

# COMP IV Section 202: Project Portfolio

Thomas O'Connor

28th April, 2023

## Table of Contents:

- PS0 Hello World with SFML
- PS1 Linear Feedback Shift Register and Image Encoding
- PS2 Sokoban
- PS3 Pythagorean Tree
- PS4 Checkers
- PS5 DNA Alignment
- PS6 RandWriter
- PS7 Kronos Log Parsing

Time to complete: 10 Hours

# **PS0 Hello World with SFML**

## **1 Description of the assignment**

This assignment involved setting up my Linux IDE and relevant audio-visual packages on my machine. I then ensured my machine was configured properly by running the “did I install everything correctly?” SFML code which displayed a window with a green circle titled “SFML works!”. I then extended this demo code by creating and loading my custom sprite into SFML. This sprite was coded to respond to keystrokes (UP, DOWN, LEFT, RIGHT), and move accordingly across the SFML window. For my additional feature, I coded the sprite to respond to keystrokes (EQUAL and HYPHEN) which set the scale of the sprite in the SFML window.

## **2 Key algorithms, Data structures, or OO designs**

For PS0 the key objects that I was introduced to were the sf::RenderWindow and sf::Drawable objects. The RenderWindow object creates a window at runtime of specific dimensions WxH pixels, and allows the depiction of Drawable objects to that window through the window.draw(object) function call. These concepts were crucial in my initial understanding of SFML objects and capabilities.

## **3 What I learned from the project**

I learned how to set the framerate of a RenderWindow object. I also learned how to process window events, including keystrokes and window closing actions. I then correlated these window events to actions for the sprite character. Tangential to the project was learning how to install and run WSL on my machine. I found the combination of WSL with Visual Studio Code to be a good mix of Linux speed with the UI elements of a modern IDE like Visual Studio. After this initial setup in ps0 I was set for all future projects.

## 4 Screenshot of program output

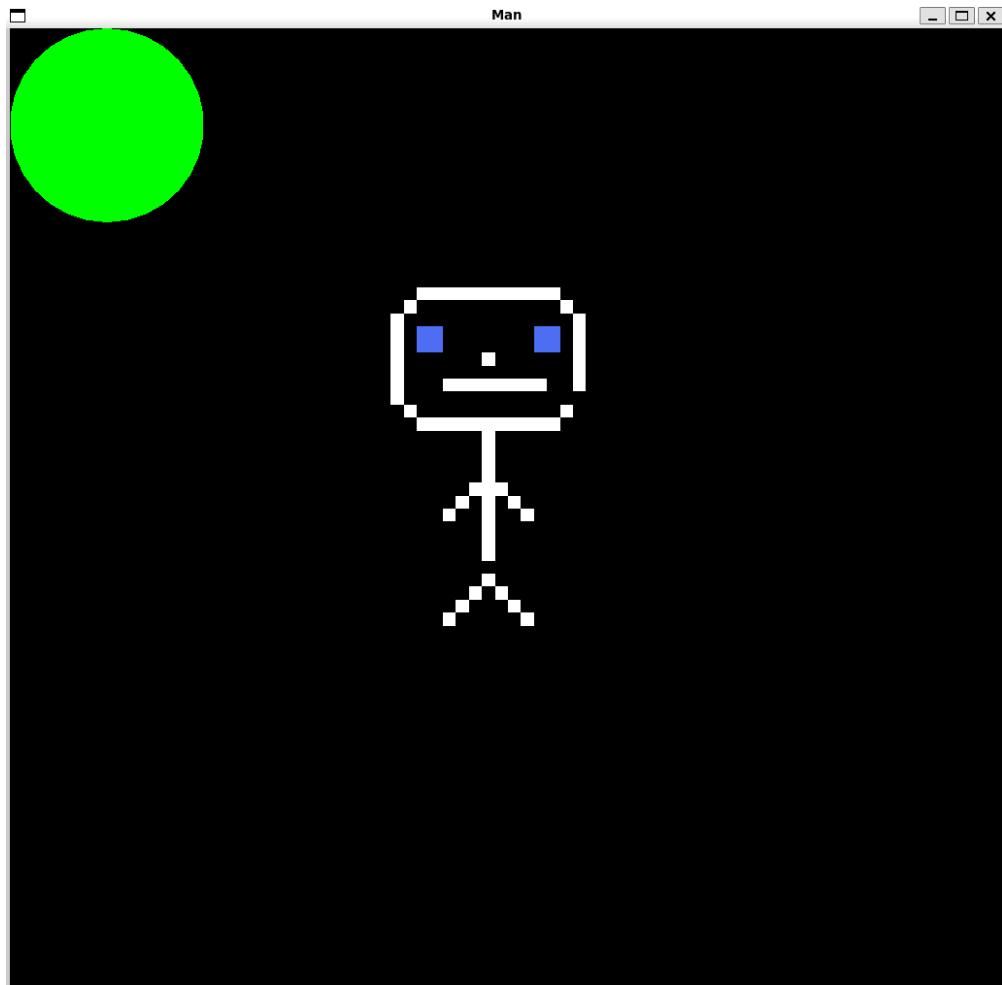


Figure 1: ps0 program output

## 5 Issues and bugs

No issues or bugs present.

The program was fully functional within the given specification.

## 6 Source code for the project

```
Makefile      Tue Apr 11 23:21:11 2023      1
1: CC = g++
2: CFLAGS = --std=c++17 -Wall -Werror -pedantic
3: LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -lboost_un
it_test_framework
4:
5: .PHONY: all clean lint
6:
7: all: lint sfml-app
8:
9: %.o: %.cpp $(DEPS)
10:        $(CC) $(CFLAGS) -c $<
11:
12: sfml-app: main.o
13:        $(CC) $(CFLAGS) -o $@ $^ $(LIB)
14:
15: clean:
16:        rm *.o sfml-app
17:
18: lint:
19:        cpplint *.cpp *.hpp
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include <SFML/Graphics.hpp>
3:
4: using sf::Keyboard;
5:
6: int main() {
7:     // Create the main window
8:     sf::RenderWindow window(sf::VideoMode(1024, 1024), "Man");
9:     window.setFramerateLimit(120);
10:
11:    // Load a sprite to display
12:    sf::Texture textureBase;
13:    if (!textureBase.loadFromFile("sprite.png"))
14:        return EXIT_FAILURE;
15:    sf::Sprite sprite(textureBase);
16:    sprite.setPosition(200, 200);
17:    sprite.setScale(1, 1);
18:
19:    // Load a circle to display
20:    sf::CircleShape shape(100.f);
21:    shape.setFillColor(sf::Color::Green);
22:
23:    // Declare offset variables
24:    int xOffset = 0, yOffset = 0;
25:
26:    // Start the game loop
27:    while (window.isOpen()) {
28:        // Process events
29:        sf::Event event;
30:        while (window.pollEvent(event)) {
31:            // Close window: exit
32:            if (event.type == sf::Event::Closed)
33:                window.close();
34:        }
35:        // Clear screen
36:        window.clear();
37:        // Process keystrokes
38:        if (Keyboard::isKeyPressed(Keyboard::Right)) xOffset = 1;
39:        if (Keyboard::isKeyPressed(Keyboard::Left)) xOffset = -1;
40:        if (Keyboard::isKeyPressed(Keyboard::Up)) yOffset = -1;
41:        if (Keyboard::isKeyPressed(Keyboard::Down)) yOffset = 1;
42:
43:        if (Keyboard::isKeyPressed(Keyboard::Equal))
44:            sprite.scale(1.05, 1.05);
45:        if (Keyboard::isKeyPressed(Keyboard::Hyphen))
46:            sprite.scale(0.95, 0.95);
47:
48:        // Offset the sprite
49:        sprite.move(xOffset, yOffset);
50:
51:        // Draw the sprite
52:        window.draw(sprite);
53:        window.draw(shape);
54:
55:        // Reset Offset values
56:        xOffset = 0; yOffset = 0;
57:
58:        // Update the window
59:        window.display();
60:    }
61:    return EXIT_SUCCESS;
62: }
```

# PS1 Linear Feedback Shift Register and Image Encoding

## 1 Description of the assignment

For part (a) of the assignment I implemented a 16 bit linear feedback shift register of class FibLFSR which generates a pseudo-random string of bits from a given seed. Using constant “tap” points at indices 15, 13, 12, and 10, bits are XOR’d together to generate a new input bit, which is then pushed back onto the register at index 0 which cycles out the bit at index 15. This process was used to generate a single bit using the step() function defined in line 18 of FibLFSR.cpp, as well as pseudo-random integers of size  $0 - 2^k$  by calling the generate(int k) function defined in line 31 of FibLFSR.cpp. To cleanse the input data for the constructor, exceptions were thrown if the seed string was not 16 bits in length, or if the seed string contained non-binary characters.

I created two helper functions on lines 56 and 60 for the XOR of two bits a and b, and for the cleaning of a string to binary characters. Additionally I added const getters on lines 42 and 45, but no setters as end-users do not need to alter the internal state of the FibLFSR model. The output operator was also overloaded at line 50 to easily output the internal state of the FibLFSR object to the end user’s ostream.

For part (b) of the assignment I encrypted a reference image using my FibLFSR from part (a). I created two RenderWindow objects “Base Image” and “Encrypted Image”. In my example program I encoded the Mona Lisa using my LFSR, then decrypted the image back into the Mona Lisa using the same seed for the LFSR. The encryption process occurred in the transform() function with an image by reference and pointer to a FibLFSR object as parameters.

## 2 Key algorithms, Data structures, or OO designs

For part (a) the data structure used for bit storage was a bitset from the bitset library. This structure allowed for an easy conversion from string to bitset using the constructor bitset(string). This structure also allows for easy bit-shifting when the step() method is called, limiting unnecessary function calls and maximizing speed.

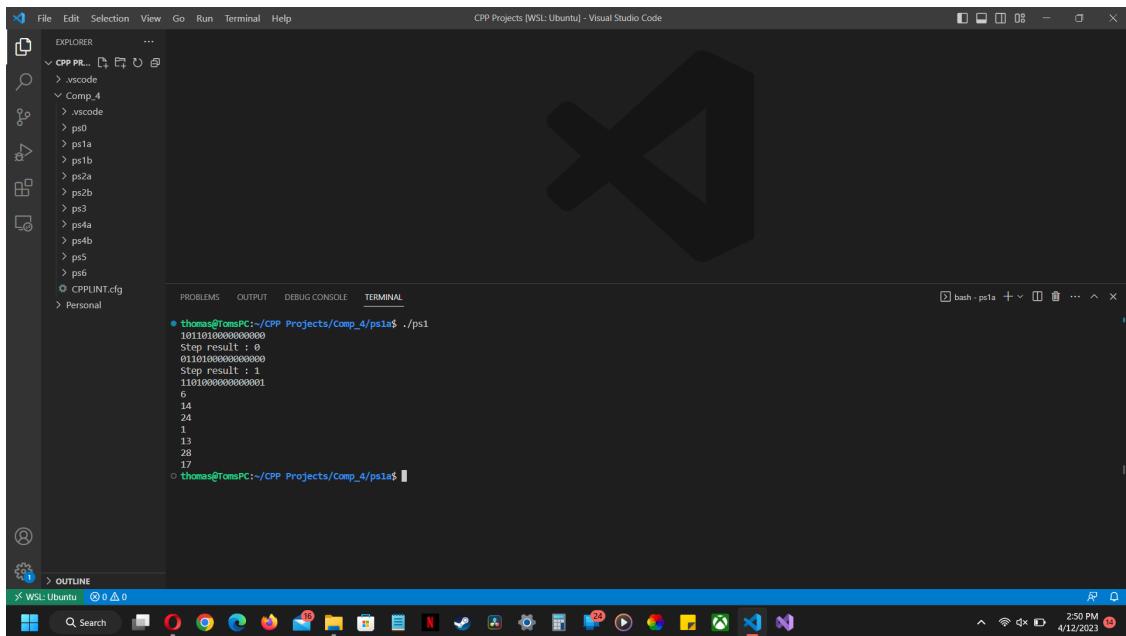
For part (b) the key algorithm was the transform() function. This algorithm took each pixel of the given image and XOR’d the rgb values with a newly generated 8-bit integer from the given FibLFSR. The decryption algorithm used this same transform() function with an identically seeded LFSR which generated the same 8-bit integers which inverted the rgb values of the same pixels in order of encryption.

### 3 What I learned from the project

For part (a) I learned the importance of choosing the correct data structure for the project at hand. I first attempted to use a std::string as my state storage method, but this object proved cumbersome when shifting bits. I tested a doubly-linked-list approach but this gave trouble when accessing tap-points. After doing some research I found the bitset library had the necessary built-in methods I was searching for, making the implementation of my own methods easy and intuitive.

For part (b) I learned the importance of using sprite objects over texture objects due to their decreased system impact. Additionally I learned how to effectively gather command-line arguments for program execution, including file names and binary strings. Finally, I learned about rgb information in color objects, allowing for the augmentation and extraction of exact color values.

### 4 Screenshot of program output

A screenshot of a Windows desktop environment showing a Visual Studio Code window. The window title is "CPP Projects [WSL: Ubuntu] - Visual Studio Code". The left sidebar shows a project structure under "EXPLORER" with a folder named "CPP PR..." containing ".vscode", "Comp\_4", and several files named "ps0", "ps1a", "ps1b", "ps2a", "ps2b", "ps3", "ps4a", "ps4b", "ps5", and "ps6". Below the project structure is a "CPPLINT.cfg" file. The bottom-left corner of the sidebar shows "Personal". The main area is the "TERMINAL" tab, which displays the output of the command "./ps1a". The terminal output shows binary data being processed step-by-step, with results like "Step result : 0", "Step result : 1", and a list of numbers: 6, 14, 24, 1, 13, 28, 17. The bottom status bar indicates the session is "WSL: Ubuntu" and shows the date and time as "4/12/2023 2:50 PM".

```
● thomas@TomsPC:~/CPP Projects/Comp_4$ ./ps1a
0111010000000000
Step result : 0
0110100000000000
Step result : 1
1101000000000001
6
14
24
1
13
28
17
● thomas@TomsPC:~/CPP Projects/Comp_4$
```

Figure 2: ps1a program output

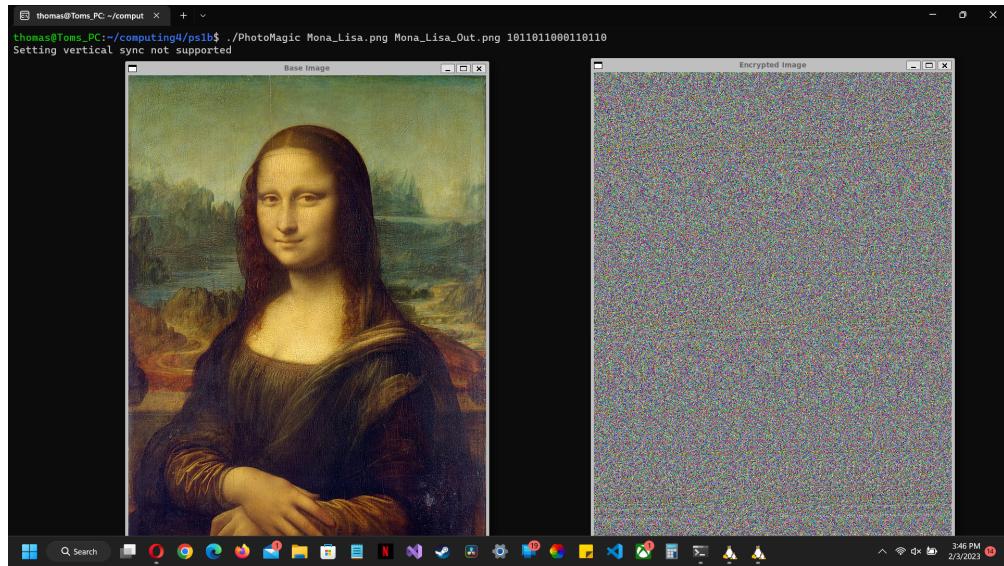


Figure 3: ps1b encryption output

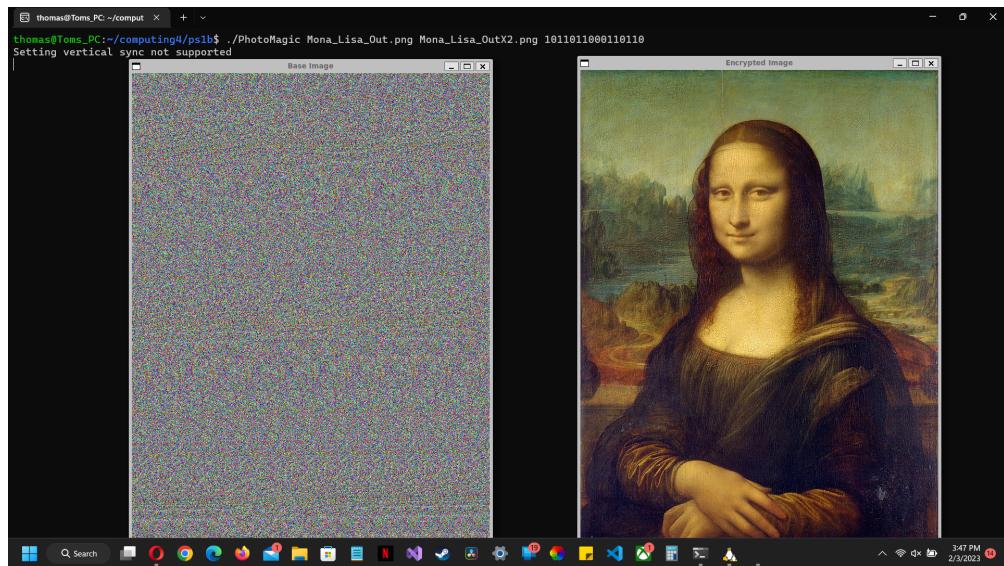


Figure 4: ps1b decryption output

## 5 Issues and bugs

No issues or bugs present.

The program was fully functional within the given specification.

## 6 Source code for the project

```
Makefile           Mon Feb 06 00:18:47 2023           1
1: CC = g++
2: FLAGS = -Wall -Werror -pedantic -std=c++17
3: LIB = -lboost_unit_test_framework
4:
5: .PHONY: all clean
6:
7: all: ps1 test
8:
9: ps1: main.o FibLFSR.o
10:      $(CC) $(FLAGS) -o ps1 $^ $(LIB)
11:
12: test: test.o FibLFSR.o
13:      $(CC) $(FLAGS) -o test $^ $(LIB)
14:
15: %.o: %.cpp $(DEPS)
16:      $(CC) $(FLAGS) -c $<
17:
18: clean:
19:      rm *.o ps1 test
```

```
1: #include "FibLFSR.hpp"
2:
3: int main() {
4:
5:     string seed("1011010000000000");
6:     FibLFSR A(seed);
7:     cout << A << endl;
8:     cout << "Step result : " << A.step() << endl;
9:     cout << A << endl;
10:    cout << "Step result : " << A.step() << endl;
11:    cout << A << endl;
12:
13:    FibLFSR abc("1100011011000011");
14:
15:    for(int i = 0; i < 7; i++) cout << abc.generate(5) << endl;
16:
17:
18:    return 0;
19: }
```

```
1: #ifndef FIB_LFSR
2: #define FIB_LFSR
3:
4: #include <iostream>
5: #include <string>
6: #include <bitset>
7: #include <exception>
8:
9: using namespace std;
10:
11: class FibLFSR {
12: public:
13:     // Constructor to create LFSR with the given initial seed
14:     FibLFSR(string seed);
15:
16:     // Simulate one step and return the new bit as 0 or 1
17:     int step();
18:
19:     // Simulate k steps and return a k-bit integer
20:     int generate(int k);
21:
22:     // Getters:
23:     string getState(void) const;
24:     const int* getTaps(void) const;
25:
26:     // Helper functions:
27:     int XOR(int a, int b);
28:     bool notZeroOne(const string seed);
29:
30: private:
31:     const int TAPS[4] = {15, 13, 12, 10};
32:     bitset<16> state;
33: };
34: ostream& operator<<(ostream& out, const FibLFSR& lfsr);
35:
36: #endif
```

```
1: #include "FibLFSR.hpp"
2:
3: // Constructor to create LFSR with the given initial seed
4: FibLFSR::FibLFSR(string seed) {
5:     // Checks for invalid input
6:     if(seed.size() != 16) {
7:         throw length_error("Parameter of size "+to_string(seed.size()));
8:     }
9:     if(notZeroOne(seed)){
10:         throw invalid_argument("Parameter contains !1 && !0");
11:     }
12:     // Converts to bitset for easy xors and shifts
13:     bitset<16> bits(seed);
14:     state = bits;
15: }
16:
17: // Simulate one step and return the new bit as 0 or 1
18: int FibLFSR::step() {
19:     int feedback = 0;
20:     for(int i = 0; i < 4; i++){
21:         // Performs the 4 xors for the 4 taps
22:         feedback = XOR(state[TAPS[i]], feedback);
23:     }
24:     // Leftshifts, inserts result, returns result
25:     state <<= 1;
26:     if (feedback) state[0] = 1;
27:     return feedback;
28: }
29:
30: // Simulate k steps and return a k-bit integer
31: int FibLFSR::generate(int k) {
32:     int total = 0;
33:     for(int i = 0; i < k; i++){
34:         // For every iteration, add one on success and shift bits left
35:         total *= 2;
36:         if(step()) total +=1;
37:     }
38:     return total;
39: }
40:
41: // Getters:
42: string FibLFSR::getState(void) const {
43:     return state.to_string();
44: }
45: const int* FibLFSR::getTaps(void) const {
46:     return TAPS;
47: }
48:
49: // Output operator overload
50: ostream& operator<<(ostream& out, const FibLFSR& lfsr){
51:     out << lfsr.getState();
52:     return out;
53: }
54:
55: // Helpers:
56: int FibLFSR::XOR(int a, int b) {
57:     if ((a || b) && !(a && b)) return 1;
58:     else return 0;
59: }
60: bool FibLFSR::notZeroOne(const string seed) {
61:     for(char a : seed){
62:         if(a != '0' && a != '1') return 1;
63:     }
64:     return 0;
65: }
```

```
Makefile      Fri Feb 03 15:35:44 2023      1
1: CC = g++
2: FLAGS = -Wall -Werror -pedantic -std=c++17
3: LIB = -lboost_unit_test_framework -lsfml-system -lsfml-graphics -lsfml-wi
ndow
4: DEPS = FibLFSR.o
5:
6: .PHONY: all clean
7:
8: all: PhotoMagic test
9:
10: PhotoMagic: PhotoMagic.o $(DEPS)
11:         $(CC) $(FLAGS) -o PhotoMagic $^ $(LIB)
12:
13: test: test.o $(DEPS)
14:         $(CC) $(FLAGS) -o test $^ $(LIB)
15:
16: %.o: %.cpp
17:         $(CC) $(FLAGS) -c $<
18:
19: clean:
20:         rm *.o PhotoMagic test
```

```
1: #include "FibLFSR.hpp"
2:
3: #include <SFML/System.hpp>
4: #include <SFML/Window.hpp>
5: #include <SFML/Graphics.hpp>
6:
7: // Transforms image using FibLFSR
8: void transform(sf::Image&, FibLFSR*);
9: // Display an encrypted copy of the picture, using the LFSR to do the encryption
10:
11:
12: int main(int argc, char* argv[]) {
13:
14:     // Command line arguments
15:     const string inputFileName = argv[1];
16:     const string outputFileName = argv[2];
17:     const string binaryString = argv[3];
18:
19:     // Create a base image and a to-be-encrypted image
20:     sf::Image imageBase, imageEncrypt;
21:     if (!imageBase.loadFromFile(inputFileName)) return -1;
22:     if (!imageEncrypt.loadFromFile(inputFileName)) return -1;
23:
24:     // Create sprite for base image
25:     sf::Vector2u size = imageBase.getSize();
26:     sf::RenderWindow windowBase(sf::VideoMode(size.x, size.y), "Base
Image");
27:     sf::Texture textureBase;
28:     textureBase.loadFromImage(imageBase);
29:     sf::Sprite spriteBase;
30:     spriteBase.setTexture(textureBase);
31:
32:     // Perform the transformation using the given binary string
33:     FibLFSR L1(binaryString);
34:     transform(imageEncrypt, &L1);
35:
36:     // Create sprite for encrypted image
37:     sf::RenderWindow windowEncrypt(sf::VideoMode(size.x, size.y), "En
crypted Image");
38:     sf::Texture textureEncrypt;
39:     textureEncrypt.loadFromImage(imageEncrypt);
40:     sf::Sprite spriteEncrypt;
41:     spriteEncrypt.setTexture(textureEncrypt);
42:
43:     // While both windows are open
44:     while (windowBase.isOpen() && windowEncrypt.isOpen()) {
45:
46:         sf::Event event;
47:         // If either receive a call to close, close both
48:         while (windowBase.pollEvent(event) || windowEncrypt.pollE
vent(event)){
49:             if (event.type == sf::Event::Closed){
50:                 windowBase.close(); windowEncrypt.close()
;
51:             }
52:         }
53:         // Display base image
54:         windowBase.clear(sf::Color::White);
55:         windowBase.draw(spriteBase);
56:         windowBase.display();
57:         // Display encrypted image
58:         windowEncrypt.clear(sf::Color::White);
59:         windowEncrypt.draw(spriteEncrypt);
60:         windowEncrypt.display();
```

```
61:         }
62:         // Save to file, else return failure/error
63:         if (!imageEncrypt.saveToFile(outputFileName)) return -1;
64:         return 0;
65:     }
66:
67: void transform(sf::Image& img, FibLFSR* lfsr){
68:
69:     sf::Color p;
70:     sf::Vector2u size = img.getSize();
71:     // For every x and y in img, xor each rgb val with a newly genera
ted 8 bit num
72:     for (int x = 0; x < (int)size.x; x++) {
73:         for (int y = 0; y < (int)size.y; y++) {
74:             p = img.getPixel(x, y);
75:             p.r ^= lfsr->generate(8);
76:             p.g ^= lfsr->generate(8);
77:             p.b ^= lfsr->generate(8);
78:             img.setPixel(x, y, p);
79:         }
80:     }
81: }
```

```
test.cpp      Fri Feb 03 09:30:42 2023      1

1: // Copyright 2022
2: // By Dr. Rykalova
3: // Editted by Dr. Daly
4: // test.cpp for PS1a
5: // updated 5/12/2022
6:
7: #include "FibLFSR.hpp"
8:
9: #define BOOST_TEST_DYN_LINK
10: #define BOOST_TEST_MODULE Main
11: #include <boost/test/unit_test.hpp>
12:
13: BOOST_AUTO_TEST_CASE(testStepInstr1) {
14:     FibLFSR l("1011011000110110");
15:     BOOST_REQUIRE_EQUAL(l.step(), 0);
16:     BOOST_REQUIRE_EQUAL(l.step(), 0);
17:     BOOST_REQUIRE_EQUAL(l.step(), 0);
18:     BOOST_REQUIRE_EQUAL(l.step(), 1);
19:     BOOST_REQUIRE_EQUAL(l.step(), 1);
20:     BOOST_REQUIRE_EQUAL(l.step(), 0);
21:     BOOST_REQUIRE_EQUAL(l.step(), 0);
22:     BOOST_REQUIRE_EQUAL(l.step(), 1);
23: }
24:
25: BOOST_AUTO_TEST_CASE(testStepInstr2) {
26:     FibLFSR l2("1011011000110110");
27:     BOOST_REQUIRE_EQUAL(l2.generate(9), 51);
28: }
29:
30: BOOST_AUTO_TEST_CASE(testStepInstr3) { // length_error - parameter length
!= 16
31:     BOOST_CHECK_THROW(FibLFSR l3(""), length_error);
32:     BOOST_CHECK_THROW(FibLFSR l3("1111"), length_error);
33:     BOOST_CHECK_THROW(FibLFSR l3("111111111111111111111111"), length_error);
34: }
35:
36: BOOST_AUTO_TEST_CASE(testStepInstr4) { // invalid_argument - parameter co
ntains !1 && !0
37:     BOOST_REQUIRE_THROW(FibLFSR l4("1011011000110112"), invalid_argument);
38:     BOOST_REQUIRE_THROW(FibLFSR l4("101101100011011c"), invalid_argument);
39: }
40:
41: BOOST_AUTO_TEST_CASE(testStepInstr5) { // Both length_error and invalid_a
rgument
42:     BOOST_REQUIRE_THROW(FibLFSR l5("abcd"), length_error);
43:     BOOST_REQUIRE_THROW(FibLFSR l5("abcdabcdabcdabcd"), length_error);
44: }
```

# PS2 Sokoban

## 1 Description of the assignment

For part (a) of the assignment I created a Sokoban UI that inherited the virtual draw() function from the sf::Drawable class. This Sokoban class took in a file name from the command line, then opened the file and extracted the game state into the Game object using the overloaded extraction operator.

In these .lvl files, the following symbols are used:

**@** The initial position of the player. Each level contains exactly one @.

**.** An empty space, which the player can move through.

**#** A wall, which blocks movement.

**A** A box, which can be pushed by the player.

**a** A storage location, where the player is trying to push a box.

**1** A box that is already in a storage location.

A window titled “sokoban” was created in the exact dimensions of the level detailed in the file (64Wx64H). Each tile was 64 pixels on each side, making tileability easy. The window depicted tiles in their corresponding locations from the .lvl file. Additionally, a gameplay clock was included in the upper left corner of the window that displayed the elapsed time. I provided getters for internal variables, as well as a custom getGameState() function which printed the internal representation of the gameState to the terminal, which I used for debugging the value constructor and extraction operator.

For part (b) of the assignment I implemented movement and gameplay mechanics. Using the (W,A,S,D) keys or (UP, DOWN, LEFT, RIGHT) keys, the end user could move the “janitor” character around the level through empty tiles, could push boxes onto other empty tiles, and could push boxes onto storage locations where they would change texture, indicating a successful storage. The player could not walk through walls, push boxes into walls, or push boxes into other boxes. The player could also walk through the empty storage locations. When all storage locations are filled with boxes, a win screen was presented to the user along with a win fanfare. This win screen presented the user’s completion time. The player had access to additional commands: (R) to restart the game, (Z) to undo turns, and (X) to close the window.

## 2 Key algorithms, Data structures, or OO designs

In part (a) the data structure I used was a 2D vector of chars. This vector was resized to WxH from the input file, then filled with the corresponding symbols. This 2D vector stored the game in row-major order, since the IDE reads in a file in row-major order. The second component was the algorithm in the draw() function override, where a switch statement triggered the drawing of the appropriately textured sprite.

In part (b) the primary data structure I used was a 3D vector of chars. This variable “allGameStates” stored the 2D matrix of game states as in part (a), but a third vector was added to include the undo function. Alongside a “turn” variable, every movement of the player incremented the turn variable and created a new game state matrix, pushing it back onto the 3D vector. If the player attempted to undo an action with command (Z), the game state would revert to the previous matrix at turn -1 for each undo action requested. Then as the player resumed normal movement, the matrix would overwrite with the new game states.

In conjunction with the 3D vector and the turn variable, a new vector of type Direction was created labeled “face”. This vector was indexed using “turn” and corresponded to the direction the player character was facing. As the player moved in a particular Direction, the texture of the character was updated accordingly. As the player performed undo commands, the character direction would correspond to the movement actions of previous turns.

## 3 What I learned from the project

For part (a) I learned about the power of nested data structures. Rather than allocating one large block of memory and doing coordinate arithmetic to locate my indices, I could use easily manageable structures with sensible indexing notation. I made frequent use of the double square-bracket operator (`gameState[h][w]`). This notation corresponded with my mental interpretation of the game board which sped up my coding and reduced `out_of_bounds` errors from the vectors. I was also reminded of the importance and power of defining global variables. In the case of ps2a, I defined a `TILE_SIZE` int of size 64, indicating the pixel dimensions of my tiles.

For part (b) I familiarized myself with companion data structures, using the same indexing element “turn” to access concurrent states across different objects. I had initial trouble combining player directionality and undo commands. Using a vector of directions allowed me to store the history of player movement at each “turn” and correspond it to the player character at any “turn”, regardless of direction of movement through the 3rd game state vector.

Additionally, I familiarized myself with offloading work to private helper functions. These functions include `noObstructions()` at line 106, `canPushBox()` at line 172, and `pushBox()` at line 206 of `Sokoban.cpp`. I deferred complex calculations, bounds-detection, and movements outside of function `movePlayer()` at line 32.

## 4 Screenshot of program output

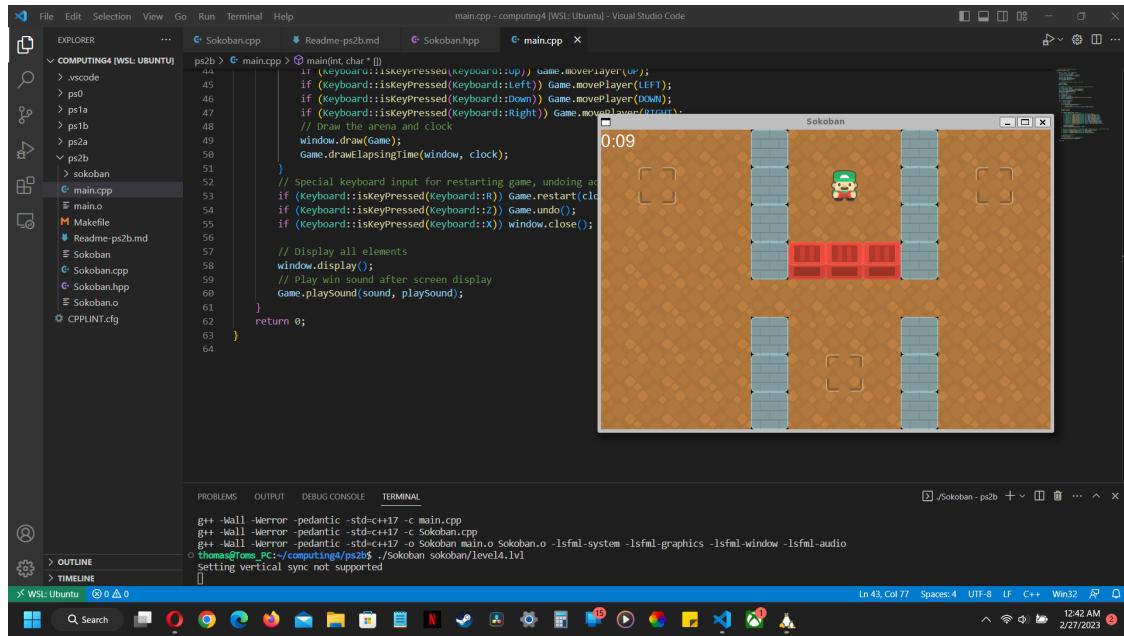


Figure 5: ps2b game output

## 5 Issues and bugs

- The Janitor was unable to push stored boxes out of storage areas.
- The stored boxes updated texture from red to blue and behaved as walls.
- Occasionally, textures flickered.

Otherwise, the program was functional within the given specification.

## 6 Source code for the project

**Makefile**        **Wed Feb 08 11:20:47 2023**        **1**

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

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Sokoban.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     // Command line arguments
6:     std::string inputFileName = argv[1];
7:     // Open file and check for failure
8:     std::ifstream myLevel;
9:     myLevel.open(inputFileName);
10:    if (!myLevel) exit(1);
11:
12:    // Create game and input values from file
13:    Sokoban Game;
14:    Game >> myLevel;
15:
16:    // Rednder the window using the appropriate game dimensions
17:    sf::RenderWindow window
18:        (sf::VideoMode(TILE_SIZE * Game.getWidth(),
19:                      TILE_SIZE * Game.getHeight()), "Sokoban");
20:    window.setFramerateLimit(120);
21:    sf::Clock clock;
22:
23:    while (window.isOpen()) {
24:        // Process events
25:        sf::Event event;
26:        while (window.pollEvent(event)) {
27:            // Close window: exit
28:            if (event.type == sf::Event::Closed) window.close();
29:        }
30:        // Clear screen
31:        window.clear();
32:        // Draw the basic arena
33:        window.draw(Game);
34:        Game.drawElapsingTime(window, clock);
35:        window.display();
36:    }
37:    return 0;
38: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef SOKOBAN_HPP
3: #define SOKOBAN_HPP
4:
5: #include <fstream>
6: #include <iostream>
7: #include <string>
8: #include <vector>
9: #include <SFML/Graphics.hpp>
10: #include <SFML/System.hpp>
11: #include <SFML/Window.hpp>
12:
13: using std::cout;
14: using std::endl;
15:
16: #define TILE_SIZE 64
17:
18: class Sokoban : public sf::Drawable {
19: public:
20:     // Constructors
21:     Sokoban() : _h(0), _w(0) {}
22:     Sokoban(int y, int x) : _h(y), _w(x) {}
23:
24:     // Getters
25:     int getWidth() const { return _w; }
26:     int getHeight() const { return _h; }
27:     void getGameState() const;
28:
29:     // Display
30:     void drawElapsingTime(sf::RenderWindow &window, sf::Clock &clock);
31:
32:     // Interactors
33:     void movePlayer(Sokoban &game, sf::Keyboard::Key key);
34:
35:     // Overload extraction operator
36:     friend Sokoban& operator>>(Sokoban& game, std::ifstream& file);
37:
38: private:
39:     // Draw game in SFML
40:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const;
41:
42: private:
43:     // Game state stored in row-major order
44:     std::vector<std::vector<char>> gameState;
45:     // Dimensions of the window/game (read from file)
46:     int _h, _w;
47: };
48: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Sokoban.hpp"
3:
4: // Extract the entire gameState into the Sokoban object
5: Sokoban& operator>>(Sokoban& game, std::ifstream& file) {
6:     file >> game._h; file >> game._w;
7:     // Given game dimensions, resize 2D vector and input from file
8:     game.gameState.resize(game._h);
9:     for (int i = 0; i < game._h; i++) {
10:         game.gameState[i].resize(game._w);
11:         for (int j = 0; j < game._w; j++) {
12:             file >> game.gameState[i][j];
13:         }
14:     }
15:     return game;
16: }
17:
18: // Output the internal representation of the gameState to the terminal
19: void Sokoban::getGameState() const {
20:     for (int i = 0; i < _h; i++) {
21:         for (int j = 0; j < _w; j++) {
22:             cout << gameState[i][j];
23:         }
24:         cout << endl;
25:     }
26: }
27:
28: // Overload the virtual draw function:
29: // Load required textures and display in window at prescribed locations
30: void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
31:     sf::Texture Wall, Box, Empty, Storage, Man;
32:     if (!Wall.loadFromFile("sokoban/block_06.png")) exit(1);
33:     if (!Box.loadFromFile("sokoban/crate_03.png")) exit(1);
34:     if (!Empty.loadFromFile("sokoban/ground_01.png")) exit(1);
35:     if (!Storage.loadFromFile("sokoban/ground_04.png")) exit(1);
36:     if (!Man.loadFromFile("sokoban/player_05.png")) exit(1);
37:
38:     for (int i = 0; i < _h; i++) {
39:         for (int j = 0; j < _w; j++) {
40:             sf::Sprite tile, backTile;
41:             tile.setPosition(j * TILE_SIZE, i * TILE_SIZE);
42:
43:             switch (gameState[i][j]) {
44:                 case '#':
45:                     tile.setTexture(Wall);
46:                     break;
47:                 case '.':
48:                     tile.setTexture(Empty);
49:                     break;
50:                 case 'a':
51:                     tile.setTexture(Storage);
52:                     break;
53:                 case 'A':
54:                     backTile.setPosition(j * TILE_SIZE, i * TILE_SIZE);
55:                     backTile.setTexture(Empty);
56:                     target.draw(backTile);
57:                     tile.setTexture(Box);
58:                     break;
59:                 case '@':
60:                     backTile.setPosition(j * TILE_SIZE, i * TILE_SIZE);
61:                     backTile.setTexture(Empty);
62:                     target.draw(backTile);
63:                     tile.setTexture(Man);
64:                     break;
```

```
65:         }
66:         target.draw(tile);
67:     }
68: }
69: }
70:
71: // Function that displays time in upper-left corner
72: void Sokoban::drawElapsingTime(sf::RenderWindow &window, sf::Clock &clock
) {
73:     sf::Time elapsed = clock.getElapsedTime();
74:     int minutes = elapsed.asSeconds() / 60;
75:     int seconds = static_cast<int>(elapsed.asSeconds()) % 60;
76:
77:     std::string timeString = std::to_string(minutes) + ":" +
78:     (seconds < 10 ? "0" : "") + std::to_string(seconds);
79:
80:     sf::Font font;
81:     font.loadFromFile("sokoban/arial.ttf");
82:     sf::Text timeText(timeString, font, 30);
83:     timeText.setFillColor(sf::Color::White);
84:
85:     window.draw(timeText);
86: }
```

```
Makefile           Wed Feb 15 12:02:11 2023           1
1: CC = g++
2: FLAGS = -Wall -Werror -pedantic -std=c++17
3: LIB = -lsfml-system -lsfml-graphics -lsfml-window -lsfml-audio
4: DEPS = Sokoban.o
5:
6: .PHONY: all clean lint
7:
8: all: lint Sokoban
9:
10: Sokoban: main.o Sokoban.o $(DEPS)
11:         $(CC) $(FLAGS) -o Sokoban $^ $(LIB)
12:
13: %.o: %.cpp
14:         $(CC) $(FLAGS) -c $<
15:
16: lint:
17:         cpplint --recursive --quiet .
18:
19: clean:
20:         rm *.o Sokoban
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Sokoban.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     // Command line arguments
6:     std::string inputFileName = argv[1];
7:     // Open file and check for failure
8:     std::ifstream myLevel;
9:     myLevel.open(inputFileName);
10:    if (!myLevel) exit(1);
11:
12:    // Create game and input values from file
13:    Sokoban Game;
14:    Game >> myLevel;
15:
16:    // Rednder the window using the appropriate game dimensions
17:    sf::RenderWindow window
18:        (sf::VideoMode(TILE_SIZE * Game.getWidth(),
19:                      TILE_SIZE * Game.getHeight()), "Sokoban");
20:    window.setFramerateLimit(15);
21:    // Create clock
22:    sf::Clock clock;
23:    // Load win condition variable
24:    bool performWinConditionOnce = 1;
25:
26:    while (window.isOpen()) {
27:        // Process events
28:        sf::Event event;
29:        while (window.pollEvent(event)) {
30:            // Close window: exit
31:            if (event.type == sf::Event::Closed) window.close();
32:        }
33:        // Clear screen
34:        window.clear();
35:
36:        if (!Game.isWon()) {
37:            // Get Keyboard input and impliment correct movement
38:            if (Keyboard::isKeyPressed(Keyboard::W)) Game.movePlayer(UP);
39:            if (Keyboard::isKeyPressed(Keyboard::A)) Game.movePlayer(LEFT);
40:            if (Keyboard::isKeyPressed(Keyboard::S)) Game.movePlayer(DOWN);
41:            if (Keyboard::isKeyPressed(Keyboard::D)) Game.movePlayer(RIGHT);
42:            if (Keyboard::isKeyPressed(Keyboard::Up)) Game.movePlayer(UP);
43:            if (Keyboard::isKeyPressed(Keyboard::Left)) Game.movePlayer(LEFT);
44:            if (Keyboard::isKeyPressed(Keyboard::Down)) Game.movePlayer(DOWN);
45:            if (Keyboard::isKeyPressed(Keyboard::Right)) Game.movePlayer(RIGHT);
46:            // Draw the arena and clock
47:            window.draw(Game);
48:            Game.drawElapsingTime(window, clock);
49:            // Display all elements
50:            window.display();
51:        } else if (performWinConditionOnce) {
52:            // Draw the win state
53:            sf::Time elapsed = clock.restart();
54:            int minutes = elapsed.asSeconds() / 60;
55:            int seconds = static_cast<int>(elapsed.asSeconds()) % 60;
56:            std::string timeString = "      you win\nyour time: " + std::to_string(minutes) + ":" +
57:                (seconds < 10 ? "0" : "") + std::to_string(seconds);
```

```
58:             sf::Font font;
59:             font.loadFromFile("sokoban/arial.ttf");
60:             sf::Text timeText(timeString, font, 30);
61:             timeText.setFillColor(sf::Color::White);
62:             timeText.setPosition(sf::Vector2f(
63:                 (Game.getWidth() * TILE_SIZE / 2) - 120,
64:                 (Game.getHeight() * TILE_SIZE / 2) - 60));
65:             window.draw(timeText);
66:             window.display();
67:             // Play win sound after screen display
68:             Game.playSound();
69:             performWinConditionOnce = 0;
70:         }
71:     }
72:     // Special keyboard input for restarting game, undoing actions, a
nd closing window
73:     if (Keyboard::isKeyPressed(Keyboard::R)) {
74:         Game.restart(clock);
75:         performWinConditionOnce = 1;
76:     }
77:     if (Keyboard::isKeyPressed(Keyboard::Z)) {
78:         Game.undo();
79:         performWinConditionOnce = 1;
80:     }
81:     if (Keyboard::isKeyPressed(Keyboard::X)) window.close();
82: }
83: return 0;
84: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef SOKOBAN_HPP
3: #define SOKOBAN_HPP
4:
5: #include <algorithm>
6: #include <fstream>
7: #include <iostream>
8: #include <string>
9: #include <vector>
10: #include <SFML/Audio.hpp>
11: #include <SFML/Graphics.hpp>
12: #include <SFML/System.hpp>
13: #include <SFML/Window.hpp>
14:
15: using sf::Keyboard;
16: using std::cout;
17: using std::endl;
18: using std::pair;
19: using std::vector;
20:
21: #define TILE_SIZE 64
22:
23: enum Direction { UP, LEFT, DOWN, RIGHT };
24:
25: class Sokoban : public sf::Drawable {
26: public:
27:     // Constructors
28:     Sokoban() : _h(0), _w(0) {}
29:     Sokoban(int y, int x) : _h(y), _w(x) {}
30:
31:     // Getters
32:     int getWidth() const { return _w; }
33:     int getHeight() const { return _h; }
34:     int getTurn() const { return turn; }
35:
36:     // Display
37:     void drawElapsingTime(sf::RenderWindow &window, sf::Clock &clock);
38:     // Check and Display
39:     bool isWon() const;
40:
41:     // Interactors
42:     void movePlayer(Direction dir);
43:     void restart(sf::Clock& clock);
44:     void undo();
45:     void playSound();
46:
47:     // Overload extraction operator
48:     friend Sokoban& operator>>(Sokoban& game, std::ifstream& file);
49:
50: private:
51:     // Draw game in SFML
52:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
53: const;
53:     // Test for conditions
54:     bool noObstructions(Direction dir, int w, int h) const;
55:     bool canPushBox(Direction dir, int w, int h) const;
56:     // Push box if conditions are met
57:     void pushBox(Direction dir, int w, int h);
58:
59: private:
60:     // Game state stored in row-major order
61:     // Vector of 2D game states - 3rd dimension is time
62:     // Allows for "undo" actions
63:     vector<vector<vector<char>>> allGameStates;
64:     // Dimensions of the window/game (read from file)
```

```
65:     int _h, _w;
66:     // Turn number for keeping track of game states
67:     int turn = 0;
68:     // Direction the man faces
69:     vector<Direction> face;
70: };
71:
72: // non-member helper functions
73: vector<vector<char>> deepCopy(const vector<vector<char>> &original);
74:
75: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Sokoban.hpp"
3:
4: // Extract the entire gameState into the Sokoban object
5: Sokoban& operator>>(Sokoban& game, std::ifstream& file) {
6:     file >> game._h; file >> game._w;
7:     game.face.resize(game.turn + 1);
8:     game.face[0] = DOWN;
9:     game.allGameStates.resize(game.turn + 1);
10:    // Given game dimensions, resize 2D vector and input from file
11:    game.allGameStates[game.turn].resize(game._h);
12:    for (int i = 0; i < game._h; i++) {
13:        game.allGameStates[game.turn][i].resize(game._w);
14:        for (int j = 0; j < game._w; j++) {
15:            file >> game.allGameStates[game.turn][i][j];
16:        }
17:    }
18:    return game;
19: }
20:
21: bool Sokoban::isWon() const {
22:     auto it = std::find_if(allGameStates[turn].begin(), allGameStates[tur
n].end(),
23:         [] (const std::vector<char>& row) { return std::any_of(row.begin()
, row.end(),
24:             [] (char c) { return c == 'A' || c == '2'; });
25:         });
26:     if (it != allGameStates[turn].end()) return false;
27:     else
28:         return true;
29: }
30:
31: // Moves the player given a direction
32: void Sokoban::movePlayer(Direction dir) {
33:     for (int i = 0; i < _h; i++) {
34:         for (int j = 0; j < _w; j++) {
35:             // When find player, check for no obstructions and execute ac
tion
36:             if (allGameStates[turn][i][j] == '@' || allGameStates[turn][i
][j] == '2') {
37:                 if (noObstructions(dir, j, i)) {
38:                     // If no obstructions, create a new turn and a new ga
me state
39:                     turn++;
40:                     allGameStates.resize(turn + 1);
41:                     allGameStates[turn] = deepCopy(allGameStates[turn - 1
]);
42:                     face.resize(turn + 1);
43:                     face[turn] = dir;
44:                     if (allGameStates[turn][i][j] == '2') allGameStates[t
urn][i][j] = 'a';
45:                     else
46:                         allGameStates[turn][i][j] = '.';
47:                     if (canPushBox(dir, j, i)) pushBox(dir, j, i);
48:                     switch (dir) {
49:                         // UP
50:                         case 0:
51:                             if (allGameStates[turn][i-1][j] == 'a') allGameS
tates[turn][i-1][j] = '2';
52:                             else
53:                                 allGameStates[turn][i-1][j] = '@';
54:                             break;
55:                         // LEFT
56:                         case 1:
57:                             if (allGameStates[turn][i][j-1] == 'a') allGameS
```

```
tates[turn][i][j-1] = '2';
58:                     else
59:                         allGameStates[turn][i][j-1] = '@';
60:                         break;
61:                         // DOWN
62:                         case 2:
63:                             if (allGameStates[turn][i+1][j] == 'a') allGameS
tates[turn][i+1][j] = '2';
64:                     else
65:                         allGameStates[turn][i+1][j] = '@';
66:                         break;
67:                         // RIGHT
68:                         case 3:
69:                             if (allGameStates[turn][i][j+1] == 'a') allGameS
tates[turn][i][j+1] = '2';
70:                     else
71:                         allGameStates[turn][i][j+1] = '@';
72:                         break;
73:                     }
74:                 return;
75:             }
76:         }
77:     }
78: }
79: }
80:
81: // Restarts the game when 'R' is pressed
82: void Sokoban::restart(sf::Clock& clock) {
83:     turn = 0;
84:     clock.restart();
85: }
86:
87: // Reverts back to the previous game state
88: void Sokoban::undo() { if (turn) turn--; }
89:
90: void Sokoban::playSound() {
91:     for (int i = 0; i < _h; i++) {
92:         for (int j = 0; j < _w; j++) {
93:             if (allGameStates[turn][i][j] == 'A' || allGameStates[turn][i]
][j] == '2') return;
94:         }
95:     }
96:     sf::SoundBuffer buffer;
97:     if (!buffer.loadFromFile("sokoban/victory.wav")) exit(1);
98:     sf::Sound sound(buffer);
99:     sound.play();
100:    while (sound.getStatus() == sf::Sound::Playing) {
101:        // Wait for the sound to finish playing
102:    }
103: }
104:
105: // Check if there is a wall or double boxes
106: bool Sokoban::noObstructions(Direction dir, int w, int h) const {
107:     switch (dir) {
108:         // UP
109:         case 0:
110:             // Out of game bounds
111:             if (!h) return false;
112:             // Wall
113:             if (allGameStates[turn][h-1][w] == '#') return false;
114:             // Full Storage
115:             if (allGameStates[turn][h-1][w] == '1') return false;
116:             // Double boxes / Box - wall / Box - storage / Box - EOL
117:             if (allGameStates[turn][h-1][w] == 'A' && (h == 1)) return false
;
;
```

```
118:         if (allGameStates[turn][h-1][w] == 'A' && (
119:             allGameStates[turn][h-2][w] == 'A' ||
120:             allGameStates[turn][h-2][w] == '#') ||
121:             allGameStates[turn][h-2][w] == '1')) return false;
122:         break;
123:     // LEFT
124:     case 1:
125:         // Out of game bounds
126:         if (!w) return false;
127:         // Full Storage
128:         if (allGameStates[turn][h][w-1] == '1') return false;
129:         // Check for wall
130:         if (allGameStates[turn][h][w-1] == '#') return false;
131:         // Double boxes / Box - wall / Box - storage / Box - EOL
132:         if (allGameStates[turn][h][w-1] == 'A' && (
133:             allGameStates[turn][h][w-2] == 'A' ||
134:             allGameStates[turn][h][w-2] == '#' ||
135:             allGameStates[turn][h][w-2] == '1' ||
136:             (w == 1))) return false;
137:         break;
138:     // DOWN
139:     case 2:
140:         // Out of game bounds
141:         if (!(h-h-1)) return false;
142:         // Full Storage
143:         if (allGameStates[turn][h+1][w] == '1') return false;
144:         // Wall
145:         if (allGameStates[turn][h+1][w] == '#') return false;
146:         // Double boxes / Box - wall / Box - storage / Box - EOL
147:         if (allGameStates[turn][h+1][w] == 'A' && (h+2 == _h)) return fa
lse;
148:         if (allGameStates[turn][h+1][w] == 'A' && (
149:             allGameStates[turn][h+2][w] == 'A' ||
150:             allGameStates[turn][h+2][w] == '#' ||
151:             allGameStates[turn][h+2][w] == '1')) return false;
152:         break;
153:     // RIGHT
154:     case 3:
155:         // Out of game bounds
156:         if (!(w-w-1)) return false;
157:         // Full Storage
158:         if (allGameStates[turn][h][w+1] == '1') return false;
159:         // Wall
160:         if (allGameStates[turn][h][w+1] == '#') return false;
161:         // Double boxes / Box - wall / Box - storage / Box - EOL
162:         if (allGameStates[turn][h][w+1] == 'A' && (
163:             allGameStates[turn][h][w+2] == 'A' ||
164:             allGameStates[turn][h][w+2] == '#' ||
165:             allGameStates[turn][h][w+2] == '1' ||
166:             (w+2 == _w))) return false;
167:         break;
168:     }
169:     return true;
170: }
171:
172: bool Sokoban::canPushBox(Direction dir, int w, int h) const {
173:     switch (dir) {
174:         // UP
175:         case 0:
176:             // Box - empty / Box - storage
177:             if (allGameStates[turn][h-1][w] == 'A' && (
178:                 allGameStates[turn][h-2][w] == '.' ||
179:                 allGameStates[turn][h-2][w] == 'a'))) return true;
180:             break;
181:         // LEFT
```

```
182:         case 1:
183:             // Box - empty / Box - storage
184:             if (allGameStates[turn][h][w-1] == 'A' && (
185:                 allGameStates[turn][h][w-2] == '.' ||
186:                 allGameStates[turn][h][w-2] == 'a')) return true;
187:             break;
188:             // DOWN
189:             case 2:
190:                 // Box - empty / Box - storage
191:                 if (allGameStates[turn][h+1][w] == 'A' && (
192:                     allGameStates[turn][h+2][w] == '.' ||
193:                     allGameStates[turn][h+2][w] == 'a')) return true;
194:                 break;
195:                 // RIGHT
196:                 case 3:
197:                     // Box - empty / Box - storage
198:                     if (allGameStates[turn][h][w+1] == 'A' && (
199:                         allGameStates[turn][h][w+2] == '.' ||
200:                         allGameStates[turn][h][w+2] == 'a')) return true;
201:                     break;
202:                 }
203:             return false;
204:         }
205:
206: void Sokoban::pushBox(Direction dir, int w, int h) {
207:     switch (dir) {
208:         // UP
209:         case 0:
210:             if (allGameStates[turn][h-2][w] == '.') allGameStates[turn][h-2]
[w] = 'A';
211:             if (allGameStates[turn][h-2][w] == 'a') allGameStates[turn][h-2]
[w] = '1';
212:             break;
213:             // LEFT
214:             case 1:
215:                 if (allGameStates[turn][h][w-2] == '.') allGameStates[turn][h][w
-2] = 'A';
216:                 if (allGameStates[turn][h][w-2] == 'a') allGameStates[turn][h][w
-2] = '1';
217:                 break;
218:                 // DOWN
219:                 case 2:
220:                     if (allGameStates[turn][h+2][w] == '.') allGameStates[turn][h+2]
[w] = 'A';
221:                     if (allGameStates[turn][h+2][w] == 'a') allGameStates[turn][h+2]
[w] = '1';
222:                     break;
223:                     // RIGHT
224:                     case 3:
225:                         if (allGameStates[turn][h][w+2] == '.') allGameStates[turn][h][w
+2] = 'A';
226:                             if (allGameStates[turn][h][w+2] == 'a') allGameStates[turn][h][w
+2] = '1';
227:                             break;
228:                         return;
229:                     }
230:     }
231:
232: // Overload the virtual draw function:
233: // Load required textures and display in window at prescribed locations
234: void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
235:     sf::Texture Wall, Box, WinBox, Empty, Storage, ManDown, ManUp, ManLeft,
ManRight;
236:     if (!Wall.loadFromFile("sokoban/block_06.png")) exit(1);
```

```
237:     if (!Box.loadFromFile("sokoban/crate_03.png")) exit(1);
238:     if (!WinBox.loadFromFile("sokoban/crate_04.png")) exit(1);
239:     if (!Empty.loadFromFile("sokoban/ground_01.png")) exit(1);
240:     if (!Storage.loadFromFile("sokoban/environment_03.png")) exit(1);
241:     if (!ManDown.loadFromFile("sokoban/player_05.png")) exit(1);
242:     if (!ManUp.loadFromFile("sokoban/player_08.png")) exit(1);
243:     if (!ManLeft.loadFromFile("sokoban/player_20.png")) exit(1);
244:     if (!ManRight.loadFromFile("sokoban/player_17.png")) exit(1);
245:
246:     for (int i = 0; i < _h; i++) {
247:         for (int j = 0; j < _w; j++) {
248:             sf::RectangleShape tile;
249:             tile.setSize(sf::Vector2f(TILE_SIZE, TILE_SIZE));
250:             tile.setPosition(j * TILE_SIZE, i * TILE_SIZE);
251:             // Draw wall tile
252:             if (allGameStates[turn][i][j] == '#') {
253:                 tile.setTexture(&Wall);
254:             // Draw empty tile
255:             } else if (allGameStates[turn][i][j] == '.') {
256:                 tile.setTexture(&Empty);
257:             // Draw background tile and overlay
258:             } else {
259:                 sf::RectangleShape backTile;
260:                 backTile.setSize(sf::Vector2f(TILE_SIZE, TILE_SIZE));
261:                 backTile.setPosition(j * TILE_SIZE, i * TILE_SIZE);
262:                 backTile.setTexture(&Empty);
263:                 target.draw(backTile);
264:                 switch (allGameStates[turn][i][j]) {
265:                     case 'a':
266:                         tile.setTexture(&Storage);
267:                         break;
268:                     case 'A':
269:                         tile.setTexture(&Box);
270:                         break;
271:                     case '1':
272:                         tile.setTexture(&WinBox);
273:                         break;
274:                     default:
275:                         // Based on face turn, select appropriate texture
276:                         if (face[turn] == DOWN) tile.setTexture(&ManDown);
277:                         if (face[turn] == UP) tile.setTexture(&ManUp);
278:                         if (face[turn] == LEFT) tile.setTexture(&ManLeft);
279:                         if (face[turn] == RIGHT) tile.setTexture(&ManRight);
280:                         break;
281:                     }
282:                 }
283:                 target.draw(tile);
284:             }
285:         }
286:     }
287:
288: // Function that displays time in upper-left corner
289: void Sokoban::drawElapsingTime(sf::RenderWindow &window, sf::Clock &clock
) {
290:     sf::Time elapsed = clock.getElapsedTime();
291:     int minutes = elapsed.asSeconds() / 60;
292:     int seconds = static_cast<int>(elapsed.asSeconds()) % 60;
293:
294:     std::string timeString = std::to_string(minutes) + ":" +
295:     (seconds < 10 ? "0" : "") + std::to_string(seconds);
296:
297:     sf::Font font;
298:     font.loadFromFile("sokoban/arial.ttf");
299:     sf::Text timeText(timeString, font, 30);
300:     timeText.setFillColor(sf::Color::White);
```

```
301:         window.draw(timeText);
302:     }
303: }
304:
305: // non-member helper functions:
306: // performs a deep copy of the game state
307: vector<vector<char>> deepCopy(const vector<vector<char>> &original) {
308:     vector<vector<char>> copy;
309:     for (const auto &row : original) copy.push_back(row);
310:     return copy;
311: }
```

# **PS3 Pythagorean Tree**

## **1 Description of the assignment**

For this assignment I created a PTree class that recursively drew a Pythagorean tree of pixel size L and depth of N. A Pythagoras tree is a plane fractal constructed from squares whose points in connection to other squares enclose a right triangle.

## **2 Key algorithms, Data structures, or OO designs**

The key algorithm used to draw the tree was the PTree function defined at line 26 of PTree.cpp. The algorithm processes the points where new squares should be drawn. Given the angle of the parent square, the angles of the child squares are calculated and rotated accordingly. Based off of the two new child squares, 4 sets of points are calculated: the left child set and right child set. The PTree function is then recursively called on the left and right sets of points until it has reached the maximum depth N. Each depth is set to a different color.

## **3 What I learned from the project**

In this assignment I learned to delegate some of the work to an algorithm rather than static storage variables in my class type. The only variables of concern for the drawing of the PTree are the depth N and the length L. All other elements of the tree can be calculated from these two variables: no internal storage structure needed.

## 4 Screenshot of program output

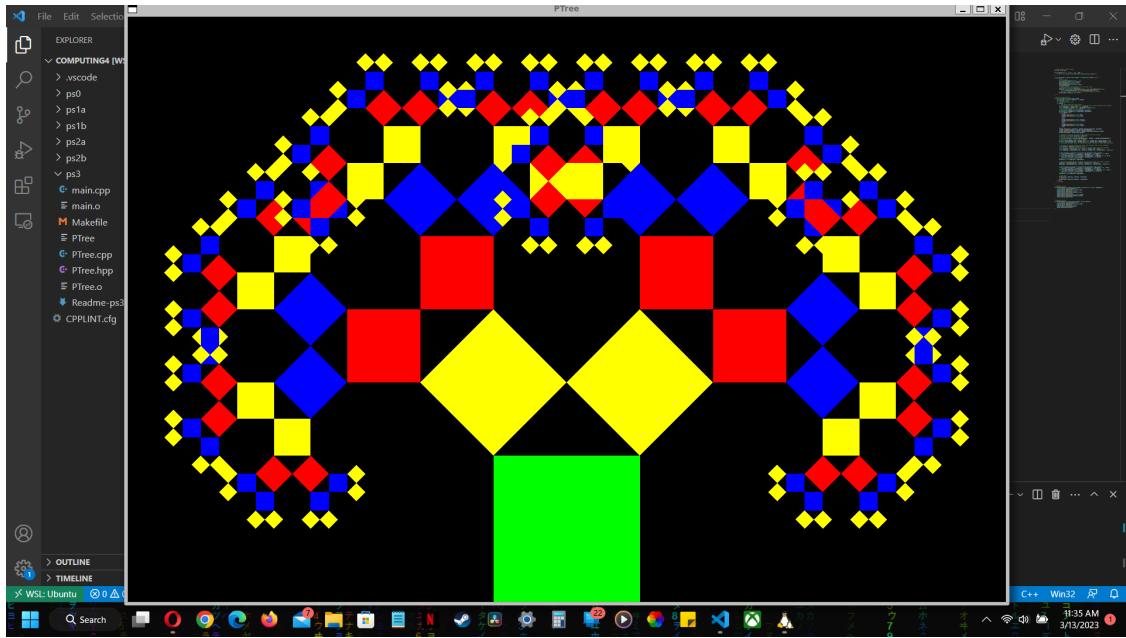


Figure 6: ps3 tree output

## 5 Issues and bugs

No issues or bugs present.

The program was fully functional within the given specification.

## 6 Source code for the project

**Makefile**            **Mon Mar 13 11:20:14 2023**            **1**

```
1: CC = g++
2: FFLAGS = -g -Wall -Werror -pedantic -std=c++17
3: LIB = -lsfml-system -lsfml-graphics -lsfml-window
4: DEPS = PTree.o
5:
6: .PHONY: all clean lint
7:
8: all: lint PTree
9:
10: PTree: main.o PTree.o $(DEPS)
11:       $(CC) $(FLAGS) -o PTree $^ $(LIB)
12:
13: %.o: %.cpp
14:       $(CC) $(FLAGS) -c $<
15:
16: lint:
17:       cpplint --recursive --quiet .
18:
19: clean:
20:       rm *.o PTree
```

```
main.cpp      Fri Mar 03 20:32:11 2023      1
1: // Copyright 2023 Thomas O'Connor
2: #include "PTree.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     // Command line arguments
6:     double lengthL = atof(argv[1]);
7:     int depthN = atoi(argv[2]);
8:
9:     // Create tree and input values from cmd line args
10:    PTree tree(lengthL, depthN);
11:
12:    // Rednder the window using the appropriate game dimensions
13:    sf::RenderWindow window(sf::VideoMode(tree.getLengthL() * 6, tree.get
LengthL() * 4), "PTree");
14:    window.setFramerateLimit(120);
15:
16:    while (window.isOpen()) {
17:        // Process events
18:        sf::Event event;
19:        while (window.pollEvent(event)) {
20:            // Close window: exit
21:            if (event.type == sf::Event::Closed) window.close();
22:        }
23:        // Clear screen
24:        window.clear();
25:        window.draw(tree);
26:        window.display();
27:        if (sf::Keyboard::isKeyPressed(sf::Keyboard::X)) window.close();
28:    }
29:    return 0;
30: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef PTREE_HPP
3: #define PTREE_HPP
4:
5: #include <cmath>
6: #include <iostream>
7: #include <utility>
8: #include <SFML/Graphics.hpp>
9: #include <SFML/System.hpp>
10: #include <SFML/Window.hpp>
11:
12: using std::cout;
13: using std::endl;
14: using std::pair;
15: using sf::RectangleShape;
16: using sf::Vector2f;
17:
18: class PTree : public sf::Drawable, sf::Transformable {
19: public:
20:     // Constructors
21:     PTree() : _L(0), _N(0) {}
22:     PTree(double L, int N);
23:
24:     // Getters
25:     int getDepthN() const { return _N; }
26:     double getLengthL() const { return _L; }
27:
28: private:
29:     // Draw game in SFML
30:     void draw(sf::RenderTarget& target, sf::RenderStates states) const override;
31:     // Recursive draw function
32:     void pTree(sf::RenderTarget& target,
33:               pair<Vector2f, Vector2f> newPoints, int depthN,
34:               float angleR) const;
35: private:
36:     double _L;
37:     int _N;
38: };
39:
40: // Debugger functions
41: void drawIndicators(sf::RenderTarget& target, pair<Vector2f, Vector2f> newPoints);
42: void drawIndicator(sf::RenderTarget& target, Vector2f point);
43:
44: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "PTree.hpp"
3:
4: PTree::PTree(double L, int N) : _L(L), _N(N) {
5:     if (N < 1 || L <= 0.0) throw std::out_of_range("Invalid arguments");
6: }
7:
8: void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    9:     if (_N) {
10:         // first square
11:         RectangleShape base(Vector2f(_L, _L));
12:         base.setFillColor(sf::Color::Green);
13:         base.setOrigin(Vector2f(_L/2, _L));
14:         base.setPosition(Vector2f(_L*3, _L*4));
15:         target.draw(base);
16:         // create a pair of Vector2f that locate the upper points
17:         pair<Vector2f, Vector2f> newPoints;
18:         newPoints = pair(base.getPosition().x-_L/2, base.getPosition().y-_L),
19:                     Vector2f(base.getPosition().x+_L/2, base.getPosition().y-_L))
;
20:         // call the recursive draw function with decremented depth and 0
angle
21:         pTree(target, newPoints, _N-1, 0.0f);
22:     }
23: }
24:
25: // Recursive draw function
26: void PTree::pTree(sf::RenderTarget& target,
27:                   pair<Vector2f, Vector2f> newPoints, int depthN,
28:                   float angleR) const {
29:     // if more depth exists: draw
30:     if (depthN) {
31:         // Find object length and object angles
32:         // drawIndicators(target, newPoints); // shows the points where n
ew squares will be drawn
33:         double objLength = _L*pow(sqrt(2) / 2, _N-depthN);
34:         float leftAngleR = angleR - 45.f, rightAngleR = angleR + 45.f;
35:         // Draw two rectangles given the newPoints
36:         RectangleShape leftObj(Vector2f(objLength, objLength));
37:         RectangleShape rightObj(Vector2f(objLength, objLength));
38:         switch (depthN % 3) {
39:             case 0:
40:                 leftObj.setFillColor(sf::Color::Red);
41:                 rightObj.setFillColor(sf::Color::Red);
42:                 break;
43:             case 1:
44:                 leftObj.setFillColor(sf::Color::Yellow);
45:                 rightObj.setFillColor(sf::Color::Yellow);
46:                 break;
47:             case 2:
48:                 leftObj.setFillColor(sf::Color::Blue);
49:                 rightObj.setFillColor(sf::Color::Blue);
50:                 break;
51:         }
52:         leftObj.setOrigin(0, objLength); rightObj.setOrigin(objLength, ob
jLength);
53:         leftObj.rotate(leftAngleR); rightObj.rotate(rightAngleR);
54:         leftObj.setPosition(newPoints.first); rightObj.setPosition(newPoi
nts.second);
55:         target.draw(leftObj); target.draw(rightObj);
56:
57:         // Use given information to calculate the location of the next 4
points
```

```
58:         pair<Vector2f, Vector2f> newPointsL, newPointsR;
59:         // Get the global bounds of the rectangles
60:         sf::FloatRect boundsL = leftObj.getGlobalBounds(), boundsR = rightObj.getGlobalBounds();
61:         // Calculate the center point of the rectangles
62:         Vector2f centerL(boundsL.left + boundsL.width / 2.f, boundsL.top +
63:                           boundsL.height / 2.f);
64:         Vector2f centerR(boundsR.left + boundsR.width / 2.f, boundsR.top +
65:                           boundsR.height / 2.f);
66:         // drawIndicator(target, centerL); drawIndicator(target, centerR)
67:         ; // shows the centerpoints
68:         // Calculate the distance between the center point and each corner
69:         // of the rectangles
70:         float distance = objLength * sqrt(2.f) / 2.f;
71:         // Calculate the angle between the center point and each corner
72:         // of the left rectangle
73:         float angleLeft = atan2(boundsL.top - centerL.y, boundsL.left - centerL.x);
74:         float angleRight = atan2(boundsL.top - centerL.y, boundsL.left +
75:                                   boundsL.width - centerL.x);
76:         // Add the angle of rotation and calculate the coordinates after
77:         // rotation
78:         Vector2f upperLeftL(centerL.x + distance * cos(angleLeft + leftAngleR * M_PI / 180.f),
79:                           centerL.y + distance * sin(angleLeft + leftAngleR * M_PI / 180.f));
80:         Vector2f upperRightL(centerL.x + distance * cos(angleRight + leftAngleR * M_PI / 180.f),
81:                               centerL.y + distance * sin(angleRight + leftAngleR * M_PI / 180.f));
82:         // input these new coordinates into pair
83:         newPointsL = pair(upperLeftL, upperRightL);
84:         // Calculate the angle between the center point and the corners
85:         // of the other rectangle
86:         angleLeft = atan2(boundsR.top - centerR.y, boundsR.left - centerR.x);
87:         angleRight = atan2(boundsR.top - centerR.y, boundsR.left + boundsR.width - centerR.x);
88:         // Add the angle of rotation and calculate the coordinates after
89:         // rotation
90:         Vector2f upperLeftR(centerR.x + distance * cos(angleLeft + rightAngleR * M_PI / 180.f),
91:                           centerR.y + distance * sin(angleLeft + rightAngleR * M_PI / 180.f));
92:         Vector2f upperRightR(centerR.x + distance * cos(angleRight + rightAngleR * M_PI / 180.f),
93:                               centerR.y + distance * sin(angleRight + rightAngleR * M_PI / 180.f));
94:         // input these new coordinates into pair
95:         newPointsR = pair(upperLeftR, upperRightR);
96:         // Left trees
97:         pTree(target, newPointsL, depthN-1, leftAngleR);
98:         // Right trees
99:         pTree(target, newPointsR, depthN-1, rightAngleR);
100:    } else {
101:        return;
```

```
102: void drawIndicators(sf::RenderTarget& target, pair<Vector2f, Vector2f> newPoints) {
103:     sf::CircleShape indicatorPoint(5.f);
104:     indicatorPoint.setFillColor(sf::Color::Red);
105:     indicatorPoint.setOrigin(5.f, 5.f);
106:     indicatorPoint.setPosition(newPoints.first);
107:     target.draw(indicatorPoint);
108:     indicatorPoint.setFillColor(sf::Color::Green);
109:     indicatorPoint.setPosition(newPoints.second);
110:     target.draw(indicatorPoint);
111: }
112:
113: // Debugger function
114: void drawIndicator(sf::RenderTarget& target, Vector2f point) {
115:     sf::CircleShape indicatorPoint(5.f);
116:     indicatorPoint.setFillColor(sf::Color::Blue);
117:     indicatorPoint.setOrigin(5.f, 5.f);
118:     indicatorPoint.setPosition(point);
119:     target.draw(indicatorPoint);
120: }
```

# PS4 Checkers

## 1 Description of the assignment

For part (a) of the assignment I created a Checkers UI that inherited the virtual draw() function from the sf::Drawable class. This Checkers class used an initializeBase() method defined at line 135 of Checkers.cpp to set the board and pieces to the default opening game configuration: 12 black pieces spread evenly on every black tile in the first three rows, and 12 red pieces spread evenly on every black tile over the last three rows. I defined two global variables: TILE\_SIZE of size 64 for the pixel size of the tiles, and BOARD\_DIMENSIONS of size 8 for the 8x8 dimensions of the game board. For extra credit I added an additional game border using a wood texture 32 pixels in depth. In part (a) I was also tasked with the selection of game pieces based on player turns. I created a mouseInGameBounds() helper function which determined if a mouse click was within the bounds of the game board including this new border texture. Using this helper function my selectPiece() method checked for the selection of a valid game piece by referencing the game data structure, the current player turn, and the number of non-selected pieces on the game board. Additionally I created a temporary switchTurn() method which artificially toggled the players' turns. The default first player was black, and when the end user pressed (T) this was toggled to red. Based on the current player's turn the end user could only select pieces of the current player. Clicking elsewhere would de-select a piece. At line 119 of Checkers.cpp I added another extra credit element: an indicator for the end user displaying the current color's turn. Using the arial font an uppercase R or B was displayed in the upper left hand corner of the game board.

For part (b) of the assignment I implemented the Checkers gameplay mechanics. This included a movePiece() method, a visual move assist, and a win condition with fanfare. I defined a new global variable: BOARD\_OFFSET of size 32 representing the game border. To move selected pieces I implemented a movePiece() method defined at line 88 of Checkers.cpp. This function manually checks the diagonal directions for moves and jumps, and completes the action if the mouse click was at one of the available end locations. At the end of this movement the finishLine() method is called which checks for the automatic crowning of pawns at the opponent's end of the board. In addition to the movePiece() method, the visualMoveAssist() method defined at line 292 uses the same diagonal direction checking and displays a star diagonal to the selected piece on moveable tiles using the drawStar() helper function. The win fanfare was triggered by the isWon() method which checked for any remaining pieces or selected pieces, or triggered a win if no pieces could move. This then triggered an execution of code in main.cpp at line 43 which played a win sound, displayed the color of the winning team in text, and disabled player movement. Afterwards or during the game, the end user could press (X) to close the window or (R) to restart the game. The restart function simply called initializeBase() on the game object and reset the win condition Boolean variable.

## 2 Key algorithms, Data structures, or OO designs

In part (a) the data structure I used was a 2D vector of characters named currentGameState.

The following symbols were used in the 2D game state vector:

- . A red background tile. Movement here is invalid.
- p** A black background tile. These squares are “playable”.
- b** A black pawn.
- r** A red pawn.
- w** A selected pawn
- B** A black king.
- R** A red king.
- W** A selected king.

The initializeBase() method went through the 2D vector placing a red background tile on every other element, and based on the line placed either a black piece, red piece, or playable black background tile.

My “controller” that translated global mouse coordinates to 2D vector coordinates was performed at line 26 of Checkers.cpp. This initializer converted mouseLocation coordinates to arenaLocation coordinates by removing the border size and dividing by the tile size for each dimension. This resulted in a coordinate valid within my currentGameState matrix.

In part (b) I used the same 2D character matrix as in part (a), but added two internal Boolean variables: stillPlaying and setWinTrue. These Boolean variables were used for edge conditions around the win conditions. stillPlaying was used primarily in the movePiece() method to ensure the game did not end prematurely when a piece was selected, while setWinTrue was used to expedite the win condition when a the final selected piece could not move in any direction. These variables could have been offloaded to boundary checking functions but were easily implemented when stored in the internal game state.

Throughout the implementation file I use the getter function getSelectedPawn() which uses two algorithms (`std::find_if` and `std::distance`) alongside three lambda expressions to parse through my matrix and return the internal coordinates of the selected pawn. I could have completed this through traditional for loops but I wanted to challenge myself to use advanced coding algorithms to accomplish my end goals, even if it was just for practice.

## 3 What I learned from the project

In part (a) of the assignment I learned the consequences of complicating my interior data storage structure. I chose to implement the red tiles in the matrix even though they would always be invalid tiles, never read or overwritten after initialization. This helped with conceptualization and debugging of the game board, since the interior state matched the real-life representation of the board. When moving on to part (b) however, this would result in complications. When performing checks on adjacent playable tiles I would always have to account for the 1x1 offset created by the red background tiles in all directions. This meant for more complex calculations and issues with the visual move assist.

Over the course of part (b) of the assignment I learned to keep my functions simple and straightforward. I should have offloaded some of the work in movePiece() and visualMoveAssist() to helper functions that check diagonal boundaries. Because I did not use helper functions here, these methods were hundreds of lines long despite checking the exact same boundaries. I could have subdivided these checks in a variety of ways, but the brute force approach was what I chose to do despite of the obvious drawbacks.

## 4 Screenshot of program output

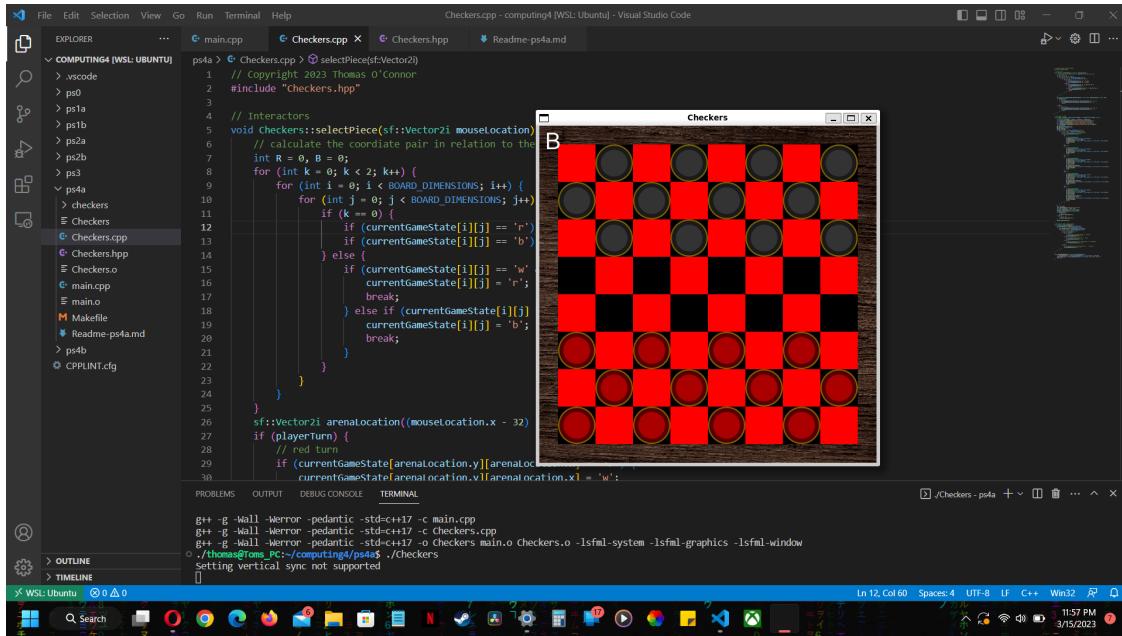


Figure 7: ps4 game output

## **5 Issues and bugs**

Pieces could not double jump.

Pieces that could jump were not required to jump.

If multiple pieces on the same team were all deadlocked,

win condition would never be met.

Occasionally, textures flickered.

Otherwise, the program was functional within the given specification.

## **6 Source code for the project**

**Makefile**        **Wed Mar 15 11:04:39 2023**        **1**

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

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Checkers.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     // Create game
6:     Checkers Game;
7:     // Rednder the window using the appropriate game dimensions
8:     sf::RenderWindow window
9:         (sf::VideoMode(TILE_SIZE * BOARD_DIMENSIONS + TILE_SIZE,
10:             TILE_SIZE * BOARD_DIMENSIONS + TILE_SIZE), "Checkers");
11:    window.setFramerateLimit(120);
12:
13:    while (window.isOpen()) {
14:        // Process events
15:        sf::Event event;
16:        while (window.pollEvent(event)) {
17:            // Close window: exit
18:            if (event.type == sf::Event::Closed) window.close();
19:        }
20:        // Clear screen
21:        window.clear();
22:        // Draw the basic gameboard
23:        window.draw(Game);
24:        // Detect button press
25:        if (sf::Mouse::isButtonPressed(sf::Mouse::Left) &&
26:            mouseInGameBounds(sf::Mouse::getPosition(window)))
27:            Game.selectPiece(sf::Mouse::getPosition(window));
28:        window.display();
29:        // Exit on (X) keypress
30:        if (sf::Keyboard::isKeyPressed(sf::Keyboard::X)) window.close();
31:        // Switch player turns
32:        if (sf::Keyboard::isKeyPressed(sf::Keyboard::T)) Game.switchTurn(
);
33:    }
34:    return 0;
35: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef CHECKERS_HPP
3: #define CHECKERS_HPP
4:
5: #include <fstream>
6: #include <iostream>
7: #include <string>
8: #include <vector>
9: #include <SFML/Graphics.hpp>
10: #include <SFML/System.hpp>
11: #include <SFML/Window.hpp>
12:
13: using std::cout;
14: using std::endl;
15: using std::vector;
16: using sf::Vector2f;
17:
18: #define TILE_SIZE 64
19: #define BOARD_DIMENSIONS 8
20:
21: class Checkers : public sf::Drawable {
22: public:
23:     // Constructors
24:     Checkers() { initializeBase(); }
25:
26:     // Interactors
27:     void selectPiece(sf::Vector2i mouseLocation);
28:     void switchTurn(void) { playerTurn = !playerTurn; }
29: private:
30:     // Draw game in SFML
31:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
32:     // Initialize game storage vectors
33:     void initializeBase();
34: private:
35:     vector<vector<char>> currentGameState;
36:     bool playerTurn = 0;
37: };
38:
39: // Helper functions
40: bool mouseInGameBounds(sf::Vector2i mouseLocation);
41:
42: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Checkers.hpp"
3:
4: // Interactors
5: void Checkers::selectPiece(sf::Vector2i mouseLocation) {
6:     // calculate the coordinate pair in relation to the 2D char array
7:     int R = 0, B = 0;
8:     for (int k = 0; k < 2; k++) {
9:         for (int i = 0; i < BOARD_DIMENSIONS; i++) {
10:             for (int j = 0; j < BOARD_DIMENSIONS; j++) {
11:                 if (k == 0) {
12:                     if (currentGameState[i][j] == 'r') R++;
13:                     if (currentGameState[i][j] == 'b') B++;
14:                 } else {
15:                     if (currentGameState[i][j] == 'w' && R == 11) {
16:                         currentGameState[i][j] = 'r';
17:                         break;
18:                     } else if (currentGameState[i][j] == 'w' && B == 11)
19:                         currentGameState[i][j] = 'b';
20:                     break;
21:                 }
22:             }
23:         }
24:     }
25: }
26: sf::Vector2i arenaLocation((mouseLocation.x - 32) / 64, (mouseLocatio
n.y - 32) / 64);
27: if (playerTurn) {
28:     // red turn
29:     if (currentGameState[arenaLocation.y][arenaLocation.x] == 'r') {
30:         currentGameState[arenaLocation.y][arenaLocation.x] = 'w';
31:         return;
32:     }
33: } else {
34:     // black turn
35:     if (currentGameState[arenaLocation.y][arenaLocation.x] == 'b') {
36:         currentGameState[arenaLocation.y][arenaLocation.x] = 'w';
37:         return;
38:     }
39: }
40: }
41:
42: // Draw game in SFML
43: void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst {
44:     // Load piece textures
45:     sf::Texture blackPiece, redPiece, whitePiece, woodBacking;
46:     if (!blackPiece.loadFromFile("checkers/blackpawn.png")) exit(1);
47:     if (!redPiece.loadFromFile("checkers/redpawn.png")) exit(1);
48:     if (!whitePiece.loadFromFile("checkers/whitepawn.png")) exit(1);
49:     if (!woodBacking.loadFromFile("checkers/wood_texture.png")) exit(1);
50:     sf::Sprite WB;
51:     WB.setTexture(woodBacking);
52:     WB.setScale(1, 1.2);
53:     target.draw(WB);
54:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
55:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
56:             // Draw states stored in currentGameState
57:             switch (currentGameState[i][j]) {
58:                 case '.':
59:                 {
60:                     sf::RectangleShape R;
61:                     R.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
62:                     R.setFillColor(sf::Color::Red);
```

```

Checkers.cpp      Wed Mar 15 21:11:10 2023      2

63:                         R.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_S
IZE + TILE_SIZE/2);
64:                         target.draw(R);
65:                         break;
66:                     }
67:                     case 'p':
68:                     {
69:                         sf::RectangleShape B;
70:                         B.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
71:                         B.setFillColor(sf::Color::Black);
72:                         B.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_S
IZE + TILE_SIZE/2);
73:                         target.draw(B);
74:                         break;
75:                     }
76:                     case 'r':
77:                     {
78:                         sf::RectangleShape B;
79:                         B.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
80:                         B.setFillColor(sf::Color::Black);
81:                         B.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_S
IZE + TILE_SIZE/2);
82:                         target.draw(B);
83:                         sf::Sprite RP;
84:                         RP.setTexture(redPiece);
85:                         RP.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_
SIZE + TILE_SIZE/2);
86:                         target.draw(RP);
87:                         break;
88:                     }
89:                     case 'b':
90:                     {
91:                         sf::RectangleShape B;
92:                         B.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
93:                         B.setFillColor(sf::Color::Black);
94:                         B.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_S
IZE + TILE_SIZE/2);
95:                         target.draw(B);
96:                         sf::Sprite BP;
97:                         BP.setTexture(blackPiece);
98:                         BP.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_
SIZE + TILE_SIZE/2);
99:                         target.draw(BP);
100:                        break;
101:                    }
102:                    case 'w':
103:                    {
104:                        sf::RectangleShape B;
105:                        B.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
106:                        B.setFillColor(sf::Color::Black);
107:                        B.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_S
IZE + TILE_SIZE/2);
108:                        target.draw(B);
109:                        sf::Sprite WP;
110:                        WP.setTexture(whitePiece);
111:                        WP.setScale(0.6, 0.6);
112:                        WP.setPosition(j * TILE_SIZE + TILE_SIZE/2, i * TILE_
SIZE + TILE_SIZE/2);
113:                        target.draw(WP);
114:                        break;
115:                    }
116:                }
117:            }
118:        }
119:    sf::Font font;

```

```
120:     font.loadFromFile("checkers/arial.ttf");
121:     sf::Text turnText("", font, 40);
122:     turnText.setFillColor(sf::Color::White);
123:     turnText.setPosition(Vector2f(10, 0));
124:     if (playerTurn) {
125:         // Red turn
126:         turnText.setString("R");
127:     } else {
128:         // Black turn
129:         turnText.setString("B");
130:     }
131:     target.draw(turnText);
132: }
133:
134: // Initialize game storage vectors
135: void Checkers::initializeBase() {
136:     currentGameState.resize(BOARD_DIMENSIONS);
137:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
138:         currentGameState[i].resize(BOARD_DIMENSIONS);
139:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
140:             // Draw red background tiles
141:             if (((j + (i%2))%2)) {
142:                 if (i <= 2) {
143:                     // Black piece represented as 'b' (black)
144:                     currentGameState[i][j] = 'b';
145:                 } else if (i >= 5) {
146:                     // Red piece represented as 'r' (red)
147:                     currentGameState[i][j] = 'r';
148:                 } else {
149:                     // Black background tile represented 'p' (playable)
150:                     currentGameState[i][j] = 'p';
151:                 }
152:             } else {
153:                 // Red background tile represented as '.' (invalid)
154:                 currentGameState[i][j] = '.';
155:             }
156:         }
157:     }
158: }
159:
160: bool mouseInGameBounds(sf::Vector2i mouseLocation) {
161:     if ((mouseLocation.x >= 32 && mouseLocation.x <= TILE_SIZE * 8 + 32)
&&
162:         (mouseLocation.y >= 32 && mouseLocation.y <= TILE_SIZE * 8 + 32)) ret
urn 1;
163:     return 0;
164: }
```

```
Makefile      Wed Mar 22 20:29:17 2023      1
1: CC = g++
2: FLAGS = -g -Wall -Werror -pedantic -std=c++17
3: LIB = -lsfml-system -lsfml-graphics -lsfml-window -lsfml-audio
4: DEPS = Checkers.o
5:
6: .PHONY: all clean lint
7:
8: all: lint Checkers
9:
10: Checkers: main.o Checkers.o $(DEPS)
11:           $(CC) $(FLAGS) -o Checkers $^ $(LIB)
12:
13: %.o: %.cpp
14:           $(CC) $(FLAGS) -c $<
15:
16: lint:
17:           cpplint --recursive --quiet .
18:
19: clean:
20:           rm *.o Checkers
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Checkers.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     // Create game
6:     Checkers Game;
7:     bool performWinConditionOnce = 1;
8:     // Rednder the window using the appropriate game dimensions
9:     sf::RenderWindow window
10:    (sf::VideoMode(TILE_SIZE * BOARD_DIMENSIONS + TILE_SIZE,
11:                  TILE_SIZE * BOARD_DIMENSIONS + TILE_SIZE), "Checkers");
12:    window.setFramerateLimit(120);
13:
14:    while (window.isOpen()) {
15:        // Process events
16:        sf::Event event;
17:        while (window.pollEvent(event)) {
18:            // Close window: exit
19:            if (event.type == sf::Event::Closed) window.close();
20:        }
21:        if (!Game.isWon()) {
22:            // Get Keyboard input and impliment correct movement
23:            window.clear();
24:            // Detect button press
25:            if (sf::Mouse::isButtonPressed(sf::Mouse::Left) &&
26:                mouseInGameBounds(sf::Mouse::getPosition(window))) {
27:                if (Game.nothingSelected())
28:                    Game.selectPiece(sf::Mouse::getPosition(window));
29:                else
30:                    Game.movePiece(sf::Mouse::getPosition(window));
31:            }
32:            // Deselect piece
33:            if (sf::Mouse::isButtonPressed(sf::Mouse::Right) &&
34:                mouseInGameBounds(sf::Mouse::getPosition(window)))
35:                Game.deselectPiece();
36:            // Draw the basic gameboard
37:            window.draw(Game);
38:            // if piece selected, draw move assist
39:            if (!Game.nothingSelected())
40:                Game.visualMoveAssist(window);
41:            window.display();
42:            // if you've won, perform win fanfare
43:        } else if (performWinConditionOnce) {
44:            // Draw the win state
45:            std::string winString;
46:            if (Game.getWinner())
47:                winString = "Red wins";
48:            else
49:                winString = "Black wins";
50:            sf::Font font;
51:            font.loadFromFile("checkers/arial.ttf");
52:            sf::Text winText(winString, font, 30);
53:            winText.setFillColor(sf::Color::White);
54:            // Centers text to middle of screen
55:            winText.setPosition(sf::Vector2f(TILE_SIZE * 3.5, TILE_SIZE *
3.9));
56:            window.draw(winText);
57:            window.display();
58:            // Play win sound after screen display
59:            Game.playSound();
60:            // Ensures win sound is only performed once
61:            performWinConditionOnce = 0;
62:        }
63:        // Exit on (X) keypress
64:        if (sf::Keyboard::isKeyPressed(sf::Keyboard::X)) window.close();
```

```
main.cpp      Fri Mar 24 14:45:24 2023      2
 65:          // Restart game on (R) keypress
 66:          if (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) Game.restart (per
formWinConditionOnce);
 67:      }
 68:      return 0;
 69: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef CHECKERS_HPP
3: #define CHECKERS_HPP
4:
5: #include <algorithm>
6: #include <fstream>
7: #include <iostream>
8: #include <string>
9: #include <vector>
10: #include <SFML/Audio.hpp>
11: #include <SFML/Graphics.hpp>
12: #include <SFML/System.hpp>
13: #include <SFML/Window.hpp>
14:
15: using std::cout;
16: using std::endl;
17: using std::vector;
18: using sf::Vector2f;
19:
20: #define TILE_SIZE 64
21: #define BOARD_DIMENSIONS 8
22: #define BOARD_OFFSET 32
23:
24: class Checkers : public sf::Drawable {
25: public:
26:     // Constructors
27:     Checkers() { initializeBase(); }
28:
29:     // Getters
30:     bool nothingSelected(void) { return !stillPlaying; }
31:     bool isWon(void);
32:     bool getWinner(void);
33:     sf::Vector2i getSelectedPawn(void);
34:
35:     // Interactors
36:     void selectPiece(sf::Vector2i mouseLocation);
37:     void movePiece(sf::Vector2i mouseLocation);
38:     void deselectPiece(void);
39:     void switchTurn(void) { playerTurn = !playerTurn; }
40:     void restart(bool& winCondition);
41:
42:     // Performers
43:     void playSound(void);
44:     void visualMoveAssist(sf::RenderTarget& target);
45:     void drawStar(sf::RenderTarget& target, sf::Texture star, int yc, int xc);
46:
47: private:
48:     // Draw game in SFML
49:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
50:     // Initialize game storage vectors
51:     void initializeBase(void);
52:     // Automatically king pawns at respective finish lines
53:     void finishLine(void);
54:
55: private:
56:     // 2D array that stores the current game state
57:     vector<vector<char>> currentState;
58:     // player turn - 0 is black 1 is red
59:     bool playerTurn = 0;
60:     // if still playing, don't end the game prematurely
61:     bool stillPlaying = 0;
62:     // if unable to move final piece, set win to true
63:     bool setWinTrue = 0;
```

```
64: };
65:
66: // Helper functions
67: bool mouseInGameBounds(sf::Vector2i mouseLocation);
68: void drawBackingRectangle(sf::RenderTarget& target, int x, int y);
69:
70: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "Checkers.hpp"
3:
4: // Getter
5: bool Checkers::isWon(void) {
6:     if (setWinTrue) return 1;
7:     bool r = 0, b = 0;
8:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
9:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
10:             if (currentGameState[i][j] == 'r' ||
11:                 currentGameState[i][j] == 'R')
12:                 r = 1;
13:             if (currentGameState[i][j] == 'b' ||
14:                 currentGameState[i][j] == 'B')
15:                 b = 1;
16:             if ((r && b) || stillPlaying) return 0;
17:         }
18:     }
19:     return 1;
20: }
21:
22: // Getter (ONLY CALL AFTER ISWON IS CONFIRMED)
23: bool Checkers::getWinner(void) {
24:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
25:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
26:             // if a red piece exists, red wins
27:             if (currentGameState[i][j] == 'r' ||
28:                 currentGameState[i][j] == 'R')
29:                 return 1;
30:         }
31:     }
32:     // else black wins
33:     return 0;
34: }
35:
36: // Interactor
37: void Checkers::selectPiece(sf::Vector2i mouseLocation) {
38:     // calculate the coordinate pair in relation to the 2D char array
39:     sf::Vector2i arenaLocation((mouseLocation.x - BOARD_OFFSET) / TILE_SIZE,
ZE,
40:                             (mouseLocation.y - BOARD_OFFSET) / TILE_SIZE);
41:     if (playerTurn) {
42:         // red turn
43:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'r') {
44:             currentGameState[arenaLocation.y][arenaLocation.x] = 'w';
45:             stillPlaying = 1;
46:             return;
47:         }
48:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'R') {
49:             currentGameState[arenaLocation.y][arenaLocation.x] = 'W';
50:             stillPlaying = 1;
51:             return;
52:         }
53:     } else {
54:         // black turn
55:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'b') {
56:             currentGameState[arenaLocation.y][arenaLocation.x] = 'w';
57:             stillPlaying = 1;
58:             return;
59:         }
60:         if (currentGameState[arenaLocation.y][arenaLocation.x] == 'B') {
61:             currentGameState[arenaLocation.y][arenaLocation.x] = 'W';
62:             stillPlaying = 1;
63:             return;
64:     }
```

```
65:         }
66:     }
67:
68: // Getter
69: sf::Vector2i Checkers::getSelectedPawn(void) {
70:     auto row_iter = std::find_if(currentGameState.begin(), currentGameState.end(),
71:                                   [] (const vector<char>& row) {
72:                                       return std::find_if(row.begin(), row.end(), [] (char c) {
73:                                           return (c == 'w' || c == 'W');
74:                                       }) != row.end();
75:                                   });
76:     if (row_iter != currentGameState.end()) {
77:         auto col_iter = std::find_if(row_iter->begin(), row_iter->end(),
78:                                       [] (char c) {
79:                                           return (c == 'w' || c == 'W');
80:                                       });
81:         return sf::Vector2i(std::distance(currentGameState.begin(), row_iter),
82:                             std::distance(row_iter->begin(), col_iter));
83:     } else {
84:         return sf::Vector2i();
85:     }
86:
87: // Interactor
88: void Checkers::movePiece(sf::Vector2i mouseLocation) {
89:     // calculate the coordinate pair in relation to the 2D char array
90:     sf::Vector2i arenaLocation((mouseLocation.y - BOARD_OFFSET) / TILE_SIZE,
91:                               (mouseLocation.x - BOARD_OFFSET) / TILE_SIZE);
92:     sf::Vector2i P = getSelectedPawn();
93:     if (P.x == 0 && P.y == 0) return;
94:     bool fourD = 0, moved = 0;
95:     if (currentGameState[P.x][P.y] == 'W') fourD = true;
96:     if (playerTurn) {
97:         // red pawn
98:         if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y-1) {
99:             if (currentGameState[P.x-1][P.y-1] == 'p') {
100:                 if (fourD) currentGameState[P.x-1][P.y-1] = 'R';
101:                 else
102:                     currentGameState[P.x-1][P.y-1] = 'r';
103:                 moved = 1;
104:             }
105:         }
106:         if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y+1) {
107:             if (currentGameState[P.x-1][P.y+1] == 'p') {
108:                 if (fourD) currentGameState[P.x-1][P.y+1] = 'R';
109:                 else
110:                     currentGameState[P.x-1][P.y+1] = 'r';
111:                 moved = 1;
112:             }
113:         }
114:         if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y-2) {
115:             if ((currentGameState[P.x-1][P.y-1] == 'b' ||
116:                  currentGameState[P.x-1][P.y-1] == 'B') &&
117:                  currentGameState[P.x-2][P.y-2] == 'p') {
118:                     if (fourD) currentGameState[P.x-2][P.y-2] = 'R';
119:                     else
120:                         currentGameState[P.x-2][P.y-2] = 'r';
121:                     currentGameState[P.x-1][P.y-1] = 'p';
122:                     moved = 1;
123:                 }
124:             }
125:             if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y+2) {
```

```
126:         if ((currentGameState[P.x-1][P.y+1] == 'b' ||
127:             currentGameState[P.x-1][P.y+1] == 'B') &&
128:             currentGameState[P.x-2][P.y+2] == 'p') {
129:             if (fourD) currentGameState[P.x-2][P.y+2] = 'R';
130:             else
131:                 currentGameState[P.x-2][P.y+2] = 'r';
132:             currentGameState[P.x-1][P.y+1] = 'p';
133:             moved = 1;
134:         }
135:     }
136:     if (fourD) {
137:         if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y-1) {
138:             if (currentGameState[P.x+1][P.y-1] == 'p') {
139:                 currentGameState[P.x+1][P.y-1] = 'R';
140:                 moved = 1;
141:             }
142:         }
143:         if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y+1) {
144:             if (currentGameState[P.x+1][P.y+1] == 'p') {
145:                 currentGameState[P.x+1][P.y+1] = 'R';
146:                 moved = 1;
147:             }
148:         }
149:         if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y-2) {
150:             if ((currentGameState[P.x+1][P.y-1] == 'b' ||
151:                 currentGameState[P.x+1][P.y-1] == 'B') &&
152:                 currentGameState[P.x+2][P.y-2] == 'p') {
153:                 currentGameState[P.x+2][P.y-2] = 'R';
154:                 currentGameState[P.x+1][P.y-1] = 'p';
155:                 moved = 1;
156:             }
157:         }
158:         if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y+2) {
159:             if ((currentGameState[P.x+1][P.y+1] == 'b' ||
160:                 currentGameState[P.x+1][P.y+1] == 'B') &&
161:                 currentGameState[P.x+2][P.y+2] == 'p') {
162:                 currentGameState[P.x+2][P.y+2] = 'R';
163:                 currentGameState[P.x+1][P.y+1] = 'p';
164:                 moved = 1;
165:             }
166:         }
167:     }
168: } else {
169:     // black pawn
170:     if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y-1) {
171:         if (currentGameState[P.x+1][P.y-1] == 'p') {
172:             if (fourD) currentGameState[P.x+1][P.y-1] = 'B';
173:             else
174:                 currentGameState[P.x+1][P.y-1] = 'b';
175:             moved = 1;
176:         }
177:     }
178:     if (arenaLocation.x == P.x+1 && arenaLocation.y == P.y+1) {
179:         if (currentGameState[P.x+1][P.y+1] == 'p') {
180:             if (fourD) currentGameState[P.x+1][P.y+1] = 'B';
181:             else
182:                 currentGameState[P.x+1][P.y+1] = 'b';
183:             moved = 1;
184:         }
185:     }
186:     if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y-2) {
187:         if ((currentGameState[P.x+1][P.y-1] == 'r' ||
188:             currentGameState[P.x+1][P.y-1] == 'R') &&
189:             currentGameState[P.x+2][P.y-2] == 'p') {
190:             if (fourD) currentGameState[P.x+2][P.y-2] = 'B';
```

```
191:             else
192:                 currentGameState[P.x+2][P.y-2] = 'b';
193:                 currentGameState[P.x+1][P.y-1] = 'p';
194:                 moved = 1;
195:             }
196:         }
197:         if (arenaLocation.x == P.x+2 && arenaLocation.y == P.y+2) {
198:             if ((currentGameState[P.x+1][P.y+1] == 'r' ||
199:                  currentGameState[P.x+1][P.y+1] == 'R') &&
200:                  currentGameState[P.x+2][P.y+2] == 'p') {
201:                 if (fourD) currentGameState[P.x+2][P.y+2] = 'B';
202:                 else
203:                     currentGameState[P.x+2][P.y+2] = 'b';
204:                 currentGameState[P.x+1][P.y+1] = 'p';
205:                 moved = 1;
206:             }
207:         }
208:         if (fourD) {
209:             if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y-1) {
210:                 if (currentGameState[P.x-1][P.y-1] == 'p') {
211:                     currentGameState[P.x-1][P.y-1] = 'B';
212:                     moved = 1;
213:                 }
214:             }
215:             if (arenaLocation.x == P.x-1 && arenaLocation.y == P.y+1) {
216:                 if (currentGameState[P.x-1][P.y+1] == 'p') {
217:                     currentGameState[P.x-1][P.y+1] = 'B';
218:                     moved = 1;
219:                 }
220:             }
221:             if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y-2) {
222:                 if ((currentGameState[P.x-1][P.y-1] == 'r' ||
223:                      currentGameState[P.x-1][P.y-1] == 'R') &&
224:                      currentGameState[P.x-2][P.y-2] == 'p') {
225:                         currentGameState[P.x-2][P.y-2] = 'B';
226:                         currentGameState[P.x-1][P.y-1] = 'p';
227:                         moved = 1;
228:                     }
229:             }
230:             if (arenaLocation.x == P.x-2 && arenaLocation.y == P.y+2) {
231:                 if ((currentGameState[P.x-1][P.y+1] == 'r' ||
232:                      currentGameState[P.x-1][P.y+1] == 'R') &&
233:                      currentGameState[P.x-2][P.y+2] == 'p') {
234:                         currentGameState[P.x-2][P.y+2] = 'B';
235:                         currentGameState[P.x-1][P.y+1] = 'p';
236:                         moved = 1;
237:                     }
238:             }
239:         }
240:     }
241:     if (moved) {
242:         currentGameState[P.x][P.y] = 'p';
243:         finishLine();
244:         deselectPiece();
245:         switchTurn();
246:         stillPlaying = 0;
247:     }
248: }
249:
250: // Interactor
251: void Checkers::deselectPiece(void) {
252:     sf::Vector2i P = getSelectedPawn();
253:     if (P.x == 0 && P.y == 0) return;
254:     if (playerTurn) {
255:         if (currentGameState[P.x][P.y] == 'W')
```

```
256:         currentGameState[P.x][P.y] = 'R';
257:     else
258:         currentGameState[P.x][P.y] = 'r';
259:     stillPlaying = 0;
260:     return;
261: } else {
262:     if (currentGameState[P.x][P.y] == 'W')
263:         currentGameState[P.x][P.y] = 'B';
264:     else
265:         currentGameState[P.x][P.y] = 'b';
266:     stillPlaying = 0;
267:     return;
268: }
269: }
270:
271: // Interactor
272: void Checkers::restart(bool& winCondition) {
273:     initializeBase();
274:     playerTurn = 0;
275:     winCondition = 1;
276:     setWinTrue = 0;
277:     stillPlaying = 0;
278: }
279:
280: // Performer
281: void Checkers::playSound(void) {
282:     sf::SoundBuffer buffer;
283:     if (!buffer.loadFromFile("checkers/victory.wav")) exit(1);
284:     sf::Sound sound(buffer);
285:     sound.play();
286:     while (sound.getStatus() == sf::Sound::Playing) {
287:         // Wait for the sound to finish playing
288:     }
289: }
290:
291: // Performer
292: void Checkers::visualMoveAssist(sf::RenderTarget &target) {
293:     sf::Texture star;
294:     if (!star.loadFromFile("checkers/star_icon.png")) exit(1);
295:     sf::Vector2i P = getSelectedPawn();
296:     if (P.x == 0 && P.y == 0) return;
297:     bool fourD = 0, moveable = 0;
298:     if (currentGameState[P.x][P.y] == 'W') fourD = true;
299:     if (playerTurn) {
300:         // red pawn
301:         if (P.x-1 >= 0 && P.y-1 >= 0) {
302:             if (currentGameState[P.x-1][P.y-1] == 'p') {
303:                 drawStar(target, star, P.x-1, P.y-1);
304:                 moveable = 1;
305:             }
306:         }
307:         if (P.x-2 >= 0 && P.y-2 >= 0) {
308:             if ((currentGameState[P.x-1][P.y-1] == 'b' ||
309:                  currentGameState[P.x-1][P.y-1] == 'B') &&
310:                  currentGameState[P.x-2][P.y-2] == 'p') {
311:                 drawStar(target, star, P.x-2, P.y-2);
312:                 moveable = 1;
313:             }
314:         }
315:         if (P.x-1 >= 0 && P.y+1 < 8) {
316:             if (currentGameState[P.x-1][P.y+1] == 'p') {
317:                 drawStar(target, star, P.x-1, P.y+1);
318:                 moveable = 1;
319:             }
320:         }
321:     }
322: }
```

```

321:     if (P.x-2 >= 0 && P.y+2 < 8) {
322:         if ((currentGameState[P.x-1][P.y+1] == 'b' ||
323:             currentGameState[P.x-1][P.y+1] == 'B') &&
324:             currentGameState[P.x-2][P.y+2] == 'p') {
325:             drawStar(target, star, P.x-2, P.y+2);
326:             moveable = 1;
327:         }
328:     }
329:     if (fourD) {
330:         if (P.x+1 < 8 && P.y-1 >=0) {
331:             if (currentGameState[P.x+1][P.y-1] == 'p') {
332:                 drawStar(target, star, P.x+1, P.y-1);
333:                 moveable = 1;
334:             }
335:         }
336:         if (P.x+2 < 8 && P.y-2 >= 0) {
337:             if ((currentGameState[P.x+1][P.y-1] == 'b' ||
338:                 currentGameState[P.x+1][P.y-1] == 'B') &&
339:                 currentGameState[P.x+2][P.y-2] == 'p') {
340:                 drawStar(target, star, P.x+2, P.y-2);
341:                 moveable = 1;
342:             }
343:         }
344:         if (P.x+1 < 8 && P.y+1 < 8) {
345:             if (currentGameState[P.x+1][P.y+1] == 'p') {
346:                 drawStar(target, star, P.x+1, P.y+1);
347:                 moveable = 1;
348:             }
349:         }
350:         if (P.x+2 < 8 && P.y+2 < 8) {
351:             if ((currentGameState[P.x+1][P.y+1] == 'b' ||
352:                 currentGameState[P.x+1][P.y+1] == 'B') &&
353:                 currentGameState[P.x+2][P.y+2] == 'p') {
354:                 drawStar(target, star, P.x+2, P.y+2);
355:                 moveable = 1;
356:             }
357:         }
358:     }
359: } else {
360:     // black pawn
361:     if (P.x+1 < 8 && P.y-1 >=0) {
362:         if (currentGameState[P.x+1][P.y-1] == 'p') {
363:             drawStar(target, star, P.x+1, P.y-1);
364:             moveable = 1;
365:         }
366:     }
367:     if (P.x+2 < 8 && P.y-2 >= 0) {
368:         if ((currentGameState[P.x+1][P.y-1] == 'r' ||
369:             currentGameState[P.x+1][P.y-1] == 'R') &&
370:             currentGameState[P.x+2][P.y-2] == 'p') {
371:                 drawStar(target, star, P.x+2, P.y-2);
372:                 moveable = 1;
373:             }
374:     }
375:     if (P.x+1 < 8 && P.y+1 < 8) {
376:         if (currentGameState[P.x+1][P.y+1] == 'p') {
377:             drawStar(target, star, P.x+1, P.y+1);
378:             moveable = 1;
379:         }
380:     }
381:     if (P.x+2 < 8 && P.y+2 < 8) {
382:         if ((currentGameState[P.x+1][P.y+1] == 'r' ||
383:             currentGameState[P.x+1][P.y+1] == 'R') &&
384:             currentGameState[P.x+2][P.y+2] == 'p') {
385:                 drawStar(target, star, P.x+2, P.y+2);

```

```
386:             moveable = 1;
387:         }
388:     }
389:     if (fourD) {
390:         if (P.x-1 >= 0 && P.y-1 >= 0) {
391:             if (currentGameState[P.x-1][P.y-1] == 'p') {
392:                 drawStar(target, star, P.x-1, P.y-1);
393:                 moveable = 1;
394:             }
395:         }
396:         if (P.x-2 >= 0 && P.y-2 >= 0) {
397:             if ((currentGameState[P.x-1][P.y-1] == 'r' ||
398:                  currentGameState[P.x-1][P.y-1] == 'R') &&
399:                  currentGameState[P.x-2][P.y-2] == 'p') {
400:                 drawStar(target, star, P.x-2, P.y-2);
401:                 moveable = 1;
402:             }
403:         }
404:         if (P.x-1 >= 0 && P.y+1 < 8) {
405:             if (currentGameState[P.x-1][P.y+1] == 'p') {
406:                 drawStar(target, star, P.x-1, P.y+1);
407:                 moveable = 1;
408:             }
409:         }
410:         if (P.x-2 >= 0 && P.y+2 < 8) {
411:             if ((currentGameState[P.x-1][P.y+1] == 'r' ||
412:                  currentGameState[P.x-1][P.y+1] == 'R') &&
413:                  currentGameState[P.x-2][P.y+2] == 'p') {
414:                 drawStar(target, star, P.x-2, P.y+2);
415:                 moveable = 1;
416:             }
417:         }
418:     }
419: }
420: if (!moveable) {
421:     bool r = 0, b = 0;
422:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
423:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
424:             if (currentGameState[i][j] == 'r' ||
425:                 currentGameState[i][j] == 'R')
426:                 r = 1;
427:             if (currentGameState[i][j] == 'b' ||
428:                 currentGameState[i][j] == 'B')
429:                 b = 1;
430:         }
431:     }
432:     if (!(r && b)) setWinTrue = 1;
433: }
434: }
435:
436: // Performer
437: void Checkers::drawStar(sf::RenderTarget& target, sf::Texture star, int y
c, int xc) {
438:     sf::Sprite icon;
439:     icon.setTexture(star);
440:     // -8 is an arbitrary offset to center the star on the tile
441:     icon.setOrigin(-8, -8);
442:     icon.setPosition(xc*TILE_SIZE+BOARD_OFFSET, yc*TILE_SIZE+BOARD_OFFSET
);
443:     target.draw(icon);
444: }
445:
446: // Draw game in SFML
447: void Checkers::draw(sf::RenderTarget& target, sf::RenderStates states) co
nst {
```

```
448:     // Load piece textures
449:     sf::Texture blackPiece, blackKing, redPiece, redKing, whitePiece, whi
teKing, woodBacking;
450:     if (!blackPiece.loadFromFile("checkers/blackpawn.png")) exit(1);
451:     if (!redPiece.loadFromFile("checkers/redpawn.png")) exit(1);
452:     if (!whitePiece.loadFromFile("checkers/whitepawn.png")) exit(1);
453:     if (!blackKing.loadFromFile("checkers/blackking.png")) exit(1);
454:     if (!redKing.loadFromFile("checkers/redking.png")) exit(1);
455:     if (!whiteKing.loadFromFile("checkers/whiteking.png")) exit(1);
456:     if (!woodBacking.loadFromFile("checkers/wood_texture.png")) exit(1);
457:     sf::Sprite WB;
458:     WB.setTexture(woodBacking);
459:     // scale up wood textures
460:     WB.setScale(1, 1.2);
461:     // draw backing
462:     target.draw(WB);
463:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
464:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
465:             // Draw states stored in currentState
466:             switch (currentState[i][j]) {
467:                 case '.':
468:                 {
469:                     sf::RectangleShape R;
470:                     R.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
471:                     R.setFillColor(sf::Color::Red);
472:                     R.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE_
SIZE + BOARD_OFFSET);
473:                     target.draw(R);
474:                     break;
475:                 }
476:                 case 'p':
477:                 {
478:                     drawBackingRectangle(target, j, i);
479:                     break;
480:                 }
481:                 case 'r':
482:                 {
483:                     drawBackingRectangle(target, j, i);
484:                     sf::Sprite RP;
485:                     RP.setTexture(redPiece);
486:                     RP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE_
SIZE + BOARD_OFFSET);
487:                     target.draw(RP);
488:                     break;
489:                 }
490:                 case 'b':
491:                 {
492:                     drawBackingRectangle(target, j, i);
493:                     sf::Sprite BP;
494:                     BP.setTexture(blackPiece);
495:                     BP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE_
SIZE + BOARD_OFFSET);
496:                     target.draw(BP);
497:                     break;
498:                 }
499:                 case 'w':
500:                 {
501:                     drawBackingRectangle(target, j, i);
502:                     sf::Sprite WP;
503:                     WP.setTexture(whitePiece);
504:                     // scale down white pieces to 64 x 64 pixels
505:                     WP.setScale(0.6, 0.6);
506:                     WP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE_
SIZE + BOARD_OFFSET);
507:                     target.draw(WP);
```

```
508:                     break;
509:                 }
510:                 case 'R':
511:                 {
512:                     drawBackingRectangle(target, j, i);
513:                     sf::Sprite RP;
514:                     RP.setTexture(redKing);
515:                     RP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
SIZE + BOARD_OFFSET);
516:                     target.draw(RP);
517:                     break;
518:                 }
519:                 case 'B':
520:                 {
521:                     drawBackingRectangle(target, j, i);
522:                     sf::Sprite BP;
523:                     BP.setTexture(blackKing);
524:                     BP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
SIZE + BOARD_OFFSET);
525:                     target.draw(BP);
526:                     break;
527:                 }
528:                 case 'W':
529:                 {
530:                     drawBackingRectangle(target, j, i);
531:                     sf::Sprite WP;
532:                     WP.setTexture(whiteKing);
533:                     // scale down white pieces to 64 x 64 pixels
534:                     WP.setScale(0.6, 0.6);
535:                     WP.setPosition(j * TILE_SIZE + BOARD_OFFSET, i * TILE
SIZE + BOARD_OFFSET);
536:                     target.draw(WP);
537:                     break;
538:                 }
539:             }
540:         }
541:     }
542:     sf::Font font;
543:     font.loadFromFile("checkers/arial.ttf");
544:     sf::Text turnText("", font, 40);
545:     turnText.setFillColor(sf::Color::White);
546:     turnText.setPosition(Vector2f(10, 0));
547:     if (playerTurn) {
548:         // Red turn
549:         turnText.setString("R");
550:     } else {
551:         // Black turn
552:         turnText.setString("B");
553:     }
554:     target.draw(turnText);
555: }
556:
557: // Initialize game storage vectors
558: void Checkers::initializeBase(void) {
559:     currentState.resize(BOARD_DIMENSIONS);
560:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
561:         currentState[i].resize(BOARD_DIMENSIONS);
562:         for (int j = 0; j < BOARD_DIMENSIONS; j++) {
563:             // Draw red background tiles
564:             if (((j + (i%2))%2)) {
565:                 if (i <= 2) {
566:                     // Black piece represented as 'b' (black)
567:                     // placed on lines 0 - 2
568:                     currentState[i][j] = 'b';
569:                 } else if (i >= 5) {
```

```
570:                     // Red piece represented as 'r' (red)
571:                     // placed on lines 5 - 7
572:                     currentGameState[i][j] = 'r';
573:                 } else {
574:                     // Black background tile represented 'p' (playable)
575:                     currentGameState[i][j] = 'p';
576:                 }
577:             } else {
578:                 // Red background tile represented as '.' (invalid)
579:                 currentGameState[i][j] = '.';
580:             }
581:         }
582:     }
583: }
584:
585: // Automatically king pawns at respective finish lines
586: void Checkers::finishLine(void) {
587:     for (int i = 0; i < BOARD_DIMENSIONS; i++) {
588:         if (!(i%2)) {
589:             // black pieces - 7 is last row
590:             if (currentGameState[7][i] == 'b') {
591:                 currentGameState[7][i] = 'B';
592:                 return;
593:             }
594:         } else {
595:             // red pieces - 0 is first row
596:             if (currentGameState[0][i] == 'r') {
597:                 currentGameState[0][i] = 'R';
598:                 return;
599:             }
600:         }
601:     }
602: }
603:
604: // Helper function
605: bool mouseInGameBounds(sf::Vector2i mouseLocation) {
606:     if ((mouseLocation.x >= BOARD_OFFSET && mouseLocation.x <=
607:          TILE_SIZE * BOARD_DIMENSIONS + BOARD_OFFSET) &&
608:          (mouseLocation.y >= BOARD_OFFSET && mouseLocation.y <=
609:          TILE_SIZE * BOARD_DIMENSIONS + BOARD_OFFSET)) return 1;
610:     return 0;
611: }
612:
613: // Performing helper function
614: void drawBackingRectangle(sf::RenderTarget& target, int x, int y) {
615:     sf::RectangleShape B;
616:     B.setSize(Vector2f(TILE_SIZE, TILE_SIZE));
617:     B.setFillColor(sf::Color::Black);
618:     B.setPosition(x * TILE_SIZE + BOARD_OFFSET, y * TILE_SIZE + BOARD_OFFSET);
619:     target.draw(B);
620: }
```

# PS5 DNA Alignment

## 1 Description of the assignment

In this assignment I created an EDistance class that computed the edit distance of two strings. Here the edit distance is the optimal alignment of the two strings. This optimal alignment is computed using a scoring system, where a gap in either string is a penalty of two, no gap but a misaligned letter is a penalty of one, and no gap and a matching letter is no penalty. The optimal edit distance has the lowest score, and is computed by dynamically allocating a matrix of integer values of size NxM, where N is the size of string one, plus one, and M is the size of string two, plus one. At the bottom right corner of the matrix is a score of zero. Here the strings “meet” when they are aligned. Going directly up or directly left from the bottom corner, the consecutive values increase by two, as this represents the penalty for a gap in either string. Beginning at the bottom right corner, the scoring algorithm is performed of all values remaining in the NxM matrix. This will be explained in the following section. After the matrix is filled in with scores, the optimal edit distance is found at (0, 0) of the matrix.

In order to find the optimal alignment we must traverse this matrix with reference to the strings, using dashes as gaps in the final alignment. This algorithm will also be discussed in the following section. This can be used for any two strings of values, but its most useful application is the alignment of DNA, as shown in the test cases. My program computed the alignment for two strings of lengths 2,500, 5,000, 7,000, 10,000, 20,000, and 28,284, as well as multiple strings of lengths less than 100 characters. These strings were taken from stdin using the “`>`” redirect, and either output to the terminal in small cases, or output to a .txt document using the “`>>`” redirect in large cases.

## 2 Key algorithms, Data structures, or OO designs

The two key algorithms used in this assignment were the scoring algorithm and the traversal algorithm. The scoring algorithm can be seen at line 27 of EDistance.cpp. The optDistance() method fills in the matrix using the following expression:

*For each :  $0 \leq i < M, 0 \leq j < N$*

$$Matrix[i][j] = \min(matrix[i+1][j+1]+penalty(), matrix[i+1][j]+2, matrix[i][j+1]+2)$$

This algorithm is executed from the bottom right of the matrix and compares values adjacent to and diagonal from the selected element. This is performed right to left, bottom to top. In the above expression, the penalty() method is the zero or one penalty from the character discrepancy check performed on the selected element. If the characters are different, a penalty of one is applied, else, zero.

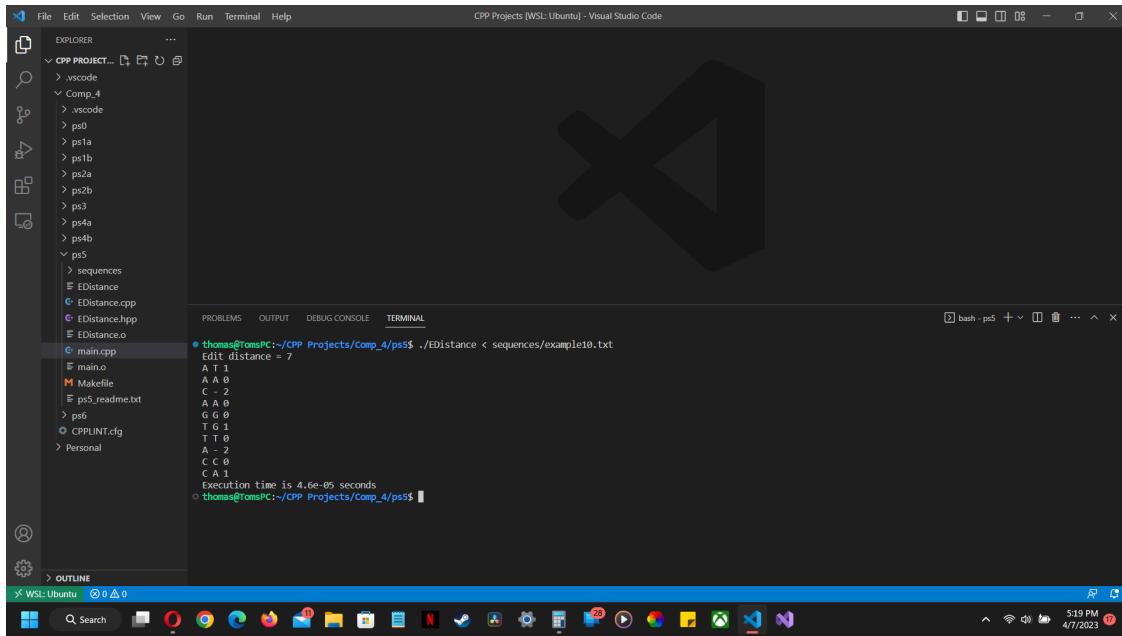
My implementation of the traversal algorithm can be seen in the alignment() method defined at line 40 of EDistance.cpp. This method starts at the top left of the scoring matrix and finds the optimal path to the bottom right. Unless at a boundary condition, the algorithm prioritizes diagonal checks, then subsequently checks for gaps in the alignment. In each case, there are a series of “if” statements that check to see if the scores at each position are equivalent to the currently selected element’s score, minus the penalty. This is either the static two or the zero/one based on the character discrepancy of the current element. The algorithm traces the matrix and only retrieves the optimal traversal, pushing the coordinates of each valid element back onto the optPath list in order. After the traversal is complete, beginning at line 70 of EDistance.cpp the list is parsed; and based on the diagonal or perpendicular movement the alignment is prepared using the reference strings’ characters and dashes to represent the gaps in alignment. This is assembled into a single contiguous string and returned by the method.

### **3 What I learned from the project**

In this assignment I learned the power of dynamic programming. Using a series of simple algorithms in combination with a fully adjustable dataset, my program could easily handle and calculate hundreds of millions of scores based off of multi-thousand character strings. This would have been effectively impossible to do by hand and without error, but my program was speedy and accurate every time. Moreover I learned about different approaches to the same problem: In this case DNA alignment. The spec outlined multiple tactics for approaching the problem, including recursive approaches, the Needleman-Wunsch method, and Hirschberg's algorithm, all with certain advantages and disadvantages. Based on my personal constraints and implementation I could adapt my method to prioritize storage space, cpu power, or processing time.

An optional component of this assignment was to work with a partner; I chose to work alone on this assignment. This was because I wanted to challenge myself to conceptualize and implement my own interpretation of the problem. This may have taken more time overall, but I'm glad I tackled the problem independently because I now have a greater capacity to debug my code. For example: during this assignment I completed 90% of the algorithm, but my results for the optimal alignment were off by two for all of the test cases. This problem stumped me for a while, and could have easily been remediated by a partner. By sitting with the problem and debugging my own code I found the issue not in my algorithm, but in my reading of the test case variables. If I had not worked alone on this assignment I wouldn't have had that same struggle and intellectual payoff.

## 4 Screenshot of program output



A screenshot of the Visual Studio Code interface running in WSL:Ubuntu. The Explorer sidebar shows a project named 'CPP PROJECT...' containing files like '.vscode', 'Comp\_4', 'ps0', 'ps1a', 'ps1b', 'ps2a', 'ps2b', 'ps3', 'ps4a', 'ps4b', 'ps5', 'sequences', 'EDistance.cpp', 'EDistance.hpp', 'EDistance.o', 'main.cpp', 'main.o', 'Makefile', 'ps5\_readme.txt', 'ps6', 'CPPLINT.cfg', and 'Personal'. The Terminal tab is active, displaying the command 'thomas@TomsPC:~/CPP Projects/Comp\_4/ps5\$ ./EDistance < sequences/example10.txt' followed by the output:

```
Edit distance = 7
A T 1
A C 0
C C 2
A A 0
G G 0
T G 1
T T 0
C C 2
C C 0
C A 1
Execution time is 4.6e-05 seconds
thomas@TomsPC:~/CPP Projects/Comp_4/ps5$
```

Figure 8: ps5 terminal output

## 5 Issues and bugs

No issues or bugs present.

The program was fully functional within the given specification.

## 6 Source code for the project

**Makefile**        **Wed Apr 05 15:01:21 2023**        **1**

```
1: CC = g++
2: FLAGS = -g -Wall -Werror -pedantic -std=c++17
3: LIB = -lsfml-system
4: DEPS = EDistance.o
5:
6: .PHONY: all clean lint
7:
8: all: lint EDistance
9:
10: EDistance: main.o EDistance.o $(DEPS)
11:         $(CC) $(FLAGS) -o EDistance $^ $(LIB)
12:
13: %.o: %.cpp
14:         $(CC) $(FLAGS) -c $<
15:
16: lint:
17:         cpplint --recursive --quiet .
18:
19: clean:
20:         rm *.o EDistance
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "EDistance.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     string stringA, stringB;
6:     std::cin >> stringA;
7:     std::cin >> stringB;
8:     // start a clock before edit distance is computed
9:     sf::Clock clock;
10:    sf::Time t;
11:    // create the object and compute the distance
12:    EDistance a(stringA, stringB);
13:    cout << "Edit distance = " << a.optDistance() << endl;
14:    cout << a.alignment();
15:    // a.printMatrix();
16:    // print elapsed time
17:    t = clock.getElapsedTime();
18:    cout << "Execution time is " << t.asSeconds() << " seconds \n";
19:
20:    return 0;
21: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef EDISTANCE_HPP
3: #define EDISTANCE_HPP
4:
5: #include <algorithm>
6: #include <cctype>
7: #include <fstream>
8: #include <iomanip>
9: #include <iostream>
10: #include <string>
11: #include <list>
12: #include <utility>
13: #include <SFML/System.hpp>
14:
15: using std::cout;
16: using std::endl;
17: using std::pair;
18: using std::setw;
19: using std::string;
20:
21: class EDistance {
22: public:
23:     // Constructor + Destructor
24:     EDistance(const string& lOp, const string& rOp);
25:     ~EDistance () { delete [] matrix; }
26:
27:     // interactor functions
28:     int penalty(char a, char b) const { return a != b; }
29:     int min(int a, int b, int c) const;
30:     int optDistance();
31:     string alignment();
32:
33:     // debug functions
34:     void printMatrix();
35:
36: private:
37:     int* matrix;
38:     std::list<pair<int, int>> optPath;
39:     // dimensions of the matrix, string.size()+1; N = x; M = y;
40:     int _M, _N;
41:     string _MString, _NString;
42: };
43:
44: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "EDistance.hpp"
3:
4: // Constructor
5: EDistance::EDistance(const string& lOp, const string& rOp) :
6:     _M(lOp.size()+1), _N(rOp.size()+1), _MString(lOp), _N
String(rOp) {
7:     matrix = new int[_M * _N];
8:     _NString.push_back('-'); _MString.push_back('-');
9:     // set bounds of matrix
10:    for (int i = 1; i < _M; i++) {
11:        matrix[i*_N-1] = 2*(i-1);
12:    }
13:    for (int j = 0; j < _N; j++) {
14:        matrix[_M*_N-_N+j] = 2*(j-1);
15:    }
16: }
17:
18: // interactor function
19: int EDistance::min(int a, int b, int c) const {
20:     int minVal = a;
21:     if (b < minVal) minVal = b;
22:     if (c < minVal) minVal = c;
23:     return minVal;
24: }
25:
26: // interactor function
27: int EDistance::optDistance() {
28:     // begin on bounds size - 2; 1 for standard bounds and 1 for addition
al dash character
29:     for (int i = _M - 2; i >= 0; i--) {
30:         for (int j = _N - 2; j >= 0; j--) {
31:             // fill the matrix using the min method
32:             matrix[i*_N+j] = min(matrix[(i+1)*_N+j+1]+penalty(_MString.at
(i), _NString.at(j)),
33:                                 matrix[(i+1)*_N+j]+2, matrix[(i*_N)+j+1]
+2);
34:         }
35:     }
36:     return matrix[0];
37: }
38:
39: // interactor function
40: string EDistance::alignment() {
41:     // traverse the matrix, collect points, add them to the list
42:     int i = 0, j = 0;
43:     while (i < _M-1 || j < _N-1) {
44:         // if at boundary of matrix:
45:         if (j == _N-1) {
46:             optPath.push_back(pair<int, int>(i+1, j));
47:             i++;
48:         } else if (i == _M-1) {
49:             optPath.push_back(pair<int, int>(i, j+1));
50:             j++;
51:         } // else perform normal checks:
52:         // diagonal
53:         } else if (matrix[i*_N+j] == matrix[(i+1)*_N+j+1] +
54:                    penalty(_MString.at(i), _NString.at(j))) {
55:             optPath.push_back(pair<int, int>(i+1, j+1));
56:             i++; j++;
57:         } // down
58:         } else if (matrix[i*_N+j] == matrix[(i+1)*_N+j] + 2) {
59:             optPath.push_back(pair<int, int>(i+1, j));
60:             i++;
61:         } // right
```

```
62:         } else {
63:             optPath.push_back(pair<int, int>(i, j+1));
64:             j++;
65:         }
66:         // the above order is important because it ensures that diagonals
are
67:         // prioritized unless at a border condition
68:     }
69:     // traverse the list, refrence the matrix, assemble string
70:     string outputString;
71:     pair<int, int> previousIter(0, 0);
72:     for (pair<int, int> iter : optPath) {
73:         // diagonal
74:         if (previousIter.first == iter.first-1 && previousIter.second ==
iter.second-1) {
75:             outputString.push_back(_MString.at(previousIter.first));
76:             outputString.append(" ");
77:             outputString.push_back(_NString.at(previousIter.second));
78:             outputString.append(" ");
79:             outputString.append(std::to_string(penalty(_MString.at(previo
usIter.first),
80:                                         _NString.at(previousIter.second))));;
81:             outputString.append("\n");
82:             // down
83:             } else if (previousIter.first == iter.first-1 && previousIter.sec
ond == iter.second) {
84:                 outputString.push_back(_MString.at(previousIter.first));
85:                 outputString.append(" - 2\n");
86:                 // right
87:                 } else {
88:                     outputString.append("- ");
89:                     outputString.push_back(_NString.at(previousIter.second));
90:                     outputString.append(" 2\n");
91:                 }
92:                 previousIter = iter;
93:             }
94:             return outputString;
95:     }
96:
97: // debug function
98: void EDistance::printMatrix() {
99:     cout << "N/M ";
100:    for (char a : _NString) cout << setw(4) << a;
101:    cout << endl;
102:    for (int i = 0; i < _M; i++) {
103:        cout << setw(3) << _MString.at(i) << " ";
104:        for (int j = 0; j < _N; j++) {
105:            cout << setw(4) << matrix[i*_N+j];
106:        }
107:        cout << endl;
108:    }
109: }
```

# PS6 RandWriter

## 1 Description of the assignment

In this assignment I created a RandWriter class that writes random text based on a reference text. The main program takes in two integers: the first “K” which is the size of the subdivisions of the text, and “L” the length of the random text output. The object then read a reference text from std::cin, then developed “kgrams”, (N) strings of size K where N is the size of the reference text. Along with these kgrams, k+1grams were developed, which is every occurrence of the kgram followed by the subsequent character in the reference text. Using the kgrams and k+1grams, a probabilistic dictionary was developed for each potential string of size K (given that it appeared at least once in the text). To develop the random text output, the first K characters were used as the seeding value for the generate() method defined at line 78 of RandWriter.cpp. Each subsequent character was chosen from the probability dictionary of the current kgram which matches the relative distribution of characters that follow that kgram in the reference text. These characters were pushed back onto the output string until the string was of size L, then returned by the method. Text generated with a low K value (0-4) frequently produces unintelligible words, makes spelling mistakes, and includes inappropriate characters. Text generated with moderate K values (5-9) frequently produces intelligible words and phrases in the style of the reference text, although there are some evident discrepancies. Text generated with high K values (10 or greater) are very accurate with the reproduction of understandable text at the risk of copying the reference material word for word. This is because some kgrams of large size are unique in a text, which lead to the reproduction of multiple consecutive sentences or a whole paragraph ripped directly from the reference material.

## 2 Key algorithms, Data structures, or OO designs

The key data structure used in this assignment was the Markov Model, which I chose to implement as an unordered map. This map contained the string kgrams as keys, and a pair of data: an int for the occurrences of the kgram, and a string for the probability dictionary of the k+1grams. The constructor for the object parsed through the reference text, and treated it as cyclical. For each occurrence of the kgram, the int at the kgram's key in the model was incremented to show a new occurrence, and the following letter was appended to the dictionary string. After iterating through the entire reference text, the model would be full of every possible K sized string as well as the occurrences of that string and the following letters that followed it, stored in the form of one contiguous string. The interior state of this Markov Model could be output using the overloaded stream insertion operator.

An algorithm that was used in the assignment was the kRand() method at line 57 of RandWriter.cpp. The kRand() method selected a random value from the probability dictionary at a given kgram by using a random\_device, a Mersenne Twister, and a uniform\_int\_distribution. The std::random\_device generated a pseudo-random integer which was used as the seed value for the Mersenne Twister. At line 68 of the implementation file a std::uniform\_int\_distribution was initialized with the bounds of the dictionary of the given kgram. A random character from the dictionary was then chosen and returned by the method using the Mersenne Twister as the seed for the uniform\_int\_distribution, where it would index a character within the bounds of the dictionary.

One helper function that was used to complete the assignment was the cycleString() function defined at line 117 of RandWriter.cpp. This function modified a string in place by cycling the values to the left and pushing back a character item to the end of the string. This was used to parse through the reference text for all kgrams, and was used for the generate() method by converting k+1grams to new kgrams, which was then used to seed the new kRand() method.

In order to deal with a potential  $K = 0$ , a global variable ORDERZERO was defined as the master reference key for the map, where the reference text itself was the probability dictionary. This meant that characters in the final text output were distributed randomly in the same proportions as they appeared in the reference text with no further context relative to kgrams.

### 3 What I learned from the project

Looking back on the project I see that my internal Markov Model was inefficient. Rather than employing an unordered map of string keys with a pair of int and string as the data values, I should have simply used an unordered map of string keys and string data. This is because the int data in the pair is redundant. The frequency of the kgram in the reference text that the int is supposed to represent is also the size of the dictionary string. This did not effect the correctness of the text generation, but did impact the speed and efficiency of the model slightly by calling the increment operator once for every character in the reference text, as seen at line 21 of the implementation file.

### 4 Screenshot of program output

The screenshot shows a Visual Studio Code interface. The Explorer sidebar on the left lists a project named 'Comp\_4' with files like 'vscode', '.vscode', 'ps0', 'ps1a', 'ps1b', 'ps2a', 'ps2b', 'ps3', 'ps4a', 'ps4b', 'ps5', 'ps6', and 'CPPLINT.cfg'. The 'OUTLINE' view is also visible. The main editor area displays the 'TextWriter.cpp' file:

```
File Edit Selection View Go Run Terminal Help
EXPLORER CPP PROJECTS [WSL: UBU... TextWriter.cpp - CPP Projects [WSL: Ubuntu] - Visual Studio Code
Comp_4 > ps6 > TextWriter.cpp > main(int argc, char *argv[])
12 refrenceText.append(line);
13
14 // create the RandWriter object
15 RandWriter item(refrenceText, k);
16
17 // output the internal state of the Markov model
18 cout << item;
19
20 // generate an L sized output of text
21 cout << item.generate(refrenceText.substr(0, k), l);
22
23 return 0;
24
25
```

The terminal below shows the output of the program:

```
thomas@Tom'sPC:~/CPP Projects/Comp_4/ps6$ ./TextWriter 5 1000 < sample_texts/tomssayer.txt
ER XXXAS the fainted, for joy in his bar'l suit hide that."They dressed you things, Tom. "Listened what I believe you kissed me evening interested in the town with might in variety in which small sons of the what's so, I reckon its soft grew more them; and that must be any very pale and looked until the stripped, and friendles revengeful and little caugher was like to that whack. She women go to it, just fishing about his hand over ailing that they will. I'm good, if he come time-not well the sleepy atmosphered interred they had take, and straight be the "Amen" was far beyond that he envy of an hour land on all refugee. Luck wait-for the patient back that it might off you i'll be a gravest of them, and stood repair; he did ready life of this system to blamed of the cling to begin town by natures together, embracing what's the was the departed him by thing in town into try, and that sort an hours more, who was about every sermon. The excession in huck?"Well," but for the graceful ca
```

Figure 9: ps6 terminal output

### 5 Issues and bugs

No issues or bugs present.

The program was fully functional within the given specification.

### 6 Source code for the project

```
Makefile           Mon Apr 10 12:34:45 2023           1
1: CC = g++
2: FLAGS = -g -Wall -Werror -pedantic -std=c++17
3: LIB = -lsfml-system
4: DEPS = RandWriter.o
5:
6: .PHONY: all clean lint
7:
8: all: lint TextWriter test
9:
10: TextWriter: TextWriter.o RandWriter.o $(DEPS)
11:      $(CC) $(FLAGS) -o TextWriter $^ $(LIB)
12:
13: test: test.o RandWriter.o $(DEPS)
14:      $(CC) $(FLAGS) -o test $^ $(LIB) -lboost_unit_test_framework
15:
16: %.o: %.cpp
17:      $(CC) $(FLAGS) -c $<
18:
19: lint:
20:      cpplint --recursive --quiet .
21:
22: clean:
23:      rm *.o TextWriter test output.txt
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "RandWriter.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     // Command-line arguments
6:     int k = atoi(argv[1]);
7:     int l = atoi(argv[2]);
8:
9:     // get all lines of reference text
10:    string line, referenceText;
11:    while (std::getline(std::cin, line))
12:        referenceText.append(line);
13:
14:    // create the RandWriter object
15:    RandWriter item(referenceText, k);
16:
17:    // output the internal state of the Markov model
18:    // cout << item;
19:
20:    // generate an L sized output of text
21:    cout << item.generate(referenceText.substr(0, k), l);
22:
23:    return 0;
24: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef RANDWRITER_HPP
3: #define RANDWRITER_HPP
4:
5: #include <algorithm>
6: #include <fstream>
7: #include <iostream>
8: #include <list>
9: #include <random>
10: #include <string>
11: #include <unordered_map>
12: #include <utility>
13: #include <SFML/System.hpp>
14:
15: #define ORDERZERO "refrence_text"
16:
17: using std::cout;
18: using std::endl;
19: using std::list;
20: using std::pair;
21: using std::string;
22: using std::unordered_map;
23:
24: class RandWriter {
25: public:
26:     // Create a Markov model of order k from given text
27:     // Assume that text has length at least k.
28:     RandWriter(string text, int k);
29:
30:     // Order k of Markov model
31:     int orderK() const { return _K; }
32:
33:     // Number of occurrences of kgram in text
34:     // Throw an exception if kgram is not length k
35:     int freq(string kgram) const;
36:
37:     // Number of times that character c follows kgram
38:     // if order=0, return num of times that char c appears
39:     // (throw an exception if kgram is not of length k)
40:     int freq(string kgram, char c) const;
41:
42:     // Random character following given kgram
43:     // (throw an exception if kgram is not of length k)
44:     // (throw an exception if no such kgram)
45:     char kRand(string kgram);
46:
47:     // Generate a string of length L characters by simulating a trajectory
48:     // through the corresponding Markov chain. The first k characters of
49:     // the newly generated string should be the argument kgram.
50:     // Throw an exception if kgram is not of length k.
51:     // Assume that L is at least k
52:     string generate(string kgram, int L);
53:
54:     // Overload the stream insertion operator << and display the internal state
55:     // of the Markov model. Print out the order, alphabet, and the frequencies
56:     // of the k-grams and k+1-grams
57:     friend std::ostream& operator<<(std::ostream& out, RandWriter& obj);
58:
59: private:
60:     int _K;
61:     // key is kgram, data is frequency of that kgram
62:     // and probability alphabet/dictionary of kgram
63:     unordered_map<string, pair<int, string>> MarkovModel;
```

```
64: };
65:
66: // helper function
67: void cycleString(string& str, char item);
68:
69: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "RandWriter.hpp"
3:
4: // Constructor:
5: RandWriter::RandWriter(string text, int k) : _K(k) {
6:     // if order 0, the text IS the dictionary of probabilities
7:     if (!k) {
8:         MarkovModel[ORDERZERO].second.assign(text);
9:         MarkovModel[ORDERZERO].first = static_cast<int>(text.size());
10:    }
11:    // else
12:    // Append the first k+1 characters to the end of the text
13:    string kgram = text.substr(0, k+1);
14:    text.append(kgram);
15:    kgram.pop_back();
16:    // for each char in the text, generate a kgram
17:    // starting with the first k-characters:
18:    for (size_t i = k; i < text.size()-1; i++) {
19:        // at this kgram push back the following char
20:        // into the personal dictionary for that kgram
21:        MarkovModel[kgram].first++;
22:        MarkovModel[kgram].second.push_back(text[i]);
23:        // generate new kgram
24:        cycleString(kgram, text[i]);
25:    }
26:    // the MarkovModel is now full of kgrams
27:    // and personal probability dictionaries
28: }
29:
30: // Number of occurrences of kgram in text
31: // Throw an exception if kgram is not length k
32: int RandWriter::freq(string kgram) const {
33:     if (kgram.size() != static_cast<size_t>(_K)) {
34:         throw std::invalid_argument("String length != k in FREQ");
35:     } else {
36:         if (!_K) kgram = ORDERZERO;
37:         return MarkovModel.at(kgram).first;
38:     }
39: }
40:
41: // Number of times that character c follows kgram
42: // if order=0, return num of times that char c appears
43: // (throw an exception if kgram is not of length k)
44: int RandWriter::freq(string kgram, char c) const {
45:     if (kgram.size() != static_cast<size_t>(_K)) {
46:         throw std::invalid_argument("String length != k in FREQ");
47:     } else {
48:         if (!_K) kgram = ORDERZERO;
49:         return std::count(MarkovModel.at(kgram).second.begin(),
50:                           MarkovModel.at(kgram).second.end(), c);
51:     }
52: }
53:
54: // Random character following given kgram
55: // (throw an exception if kgram is not of length k)
56: // (throw an exception if no such kgram)
57: char RandWriter::kRand(string kgram) {
58:     if (kgram.size() != static_cast<size_t>(_K)) {
59:         throw std::invalid_argument("String length != k in KRAND");
60:     } else {
61:         // check for order zero
62:         if (!_K) kgram = ORDERZERO;
63:         // get bounds of dictionary
64:         int lengthOfDictionary = MarkovModel.at(kgram).second.size() - 1;
65:         // set randomizing variables
```

```
66:         std::random_device rd;
67:         std::mt19937 gen(rd());
68:         std::uniform_int_distribution<> distrib(0, lengthOfDictionary);
69:         return MarkovModel.at(kgram).second[distrib(gen)];
70:     }
71: }
72:
73: // Generate a string of length L characters by simulating a trajectory
74: // through the corresponding Markov chain. The first k characters of
75: // the newly generated string should be the argument kgram.
76: // Throw an exception if kgram is not of length k.
77: // Assume that L is at least k
78: string RandWriter::generate(string kgram, int L) {
79:     if (kgram.size() != static_cast<size_t>(_K)) {
80:         throw std::invalid_argument("String length != k in GENERATE " + k
gram);
81:     }
82:     string outputString;
83:     char newItem;
84:     for (int i = 0; i < L; i++) {
85:         newItem = kRand(kgram);
86:         outputString.push_back(newItem);
87:         cycleString(kgram, newItem);
88:     }
89:     outputString.push_back('\n');
90:     return outputString;
91: }
92:
93: // Overload the stream insertion operator << and display the internal sta
te
94: // of the Markov model. Print out the order, alphabet, and the frequencie
s
95: // of the k-grams and k+1-grams
96: std::ostream& operator<<(std::ostream& out, RandWriter& obj) {
97:     out << "Order: " << obj.orderK() << endl;
98:     for (auto const &outerMap : obj.MarkovModel) {
99:         cout << "Kgram: " << outerMap.first << " Frequency: " << outer
Map.second.first;
100:        cout << " Dictionary: " << outerMap.second.second << endl;
101:        list<char> usedChars;
102:        for (char kplgram : outerMap.second.second) {
103:            // if it could not find the element in the used chars list, c
reate a k+1 gram
104:            if (std::find(usedChars.begin(), usedChars.end(), kplgram) ==
usedChars.end()) {
105:                cout << "|\\tK+1gram: " << outerMap.first << kplgram << "
Frequency: " <<
106:                std::count(outerMap.second.second.begin(),
107:                           outerMap.second.second.end(), kplgram)
108:                           << endl;
109:                usedChars.push_back(kplgram);
110:            }
111:        }
112:    }
113:    return out;
114: }
115:
116: // helper function
117: void cycleString(string& str, char item) {
118:     if (!str.size()) return;
119:     str.erase(0, 1);
120:     str.push_back(item);
121: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "RandWriter.hpp"
3:
4: #define BOOST_TEST_DYN_LINK
5: #define BOOST_TEST_MODULE Main
6: #include <boost/test/unit_test.hpp>
7:
8: #define TESTING_ITEM "gagggagagggcgagaaa"
9: using std::invalid_argument;
10: using std::out_of_range;
11:
12: BOOST_AUTO_TEST_CASE(testOrderK) {
13:     RandWriter l(TESTING_ITEM, 3);
14:     BOOST_REQUIRE_EQUAL(l.orderK(), 3);
15:     RandWriter m(TESTING_ITEM, 0);
16:     BOOST_REQUIRE_EQUAL(m.orderK(), 0);
17:     RandWriter n(TESTING_ITEM, 100);
18:     BOOST_REQUIRE_EQUAL(n.orderK(), 100);
19: }
20:
21: BOOST_AUTO_TEST_CASE(testFreqKGram) {
22:     RandWriter l(TESTING_ITEM, 4);
23:     BOOST_REQUIRE_EQUAL(l.freq("gagg"), 2);
24:     BOOST_REQUIRE_NO_THROW(l.freq("gagg"));
25:     BOOST_REQUIRE_THROW(l.freq("gag"), invalid_argument);
26:     BOOST_REQUIRE_THROW(l.freq("gaggg"), invalid_argument);
27:     BOOST_REQUIRE_THROW(l.freq(" "), invalid_argument);
28:     BOOST_REQUIRE_THROW(l.freq(""), invalid_argument);
29:     RandWriter m(TESTING_ITEM, 1);
30:     BOOST_REQUIRE_EQUAL(m.freq("c"), 1);
31:     BOOST_REQUIRE_EQUAL(m.freq("g"), 9);
32:     BOOST_REQUIRE_EQUAL(m.freq("a"), 7);
33:     BOOST_REQUIRE_THROW(m.freq(" "), out_of_range);
34: }
35:
36: BOOST_AUTO_TEST_CASE(testFreqKGramC) {
37:     RandWriter l(TESTING_ITEM, 4);
38:     BOOST_REQUIRE_EQUAL(l.freq("gagg", 'c'), 1);
39:     BOOST_REQUIRE_NO_THROW(l.freq("gagg", 'c'));
40:     BOOST_REQUIRE_THROW(l.freq("gag", 'a'), invalid_argument);
41:     BOOST_REQUIRE_THROW(l.freq("gaggg", 'a'), invalid_argument);
42:     BOOST_REQUIRE_THROW(l.freq(" ", 'a'), invalid_argument);
43:     BOOST_REQUIRE_THROW(l.freq("", 'a'), invalid_argument);
44:     RandWriter m(TESTING_ITEM, 1);
45:     BOOST_REQUIRE_EQUAL(m.freq("c", 'g'), 1);
46:     BOOST_REQUIRE_EQUAL(m.freq("g", 'a'), 5);
47:     BOOST_REQUIRE_EQUAL(m.freq("a", 'g'), 5);
48:     BOOST_REQUIRE_NO_THROW(m.freq("c", 'g'));
49:     BOOST_REQUIRE_NO_THROW(m.freq("g", 'a'));
50:     BOOST_REQUIRE_NO_THROW(m.freq("a", 'g'));
51:     BOOST_REQUIRE_THROW(m.freq(" ", 'a'), out_of_range);
52:     RandWriter n(TESTING_ITEM, 0);
53:     BOOST_REQUIRE_EQUAL(n.freq("", 'c'), 1);
54:     BOOST_REQUIRE_EQUAL(n.freq("", 'g'), 9);
55:     BOOST_REQUIRE_EQUAL(n.freq("", 'a'), 7);
56: }
57:
58: BOOST_AUTO_TEST_CASE(testKRand) {
59:     RandWriter l(TESTING_ITEM, 6);
60:     BOOST_REQUIRE_EQUAL(l.kRand("gagagg"), 'c');
61:     BOOST_REQUIRE_NO_THROW(l.kRand("gagagg"));
62:     BOOST_REQUIRE_THROW(l.kRand("gag"), invalid_argument);
63:     BOOST_REQUIRE_THROW(l.kRand("gggggg"), out_of_range);
64:     RandWriter m(TESTING_ITEM, 2);
65:     BOOST_REQUIRE_EQUAL(m.kRand("cg"), 'a');
```

```
66:     BOOST_REQUIRE_NO_THROW(m.kRand("cg"));
67:     BOOST_REQUIRE_THROW(m.kRand("gag"), invalid_argument);
68:     BOOST_REQUIRE_THROW(m.kRand("cc"), out_of_range);
69: }
70:
71: BOOST_AUTO_TEST_CASE(testGenerate) {
72:     RandWriter l(TESTING_ITEM, 3);
73:     BOOST_REQUIRE_THROW(l.generate("gagagg", 4), invalid_argument);
74:     BOOST_REQUIRE_NO_THROW(l.generate("gag", 4));
75:     BOOST_REQUIRE_THROW(l.generate("ccc", 4), out_of_range);
76:     RandWriter m(TESTING_ITEM, 0);
77:     BOOST_REQUIRE_THROW(m.generate("cg", 4), invalid_argument);
78:     BOOST_REQUIRE_NO_THROW(m.generate("", 4));
79: }
```

# **PS7 Kronos Log Parsing**

## **1 Description of the assignment**

In this assignment I created a LogParser class which accepted a std::ifstream input and the file path. This class then generated a report based on the log input using the generateRPT() method. The generate method logged the beginning, end, and failure of boot sequences for Kronos InTouch devices, assuming no nesting booting.

## **2 Key algorithms, Data structures, or OO designs**

The key algorithm used in my implementation was my generateRPT() method defined at line 4 of LogParser.cpp. This function parsed through Kronos .log files and generated a string which was output to a .log.rpt file in main.cpp at line 16. The algorithm used the syntax provided in the device5\_intouch.log\_BOOTONLY.rpt, noting the line and time of boot start, as well as the line and time of boot completion, and elapsed time, or the indication of an incomplete boot. The method performed this documentation across multiple attempted boot cycles.

Another algorithm used can be found at lines 37 – 39 of the implementation file. These lines used ptime and time\_duration variables found in the boost::posix\_time library. The constructor of ptime automatically converts the date and time string to a valid posix\_time, then the time\_duration object retrieves the difference between the start and end times using the built-in subtraction operator and assignment operator. This time discrepancy is then reported in milliseconds as prescribed by the spec.

## **3 What I learned from the project**

In this assignment I learned about the effective use of regular expressions (regex). Rather than doing manual comparisons to predefined static strings, regexes allow you to catch expressions dynamically. If there is known variation in a program output, a regex may account for a variety of scenarios including unique number or name strings. In this assignment I used two regexes to catch the beginning and conclusion of the boot sequence.

## 4 Screenshot of program output

File Edit Selection View Go Run Terminal Help device5\_intouch.log.rpt - CPP Projects [WSL: Ubuntu] - Visual Studio Code

EXPLORER

- CPP PROJECTS [WSL: UBUNTU]
  - > vscode
  - > Comp\_4
  - > <vscode>
  - > ps0
  - > ps1a
  - > ps1b
  - > ps2a
  - > ps2b
  - > ps3
  - > ps4a
  - > ps4b
  - > ps5
  - > ps6
  - > ps7
  - > logs
    - device1\_intouch.log
    - device2\_intouch.log
    - device3\_intouch.log
    - device4\_intouch.log
    - device5\_intouch.log
    - device5\_intouch.log.rpt
    - device6\_intouch.log
- > reports
- > LogParser.cpp
- > LogParser.hpp
- > LogParser.o
- > main.cpp
- > main.o
- > Makefile
- > ps7
- > Readme-ps7.md

OUTLINE

LogParser.cpp

device5\_intouch.log.rpt

```
Comp_4 ps7 logs > device5_intouch.log.rpt
1 === Device boot ===
2 31063(logs/device5_intouch.log): 2014-01-26 09:55:07 Boot Started
3 31176(logs/device5_intouch.log): 2014-01-26 09:58:04 Boot Completed
4     Boot Time: 177000ms
5
6 === Device boot ===
7 31274(logs/device5_intouch.log): 2014-01-26 12:15:18 Boot Start
8 **** Incomplete boot ****
9
10 === Device boot ===
11 31293(logs/device5_intouch.log): 2014-01-26 14:02:39 Boot Start
12 31401(logs/device5_intouch.log): 2014-01-26 14:05:24 Boot Completed
13     Boot Time: 165000ms
14
15 === Device boot ===
16 32623(logs/device5_intouch.log): 2014-01-27 12:27:55 Boot Start
17 **** Incomplete boot ****
18
19 === Device boot ===
20 32641(logs/device5_intouch.log): 2014-01-27 12:30:23 Boot Start
21 **** Incomplete boot ****
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

bash-ps7 +

Ln 1, Col 1 Tab Size: 4 UTF-8 LF Plain Text

Figure 10: ps7 .rpt output

## 5 Issues and bugs

For all given logs, the provided implementation works.

If another log file contains a different syntax for the date and time, the current methods would be insufficient.

## 6 Source code for the project

**Makefile**      **Sat Apr 22 20:42:29 2023**      **1**

```
1: CC = g++
2: FFLAGS = -g -Wall -Werror -pedantic -std=c++17
3: LIB = -lsfml-system
4: DEPS = LogParser.o
5:
6: .PHONY: all clean lint
7:
8: all: lint ps7
9:
10: ps7: main.o $(DEPS)
11:       $(CC) $(FLAGS) -o ps7 $^ $(LIB)
12:
13: %.o: %.cpp
14:       $(CC) $(FLAGS) -c $<
15:
16: lint:
17:       cpplint --recursive --quiet .
18:
19: clean:
20:       rm *.o ps7
```

**main.cpp**            **Sat Apr 22 21:13:18 2023**            **1**

```
1: // Copyright 2023 Thomas O'Connor
2: #include "LogParser.hpp"
3:
4: int main(int argc, char* argv[]) {
5:     // Command-line arguments
6:     string filePath = argv[1];
7:
8:     // Open file and check for failure
9:     std::ifstream myLog;
10:    myLog.open(filePath);
11:    if (!myLog) exit(1);
12:    std::ofstream myRpt(filePath + ".rpt");
13:    if (!myRpt) exit(1);
14:
15:    LogParser object(myLog, filePath);
16:    myRpt << object.generateRPT();
17:
18:    myLog.close();
19:    myRpt.close();
20:
21:    return 0;
22: }
```

```
1: // Copyright 2023 Thomas O'Connor
2: #ifndef LOGPARSER_HPP
3: #define LOGPARSER_HPP
4:
5: #include <algorithm>
6: #include <fstream>
7: #include <iostream>
8: #include <regex>
9: #include <string>
10: #include <SFML/System.hpp>
11:
12: #include "boost/date_time posix_time posix_time.hpp"
13: #include "boost/date_time/gregorian/gregorian.hpp"
14:
15: using std::string;
16: using boost::posix_time::ptime;
17: using boost::posix_time::time_from_string;
18:
19: class LogParser {
20: public:
21:     // Constructor
22:     explicit LogParser(std::ifstream& log, const string& filePath) :
23:         _referenceLog(log), _path(filePath) {}
24:     // generator function
25:     const string generateRPT() const;
26:
27: private:
28:     // log file
29:     std::ifstream& _referenceLog;
30:     // file path
31:     string _path;
32: };
33:
34: #endif
```

```
1: // Copyright 2023 Thomas O'Connor
2: #include "LogParser.hpp"
3:
4: const string LogParser::generateRPT() const {
5:     // get all lines of reference text
6:     string line, outputString, startDate, endDate;
7:     unsigned int lineNum = 0;
8:     bool booting = 0;
9:     std::regex rgx1("^(log\\.c\\.166) server started"),
10:        rgx2("oejs\\.AbstractConnector:Started SelectChannelConnec
tor@0\\.0\\.0\\.0:9080");
11:    while (std::getline(_referenceLog, line)) {
12:        lineNum++;
13:        // if "server started" is found, boot began at current line
14:        if (std::regex_search(line, rgx1)) {
15:            // early termination of previous boot sequence. End and resta
rt log
16:            if (booting) {
17:                outputString.append("**** Incomplete boot ****\n\n");
18:            }
19:            booting = true;
20:            outputString.append("== Device boot ==\n");
21:            outputString.append(std::to_string(lineNum));
22:            outputString.append("(" + _path + "): ");
23:            startDate = line.substr(0, 19);
24:            outputString.append(startDate);
25:            outputString.append(" Boot Start\n");
26:            // boot process terminated at current line
27:        } else if (std::regex_search(line, rgx2)) {
28:            booting = false;
29:            outputString.append(std::to_string(lineNum));
30:            outputString.append("(" + _path + "): ");
31:            endDate = line.substr(0, 19);
32:            outputString.append(endDate);
33:            outputString.append(" Boot Completed\n");
34:            outputString.append("\tBoot Time: ");
35:            // calculate difference in time in ms
36:            // construct the times from the strings directly
37:            ptime t1(time_from_string(startDate)), t2(time_from_string(en
dDate));
38:            boost::posix_time::time_duration td = t2 - t1;
39:            outputString.append(std::to_string(td.total_milliseconds()) +
"ms\n\n");
40:        }
41:    }
42:    // outside of the loop: reached end of file
43:    if (booting) {
44:        outputString.append("**** Incomplete boot ****\n\n");
45:    }
46:    return outputString;
47: }
```