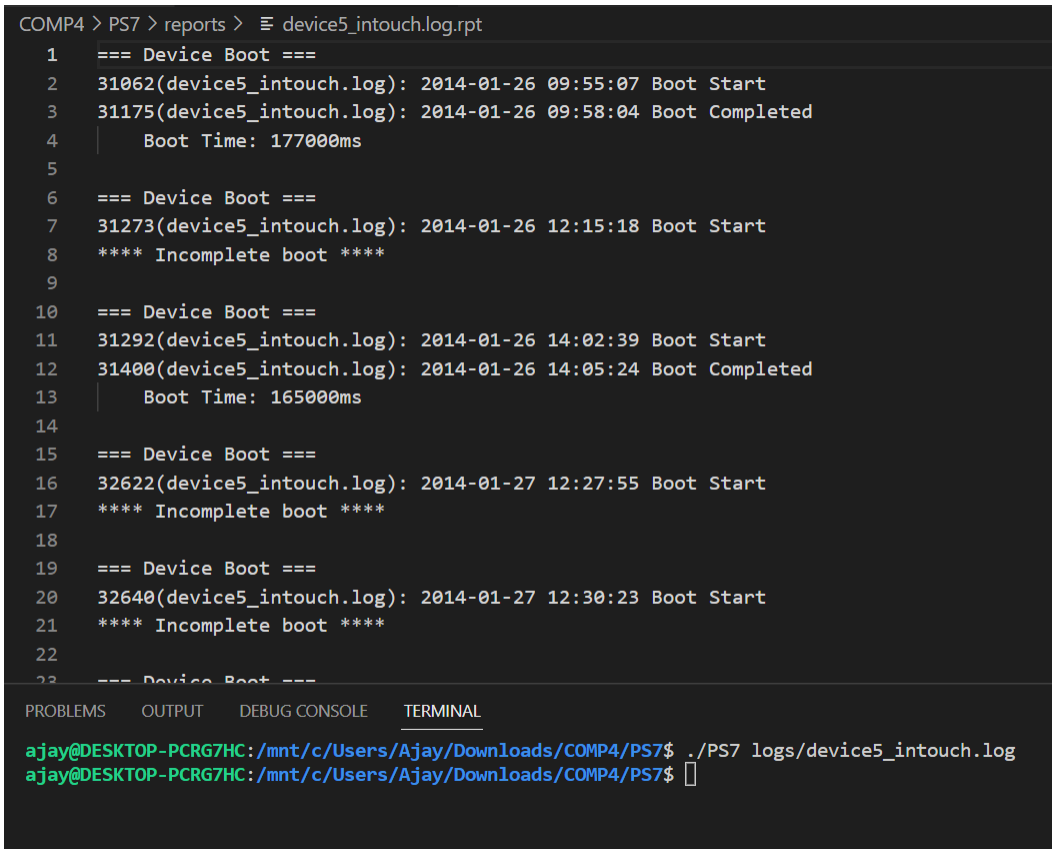
```
6
7  #include "RandWriter.hpp"
8
9  #define BOOST_TEST_DYN_LINK
10 #define BOOST_TEST_MODULE Main
11 #include <boost/test/unit_test.hpp>
12
13 BOOST_AUTO_TEST_CASE(passAll) {
14     RandWriter write("gagggagaggcgagaaa", 2);
15     BOOST_REQUIRE_EQUAL(write.orderK(), 2);
16     BOOST_REQUIRE_NO_THROW(write.freq("ga"));
17     BOOST_REQUIRE_NO_THROW(write.freq("gg", 'c'));
18     BOOST_REQUIRE_NO_THROW(write.kRand("ga"));
19     BOOST_REQUIRE_NO_THROW(write.generate("ga", 80));
20 }
21
22 BOOST_AUTO_TEST_CASE(failALL) {
23     RandWriter write("gagggagaggcgagaaa", 2);
24     BOOST_REQUIRE_EQUAL(write.orderK(), 2);
25     BOOST_REQUIRE_THROW(write.freq("z"), std::runtime_error);
26     BOOST_REQUIRE_THROW(write.freq("1", 'c'), std::runtime_error);
27     BOOST_REQUIRE_THROW(write.kRand("q"), std::runtime_error);
28     BOOST_REQUIRE_THROW(write.kRand("qe"), std::runtime_error);
29     BOOST_REQUIRE_THROW(write.generate("r", 16), std::runtime_error);
30 }
```

8 PS7 Kronos Log Parsing

8.1 Discussion:

Although this project was the final project, it was also the easiest due to the usage of regular expressions. By using regular expressions to check each line, finding data that was valid was incredibly easy and fast. Originally it would have been a lot longer each input substring would need to be checked against the valid string, but regex searching reduced the work to find valid strings from at least 10 steps to 1 line of code. By implementing well established libraries and using the objects that they provide, a lot of work can be cut down on the programmer's end which results in efficient and concise code. By getting out of my comfort zone and constantly using new utilities, I can make the programs faster and more effective than I've ever made them before.

8.2 Output:



8.3 Codebase:

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

.PHONY: lint all clean

all: lint PS7

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

PS7: main.o
  $(CC) $(CFLAGS) -o $@ $^ $(LIB)

clean:
  rm *.o PS7
lint:
  cpplint *.cpp *.hpp
```

```
// Copyright 2023 Ajay

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <boost/regex.hpp>

#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"

int main(int argc, char** argv) {
    if (argc != 2) {
        std::cout << "Incorrect Input" << std::endl;
        exit(1);
    }
    std::string filename = argv[1];
    std::ifstream fp(filename);
    if (!fp.is_open()) {
        throw std::runtime_error("File could not be opened");
    }
    filename = filename.substr(5);
    std::ofstream fout("reports/" + filename + ".rpt");
    if (!fout.is_open()) {
        throw std::runtime_error("File could not be opened");
    }
    std::string input;
    boost::regex boot{"\\(log.c.166\\) server started"};
    boost::regex close{"oejs.AbstractConnector:Started
    SelectChannelConnector"};
    int started = 0;
    boost::posix_time::ptime beginning;
```

```
32      std::string nesting;
33      std::vector<char> charvec;
34      for (int i = 0; std::getline(fp, input); ++i) {
35          for (size_t j = 0; j < input.size(); j++) {
36              if (input[j] == '{') {
37                  charvec.push_back(input[j]);
38              }
39          }
40          if (regex_search(input, boot)) {
41              if (started == 1) {
42                  fout << "**** Incomplete boot ****" << std::endl << std::
    endl;
43                  nesting = "";
44                  started = 0;
45              }
46              std::string nested;
47              for (size_t k = 0; k < charvec.size(); k++) {
48                  nested += charvec[k];
49              }
50              nesting = nested;
51              std::string start;
52              start = input.substr(0, 19);
53              boost::posix_time::ptime startdatetime(
54                  boost::posix_time::time_from_string(start));
55              beginning = startdatetime;
56              fout << "=== Device Boot ===" << std::endl;
57              fout << i << "(" << filename << "): " <<
58                  start << " Boot Start" << std::endl;
59              started = 1;
60          } else if (regex_search(input, close)) {
61              std::string nestcheck;
62              for (size_t k = 0; k < charvec.size(); k++) {
63                  nestcheck += charvec[k];
64              }
65              if (nestcheck != nesting)
66                  fout << "**** Incomplete boot ****" << std::endl << std::
    endl;
67              if (started == 0)
68                  fout << "**** Incomplete boot ****" << std::endl << std::
    endl;
69              std::string end;
70              end = input.substr(0, 19);
71              boost::posix_time::ptime enddatetime(
72                  boost::posix_time::time_from_string(end));
73              boost::posix_time::time_duration td = enddatetime - beginning;
74              fout << i << "(" << filename << "): " << end
75                  << " Boot Completed" << std::endl << "\tBoot Time: " <<
76                      td.total_milliseconds() << "ms" << std::endl << std::
    endl;
77              started = 0;
78          }
79          for (size_t l = 0; l < input.size(); l++) {
80              if (input[l] == '}') {
81                  if (charvec.back() == '{')
82                      charvec.pop_back();
```

```
            }
        }
    }
    if (started == 1) {
        fout << "**** Incomplete boot ****" << std::endl << std::endl;
    }
    return 0;
}
```