Computing IV Sec 204: Project Portfolio

Hunter M Hasenfus

Spring 2023

Contents

TIME TO COMPLETE PORTFOLIO : 12 Hours

1 PS0: Hello World with SFML

1.1 Discussion:

The first project of CompIV 22 is **Hello World with SFML**. In this assignment, At first we set up our environment by installing the SFML library. We also check for the newest version of the C++. The assignment tasked me with displaying two sprites to the screen, one being a green ball and another being a graphic of my choice from the internet.

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

This assignment did not require any of the key algorithm, Data structures and OO Designs, as the project it self is basic project with the code provided and we just had to add SFML sprite and few Keyboard events to complete the project. In the assignment I implemented a feature that takes input from the keyboard and then moves the sprite around the screen. By formatting conditional statements centered around key pressed I was able to nudge the sprite in the given direction based on the key chosen. This was an interesting project in learning how other libraries are implemented and attempting to use their objects and data structures. I learned how to configure the setup of 3rd party libraries, and I learned how to the libraries and paths are setup on linux machines.

1.3 Images used:

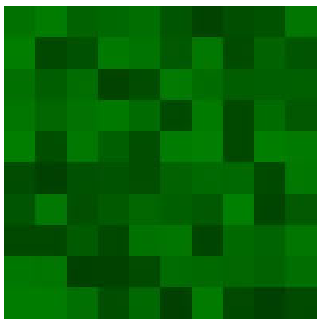


Figure 1: Sprite Image

1.4 What I learned :

I learned to use SFML for the first time and also how to add events and manipulate them in the SFML field. Overall, It was fun to do this assignment, as everything for me in this assignment was new an amazing.

1.5 Acknowledgements:

- `https://www.sfml-dev.org/tutorials/2.5/`
- `https://youtu.be/axIgxBQVBg0`

1.6 Codebase

main.cpp:
The main file where the code runs and provides the valid output as shown in figure 2.

```
// Hunter M Hasenfus
// Computing IV
// PS0
// Dr. Rykalova
// Jan 30, 2023
// Demonstrate ability to compile program using SFML and try out various
   functions.
```

```cpp

#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
int main()
{
    // Create the main window
    sf::RenderWindow window(sf::VideoMode(800, 600), "SFML window");
    // Load a sprite to display
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

    sf::Texture texture;
    if (!texture.loadFromFile("sprite.jpg"))
        return EXIT_FAILURE;
    sf::Sprite sprite(texture);
    // Create a graphical text to display

    // Start the game loop
    while (window.isOpen())
    {
        // Process events
        sf::Event event;
        while (window.pollEvent(event))
        {
            // Close window: exit
            if (event.type == sf::Event::Closed)
                window.close();
            if ((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Left))
                sprite.move(-10,0);
            if ((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Right))
                sprite.move(10,0);
            if ((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Up))
                sprite.move(0,-10);
            if ((event.type == sf::Event::KeyPressed) && (event.key.code == sf::Keyboard::Down))
                sprite.move(0,10);
            if (event.type == sf::Event::MouseButtonPressed)
                sprite.rotate(90);

        }
        // Clear screen
        window.clear();
        // Draw the sprite
        window.draw(shape);
        window.draw(sprite);
        // Update the window
        window.display();
    }
    return EXIT_SUCCESS;
}
```

1.7 Output:

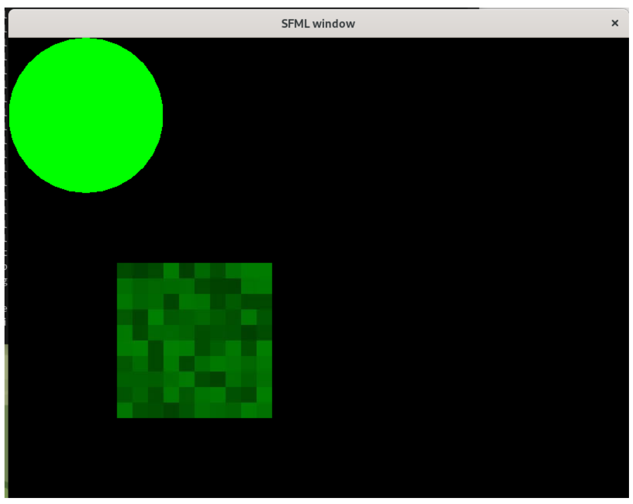


Figure 2: Output of PS0 Assignment

2 PS1a: Linear Feedback Shift Register

2.1 Discussion:

The assignment focused on constructing a class based around the LFSR, providing opaque object design through a header file, and using the boost framework to link the design to a series of tests that verified the correct implementation of the code. The ps1a assignment is an implementation of **LFSR(Linear Feedback shift Register) which is the Fibonacci LFSR**. This assignment is used for the ps1b i.e PhotoMagic. In this project, there are two main functions i.e, step() and generate(), The step() funtion is used for left shifting the one bit of the given seed, along the lsb is the result of the tap positions. These tap positions use the XOR operations and later it gives the result. The generate() generates the states according to the given k inputs.
XOR Truth Table:

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

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

I accomplished all parts of the assignment, utilizing a simple vector of ints for the register bits, which allowed me to easily configure the spots and access all elements with random access memory. **The Tap position algorithm is as follows:**

```
 int _TAPbitvalue = funXOR(rgs[0], rgs[2]);
_TAPbitvalue = funXOR(_TAPbitvalue, funGetBit(rgs[3]));
_TAPbitvalue = funXOR(_TAPbitvalue, funGetBit(rgs[5]));

```

2.3 What I learned :

I learned how to implement again the boost library tests.

2.4 Codebase

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

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lboost_unit_test_framework
DEP = FibLFSR.h
OBJS = FibLFSR.o test.o

all:ps1a

ps1a: $(OBJS)
    $(CC) -o ps1a $(OBJS) $(LFLAGS)
FibLFSR.o: FibLFSR.cpp FibLFSR.h
    $(CC) $(CFLAGS) FibLFSR.cpp
test.o: test.cpp FibLFSR.h
    $(CC) $(CFLAGS) test.cpp $(LFLAGS)

clean :
    rm ps1a *.o
```

FibLFSR.h:

```cpp
#ifndef FIBLFSR_H
#define FIBLFSR_H

#include <string>
#include <vector>
using namespace std;


class FibLFSR {
public:
    FibLFSR(string seed);
    int step();
    int generate(int k);

private:
    vector<int> registers;
    vector<int> tapPositions = {2,3,5};
};



#endif
```

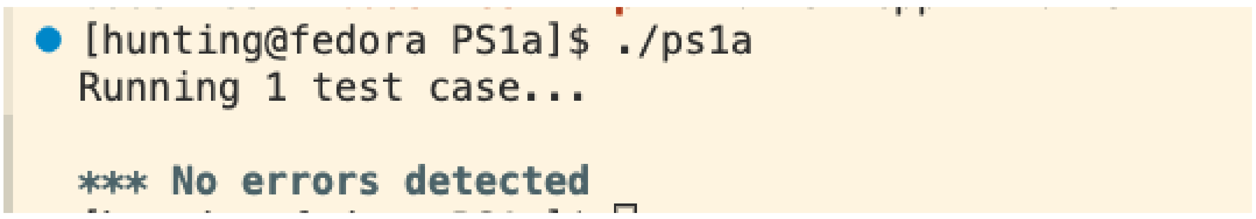
FibLFSR.cpp:

```cpp
// Hunter M Hasenfus
// Computing IV
// Prof. Rykalova
//
//


#include <iostream>
#include <cstdlib>
#include <string>
#include <vector>
#include <queue>
#include <cmath>
#include "FibLFSR.h"

using namespace std;


// int main(int arg, char* argv[])
// {
//     string test = "1011011000110110";
//     FibLFSR tes(test);
//     tes.generate(5);
//     return 0;
// }

#define PRINT 0
FibLFSR::FibLFSR(string seed)
{
    this->registers = vector<int>(seed.size());
    for(long unsigned int i = 0; i < seed.size(); i++)
    {
        this->registers[i] = seed[i]-48;
    }
}

int FibLFSR::step()
{
    int x = this->registers[0];
    for(long unsigned int i = 0; i < this->tapPositions.size(); i++)
    {
        x = this->registers[this->tapPositions[i]] ^ x;
    }
    for(long unsigned int i = 0; i < this->registers.size()-1; i++)
    {
        this->registers[i] = this->registers[i+1];
    }
    this->registers[this->registers.size()-1] = x;

    return x;
}

int FibLFSR::generate(int k)
{
    int x= 0, y;
    for(int i = 0; i < k; i++)
    {
        y = this->step();
        x+= y * pow(2,k-1-i);
    }
    if(PRINT)
    {
        for(long unsigned int j = 0; j < this->registers.size(); j++)
        {
            cout << this->registers[j];
        }
        cout << " " << x << endl;
    }

    return x;
}
```

test.cpp:

```cpp
// Dr. Rykalova
// test.cpp for PS1a
// updated 1/31/2020

#include <iostream>
#include <string>

#include "FibLFSR.h"

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

BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {

  FibLFSR l("1011011000110110");
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 1);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 0);
  BOOST_REQUIRE(l.step() == 1);

  FibLFSR l2("1011011000110110");
  BOOST_REQUIRE(l2.generate(9) == 51);

  FibLFSR l3("0000000000000000");
  BOOST_REQUIRE(l3.step() != 1);
  BOOST_REQUIRE(l3.generate(100) == 0);

  BOOST_REQUIRE(sizeof(l3.generate(100)) == sizeof(int));
  BOOST_REQUIRE(sizeof(l3.step()) == sizeof(int));
  BOOST_REQUIRE(sizeof(l3.generate(100)) == sizeof(l3.step()));
  BOOST_REQUIRE(l2.step() == 1 || l2.step() == 0);
}
```

2.5 Output:



```
[hunting@fedora PS1a]$ ./ps1a
Running 1 test case...

*** No errors detected
```

3 PS1b: Application of LFSR with SFML library

3.1 Discussion:

This assignment branched off of the work finished in part (a). The linear feedback shift register was implemented in a way to encode images. The LFSR provides the perception of randomness and almost the trapdoor mechanism of encryption. The program steps through each pixel in the respective image and uses the object oriented nature of LFSR to reconfigure each component of the RGB based on the seed provided in the command line argument. As shown in the images below, the image can be encoded and decoded without any loss of information.

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

I used The Vector STL in the LFSR as it is much easier to use and I felt flexible in it. I used Two windows and two sprites for showing the difference between the normal cat image and the encoded and decoded image.

3.3 Images used:

Figure 3: Cat Image

3.4 What I learned :

In this assignment it was very eye opening and enlightening. First we spent the time to develop our own library for LFSR and then promptly created an application that puts the technology into work. The assignment both exercised my understanding of the SFML library and also how to utilize opaque object design.

3.5 Acknowledgements :

- `https://www.sfml-dev.org/tutorials/2.5/`

3.6 Codebase

Makefile:
This Makefile is contains no lint but it includes the flags as well as it is extension of the ps1a Makefile.

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
DEP = FibLFSR.h
OBJS = FibLFSR.o PhotoMagic.o

all:ps1b

ps1b: $(OBJS)
    $(CC) -o ps1b $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
    $(CC) -c $(CFLAGS) $< -o $@ $(LFLAGS)

cleanall : cleanobj
    rm ps1b

cleanobj :
    rm *.o
```

PhotoMagic.cpp:
This file is the main file where the reading and writing also the encoding and decoding of the image takes place. This file gives the output in two windows. Input window and output window of the file.

```
// Hunter M Hasenfus
// Computing IV
// Prof. Rykalova
//
//

#include <iostream>
#include <cstdlib>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "FibLFSR.h"

#define DISPLAY 1

using namespace std;

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

int main(int argc, char* argv[])
{
    if (argc != 4){
        cout << argc << endl;
        return -1;}

    FibLFSR seed(argv[3]);

    sf::Image input;
    input.loadFromFile(argv[1]);
    sf::Image output;
    output.loadFromFile(argv[1]);;

    transform(output, &seed);

    sf::Texture texture1;
    texture1.loadFromImage(input);
    sf::Texture texture2;
    texture2.loadFromImage(output);
    sf::Sprite sprite1(texture1);
    sf::Sprite sprite2(texture2);

    sf::RenderWindow window1(sf::VideoMode(sprite1.getTexture()->getSize().x
     * sprite1.getScale().x, sprite1.getTexture()->getSize().y * sprite1.
    getScale().y ), "Before");
    sf::RenderWindow window2(sf::VideoMode(sprite1.getTexture()->getSize().x
     * sprite1.getScale().x, sprite1.getTexture()->getSize().y * sprite1.
    getScale().y ), "After");
    if(DISPLAY)
    {
        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(sprite1);
            window1.display();
            window2.clear();
            window2.draw(sprite2);
            window2.display();
        }
    }

    output.saveToFile(argv[2]);



    return 0;
}

void transform(sf::Image& input, FibLFSR* seed)
{
    sf::Color p;
    cout << input.getSize().x << " " << input.getSize().y << endl;
    for(int i = 0; i < input.getSize().x; i++)
    {
        for(int j = 0; j < input.getSize().y; j++)
        {
            p = input.getPixel(i,j);
            p.r = p.r ^ seed->generate(8);
            p.g = p.g ^ seed->generate(8);
            p.b = p.b ^ seed->generate(8);
            input.setPixel(i,j,p);
        }
    }
}
```

FibLFSR.h:

```
#ifndef FIBLFSR_H
#define FIBLFSR_H

#include <string>
#include <vector>
using namespace std;


class FibLFSR {
public:
    FibLFSR(string seed);
    int step();
    int generate(int k);

private:
    vector<int> registers;
    vector<int> tapPositions = {2,3,5};
};



#endif
```

FibLFSR.cpp:

```
// Hunter M Hasenfus
// Computing IV
// Prof. Rykalova
//
//


#include <iostream>
#include <cstdlib>
#include <string>
#include <vector>
#include <queue>
#include <cmath>
#include "FibLFSR.h"

using namespace std;


// int main(int arg, char* argv[])
// {
//     string test = "1011011000110110";
//     FibLFSR tes(test);
//     tes.generate(5);
//     return 0;
// }

#define PRINT 0
FibLFSR::FibLFSR(string seed)
{
    this->registers = vector<int>(seed.size());
    for(long unsigned int i = 0; i < seed.size(); i++)
    {
        this->registers[i] = seed[i]-48;
    }
}

int FibLFSR::step()
{
    int x = this->registers[0];
    for(long unsigned int i = 0; i < this->tapPositions.size(); i++)
    {
        x = this->registers[this->tapPositions[i]] ^ x;
    }
    for(long unsigned int i = 0; i < this->registers.size()-1; i++)
    {
        this->registers[i] = this->registers[i+1];
    }
    this->registers[this->registers.size()-1] = x;

    return x;
}

int FibLFSR::generate(int k)
{
    int x= 0, y;
    for(int i = 0; i < k; i++)
    {
        y = this->step();
        x+= y * pow(2,k-1-i);
    }
    if(PRINT)
    {
        for(long unsigned int j = 0; j < this->registers.size(); j++)
        {
            cout << this->registers[j];
        }
        cout << " " << x << endl;
    }

    return x;
}
```

3.7 Output:

Figure 4: Encoded Image

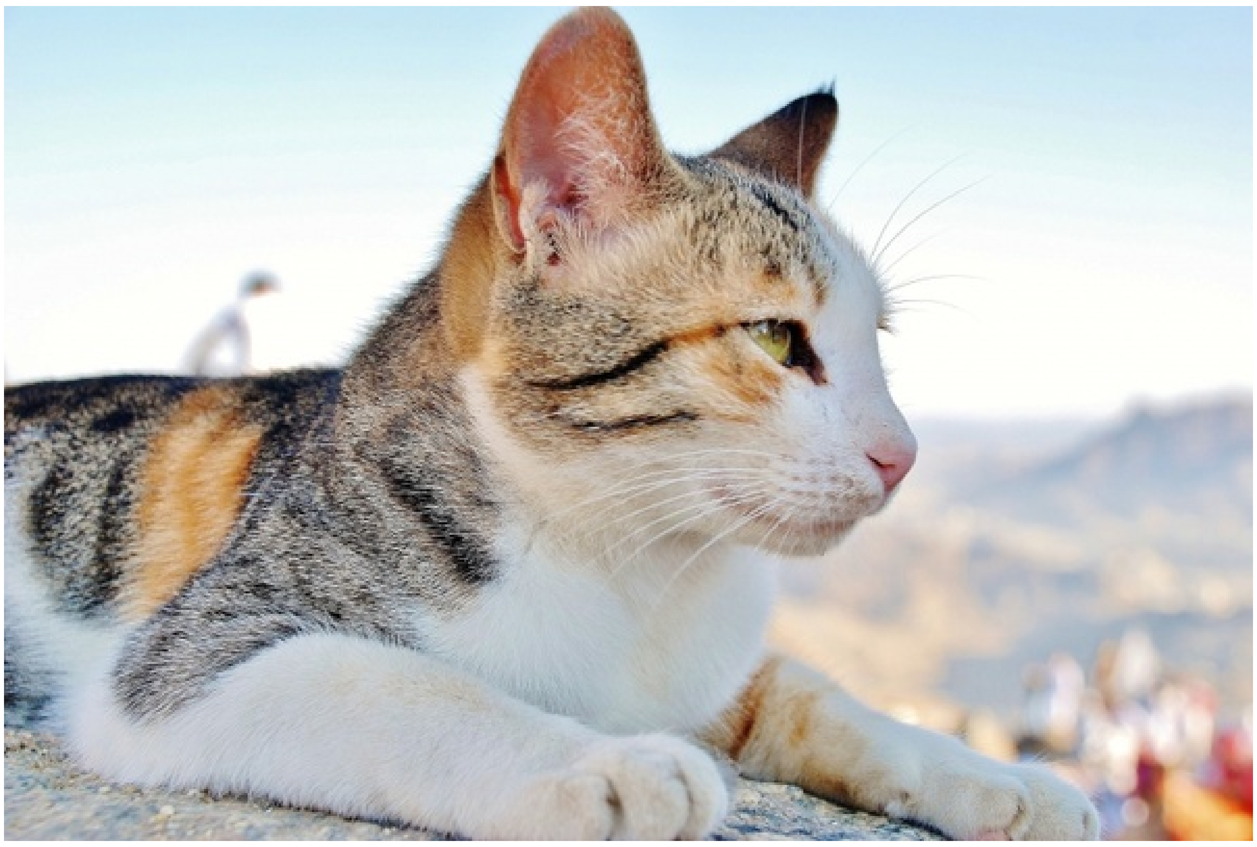


Figure 5: Decode Image

4 PS2a: Sokaban Visual component

4.1 Discussion:

The task for this project was in recreating the age-old game of Sokaban. For the first part of the assignment(a), I had to create the visual aspect. Beginning with an opague object design, I created an object for the game and had the rest of the functions within it. I did this by inheriting the public Drawable class from the SFML library, and by overloading the draw function in a virtual format. This allowed myself the ability to draw the entire class to the screen and utilize members and data structures in the process.

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

The Sokaban game is inherently grid like, moving boxes around the screen to loading locations. I designed the game using an array of characters that instantiated at the start of each level, the grid included the movable locations as well as the wall locations. For the other important data points like player location, box location, and dropoff location I created different data structures to hold them including a pair of integers and two vectors consisting of pairs of integers. The first time I implemented this, I had all of the information embedded into the background grid, however I soon realized that tht it would have been difficult to model multiple figures at the same location. These pieces of data were not only dynamic, but they were a layer above the background. By removing them from the grid and giving them their own data structure I was able to always draw the background, and be able to search for their locations very easily.

4.3 What I learned :

In this project I learned a lot about foreground and background visual effects and the utilization of different data structures to achieve the effect. I learned how to operate objected oriented design into existing libraries with the overloading of the draw function. I learned about the relationship between textures and sprites.

4.4 Acknowledgments :

Links :

- `https://www.sfml-dev.org/tutorials/2.5/graphics-sprite.php`
- `https://icarus.cs.weber.edu/~dab/cs1410/textbook/11.Operators/io_overload.html`
- `https://www.cplusplus.com/reference/vector/vector/`
- `https://www.sfml-dev.org/documentation/2.5.1/classsf_1_1Drawable.php`
- `https://stackoverflow.com/questions/34458791/making-custom-types-drawable-with-sfml`

4.5 Codebase

Makefile

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

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
DEP = Sokaban.h
OBJS = main.o Sokaban.o

all: Sokaban

Sokaban: $(OBJS)
```

```
	$(CC) -o KSGuitarSim $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
	$(CC) -c $(CFLAGS) $< -o $@ $(LFLAGS)

lint:
	cpplint.py --filter=-runtime/references,-build/c++11 --root=. *


cleanall : cleanobj
	rm KSGuitarSim

cleanobj :
	rm *.o
```

main.cpp

```
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "Sokaban.h"


using namespace std;



int main(int arg, char* argc[])
{
    Sokaban s;
    for(int i = 1; i < arg; i++)
    {
        fstream f(argc[i], ios::in);
        f >> s;
        sf::RenderWindow window(sf::VideoMode(s.getWidth() * 64, s.getHeight
    () * 64), argc[i]);

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

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

    return 0;
}
```

Sokaban.h

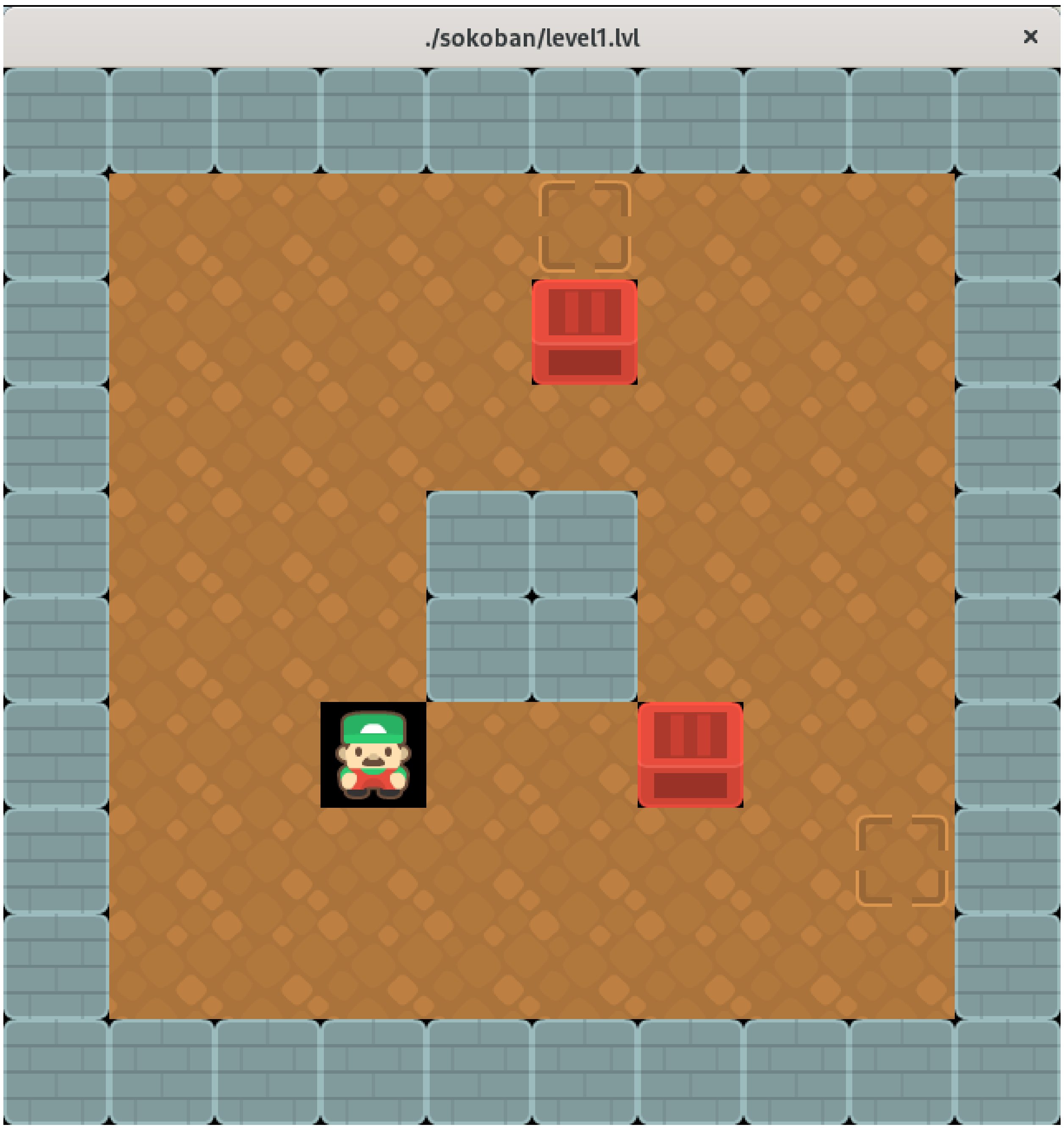
```cpp
#ifndef SOKABAN_H
#define SOKABAN_H


#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

using namespace std;

class Sokaban: public sf::Drawable{
    public:
        Sokaban();
        friend istream& operator>>(istream& in, Sokaban& s);
        ~Sokaban();
        int getHeight() const;
        int getWidth() const;
        void setHeight(int h);
        void setWidth(int w);
        char operator[](pair<int,int> p) const;
    private:
        virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
    const;
        int height;
        int width;
        char** map;
        sf::Texture Wall;
        sf::Texture Box;
        sf::Texture Empty;
        sf::Texture Storage;
        sf::Texture Player[4];
        pair<int,int> playerLocation;
        vector<pair<int,int>> boxLocation;
        vector<pair<int,int>> storageLocation;
};

#endif
```

Sokoban.cpp

```cpp
#include "Sokaban.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <utility>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>



using namespace std;

Sokaban::Sokaban(){
    Wall.loadFromFile("sokoban/block_06.png");
    Box.loadFromFile("sokoban/crate_03.png");
    Empty.loadFromFile("sokoban/ground_01.png");
    Storage.loadFromFile("sokoban/ground_04.png");
    Player[0].loadFromFile("sokoban/player_05.png");
    Player[1].loadFromFile("sokoban/player_08.png");
    Player[2].loadFromFile("sokoban/player_17.png");
    Player[3].loadFromFile("sokoban/player_20.png");
}

Sokaban::~Sokaban(){
    for(int i = 0; i < height; i++){
        delete[] map[i];
    }
    delete[] map;
}
int Sokaban::getHeight() const {return height;}
int Sokaban::getWidth() const {return width;}
void Sokaban::setHeight(int h) {height = h;}
void Sokaban::setWidth(int w) {width = w;}
char Sokaban::operator[](pair<int,int> p) const {return map[p.first][p.
   second];}

istream& operator>>(istream& in, Sokaban& s){
    in >> s.height >> s.width;
    s.map = new char*[s.height];
    for(int i = 0; i < s.height; i++){
        s.map[i] = new char[s.width];
        for(int j = 0; j < s.width; j++){
            in >> s.map[i][j];
            if(s.map[i][j] == '@'){
                s.playerLocation.first = i;
                s.playerLocation.second = j;
            }
            if(s.map[i][j] == 'A'){
                s.boxLocation.push_back(make_pair(i,j));}
            if(s.map[i][j] == 'a'){
                s.storageLocation.push_back(make_pair(i,j));
            }
        }
    }
    return in;
}

void Sokaban::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            sf::Sprite sprite;
            switch(map[i][j]){
                case '#':
                    sprite.setTexture(Wall);
                    break;
                case 'A':
                    sprite.setTexture(Box);
                    break;
                case '.':
                    sprite.setTexture(Empty);
                    break;
                case 'a':
                    sprite.setTexture(Storage);
                    break;
                case '@':
                    sprite.setTexture(Player[0]);
                    break;
                case 'S':
                    sprite.setTexture(Player[0]);
                    break;
                case 'N':
                    sprite.setTexture(Player[1]);
                    break;
                case 'E':
                    sprite.setTexture(Player[2]);
                    break;
                case 'W':
                    sprite.setTexture(Player[3]);
                    break;
            }
            sprite.setPosition(j * 64, i * 64);
            target.draw(sprite);
        }
    }
}
```

```cpp
#include "Sokaban.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <utility>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>



using namespace std;

Sokaban::Sokaban(){
    Wall.loadFromFile("sokoban/block_06.png");
    Box.loadFromFile("sokoban/crate_03.png");
    Empty.loadFromFile("sokoban/ground_01.png");
    Storage.loadFromFile("sokoban/ground_04.png");
    Player[0].loadFromFile("sokoban/player_05.png");
    Player[1].loadFromFile("sokoban/player_08.png");
    Player[2].loadFromFile("sokoban/player_17.png");
    Player[3].loadFromFile("sokoban/player_20.png");
}

Sokaban::~Sokaban(){
    for(int i = 0; i < height; i++){
        delete[] map[i];
    }
    delete[] map;
}
int Sokaban::getHeight() const {return height;}
int Sokaban::getWidth() const {return width;}
void Sokaban::setHeight(int h) {height = h;}
void Sokaban::setWidth(int w) {width = w;}
char Sokaban::operator[](pair<int,int> p) const {return map[p.first][p.
   second];}

istream& operator>>(istream& in, Sokaban& s){
    in >> s.height >> s.width;
    s.map = new char*[s.height];
    for(int i = 0; i < s.height; i++){
        s.map[i] = new char[s.width];
        for(int j = 0; j < s.width; j++){
            in >> s.map[i][j];
            if(s.map[i][j] == '@'){
                s.playerLocation.first = i;
                s.playerLocation.second = j;
            }
            if(s.map[i][j] == 'A'){
                s.boxLocation.push_back(make_pair(i,j));}
            if(s.map[i][j] == 'a'){
                s.storageLocation.push_back(make_pair(i,j));
            }
        }
    }
    return in;
}

void Sokaban::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            sf::Sprite sprite;
            switch(map[i][j]){
                case '#':
                    sprite.setTexture(Wall);
                    break;
                case 'A':
                    sprite.setTexture(Box);
                    break;
                case '.':
                    sprite.setTexture(Empty);
                    break;
                case 'a':
                    sprite.setTexture(Storage);
                    break;
                case '@':
                    sprite.setTexture(Player[0]);
                    break;
                case 'S':
                    sprite.setTexture(Player[0]);
                    break;
                case 'N':
                    sprite.setTexture(Player[1]);
                    break;
                case 'E':
                    sprite.setTexture(Player[2]);
                    break;
                case 'W':
                    sprite.setTexture(Player[3]);
                    break;
            }
            sprite.setPosition(j * 64, i * 64);
            target.draw(sprite);
        }
    }
}
```

4.6 Output:



5 PS2b: Sokoban Game Mechanics

5.1 Discussion:

This assignment was implementing the actions, mechanics, and dynamism into the game. By transferring the player, the boxes, and the locations into vectors or pairs, it made it very easy to search and validate locations. It also made it very easy to move pieces to new location. The search and move features were very important and helpful for establishing the edge cases in the game.

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

In this assignment as mentioned prior, I put the pieces and the locations into vectors. I coupled this choice with an easily integrated function from the standard library find if, this takes iterators as well as functors. The functor I applied would check certain locations and return the iterator or the end depending on the input. This was very helpful for the not only checking the wall locations, but also checking if there were multiple boxes in front of each other.

Here is an excerpt from the code, I implemented a while loop with the find if function to simulate moving the line of boxes.

code:

```
    while(x != boxLocation.end())
            {
                if(map[playerLocation.first - (i+1)][playerLocation.second] == '.' || map[playerLocation.first - (i+1)][playerLocation.second] == 'a')
                {
                    boxes.push_back(*x);
                }
                i++;
                x = find_if(boxLocation.begin(), boxLocation.end(), [this,i](pair<int,int> p){return p.first == playerLocation.first - i && p.second == playerLocation.second;});
            }
            if(map[playerLocation.first - (i)][playerLocation.second] == '.' || map[playerLocation.first - (i)][playerLocation.second] == 'a')
            {
                for(auto y : boxes)
                {
                    vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [y](pair<int,int> p){return p.first == y.first && p.second == y.second;});
                    x->first--;
                }
                playerLocation.first--;
            }


```

5.3 What I learned :

I learned how to implement features that have higher levels of complexity. Adding the feature of multiple boxes took a certain level of implementation. I also learned how to use lambda functions and to take advantage of the already implemented algorithms in the standard library. I also learned how to lint my code, and what proper programming looks like.

5.4 Codebase

Makefile:

This Makefile contains the linting too.

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
DEP = Sokaban.h
OBJS = main.o Sokaban.o

all: Sokaban

Sokaban: $(OBJS)
    $(CC) -o KSGuitarSim $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
    $(CC) -c $(CFLAGS) $< -o $@ $(LFLAGS)

lint:
    cpplint.py --filter=-runtime/references,-build/c++11 --root=. *


cleanall : cleanobj
    rm KSGuitarSim

cleanobj :
    rm *.o
```

main.cpp

```
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "Sokaban.h"


using namespace std;

int main(int arg, char* argc[])
{

    if(arg == 1)
        return -1;
    for(int i = 1; i < arg; i++)
    {
        Sokaban s;
        fstream f(argc[i], ios::in);
        f >> s;

        sf::RenderWindow window(sf::VideoMode(s.getWidth() * 64, s.getHeight
    () * 64), argc[i]);
        while (window.isOpen())
        {
            sf::Event event;
            while (window.pollEvent(event))
            {
                if (event.type == sf::Event::Closed)
                    window.close();
                if (event.type == sf::Event::KeyPressed)
                {
                    if(event.key.code == sf::Keyboard::R)
                    {
                        f.clear();
                        f.seekg(0, ios::beg);
                        f >> s;
                    }
                    else
                        s.movePlayer(event.key.code);
                }

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

    return 0;
}
```

Sokoban.h

```
#ifndef SOKABAN_H
#define SOKABAN_H


#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

using namespace std;

class Sokaban: public sf::Drawable{
    public:
        Sokaban();
        friend istream& operator>>(istream& in, Sokaban& s);
        ~Sokaban();
        int getHeight() const;
        int getWidth() const;
        void setHeight(int h);
        void setWidth(int w);
        char operator[](pair<int,int> p) const;
        void movePlayer(sf::Keyboard::Key k);
        void isWon();
    private:
        virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
     const;
        int height;
        int width;
        char** map;
        sf::Texture Wall;
        sf::Texture Box;
        sf::Texture GBox;
        sf::Texture Empty;
        sf::Texture Storage;
        sf::Texture Player[4];
        sf::Texture Win;
        bool won;
        pair<int,int> playerLocation;
        vector<pair<int,int>> boxLocation;
        vector<pair<int,int>> storageLocation;
        char direction;
};

#endif
```

Sokoban.cpp

```
#include "Sokaban.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <utility>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>



using namespace std;

Sokaban::Sokaban(){
    Wall.loadFromFile("sokoban/block_06.png");
    GBox.loadFromFile("sokoban/crate_03G.png");
    Box.loadFromFile("sokoban/crate_03.png");
    Empty.loadFromFile("sokoban/ground_01.png");
    Storage.loadFromFile("sokoban/ground_04.png");
    Player[1].loadFromFile("sokoban/player_05.png");
    Player[0].loadFromFile("sokoban/player_08.png");
    Player[2].loadFromFile("sokoban/player_17.png");
    Player[3].loadFromFile("sokoban/player_20.png");
    Win.loadFromFile("sokoban/win.png");

}
void Sokaban::isWon()
{
    if (storageLocation.size() < boxLocation.size()){
        for(auto y : storageLocation)
        {
            vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(),
     boxLocation.end(), [y](pair<int,int> p){return p.first == y.first && p.
    second == y.second;});
            if(x == boxLocation.end()){
                won = false;
                return;}
        }}
    else{
        for(auto y : boxLocation)
        {
            vector<pair<int,int>>::iterator x = find_if(storageLocation.
    begin(), storageLocation.end(), [y](pair<int,int> p){return p.first == y
    .first && p.second == y.second;});
            if(x == storageLocation.end()){
```

```cpp
                won = false;
                return;}
        }}

    won = true;
}
void Sokaban::movePlayer(sf::Keyboard::Key k)
{
    vector<pair<int,int>> boxes;
    int i = 1;
    if(k == sf::Keyboard::Up || k == sf::Keyboard::W)
    {
        if(map[playerLocation.first - 1][playerLocation.second] == '.' || map[playerLocation.first - 1][playerLocation.second] == 'a')
        {
            vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [this,i](pair<int,int> p){return p.first == playerLocation.first - i && p.second == playerLocation.second;});
            if(x != boxLocation.end())
            {
                while(x != boxLocation.end())
                {
                    if(map[playerLocation.first - (i+1)][playerLocation.second] == '.' || map[playerLocation.first - (i+1)][playerLocation.second] == 'a')
                    {
                        boxes.push_back(*x);
                    }
                    i++;
                    x = find_if(boxLocation.begin(), boxLocation.end(), [this,i](pair<int,int> p){return p.first == playerLocation.first - i && p.second == playerLocation.second;});
                }
                if(map[playerLocation.first - (i)][playerLocation.second] == '.' || map[playerLocation.first - (i)][playerLocation.second] == 'a')
                {
                    for(auto y : boxes)
                    {
                        vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [y](pair<int,int> p){return p.first == y.first && p.second == y.second;});
                        x->first--;
                    }
                    playerLocation.first--;
                }
            }
            else
            {
                playerLocation.first--;
            }
        }
        direction = 'N';
    }
    if(k == sf::Keyboard::Down || k == sf::Keyboard::S)
    {
        if(map[playerLocation.first + 1][playerLocation.second] == '.' || map[playerLocation.first + 1][playerLocation.second] == 'a')
        {
            vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(),
    boxLocation.end(), [this](pair<int,int> p){return (p.first == (playerLocation.first + 1)) && p.second == playerLocation.second;});
            if(x != boxLocation.end())
            {
                while(x != boxLocation.end())
                {
                    if(map[playerLocation.first + (i+1)][playerLocation.second] == '.' || map[playerLocation.first + (i+1)][playerLocation.second] == 'a')
                    {
                        boxes.push_back(*x);
                    }
                    i++;
                    x = find_if(boxLocation.begin(), boxLocation.end(), [this,i](pair<int,int> p){return (p.first == (playerLocation.first + i)) && p.second == playerLocation.second;});
                }
                if(map[playerLocation.first + (i)][playerLocation.second] == '.' || map[playerLocation.first + (i)][playerLocation.second] == 'a')
                {
                    for(auto y : boxes)
                    {
                        vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [y](pair<int,int> p){return p.first == y.first && p.second == y.second;});
                        x->first++;
                    }
                    playerLocation.first++;
                }
            }
            else
            {
                playerLocation.first++;
            }

        }
        direction = 'S';
    }
    if(k == sf::Keyboard::Left || k == sf::Keyboard::A)
    {
        if(map[playerLocation.first][playerLocation.second - 1] == '.' || map[playerLocation.first][playerLocation.second - 1] == 'a')
        {
            vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [this](pair<int,int> p){return p.first == playerLocation.first && p.second == playerLocation.second - 1;});
            if(x != boxLocation.end())
            {
                while(x != boxLocation.end())
                {
                    if(map[playerLocation.first][playerLocation.second - (i+1)] == '.' || map[playerLocation.first][playerLocation.second - (i+1)] == 'a')
                    {
                        boxes.push_back(*x);
                    }
                    i++;
                    x = find_if(boxLocation.begin(), boxLocation.end(), [this,i](pair<int,int> p){return p.first == playerLocation.first  && p.
   second == (playerLocation.second - i);});
                }
                if(map[playerLocation.first][playerLocation.second- (i)] == '.' || map[playerLocation.first][playerLocation.second - (i)] == 'a')
                {
                    for(auto y : boxes)
                    {
                        vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [y](pair<int,int> p){return p.first == y.first && p.second == y.second;});
                        x->second--;
                    }
                    playerLocation.second--;
                }
            }
            else
            {
                playerLocation.second--;
            }
        }
        direction = 'W';
    }
    if(k == sf::Keyboard::Right || k == sf::Keyboard::D)
    {
        if(map[playerLocation.first][playerLocation.second + 1] == '.' || map[playerLocation.first][playerLocation.second + 1] == 'a')
        {
            vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [this](pair<int,int> p){return p.first == playerLocation.first && (p.second == (playerLocation.second + 1));});
            if(x != boxLocation.end())
            {
                while(x != boxLocation.end())
                {
                    if(map[playerLocation.first][playerLocation.second + (i+1)] == '.' || map[playerLocation.first][playerLocation.second + (i+1)] == 'a')
                    {
                        boxes.push_back(*x);
                    }
                    i++;
                    x = find_if(boxLocation.begin(), boxLocation.end(), [this,i](pair<int,int> p){return (p.first == playerLocation.first) && (p.second == (playerLocation.second + i));});
                }
                if(map[playerLocation.first][playerLocation.second + i] == '.' || map[playerLocation.first][playerLocation.second + i] == 'a')
                {
                    for(auto y : boxes)
                    {
                        vector<pair<int,int>>::iterator x = find_if(boxLocation.begin(), boxLocation.end(), [y](pair<int,int> p){return (p.first == y.first) && (p.second == y.second);});
                        x->second++;
                    }
                    playerLocation.second++;
                }
            }
            else
```

```cpp
            {
                playerLocation.second++;
            }
        }
        direction = 'E';
    }
    isWon();
}
Sokaban::~Sokaban(){
    for(int i = 0; i < height; i++){
        delete[] map[i];
    }
    delete[] map;
}
int Sokaban::getHeight() const {return height;}
int Sokaban::getWidth() const {return width;}
void Sokaban::setHeight(int h) {height = h;}
void Sokaban::setWidth(int w) {width = w;}
char Sokaban::operator[](pair<int,int> p) const {return map[p.first][p.
    second];}

istream& operator>>(istream& in, Sokaban& s){
    in >> s.height >> s.width;
    s.won = false;
    s.map = new char*[s.height];
    s.boxLocation.clear();
    s.storageLocation.clear();
    for(int i = 0; i < s.height; i++){
        s.map[i] = new char[s.width];
        for(int j = 0; j < s.width; j++){
            char x;
            in >> x;
            if(x == '@'){
                s.playerLocation.first = i;
                s.playerLocation.second = j;
                s.map[i][j] = '.';
                s.direction = '@';
            }
            if(x == 'A'){
                s.boxLocation.push_back(make_pair(i,j));}
                s.map[i][j] = '.';
            if(x == 'a'){
                s.storageLocation.push_back(make_pair(i,j));
                s.map[i][j] = x;
            }
            if(x == '#'){
                s.map[i][j] = x;
            }
            if(x == '.'){
                s.map[i][j] = x;
            }

        }

    }
    return in;
}

void Sokaban::draw(sf::RenderTarget& target, sf::RenderStates states) const
```

```
{
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            sf::Sprite sprite;
            switch(map[i][j]){
                case '#':
                    sprite.setTexture(Wall);
                    break;
                case '.':
                    sprite.setTexture(Empty);
                    break;
                case 'a':
                    sprite.setTexture(Storage);
                    break;
            }
            sprite.setPosition(j * 64, i * 64);
            target.draw(sprite);
        }
    }

    for(auto x: boxLocation){
        sf::Sprite sprite;
        auto y = find_if(storageLocation.begin(),  storageLocation.end(), [x
   ](pair<int,int> p){return (p.first == x.first) && (p.second == x.second)
   ;});
        if(y != storageLocation.end())
            sprite.setTexture(GBox);
        else
            sprite.setTexture(Box);
        sprite.setPosition(x.second * 64, x.first * 64);
        target.draw(sprite);

    }
    sf::Sprite player;
    switch(direction){
        case '@':
            player.setTexture(Player[0]);
            break;
        case 'N':
            player.setTexture(Player[0]);
            break;
        case 'S':
            player.setTexture(Player[1]);
            break;
        case 'E':
            player.setTexture(Player[2]);
            break;
        case 'W':
            player.setTexture(Player[3]);
            break;
    }
    player.setPosition(playerLocation.second * 64, playerLocation.first *
   64);
    target.draw(player);

    if(won)
    {
        sf::Sprite sprite;
        sprite.setTexture(Win);
        sprite.scale(64.0/Win.getSize().x, 64.0/Win.getSize().y);
        sprite.setPosition(playerLocation.second * 64, playerLocation.first * 64);
        target.draw(sprite);
    }
}
```

5.5 Output:

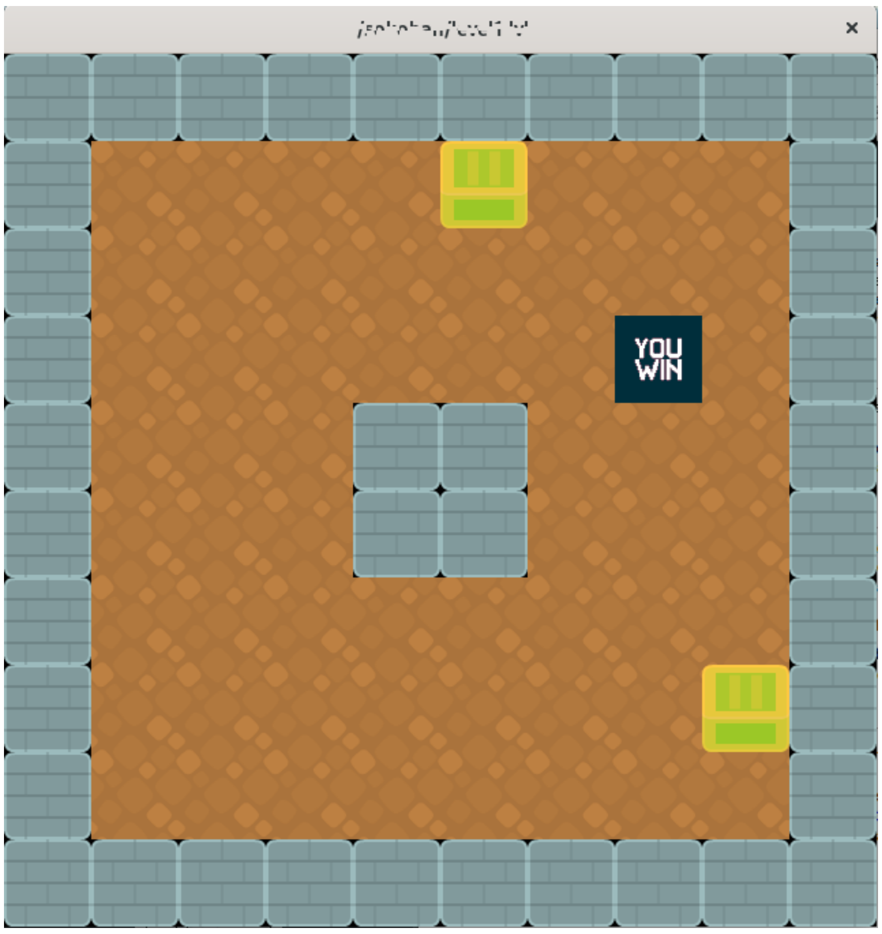


Figure 6: Player wins game

6 PS3: Pythagoras Tree

6.1 Discussion:

The assignment's objective was the create a primary function that recursively drew a fractal like image. Fractals are dictated by their self-similarity, and in this drawing, at the top of each square the triangle is drawn. Whereby each triangle can make two more squares. The recursive function has a base case, which allows for setting a finite limit on the number of calls. In our assignment, we were able to first create a function that worked for the 45 degree triangle, then afterwards we implemented the function for all angles between 1-89. We then added a coloring scheme that spanned the amount of function calls.

The extra credit feature we implemented was slightly changing the function so that it could operate with any starting angle, not just 45 degrees. The was done by fixing some aspects of the first algorithm so that they were more abstract and not specifically catered towards 45 degrees. One feature I implemented was forming a mapping from the cartesian coordinates into the video window coordinates. The dx and dy were conceptually born from the understanding of cosine and sin in the cartesian plane, so the mapping made the entire problem much easier. Lastly the coloring was implemented with two different methods, one of logarithmic and one linear.

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

A key data structure in the assignment was a vector of rectangles. This was essential because it allowed us to create rectangles for each function call and store them all in a central data structure. Afterwards the draw feature could very easily display all the rectangles that were pushed into the vector.

6.3 What I learned :

I learned the difficulty in using parametric functions that are catered to euclidean space which treats the 2 dimensional plane of real numbers. In the SFML library the top left is 0 and everything else is positive. As mentioned prior, this was overcome with a mapping into euclidean space instead of trying to reconfigure the trigonometric functions for the strange orientation.

6.4 Codebase

Makefile

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

```

CXX = g++
CXXFLAGS = -std=c++11 -Wall -Wextra -Wpedantic
LDFLAGS = -lsfml-graphics -lsfml-window -lsfml-system

all: ptree

ptree: main.o PTree.o
	$(CXX) $(CXXFLAGS) -o $@ $^ $(LDFLAGS)

main.o: main.cpp PTree.h
	$(CXX) $(CXXFLAGS) -c -o $@ $<

PTree.o: PTree.cpp PTree.h
	$(CXX) $(CXXFLAGS) -c -o $@ $<

lint:
	cpplint.py --filter=-runtime/references,-build/c++11 --root=. *
clean:
	rm -f *.o ptree
```

```

.PHONY: all clean
```

main.cpp

```cpp
// Copyright 2023 Hunter Hasenfus and Daniel Olen
#include <iostream>
#include <SFML/Graphics.hpp>
#include "PTree.h"

int main(int argc, char* argv[]) {
    if (argc < 4) {
        std::cerr << "Usage: " << argv[0] << " L N Theta" << std::endl;
        return 1;
    }

    // Parse command-line arguments for the
    // base square size (L) and recursion depth (N)
    double L = std::stod(argv[1]);
    int N = std::stoi(argv[2]);
    double startingAngle = std::stoi(argv[3]);

    // Create a window with dimensions 6L x 4L
    sf::RenderWindow window1(sf::VideoMode(6 * L, 4 * L), "Pythagoras Tree")
    ;
    window1.setFramerateLimit(60);

    PTree tree1(L, N, 45);
    PTree tree2(L, N, startingAngle);

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

        window1.clear();
        window1.draw(tree1);
        window1.display();
    }
    sf::RenderWindow window2(sf::VideoMode(10 * L, 4 * L), "Pythagoras Tree"
    );
    window2.setFramerateLimit(60);

    while (window2.isOpen()) {
        sf::Event event;
        while (window2.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window2.close();
            }
        }

        window2.clear();
        window2.draw(tree2);
        window2.display();
    }

    return 0;
}
```

PTree.h

```cpp
// Copyright 2023 Hunter Hasenfus and Daniel Olen
#ifndef PTREE_H_
#define PTREE_H_

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


class PTree : public sf::Drawable {
 public:
    // Constructor with two arguments, takes in
    // the length of a rectangle and the number of iterations
    PTree(double L, int N, double startingAngle);

 private:
    // Override the draw function from
    // sf::Drawable to draw the rectangles in the window
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
   const;

    // Recursive function to generate the Pythagoras Tree
    void pTree(double L, int N, double x, double y,
     double angle, double startingAngle);
    // Vector to store the rectangles that make up the tree
    std::vector<sf::RectangleShape> m_rectangles;

    double logColor;
    double linColor;
};

#endif  // PTREE_H_
```

PTree.cpp

```cpp
// Copyright 2023 Hunter Hasenfus and Daniel Olen
#include "PTree.h"
#include <cmath>



// Constant value to convert degrees to radians
const double DEG_TO_RAD = M_PI / 180.0;

PTree::PTree(double L, int N, double startingAngle) {
    logColor = pow(255.0 , (1.0/N));
    linColor = 255.0/N;
    pTree(L, N, 3 * L - L/2, 3 * L, 0, startingAngle);
}

void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    // Iterates through the vector of rectangles
    // and draws each rectangle on the target
    for (const auto& rect : m_rectangles) {
        target.draw(rect, states);
    }
}

// Recursive function that generates
// the Pythagoras Tree
void PTree::pTree(double L, int N, double x,
double y, double angle, double startingAngle) {
    // Base case: if N is 0, return from the function
    if (N == 0) {
        return;
    }
    // Create a rectangle with size L x L
    sf::RectangleShape rect(sf::Vector2f(L, L));
    rect.setPosition(x, y);
    rect.setRotation(-1 * angle);
    rect.setFillColor(sf::Color(255 - pow(logColor,
    N), pow(logColor, N), linColor * N));
    rect.setOutlineColor(sf::Color::Black);
    rect.setOutlineThickness(1.0f);

    // Add the rectangle to the vector of rectangles
    m_rectangles.push_back(rect);

    // Convert the angle from degrees to radians
    double angleRad = startingAngle * DEG_TO_RAD;

    // Calculate the new length of the rectangles for the next iteration
    double newL1 = L * sin(angleRad);
    double newL2 = L * cos(angleRad);

    // Convert the angle for the second rectangle from degrees to radians
    double angle2Rad = (angle + startingAngle) * DEG_TO_RAD;

    // Calculate the new x and y coordinates for
    // the first rectangle in the next iteration
    double dx = (newL1 + newL2) * cos(angle2Rad);
    double dy = (newL1 + newL2) * sin(angle2Rad);

    double newX1, newY1, newX2, newY2;
    if (dx >= 0 && dy >= 0) {
        newX1 = x + abs(dx);
        newY1 = y - abs(dy);
    } else { if (dx <= 0 && dy <= 0) {
        newX1 = x - abs(dx);
        newY1 = y + abs(dy);
    } else { if (dx <= 0 && dy >= 0) {
        newX1 = x - abs(dx);
        newY1 = y - abs(dy);
    } else { if (dx >= 0 && dy <= 0) {
        newX1 = x + abs(dx);
        newY1 = y + abs(dy);
    }}}}

    angle2Rad = (angle + startingAngle + 90) * DEG_TO_RAD;

    // Calculate the new x and y coordinates
    // for the second rectangle in the next iteration
    dx = newL2 * cos(angle2Rad);
    dy = newL2 * sin(angle2Rad);
    if (dx >= 0 && dy >= 0) {
        newX2 = x + abs(dx);
        newY2 = y - abs(dy);
    } else {if (dx <= 0 && dy <= 0) {
        newX2 = x - abs(dx);
        newY2 = y + abs(dy);
    } else {if (dx <= 0 && dy >= 0) {
        newX2 = x - abs(dx);
        newY2 = y - abs(dy);
    } else {if (dx >= 0 && dy <= 0) {
        newX2 = x + abs(dx);
        newY2 = y + abs(dy);
    }}}}

    // Recursively call the pTree function for
    // the first rectangle with the new parameters
    pTree(newL1, N - 1, newX1, newY1,
    angle - (90 - startingAngle), startingAngle);
    pTree(newL2, N - 1, newX2, newY2, angle + startingAngle, startingAngle);
}
```

6.5 Output:

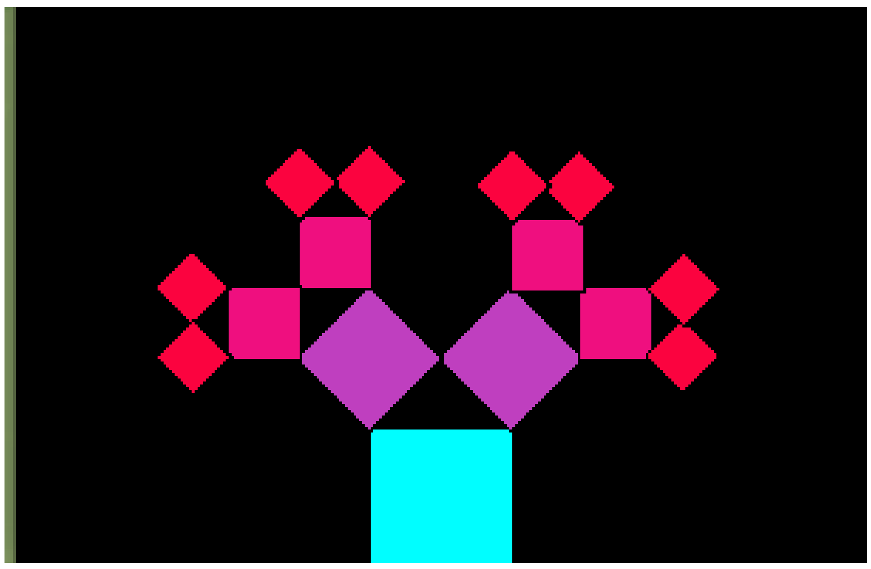


Figure 7: PS3 Output Window

7 PS4a: Checkers Visual mechanics

7.1 Discussion:

The assignment is tasked with creating a game using the SFML library that mimics the game of checkers. The game of checkers is inherently designed in the form of an array, which makes the functionality of it very easy to design and implement. By using indices as locations the area of clicking as well the entire drawing process can be circumvented into a simple fashion.

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

The key data structures that I implemented was a map of pairs. This allowed me to have all the pieces designated to a given user in a single data structure, and made it very easy to decipher what locations were valid when a given user was up. It also allowed me to figure out where a piece was then easily decipher its type.

I did not use smart pointers. I created an array of the game board, and then 2 map containers for the pieces relative to each user. For selecting the piece based on the button pressed and mouse location, I made a variable that was a member of that class. This allowed a dynamic nature to the 'selected variable'.

7.3 What I learned :

I learned how to decipher spatial locations within the SFML window. This was an interesting process and one that I can completely understand can take up a great deal of time for game creators.

7.4 Codebase

Makefile:

This Makefile has linting included.

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
DEP = Checkers.h
OBJS = main.o Checkers.o

all: Checkers

Checkers: $(OBJS)
	$(CC) -o Checkers $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
	$(CC) -c $(CFLAGS) $< -o $@ $(LFLAGS)

lint:
	cpplint.py --filter=-runtime/references,-build/c++11 --root=. *


cleanall : cleanobj
	rm Checkers

cleanobj :
	rm *.o
```

main.cpp:

```
// Copyright 2023 Hunter Hasenfus
#include <cstdlib>
#include <ctime>
#include <iostream>
```

```cpp
#include <fstream>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "Checkers.h"

int main(int arg, char* argc[]) {
    bool replay = false;
    do {
        Checkers s;

        sf::RenderWindow window(sf::VideoMode(8 * 64, 8 * 64), argc[0]);
        while (window.isOpen()) {
            sf::Event event;
            while (window.pollEvent(event)) {
                if (event.type == sf::Event::Closed)
                    window.close();
                if (event.type == sf::Event::MouseButtonPressed
                 && sf::Mouse::Left == event.mouseButton.button) {
                    // sf::Vector2i localPosition =
                    //  sf::Mouse::getPosition(window);
                    // s.recognizePiece(localPosition.x, localPosition.y);
                    s.recognizePiece(event.mouseButton.x, event.mouseButton.
    y);
                    // cout << event.mouseButton.x << "," <<
                    // event.mouseButton.y << endl;
                }}
            window.clear();
            window.draw(s);
            window.display();
        }
        std::cout << "Would you like to play again? (y/n)";
        std::cin >> replay;
        if (replay == 'y')
            replay = true;
        else
            replay = false;
    } while (replay);

    return 0;
}
```

Checkers.h:

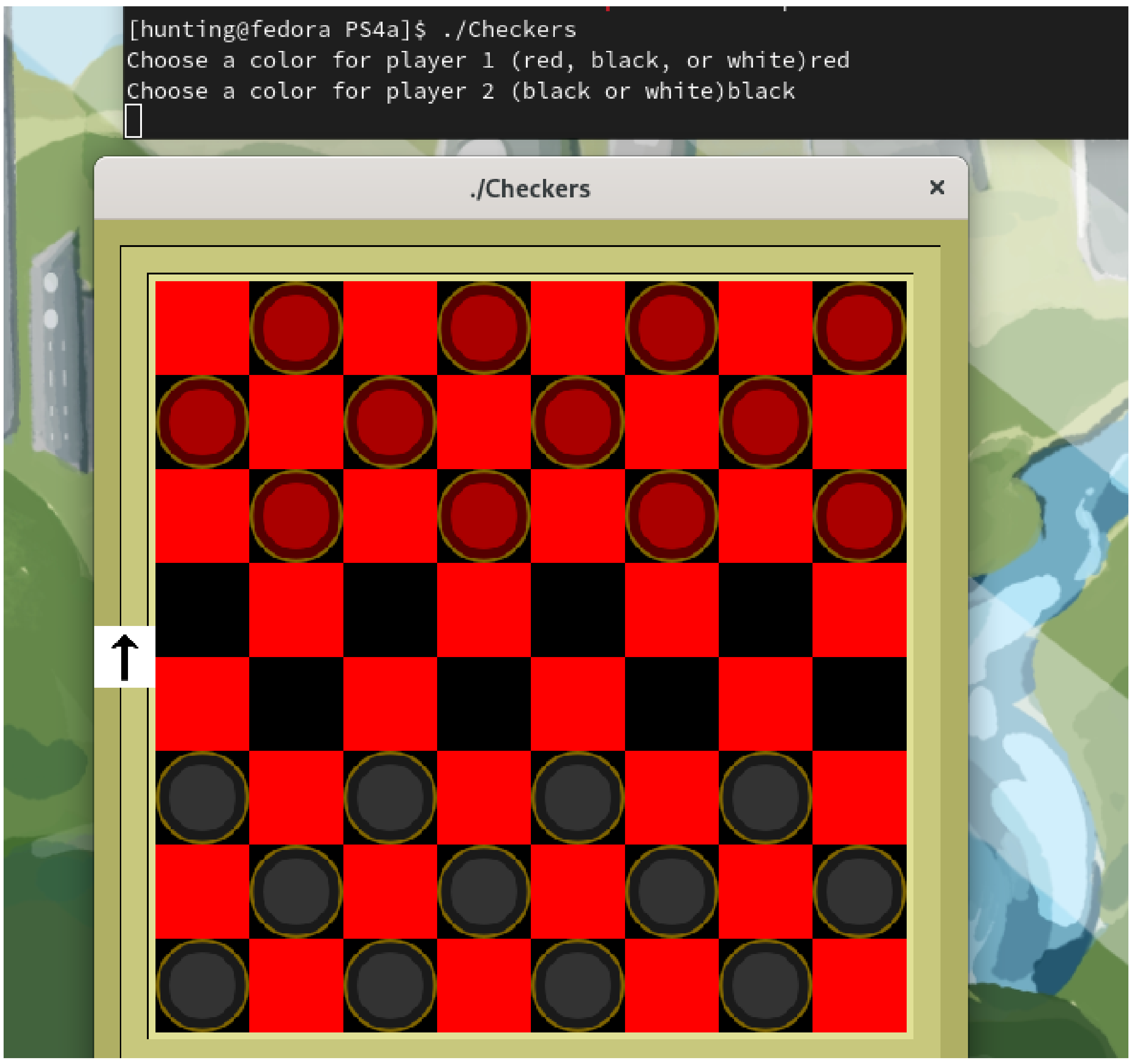
```cpp
// Copyright 2023 Hunter Hasenfus
#ifndef CHECKERS_H_
#define CHECKERS_H_
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <utility>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

class Checkers : public sf::Drawable {
 public:
        Checkers();
        ~Checkers();
        void recognizePiece(int x, int y);
 private:

        virtual void draw(sf::RenderTarget& target,
        sf::RenderStates states) const;
        char **board;

        bool turn;

        std::vector<std::string> colors = {"red", "black", "white"};
        std::string p1color;
        std::string p2color;

        sf::Texture blackPawn;
        sf::Texture redPawn;
        sf::Texture blackKing;
        sf::Texture redKing;
        sf::Texture whitePawn;
        sf::Texture whiteKing;
        sf::Texture blackBoard;
        sf::Texture redBoard;

        sf::Texture arrow;

        sf::Texture p1pawn;
        sf::Texture p1king;
        sf::Texture p2pawn;
        sf::Texture p2king;

        std::pair<std::pair<int, int>, std::pair<char, char>> selectedPiece;

        std::vector<std::pair<int, int>> p1pawns;
        std::vector<std::pair<int, int>> p1kings;

        std::vector<std::pair<int, int>> p2pawns;
        std::vector<std::pair<int, int>> p2kings;

        std::map<std::pair<int, int>, char> p1pieces;
        std::map<std::pair<int, int>, char> p2pieces;
};

#endif  // CHECKERS_H_
```

Checkers.cpp:

```
// Copyright 2023 Hunter Hasenfus
#include "Checkers.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <utility>
#include <string>
#include <algorithm>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

```

```cpp

void Checkers::recognizePiece(int x, int y) {
    double n, m;
    if (x > 36 && x < 36 + 55 * 8 && y > 36 && y < 36 + 55 * 8) {
        m = (x-36) / 55;
        n = (y-36) / 55;
        if (turn) {
            auto Piece = find_if(p1pieces.begin(), p1pieces.end(),
             [n, m](std::pair<std::pair<int, int>
            , char> x){return x.first.first == n && x.first.second == m;});
            if (Piece != p1pieces.end())
                selectedPiece = make_pair(Piece->first,
                 std::make_pair(Piece->second, '1'));
            else
                selectedPiece = std::make_pair(std::make_pair(-1, -1),
                 std::make_pair('0', '0'));
        } else {
            auto Piece = find_if(p2pieces.begin(), p2pieces.end(),
             [n, m](std::pair<std::pair<int, int>
            , char> x){return x.first.first == n && x.first.second == m;});
            if (Piece != p2pieces.end())
                selectedPiece = make_pair(Piece->first,
                 std::make_pair(Piece->second, '2'));
            else
                selectedPiece = std::make_pair(std::make_pair(-1, -1),
                 std::make_pair('0', '0'));
        }} else {
        selectedPiece = std::make_pair(std::make_pair(-1, -1),
         std::make_pair('0', '0'));
    }
    // cout << n << "," << m << endl;
    // cout << selectedPiece.first.first << "," <<
    // selectedPiece.first.second << " "
    //  << selectedPiece.second.first << " " <<
    // selectedPiece.second.second << endl;
}

Checkers::~Checkers() {
    for (int i = 0; i < 8; i++) {
        delete[] board[i];
    }
    delete[] board;
}


Checkers::Checkers() {
    blackPawn.loadFromFile("checkers/blackpawn.png");
    redPawn.loadFromFile("checkers/redpawn.png");
    blackKing.loadFromFile("checkers/blackking.png");
    redKing.loadFromFile("checkers/redking.png");
    whitePawn.loadFromFile("checkers/whitepawn.png");
    whiteKing.loadFromFile("checkers/whiteking.png");
    arrow.loadFromFile("checkers/download.png");


    turn = true;
    selectedPiece = std::make_pair(std::make_pair(-1, -1),
     std::make_pair('0', '0'));

    std::vector<std::string>::iterator y;
    std::cout << "Choose a color for player 1 (red, black, or white)";
    do {
        std::cin >> p1color;
        y = find_if(colors.begin(), colors.end(), [this](std::string x)
        {return x == this->p1color;});
    } while (y == colors.end());

    if (p1color == "black") {
        p1pawn = blackPawn;
        p1king = blackKing;
    } else { if (p1color == "red") {
        p1pawn = redPawn;
        p1king = redKing;
    } else {if (p1color == "white") {
        p1pawn = whitePawn;
        p1king = whiteKing;
    }}}
    colors.erase(y);

    std::cout << "Choose a color for player 2 (" << colors[0] << " or "
     << colors[1] << ")";
    do {
        std::cin >> p2color;
    } while (find_if(colors.begin(), colors.end(), [this](std::string x)
    {return x == this->p2color;}) == colors.end());

    if (p2color == "black") {
        p2pawn = blackPawn;
        p2king = blackKing;
    } else {if (p2color == "red") {
        p2pawn = redPawn;
        p2king = redKing;
    } else {if (p2color == "white") {
        p2pawn = whitePawn;
        p2king = whiteKing;
    }}}

    board = new char*[8];
    for (int i = 0; i < 8; i++) {
        board[i] = new char[8];
        for (int j = 0; j < 8; j ++) {
            if (i % 2) {
                if (j % 2) {
                    board[i][j] = 'r';
                } else {
                    board[i][j] = 'b';
                    if (i < 3) {
                        p1pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    } else if (i > 4) {
                        p2pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    }}} else {
                if (j % 2) {
                    board[i][j] = 'b';
                    if (i < 3) {
                        p1pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    } else {if (i > 4) {
                        p2pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    }}} else {
                    board[i][j] = 'r';
                }}}}
    // cout << "p1pawns:" << endl;
    // for(auto x: p1pieces) {
    //     std::cout << x.first.first << " " << x.first.second << " - ";
    // }
    // cout << endl << "p2pawns:" << endl;
    // for(auto x: p2pieces) {
    //     std::cout << x.first.first << " " << x.first.second << " - ";
    // }
}

void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
    for (int i = 0; i < 3; i ++) {
        sf::RectangleShape border(sf::Vector2f(8 * (64 - i * 4),
         8 * (64 - i * 4)));
        border.setPosition(i * 16, i * 16);
        border.setFillColor(sf::Color(175 + i * 25, 175 + i * 25,
         100 + i * 25));
        border.setOutlineColor(sf::Color::Black);
        border.setOutlineThickness(0.5f);
        target.draw(border);
    }
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            sf::RectangleShape rect(sf::Vector2f(55, 55));
            rect.setPosition(36 + i * 55, 36 + j * 55);

            rect.setOutlineColor(sf::Color::Black);
            rect.setOutlineThickness(0.1f);
            sf::Sprite sprite;
            switch (board[i][j]) {
                case 'b':
                    rect.setFillColor(sf::Color::Black);
                    break;
                case 'r':
                    rect.setFillColor(sf::Color::Red);
                    break;
            }
            target.draw(rect);
        }
    }


    for (auto x : p1pieces) {
        sf::Sprite sprite;
        if (x.second == 'p')
            sprite.setTexture(p1pawn);
        else
            sprite.setTexture(p1king);
        sprite.scale(55.0/p1pawn.getSize().x, 55.0/p1pawn.getSize().y);
        sprite.setPosition(36 + x.first.second * 55, 36 + x.first.first *
    55);
        target.draw(sprite);
    }

    for (auto x : p2pieces) {
        sf::Sprite sprite;
        if (x.second == 'p')
            sprite.setTexture(p2pawn);
        else
            sprite.setTexture(p2king);
        sprite.scale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y);
        sprite.setPosition(36 + x.first.second * 55, 36 + x.first.first *
55);
        target.draw(sprite);
    }

    sf::Sprite sprite;
    if (selectedPiece.second.second != '0') {
        if (selectedPiece.second.second == '1') {
            if (selectedPiece.second.first == 'p')
                sprite.setTexture(p1pawn);
            else
                sprite.setTexture(p1king);
        } else { if (selectedPiece.second.second == '2') {
            if (selectedPiece.second.first == 'p')
                sprite.setTexture(p2pawn);
            else
                sprite.setTexture(p2king);
        }}

        // sprite.setScale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y)
;
        // cout << "SCALE: " << sprite.getScale().x << ","
        // << sprite.getScale().y << endl;
        // sprite.setColor(sf::Color::Yellow);
        // cout << "COLOR: " <<sprite.getScale().x << ","
        // << sprite.getScale().y << endl;

        // cout << "texture size: " << p2pawn.getSize().x << ","
        // << p2pawn.getSize().y << endl;
        // cout << "intendend scale: " << 55.0/p2pawn.getSize().x
        //  << "," <<  55.0/p2pawn.getSize().y << endl;
        sprite.setColor(sf::Color(0, 255, 0));
        // cout << "COLOR: " <<sprite.getScale().x << "," <<
        // sprite.getScale().y << endl;
        sprite.setScale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y);
        // cout << "SCALE: " << sprite.getScale().x << "," <<
        // sprite.getScale().y << endl;

        // sprite.scale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y);
        // cout << sprite.getScale().x << "," << sprite.getScale().y << endl
;
        sprite.setPosition(36 + selectedPiece.first.second * 55,
        36 + selectedPiece.first.first * 55);
        target.draw(sprite);
    }

    if (turn) {
        sf::Sprite sprite;
        sprite.setTexture(arrow);
        sprite.setOrigin(arrow.getSize().x/2, arrow.getSize().y/2);
        sprite.scale(36.0/arrow.getSize().x, 36.0/arrow.getSize().y);
        sprite.setPosition(18, 4 * 64);
        sprite.setRotation(0);
        target.draw(sprite);
    } else {
        sf::Sprite sprite;
        sprite.setTexture(arrow);
        sprite.setOrigin(arrow.getSize().x/2, arrow.getSize().y/2);
        sprite.scale(36.0/arrow.getSize().x, 36.0/arrow.getSize().y);
        sprite.setPosition(18, 4 * 64);
        sprite.setRotation(180);
        target.draw(sprite);
    }

    // if(won)
    // {
    //     sf::Sprite sprite;
    //     sprite.setTexture(Win);
    //     sprite.scale(64.0/Win.getSize().x, 64.0/Win.getSize().y);
    //     sprite.setPosition(playerLocation.second * 64,
    //  playerLocation.first * 64);
    //     target.draw(sprite);
    // }
}
```

7.5 Output :



```
[hunting@fedora PS4a]$ ./Checkers
Choose a color for player 1 (red, black, or white)red
Choose a color for player 2 (black or white)black
```

Figure 8: PS4a Output in Terminal

8 PS4b: Checkers Game mechanics

8.1 Discussion:

The assignent is the task of applying functionality to the UI that was developed in the previous assignment. This assignment is where the movement of the players is conditioned as well as the crowning of pawns and the decision of victory.

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

The key algorithms I used were maps, this allowed myself the ability to track various pieces based on certain mappings. It made it very easy and transferable to coordinate functions based on the mappings of different different groups. The object oriented nature allowed myself to store various data structures that made it indispensable when coordinating all the functions together.

8.3 What I learned :

I created a few features and actually made and remade aspects as I went on. I noticed smoother and more versatile methods by which I could implement the game and noticed that traveling onward with legacy structures would have brought along handicaps that could have made the development much harder. The key points of this were abstracting away the nextMove functionality. This allowed myself the capacity to check if any peice had any next moves which aided in the implementation of both multijumps as well as endgame scenarios.

8.4 Codebase

Makefile:

This Makefile has linting included.

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
DEP = Checkers.h
OBJS = main.o Checkers.o

all: Checkers

Checkers: $(OBJS)
	$(CC) -o Checkers $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
	$(CC) -c $(CFLAGS) $< -o $@ $(LFLAGS)

lint:
	cpplint.py --filter=-runtime/references,-build/c++11 --root=. *


cleanall : cleanobj
	rm Checkers

cleanobj :
	rm *.o
```

main.cpp:

```
// Copyright 2023 Hunter Hasenfus
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
```

```cpp
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "Checkers.h"

int main(int arg, char* argc[]) {
    bool replay = false;
    do {
        Checkers s;

        sf::RenderWindow window(sf::VideoMode(8 * 64, 8 * 64), argc[0]);
        while (window.isOpen()) {
            sf::Event event;
            while (window.pollEvent(event)) {
                if (event.type == sf::Event::Closed)
                    window.close();
                if (event.type == sf::Event::MouseButtonPressed
                 && sf::Mouse::Left == event.mouseButton.button) {
                    if (s.pieceSelected())
                        s.checkMove(event.mouseButton.x, event.mouseButton.y
   );
                    s.recognizePiece(event.mouseButton.x, event.mouseButton.
   y);
                }}
            window.clear();
            window.draw(s);
            window.display();
        }
        std::cout << "Would you like to play again? (y/n)";
        std::cin >> replay;
        if (replay == 'y')
            replay = true;
        else
            replay = false;
    } while (replay);

    return 0;
}
```

Checkers.h:

```cpp
// Copyright 2023 Hunter Hasenfus
#ifndef CHECKERS_H_
#define CHECKERS_H_
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <vector>
#include <map>
#include <string>
#include <utility>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

class Checkers : public sf::Drawable {
 public:
     Checkers();
     ~Checkers();
     void pawn2King(std::map<std::pair<int, int>, char> &pieces,
      std::pair<int, int> pos);
     void recognizePiece(int x, int y);
     bool pieceSelected();
     void checkMove(int x, int y);
     void nextMoves(std::pair<std::pair<int, int>, std::pair<char,
      char>> &piece, std::map<std::pair<int, int>, char> &Moves);

 private:
     void movePiece(std::map<std::pair<int, int>, char> &pieces,
      std::pair<int, int> newPos);
     virtual void draw(sf::RenderTarget& target,
     sf::RenderStates states) const;
     char **board;

     bool turn;

     std::vector<std::string> colors = {"red", "black", "white"};
     std::string p1color;
     std::string p2color;

     sf::Texture blackPawn;
     sf::Texture redPawn;
     sf::Texture blackKing;
     sf::Texture redKing;
     sf::Texture whitePawn;
     sf::Texture whiteKing;
     sf::Texture blackBoard;
     sf::Texture redBoard;

     sf::Texture arrow;

     sf::Texture p1pawn;
     sf::Texture p1king;
     sf::Texture p2pawn;
     sf::Texture p2king;
     sf::Font loserFont;

     std::map<std::pair<int, int>, char> potMoves;

     std::pair<std::pair<int, int>, std::pair<char, char>> selectedPiece;

     //  std::vector<std::pair<int, int>> p1pawns;
     //  std::vector<std::pair<int, int>> p1kings;

     //  std::vector<std::pair<int, int>> p2pawns;
     //  std::vector<std::pair<int, int>> p2kings;

     std::map<std::pair<int, int>, char> p1pieces;
     std::map<std::pair<int, int>, char> p2pieces;

     bool p1Win;
     bool p2Win;
};

#endif  // CHECKERS_H_
```

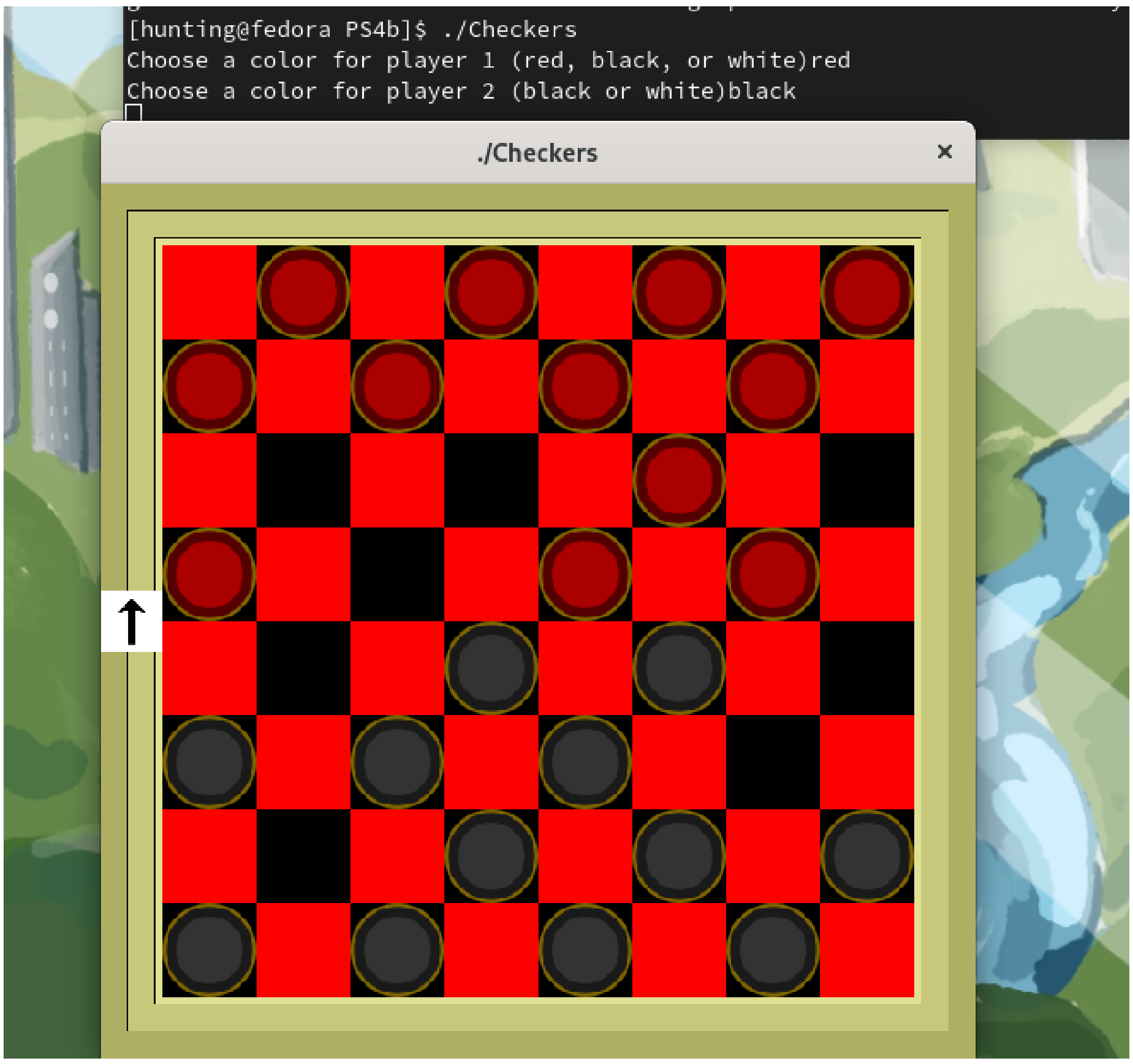
Checkers.cpp:

```
// Copyright 2023 Hunter Hasenfus
#include "Checkers.h"
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <utility>
#include <string>
#include <algorithm>
#include <vector>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

void Checkers::pawn2King(std::map<std::pair<int, int>, char> &pieces,
 std::pair<int, int> pos) {
    auto x = pieces.find(pos);
    pieces.erase(x);
    pieces.insert(std::make_pair(pos, 'k'));
}

void Checkers::movePiece(std::map<std::pair<int, int>, char> &pieces,
 std::pair<int, int> newPos) {
    pieces.erase(selectedPiece.first);
    pieces.insert(std::make_pair(std::make_pair(newPos.first, newPos.second)
   ,
    selectedPiece.second.first));
    selectedPiece = std::make_pair(std::make_pair(-1, -1),
    std::make_pair('0', '0'));
}

void Checkers::nextMoves(std::pair<std::pair<int, int>, std::pair<char,
 char>> &piece,  std::map<std::pair<int, int>, char> &Moves) {
    int m, n;
    Moves.clear();
    if (piece.second.second == '1') {
            if (piece.second.first == 'p') {
                // if click is in a valid close square
                for (int j = -1; j < 2; j += 2) {
                    n = piece.first.first + 1;
                    m = piece.first.second + j;
                    if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7) && (
   p2pieces.find(std::make_pair(n, m)) ==
                     p2pieces.end()) && (p1pieces.find(std::make_pair(n, m))
                     == p1pieces.end())) {
                        Moves.insert(std::make_pair(std::make_pair(n, m), 'r
   '));
                    }
                    auto x = p2pieces.find(std::make_pair(n, m));
                    if (x != p2pieces.end()) {
                        n = piece.first.first + 2;
                        m = piece.first.second + j * 2;
                        if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7) && (
   p2pieces.find(std::make_pair(n, m)) ==
                         p2pieces.end()) && (p1pieces.find(std::make_pair(n,
    m))
                         == p1pieces.end())) {
                            Moves.insert(std::make_pair(std::make_pair(n,
                             m), 'j'));
                }}}}
            if (piece.second.first == 'k') {
                for (int i = -1; i < 2; i += 2) {
                    for (int j = -1; j < 2; j += 2) {
                        n = piece.first.first + i;
                        m = piece.first.second + j;
                        if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7) && (
    p2pieces.find(std::make_pair(n, m)) ==
                         p2pieces.end()) && (p1pieces.find(std::make_pair(n,
     m))
                         == p1pieces.end())) {
                            Moves.insert(std::make_pair(std::make_pair(n,
                             m), 'r'));
                        }
                        auto x = p2pieces.find(std::make_pair(n, m));
                        if (x != p2pieces.end()) {
                            n = piece.first.first + i * 2;
                            m = piece.first.second + j * 2;
                            if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7) &&
    (p2pieces.find(std::make_pair(n, m)) ==
                             p2pieces.end()) && (p1pieces.find(
                                std::make_pair(n, m))
                             == p1pieces.end())) {
                                Moves.insert(std::make_pair(
                                    std::make_pair(n, m), 'j'));
     }}}}}} else { if (piece.second.second == '2') {
            if (piece.second.first == 'p') {
                    // if click is in a valid close square
                    for (int j = -1; j < 2; j += 2) {
                        n = piece.first.first - 1;
                        m = piece.first.second + j;
                        if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7) && (
    p2pieces.find(std::make_pair(n, m)) ==
                         p2pieces.end()) && (p1pieces.find(std::make_pair(n,
     m))
                         == p1pieces.end())) {
                            Moves.insert(std::make_pair(
                                std::make_pair(n, m), 'r'));
                        }
                        auto x = p1pieces.find(std::make_pair(n, m));
                        if (x != p1pieces.end()) {
                            n = piece.first.first -2;
                            m = piece.first.second + j * 2;
                            if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7) &&
    (p2pieces.find(std::make_pair(n, m))
                             == p2pieces.end()) && (p1pieces.find(
                                std::make_pair(n, m))
                             == p1pieces.end())) {
                                Moves.insert(std::make_pair(
                                    std::make_pair(n, m), 'j'));
                    }}}}
                if (piece.second.first == 'k') {
                    for (int i = -1; i < 2; i += 2) {
                        for (int j = -1; j < 2; j += 2) {
                            n = piece.first.first + i;
                            m = piece.first.second + j;
                            if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7) &&
    (p2pieces.find(std::make_pair(n, m)) ==
                            p2pieces.end()) && (p1pieces.find(
                                std::make_pair(n, m))
                             == p1pieces.end())) {
                                Moves.insert(std::make_pair(
                                    std::make_pair(n, m), 'r'));
                            }
                            std::cout << "here1";
                            auto x = p1pieces.find(std::make_pair(n, m));
                            if (x != p1pieces.end()) {
                                std::cout << "here2";
                                n = piece.first.first + i * 2;
                                m = piece.first.second + j * 2;
                                if ((n >= 0 && n <= 7) && (m >= 0 && m <= 7)
     && (p2pieces.find(std::make_pair(n, m))
                                 == p2pieces.end()) && (p1pieces.find(
                                    std::make_pair(n, m)) == p1pieces.end())
    ) {
                                        std::cout << "here3";
                                    Moves.insert(std::make_pair(
                                        std::make_pair(n, m), 'j'));
}}}}}}}}


void Checkers::checkMove(int x, int y) {
    double n, m;
    if (x > 36 && x < 36 + 55 * 8 && y > 36 && y < 36 + 55 * 8) {
        m = (x-36) / 55;
        n = (y-36) / 55;
        // if p1s turn
        for (auto moves : potMoves) {
            if (moves.first.first == n && moves.first.second == m) {
                if (moves.second == 'r') {
                    if (turn) {
                        movePiece(p1pieces, std::make_pair(n, m));
                    } else {
                        movePiece(p2pieces, std::make_pair(n, m));
                    }
                    turn = !turn;
                    break;
                }
                if (moves.second == 'j') {
                    if (turn) {
                        auto temp = std::make_pair(selectedPiece.first.first
                         - (selectedPiece.first.first - n)/2,
                         selectedPiece.first.second - (
                            selectedPiece.first.second - m)/2);
                        auto newPiece = std::make_pair(std::make_pair(n, m),
                         std::make_pair(selectedPiece.second.first, '1'));
                        movePiece(p1pieces, std::make_pair(n, m));
                        p2pieces.erase(temp);
                        selectedPiece = newPiece;
                        nextMoves(selectedPiece, potMoves);
                        auto temp2 = find_if(potMoves.begin(), potMoves.end
    (),
                          [](std::pair<std::pair<int, int>, char> piece) {
                            return piece.second == 'j';
                        });
                        if (temp2 == potMoves.end()) {
                            turn = !turn;
                            selectedPiece = std::make_pair(
                                std::make_pair(-1, -1), std::make_pair(
                                    '0', '0'));
                        }
                        if (n == 7) {
                            pawn2King(p1pieces, std::make_pair(n, m));
                    }} else {
                        auto temp = std::make_pair(selectedPiece.first.first
                         - (selectedPiece.first.first - n)/2,
                            selectedPiece.first.second - (
                                selectedPiece.first.second - m)/2);
                        auto newPiece = std::make_pair(std::make_pair(n, m),
                         std::make_pair(selectedPiece.second.first, '2'));
                        movePiece(p2pieces, std::make_pair(n, m));
                        p1pieces.erase(temp);
                        selectedPiece = newPiece;
                        nextMoves(selectedPiece, potMoves);
                        auto temp2 = find_if(potMoves.begin(),
                         potMoves.end(), [](std::pair<std::pair<int, int>,
                          char> piece) {
                            return piece.second == 'j';
                        });
                        if (temp2 == potMoves.end()) {
                            turn = !turn;
                            selectedPiece = std::make_pair(std::make_pair(
                                -1, -1), std::make_pair('0', '0'));
                        }
                        if (n == 0) {
                            pawn2King(p2pieces, std::make_pair(n, m));
                        }
                    }
                    break;
}}}
    if (p1pieces.empty()) {
        p1Win = true;
    }
    if (p2pieces.empty()) {
        p2Win = true;
    }
    std::map<std::pair<int, int>, char> noMoves;
    std::vector<bool> noMovesVector;
    for (auto x : p1pieces) {
        auto temp = std::make_pair(x.first, std::make_pair(x.second, '1'));
        nextMoves(temp, noMoves);
        if (noMoves.empty())
            noMovesVector.push_back(true);
    }
    if (noMovesVector.empty())
        p1Win = true;
    noMovesVector.clear();
    for (auto x : p2pieces) {
        auto temp = std::make_pair(x.first, std::make_pair(x.second, '2'));
        nextMoves(temp, noMoves);
        if (noMoves.empty())
            noMovesVector.push_back(true);
    }
    if (noMovesVector.empty())
        p2Win = true;
    noMovesVector.clear();
```

```
}}





bool Checkers::pieceSelected() {
    return selectedPiece.second.second != '0';
}
void Checkers::recognizePiece(int x, int y) {
    double n, m;
    if (x > 36 && x < 36 + 55 * 8 && y > 36 && y < 36 + 55 * 8) {
        m = (x-36) / 55;
        n = (y-36) / 55;
        if (turn) {
            auto Piece = find_if(p1pieces.begin(), p1pieces.end(),
             [n, m](std::pair<std::pair<int, int>
            , char> x){return x.first.first == n && x.first.second == m;});
            if (Piece != p1pieces.end())
                selectedPiece = make_pair(Piece->first,
                 std::make_pair(Piece->second, '1'));
            else
                selectedPiece = std::make_pair(std::make_pair(-1, -1),
                 std::make_pair('0', '0'));
        } else {
            auto Piece = find_if(p2pieces.begin(), p2pieces.end(),
             [n, m](std::pair<std::pair<int, int>
            , char> x){return x.first.first == n && x.first.second == m;});
            if (Piece != p2pieces.end())
                selectedPiece = make_pair(Piece->first,
                 std::make_pair(Piece->second, '2'));
            else
                selectedPiece = std::make_pair(std::make_pair(-1, -1),
                 std::make_pair('0', '0'));
        }} else {
        selectedPiece = std::make_pair(std::make_pair(-1, -1),
         std::make_pair('0', '0'));
    }

    potMoves.clear();
    nextMoves(selectedPiece, potMoves);
}

Checkers::~Checkers() {
    for (int i = 0; i < 8; i++) {
        delete[] board[i];
    }
    delete[] board;
}


Checkers::Checkers() {
    blackPawn.loadFromFile("checkers/blackpawn.png");
    redPawn.loadFromFile("checkers/redpawn.png");
    blackKing.loadFromFile("checkers/blackking.png");
    redKing.loadFromFile("checkers/redking.png");
    whitePawn.loadFromFile("checkers/whitepawn.png");
    whiteKing.loadFromFile("checkers/whiteking.png");
    arrow.loadFromFile("checkers/download.png");
    loserFont.loadFromFile("checkers/UbuntuMono-R.ttf");

    p1Win = false;
    p2Win = false;
    turn = true;
    selectedPiece = std::make_pair(std::make_pair(-1, -1),
     std::make_pair('0', '0'));

    std::vector<std::string>::iterator y;
    std::cout << "Choose a color for player 1 (red, black, or white)";
    do {
        std::cin >> p1color;
        y = find_if(colors.begin(), colors.end(), [this](std::string x)
        {return x == this->p1color;});
    } while (y == colors.end());

    if (p1color == "black") {
        p1pawn = blackPawn;
        p1king = blackKing;
    } else { if (p1color == "red") {
        p1pawn = redPawn;
        p1king = redKing;
    } else {if (p1color == "white") {
        p1pawn = whitePawn;
        p1king = whiteKing;
    }}}
    colors.erase(y);

    std::cout << "Choose a color for player 2 (" << colors[0] << " or "
     << colors[1] << ")";
    do {
        std::cin >> p2color;
    } while (find_if(colors.begin(), colors.end(), [this](std::string x)
    {return x == this->p2color;}) == colors.end());

    if (p2color == "black") {
        p2pawn = blackPawn;
        p2king = blackKing;
    } else {if (p2color == "red") {
        p2pawn = redPawn;
        p2king = redKing;
    } else {if (p2color == "white") {
        p2pawn = whitePawn;
        p2king = whiteKing;
    }}}

    board = new char*[8];
    for (int i = 0; i < 8; i++) {
        board[i] = new char[8];
        for (int j = 0; j < 8; j ++) {
            if (i % 2) {
                if (j % 2) {
                    board[i][j] = 'r';
                } else {
                    board[i][j] = 'b';
                    if (i < 3) {
                        p1pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    } else if (i > 4) {
                        p2pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    }}} else {
                if (j % 2) {
                    board[i][j] = 'b';
                    if (i < 3) {
                        p1pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    } else {if (i > 4) {
                        p2pieces.insert(std::make_pair(
                            std::make_pair(i, j), 'p'));
                    }}} else {
                    board[i][j] = 'r';
}}}}}

void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
    for (int i = 0; i < 3; i ++) {
        sf::RectangleShape border(sf::Vector2f(8 * (64 - i * 4),
         8 * (64 - i * 4)));
        border.setPosition(i * 16, i * 16);
        border.setFillColor(sf::Color(175 + i * 25, 175 + i * 25,
         100 + i * 25));
        border.setOutlineColor(sf::Color::Black);
        border.setOutlineThickness(0.5f);
        target.draw(border);
    }
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            sf::RectangleShape rect(sf::Vector2f(55, 55));
            rect.setPosition(36 + i * 55, 36 + j * 55);

            rect.setOutlineColor(sf::Color::Black);
            rect.setOutlineThickness(0.1f);
            sf::Sprite sprite;
            switch (board[i][j]) {
                case 'b':
                    rect.setFillColor(sf::Color::Black);
                    break;
                case 'r':
                    rect.setFillColor(sf::Color::Red);
                    break;
            }
            target.draw(rect);
        }
    }


    for (auto x : p1pieces) {
        sf::Sprite sprite;
        if (x.second == 'p')
            sprite.setTexture(p1pawn);
        else
            sprite.setTexture(p1king);
        sprite.scale(55.0/p1pawn.getSize().x, 55.0/p1pawn.getSize().y);
        sprite.setPosition(36 + x.first.second * 55, 36 + x.first.first *
    55);
        target.draw(sprite);
    }

    for (auto x : p2pieces) {
        sf::Sprite sprite;
        if (x.second == 'p')
            sprite.setTexture(p2pawn);
        else
            sprite.setTexture(p2king);
        sprite.scale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y);
        sprite.setPosition(36 + x.first.second * 55, 36 + x.first.first *
   55);
        target.draw(sprite);
    }

    sf::Sprite sprite;
    if (selectedPiece.second.second != '0') {
        if (selectedPiece.second.second == '1') {
            if (selectedPiece.second.first == 'p')
                sprite.setTexture(p1pawn);
            else
                sprite.setTexture(p1king);
        } else { if (selectedPiece.second.second == '2') {
            if (selectedPiece.second.first == 'p')
                sprite.setTexture(p2pawn);
            else
                sprite.setTexture(p2king);
        }}



        // sprite.setScale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y)
   ;
        // cout << "SCALE: " << sprite.getScale().x << ","
        // << sprite.getScale().y << endl;
        // sprite.setColor(sf::Color::Yellow);
        // cout << "COLOR: " <<sprite.getScale().x << ","
        // << sprite.getScale().y << endl;

        // cout << "texture size: " << p2pawn.getSize().x << ","
        // << p2pawn.getSize().y << endl;
        // cout << "intendend scale: " << 55.0/p2pawn.getSize().x
        //  << "," <<  55.0/p2pawn.getSize().y << endl;
        sprite.setColor(sf::Color(0, 255, 0));
        // cout << "COLOR: " <<sprite.getScale().x << "," <<
        // sprite.getScale().y << endl;
        sprite.setScale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y);
        // cout << "SCALE: " << sprite.getScale().x << "," <<
        // sprite.getScale().y << endl;

        // sprite.scale(55.0/p2pawn.getSize().x, 55.0/p2pawn.getSize().y);
        // cout << sprite.getScale().x << "," << sprite.getScale().y << endl
   ;
        sprite.setPosition(36 + selectedPiece.first.second * 55,
        36 + selectedPiece.first.first * 55);
        target.draw(sprite);
    }

    if (turn) {
        sf::Sprite sprite;
        sprite.setTexture(arrow);
```

```cpp
        sprite.setOrigin(arrow.getSize().x/2, arrow.getSize().y/2);
        sprite.scale(36.0/arrow.getSize().x, 36.0/arrow.getSize().y);
        sprite.setPosition(18, 4 * 64);
        sprite.setRotation(0);
        target.draw(sprite);
    } else {
        sf::Sprite sprite;
        sprite.setTexture(arrow);
        sprite.setOrigin(arrow.getSize().x/2, arrow.getSize().y/2);
        sprite.scale(36.0/arrow.getSize().x, 36.0/arrow.getSize().y);
        sprite.setPosition(18, 4 * 64);
        sprite.setRotation(180);
        target.draw(sprite);
    }


    if (p1Win) {
        sf::Text text;
        text.setFont(loserFont);
        text.setPosition(0, 64 * 4);
        text.setColor(sf::Color::White);
        text.setStyle(sf::Text::Bold | sf::Text::Underlined);
        text.setCharacterSize(64);
        text.setString("Player 2 Wins!");
        target.draw(text);
    }
    if (p2Win) {
        sf::Text text;
        text.setFont(loserFont);
        text.setPosition(0, 64 * 4);
        text.setColor(sf::Color::White);
        text.setStyle(sf::Text::Bold | sf::Text::Underlined);
        text.setCharacterSize(64);
        text.setString("Player 1 Wins!");
        target.draw(text);
    }
}
```

8.5 Output :



```
[hunting@fedora PS4b]$ ./Checkers
Choose a color for player 1 (red, black, or white)red
Choose a color for player 2 (black or white)black
```

Figure 9: PS4b Output in Terminal

9 PS5: DNA Sequence Alignment

9.1 Discussion:

The main objective in this assignment was to construct a systematic way of identifying differences in strings and calculating the string that minimizes distance between the two strings. For the runtime and space complexity scale pretty drastically depending on the implementation of this program. It is very important to consider many different methods and how each influences the program.

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

The dynamic programming approach is used to calculate the edit distance between two strings. The choice of this approach is based on its efficiency and relative ease of implementation. The method stores intermediate results in a two-dimensional vector, thus avoiding redundant computations This approach is more efficient than the recursive method without memoization, which can have exponential time complexity. Although recursive with memoization and Hirschberg's algorithm can also be efficient, dynamic programming is often simpler to implement and understand. The main disadvantage of dynamic programming is that it can consume more memory than other methods, particularly when working with very long strings.

9.3 What I learned :

I learned how to calculate the exponential growth and thus the runtime of the program. I did so by both utilizing the time command in C and also using valgrind to identify the amount of heap space allocated during runtime.

9.4 Codebase

Makefile :

This Makefile contains the lint.

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lsfml-graphics -lsfml-window -lsfml-system
DEP = EDistance.h
OBJS = main.o EDistance.o

all: EDistance

EDistance: $(OBJS)
    $(CC) -g -o EDistance $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
    $(CC) -g -c $(CFLAGS) $< -o $@ $(LFLAGS)

lint:
    cpplint.py --filter=-runtime/references,-build/c++11 --root=. *


cleanall : cleanobj
    rm EDistance

cleanobj :
    rm *.o
```

main.cpp :

```
#include <iostream>
#include <fstream>
```

```cpp
#include <SFML/System.hpp>
#include "EDistance.h"

// Copyright [2023] Daniel Olen & Hunter Hasenfus

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

    if (argc != 2) {
        std::cerr << "Usage: " << argv[0] << " <input_file>" << std::endl;
        return 1;
    }

    std::string input_file = argv[1];
    std::ifstream infile(input_file);

    if (!infile) {
        std::cerr << "Error: Unable to open input file" << std::endl;
        return 1;
    }

    std::string x, y;
    std::getline(infile, x);
    std::getline(infile, y);
    infile.close();

    EDistance eDistance(x, y);
    int distance = eDistance.optDistance();
    std::string alignment_str = eDistance.alignment();

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


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

    return 0;
}
```

EDistance.h :

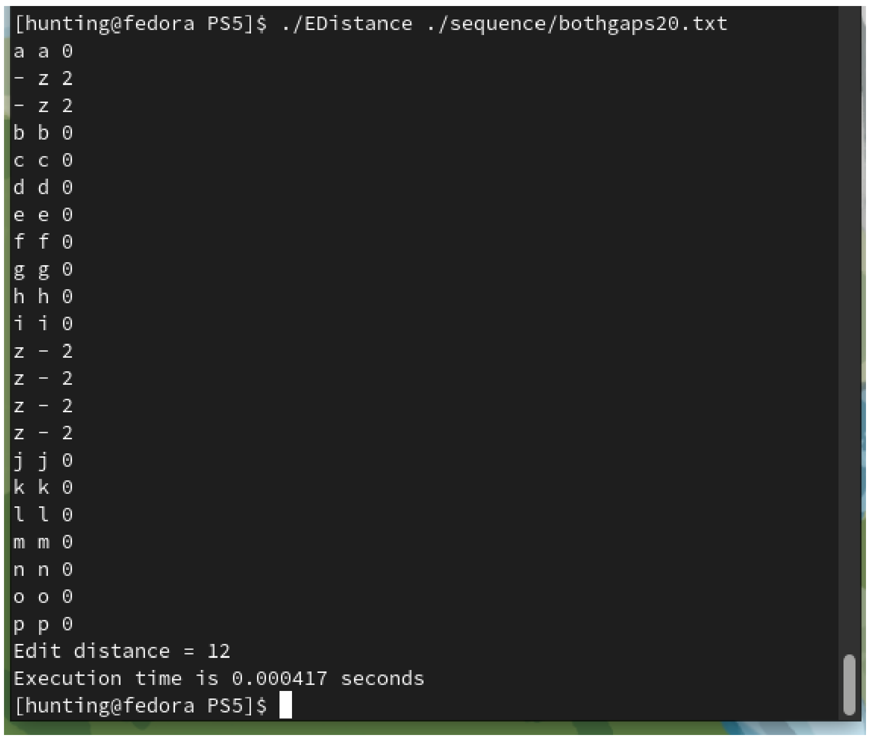
```cpp
#pragma once

#ifndef EDISTANCE_H
#define EDISTANCE_H

// Copyright [2023] Daniel Olen & Hunter Hasenfus

#include <string>
#include <vector>
#include <algorithm>

class EDistance {
 public:
    EDistance(const std::string &x, const std::string &y);

    static int penalty(char a, char b);
    static int min(int a, int b, int c);
    int optDistance();
    std::string alignment();
    int n;
 private:
    std::string x, y;
    std::vector<std::vector<int>> opt;
};

#endif
```

EDistance.cpp :

```cpp
#include "EDistance.h"
#include <algorithm>

// Copyright [2023] Daniel Olen & Hunter Hasenfus

EDistance::EDistance(const std::string &x, const std::string &y) :
x(x), y(y), opt(x.length() + 1, std::vector<int>(y.length() + 1)) {}

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

int EDistance::min(int a, int b, int c) {
    return std::min(std::min(a, b), c);
}

int EDistance::optDistance() {
    int n = x.length();
    int m = y.length();

    for (int i = 0; i <= n; ++i) {
        opt[i][0] = i * 2;
    }

    for (int j = 0; j <= m; ++j) {
        opt[0][j] = j * 2;
    }

    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            int match = opt[i - 1][j - 1] + penalty(x[i - 1], y[j - 1]);
            int insert = opt[i - 1][j] + 2;
            int delete_ = opt[i][j - 1] + 2;
            opt[i][j] = min(match, insert, delete_);
        }
    }

    return opt[n][m];
}

std::string EDistance::alignment() {
    std::string result;
    // int n = 0, temp;
    int i = x.length(), j = y.length();

    while (i > 0 || j > 0) {
        if (i > 0 && opt[i][j] == opt[i - 1][j] + 2) {
            result = std::string(1, x[i - 1]) +
            " - " + std::to_string(2) + "\n" + result;
            // n+=2;
            --i;
        } else if (j > 0 && opt[i][j] == opt[i][j - 1] + 2) {
            result = "- " + std::string(1, y[j - 1]) +
            " " + std::to_string(2) + "\n" + result;
            // n+=2;
            --j;
        } else {
        //     temp = penalty(x[i - 1], y[j - 1]);
```

```
            result = std::string(1, x[i - 1]) + " " + std::string(1, y[j -
    1]) +
            " " + std::to_string(penalty(x[i - 1], y[j - 1])) + "\n" +
    result;
            // n+=temp;
            --i;
            --j;
        }
    }


    return result;
}
```

9.5 Output:



```
[hunting@fedora PS5]$ ./EDistance ./sequence/bothgaps20.txt
a a 0
- z 2
- z 2
b b 0
c c 0
d d 0
e e 0
f f 0
g g 0
h h 0
i i 0
z - 2
z - 2
z - 2
z - 2
j j 0
k k 0
l l 0
m m 0
n n 0
o o 0
p p 0
Edit distance = 12
Execution time is 0.000417 seconds
[hunting@fedora PS5]$
```

Figure 10: PS5 Output Window

10 PS6: Random Writer

10.1 Discussion:

The assignment delves into the topic of markov models and markov chains. A markov probability is one where the cumulative sum is 1.0, this is done by using frequency probabilities. In the assignment the frequencies of kgrams are found and then the frequencies of the subsequent letters are found. This enables the ability to create probability distributions for each kgram and then randomly generate streams of characters based off the training data sets acquired probabilities.

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

In this implementation I constructed such markov model using three seperate data types. The first was a simple mapping from strings to ints, this was used for the frequencies of the kgrams. I then made a seperate data structure of a mapping into a vector, this allowed me to create a vector for each kgram and denote probabiltites for the following characters based on the frequency in which they were found in the training data. I chose to implement this with a vector because the last data structure I implemented was a mapping from strings into discrete distributions. By nature, the discrete distribution requires an input that is returns unlabeled integers. By using the sparse 127 element vector I was able to very seemlessly return the integer equivalent of the asci letters and conver to characters then and there.

The mapping to discrete distributions is pretty interesting in that, it seemlessly takes the vector of character probabilites and converts it into a probaility distribution based on the discrete distribution. This allows the ability to pass a random generator into the probablity distribution and pass out a value. Provides a great deal of simplification for the latter half of the program. line 63-67 RandWriter.cpp

```cpp
for (auto it: kgramCharProb) {
        std::discrete_distribution<int> dist(it.second.begin(), it.second.
    end());
        kgramCharDist[it.first] = dist;
    }

```

10.3 What I learned :

I have learnt about the Markov model and how it works. Also learnt about the Hash Map. I used exceptions on all the points listed in the pdf. Mainly invalid argument exceptions that made sure the length of the kgrams were accurate. line 78-81 RandWriter.cpp

```cpp
if (kgram.length() != order) {
        throw std::invalid_argument("kgram is not the same length as order"
         + kgram);
    }

```

This was an exception to catch invalid kgrams by length == order. line 126-130 RandWriter.cpp

```cpp
if (n == 0) {
        throw std::invalid_argument("kgram does not exist: " + kgram
         + ", " + static_cast<char>(c));
    }

```

This was an exception to catch kgrams that were not apart of the original text.

10.4 Codebase

Makefile :

This Makefile contains the lint.

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lboost_unit_test_framework
DEP = RandWriter.h
OBJS = RandWriter.o TextWriter.o

all: TextWriter test

TextWriter: $(OBJS)
    $(CC) -g -o TextWriter $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
    $(CC) -g -c $(CFLAGS) $< -o $@ $(LFLAGS)

test: test.o $(DEP)
    $(CC) -g -o test test.o RandWriter.o $(LFLAGS)
test.o:test.cpp
    $(CC) -g -c $(CFLAGS) test.cpp $(LFLAGS)


lint:
    cpplint.py --filter=-runtime/references,-build/c++11 --root=. *


cleanall : cleanobj
    rm TextWriter

cleanobj :
    rm *.o
```

TextWriter.cpp :

```
// Copyright [2023] Hunter Hasenfus

#include <string>
#include <iostream>
#include <cstdlib>
#include "RandWriter.h"

int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " k L" << std::endl;
        return 1;
    }
    int k = atoi(argv[1]);
    int L = atoi(argv[2]);
    std::string text;
    std::string line;
    while (std::getline(std::cin, line)) {
        text += line;
    }
    RandWriter rw(text, k);

    //std::cout << rw << rw.generate(text.substr(0, k), L) << std::endl;
    return 0;
}
```

RandWriter.h :

```cpp
// Copyright [2023] Hunter Hasenfus

#ifndef RANDWRITER_H_
#define RANDWRITER_H_

#include <string>
#include <vector>
#include <iostream>
#include <cstdlib>
#include <map>
#include <random>
#include <chrono>


class RandWriter{
 public:
    RandWriter(std::string text, int k);
    int orderK() const;
    int freq(std::string kgram) const;
    int freq(std::string kgram, int c) const;
    char kRand(std::string kgram);
    std::string generate(std::string kgram, int L);
    friend std::ostream& operator<<(std::ostream &out, RandWriter &rw);
 private:
    std::string BaseText;
    std::map<std::string, int> kgrams;
    std::map<std::string, std::vector<float>> kgramCharProb;
    std::map<std::string, std::discrete_distribution<int>> kgramCharDist;
    int order;
    std::vector<char> alphabet;
    std::default_random_engine generator;
};

#endif  // RANDWRITER_H_
```

RandWriter.cpp :

```
// Copyright [2023] Hunter Hasenfus

#include <string>
#include <algorithm>
#include <cstdlib>
#include <iostream>
#include <vector>
#include <map>
#include <random>
#include <chrono>
#include "RandWriter.h"


RandWriter::RandWriter(std::string text, int k) {
    // unsigned nseed = std::chrono::system_clock::now().
    // time_since_epoch().count();
    // generator.seed(nseed);
    BaseText = text;
    BaseText.erase(std::remove(BaseText.begin(), BaseText.end(), '\n'),
     BaseText.cend());
    // std::cout << BaseText << std::endl;
    order = k;
    int x = orderK();
    for (int i = 0; i < 127; i ++) {
        if (BaseText.find(static_cast<char>(i)) != std::string::npos) {
            alphabet.push_back(static_cast<char>(i));
        }
    }
    qsort(&alphabet[0], alphabet.size(), sizeof(char), [](const void *a,
     const void *b) { return (*(char *)a - *(char *)b); });
    for (int i = 0; i < BaseText.length(); i++) {
        std::string kgram = BaseText.substr(i, order);
        if (kgram.size() < order) {
            int x = (i + order)- BaseText.length();
            kgram += BaseText.substr(0, x);

            if (kgrams.find(kgram) == kgrams.end()) {
                kgrams[kgram] = freq(kgram);
                kgramCharProb[kgram] = std::vector<float>(127, 0);
                kgramCharProb[kgram][BaseText[x]] = static_cast<float>
                (freq(kgram,
                 BaseText[x])) / kgrams[kgram];
                // std:: cout << kgramCharProb[kgram][BaseText[x]]
                //  << std::endl;
            } else { if (kgramCharProb[kgram][BaseText[x]] == 0) {
                kgramCharProb[kgram][BaseText[x]] = static_cast<float>
                (freq(kgram,
                 BaseText[x])) / kgrams[kgram];
                //  std:: cout << kgramCharProb[kgram]
                // [BaseText[x]] << std::endl;
            }}

        } else {
        int x = (i + order) == BaseText.length() ? 0 : i + order;

        if (kgrams.find(kgram) == kgrams.end()) {
            kgrams[kgram] = freq(kgram);
            kgramCharProb[kgram] = std::vector<float>(127, 0);
            kgramCharProb[kgram][BaseText[x]] = static_cast<float>
            (freq(kgram, BaseText[x]))
             / kgrams[kgram];
        } else { if (kgramCharProb[kgram][BaseText[x]] == 0) {
            kgramCharProb[kgram][BaseText[x]] = static_cast<float>(freq(
    kgram,
             BaseText[x])) / kgrams[kgram];
    }}}}
    // std::cout << "HERE" << std::endl;
    for (auto it : kgramCharProb) {
        std::discrete_distribution<int> dist(it.second.begin(),
         it.second.end());
        kgramCharDist[it.first] = dist;
    }
    }

int RandWriter::orderK() const {
    return order;
}
int RandWriter::freq(std::string kgram) const {
    if (kgram.length() != order) {
        throw std::invalid_argument("kgram is not the same length as order"
         + kgram);
    }
    if (kgrams.find(kgram) == kgrams.end()) {
        throw std::invalid_argument("kgram is not valid");
    }
    int n = 0;
    for (int i = 0; i < BaseText.length(); i++) {
        std::string kgram2 = BaseText.substr(i, order);
        if (kgram2.size() < order) {
            int x = (i + order)- BaseText.length();
            kgram2 += BaseText.substr(0, x);
            if (kgram2 == kgram) {
                n++;
            }
        } else {
            if (kgram2 == kgram) {
                n++;
            }}
    }
    return n;
}

int RandWriter::freq(std::string kgram, int c) const {
    if (kgram.length() != order) {
        throw std::invalid_argument("kgram is not the same length as order"
    +
         kgram);
    }
    int n = 0;
    for (int i = 0; i < BaseText.length(); i++) {
        std::string kgram2 = BaseText.substr(i, order);

        if (kgram2.size() < order) {
            int x = (i + order)- BaseText.length();
            kgram2 += BaseText.substr(0, x);
            // std::cout << "(1) " << i << " - ";
            // std::cout << BaseText[x] <<  ": " << x << std::endl;
```

```cpp
            if (kgram2 == kgram && BaseText[x] == c) {
                n++;
            }

        } else {
            int x = (i + order) == BaseText.length() ? 0 : i + order;
            // std::cout << "(2) " << i << " - ";
            // std::cout << BaseText[x] <<  ": " << x << std::endl;
            if (kgram2 == kgram  && BaseText[x] == c) {
                n++;
            }}
    }

    if (n == 0) {
        throw std::invalid_argument("kgram does not exist: " + kgram
         + ", " + static_cast<char>(c));
    }

    return n;
}

char RandWriter::kRand(std::string kgram) {
    int x = kgramCharDist[kgram](generator);
    // int i = 0;
    // std::cout << kgram << ":" << std::endl;
    // for (double x: kgramCharDist[kgram].probabilities()) {
    //     if (x > 0)
    //         std::cout << char(i) << "- " << x << std::endl;
    //     i++;
    // }

    return x;
}
std::string RandWriter::generate(std::string kgram, int L) {
    if (kgram.length() != order) {
        throw std::invalid_argument("kgram is not the same length as order")
    ;
    }
    if (kgrams.find(kgram) == kgrams.end()) {
        throw std::invalid_argument("kgram does not exist: " + kgram);
    }
    std::string kgram2 = kgram;
    std::string  output = kgram2;
    for (int i = 0; i < L - kgram.length(); i++) {
        output += static_cast<char>(kRand(kgram2));
        // std::cout << kRand(kgram2) << std::endl;
        // std::cout << "kgram2: " << kgram2 << std::endl;
        kgram2 = output.substr(output.length() - order, order);
    }
    return output;
}
std::ostream& operator<<(std::ostream &out, RandWriter &rw) {
    std::cout << "Order: " << rw.orderK() << std::endl;
    std::cout << "Alphabet: " << std::endl;
    for (char c : rw.alphabet) {
        std::cout << c << ", ";
    }
    std::cout << std::endl << "kgrams frequencies: " << std::endl;
    for (auto x : rw.kgrams) {
        std::cout << x.first << ": " << x.second << std::endl;
    }
    std::cout << "kgrams + 1 frequencies: " <<  std::endl;

    return out;
}
```

test.cpp :

```
// Copyright [2023] Hunter Hasenfus

#define BOOST_TEST_MODULE RandWriterTest
#include <boost/test/included/unit_test.hpp>
#include "./RandWriter.h"

BOOST_AUTO_TEST_SUITE(RandWriterTest)
BOOST_AUTO_TEST_CASE(test1) {
    RandWriter rw("abc", 1);
    BOOST_REQUIRE_THROW(rw.freq("ab"), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(test2) {
    RandWriter rw("abc", 1);
    BOOST_REQUIRE_THROW(rw.freq("ab", 'c'), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(test3) {
    RandWriter rw("abc", 2);
    BOOST_REQUIRE_THROW(rw.freq("xy"), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(test4) {
    RandWriter rw("abc", 2);
    BOOST_REQUIRE_THROW(rw.freq("xy", 'c'), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(test5) {
    RandWriter rw("abc", 1);
    BOOST_REQUIRE_THROW(rw.generate("x", 1), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(test6) {
    RandWriter rw("abc", 1);
    BOOST_REQUIRE_THROW(rw.generate("ab", 2), std::invalid_argument);
}
BOOST_AUTO_TEST_CASE(test7) {
    RandWriter rw("abc", 0);
    BOOST_REQUIRE_EQUAL(rw.orderK(), 0);
}
BOOST_AUTO_TEST_CASE(test8) {
    RandWriter rw("abc", 1);
    BOOST_REQUIRE_EQUAL(rw.kRand("a"), 'b');
}
BOOST_AUTO_TEST_CASE(test9) {
    RandWriter rw("abc", 0);
    BOOST_REQUIRE_EQUAL(rw.freq(""), 3);
}
BOOST_AUTO_TEST_SUITE_END()
```

11 PS7: Kronos Log Parsing

11.1 Discussion:

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

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

I used the lambda expression for getting the time from first 19 characters i.e substring(0,19). I prefer using lambda expression than ordinary method as its more efficient. Just utilized the three functions from the regex library to finish my project.

11.3 Explanation of the code:

Firstly, I initialize regular expressions to the starting and ending log entries so it can be used to compare the regex to find. if there is a match in every line of the file(.log). I create an -outputfile where the data of the result is stored as filename.log.rpt .There is a use of bool exp to keep tracking of the incomplete booting. I utilize the regex-search function to search a match for the starting regular expression(log.c.166). if there is no incomplete booting, I write the date, time, line number and status of the boot to our -outputfile. The else if condn is used to search the line for finding the match for the ending regular expression. if that is found, The calculation of the duration of the time is begun by using posix. Again I write the line-number, date , time and duration it took for the completion of sequence.

11.4 What I learned :

I learnt implementing of the regex library <regex> More efficiently.
I have implemented following regex functions into my code.

- boost::regex startMessage("(
  (log.c.166
) server started)");
- boost::regex endMessage("(oejs.AbstractConnector:Started SelectChannelConnector)");
- regex-search(s, startMessage)
- regex-search(s, endMessage)

11.5 Codebase

Makefile:
This Makefile has lint.

```
CC = g++
CFLAGS = -c -Wall -Werror -std=c++14
LFLAGS = -lboost_regex
DEP =
OBJS =  stdinboost.o

all: ps7

ps7: $(OBJS)
    $(CC) -g -o ps7 $(OBJS) $(LFLAGS)
$(objects): %.o: %.cpp $(DEP)
    $(CC) -g -c $(CFLAGS) $< -o $@ $(LFLAGS)

```

```
lint:
	cpplint.py --filter=-runtime/references,-build/c++11 --root=. *


cleanall : cleanobj
	rm ps7

cleanobj :
	rm *.o
```

stdinboost.cpp:

```cpp
// // Copyright [2023] Hunter M Hasenfus

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

int main(int argc, char* argv[]) {
  std::ifstream lf(argv[1]);
  if (!lf) {
      std::cerr << "Error with log file: " << argv[1] << std::endl;
      return 1;
  }

  std::ofstream of(std::string(argv[1]) + ".rpt");
  if (!of) {
      std::cerr << "Error with output file: "
        << argv[1] << ".rpt" << std::endl;
      return 1;
  }

  std::string line, rs;
  std::regex timePattern(R"((\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}))");
  std::regex startPattern(R"(\((log\.c\.166)\) server started)");
  std::regex finishPattern(R"(oejs\.AbstractConnector:Started
   SelectChannelConnector@\d+\.\d+\.\d+\.\d+:\d+)");
  boost::posix_time::ptime start_t, finish_t;
  int n;
  bool start = false;


  while (getline(lf, line)) {
    n++;

    std::smatch match;
    if (std::regex_search(line, match, timePattern)
      && match.size() > 1) {
        std::string timestamp_str = match[1];
        boost::posix_time::ptime current_time =
        boost::posix_time::time_from_string(timestamp_str);

      if (std::regex_search(line, match, startPattern)) {
          if (start) {
            of << "\t\t**** Incomplete boot **** \n\n";
            start = false;
          } else {
            of << "=== Device boot ===\n";
            of << n << " (" << argv[1] << "): " <<
```

```
              start_t << " Boot Start" << std::endl;
            start_t = current_time;
            start = true;
          }
      } else if (std::regex_search(line, match, finishPattern)) {
          if (start) {
            finish_t = current_time;
            boost::posix_time::time_duration elapsed =
              finish_t - start_t;
            of << n << " (" << argv[1] << "): " <<
              finish_t << " Boot Completed" << std::endl;
            of << "\t\tBoot Time: " <<
              elapsed.total_milliseconds() << "ms\n" << std::endl;
            start = false;
  }}}}



  if (start) {
    of << "=== Device boot ===\n";
    of << n << " (" << argv[1] << "): " <<
    start_t << " Boot Start" << std::endl;
  }

  lf.close();
  of.close();
  return 0;
}
```