Project Portfolio

Daniel Bergeron

COMP IV Sec 203

Spring 2023

**PS0** **Hello World with SFML**

**PS1** Linear Feedback Shift Register and Image Encoding

PS1a Linear Feedback  
 PS1b Image Encoding

**PS2** Sokoban

**PS3** Pythagoras Tree

**PS4** Checkers

**PS5** DNA Sequence Alignment

**PS6** Random Writer

**PS7** Kronos Time Clock

Time to complete: 20hrs

# Table of Contents

**PS0 – Hello World** …………………………………………………………………………………….…….…. 3

**PS1A – Linear Feedback Shift Register** …………………………………….………………….……… 6

**PS1B – Image Encoding** …………………………………………………….…………..…………………… 12

**PS2A – Sokoban** ………………………………..………………………………………………………………… 16

**PS2B – Sokoban** …………………………………………………………………………………………………… 25

**PS3 – Pythagoras Triangles** ………………...…………………………….………………………………… 43

**PS4A – Checkers** …………….…………………...…………………………….………………………………… 50

**PS4B – Checkers** ..………….…………………...…………………………….………………………………… 59

**PS5 – DNA Sequence Alignment** ..……....…………………………….………………………………… 75

**PS6 – Random Writer** …….…………………...…………………………….………………………………… 81

**PS7 – Kronos Time Clock** …………..……....…………………………….………………………………… 91

# PS0 – Hello World

A screenshot of a computer

Description automatically generated with medium confidence

## Program Discussion

PS0- Hello World, was an introductory program to the Computing IV class, as well as an introduction/tutorial for the SFML library. SFML is an API that assists users in many aspects, for this assignment I used SFML to deploy PNG images into a visual environment. The environment I used to launch windows from is called XLAUNCH. I took the image out of its location and loaded it into an object called a sprite which is light weight and easy to interact with object. I was then able to play around with the object’s methods, to resize and move the object around the screen. It should be noted that at the beginning of this project I thought a sprite was an actual sprite drink, hence the theme. Using the sf::Keyboard class I was able to give the user the ability to change the image attributes. When they hit a certain key, the sprite would move, and its size would increase. This program was built out of the tutorial code in the SFML library, hence the green circle which is supposed to represent the world playing into the title of Hello World.

Window Output

## What I Learned

This program acted as a sort of warm up for the heavy amount of coding I would be doing this semester. The main way ps0 did this was re-introducing me to C++ and it forced me to interact with an API that I had very little experience with. To be pedantic the exact items I learned how to interact with were the sf::RenderWindow class, the sf::Keyboard class, the sf::Sprite, the sf::CircleShape and the sf::sprite class. While none of what I accomplished and learned in this program taught me anything new regarding the C++ program. The program introduced me to import aspects of SFML that I would use throughout the semester.

## Issues

In this assignment there was not much in the way of classical programing bugs. However, I had issues getting my window to work properly, initially I was experiencing segmentation faults every time I opened a new window. Eventually through some debugging I was able to get my window to work properly. I was no longer experiencing any issues with segmentation faults, but I was never able to figure out why setting up vertical sync in terminal kept failing. This had no effect on my program, so I ignored the issue and continued with the assignment.

## Code

### Main

1. /\*
2. \*Daniel Bergeron
3. \*COMP IV
4. \*helloworld-ps0
5. \*Dr. Yelena Rykalova
6. \*1/24/2023
7. \*Program tests the smlf library
8. \*
9. \*/
10. #include <SFML/Graphics.hpp>
11. #include <SFML/Audio.hpp>
12. int main()
13. {
14. //keystroke detection variables
15. float x = 400, y = 400, currentXScale = .1, currentYScale = .1 ;
16. sf::RenderWindow window(sf::VideoMode(1200, 1200), "SFML works!");
17. window.setFramerateLimit(60);
18. sf::CircleShape shape(100.f);
19. shape.setFillColor(sf::Color::Green);
20. //load image of sprite
21. sf::Texture texture;
22. sf::Texture texture2;
23. if(!texture.loadFromFile("sprite.png")){
24. return EXIT\_FAILURE;
25. }
26. if(!texture2.loadFromFile("sprite2.png")){
27. return EXIT\_FAILURE;
28. }
29. sf::Sprite sprite;
30. sf::Sprite sprite2;
32. sprite.setTexture(texture);
33. sprite.setPosition(x, y);
34. sprite.setScale(currentXScale, currentYScale);
35. sprite2.setTexture(texture2);
36. sprite2.setPosition(x + 100, y - 100);
37. sprite2.setScale(currentXScale, currentYScale);
38. while(window.isOpen())
39. {
40. sf::Event event;
41. while(window.pollEvent(event)){
42. if(event.type == sf::Event::Closed)
43. window.close();
44. }
45. window.clear();
46. if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
47. {
48. sprite.move(-10.00,0);
49. currentXScale += .01;
50. sprite.setScale(currentXScale, currentYScale);
51. }
52. else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
53. sprite.move(10.00,0);
54. currentXScale += .01;
55. sprite.setScale(currentXScale, currentYScale);
56. }
57. else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
58. sprite.move(0, -10.00);
59. currentYScale += .01;
60. sprite.setScale(currentXScale, currentYScale);
61. }
62. else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down)){
63. sprite.move(0, 10.00);
64. currentYScale += .01;
65. sprite.setScale(currentXScale, currentYScale);
66. }
67. sprite.setRotation(sprite.getRotation() + 20.00 );
68. window.draw(shape);
69. window.draw(sprite);
70. window.draw(sprite2);
71. window.display();
72. }
73. return 0;
74. }

# PS1A – Linear Feedback Shift Register

## Program Discussion

The Goal of this assignment was to create a Linear Feedback shift register (LFSR), the idea is to construct one of these registers and then use it to as a pseudo random number generator. To emulate an LFSR I implemented a class that uses a binary string to construct an object that would represent an LSFR with three tap points. The tap points are the area in the register which is accessed and then XORed together to create the pseudo random aspect of an LFSR. This object would then be used in my main routine to generate these random values. I was fully able Text

Description automatically generatedto create the LFSR. I was successful in my approach to this project because I was able to channel OOP principles to generate an effective class system to handle the register. Another Important aspect this program introduced, was testing, and it is at this point that I must introduce the boost library, this API allows a programmer to efficiently test their code for any issues. I had two drivers for this program, one for my main routine, and one for my boost test cases. Boost allowed me to really test my code and by the end of the project I was able to pass all my cases, and my main routine worked as it was supposed to.

Main Routine & Test Case Output

## Key Elements Discussion

One of the main features of this program that led to my success, was its object-oriented approach. The area this was applied was in creating my virtual, LSFR. I took the idea of an LSFR and wrapped It into a class. This class was responsible for building my virtual LSFR. Essentially my class gave the user an interphase to interact with that acted as a LSFR. From the outside looking in it would be very difficult to tell that you weren’t dealing with an actual LSFR but rather something designed to emulate it. I accomplished this with a key principle of OOP called encapsulation essentially, I took a set of methods and variables and wrapped them into one concise thing that the user does not get to see. If you are looking at my main routine all you see is the object and the methods and variable that I allow you to see. This level of abstraction made the main routine in my code simple. In addition, this program also contained a header, that held all the important information about my LSFR class but abstracted away any information about implementation. This allowed me to create a very easy to read program, with a wide range of functionality. For example, there was a step operation that took the inner representation of the LSFR and moved it left once and then generated a new bit. There was also a generate function that generated a random number for the user to use. Since the eternal representation of this code is in an area where the user can’ t reach it makes it very hard to break code. Behind all this the data which the user was manipulating was my own personal take on the best internal representation. For this I used another object called a bit set which acted as an interphase between the me and what in my head was a 16-bit place in memory. Me being the user in this situation I could only interact with the object so anything under the hood did not concern me. Also, in this assignment I overloaded the extraction operator to output my data in way that would make since to the idea of a LSFR register. This plays into a different component of OOP called polymorphism, or the idea that a function can mean different things depending on how it is applied. This paradigm of programming that I implemented for this project allowed me to complete this project.

## What I Learned

In this assignment, I learned and improved a great deal. Firstly, I learned how to use the boost testing library for my code. I also learned what an LFSR was, to create an object to understand it I had to have a great understanding of how the machine worked. This was also my first experience using the bit set library which was a great asset for me when I was trying to come up with internal representation of the LFSR. As a programmer this program improved my understanding of OOP design and how powerful it is at simplifying code.

## Issues

In this program, I faced potentially my most tragic mistake as a programmer. I didn’t back up my files and an error caused me to accidentally delete my entire program. I had to rush a couple hours before the due date to finish the assignment. This became one of my fastest ever programmed assignments. From this I learned to always back up code. One good way is creating a GitHub repository.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
3. DEPS = FibLFSR.hpp
4. OBJ = FibLFSR.o main.o
5. OBJ\_A = FibLFSR.o test.o
6. LIBS = -lboost\_unit\_test\_framework
7. all: ps1 driver
8. ps1: $(OBJ) $(DEPS)
9. $(CC) $(CFLAGs) -o ps1 $(OBJ)
10. driver: $(OBJ\_A) $(DEPS)
11. $(CC) $(CFLAGs) -o driver $(OBJ\_A) $(LIBS)
12. test.o: test.cpp
13. $(CC) $(CFLAGS) -c test.cpp -o test.o
14. main.o: main.cpp
15. $(CC) $(CFLAGS) -c main.cpp -o main.o
16. FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
17. $(CC) $(CFLAGS) -c FibLFSR.cpp -o FibLFSR.o
18. clean:
19. rm -f $(OBJ) main.o test.o ps1 driver

### Main.cpp

1. #include "FibLFSR.hpp"
2. int main(int argc, char\* argv[]){
3. FibLFSR test1("0110110001101100");
4. int newNum = 0, newNum2 = 0;
5. cout << "Stepping a LFSR\n";
6. for(int i = 10; i >= 0; i--){
7. newNum = test1.step();
8. cout << test1 << " " << newNum << endl;
9. }
10. cout << endl;
11. cout << "Random Number Generation\n";
12. FibLFSR test2("1100011011000011");
13. for (int i = 0; i < 6; i++) {
14. newNum2 = test2.generate(5);
15. cout << test2 << " " << newNum2 << endl;
16. }
17. return 0;
18. }

### FibLFSR.hpp

1. #include <bitset>
2. #include <string>
3. #include <iostream>
4. using namespace std;
5. class FibLFSR {
6. public:
7. FibLFSR(std::string seed); // constructor to create LFSR with
8. // the given initial seed
9. int step(); // simulate one step and return the
10. // new bit as 0 or 1
11. int generate(int k); // simulate k steps and return
12. // k-bit integer
13. friend ostream& operator<<(ostream& out, FibLFSR reg);
14. private:
15. std::bitset<16> data;
16. };

### FibLFSR.cpp

1. #include "FibLFSR.hpp"
2. FibLFSR::FibLFSR(std::string seed){
3. if(seed.size() != 16){
4. cout << "ERROR: seed must have size of 16\n" ;
5. return;
6. }
7. std::bitset<16> newData(seed);
8. data = newData;
9. }
10. ostream& operator<<(ostream& out, FibLFSR reg)
11. {
12. out << reg.data;
13. return out;
14. }
15. int FibLFSR::step(){
16. int tap1 = 15, tap2 = 13, tap3 = 12 , tap4 = 10;
17. bool newVal = false;
18. std::bitset<16> newData = data;
19. for(int i = 15; i > 0; i--){
20. newData[i] = data[i -1];
21. }
22. newVal = data[tap1] ^ data[tap2];
23. newVal = newVal ^ data[tap3];
24. newVal = newVal ^ data[tap4];
25. newData[0] = newVal;
26. data = newData;
27. return newVal;
28. }
29. int FibLFSR::generate(int k){
30. int value = 0;
31. for(int i = 0; i < k; i++ ){
32. value \*= 2;
33. value += step();
34. }
35. return value;
36. }

### test.cpp

1. // Dr. Rykalova
2. // test.cpp for PS1a
3. // updated 1/31/2020
4. #include <iostream>
5. #include <string>
6. #include "FibLFSR.hpp"
7. #define BOOST\_TEST\_DYN\_LINK
8. #define BOOST\_TEST\_MODULE TEST
9. #include <boost/test/unit\_test.hpp>
11. BOOST\_AUTO\_TEST\_CASE(sixteenBitsThreeTaps) {
12. FibLFSR l("1011011000110110");
13. BOOST\_REQUIRE(l.step() == 0);
14. BOOST\_REQUIRE(l.step() == 0);
15. BOOST\_REQUIRE(l.step() == 0);
16. BOOST\_REQUIRE(l.step() == 1);
17. BOOST\_REQUIRE(l.step() == 1);
18. BOOST\_REQUIRE(l.step() == 0);
19. BOOST\_REQUIRE(l.step() == 0);
20. BOOST\_REQUIRE(l.step() == 1);
21. FibLFSR l2("1011011000110110");
22. BOOST\_REQUIRE(l2.generate(9) == 51);
23. }
24. BOOST\_AUTO\_TEST\_CASE(ZeroTest) {
25. FibLFSR l("0000000000000000");
26. for(int i = 0; i < 1000; i++){
27. BOOST\_REQUIRE(l.step() == 0);
28. }
29. }
30. BOOST\_AUTO\_TEST\_CASE(OnesTest) {
31. FibLFSR l("1111111111111111");
32. BOOST\_REQUIRE(l.generate(11) == 0);
33. BOOST\_REQUIRE(l.generate(12) != 0)}

# PS1B – Image Encoding

## Program discussion

As shown by the header this program is built on the original program and introduced an interesting use of the LFSR. That use is the encoding and decoding of an image’s pixels using the pseudo random generation of the LFSR. This program forced me to interact with the SFML library, which as discovered in PS0 is a great way to manipulate and use images. With this program I was able to take a PNG image and scramble its pixels. Then I could take that scrambled image, send it back through the program and get the same image back. Thus, demonstrating the powers of the LFSR for encoding and decoding. I was able to accomplish this with a cat image.

Background pattern

Description automatically generatedA picture containing cat, indoor, mammal, domestic cat

Description automatically generated

Window Output

## Key Elements Discussion

The key elements of this program are the same as the previous assignment since it is built of it. This Program still implements the OOP design elements of part A. I did have to interact more with SFML more, which is something that employs Object-Oriented principles. I had to interact with the image object far more in this program. This is because I had to use the LFSR on each individual pixel in the PNG. I changed the color of each of the images based on the number in the LFSR, and that is what scrambles the image.

## What I Learned

In this program I learned to better understand the object I created I the previous assignments and how to better use it. The heavy focus of this program was on the main routine since it interacted with SFML and is responsible for the interactions between my object and the ones in the SFML library. From this assignment I learned more about the draw loops in main that maintain the windows that SFML uses. But most of all this program taught me that images are data and data can be manipulated in ways that help or hurt the user depending on what your objective is.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. DEPS = FibLFSR.hpp
4. OBJ = FibLFSR.o PhotoMagic.o
5. LIBS = -lsfml-window -lsfml-graphics -lsfml-system
6. all: ps1B
7. ps1B: $(OBJ)
8. $(CC) $(CFLAGS) -o ps1B $(OBJ) $(LIBS)
9. %.o: %.cpp $(DEPS)
10. $(CC) $(CFLAGS) -c $<
11. clean:
12. rm -f $(OBJ) ps1B

### PhotoMagic.cpp

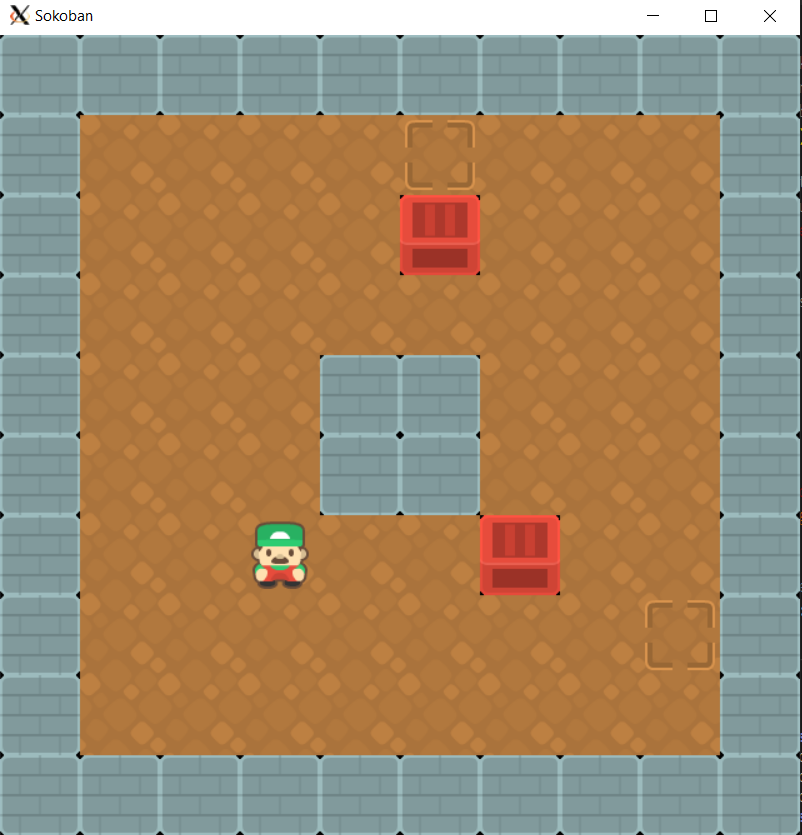
1. // pixels.cpp:
2. // using SFML to load a file, manipulate its pixels, write it to disk
3. // Fred Martin, fredm@cs.uml.edu, Sun Mar  2 15:57:08 2014
4. // g++ -o pixels pixels.cpp -lsfml-graphics -lsfml-window
5. #include <SFML/System.hpp>
6. #include <SFML/Window.hpp>
7. #include <SFML/Graphics.hpp>
8. #include "FibLFSR.hpp"
9. #include <vector>
10. void transform( sf::Image& image, FibLFSR\* password);
11. int main(int argc, char\* argv[])
12. {
13. string inputFileName, outputFileName, seed;
14. cin >> inputFileName >> outputFileName >> seed;
15. sf::Image input;
16. if (!input.loadFromFile(inputFileName))
17. return -1;
18. sf::Image output;
19. if (!output.loadFromFile(inputFileName))
20. return -2;
21. if(seed.size() != 16)
22. return -3;
23. FibLFSR password(seed);
24. output = input;
25. cout << "input in!\n";
26. transform(output, &password);
27. cout << "Transformation Complete\n";
28. sf::Vector2u size1 = input.getSize();
29. sf::RenderWindow window1(sf::VideoMode((size1.x), (size1.y)), "Input");
30. sf::Vector2u size2 = output.getSize();
31. sf::RenderWindow window2(sf::VideoMode((size2.x), (size2.y)), "output");
32. sf::Texture texture1;
33. texture1.loadFromImage(input);
34. sf::Texture texture2;
35. texture2.loadFromImage(output);
36. sf::Sprite sprite1;
37. sprite1.setTexture(texture1);
38. sf::Sprite sprite2;
39. sprite2.setTexture(texture2);
40. while (window1.isOpen() && window2.isOpen()) {
41. sf::Event event;
42. while (window1.pollEvent(event)) {
43. if (event.type == sf::Event::Closed){
44. window1.close();
45. }
46. }
47. while (window2.pollEvent(event)) {
48. if (event.type == sf::Event::Closed){
49. window2.close();
50. }
51. }
52. window1.clear(sf::Color::White);
53. window1.draw(sprite1);
54. window1.display();
55. window2.clear(sf::Color::White);
56. window2.draw(sprite2);
57. window2.display();
58. }
59. if (!output.saveToFile("output-file.png"))
60. return -1;
61. return 0;
62. }
63. void transform( sf::Image& image, FibLFSR\* password){
64. sf::Color newColor;
65. int8\_t redVal, greenVal, blueVal;
66. sf::Vector2u size = image.getSize();
67. for(unsigned int x = 0; x < size.x; x++){
68. for(unsigned int y = 0; y < size.y; y++){
69. redVal = password->generate(8);
70. greenVa l = password->generate(8);
71. blueVal = password->generate(8);
72. newColor = image.getPixel(x , y);
73. newColor.r = newColor.r ^ redVal;
74. newColor.g = newColor.g ^ greenVal;
75. newColor.b = newColor.b ^ blueVal;
76. image.setPixel(x, y, newColor);
77. }
78. }
79. return;
80. }

# PS2A – Sokoban

## Program Discussion

A picture containing table

Description automatically generatedThis program is one of a two-part program. The goal of this program is to load and display a Sokoban map using SFML. This project was challenging because it involved a lot of moving parts. I was able to load and display the map, but due to lack of foresight I was unable to see that some decisions I made in this program would have a very negative effect on my next version of this project. PS2A focuses on taking in input, storing it, and the using that internal representation to generate a game map in an interactive window. I was able to accomplish this in a series of steps. The first being creating a place to store data, but to do that I needed to know where that data was coming from. Lucky enough that part was given. The data came from the file stream in the form of character symbols which were place holders for the types of objects that should be used in their place. This was useful because the layout of the input data gave me positional data on where objects should be on the game map, as well as the size of the game map. To gain the benefits of the input’s layout, I needed to load it directly into the object I created. So, in my class definition I overloaded the insertion operator to take a file stream and put its content directly into my object. In my main function I got the file and instantiated my Sokoban object. It is here that I made a critical error that would make this project much more difficult than it needed to be. The mistake was not building in a static map background into the object. In either case, the constructor, or the insertion I could have created a static background for the object. This doesn’t seem like a big deal for this part because the entire display is going to be static. And, while this is true, it was not good foresight to leave mix the static elements with the future would be dynamic items. This also would have been an easy fix. In my class definition, I used a matrix of special objects called block, to store each symbol. I chose a matrix so I could mirror the grid layout of the input file and by extension the game board. I could have made a separate constant matrix called background that loads the entire background floor tiles, and the walls if I wanted to be more pedantic. Moving past this design oversight the matrix in combination with my block class created a nice interphase for my main routine to interact with. It’s also important note, that the Sokoban class inherits from the sf::drawable class. This feature allowed me to override SFMLs draw function and teach it how to draw my object. This was difficult to understand at first, but it took a lot of stress off me. Due to the fact that this function and this function alone were responsible for all the drawing happening in the main routine, I essentially didn’t have to worry about the SFML aspect of the program. While making my methods I left everything visual up to the draw function. In conclusion my matrix system did allow my program to handle a diverse set A picture containing chart

Description automatically generatedof maps. After this teaching my program how to handle and display input, I needed only teach it how to play the game. Decisions I made in this assignment’s implementation translated for better or worse into the following program.

Input

Window Output

## Key Elements Discussion

To create this program, I had to use some powerful programming tools. Firstly, to construct my matrix data structure I implemented the C++ vector library. Then with my vector objects I created a vector of vectors that mimicked how an actual matrix would work. I used the idea of row major order to implement this matrix. I also used a vector that holds class objects called gameobjects. I think of these game objects as actual blocks that represent an in-game tile. Using the insertion operator, I gathered symbols from the input stream, forged them into blocks and inserted them into this vector. From this vector I moved the objects into the matrix, and destroyed the vector. These blocks keep track of their own position, so when finally put them into my matrix they are in the correct position. Those are the two main data structures I used In this program. The next important attribute of this program is its OOP elements. The program uses inheritance to take information from the parent class sf::drawable and deliver it to Sokoban class. This is then used in a polymorphic way regarding the draw function. This Sokoban class is further abstracted by the GameObject class who handles the more precise interactions between blocks. All of this is encapsulated away from the user inside my classes. Somethings like the drawable class are even abstracted away from me. Another interesting aspect of this program is it is the first of my programs to employ a linter. Linter is a program that goes through code and checks for stylization errors. The implementation of a liner helped me produce neat and organized code.

## What I Learned

This program introduced two new things to me. The first of these things is the draw function. This was the first time I used inheritance and polymorphism in this way. The SFML API abstracted away a lot of details, so I had to try to understand things from a top-down perspective, where things were abstracted away. It took me a while in this project to figure out how the draw function interacted with the API. The second thing that was completely new to me was the Linter. Before this I didn’t even know these scripts existed, let alone how they benefited my time. Instead of searching through hundreds of lines of code for formatting errors I had the Linter tell me exactly where and what I need to fix. In addition, it gave me tips to do this. This program also built off a lot of things I am already familiar with, the most significant being OOP. Out of all the programs I’ve done this year this one increased my understanding the most.

## Issues

In this code I couldn’t figure out how to implement the linter in my makefile so I used it as a VScode extension.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = Sokoban.o main.o
4. LIBS = -lsfml-window -lsfml-graphics -lsfml-system
5. DEPS = Sokoban.hpp
6. all: test
7. test: $(OBJ)
8. $(CC) $(CFLAGS) -o test $(OBJ) $(LIBS)
9. %.o: %.cpp $(DEPS)
10. $(CC) $(CFLAGS) -c $<
11. clean:
12. rm -f $(OBJ) test

### Main.cpp

1. /\* "Copyright [2023] <Daniel Bergeron>" \*/
2. #include "Sokoban.hpp"
3. // Function abstracts the grunt work of
4. // making the visuals away from main
5. // pre: Is passed a valid Sokoban object
6. void generateGameMap(Sokoban& level);
7. // pre: Passed valid texture and int pair by refrence
8. // post: pair will be given new value
9. void getTexturePixelSize(sf::Texture texture, pair<int, int>& cor);
10. int main() {
11. // Simple driver to load and test maps
12. Sokoban lv;
13. ifstream fp;
14. // test1
15. fp.open("level1.txt");
16. fp >> lv;
17. generateGameMap(lv);
18. fp.close();
19. // test2
20. fp.open("level2.txt");
21. fp >> lv;
22. generateGameMap(lv);
23. fp.close();
24. // stub test
25. lv.movePlayer(noMove);
26. return 0;
27. }
28. void generateGameMap(Sokoban& level) {
29. sf::Texture texture, ground;;
30. sf::Sprite sprite, floor;
31. sf::Event event;
32. sf::Vector2f location;
33. pair<int, int> textureSize;
34. GameObject block;
35. bool drawn = false;
36. // gets pixel sizes & creats a window that will fit all gamObjects
37. texture.loadFromFile("sokoban/images/environment\_03.png");
38. getTexturePixelSize(texture, textureSize);
39. sf::RenderWindow window(sf::VideoMode(textureSize.first \* level.getWidth(),
40. textureSize.second \* level.getHeight()), "Input");
41. // Game Loop, draw and maintain map
42. while (window.isOpen()) {
43. while (window.pollEvent(event)) {
44. if (event.type == sf::Event::Closed) {
45. window.close();
46. }
47. }
48. if (drawn == false) {
49. // pulls blocks from matrix to be drawn
50. for (int y = 0; y < level.getHeight(); y++) {
51. for (int x = 0; x < level.getWidth(); x++) {
52. block = level.getGameObject(y , x);
53. setDrawPos(location, block, textureSize);
54. // Special Case Player Block, ground needs to be drawn first
55. if (block.getTypeObject() == player) {
56. ground.loadFromFile("sokoban/images/ground\_01.png");
57. floor.setTexture(ground);
58. floor.setPosition(location);
59. window.draw(floor);
60. }
61. // Draws and updates window
62. level.setSprite(block);
63. sprite = level.getSprite();
64. sprite.setPosition(location);
65. window.draw(sprite);
66. window.display();
67. }
68. }
69. drawn = true;
70. }
71. }
72. }
73. void getTexturePixelSize(sf::Texture texture, pair<int, int>& cor) {
74. sf::Vector2u pixels;
75. texture.loadFromFile("sokoban/images/environment\_03.png");
76. pixels = texture.getSize();
77. cor.first = static\_cast<int>(pixels.x);
78. cor.second = static\_cast<int>(pixels.y);
79. return;
80. }

### Sokoban.hpp

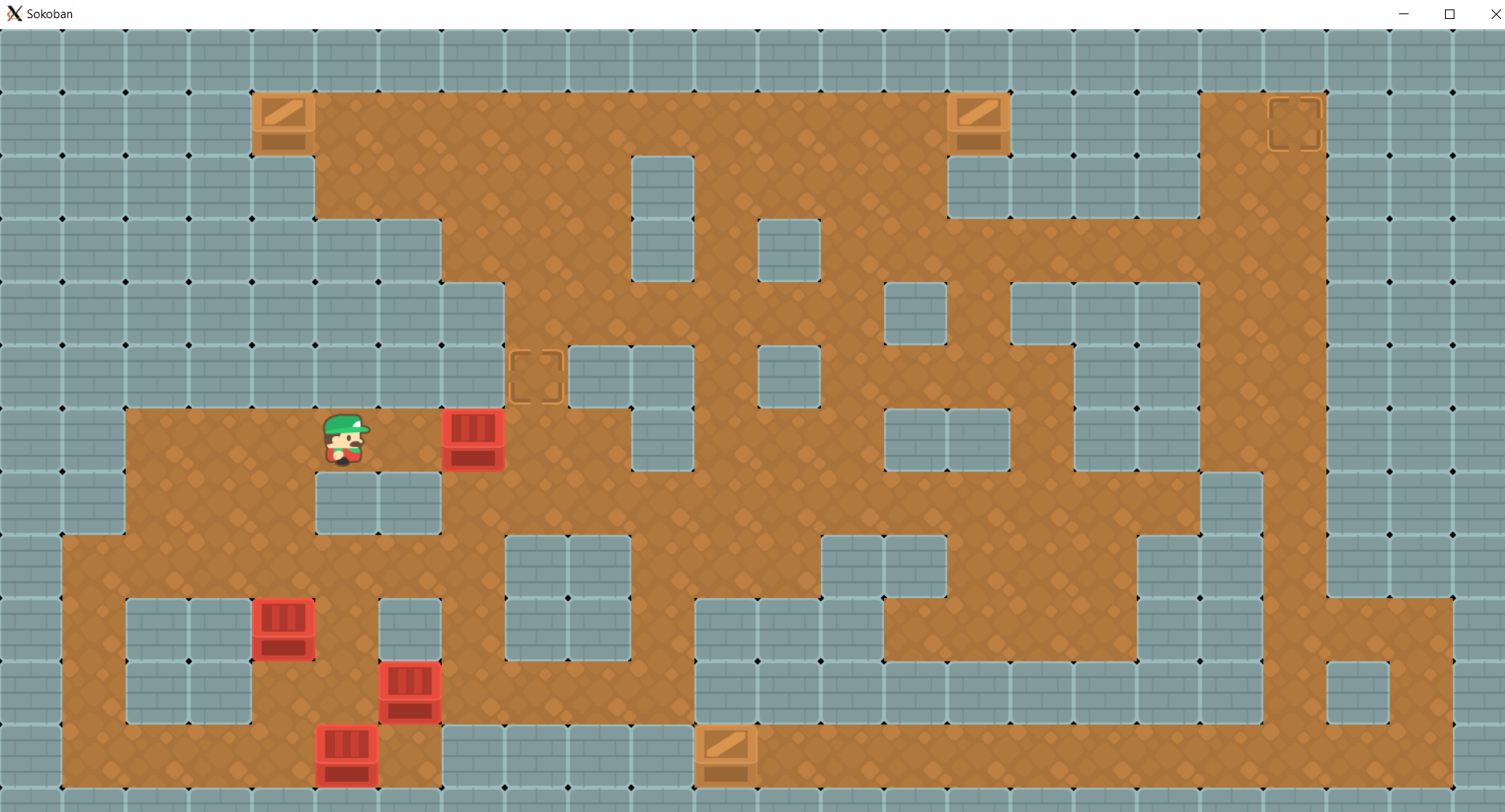
1. /\* "Copyright [2023] <Daniel Bergeron>" \*/
2. #include <fstream>
3. using std::ifstream;
4. #include <utility>
5. using std::pair;
6. #include <iostream>
7. using std::cin;
8. using std::cout;
9. using std::endl;
10. #include <vector>
11. using std::vector;
12. #include <SFML/System.hpp>
13. #include <SFML/Window.hpp>
14. #include <SFML/Graphics.hpp>
15. // enum describes set of object types and move types
16. enum typeObject{nill, player, wall, floor, crate, storage\_space, stored};
17. enum moveSet{noMove, up, down, left, right};
18. class GameObject {
19. private:
20. int x;
21. int y;
22. typeObject object;
23. public:
24. // constructors
25. GameObject() : x(0), y(0), object(nill) {}
26. GameObject(int \_x, int \_y, typeObject \_object);
27. // accessors
28. int getXlocation(void) const {return x; }
29. int getYlocation(void) const {return y; }
30. typeObject getTypeObject(void) const {return object; }
31. // mutators
32. void setBlock(int \_x, int \_y, typeObject \_object) {x = \_x; y = \_y; object = \_object; }
33. };
34. class Sokoban : public sf::Drawable {
35. public:
36. // TODO(Daniel): develop moveplayer in part B
37. void movePlayer(moveSet move);
38. // muttator
39. void setSokobanMatrix(vector<GameObject>& blocks);
40. void setSprite(GameObject& block);
41. // accessors
42. // pre: passed x & y values inside bounds of matrix
43. // post: returns GameObject at location
44. GameObject& getGameObject(const int y, const int x) {return matrix[y][x]; }
45. int getWidth(void) {return xSize; }
46. int getHeight(void) {return ySize; }
47. sf::Sprite getSprite(void) {return sprite; }
48. friend ifstream& operator>>(ifstream& in, Sokoban& object);
49. private:
50. // overides sf::Drawable virtual function
51. virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
52. int xSize;
53. int ySize;
54. sf::Sprite sprite;
55. sf::Texture texture;
56. vector<vector<GameObject>> matrix;
57. };
58. // pre: is passed valid file stream & Sokoban object
59. // post: returns file stream & fills Sokoban
60. ifstream& operator>>(ifstream& in, Sokoban& object);
61. // insert is a helper fucntion for the >> operator overload
62. // pre: takes a vaild filestream, vector of gameObjects and coordintes
63. // post: returns true if item was succesfully put in vector
64. bool insert(ifstream& in, vector<GameObject>& objects, int x, int y);
65. // pre: Is passed valid Vector2f, GameObjects, and int pair, pixel sizes
66. // post: returns updated an updated Vector2f
67. void setDrawPos(sf::Vector2f& location, const GameObject& block, const pair<int, int>& xyCor);

### Sokoban.cpp

1. /\* "Copyright [2023] <Daniel Bergeron>" \*/
2. #include "Sokoban.hpp"
3. GameObject::GameObject(int \_x, int \_y, typeObject \_object) {
4. x = \_x;
5. y = \_y;
6. object = \_object;
7. }
8. void Sokoban::movePlayer(moveSet) {
9. // TODO(Daniel): Finish.
10. cout << "Player Doesn't feel like moving!\n";
11. return;
12. }
13. void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
14. target.draw(sprite, states);
15. }
16. void Sokoban::setSprite(GameObject& block) {
17. // pulls textures from files based of block type
18. switch (block.getTypeObject()) {
19. case player:
20. texture.loadFromFile("sokoban/images/player\_05.png");
21. break;
22. case wall:
23. texture.loadFromFile("sokoban/images/block\_06.png");
24. break;
25. case floor:
26. texture.loadFromFile("sokoban/images/ground\_01.png");
27. break;
28. case crate:
29. texture.loadFromFile("sokoban/images/crate\_03.png");
30. break;
31. case storage\_space:
32. texture.loadFromFile("sokoban/images/ground\_04.png");
33. break;
34. default:
35. // Will set any other GameObject to wall
36. // TODO(Daniel): Make code to deal with states that
37. // are stored in matirx
38. texture.loadFromFile("sokoban/images/block\_06.png");
39. }
40. sprite.setTexture(texture);
41. }
42. ifstream& operator>>(ifstream& in, Sokoban& level) {
43. // objects and var
44. int xtemp, ytemp;
45. vector<GameObject> objects, tempVec;
46. // make sure matirx is clear before it is used again.
47. level.matrix.clear();
48. // take map size input
49. in >> xtemp >> ytemp;
50. if (xtemp <= 0 && ytemp <= 0) {
51. exit(1);
52. }
53. level.xSize = xtemp;
54. level.ySize = ytemp;
55. // make matrix outline
56. level.matrix.resize(ytemp);
57. for (int i = 0; i < static\_cast<int>(level.matrix.size()); i++) {
58. for ( int j = 0; j < xtemp; j++ ) {
59. GameObject empty;
60. level.matrix[i].push\_back(empty);
61. }
62. }
63. // start reading chars
64. // use to build Game objects
65. // place in input vector then build matrix
66. if (insert(in, objects, xtemp, ytemp)) {
67. level.setSokobanMatrix(objects);
68. }
69. return in;
70. }
71. bool insert(ifstream& in, vector<GameObject>& objects, int xLimit, int yLimit ) {
72. char input;
73. int x = 0, y = 0;
74. // Takes input and puts into a vector
75. // based of what the char input is.
76. // Two guards at the end to ensure
77. // it dosen't go out of bounds.
78. while ( !(in.eof()) ) {
79. GameObject block;
80. in >> input;
81. switch ( input ) {
82. case '@':
83. block.setBlock(x, y, player);
84. break;
85. case '#':
86. block.setBlock(x, y, wall);
87. break;
88. case '.':
89. block.setBlock(x, y, floor);
90. break;
91. case 'A':
92. block.setBlock(x, y, crate);
93. break;
94. case 'a':
95. block.setBlock(x, y, storage\_space);
96. break;
97. case '1':
98. block.setBlock(x, y, stored);
99. break;
100. default:
101. return false; }
102. x++;
103. if (y < yLimit) {
104. objects.push\_back(block);
105. }
106. if (x >= xLimit) {
107. y++;
108. x = 0;
109. }
110. }
111. return true;
112. }
113. void Sokoban::setSokobanMatrix(vector<GameObject>& blocks) {
114. // puts block in matrix based of the objects
115. // predetermined location
116. int x = 0, y = 0;
117. int val = blocks.size();
118. for (int i = 0; i < val; i++) {
119. x = blocks[i].getXlocation();
120. y = blocks[i].getYlocation();
121. matrix[y][x] = blocks[i];
122. }
123. }
124. void setDrawPos(sf::Vector2f& location, const GameObject& block, const pair<int, int>& xyCor) {
125. location.x = block.getXlocation() \* xyCor.first;
126. location.y = block.getYlocation() \* xyCor.second;
127. }

# PS2B – Sokoban

## Program Discussion

In this program I built of the features I had in PS2A. This program implemented player movement, object movement and win conditions. This program is the first time I Had to manipulate visuals in this way. I took basic PNG images and brought them to life. This program required me to take a player PNG image and link it with the keyboard so when the play presses a key the image would respond and move in that direction. Then I built a system that would allow my player image to interact with the environment. I need rules, what blocks can the player move onto, what blocks can the player not move past, and what blocks can the player interact with. The player needs to move freely across the floor blocks. It needed to be restricted by the wall blocks. The player needed to be able to move the crate block, and finally change the crate block’s state when it reaches a storage block. To accomplish this, I maintained my gameObject blocks that held that data. I then had the matrix for movement and whenever a move command was inputted it directly manipulates the GameObjects in the matrix. So, pressing the ‘w’ key activated the move function in my class which moved the block to the next point in the matrix. However as stated earlier the design oversight I made in the previous assignment came into play. When a player block moved it treated a floor block as something dynamic, hence the program did a few weird things. First every time

Window Output

the player moved the block underneath them state changed. This caused floor blocks to transform into other things. Sometimes it would just disappear. To fix this, I was forced to complicate my classes. More specifically my move function, which needed to have an added level of complexity to handle the niche bugs that kept appearing. The problem became amplified when the player interacted with crate blocks. Now my program had two blocks that were changing the states of other blocks around them. To combat this spiraling issue, I had to add a new class that handled the transition. I also had to add complicated conditionals to check the matrix for these potential situations. This whole situation that took me hours to fix could have been avoided if I had only made a static background. Essentially what I did by not including the static background was give the player the ability to manipulate the background. Through hours of tedious effort, I produced the setMoveObjects and the executeMoveObjects functions which resolved this issue. Though it should be noted that to accomplish this, the functions contain copious amounts of spaghetti code. If I ever redo this project this will be the first edit, I make. Other than this glaring issue there were some fun SFML feature I got to work with, I learned how to display text in my window. I was able to download a font and use it in my game. Learning how to use resources like fonts, sound bites and images was one of the more enjoyable parts of this project. Other parts of the project that I implemented successfully was the win conditions, player directional movement and finally the ability to restart the game at any point. For the win condition I created a function called isWon.This function went through the matrix and checked the state of the map. If all the crates had a stored state, the endgame loop triggered. I was able to accomplish the extra credit, that asked for the players orientation to change based on the direction of movement. I also implemented a feature that allowed a player to press a restart key, and the entire game would be set back to its original state. There where two extra credit options that I wanted to accomplish but could not be due to time and other issues. The first is I wanted to be able to play a sound when the player one. I had the sound bite implemented, but I had compatibility issues with my computer drivers and the SFML library that I could not resolve. The undo feature I couldn’t implement due to time. However, if I had the chance to implement it, I would have created a stack object and would have pushed the state of the game onto the stack after every turn, and when the undo button was pressed, I would set the current state to the top of the stack then pop it. The final aspect of this program I felt I exceled at was implementing a texture map. Texture objects are heavy so creating new texture every time a sprite needs it heavily impacts program performance. To avoid this, I put all my textures in a map, thus enabling me to create multiple sprites that use the same object. I managed to accomplish everything I set out to do with this assignment and more. I learned many good lessons from this assignment, some the hard way.

Window Output

## Key Elements Discussion

This program forced me to deal with OOP up close. I feel that while I was able to use some important object-oriented programing principles, I feel I fell short. My main class in this program is very heavy and I should have broken it down into smaller classes. My Sokoban class can only be used I am very specific way and doesn’t feel modular. Yet despite this, my Sokoban program has improved my understanding extensively. In this program I also implemented an algorithm from the C++ algorithm library, I used the for\_range function with Lambda. The lambda Is what checks the win condition, and it is applied over the entire matrix by the for\_range function.

## What I Learned

The important lesson I learned from this code was how bad design decisions can have consequences that are not always apparent until later, where they can become big problems. I learned it’s always better to pay the inconvenience tax in the moment to make your code more robust, rather than deal with it later.

## Issues

There was one noticeable but small bug in my final program. If the player spams the reset button the player object bounces around the map. I only have theories for why this happens, but it seems the operation of reading the map into the Sokoban class is a heavy operation, and the spamming the restart button might change some data in the input buffer. I did not have time to really debug this, and it was an issue that did not affect the game at all. The only way the issue became present is if you were actively seeking to cause it, an average player would not run into it.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = Sokoban.o main.o
4. LIBS = -lsfml-window -lsfml-graphics -lsfml-system
5. DEPS = ps2b/Sokoban.hpp
6. all: Sokoban lint
7. Sokoban: $(OBJ)
8. $(CC) $(CFLAGS) -o Sokoban $(OBJ) $(LIBS)
9. lint:
10. cpplint \*.cpp \*.hpp
11. %.o: %.cpp $(DEPS)
12. $(CC) $(CFLAGS) -c $<
13. clean:
14. rm -f $(OBJ) Sokoban lint

### Main.cpp

1. /\* "Copyright [2023] <Daniel Bergeron>" \*/
2. #include <string>
3. #include "ps2b/Sokoban.hpp"
4. moveSet getKeyBoardInput(void);
5. bool resetLevel(moveSet reKey, ifstream& fp, Sokoban& lv, std::string gameMap);
6. int main(int argc, char\* argv[]) {
7. // Simple driver to load and test maps
8. Sokoban lv;
9. ifstream fp;
10. moveSet input = noMove;
11. std::string gameMap;
12. int fontSize = 100;
13. gameMap = argv[1];
14. fp.open(gameMap);
15. if (fp.fail()) {
16. cout << "Failed to open file\n";
17. return -1;
18. }
19. fp >> lv;
20. fp.close();
21. sf::Text text, text2;
22. sf::Font font;
23. if (!font.loadFromFile("endGameFont.ttf")) {
24. cout << "Font failed to load\n";
25. }
26. text.setFont(font);
27. text2.setFont(font);
28. text.setString("YOU WON!!");
29. text2.setString("Press Q to quit");
30. text.setCharacterSize(100);
31. text2.setCharacterSize(30);
32. text.setPosition((lv.getTextureSizeX() \* lv.getWidth())/2.50,
33. (lv.getTextureSizeY() \* lv.getHeight())/2.00);
34. text.setFillColor(sf::Color::Green);
35. text2.setFillColor(sf::Color::Red);
36. text.setStyle(sf::Text::Bold | sf::Text::Underlined);
37. text2.setStyle(sf::Text::Bold | sf::Text::Underlined);
38. auto endGame = [](moveSet in) {  // Lambda expression
39. if (in == quiet) {
40. return true;
41. }
42. return false;
43. };
44. sf::RenderWindow window1(sf::VideoMode(lv.getTextureSizeX() \* lv.getWidth(),
45. lv.getTextureSizeY() \* lv.getHeight()), "Sokoban");
46. while (window1.isOpen()) {
47. sf::Event event;
48. while (window1.pollEvent(event)) {
49. if (event.type == sf::Event::Closed) {
50. window1.close();
51. }
52. }
53. input = getKeyBoardInput();
54. if (endGame(input)) {
55. return 0;
56. }
57. if ( resetLevel(input, fp, lv, gameMap) ) {
58. window1.clear(sf::Color::Black);
59. input = getKeyBoardInput();
60. }
61. lv.movePlayer(input);
62. window1.draw(lv);
63. window1.display();
64. while (lv.isWon() == true) {
65. // inside the main loop, between window.clear()
66. // and window.display()
67. window1.draw(text);
68. window1.draw(text2);
69. if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {
70. return 0;
71. }
72. input = getKeyBoardInput();
73. if (resetLevel(input, fp, lv, gameMap)) {
74. window1.clear(sf::Color::Black);
75. }
76. window1.display();
77. while (window1.pollEvent(event)) {
78. if (event.type == sf::Event::Closed) {
79. window1.close();
80. return 0;
81. }
82. }
83. }
84. }
85. return 0;
86. }
87. bool resetLevel(moveSet reKey, ifstream& fp, Sokoban& lv, std::string gameMap) {
88. Sokoban newlv;
89. if (reKey == restart) {
90. fp.open(gameMap);
91. fp >> lv;
92. fp.close();
93. return true;
94. }
95. return false;
96. }
97. moveSet getKeyBoardInput(void) {
98. if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
99. while (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {}
100. return up;
101. } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
102. while (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {}
103. return down;
104. } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
105. while (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {}
106. return left;
107. } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
108. while (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {}
109. return right;
110. } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {
111. while (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {}
112. return restart;
113. } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {
114. return quiet;
115. } else {
116. return noMove;
117. }
118. }

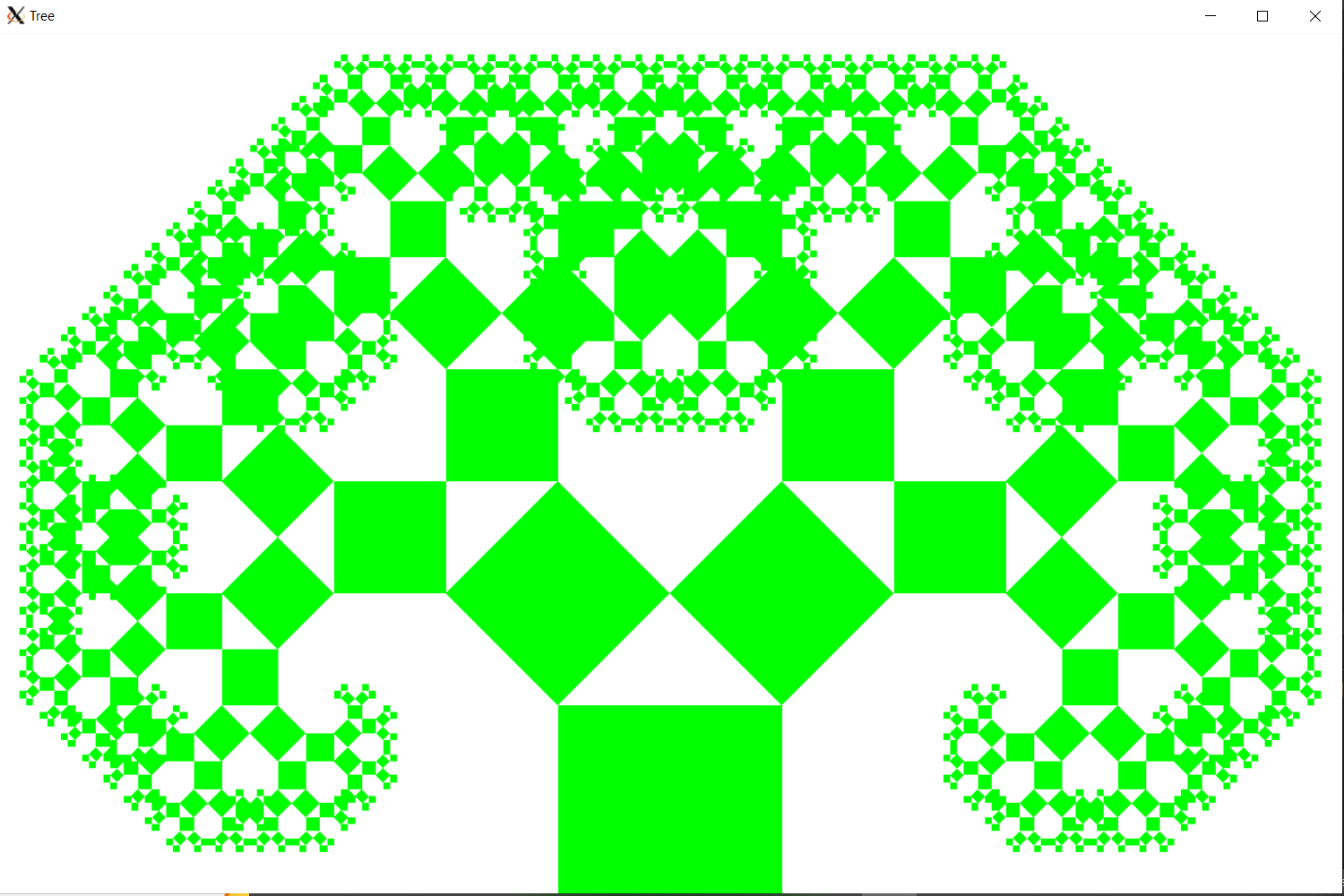
### Sokoban.hpp

1. #ifndef \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS2B\_PS2B\_SOKOBAN\_HPP\_
2. #define \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS2B\_PS2B\_SOKOBAN\_HPP\_
3. /\* "Copyright [2023] <Daniel Bergeron>" \*/
4. #include <fstream>
5. using std::ifstream;
6. #include <utility>
7. using std::pair;
8. #include <iostream>
9. using std::cin;
10. using std::cout;
11. using std::endl;
12. #include <vector>
13. using std::vector;
14. #include <map>
15. using std::map;
16. #include <algorithm>
17. #include <SFML/System.hpp>
18. #include <SFML/Window.hpp>
19. #include <SFML/Graphics.hpp>
20. #include <SFML/Audio.hpp>
21. // enum describes set of object types and move types
22. enum typeObject{nill, player, playerDown, playerUp,
23. playerLeft, playerRight, wall, floor, crate, storage\_space, stored};
24. enum moveSet{noMove, up, down, left, right, restart, quiet};
25. class MoveObject {
26. public:
27. // constructor
28. MoveObject();
29. // accessor
30. pair<int, int> getCurrSpace(void)const {return currSpace; }
31. pair<int, int> getNextSpace(void)const {return nextSpace; }
32. bool getMovable(void) {return moveable; }
33. // mutators
34. void setCurrSpace(pair<int, int> \_next) {currSpace = \_next; }
35. void setNextSpace(pair<int, int> \_next) {nextSpace = \_next; }
36. void setMoveable(bool \_moveable) {moveable = \_moveable; }
37. private:
38. bool moveable;
39. pair<int, int> currSpace;
40. pair<int, int> nextSpace;
41. };
42. class GameObject {
43. private:
44. int x;
45. int y;
46. typeObject object;
47. typeObject currTypeUnder;  // the type under if object is layered
48. sf::Sprite sprite;
49. public:
50. // constructors
51. GameObject() : x(0), y(0), object(nill), currTypeUnder(nill) {}
52. GameObject(int \_x, int \_y, typeObject \_object);
53. // accessors
54. int getXlocation(void) const {return x; }
55. int getYlocation(void) const {return y; }
56. typeObject getCurrTypeUnder(void) const {return currTypeUnder; }
57. typeObject getTypeObject(void) const {return object; }
58. sf::Sprite getSprite(void) const {return sprite; }
59. // mutators
60. void setBlock(int \_x, int \_y, typeObject \_object,
61. map<typeObject, sf::Texture>& texture);
62. void setCurrTypeUnder(typeObject type) {currTypeUnder = type; }
63. void setSprite(sf::Sprite \_sprite) {sprite = \_sprite; }
64. };
65. class Sokoban : public sf::Drawable {
66. public:
67. // constructor
68. Sokoban();
69. // pre: is given a move command
70. // post: player will be moved by command
71. void movePlayer(moveSet move);
72. // muttator
73. void setSokobanMatrix(const vector<GameObject>& blocks);
74. void setPlayerPosition(int x, int y) {playerLocation.first = x,
75. playerLocation.second = y; }
76. // accessors
77. // pre: passed x & y values inside bounds of matrix
78. // post: returns GameObject at location
79. GameObject& getGameObject(const int y, const int x) {return matrix[y][x]; }
80. typeObject getTypeUnderAtPosition(pair<int, int>);
81. int getWidth(void) const {return xSize; }
82. int getHeight(void) const  {return ySize; }
83. int getTextureSizeX(void) const {return textureSize.first; }
84. int getTextureSizeY(void) const {return textureSize.second; }
85. GameObject& getNextSpace(moveSet move, pair<int, int> CurrObjectLoc);
86. const bool isWon(void);
87. friend ifstream& operator>>(ifstream& in, Sokoban& object);
88. private:
89. // overides sf::Drawable virtual function
90. virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
91. // is a helper fucntion for the >> operator overload
92. // pre: takes a vaild filestream, vector of gameObjects and coordintes
93. // post: returns true if item was succesfully put in vector
94. bool buildGameObjectVector(ifstream& in, vector<GameObject>& objects);
95. // pre: takes valid type and texture reffrence
96. // post: sets inside texture map
97. void setTextureInMap(typeObject type, sf::Texture& texture);
98. // pre: takes commands
99. // post: returns list of commands in order
100. void setMoveObjects(moveSet move);
101. // pre: is called
102. // updates matrix using movevlocks
103. bool executeMoveObjects(moveSet move);
104. // pre: is given two valid move blocks
105. // post: places them in matrix location
106. void moveMatrixBlocks(const GameObject& replaceBlock,
107. const GameObject& movingBlock);
108. // pre: is given a move
109. // post: returns enum for animation type
110. typeObject getPlayerMovement(moveSet move);
111. int xSize;
112. int ySize;
113. int pointsToWin;
114. typeObject lastMove = player;
115. MoveObject nextPlayerMove;
116. MoveObject nextBlockMove;
117. pair<int, int> playerLocation;
118. pair<int, int> currentCrateLocation;
119. pair<int, int> textureSize;
120. vector<vector<GameObject>> matrix;
121. map<typeObject, sf::Texture> textureMap;
122. };
123. // pre: is passed valid file stream & Sokoban object
124. // post: returns file stream & fills Sokoban
125. ifstream& operator>>(ifstream& in, Sokoban& object);
126. // pre: Is passed valid Vector2f, GameObjects, and int pair, pixel sizes
127. // post: returns updated an updated Vector2f
128. void setDrawPos(sf::Vector2f& location, const GameObject& block,
129. const pair<int, int>& xyCor);
130. // pre: passed a valid texture size
131. // post: returns a pair with xy size
132. void getTexturePixelSize(const sf::Texture&, pair<int, int>& cor);
133. #endif  // \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS2B\_PS2B\_SOKOBAN\_HPP\_

### Sokoban.cpp

1. /\* "Copyright [2023] <Daniel Bergeron>" \*/
2. #include "ps2b/Sokoban.hpp"
3. GameObject::GameObject(int \_x, int \_y, typeObject \_object) {
4. x = \_x;
5. y = \_y;
6. object = \_object;
7. currTypeUnder = \_object;
8. }
9. void GameObject::setBlock(int \_x, int \_y, typeObject \_object,
10. map<typeObject, sf::Texture>& texture) {
11. // reasigns textures
12. x = \_x;
13. y = \_y;
14. object = \_object;
15. // gets texture from map of stored textures
16. // so no new textures are created.
17. sprite.setTexture(texture[\_object]);
18. if (\_object == playerDown || \_object == playerLeft
19. || \_object == playerRight || \_object == playerUp) {
20. object = player;
21. }
22. }
23. MoveObject::MoveObject() {
24. pair<int, int> empty;
25. empty.first = 0;
26. empty.second = 0;
27. currSpace = empty;
28. nextSpace = empty;
29. moveable = false;
30. }
31. Sokoban::Sokoban() {
32. sf::Texture newText;
33. xSize = 0;
34. ySize = 0;
35. newText.loadFromFile("sokoban/images/crate\_02.png");
36. getTexturePixelSize(newText, textureSize);
37. textureMap[stored] = newText;
38. pointsToWin = 0;
39. playerLocation.first = 0;
40. playerLocation.second = 0;
41. }
42. // gets the pixels sizes of the the texture passed in
43. void getTexturePixelSize(const sf::Texture& texture, pair<int, int>& cor) {
44. sf::Vector2u pixels;
45. pixels = texture.getSize();
46. cor.first = static\_cast<int>(pixels.x);
47. cor.second = static\_cast<int>(pixels.y);
48. return;
49. }
50. // functions takes move
51. // checks for no move then
52. // sets Sokobans move Objects then
53. // it excuetes the commands within
54. // the move objects
55. void Sokoban::movePlayer(moveSet move) {
56. MoveObject moveInfo;
57. if (move == noMove) {
58. return;
59. }
60. setMoveObjects(move);
61. executeMoveObjects(move);
62. return;
63. }
64. // returns the type under from anypoint in the matrix
65. // importent for setting new blocks
66. typeObject Sokoban::getTypeUnderAtPosition(pair<int, int> position) {
67. return matrix[position.second][position.first].getCurrTypeUnder();
68. }
69. // checks win condition
70. // returns win/true when all
71. // points of ben aquired by player
72. const bool Sokoban::isWon(void) {
73. // std::count\_if(v.begin(), v.end(), [](int i) { return i % 4 == 0; });
74. int points = 0;
75. std::for\_each(matrix.begin(), matrix.end(),
76. [&](vector<GameObject> i) {
77. for (GameObject obj : i) {
78. if (obj.getTypeObject() == stored) {
79. points++;
80. }
81. }
82. });
83. // cout << "pointed scored score: " << points << endl;
84. if (pointsToWin > points) {
85. return false;
86. } else {
87. return true;
88. }
89. }
90. // function goal is to validate if the next space is avaible for movement
91. void Sokoban::setMoveObjects(moveSet move) {
92. GameObject nextBlock, nextCrateBlock;
93. pair<int, int> previousLoc, newLoc;
94. // see if player can move
95. // getSpace at move
96. nextPlayerMove.setMoveable(false);
97. nextBlockMove.setMoveable(false);
98. nextBlock = getNextSpace(move, playerLocation);
99. // no move if wall or object is in state stored
100. if (nextBlock.getTypeObject() == wall
101. || nextBlock.getTypeObject() == stored) {
102. nextPlayerMove.setMoveable(false);
103. return;
104. // if next block is crate condition
105. } else if (nextBlock.getTypeObject() == crate) {
106. // checks to see if a crate is a moveable object
107. currentCrateLocation.first = nextBlock.getXlocation();
108. currentCrateLocation.second = nextBlock.getYlocation();
109. nextCrateBlock = getNextSpace(move, currentCrateLocation);
110. // if wall or another crate both player and crate can't move
111. if (nextCrateBlock.getTypeObject() == wall
112. || nextCrateBlock.getTypeObject() == crate) {
113. nextBlockMove.setMoveable(false);
114. nextPlayerMove.setMoveable(false);
115. } else if (nextCrateBlock.getTypeObject() == storage\_space
116. || nextCrateBlock.getTypeObject() == floor) {
117. // set move player
118. nextPlayerMove.setMoveable(true);
119. nextPlayerMove.setCurrSpace(playerLocation);
120. newLoc.first = nextBlock.getXlocation();
121. newLoc.second = nextBlock.getYlocation();
122. nextPlayerMove.setNextSpace(newLoc);
123. // set move block
124. nextBlockMove.setMoveable(true);
125. previousLoc.first = nextBlock.getXlocation();
126. previousLoc.second = nextBlock.getYlocation();
127. nextBlockMove.setCurrSpace(previousLoc);
128. newLoc.first = nextCrateBlock.getXlocation();
129. newLoc.second = nextCrateBlock.getYlocation();
130. nextBlockMove.setNextSpace(newLoc);
131. }
132. return;
133. }
134. // default condtion player is movable
135. nextPlayerMove.setMoveable(true);
136. nextPlayerMove.setCurrSpace(playerLocation);
137. newLoc.first = nextBlock.getXlocation();
138. newLoc.second = nextBlock.getYlocation();
139. nextPlayerMove.setNextSpace(newLoc);
140. }
141. // checks what game object is at the next space based of user input
142. GameObject& Sokoban::getNextSpace(moveSet move, pair<int, int> CurrObjectLoc) {
143. int nextX, nextY;
144. nextX = CurrObjectLoc.first;
145. nextY = CurrObjectLoc.second;
146. switch (move) {
147. case up:
148. nextY -= 1;
149. break;
150. case down:
151. nextY += 1;
152. break;
153. case left:
154. nextX -= 1;
155. break;
156. case right:
157. nextX += 1;
158. break;
159. default:
160. return matrix[CurrObjectLoc.first][CurrObjectLoc.second];
161. break;
162. }
163. return matrix[nextY][nextX];
164. }
165. // function uses two moveObjects one to replace
166. // the object on the space it is being moved from
167. // i.e. the replace object, the other object is
168. // the one being move to another location the moving
169. // object, which retains its curr type but the type
170. // under changes to that of the object which was
171. // originaly there
172. bool Sokoban::executeMoveObjects(moveSet move) {
173. GameObject replaceBlock, movingBlock;
174. typeObject playerMovement = player;
175. pair<int, int> oldPosition, newPosition;
176. if (nextPlayerMove.getMovable() == false
177. && nextBlockMove.getMovable() == false) {
178. return false;
179. } else if (nextPlayerMove.getMovable() == true
180. && nextBlockMove.getMovable() == true) {
181. // move crate & updateMap;
182. oldPosition = nextBlockMove.getCurrSpace();
183. newPosition = nextBlockMove.getNextSpace();
184. // if crate is on storage space change to stored state
185. if (getTypeUnderAtPosition(newPosition) == storage\_space) {
186. replaceBlock.setBlock(oldPosition.first, oldPosition.second,
187. getTypeUnderAtPosition(oldPosition), textureMap);
188. // set stored state on crate at new location
189. movingBlock.setBlock(newPosition.first,
190. newPosition.second, stored, textureMap);
191. movingBlock.setCurrTypeUnder(getTypeUnderAtPosition(newPosition));
192. moveMatrixBlocks(replaceBlock, movingBlock);
193. } else {
194. replaceBlock.setBlock(oldPosition.first, oldPosition.second,
195. getTypeUnderAtPosition(oldPosition), textureMap);
196. replaceBlock.setCurrTypeUnder(replaceBlock.getTypeObject());
197. movingBlock.setBlock(newPosition.first,
198. newPosition.second, crate, textureMap);
199. movingBlock.setCurrTypeUnder(getTypeUnderAtPosition(newPosition));
200. moveMatrixBlocks(replaceBlock, movingBlock);
201. }
202. // move player, updateMap, update currLocation
203. oldPosition = nextPlayerMove.getCurrSpace();
204. newPosition = nextPlayerMove.getNextSpace();
205. replaceBlock.setBlock(oldPosition.first, oldPosition.second,
206. getTypeUnderAtPosition(oldPosition), textureMap);
207. replaceBlock.setCurrTypeUnder(replaceBlock.getTypeObject());
208. // moving block is player
209. playerMovement = getPlayerMovement(move);
210. movingBlock.setBlock(newPosition.first,
211. newPosition.second, playerMovement, textureMap);
212. movingBlock.setCurrTypeUnder(
213. matrix[newPosition.second][newPosition.first].getTypeObject());
214. playerLocation = newPosition;
215. moveMatrixBlocks(replaceBlock, movingBlock);
216. return true;
217. } else if (nextPlayerMove.getMovable() == true
218. && nextBlockMove.getMovable() == false) {
219. oldPosition = nextPlayerMove.getCurrSpace();
220. newPosition = nextPlayerMove.getNextSpace();
221. // always get the type that will be under player
222. replaceBlock.setBlock(oldPosition.first, oldPosition.second,
223. getTypeUnderAtPosition(oldPosition), textureMap);
224. replaceBlock.setCurrTypeUnder(replaceBlock.getTypeObject());
225. //  player
226. playerMovement = getPlayerMovement(move);
227. movingBlock.setBlock(newPosition.first,
228. newPosition.second, playerMovement, textureMap);
229. movingBlock.setCurrTypeUnder(
230. matrix[newPosition.second][newPosition.first].getTypeObject());
231. playerLocation = newPosition;
232. moveMatrixBlocks(replaceBlock, movingBlock);
233. return true;
234. }
235. return false;
236. }
237. typeObject Sokoban::getPlayerMovement(moveSet move) {
238. switch (move) {
239. case up:
240. return playerUp;
241. case down:
242. if (lastMove == player) {
243. lastMove = playerDown;
244. return playerDown;
245. } else {
246. lastMove = player;
247. return player;
248. }
249. return playerDown;
250. case left:
251. return playerLeft;
252. case right:
253. return playerRight;
254. default:
255. return player;
256. }
257. }
258. // replaces the block being moved with replace block
259. // and puts moving block in its new position
260. void Sokoban::moveMatrixBlocks(const GameObject& replaceBlock,
261. const GameObject& movingBlock) {
262. matrix[replaceBlock.getYlocation()][replaceBlock.getXlocation()]
263. = replaceBlock;
264. matrix[movingBlock.getYlocation()][movingBlock.getXlocation()]
265. = movingBlock;
266. }
267. // renders the game map using the information
268. // contained in the matrix's gameObjects
269. void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
270. GameObject block;
271. sf::Vector2f location;
272. sf::Texture floorTexture;
273. sf::Sprite floorSprite, sprite;
274. map<typeObject, sf::Texture>::const\_iterator pos = textureMap.find(floor);
275. floorTexture = pos->second;
276. floorSprite.setTexture(floorTexture);
277. target.draw(sprite, states);
278. // pulls blocks from matrix to be drawn
279. for (int y = 0; y < ySize; y++) {
280. for (int x = 0; x < xSize; x++) {
281. block = matrix[y][x];
282. setDrawPos(location, block, textureSize);
283. // if there is a layered block the type under must be drawn first
284. if (block.getTypeObject() == player
285. || block.getTypeObject() == stored) {
286. pos = textureMap.find(block.getCurrTypeUnder());
287. floorTexture = pos->second;
288. floorSprite.setTexture(floorTexture);
289. floorSprite.setPosition(location);
290. target.draw(floorSprite, states);
291. }
292. sprite = matrix[y][x].getSprite();
293. sprite.setPosition(location);
294. target.draw(sprite, states);
295. }
296. }
297. }
298. bool Sokoban::buildGameObjectVector(ifstream& in, vector<GameObject>& objects) {
299. char input;
300. int x = 0, y = 0;
301. // Takes input and puts into a vector
302. // based of what the char input is.
303. // Two guards at the end to ensure
304. // it dosen't go out of bounds.
305. while ( !(in.eof()) ) {
306. GameObject block;
307. sf::Texture texture;
308. in >> input;
309. switch ( input ) {
310. case '@':
311. texture.loadFromFile("sokoban/images/player\_up.png");
312. setTextureInMap(playerUp, texture);
313. texture.loadFromFile("sokoban/images/player\_down.png");
314. setTextureInMap(playerDown, texture);
315. texture.loadFromFile("sokoban/images/player\_left.png");
316. setTextureInMap(playerLeft, texture);
317. texture.loadFromFile("sokoban/images/player\_right.png");
318. setTextureInMap(playerRight, texture);
319. texture.loadFromFile("sokoban/images/player\_05.png");
320. setTextureInMap(player, texture);
321. block.setBlock(x, y, player, textureMap);
322. playerLocation.first = x;
323. playerLocation.second = y;
324. block.setCurrTypeUnder(floor);
325. break;
326. case '#':
327. texture.loadFromFile("sokoban/images/block\_06.png");
328. setTextureInMap(wall, texture);
329. block.setBlock(x, y, wall, textureMap);
330. block.setCurrTypeUnder(floor);
331. break;
332. case '.':
333. texture.loadFromFile("sokoban/images/ground\_01.png");
334. setTextureInMap(floor, texture);
335. block.setBlock(x, y, floor, textureMap);
336. block.setCurrTypeUnder(floor);
337. break;
338. case 'A':
339. texture.loadFromFile("sokoban/images/crate\_03.png");
340. setTextureInMap(crate, texture);
341. block.setBlock(x, y, crate, textureMap);
342. block.setCurrTypeUnder(floor);
343. break;
344. case 'a':
345. texture.loadFromFile("sokoban/images/ground\_04.png");
346. setTextureInMap(storage\_space, texture);
347. block.setBlock(x, y, storage\_space, textureMap);
348. block.setCurrTypeUnder(storage\_space);
349. pointsToWin++;
350. break;
351. case '1':
352. texture.loadFromFile("sokoban/images/ground\_01.png");
353. setTextureInMap(crate, texture);
354. block.setBlock(x, y, stored, textureMap);
355. block.setCurrTypeUnder(floor);
356. break;
357. default:
358. return false; }
359. x++;
360. if (y < ySize) {
361. objects.push\_back(block);
362. }
363. if (x >= xSize) {
364. y++;
365. x = 0;
366. }
367. }
368. return true;
369. }
370. void Sokoban::setSokobanMatrix(const vector<GameObject>& blocks) {
371. // puts block in matrix based of the objects
372. // predetermined location
373. int x = 0, y = 0;
374. int val = blocks.size();
375. for (int i = 0; i < val; i++) {
376. x = blocks[i].getXlocation();
377. y = blocks[i].getYlocation();
378. matrix[y][x] = blocks[i];
379. }
380. }
381. ifstream& operator>>(ifstream& in, Sokoban& level) {
382. // objects and var
383. int xtemp, ytemp;
384. vector<GameObject> objects, tempVec;
385. level.pointsToWin = 0;
386. // make sure matirx is clear before it is used again.
387. level.matrix.clear();
388. // take map size input
389. in >> xtemp;
390. in >> ytemp;
391. if (xtemp <= 0 && ytemp <= 0) {
392. exit(1);
393. }
394. level.xSize = xtemp;
395. level.ySize = ytemp;
396. // make matrix outline
397. level.matrix.resize(ytemp);
398. for (int i = 0; i < static\_cast<int>(level.matrix.size()); i++) {
399. for ( int j = 0; j < xtemp; j++ ) {
400. GameObject empty;
401. level.matrix[i].push\_back(empty);
402. }
403. }
404. // start reading chars
405. // use to build Game objects
406. // place in input vector then build matrix
407. if (level.buildGameObjectVector(in, objects)) {
408. level.setSokobanMatrix(objects);
409. }
410. return in;
411. }
412. // loads new textures into map
413. void Sokoban::setTextureInMap(typeObject type, sf::Texture& texture) {
414. textureMap[type] = texture;
415. }
416. // get drawing position of game object
417. void setDrawPos(sf::Vector2f& location,
418. const GameObject& block, const pair<int, int>& xyCor) {
419. location.x = block.getXlocation() \* xyCor.first;
420. location.y = block.getYlocation() \* xyCor.second;
421. }

# PS3 – Pythagoras Tree



## Program Discussion

This assignment required me to implement a Pythagorean Tree. The idea was to create a class called PTree that uses a recursive function to draw fractal in the shape of a tree. I was able to accomplish this by using the visual capabilities of the SFML library and a vector data structure. On top of this, to do the math required by this programming I had to use the Cmath library. This is the first time in a C++ program I was required to use a C library. It added an interesting dynamic to my linter, since the linter requires the header file to be ordered in a specific way. This program did not require much in the way of programing, the logic behind it made it difficult. This program forced me to really think about recursion. Recursions can be difficult to manage, but I was able to break it down. I used a recursion to create squares of different orientations and place them in a vector for the draw function to use later. I broke the recursion into two functions. The first function was not recursive. It was responsible for the base case, a larger square that would act as the platform from which the other squares are derived. After the base case was handled the function called the recursive function. This recursive function made the next two squares and then called itself twice on the two new squares. This program also used the command line to get information about how to draw the shape. I created a class to store sf::rectangles. Though this class did not do much for this program, I wanted to create something modular and simple to interact with SFML in my place. In this program the SFML documentation was crucial. Once I figured out the logic, the only difficult part of this program was the draw function. Initially I was only using global bounds to interact with the sf::rectangle object. This however refers to its location on my entire screen, and not its position in the window I generated. This caused the program shapes to draw in strange areas. The other problem in my function to calculate the size of a square is that I accidentally made a miss calculation that resulted in my square size being negative. This did not cause a crash but rather it inverted all the positional data in my square object causing them to draw in random places. Overall, a good assignment, that furthered my understanding of recursion.

## Key Elements Discussion

Vectors were crucial for me since they allowed me to create all the visuals in the recursion and store them for later use. This made my draw function very simple since all it had to do was run through the vector and draw each object. I did not want to mix the visuals with the background work so created independent systems for each which were called in the driver together to accomplish the task of making the tree. Another important element of this code was the logic behind the recursive elements of the program. In each step the program uses the previous lengths to calculate the next size and trigonometry properties to calculate the angles of the new rectangles. This was done in two recursions. I used this to work down the left and write sub-tree. This project required me to develop an interesting algorithm and implement it using object-oriented programing. This project really forced me to think about algorithms in a different way. I feel it has grown my ability to understand and translate problems.

## What I Learned

In this program I learned the true power of recursion, and how it can take a complex problem and trivialize it. If you looked at the output of this code alone, you would think this program is a very complex problem, but the recursion simplified the code greatly.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = PTree.o PTree\_driver.o
4. LIBS = -lsfml-window -lsfml-graphics -lsfml-system
5. DEPS = PTree.hpp
6. all: PTree lint
7. PTree: $(OBJ)
8. $(CC) $(CFLAGS) -o PTree $(OBJ) $(LIBS)
9. lint:
10. cpplint \*.cpp \*.hpp
11. %.o: %.cpp $(DEPS)
12. $(CC) $(CFLAGS) -c $<
13. clean:
14. rm -f $(OBJ) PTree lint

### Main.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "ps3\_header/PTree.hpp"
3. int main(int argc, char\* argv[]) {
4. string lBase, nDepth;
5. double L;
6. int N;
7. lBase = argv[1];
8. nDepth = argv[2];
9. L = stod(lBase);
10. N = stoi(nDepth);
11. cout << "The Base triangle size input is " << L << endl;
12. cout << "The depth input is " << N << endl;
13. PTree sappling(N, L);
14. sappling.pTree();
15. sf::RenderWindow window1(sf::VideoMode(6 \* L, 4 \* L), "Tree");
16. while (window1.isOpen()) {
17. sf::Event event;
18. while (window1.pollEvent(event)) {
19. if (event.type == sf::Event::Closed) {
20. window1.close();
21. }
22. }
23. window1.clear(sf::Color::White);
24. window1.draw(sappling);
25. window1.display();
26. }
27. return 0;
28. }

### PTree.hpp

1. #ifndef \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS3\_PS3\_HEADER\_PTREE\_HPP\_
2. #define \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS3\_PS3\_HEADER\_PTREE\_HPP\_
3. // Copyright [2023] <Daniel Bergeron>
4. // ptree class that derives from sf::Drawable
5. #include <iostream>
6. using std::cout;
7. using std::endl;
8. #include <utility>
9. using std::pair;
10. #include <string>
11. using std::string;
12. using std::stoi;
13. using std::stod;
14. #include <algorithm>
15. #include <vector>
16. using std::vector;
17. #include <cmath>
18. using std::sin;
19. using std::cos;
20. #include <SFML/System.hpp>
21. #include <SFML/Window.hpp>
22. #include <SFML/Graphics.hpp>
23. const float THETA = 45;
24. // need an object to hold rectangle
25. // hold a pair for its map loaction
26. // tells draw func where and how to draw
27. class PRec {
28. public:
29. // constructors
30. PRec();
31. PRec(sf::RectangleShape \_rectangle, int \_theta);
32. // accessors
33. sf::RectangleShape getRectangleShape(void) {return rectangle; }
34. int getTheta(void) {return theta; }
35. // muttators
36. void SetRectangleShape(sf::RectangleShape \_rectangle)
37. {rectangle = \_rectangle; }
38. void setTheta(int \_theta) {theta = \_theta; }
39. private:
40. int theta;
41. sf::RectangleShape rectangle;
42. };
43. class PTree : public sf::Drawable {
44. public:
45. PTree();
46. PTree(int \_depth, double \_baseRecSize);
47. // uses recurison to produce tree
48. // pre: is called
49. // post: tree is drawn
50. void pTree(void);
51. private:
52. virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
53. void pTreeRecursion(int currdepth, PRec base);
54. // helper function to ptree
55. // this function makes a rectangle
56. vector<PRec> treeRectangle;
57. int depth;
58. double baseRecSize;
59. };
60. #endif  //  \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS3\_PS3\_HEADER\_PTREE\_HPP\_

### PTree.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "ps3\_header/PTree.hpp"
3. PRec::PRec() {
4. theta = 0;
5. rectangle.setSize(sf::Vector2f(0, 0));
6. }
7. PRec::PRec(sf::RectangleShape \_rectangle, int \_theta) {
8. rectangle = \_rectangle;
9. theta = \_theta;
10. }
11. PTree::PTree() {
12. depth = 0;
13. baseRecSize = 0.0;
14. }
15. PTree::PTree(int \_depth, double \_baseRecSize) {
16. depth = \_depth;
17. baseRecSize = \_baseRecSize;
18. }
19. void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
20. sf::RectangleShape shape;
21. PRec rectangle;
22. int i;
23. for (i = 0; i < treeRectangle.size(); i++) {
24. rectangle = treeRectangle[i];
25. shape = rectangle.getRectangleShape();
26. shape.setFillColor(sf::Color::Green);
27. target.draw(shape, states);
28. }
29. return;
30. }
31. void PTree::pTree(void) {
32. if (depth <= 0 && baseRecSize <= 0) {
33. return;
34. }
35. sf::RectangleShape baseShape(sf::Vector2f(static\_cast<float>(baseRecSize),
36. static\_cast<float>(baseRecSize)));
37. PRec basePRec;
38. pair<float, float> coord;
39. coord.first = (baseRecSize \* 3.0) - (baseShape.getSize().x/2.0);
40. coord.second = (baseRecSize \* 3.5) - (baseShape.getSize().x/2.0);
41. baseShape.setPosition(coord.first, coord.second);
42. basePRec.SetRectangleShape(baseShape);
43. basePRec.setTheta(0);
44. treeRectangle.push\_back(basePRec);
45. pTreeRecursion(depth, basePRec);
46. }
47. // function will make two rectangles and push them onto the vector
48. void PTree::pTreeRecursion(int currdepth, PRec base) {
49. if (currdepth == 0) {
50. return;
51. }
52. PRec newLeftPRec, newRightPRec;
53. sf::RectangleShape rectangleLeft, rectangleRight;
54. sf::Vector2f sizeL, sizeR;
55. // set the size
56. sizeL.x = (base.getRectangleShape().getSize().x \* cos((M\_PI/180) \* THETA));
57. sizeL.y = sizeL.x;
58. cout << sizeL.x << endl;
59. sizeR.x = (base.getRectangleShape().getSize().y \* sin((M\_PI/180) \* THETA));
60. sizeR.y = sizeR.x;
61. rectangleLeft.setSize(sizeL);
62. rectangleRight.setSize(sizeR);
63. // make new theta
64. newLeftPRec.setTheta(base.getTheta() - THETA);
65. newRightPRec.setTheta(base.getTheta() + THETA);
66. // positioning
67. sf::Vector2f posL, posR;
68. sf::Rect gridLoc = base.getRectangleShape().getLocalBounds();
69. rectangleLeft.setOrigin(0, rectangleLeft.getSize().y);
70. rectangleRight.setOrigin(rectangleRight.getLocalBounds().width,
71. rectangleRight.getLocalBounds().height);
72. posL = base.getRectangleShape().getTransform().
73. transformPoint(gridLoc.left, gridLoc.top);
74. posR = base.getRectangleShape().getTransform().
75. transformPoint(gridLoc.left + gridLoc.width, gridLoc.top);
76. rectangleLeft.setPosition(posL);
77. rectangleRight.setPosition(posR);
78. // applying to rectangle object
79. rectangleLeft.setRotation(newLeftPRec.getTheta());
80. rectangleRight.setRotation(newRightPRec.getTheta());
81. newLeftPRec.SetRectangleShape(rectangleLeft);
82. newRightPRec.SetRectangleShape(rectangleRight);
83. // push
84. treeRectangle.push\_back(newLeftPRec);
85. treeRectangle.push\_back(newRightPRec);
86. pTreeRecursion(currdepth - 1, newLeftPRec);
87. pTreeRecursion(currdepth - 1, newRightPRec);
88. return;
89. }

# PS4A – Checkers

## Program Discussion

This assignment involved setting up a checkers board for a checkers game. It did not require the addition of the actual game play mechanics, but it did require the development of an interactive highlight feature. Whenever a player moused over a piece the background becomes yellow. I was able to get all these features working properly. The key takes aways from this program for me was learning from past mistakes. In Sokoban I did not make the background static which caused me problems down the road. In this program I created the checkerboard/background in the constructor. After this I did not have to worry about losing visuals when the dynamic elements were introduced, since everything would be layered over this background. In the constructor I also loaded all the textures into a texture into a map. This allowed me to access them at my convenience at anytime in my other functions. In my constructor I loaded the pieces into my piece matrix. Which in my head was a separate entity. In the draw function the checkerboard and matrix would be brought together to generate the game time board. It was here that I made an interesting design choice. In the class Checkers there existed a matrix that contained an object from the piece class. It was these pieces that contained all the information the game required for movement and visuals. I treated my board as a matrix filled with light bulbs, where a light is either on or off, light is visible, or it is not visible. In the same vein the entire matrix was filled with pieces but somewhere *on* while somewhere *off.* This was so in future assignments when I needed to move a piece, I didn’t actually move anything I simply turned one piece off and another on. This also played into the selection feature of a piece. A piece is either selected or it is not. When it was turned on it communicated to the draw function that it needed to draw a highlight before the piece. Since this is all contained to the pieces and by extension the matrix they exist in, it is impossible for the user to alter the integrity of the background. I feel that my development of this program grew Sokoban and better channeled Object-Oriented Programing principles. I did run into a new problem. The problem was a result of left over input from the mouse causing extra input to affect the program, this was resolved by creating a clearMouseBuffer function. A simple function that loops while the program picks up extra input data.

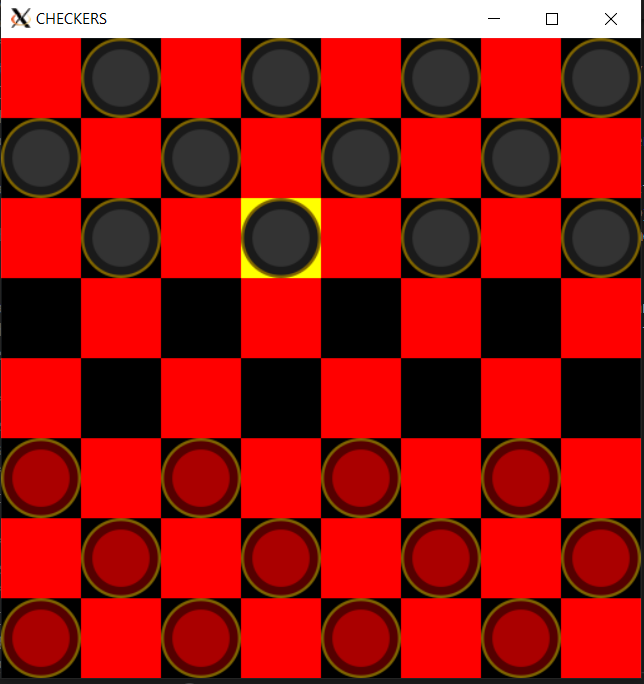
## Key Elements Discussion

There are two elements that were central to this assignment, the first being the vector. I used a vector to create a matrix in which I placed and stored store game pieces which were crucial to this program. Pieces are a class used to give each piece characteristics that allow it to interact with the checkers in main as its own little interphase, that builds into the entire checkers’ game. This program helped me to continue to develop my understanding of object-oriented programs. And, while I did not have to implement any complicated algorithms for this project, it did give me the opportunity to build off my failures in the last assignment.

## What I Learned

In this program I learned how to go out of my way to make my program more robust for my future self. Given the deadlines for this assignment I couldn’t make all the necessary improvements I wanted, so I learned from this how to prioritize the most important components.

## Issues

Despite the growth I felt in this assignment, there were a couple unresolved issues. The first was a segmentation fault that occurred whenever the window is resized. This happens since none of my calculations in the code account for a window resize. However, since this does not affect the game play and the window starts off in the correct size, I did not patch it. The other issue was that my scaling for the checkerboard was slightly off, so it was a little larger than the window. This issue was so small that it is almost unnoticeable, which is how it escaped my gaze.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = main.o Checkers.o
4. LIBS = -lsfml-window -lsfml-graphics -lsfml-system
5. DEPS = Checkers.hpp
6. all: Checkers\_driver lint
7. Checkers\_driver: $(OBJ)
8. $(CC) $(CFLAGS) -o Checkers\_driver $(OBJ) $(LIBS)
9. lint:
10. cpplint \*.cpp Headers/Checkers.hpp
11. %.o: %.cpp $(DEPS)
12. $(CC) $(CFLAGS) -c $<
13. clean:
14. rm -f $(OBJ) Checkers\_driver lint

### main.cpp

1. /\* "Copyright [2023] <Daniel Bergeron>" \*/
2. #include "Headers/Checkers.hpp"
3. void clearBuffer(void);
4. int main(int argc, char\* argv[]) {
5. sf::RenderWindow window1(sf::VideoMode(GAMEBOARDLENGTH \* SPACESIZES,
6. GAMEBOARDLENGTH \* SPACESIZES), "CHECKERS");
7. Checkers game1;
8. while (window1.isOpen()) {
9. sf::Event event;
10. while (window1.pollEvent(event)) {
11. if (event.type == sf::Event::Closed) {
12. clearBuffer();
13. window1.close();
14. }
15. }
16. if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
17. // left mouse button is pressed:
18. game1.highLight(sf::Mouse::getPosition(window1));
19. }
20. window1.clear(sf::Color::Black);
21. window1.draw(game1);
22. window1.display();
23. }
24. return 0;
25. }
26. // removes junk input from buffer so next run doesn't
27. // segment fault
28. void clearBuffer(void) {
29. while (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {}
30. }

### Checkers.hpp

1. #ifndef \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS4A\_HEADERS\_CHECKERS\_HPP\_
2. #define \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS4A\_HEADERS\_CHECKERS\_HPP\_
3. // Copyright [2023] <Daniel Bergeron>
4. // ptree class that derives from sf::Drawable
5. #include <iostream>
6. using std::cout;
7. using std::endl;
8. #include <utility>
9. using std::pair;
10. #include <string>
11. using std::string;
12. using std::stoi;
13. using std::stod;
14. #include <vector>
15. using std::vector;
16. #include <map>
17. using std::map;
18. #include <SFML/System.hpp>
19. #include <SFML/Window.hpp>
20. #include <SFML/Graphics.hpp>
21. // constants for conversions and intial set up
22. const int GAMEBOARDLENGTH = 8;
23. const int SPACESIZES = 64;
24. const int UPPERBOARDLIMIT = 2;
25. const int LOWERBOARDLIMIT = 5;
26. // type for pieces
27. enum pieceType {red, black, redKing, blackKing};
28. class Piece {
29. public:
30. // constructors
31. Piece() : isKing(false), isHighLighted(false), type(red),
32. isPresent(false) {}
33. Piece(bool \_isKing, bool \_isHighLighted, bool \_isPresent, pieceType \_type,
34. sf::Sprite \_sprite, sf::RectangleShape \_highlight);
35. // accessors
36. inline bool getIsPresent(void) {return isPresent; }
37. inline bool getIsKing(void) {return isKing; }
38. inline bool getIsHighLighted(void) {return isHighLighted; }
39. inline pieceType getpieceType(void) {return type; }
40. inline sf::Sprite getSprite(void) {return sprite; }
41. inline sf::RectangleShape getHighLighted(void) {return highlight; }
42. inline sf::Vector2f getPosition(void) {return position; }
43. // muttators
44. inline void setIsKing(bool \_isKing) {isKing = \_isKing; }
45. inline void setIsHighLighted(bool \_isSelected)
46. {isHighLighted = \_isSelected; }
47. inline void setType(pieceType \_type) {type = \_type; }
48. inline void setSprite(sf::Sprite \_sprite) {sprite = \_sprite; }
49. inline void setHighLight(sf::RectangleShape \_selected)
50. {highlight = \_selected; }
51. inline void setIsPresent(bool \_isPresent) {isPresent = \_isPresent; }
52. inline void setPosition(sf::Vector2f \_position) {position = \_position; }
53. private:
54. // bool vars tell if specifc piece attributes are ture or not
55. bool isKing;
56. bool isHighLighted;
57. bool isPresent;
58. // other mem vars
59. pieceType type;
60. sf::Sprite sprite;
61. sf::RectangleShape highlight;
62. sf::Vector2f position;
63. };
64. class Checkers : public sf::Drawable {
65. public:
66. // constructor
67. Checkers();
68. // tells if checkers class to enable a pieces highlight attribute
69. void highLight(sf::Vector2i localPosition);
70. // accessors
71. private:
72. // helper & private funcs
73. virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
74. // load intial elements
75. void loadTextureMap(void);
76. void loadPieceMatrix(void);
77. void loadBackground(void);
78. // member vars
79. sf::Vector2f tileSize;
80. sf::Vector2f currHighLight;
81. sf::RectangleShape background[GAMEBOARDLENGTH][GAMEBOARDLENGTH];
82. vector<vector<Piece>> pieceMatrix;
83. map<pieceType, sf::Texture> textureMap;
84. };
85. #endif  // \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS4A\_HEADERS\_CHECKERS\_HPP\_

### Checkers.cpp

1. // "Copyright [2023] Daniel Bergeron"
2. #include "Headers/Checkers.hpp"
3. // consturctors
4. Piece::Piece(bool \_isKing, bool \_isHighLighted, bool \_isPresent,
5. pieceType \_type, sf::Sprite \_sprite, sf::RectangleShape \_highlight) {
6. isKing = \_isKing;
7. isHighLighted = \_isHighLighted;
8. type = \_type;
9. sprite = \_sprite;
10. highlight = \_highlight;
11. isPresent = \_isPresent;
12. }
13. Checkers::Checkers() {
14. tileSize.x = static\_cast<float>(SPACESIZES);
15. tileSize.y = tileSize.x;
16. currHighLight.x = -1;
17. currHighLight.y = -1;
18. // load background()
19. loadBackground();
20. // loadstextures into map
21. loadTextureMap();
22. // adds pieces to matrix
23. loadPieceMatrix();
24. }
25. // helper functions for constructor
26. // loads static tile background
27. void Checkers::loadBackground(void) {
28. sf::RectangleShape tile;
29. pieceType previousColor = black;
30. tile.setSize(tileSize);
31. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
32. // gives checker offsett for each row
33. if (i % 2 != 0) {
34. previousColor = red;
35. } else {
36. previousColor = black;
37. }
38. // gives checker offsett for each cloumn
39. for (int j = 0; j < GAMEBOARDLENGTH; j++) {
40. if (previousColor == black) {
41. tile.setFillColor(sf::Color::Red);
42. previousColor = red;
43. } else if (previousColor == red) {
44. tile.setFillColor(sf::Color::Black);
45. previousColor = black;
46. }
47. tile.setPosition(tileSize.x \* i, tileSize.x \* j);
48. background[i][j] = tile;
49. }
50. }
51. }
52. // loads the piece matirx into checkers
53. // this matrix is dynamic
54. void Checkers::loadPieceMatrix(void) {
55. // bool \_isKing, bool \_isHighLighted, bool \_isPresent, pieceType \_type,
56. // sf::Sprite \_sprite, sf::RectangleShape \_highlight)
57. // vector<vector<Piece>> pieceMatrix;
58. sf::Sprite sprite;
59. sf::RectangleShape highlight;
60. sf::Vector2f mapPos;
61. int offset = 0;
62. highlight.setSize(tileSize);
63. highlight.setFillColor(sf::Color::Yellow);
64. sprite.setTexture(textureMap[black]);
65. Piece pawn(false, false, true, black, sprite, highlight);
66. // fill y axis
67. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
68. vector<Piece> pRow;
69. pieceMatrix.push\_back(pRow);
70. }
71. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
72. offset = i;
73. for (int j = 0; j < GAMEBOARDLENGTH; j++) {
74. mapPos.x = i;
75. mapPos.y = j;
76. pawn.setPosition(mapPos);
77. if (j > UPPERBOARDLIMIT) {
78. pawn.setType(red);
79. sprite.setTexture(textureMap[red]);
80. pawn.setSprite(sprite);
81. } else {
82. pawn.setType(black);
83. sprite.setTexture(textureMap[black]);
84. pawn.setSprite(sprite);
85. }
86. if (j > UPPERBOARDLIMIT && j < LOWERBOARDLIMIT) {
87. pawn.setIsPresent(false);
88. } else {
89. (offset % 2 == 0) ? pawn.setIsPresent(false)
90. :pawn.setIsPresent(true);
91. }
92. pieceMatrix[i].push\_back(pawn);
93. offset++;
94. }
95. }
96. }
97. // loads a texture map for sprites to
98. // acces since they having a lot of textures
99. // is heavy
100. void Checkers::loadTextureMap(void) {
101. sf::Texture texture;
102. texture.loadFromFile("checkers/blackking.png");
103. textureMap[blackKing] = texture;
104. texture.loadFromFile("checkers/blackpawn.png");
105. textureMap[black] = texture;
106. texture.loadFromFile("checkers/redking.png");
107. textureMap[redKing] = texture;
108. texture.loadFromFile("checkers/redpawn.png");
109. textureMap[red] = texture;
110. }
111. // draw function is a overrides of sfml drawable
112. void Checkers::draw(sf::RenderTarget &target, sf::RenderStates states) const {
113. Piece piece;
114. sf::Vector2f posOffsett;
115. sf::Sprite spritePiece;
116. sf::RectangleShape highLight;
117. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
118. for (int j = 0; j < GAMEBOARDLENGTH; j++) {
119. target.draw(background[i][j], states);
120. piece = pieceMatrix[i][j];
121. if (piece.getIsPresent() == true) {
122. posOffsett = piece.getPosition();
123. posOffsett.x = posOffsett.x \* SPACESIZES;
124. posOffsett.y = posOffsett.y \* SPACESIZES;
125. if (piece.getIsHighLighted() == true) {
126. highLight = piece.getHighLighted();
127. highLight.setPosition(posOffsett);
128. target.draw(highLight, states);
129. }
130. spritePiece = piece.getSprite();
131. spritePiece.setPosition(posOffsett);
132. target.draw(spritePiece, states);
133. }
134. }
135. }
136. return;
137. }
138. // is passed a moses position in pixel space
139. // highlights the piece in tile space
140. void Checkers::highLight(sf::Vector2i localPosition) {
141. // highlight current selection
142. Piece selected = pieceMatrix[localPosition.x / SPACESIZES]
143. [localPosition.y / SPACESIZES];
144. if (selected.getIsPresent() == true) {
145. selected.setIsHighLighted(true);
146. }
147. pieceMatrix[localPosition.x / SPACESIZES]
148. [localPosition.y / SPACESIZES] = selected;
149. // remove previous selection
150. if (currHighLight == selected.getPosition()) {
151. return;
152. }
153. if (currHighLight.x >= 0 && currHighLight.y >= 0) {
154. pieceMatrix[currHighLight.x][currHighLight.y].setIsHighLighted(false);
155. }
156. currHighLight.x = selected.getPosition().x;
157. currHighLight.y = selected.getPosition().y;
158. return;
159. }

# PS4B – Checkers

## Program Discussion

The goal of this assignment was to build of the previous assignments’ implementation of a checkers board and add the mechanics. I was able to bring the pieces to life, giving them movement and adding the rules that the movement had to adhere to. I also added a win condition for each side. This program took the interphase I developed in the first assignment and utilized it to great effect. Compared to the previous assignment, there were a lot of difficult decisions and issues I needed to overcome. At this point I came across another object-oriented programing design mistake. While every piece in the game had the power to move, the movement depended on the type, of which there were two. King pieces and pawn pieces. It was at this point that I should have modified my old code and created two new classes to interphase with. The first would be the pawn class and the second would be the king class. Both classes should have been inherited from the piece class. Which would have been abstract. With this I could have had each child’s class manage the rules of movement for the pieces rather than the functions I created. If I took this route, it would have wider implications throughout my program, and would have required a complete rework of the entire program structure. Given that the deadline for these assignments is quite strict I decided to stick with what I had. This created more complicated functions, but in the end, it was able to accomplish its tasks. I created a playable checkers game.

## Key Elements Discussion

For data structures I used maps & vectors. I used vectors because of their built-in interphase along with its ability to make a marix and model a 2-d surface in a program. Maps where use full for encoding and decoding, movement/texture information that I needed throughout the program, since the amount of data for these two cases was less important than the type of data, the maps’ ability to use key allowed me to have vital info on hand. OOP in the checkers class I created a supported a piece class that could interphase with the checkers’ class. This idea of interfacing was imperative for this project’s success. For an algorithm & lambda I used a for\_each range function which allowed me to apply a lambda function that calculated how many pieces were left on the board. Using the lambda here allowed me to remove a nested loop and make the program more readable. I addition to these tools I had to develop my own logical structure on how the code should run. So, I came up with some important logic.

In this program as stated before I have a checkers class, this class is responsible for visuals along with enforcing game rules on those visuals. It accomplishes this with three main functions an overridden draw function from sf::drawable, this function is solely responsible for drawing everything. No other function is allowed to make anything visible to the users without going through the draw function. There is a move function which breaks the problem of movement down into several important aspects, first can we move, if so what type of move can we do, then if possible execute the move, if move succeeded, then we no longer have a selected piece and finally now that we moved who gets to move next; (in hindsight the turns could have been separate function, a lot I would change with more time). Along with a couple accessor functions the next important function is the endgame check, this function which is responsible for checking if the game can move to an endgame state. The Checkers class then interphases with the piece class, which contains information on all the attributes of a piece object. As stated in the previous discussion it is here, I would have broken up the class. However, the piece class develops a strong enough interphase that allows checkers to use its methods to build a sturdy game.

## What I Learned

I learned in this program the importance of adhering to the design principles of the paradigm. This code would have been much shorter and more robust if had adhered more to Object-Oriented design principles. However, this program was a significant step in the right direction compared to previous programs.

## Issues

This program retains the same issues as the previous program, yet it did not cause any issues in this implementation. The main issue faced in this program had to do with the input from the mouse. Input buffering was difficult to manage, causing segmentation faults to occur as well as selection delays. The segmentation faults were caused by the same issue in the previous assignment. Yet it became slightly more relevant in this assignment because the players are selecting and moving pieces frequently which may accidentally cause them to touch the window edge slightly resizing it. This overtime with many moves creates a banned around the game board of bad positional data. If a player accidentally selects in this area, it causes the program to have a segmentation fault.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = main.o Checkers.o
4. LIBS = -lsfml-window -lsfml-graphics -lsfml-system
5. DEPS = Checkers.hpp
6. all: Checkers lint
7. Checkers: $(OBJ)
8. $(CC) $(CFLAGS) -o Checkers $(OBJ) $(LIBS)
9. lint:
10. cpplint \*.cpp Headers/Checkers.hpp
11. %.o: %.cpp $(DEPS)
12. $(CC) $(CFLAGS) -c $<
13. clean:
14. rm -f $(OBJ) Checkers lint

### main.cpp

1. /\* "Copyright [2023] <Daniel Bergeron>" \*/
2. #include "Headers/Checkers.hpp"
3. void clearBuffer(void);
4. int main(int argc, char\* argv[]) {
5. sf::RenderWindow window1(sf::VideoMode(GAMEBOARDLENGTH \* SPACESIZES,
6. GAMEBOARDLENGTH \* SPACESIZES), "CHECKERS");
7. Checkers game1;
8. sf::Vector2i location;
9. // fonts
10. sf::Text text, text2;
11. sf::Font font;
12. if (!font.loadFromFile("resources/endGameFont.ttf")) {
13. cout << "Font failed to load\n";
14. }
15. text.setFont(font);
16. text.setString("Game Over!!");
17. text.setCharacterSize(100);
18. text.setPosition((static\_cast<float>(window1.getSize().x / 5.00)),
19. (static\_cast<float>(window1.getSize().y / 4.00)));
20. text.setFillColor(sf::Color::White);
21. text.setStyle(sf::Text::Bold | sf::Text::Underlined);
22. text2.setFont(font);
23. text2.setString("Press Q to quit");
24. text2.setCharacterSize(30);
25. text2.setFillColor(sf::Color::Blue);
26. text2.setStyle(sf::Text::Bold | sf::Text::Underlined);
27. clearBuffer();
28. while (window1.isOpen()) {
29. sf::Event event;
30. while (window1.pollEvent(event)) {
31. if (event.type == sf::Event::Closed) {
32. clearBuffer();
33. window1.close();
34. }
35. }
36. // tells if its a move or not based on if present
37. if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
38. // left mouse button is pressed:
39. game1.highLight(sf::Mouse::getPosition(window1));
40. location = sf::Mouse::getPosition(window1);
41. location.x /= SPACESIZES;
42. location.y /= SPACESIZES;
43. game1.move(location);
44. }
45. while (game1.endGameCheck()) {
46. // inside the main loop, between window.clear()
47. // and window.display()
48. while (window1.pollEvent(event)) {
49. if (event.type == sf::Event::Closed) {
50. clearBuffer();
51. window1.close();
52. return 0;
53. }
54. }
55. if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {
56. return 0;
57. }
58. window1.clear(sf::Color::Black);
59. window1.draw(game1);
60. if (game1.getBlackPieceCount() == 0) {
61. text.setString("RED WINS!!!");
62. } else if (game1.getRedPieceCount() == 0) {
63. text.setString("Black WINS!!!");
64. } else {
65. // For possible edge case
66. text.setString("Draw!");
67. }
68. window1.draw(text);
69. window1.draw(text2);
70. window1.display();
71. }
72. // take selected input and see if next position clicked position is good
73. // then move
74. window1.clear(sf::Color::Black);
75. window1.draw(game1);
76. window1.display();
77. }
78. // clearBuffer();
79. return 0;
80. }
81. // removes junk input from buffer so next run doesn't
82. // segment fault
83. void clearBuffer(void) {
84. while (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {}
85. while (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {}
86. }

### Checkers.hpp

1. #ifndef \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS4B\_HEADERS\_CHECKERS\_HPP\_
2. #define \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS4B\_HEADERS\_CHECKERS\_HPP\_
3. // Copyright [2023] <Daniel Bergeron>
4. // ptree class that derives from sf::Drawable
5. #include <ranges>
6. #include <cmath>
7. using std::abs;
8. #include <iostream>
9. using std::cout;
10. using std::endl;
11. #include <utility>
12. using std::pair;
13. #include <algorithm>
14. #include <string>
15. using std::string;
16. using std::stoi;
17. using std::stod;
18. #include <vector>
19. using std::vector;
20. #include <map>
21. using std::map;
22. #include <SFML/System.hpp>
23. #include <SFML/Window.hpp>
24. #include <SFML/Graphics.hpp>
25. // constants for conversions and intial set up
26. const int GAMEBOARDLENGTH = 8;
27. const int SPACESIZES = 64;
28. const int UPPERBOARDLIMIT = 2;
29. const int LOWERBOARDLIMIT = 5;
30. // type for pieces
31. enum pieceType {empty, red, redKing, black, blackKing};
32. enum moveSet { nil, upLeft = 1, upLeftJump = -1, upRight = 2,
33. upRightJump = -2, downRight = 3, downRightJump = -3,
34. downLeft = 4, downLeftJump = -4};
35. class Piece {
36. public:
37. // constructors
38. Piece() : isKing(false), isHighLighted(false), type(red),
39. isPresent(false) {}
40. Piece(bool \_isKing, bool \_isHighLighted, bool \_isPresent, pieceType \_type,
41. sf::Sprite \_sprite, sf::RectangleShape \_highlight);
42. // accessors
43. inline bool getIsPresent(void) {return isPresent; }
44. inline bool getIsKing(void) {return isKing; }
45. inline bool getIsHighLighted(void) {return isHighLighted; }
46. inline pieceType getpieceType(void) {return type; }
47. inline sf::Sprite getSprite(void) {return sprite; }
48. inline sf::RectangleShape getHighLighted(void) {return highlight; }
49. inline sf::Vector2f getPosition(void) {return position; }
50. // muttators
51. inline void setIsKing(bool \_isKing) {isKing = \_isKing; }
52. inline void setIsHighLighted(bool \_isSelected)
53. {isHighLighted = \_isSelected; }
54. inline void setType(pieceType \_type) {type = \_type; }
55. inline void setSprite(sf::Sprite \_sprite) {sprite = \_sprite; }
56. inline void setHighLight(sf::RectangleShape \_selected)
57. {highlight = \_selected; }
58. inline void setIsPresent(bool \_isPresent) {isPresent = \_isPresent; }
59. inline void setPosition(sf::Vector2f \_position) {position = \_position; }
60. private:
61. // bool vars tell if specifc piece attributes are ture or not
62. bool isKing;
63. bool isHighLighted;
64. bool isPresent;
65. // other mem vars
66. pieceType type;
67. sf::Sprite sprite;
68. sf::RectangleShape highlight;
69. sf::Vector2f position;
70. };
71. class Checkers : public sf::Drawable {
72. public:
73. // constructor
74. Checkers();
75. // tells if checkers class to enable a pieces highlight attribute
76. void highLight(sf::Vector2i localPosition);
77. // move function
78. bool move(sf::Vector2i LocalPosition);
79. // turn management
80. inline void increaseTurn(void) {turn++; }
81. int getTurn(void);
82. // Tells weather or not to end the game
83. bool endGameCheck();
84. // accesors
85. inline int getRedPieceCount(void) {return redWin; }
86. inline int getBlackPieceCount(void) {return blackWin; }
87. private:
88. // helper & private funcs
89. virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
90. // move helper functions
91. moveSet getMovetype(sf::Vector2i LocalPosition);
92. // function resets the highlighted square
93. void resetHighLight(void);
94. // when given a move it is responsible
95. bool executeMove(moveSet currMove);
96. // execute helper
97. bool removePiece(moveSet destroy);
98. // function converts a normal pawn to king
99. // Needs the location and the type to do it
100. void makeKing(pieceType currType, sf::Vector2i LocalPosition);
101. // load intial elements
102. void loadTextureMap(void);
103. void loadPieceMatrix(void);
104. void loadBackground(void);
105. void loadMoveMap(void);
106. // member vars
107. int turn = 0;
108. int redWin = 0;
109. int blackWin = 0;
110. Piece currPiece;
111. sf::Vector2f tileSize;
112. sf::Vector2f currHighLight;
113. sf::RectangleShape background[GAMEBOARDLENGTH][GAMEBOARDLENGTH];
114. vector<vector<Piece>> pieceMatrix;
115. map<pieceType, sf::Texture> textureMap;
116. map<moveSet, sf::Vector2i> moveMap;
117. };
118. #endif  // \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS4B\_HEADERS\_CHECKERS\_HPP\_

### Checkers.cpp

1. // "Copyright [2023] Daniel Bergeron"
2. #include "Headers/Checkers.hpp"
3. // consturctors
4. Piece::Piece(bool \_isKing, bool \_isHighLighted, bool \_isPresent,
5. pieceType \_type, sf::Sprite \_sprite, sf::RectangleShape \_highlight) {
6. isKing = \_isKing;
7. isHighLighted = \_isHighLighted;
8. type = \_type;
9. sprite = \_sprite;
10. highlight = \_highlight;
11. isPresent = \_isPresent;
12. }
13. Checkers::Checkers() {
14. tileSize.x = static\_cast<float>(SPACESIZES);
15. tileSize.y = tileSize.x;
16. currHighLight.x = -1;
17. currHighLight.y = -1;
18. // load background()
19. loadBackground();
20. // loadstextures into map
21. loadTextureMap();
22. // adds pieces to matrix
23. loadPieceMatrix();
24. // loads moves into map
25. loadMoveMap();
26. }
27. void Checkers::loadMoveMap(void) {
28. // enum moveSet {nil, upLeft, upRight, downLeft, downRight, upLeftJump,
29. // upRightJump, downRightJump, downLeftJump };
30. sf::Vector2i moveVal;
31. moveVal.x = 0; moveVal.y = 0;
32. moveMap[nil] = moveVal;
33. moveVal.x = -1; moveVal.y = -1;
34. moveMap[upLeft] = moveVal;
35. moveVal.x = 1; moveVal.y = -1;
36. moveMap[upRight] = moveVal;
37. moveVal.x = 1; moveVal.y = 1;
38. moveMap[downLeft] = moveVal;
39. moveVal.x = -1; moveVal.y = 1;
40. moveMap[downRight] = moveVal;
41. moveVal.x = -2; moveVal.y = -2;
42. moveMap[upLeftJump] = moveVal;
43. moveVal.x = 2; moveVal.y = -2;
44. moveMap[upRightJump] = moveVal;
45. moveVal.x = -2; moveVal.y = 2;
46. moveMap[downRightJump] = moveVal;
47. moveVal.x = 2; moveVal.y = 2;
48. moveMap[downLeftJump] = moveVal;
49. return;
50. }
51. // helper functions for constructor
52. // loads static tile background
53. void Checkers::loadBackground(void) {
54. sf::RectangleShape tile;
55. pieceType previousColor = black;
56. tile.setSize(tileSize);
57. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
58. // gives checker offsett for each row
59. if (i % 2 != 0) {
60. previousColor = red;
61. } else {
62. previousColor = black;
63. }
64. // gives checker offsett for each cloumn
65. for (int j = 0; j < GAMEBOARDLENGTH; j++) {
66. if (previousColor == black) {
67. tile.setFillColor(sf::Color::Red);
68. previousColor = red;
69. } else if (previousColor == red) {
70. tile.setFillColor(sf::Color::Black);
71. previousColor = black;
72. }
73. tile.setPosition(tileSize.x \* i, tileSize.x \* j);
74. background[i][j] = tile;
75. }
76. }
77. }
78. // loads the piece matirx into checkers
79. // this matrix is dynamic
80. void Checkers::loadPieceMatrix(void) {
81. // bool \_isKing, bool \_isHighLighted, bool \_isPresent, pieceType \_type,
82. // sf::Sprite \_sprite, sf::RectangleShape \_highlight)
83. // vector<vector<Piece>> pieceMatrix;
84. sf::Sprite sprite;
85. sf::RectangleShape highlight;
86. sf::Vector2f mapPos;
87. int offset = 0;
88. highlight.setSize(tileSize);
89. highlight.setFillColor(sf::Color::Yellow);
90. sprite.setTexture(textureMap[black]);
91. Piece pawn(false, false, true, black, sprite, highlight);
92. // fill y axis
93. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
94. vector<Piece> pRow;
95. pieceMatrix.push\_back(pRow);
96. }
97. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
98. offset = i;
99. for (int j = 0; j < GAMEBOARDLENGTH; j++) {
100. mapPos.x = i;
101. mapPos.y = j;
102. pawn.setPosition(mapPos);
103. if (j > UPPERBOARDLIMIT) {
104. pawn.setType(red);
105. sprite.setTexture(textureMap[red]);
106. pawn.setSprite(sprite);
107. } else {
108. pawn.setType(black);
109. sprite.setTexture(textureMap[black]);
110. pawn.setSprite(sprite);
111. }
112. if (j > UPPERBOARDLIMIT && j < LOWERBOARDLIMIT) {
113. pawn.setType(empty);
114. pawn.setIsPresent(false);
115. } else {
116. (offset % 2 == 0) ? pawn.setIsPresent(false)
117. :pawn.setIsPresent(true);
118. }
119. pieceMatrix[i].push\_back(pawn);
120. offset++;
121. }
122. }
123. }
124. // loads a texture map for sprites to
125. // acces since they having a lot of textures
126. // is heavy
127. void Checkers::loadTextureMap(void) {
128. sf::Texture texture;
129. texture.loadFromFile("resources/blackking.png");
130. textureMap[blackKing] = texture;
131. texture.loadFromFile("resources/blackpawn.png");
132. textureMap[black] = texture;
133. texture.loadFromFile("resources/redking.png");
134. textureMap[redKing] = texture;
135. texture.loadFromFile("resources/redpawn.png");
136. textureMap[red] = texture;
137. }
138. // draw function is a overrides of sfml drawable
139. void Checkers::draw(sf::RenderTarget &target, sf::RenderStates states) const {
140. Piece piece;
141. sf::Vector2f posOffsett;
142. sf::Sprite spritePiece;
143. sf::RectangleShape highLight;
144. for (int i = 0; i < GAMEBOARDLENGTH; i++) {
145. for (int j = 0; j < GAMEBOARDLENGTH; j++) {
146. target.draw(background[i][j], states);
147. piece = pieceMatrix[i][j];
148. if (piece.getIsPresent() == true) {
149. posOffsett = piece.getPosition();
150. posOffsett.x = posOffsett.x \* SPACESIZES;
151. posOffsett.y = posOffsett.y \* SPACESIZES;
152. if (piece.getIsHighLighted() == true) {
153. highLight = piece.getHighLighted();
154. highLight.setPosition(posOffsett);
155. target.draw(highLight, states);
156. }
157. spritePiece = piece.getSprite();
158. spritePiece.setPosition(posOffsett);
159. target.draw(spritePiece, states);
160. }
161. }
162. }
163. return;
164. }
165. // is passed a moses position in pixel space
166. // highlights the piece in tile space
167. void Checkers::highLight(sf::Vector2i localPosition) {
168. // highlight current selection
169. int colorTurn = getTurn();
170. Piece selected = pieceMatrix[localPosition.x / SPACESIZES]
171. [localPosition.y / SPACESIZES];
172. // if piece exsits highlight else do nothing
173. if (selected.getIsPresent() == true) {
174. if (colorTurn == 0 && selected.getpieceType() >= black) {
175. return;
176. } else if (colorTurn == 1 && selected.getpieceType() < black) {
177. return;
178. }
179. } else {
180. return;
181. }
182. selected.setIsHighLighted(true);
183. pieceMatrix[localPosition.x / SPACESIZES]
184. [localPosition.y / SPACESIZES] = selected;
185. // remove previous selection
186. if (currHighLight == selected.getPosition()) {
187. currPiece = selected;
188. return;
189. }
190. if (currHighLight.x >= 0 && currHighLight.y >= 0) {
191. pieceMatrix[currHighLight.x][currHighLight.y].setIsHighLighted(false);
192. }
193. currHighLight.x = selected.getPosition().x;
194. currHighLight.y = selected.getPosition().y;
195. currPiece = selected;
196. return;
197. }
198. // is already converted to tile space
199. bool Checkers::move(sf::Vector2i LocalPosition) {
200. // first check if piece at location is present
201. // or if it is the selected piece
202. Piece newPiece = pieceMatrix[LocalPosition.x][LocalPosition.y];
203. if (newPiece.getIsPresent() == true
204. || newPiece.getIsHighLighted() == true) {
205. return false;
206. }
207. // call function to seee if local position matches anything in moveset
208. // if so return move to use
209. moveSet currMove;
210. currMove = getMovetype(LocalPosition);
211. if (currMove == nil) {
212. return false;
213. }
214. // check if king
215. // if king all moves from here are valid
216. // call a function to execute move if
217. // if king condition fails move based on color
218. // get piece color & branch based off piece color
219. // red only moves up and black down
220. // inside color if's
221. pieceType currType = currPiece.getpieceType();
222. if (currType == red) {
223. if (currMove == downLeft || currMove == downRight
224. || currMove == downLeftJump || currMove == downRightJump) {
225. return false;
226. }
227. } else if (currType == black) {
228. if (currMove == upLeft || currMove == upRight
229. || currMove == upLeftJump || currMove == upRightJump) {
230. return false;
231. }
232. }
233. resetHighLight();
234. // if execute happens piece is move into Local position on matrix
235. // if executed then new turn
236. if (executeMove(currMove)) {
237. increaseTurn();
238. }
239. // after move, check if piece is in the bottom(black)
240. // or top(red) if so call king function
241. if ((currType == black && LocalPosition.y == 7)
242. || (currType == red && LocalPosition.y == 0)) {
243. makeKing(currType, LocalPosition);
244. }
245. // return true if move worked
246. return true;
247. }
248. // goal of function is to validate the seleceted space
249. // need to figure out where curr piece is in regards to newSpace
250. moveSet Checkers::getMovetype(sf::Vector2i LPosition) {
251. Piece emptySpace;
252. moveSet move = nil;
253. sf::Vector2i offset;
254. offset.x = currPiece.getPosition().x;
255. offset.y = currPiece.getPosition().y;
256. // 1) use map to store what each move does mathamtically
257. // 2) using the curr pieces positon use the pair retieved from
258. // map add x & y together if the the sums agree return move type
259. map<moveSet, sf::Vector2i>::iterator m;
260. for (m = moveMap.begin(); m != moveMap.end(); ++m) {
261. offset.x += m->second.x;
262. offset.y += m->second.y;
263. move = m->first;
264. if (offset == LPosition) {
265. return move;
266. } else {
267. offset.x = currPiece.getPosition().x;
268. offset.y = currPiece.getPosition().y;
269. move = nil;
270. }
271. }
272. return move;
273. }
274. // resest highlight for next draw cycle
275. void Checkers::resetHighLight(void) {
276. Piece selected = pieceMatrix[currHighLight.x][currHighLight.y];
277. selected.setIsHighLighted(false);
278. pieceMatrix[currHighLight.x][currHighLight.y] = selected;
279. return;
280. }
281. bool Checkers::executeMove(moveSet currMove) {
282. // if move is jump remove middle piece
283. // jumps are negative since piece will be taken
284. // so go to a normal move space in that direction
285. // and check if piece is present if not move fails
286. bool failedJumped;
287. if (currMove < 0) {
288. failedJumped = removePiece(static\_cast<moveSet>(abs(currMove)));
289. if (failedJumped) {
290. cout << "jump failed\n";
291. return false;
292. }
293. }
294. Piece newPiece;
295. // set new position
296. sf::Vector2i pos = moveMap[currMove];
297. pos.x += currPiece.getPosition().x;
298. pos.y += currPiece.getPosition().y;
299. newPiece = currPiece;
300. newPiece.setPosition(static\_cast<sf::Vector2f>(pos));
301. newPiece.setIsHighLighted(false);
302. // enable new piece with same attributes as currpiece
303. pieceMatrix[pos.x][pos.y] = newPiece;
304. // disable currPiece
305. currPiece.setIsPresent(false);
306. currPiece.setType(empty);
307. currPiece.setIsHighLighted(false);
308. pieceMatrix[currPiece.getPosition().x][currPiece.getPosition().y]
309. = currPiece;
310. return true;
311. }
312. void Checkers::makeKing(pieceType currType, sf::Vector2i pos) {
313. sf::Sprite sprite;
314. Piece king =  pieceMatrix[pos.x][pos.y];
315. int x = currType;
316. x++;
317. currType = static\_cast<pieceType>(x);
318. king.setType(currType);
319. king.setIsKing(true);
320. sprite.setTexture(textureMap[currType]);
321. king.setSprite(sprite);
322. pieceMatrix[pos.x][pos.y] = king;
323. }
324. bool Checkers::removePiece(moveSet destroyMove) {
325. // need moving pieces to test come back to later
326. sf::Vector2i pos = moveMap[destroyMove];
327. pos.x += currPiece.getPosition().x;
328. pos.y += currPiece.getPosition().y;
329. if ( pieceMatrix[pos.x][pos.y].getIsPresent() == false ) {
330. return true;
331. }
332. // checks to see if they are jumping their own pawn
333. pieceType currType = currPiece.getpieceType();
334. if (currPiece.getIsKing()) {
335. if ( pieceMatrix[pos.x][pos.y].getpieceType() == currType
336. || pieceMatrix[pos.x][pos.y].getpieceType() == currType - 1) {
337. return true;
338. }
339. } else {
340. if ( pieceMatrix[pos.x][pos.y].getpieceType() == currType
341. || pieceMatrix[pos.x][pos.y].getpieceType() == currType + 1) {
342. return true;
343. }
344. }
345. pieceMatrix[pos.x][pos.y].setIsPresent(false);
346. pieceMatrix[pos.x][pos.y].setType(empty);
347. return false;
348. }
349. int Checkers::getTurn(void) {
350. return turn % 2;
351. }
352. // returns last color standing
353. bool Checkers::endGameCheck(void) {
354. redWin = 0;
355. blackWin = 0;
356. vector<vector<Piece>>::iterator pieceVec = pieceMatrix.begin();
357. vector<Piece>::iterator p;
358. auto presentCheck = [&](Piece guy){
359. if (guy.getIsPresent() == true) {
360. if (guy.getpieceType() == red
361. || guy.getpieceType() == redKing) {
362. redWin++;
363. } else if (guy.getpieceType() == black
364. || guy.getpieceType() == blackKing) {
365. blackWin++;
366. }
367. }
368. };
369. for ( ; pieceVec != pieceMatrix.end(); pieceVec++) {
370. std::for\_each(pieceVec->begin(), pieceVec->end(), presentCheck);
371. }
372. if (redWin == 0 || blackWin == 0) {
373. return true;
374. } else {
375. return false;
376. }
377. }

# PS5 – DNA Sequence Alignment

## Program Discussion

In this program, we learned about a programing paradigm called, dynamic programming. The goal was computing the optimal sequence alignment of two DNA strings. This program allows the user to figure out how closely related two DNA sequences are. I was able to accomplish this by building something called a cost matrix. Which uses a cost system to determine the optimal way through the matrix. To align the DNA, you can insert a gap which will add 2 to the cost, you can mismatch the DNA which will incur a cost of 1 or you have a match that incurs a cost of zero. These costs are laid out on a matrix, starting from the ends of the string to the top, or the bottom right corner of the matrix to the top left. Once the matrix is filled it can be traversed from the top left to the bottom right to find the optimal alignment. This was a difficult problem to solve and like other problems in this course it didn’t require a lot of code, but it required a lot of thinking. To accomplish this task, I had to really break down the problem. To make sure I didn’t get confused I heavily commented on the areas I didn’t understand. Most ofA picture containing graphical user interface

Description automatically generated these comments took place in the header file so I could check back in on what the code for the function was implementing. The program faced many bugs when it came to the traversing of the matrix and the rules that governed that process. The main bug in this code that took me hours to decode didn’t even come from the code but the files I inputted. Some of the files I input were not in Linux format. Instead, they were in windows format, so they had a ‘\r’ invisible character in the input. So, when I used the get line function the character was being sucked into my string. This caused the ends of my DNA sequences to be warped and off since my program was considering an incorrect character. I was able to fix this by adding a guard in main that checked for the character and popped it out of the string if it saw it. I did this rather than changing the file type so the program could take both Linux and windows files. Once this issue was solved the program performed as intended.

Terminal Output

## Key Elements Discussion

The most important element of this program is the EDistance class and how its’ methods interact with the cost matrix. The first of these key Elements is the constructor. The constructor is what builds the N x M array and initializes the values. The constructor also puts the true values in the farthest row and columns. This is because no matter the situation we know what those values are going to be. The next important method is optDistance. This is the Method uses dynamic programming to populate the matrix with cost values. The next important thing to do is to traverse the matrix based on these values to get the optimal distance value. The alignment method is what accomplishes this. The method goes through the matrix from the top left down choosing values based of three cases. Based of these cases a main string will be concatenated with a corresponding substring derived from the case itself. It’s important to note that if everything in this function is working properly it will always end in the bottom left corner or the zero condition. The complexity of this program comes from the nature of the problem, but the Object-Oriented Principles, as well as the dynamic programing I implemented here are what made this program manageable.

## What I Learned

I learned two keys’ things from this program. The first was how to break down a problem can be made much easier in programing when you use a programing paradigm tailored for the problem. The second thing I learned was that when programing it is important to always be aware of the environment you are in, as well as what environments your program pulls from.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = EDistance.o main.o
4. LIBS = -lsfml-system
5. DEPS = Header/EDistance.hpp
6. all: EDistance lint
7. EDistance: $(OBJ)
8. $(CC) $(CFLAGS) -o EDistance $(OBJ) $(LIBS)
9. lint:
10. cpplint \*.cpp Header/EDistance.hpp
11. %.o: %.cpp $(DEPS)
12. $(CC) $(CFLAGS) -c $<
13. clean:
14. rm -f $(OBJ) EDistance lint

### main.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "Header/EDistance.hpp"
3. int main(int argc, char \*argv[]) {
4. // input handling
5. std::ifstream fp;
6. fp.open(argv[1]);
7. string input1, input2;
8. getline(fp, input1);
9. if (input1.back() == '\r') {
10. input1.pop\_back();
11. }
12. getline(fp, input2);
13. // deals with windows carriage return
14. if (input2.back() == '\r') {
15. input2.pop\_back();
16. }
17. fp.close();
18. // Sequencing
19. int x = 0;
20. EDistance test(input1, input2);
21. x = test.optDistance();
22. cout << "optDistance: "<< x << endl;
23. cout << "X   Y  Cost\n";
24. cout << "\_\_\_\_\_\_\_\_\_\_\_\_\n\n";
25. cout << test.alignment();
26. return 0;
27. }

### EDistance.hpp

1. // Copyright [2023] <Daniel Bergeron>
2. #ifndef \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS5\_HEADER\_EDISTANCE\_HPP\_
3. #define \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS5\_HEADER\_EDISTANCE\_HPP\_
4. #include <fstream>
5. #include <iostream>
6. using std::cout;
7. using std::cin;
8. using std::endl;
9. #include <string>
10. using std::string;
11. #include <vector>
12. using std::vector;
13. #include <algorithm>
14. class EDistance {
15. public:
16. EDistance(const string& ip1, const string& ip2);
17. static int penalty(char a, char b);
18. static int min(int a, int b, int c);
19. int optDistance(void);
20. string alignment(void);
21. private:
22. void testPrint() const;
23. vector<vector<int>> costMatrix;
24. string x;
25. string y;
26. };
27. #endif  // \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS5\_HEADER\_EDISTANCE\_HPP\_

### EDistance.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "Header/EDistance.hpp"
3. void EDistance::testPrint() const {
4. std::cout << "===" << std::endl;
5. for (int i = 0; i < static\_cast<int>(x.size() + 1); i++) {
6. for (int j = 0; j < static\_cast<int>(y.size() + 1); j++) {
7. std::cout << " " << costMatrix[i][j] << " ";
8. }
9. std::cout << std::endl;
10. }
11. std::cout << "===" << std::endl;
12. }
13. EDistance::EDistance(const string& ip1, const string& ip2) {
14. x = ip1;  // our x side
15. y = ip2;  // our y side
16. int size1 = ip1.length();
17. int size2 = ip2.length();
18. costMatrix.resize(size1 + 1);
19. for (int i = 0; i < size1 + 1; i++) {
20. costMatrix[i].resize(size2 + 1);
21. }
22. costMatrix[size1][size2] = 0;
23. for (int i = size2 - 1; i >= 0; i--) {
24. costMatrix[size1][i] = costMatrix[size1][i + 1] + 2;
25. }
26. for (int i = size1 - 1; i >= 0; i--) {
27. costMatrix[i][size2] = costMatrix[i + 1][size2] + 2;
28. }
29. testPrint();
30. }
31. int EDistance::penalty(char a, char b) {
32. if (a == b) {
33. return 0;
34. }
35. return 1;
36. }
37. int EDistance::min(int a, int b, int c) {
38. return std::min(std::min(a, b), c);
39. }
40. int EDistance::optDistance(void) {
41. for (int i = costMatrix.size() - 2; i >= 0; i--) {
42. for (int j = costMatrix[i].size() - 2; j >= 0; j--) {
43. costMatrix[i][j] = min(costMatrix[i+1][j+1] + penalty(x[i], y[j]),
44. costMatrix[i+1][j] + 2, costMatrix[i][j+1] + 2);
45. }
46. }
47. testPrint();
48. return costMatrix[0][0];
49. }
50. string EDistance::alignment(void) {
51. auto toChar = [](int cost) {
52. return static\_cast<char>(cost + '0');
53. };
54. string oString;  // output string
55. // Resoureces
56. // need a return string
57. // cost vector
58. // function to convert num to chars
59. // main case
60. int i = 0, j = 0;
61. int minVal;
62. int cost = 0;
63. while (i <= static\_cast<int>(x.size()-1)
64. && j <= static\_cast<int>(y.size()-1)) {
65. // base case at costmatrix[0][0]
66. // load directly into string, inset x[0], y[0], then num
67. // most optimal alligment
68. // letters are matched
69. // therefore we put both letters in ouput string
70. // insert char 0
71. // push into cost matrix
72. minVal = min(costMatrix[i+1][j+1],
73. costMatrix[i+1][j], costMatrix[i][j+1]);
74. if (minVal == costMatrix[i+1][j+1]) {
75. i++;
76. j++;
77. cost = penalty(x[i], y[j]);
78. if(cost == 0 && (i > static\_cast<int>(x.size()-1)
79. || j > static\_cast<int>(x.size()-1))) {
80. return oString;
81. }
82. oString += x[i];
83. oString += "   ";
84. oString += y[j];
85. oString += "   ";
86. oString += toChar(cost);
87. oString += '\n';
88. } else if (minVal == costMatrix[i+1][j]) {
89. i++;
90. oString += x[i];
91. oString += "   ";
92. oString += '-';
93. oString += "   ";
94. oString += '2';
95. oString += '\n';
96. } else if (minVal == costMatrix[i][j + 1]) {
97. j++;
98. oString += '-';
99. oString += "   ";
100. oString += y[j];
101. oString += "   ";
102. oString += '2';
103. oString += '\n';
104. }
105. }
106. return oString;
107. }

# PS6 – Random Writer

## Program Discussion

Text

Description automatically generatedThis program was a very conceptually difficult program to deal with as well as frustrating. This program took in large input file of text along with two numbers. The two numbers K is used to indicate the size of substrings, and L used to indicate the amount of random text to be generated. Then the file was read in. Once the information was read in the program constructed a randWriter using that information. The randWriter uses K to create every possible substring of size K with matching order. So, no substrings were constructed from random points in the input it was all sequential. These substrings referred to as Kgrams were then put in a map which were mapped to another map containing all the existing K+1 instances of themselves in the input string. The K+1 instances were then mapped to their number of occurrences. A method called kRand was called to create a random character using the Kgram and return it. This value is given to the generate which calls the KRand method L times. This will generate pseudo random character. It is here an important aspect of the program must be introduced. The occurrences of each character must match that of the original string. For example, if the input contains no ‘a’s then the generator can’t make an ‘a’. If there are 50 ‘a’s to 20 ‘b’s in the original text then the generate function has to match that ratio. I was able to accomplish this by taking the occurrences values of the k+1 grams and putting them on a theoretical number line where each k+1 grams domain matched the probability of the character appearing. The reason I used the K+1 grams is because they are the only thing in the program containing any information of what characters could come next. The three components of this program for me were trying to understand the problem, since the directions at time could be confusing. Mapping the Kgrams to their K+1 self and then mapping that to number of appearances. Finally creating the probability logic was a logical heavy challenge. I was able to test these functions extensively using my test driver. This ensured that they worked and that I accomplished my task.

Terminal Output 2

## Key Elements Discussion

The three key elements of this program were the class system, its driver implementation, and the testing. The randWriter is encapsulated in its own class, this is important because it isolates two of the most difficult parts of this assignment. The rand method and the generator method. My main function needs to only call these methods and not have to worry about the complexity of the implementation. The next was the drivers I created for the project, one is the actual routine for randWriter and the other is for testing purposes. Due to the Object-Oriented nature of the program the first driver was very easy to build since it mostly just handled input and output(IO). The second driver was an isolated environment to try and break my methods. In this driver, I used the boost library to test my class object. Using boost is key because it helps you Isolate the issue in your program, by giving you good feedback.

## What I Learned

This programing assignment helped to understand testing and why it’s important. The program helped me once again in learning how to logically break down a complex problem into smaller more manageable bits.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = RandWriter.o test.o
4. OBJ2 = TextWriter.o RandWriter.o
5. LIBS = -lboost\_unit\_test\_framework
6. DEPS = Header/RandWriter.hpp
7. all: TextWriter Test lint
8. TextWriter: $(OBJ2)
9. $(CC) $(CFLAGS) -o TextWriter $(OBJ2) $(LIBS)
10. Test: $(OBJ)
11. $(CC) $(CFLAGS) -o Test $(OBJ) $(LIBS)
12. lint:
13. cpplint \*.cpp Header/RandWriter.hpp
14. %.o: %.cpp $(DEPS)
15. $(CC) $(CFLAGS) -c $<
16. clean:
17. rm -f $(OBJ) $(OBJ2) TextWriter Test lint

### textWriter.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "Header/RandWriter.hpp"
3. int main(int argc, char\* argv[]) {
4. string Kstr = argv[1];
5. string Lstr = argv[2];
6. int K = stoi(Kstr);
7. int L = stoi(Lstr);
8. std::cout << "k value: " << K << std::endl;
9. std::cout << "L value: " << L << std::endl;
10. string input, temp;
11. while (!std::cin.eof()) {
12. std::cin >> temp;
13. input += temp;
14. }
15. std::cout << input;
16. string seed;
17. for (int i = 0; i < K; i++) {
18. seed += input[i];
19. }
20. std::cout << "Intial K-gram: " << seed << std::endl;
21. RandWriter writer(input, K);
22. try {
23. writer.generate(seed, L);
24. } catch (const std::runtime\_error& error) {
25. std::cout << "Exception caught: " << error.what() << std::endl;
26. }
27. writer.operator<<(std::cout);
28. return 0;
29. }

### RandWriter.hpp

1. // Copyright [2023] <Daniel Bergeron>
2. #ifndef \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS6\_HEADER\_RANDWRITER\_HPP\_
3. #define \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS6\_HEADER\_RANDWRITER\_HPP\_
4. #include <iostream>
5. #include <string>
6. using std::string;
7. #include <map>
8. #include <utility>
9. using std::pair;
10. #include <chrono>
11. #include <random>
12. #include <algorithm>
13. class RandWriter {
14. public:
15. // Note: all of the below constructors/methods should be public.
16. // -------------------------------------------------------------------
17. // create a Markov model of order k from given text
18. RandWriter(string text, int k);  // Assume that text has length at least k.
19. //  -------------------------------------------------------------------
20. int orderK() const;  // order k of Markov model
21. // -------------------------------------------------------------------
22. // number of occurrences of kgram in text
23. int freq(string kgram) const;  // throw an exception if kgram is not of
24. // length k
25. // -------------------------------------------------------------------
26. // number of times that character c follows kgram
27. // if order=0, return num of times char c appears
28. // (throw an exception if kgram is not of length k)
29. int freq(string kgram, char c) const;
30. // -------------------------------------------------------------------
31. // random character following given kgram
32. // (Throw an exception if kgram is not of length k.
33. // Throw an exception if no such kgram.)
34. char kRand(string kgram);
35. // -------------------------------------------------------------------
36. // generate a string of length L characters
37. // by simulating a trajectory through the corresponding
38. // Markov chain. The first k characters of the newly
39. // generated string should be the argument kgram.
40. // Throw an exception if kgram is not of length k.
41. // Assume that L is at least k.
42. string generate(string kgram, int L);
43. // -------------------------------------------------------------------
44. // overload the stream insertion operator and display
45. std::ostream& operator<<(std::ostream& out);
46. // the internal state of the Markov Model. Print out
47. // the order, the alphabet, and the frequencies of
48. // the k-grams and k+1-grams.4
49. private:
50. int order;
51. string input;
52. string alphabet;
53. string currState;
54. // map to hold k grams and the maps it to K+1 grams and there occurence
55. std::map<string, int> kfreq;
56. std::map<string, std::map<string, int>> symbolTable;
57. };
58. #endif  // \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS6\_HEADER\_RANDWRITER\_HPP\_

### RandWriter.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "Header/RandWriter.hpp"
3. template<class Function>
4. void loadHeader(Function fn);
5. // Note: all of the below constructors/methods should be public.
6. // -------------------------------------------------------------------
7. // create a Markov model of order k from given text
8. // Assume that text has length at least k.
9. RandWriter::RandWriter(string text, int k) {
10. input = text;
11. currState = text;
12. int size = static\_cast<int>(text.size());
13. order = k;
14. //  make alpha bet
15. char c;
16. alphabet += text[0];
17. for (int i = 1; i < size; i++) {
18. c = text[i];
19. // find returns npos -1 if nothing found
20. if (static\_cast<int>(alphabet.find(c, 0)) < 0) {
21. alphabet += c;
22. }
23. }
24. std::string kgram, temp;
25. std::map<string, int> kGramPlus1;
26. // loop to fill symbol table
27. // i increments the entire sting
28. // j increments the kgram on the string
29. int j = 0;
30. int index = 0;
31. for (int i = 0; i < size; i++) {
32. kgram.clear();
33. index = i;
34. for (j = 0; j < order; j++) {
35. if (index > size - 1) {
36. index = 0;
37. kgram += text[index];
38. } else {
39. kgram += text[index];
40. }
41. index++;
42. }
43. // insert kgram into its text freq map
44. kfreq[kgram]++;
45. // we have a kgram now we must break it into
46. // its k + 1
47. // builds new map with value and ties it to
48. // old map
49. symbolTable[kgram];
50. if (index > size - 1) {
51. index = 0;
52. }
53. temp = kgram;
54. temp += text[index];
55. kGramPlus1[temp] = 0;
56. (symbolTable[kgram])[temp]++;
57. }
58. }
59. int RandWriter::orderK() const {  // order k of Markov model
60. return order;
61. }
62. // -------------------------------------------------------------------
63. // number of occurrences of kgram in text
64. // throw an exception if kgram is not of length k
65. int RandWriter::freq(string kgram) const {
66. int size = 0;
67. size = static\_cast<int>(kgram.length());
68. int freq = 0;
69. if (order == 0 && kgram == " ") {
70. return static\_cast<int>(input.size());
71. }
72. if (size != order) {
73. std::runtime\_error error(string("Size Mismatch"));
74. throw error;
75. }
76. if (kfreq.find(kgram) == kfreq.end()) {
77. return 0;
78. }
79. freq = kfreq.find(kgram)->second;
80. return freq;
81. }
82. // number of times that character c follows kgram
83. // if order=0, return num of times char c appears
84. // (throw an exception if kgram is not of length k)
85. int RandWriter::freq(string kgram, char c) const {
86. int count = 0;
87. int size = 0;
88. size = static\_cast<int>(kgram.length());
89. if (size != order) {
90. std::runtime\_error error(string("Size Mismatch"));
91. throw error;
92. }
93. string p1kgram;  // k + 1
94. // without lamda
95. if (order == 0) {
96. for (int i = 0; i < static\_cast<int>(input.size()); i++) {
97. if (input[i] == c) {
98. count++;
99. }
100. }
101. return count;
102. }
103. // with lambda
104. // we need to kgram plus one
105. if (kfreq.find(kgram) == kfreq.end()) {
106. return 0;
107. }
108. auto n = symbolTable.find(kgram)->second.begin();
109. for (; n != symbolTable.find(kgram)->second.end(); n++) {
110. p1kgram = n->first;
111. // if first char == c increase the count
112. if (p1kgram[static\_cast<int>((p1kgram.size()-1))] == c) {
113. count += n->second;
114. }
115. }
116. return count;
117. }
118. // -------------------------------------------------------------------
119. // random character following given kgram
120. // (Throw an exception if kgram is not of length k.
121. // Throw an exception if no such kgram.)
122. char RandWriter::kRand(string kgram) {
123. // get all k plus one grams there occurences
124. // add occurences up
125. // generate random number of occurences and from 0 to total occurences
126. // put number on range if
127. // construct a trivial random generator engine from a time-based seed:
128. int size = static\_cast<int>(kgram.length());
129. if (size != order) {
130. std::runtime\_error error(string("Size Mismatch"));
131. throw error;
132. }
133. if (kfreq.find(kgram) == kfreq.end()) {
134. std::runtime\_error error(string("Invalid Kgram"));
135. throw error;
136. }
137. // create generator
138. unsigned seed = std::chrono::system\_clock::now().time\_since\_epoch().count();
139. std::default\_random\_engine generator(seed);
140. auto p = symbolTable[kgram].begin();
141. int upperBounds = 0, randNum;
142. // gets total amount for upperbounds
143. for (unsigned i = 0; i < alphabet.size(); i++) {
144. upperBounds += freq(kgram, alphabet[i]);
145. }
146. // generate number
147. randNum = (generator() % upperBounds) + 1;
148. int first = 0, second = 0;
149. char c = kgram[0];
150. // use randnum to get next letter out of range of possible next char
151. for (p = symbolTable[kgram].begin(); p != symbolTable[kgram].end(); p++) {
152. second += p->second;
153. // std::cout << "First: " << first << "Second: " << second << std::endl;
154. if (randNum <= second) {
155. c = p->first[size];
156. break;
157. } else {
158. first += p->second;
159. }
160. }
161. return c;
162. }
163. // -------------------------------------------------------------------
164. // generate a string of length L characters
165. // by simulating a trajectory through the corresponding
166. // Markov chain. The first k characters of the newly
167. // generated string should be the argument kgram.
168. // Throw an exception if kgram is not of length k.
169. // Assume that L is at least k.
170. string RandWriter::generate(string kgram, int L) {
171. // first get kgram size
172. string chain;
173. int size = static\_cast<int>(kgram.length());
174. if (size != order) {
175. std::runtime\_error error(string("Size Mismatch"));
176. throw error;
177. }
178. if (kfreq.find(kgram) == kfreq.end()) {
179. std::runtime\_error error(string("Invalid Kgram"));
180. throw error;
181. }
182. if (size > L) {
183. for (int i = 0; i < L; i++) {
184. chain += kgram[i];
185. }
186. currState = chain;
187. return chain;
188. }
189. // else we begin the generation
190. chain = kgram;
191. char c;
192. int i = size;
193. // std::cout << size-1 << std::endl;
194. // gets random character from the kgrams
195. while (i < L) {
196. c = kRand(kgram);
197. chain += c;
198. kgram.replace(0, size, chain, i - (size - 1), size);
199. i++;
200. }
201. currState = chain;
202. return chain;
203. }
204. // -------------------------------------------------------------------
205. // << // overload the stream insertion operator and display
206. // the internal state of the Markov Model. Print out
207. // the order, the alphabet, and the frequencies of
208. // the k-grams and k+1-grams.
209. std::ostream& RandWriter::operator<<(std::ostream& out) {
210. auto print = [&]() {
211. out << "\nOrder: " << order << std::endl;
212. out << "Alphabet:\n" << alphabet << std::endl;
213. out << "CurrState:\n" << currState << std::endl;
214. out << "Frequcies k-grams:\n" << std::endl;
215. };
216. loadHeader(print);
217. for (auto i : symbolTable) {
218. out << i.first << " " << freq(i.first) << std::endl;
219. }
220. out << "Frequcies k+1-grams:\n" << std::endl;
221. string str;
222. for (auto i : symbolTable) {
223. for (auto j : i.second) {
224. str = j.first;
225. out << str << " " << j.second << std::endl;
226. }
227. }
228. return out;
229. }
230. // lambda practice
231. template<class Function>
232. void loadHeader(Function fn) {
233. fn();
234. }

### Test.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "Header/RandWriter.hpp"
3. #define BOOST\_TEST\_DYN\_LINK
4. #define BOOST\_TEST\_MODULE TEST
5. #include <boost/test/unit\_test.hpp>
6. // tests exception for size mismatchs and if it is the wrong kgram
7. BOOST\_AUTO\_TEST\_CASE(exceptionThrows) {
8. RandWriter test("gagggagaggcgagaaa", 2);
9. BOOST\_REQUIRE\_THROW(test.freq("gdg"), std::runtime\_error);
10. BOOST\_REQUIRE\_THROW(test.freq("gdg", 'c'), std::runtime\_error);
11. BOOST\_REQUIRE\_THROW(test.kRand("gdg"), std::runtime\_error);
12. BOOST\_REQUIRE\_THROW(test.kRand("fr"), std::runtime\_error);
13. BOOST\_REQUIRE\_THROW(test.kRand("gdgcd"), std::runtime\_error);
14. BOOST\_REQUIRE\_THROW(test.generate("gdg", 1), std::runtime\_error);
15. BOOST\_REQUIRE\_THROW(test.generate("eg", 1), std::runtime\_error);
16. }
17. BOOST\_AUTO\_TEST\_CASE(exceptionNoThrow) {
18. RandWriter test("gagggagaggcgagaaa", 2);
19. BOOST\_REQUIRE\_NO\_THROW(test.freq("gg"));
20. BOOST\_REQUIRE\_NO\_THROW(test.freq("gg", 'c'));
21. BOOST\_REQUIRE\_NO\_THROW(test.kRand("gg"));
22. BOOST\_REQUIRE\_NO\_THROW(test.kRand("gc"));
23. BOOST\_REQUIRE\_NO\_THROW(test.generate("ga", 100));
24. }
25. BOOST\_AUTO\_TEST\_CASE(methodAccuracyTests) {
26. RandWriter test("gagggagaggcgagaaa", 2);
27. BOOST\_REQUIRE(test.orderK() == 2);
28. BOOST\_REQUIRE(test.freq("gg") == 3);
29. BOOST\_REQUIRE(test.freq("ff") == 0);
30. BOOST\_REQUIRE(test.freq("gg", 'c') == 1);
31. BOOST\_REQUIRE(test.kRand("cg") == 'a');
32. string str;
33. str = test.generate("gg", 5);
34. BOOST\_REQUIRE(str.length() == 5);
35. }
36. BOOST\_AUTO\_TEST\_CASE(orderZeroTest) {
37. string str = "gagggagaggcgagaaa";
38. RandWriter test(str, 0);
39. BOOST\_REQUIRE(test.orderK() == 0);
40. BOOST\_REQUIRE(test.freq("", 'c') == 1);
41. BOOST\_REQUIRE(test.kRand(""));
42. str = test.generate("", 5);
43. BOOST\_REQUIRE(str.length() == 5);
44. }

# PS7 – Kronos Time Clock

## Program Discussion

This program was my formal introduction to working with regular expression. Regular expression is a way a programmer can parse strings and build substrings with the data they desire. This was one of the more trivial assignments of this semester and did not require much. This program took input from a Kronos Device log file and used regular expression to pull out information about successful device boots. You are then required to put that information into a .rpt file which only contains successful starts and line number accompanied by if it failed or succeeded. After that the .rpt file contains a time stamp for when each of these events succeeded or failed. The main difficulty with this program for me didn’t come from the regular expression but the managing the flags that tell if the program has already run into a start or not. This was difficult logic to construct, but surprisingly using a more descriptive name for the flag variable helped me think it out. Other than that, integrating the boost time library was a little difficult but by reading the documentation I was able to figure it out.

Text

Description automatically generated

File Output

## Key Elements Discussion

The heart of this program was regular expression, and the boost time library. These Two components are what built this program. After these two features were implemented, it just became a trivial I’ll be it slightly frustrating programming problem.

## What I Learned

Despite doing very little in the way of work for this problem I learned two very important things. First how to use regular expression, which is a very powerful tool I don’t plan on putting down. In handling strings, it is unmatched, many possibilities have opened to me because I learned this skill. Going forward Ill be able to write programs that can parse huge text files and find information I need with little effort. Before this I had to go character by character or do string comparisons. This skill is going to save me a great deal of time in future endeavors. The second major thing I learned from this program is the importance of reading documentation. This program would have been very difficult to implement on my own if I had not implemented libraries. These libraries, while convenient, need to be understood to be utilized to their fullest potential. The only what to do that is to read the documentation.

## Code

### Makefile

1. CC = g++
2. CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
3. OBJ = main.o
4. LIBS = -lboost\_regex -lboost\_date\_time
5. DEPS = Header/KronosLog.hpp
6. all: ps7 lint
7. ps7: $(OBJ)
8. $(CC) $(CFLAGS) -o ps7 $(OBJ) $(LIBS)
9. lint:
10. cpplint \*.cpp $(DEPS)
11. %.o: %.cpp $(DEPS)
12. $(CC) $(CFLAGS) -c $<
13. clean:
14. rm -f $(OBJ) lint ps7

### main.cpp

1. // Copyright [2023] <Daniel Bergeron>
2. #include "Header/KronosLog.hpp"
3. int main(int argc, char\* argv[]) {
4. if (argc != 2) {
5. cout << "Wrong number of arguments\n";
6. return -1;
7. }
8. bool found\_front = false;
9. int lineNum = 1;
10. ifstream fp;
11. std::ofstream ofp;
12. string line, rslt1, rslt2;
13. string filename(argv[1]);
14. boost::posix\_time::ptime start\_Time, end\_Time, total\_Time;
15. std::match\_results<string::const\_iterator> start\_result, end\_result;
16. fp.open(argv[1]);
17. ofp.open(filename + ".rpt");
18. // alternative
19. // std::regex start\_pattern("(\\d+-\\d+-\\d+ \\d+:\\d+:\\d+):
20. // \\(log\\.c\\.\\d+\\) server started ");
21. std::regex start\_pattern(R"((\d+-\d+-\d+ \d+:\d+:\d+): \(log\.c\.166\) server started )");
22. std::regex end\_pattern(R"((\d+-\d+-\d+ \d+:\d+:\d+\.\d+)(.\*)oejs\.AbstractConnector:Started SelectChannelConnector@0\.0\.0\.0:9080)");
23. while (std::getline(fp, line)) {
24. if (std::regex\_match(line, start\_result, start\_pattern)) {
25. if (found\_front == true) {
26. ofp << "\*\*\*\* Incomplete boot \*\*\*\*\n\n";
27. found\_front = false;
28. }
29. found\_front = true;
30. ofp << "=== Device boot ===\n";
31. ofp << lineNum;
32. rslt1 = start\_result[1];
33. ofp << "(" << filename << "): " << rslt1 << " Boot Start\n";
34. } else if (std::regex\_match(line, end\_result, end\_pattern)) {
35. found\_front = false;
36. rslt2 = end\_result[1];
37. ofp << lineNum << "(" << filename << "): "
38. << rslt2 << " Boot Completed\n";
39. start\_Time = time\_from\_string(rslt1);
40. end\_Time = time\_from\_string(rslt2);
41. ofp << "Boot Time: "
42. << (end\_Time - start\_Time).total\_milliseconds() <<"ms\n\n";
43. }
44. lineNum++;
45. }
46. fp.close();
47. return 0;
48. }

### KronosLog.hpp

1. // Copyright [2023] <Daniel Bergeron>
2. #ifndef \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS7\_HEADER\_KRONOSLOG\_HPP\_
3. #define \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS7\_HEADER\_KRONOSLOG\_HPP\_
4. #include <iostream>
5. using std::cin;
6. using std::cout;
7. using std::endl;
8. #include <string>
9. using std::string;
10. #include <fstream>
11. using std::ifstream;
12. #include <regex>
13. #include <boost/regex.hpp>
14. #include "boost/date\_time/posix\_time/posix\_time.hpp"
15. using boost::posix\_time::time\_from\_string;
16. #endif  // \_HOME\_DBERGERO3\_PROGRAMS\_COMP4\_PS7\_HEADER\_KRONOSLOG\_HPP\_