Computing IV Sec 202: Project Portfolio

Andrew Allman

Fall 2022

Contents

Time to Complete Portfolio: 24 hours

Chapter 1

PS0

1.1 Hello SFML

1.1.1 Discussion

PS0: Hello SFML, was our first assignment. The assignment involved setting up an environment with the requisite packages and libraries we would be using over the course of the semester. The coding portion of the assignment consisted of designing a simple, interactive program involving a sprite responsive to user keystrokes. Given such basic instructions, there were a few different routes one could take on this assignment: expeditiously fulfill the criteria, or take advantage of the lax guidelines to reacquaint oneself with C++ development. I figured it would serve me well to choose the latter. What you see below is by no means good code, but a promise to myself to execute the forthcoming projects with effort and creativity. Though this effort waned at the start, investment in the following projects started to materialize most prominently at the start of PS3.

Figure 1.1: Screenshot of final gameplay

In Figure 1, you can see the final product - a simple, spaceship game, reminiscent of mini-game formatted levels in early Gameboy Color games. The gameplay was rather limited, consisting of variable speed spaceships flying across an astral field, disappearing through one edge of the map, and reappearing from a variable x or y position on the opposite edge. Controls used WASD for scope movement, and space to fire your onboard laser. Fitted to background was a tense, 8-bit track and responsive laser-fire sounds and accompanying

hit-sounds. When a spaceship was destroyed, the sprite cycled through 5 textures for an animated explosion. Some time in Photoshop was spent adding dirt and grain to the Gameboy skin, and a plastic overlay to the screen to make playing the game feel more nostalgic.

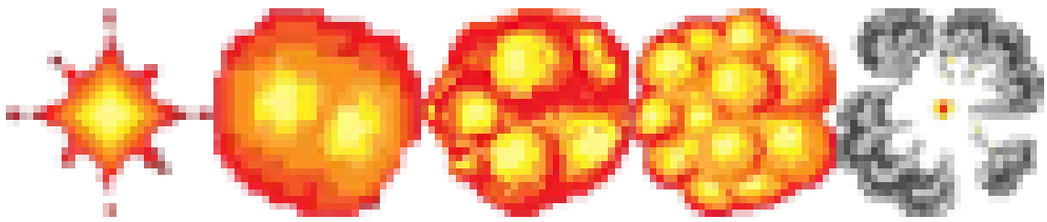


Figure 1.2: Sequence of hit elements used to produced animated ship explosions.
Credit: *VectorStock*

1.1.2 What I accomplished

Looking back at the code, making some halfway decent product seems more a testament to effort and engagement than any pronounced skills coming into the class. Initially, the game was intended to be a series of floating targets, moving across the screen, through one edge and out the other. In recollection, my initial inspiration was the slingshot, shooting gallery mini-game from Nintendo's *Legend of Zelda: Ocarina of Time*. Things clearly took a different shape. Though, no analogous gameplay elements are sampled, I believe I kept in mind *Megaman*, *Oddworld*, and *Super Metroid* when considering different gameplay options (like 3rd Person, character combat). In fact, I believe some of the hit sounds may be repackaged *Megaman* sounds.

From a technical aspect, you may be able to read from the code that a lot of the mechanics were not implemented soundly, or efficiently. Lining up hit-markers with scope/ship positioning was not 1:1, and in fact involved a unique positional offset and allowance for every direction of ship. A lot of specific, and often non-benchmark numbers littered the initial writing of the code, and the solution for readability seemed to repackage each 'magic-number' as a global, const float. In retrospect, with so many constant globals, and enums, for such few elements, perhaps the organization more served the purpose of making the code *appear* neat, while still at heart being the same sweaty, under-thought/overdeveloped 'make-it-work' implementation it was along the way.

I believe that sentiment ties nicely into what I accomplished here. At a certain point, there was a very clear idea of what I wanted to make. I wanted to build a simple, low-skill, space mini-game complete with all the requisite thematic components. And, in shaking off the C++ rust, and exploring the nuances of this new SFML library, I was able to do everything I wanted. Not in the best way, in some cases, not in a good way, but fruitful to completion all the same.

In summation I reinvigorated a discipline, and the making of a technical skill set, that believe would set me on the right track for the semester to come.

1.1.3 What I already knew

The relevant things I knew at the start of this project were procedural programming, data structure usage, and class implementation. I often find when I'm lacking sophistication in either of the latter two, the more cluttered, and explicit the code becomes. As such, I depended heavily on seemingly redundant blocks of either value assignment or logic to handle differentiating cases. But, I could build classes, which made up the interactive graphical elements of the project, constructing and setting static elements, returning relevant object information to the main method, and implementing object specific methods and sequences. I could implement data structures, such as the vector of dynamically allocated ship objects that were iteratively updated with each pass through the while loop in main. I could also manage memory, and free all the ship objects at close.

1.1.4 What I learned

Although, I believe the most obvious lessons learned in this assignment regard familiarization and usage of the SFML library - things like sprites, textures, audio, drawing to a screen,

```
for(auto& ship: vsShips){
                std::pair<float, float> temp_ship_pos = ship->getPosition();
                switch(ship->getDirection()){
                    case UP:
                        if(temp_ship_pos.first + UP_SHOOT_OFFSET_X >=
   temp_crosshair_pos.first - SHOOT_ALLOWANCE &&
                        temp_ship_pos.first + UP_SHOOT_OFFSET_X <=
   temp_crosshair_pos.first + SHOOT_ALLOWANCE &&
                        temp_ship_pos.second + UP_SHOOT_OFFSET_Y >=
   temp_crosshair_pos.second - SHOOT_ALLOWANCE &&
                        temp_ship_pos.second + UP_SHOOT_OFFSET_Y <=
   temp_crosshair_pos.second + SHOOT_ALLOWANCE &&
                        ship->isVisible())
                        {
                            ship->explode();
                            laserhitsfx.play();
                        }
                        break;
                    case DOWN:
                        if(temp_ship_pos.first + DOWN_SHOOT_OFFSET_X >=
   temp_crosshair_pos.first - SHOOT_ALLOWANCE &&
                            temp_ship_pos.first + DOWN_SHOOT_OFFSET_X <=
   temp_crosshair_pos.first + SHOOT_ALLOWANCE &&
                            temp_ship_pos.second + DOWN_SHOOT_OFFSET_Y >=
   temp_crosshair_pos.second - SHOOT_ALLOWANCE &&
                            temp_ship_pos.second + DOWN_SHOOT_OFFSET_Y <=
   temp_crosshair_pos.second + SHOOT_ALLOWANCE &&
                            ship->isVisible())
                        {
                            ship->explode();
                            laserhitsfx.play();
                        }

                        break;
      //  ...continued
```

Figure 1.3: Example of redundancy in main method

etc. - the most useful lesson was one that I often have to learn every so often - which is to plan ahead. Many technically dense, or inventive/creative pursuits require a demanding amount of experimentation, often limiting the capacity for organization. However, this is not the case with every aspect of an intended program, and there are usually basic commonalities between possible divergences that can be accommodated from the jump. I tend to think of generic programming as responsible programming, and in straying away from what is common for the sake of specificity, you often miss an opportune reduction towards simplicity.

In the case of this project, too many differing controls and values existed for each ship. A long logic block exists in a switch statement in the main method that handles user-input laser shooting, that tests a variety of different cases corresponding to each direction of ship. In recognizing that the ships are essentially the same, differing only by speed and direction, I didn't need to create 4 different conditions with slight modifications to hit-marker offsets or allowances or inequalities. I should have corrected, at the source, the reason why these values differed, and implemented a single, positional check of all the ships to see which one was hit. Also, had I thought of efficiency, I would have used some sort of map that search for a ship given the known location of the shot.

So, in addition to learning the basics of the SFML library, at the end of the project, I learned again to plan ahead, consider efficiency before building, and strive towards simplicity.

1.1.5 Challenges

There were a few challenges along the way. Most of them were rooted in ambition and being rusty in C++. Some of the issues I encountered:

- Making the ships visible (required dynamic allocation of sprites)
- Making the ships invisible and non-interactive after destruction (involved a custom 'visible' flag)
- Getting the ships to appear in the window of the Gameboy skin, disappear at the right time and place, and reappear at the right time and place
- Accurate hit-markers/boxes for each variation of ship

There were other challenges, but they were mostly among the same breed of mechanical issues, or unintended behaviors that were resolved in time. No major issues really existed during development, and each challenge was overcome in relatively short order.

1.1.6 Codebase

Makefile

```
CC = g++
CFLAGS = --std=c++14 -Wall -Werror -pedantic
LIB = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system -
   lboost_unit_test_framework

all: sfml-app

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

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

clean:
    rm *.o sfml-app
```

main.cpp

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include<iostream>
#include<vector>
#include<ctime>

//CONSTANT GLOBAL VALUES
const float WINDOW_LENGTH = 500, WINDOW_HEIGHT = 859, PLAYABLE_WIDTH = 300,
   PLAYABLE_HEIGHT = 270;
const float GAMEBOY_SKIN_WIDTH = 500, GAMEBOY_SKIN_HEIGHT = 859;
const float UP_SHOOT_OFFSET_X = 10, UP_SHOOT_OFFSET_Y = 20;
const float DOWN_SHOOT_OFFSET_X = -25, DOWN_SHOOT_OFFSET_Y = -25;
const float LEFT_SHOOT_OFFSET_X = 10, LEFT_SHOOT_OFFSET_Y = -25;
const float RIGHT_SHOOT_OFFSET_X = -30, RIGHT_SHOOT_OFFSET_Y = 10;
const float SCREEN_X_OFFSET = 100, SCREEN_Y_OFFSET = 80;
const float THRUSTER_SIZE = 30;
const float SHIP_SIZE = 32;
const float SHOOT_ALLOWANCE = 15;
const float DELAY_MS = 20;
const float MIN_SHIP_SPEED = 200, MAX_SHIP_SPEED = 400;
const float CROSSHAIR_MOVE_VALUE = 5;
const float SHIP_COUNT = 50;
```

```
22
23 //GLOBAL ENUMS
24 enum SHIP_DIRECTION {UP, DOWN, LEFT, RIGHT};
25 enum THRUSTER_POWER {T_LOW, T_MID, T_HIGH};
26 enum EXPLOSION_SEQ {E_NOT, E_INIT, E_LOW, E_MID, E_HIGH, E_RESOLVING, E_DONE
       };
27 enum KEY_PRESS {K_UP, K_DOWN, K_LEFT, K_RIGHT};
28
29 //FOR INITIALIZING SHIP POSITION
30 const float getRandX(void);
31 const float getRandY(void);
32
33 //CROSSHAIR CLASS - USER'S CURSOR
34 class crosshair{
35 public:
36     const float RIM_RADIUS = 10.f,
37         RIM_THICKNESS = 2,
38         TICK_LENGTH = 6.7,
39         TICK_THICKNESS = 1,
40         RIM_OFFSET = -2,
41         TICK_CUS_SEMIRAD_OFFSET = -5.3,
42         TICK_CUS_RAD_OFFSET = 8,
43         TICK_CUS_DIAM_OFFSET = 14.7;
44
45     const std::size_t RIM_POINT_COUNT = 100;
46     const sf::Color CROSSHAIR_COLOR = sf::Color::White;
47
48     crosshair();
49     void setPositionMouse(const float x, const float y);
50     void setPositionKeys(const KEY_PRESS);
51     void drawCrosshair(sf::RenderWindow& rendWindow);
52     const std::pair<float,float> getPosition(void) {return _position;}
53
54 private:
55     sf::CircleShape _rim;
56     sf::RectangleShape _uc_tick,_lc_tick,_l_tick, _r_tick;
57     std::pair<float, float> _position;
58 };
59
60 //DEFAULT CONSTRUCTOR - CREATE SPRITE WITH ESTABLISHED VALUES
61 crosshair::crosshair():
62     _rim(sf::CircleShape(RIM_RADIUS, RIM_POINT_COUNT)),
63     _uc_tick(sf::RectangleShape(sf::Vector2f(0, TICK_LENGTH))),
64     _lc_tick(sf::RectangleShape(sf::Vector2f(0, TICK_LENGTH))),
65     _l_tick(sf::RectangleShape(sf::Vector2f(TICK_LENGTH, 0))),
66     _r_tick(sf::RectangleShape(sf::Vector2f(TICK_LENGTH, 0))),
67     _position(std::pair<float, float>(SCREEN_X_OFFSET + (PLAYABLE_WIDTH/
       2.0), SCREEN_Y_OFFSET + (PLAYABLE_HEIGHT / 2.0)))
68
69 {
70     _rim.setFillColor(sf::Color::Transparent);
71     _rim.setOutlineColor(CROSSHAIR_COLOR);
72     _rim.setOutlineThickness(RIM_THICKNESS);
73
74     _uc_tick.setOutlineThickness(TICK_THICKNESS);
75     _lc_tick.setOutlineThickness(TICK_THICKNESS);
76     _l_tick.setOutlineThickness(TICK_THICKNESS);
77     _r_tick.setOutlineThickness(TICK_THICKNESS);
78
```

```
79      _uc_tick.setOutlineColor(CROSSHAIR_COLOR);
80      _lc_tick.setOutlineColor(CROSSHAIR_COLOR);
81      _l_tick.setOutlineColor(CROSSHAIR_COLOR);
82      _r_tick.setOutlineColor(CROSSHAIR_COLOR);
83      setPositionKeys(KEY_PRESS(-1)); //FOR INITIALIZING POSITION (-1 FOR
        TRIGGERING DEFAULT CASE)
84  }
85
86  //ENUM SWITCH FOR KEYSTROKES CONTROLLING MOVEMENT
87  void crosshair::setPositionKeys(const KEY_PRESS key_press){
88      switch (key_press)
89      {
90      case K_UP:
91          _position.second -= CROSSHAIR_MOVE_VALUE;
92          break;
93      case K_DOWN:
94          _position.second += CROSSHAIR_MOVE_VALUE;
95          break;
96      case K_RIGHT:
97          _position.first += CROSSHAIR_MOVE_VALUE;
98          break;
99      case K_LEFT:
100     _position.first -= CROSSHAIR_MOVE_VALUE;
101         break;
102     default:
103         break;
104     }
105
106     _rim.setPosition(_position.first + RIM_OFFSET, _position.second +
        RIM_OFFSET);
107     _uc_tick.setPosition(_position.first + TICK_CUS_RAD_OFFSET, _position.
        second + TICK_CUS_SEMIRAD_OFFSET);
108     _lc_tick.setPosition(_position.first + TICK_CUS_RAD_OFFSET, _position.
        second + TICK_CUS_DIAM_OFFSET);
109     _l_tick.setPosition(_position.first + TICK_CUS_SEMIRAD_OFFSET, _position
        .second + TICK_CUS_RAD_OFFSET);
110     _r_tick.setPosition(_position.first + TICK_CUS_DIAM_OFFSET, _position.
        second + TICK_CUS_RAD_OFFSET);
111 }
112
113
114 //DEPRECATED - GAMEPLAY USED TO BE MOUSE-BASED, BUT WAS TOO EASY
115 void crosshair::setPositionMouse(const float cursor_x, const float cursor_y)
        {
116
117     _rim.setPosition(cursor_x + RIM_OFFSET, cursor_y + RIM_OFFSET);
118
119     _uc_tick.setPosition(cursor_x + TICK_CUS_RAD_OFFSET, cursor_y +
        TICK_CUS_SEMIRAD_OFFSET);
120     _lc_tick.setPosition(cursor_x + TICK_CUS_RAD_OFFSET, cursor_y +
        TICK_CUS_DIAM_OFFSET);
121     _l_tick.setPosition(cursor_x + TICK_CUS_SEMIRAD_OFFSET, cursor_y +
        TICK_CUS_RAD_OFFSET);
122     _r_tick.setPosition(cursor_x + TICK_CUS_DIAM_OFFSET, cursor_y +
        TICK_CUS_RAD_OFFSET);
123
124     _position.first = cursor_x, _position.second = cursor_y;
125
126 }
```

```
127 //DRAWS THE MEMBERS THAT MAKE UP THE CLASS'S SPRITE
128 void crosshair::drawCrosshair(sf::RenderWindow& rendWindow){
129     rendWindow.draw(_rim);
130     rendWindow.draw(_uc_tick);
131     rendWindow.draw(_lc_tick);
132     rendWindow.draw(_l_tick);
133     rendWindow.draw(_r_tick);
134 }
135
136 //SHIP CLASS - THE TARGET THAT USER SHOOTS AT
137 class ship{
138 public:
139     void moveShip(void);
140     void setPosition(const float x, const float y);
141     void animateEngine(void);
142     void animateExplosion(void);
143     void draw(sf::RenderWindow& rendWind) const;
144     const bool isExploding(void) const{return _exploding;}
145     void explode(void);
146     const std::pair<float, float>& getPosition(void) const {return _position
    ;}
147     const bool isVisible(void){return _visible;}
148     const SHIP_DIRECTION getDirection(void){return _direction;}
149     ship(const SHIP_DIRECTION direction, const float initX, const float
    initY, const float speed);
150 private:
151     sf::Sprite _ship;
152     sf::Texture _ship_texture;
153     sf::Sprite _thrusters;
154     sf::Texture _thruster_texture;
155     bool _exploding;
156     bool _visible;
157     float _speed;
158     THRUSTER_POWER _thruster_power;
159     SHIP_DIRECTION _direction;
160     EXPLOSION_SEQ _explosion_seq;
161     std::pair<float, float> _position;
162
163 };
164 //CONSTRUCTOR TAKING SHIP'S DIRECTION, INITIAL X VALUE, INITIAL Y VALUE, AND
     SHIP SPEED
165 ship::ship(const SHIP_DIRECTION direction, const float initX, const float
    initY, const float speed)
166 {
167     _direction = direction;
168     _thruster_power = T_LOW;
169     _speed = speed;
170     _visible = true;
171     _position.first = initX, _position.second = initY;
172
173     _ship_texture.loadFromFile("Main Ship_Base.png");
174     _thruster_texture.loadFromFile("Main Ship_Thrusters_low.png");
175     _thrusters.setTextureRect(sf::IntRect(0,0,THRUSTER_SIZE,THRUSTER_SIZE));
176     _thrusters.setTexture(_thruster_texture);
177     _ship.setTextureRect(sf::IntRect(0,0,SHIP_SIZE,SHIP_SIZE));
178     _ship.setTexture(_ship_texture);
179
180     //ROTATION FOR DIRECTION
181     switch (_direction)
```

```
    {
        case DOWN:
            _ship.setRotation(180.f);
            _thrusters.setRotation(180.f);
            break;
        case LEFT:
            _ship.setRotation(270.f);
            _thrusters.setRotation(270.f);
            break;
        case RIGHT:
            _ship.setRotation(90.f);
            _thrusters.setRotation(90.f);
            break;
        default:
            break;
    }
}
//INITIALIZES EXPLODE SEQUENCE
void ship::explode(){
    _explosion_seq = E_INIT;
    _exploding = true;
}

//EXPLOSION SEQUENCE, WITH EACH PASS THROUGH THE WHILE IN MAIN, IF EXPLODING
    , TRIGGERS NEXT TEXTURE IN SEQUENCE
void ship::animateExplosion(void){
    switch(_explosion_seq){
        case E_NOT:
            break;
        case E_INIT:
            _ship_texture.loadFromFile("explode_1.png");
            _explosion_seq = E_LOW;
            break;
        case E_LOW:
            _ship_texture.loadFromFile("explode_2.png");
            _explosion_seq = E_MID;
            break;
        case E_MID:
            _ship_texture.loadFromFile("explode_3.png");
            _explosion_seq = E_HIGH;
            break;
        case E_HIGH:
            _ship_texture.loadFromFile("explode_4.png");
            _explosion_seq = E_RESOLVING;
            break;
        case E_RESOLVING:
            _ship_texture.loadFromFile("explode_5.png");
            _explosion_seq = E_DONE;
            break;
        case E_DONE:
            _visible = false;
    }
}
//THRUSTER SEQUENCE - LOOPS THROUGH TEXTURES
void ship::animateEngine(void){
    switch (_thruster_power)
    {
        case T_LOW:
            _thruster_texture.loadFromFile("Main Ship_Thrusters_mid.png");
```

```
240             _thruster_power = T_MID;
241             break;
242         case T_MID:
243             _thruster_texture.loadFromFile("Main Ship_Thrusters_high.png");
244             _thruster_power = T_HIGH;
245             break;
246         case T_HIGH:
247             _thruster_texture.loadFromFile("Main Ship_Thrusters_low.png");
248             _thruster_power = T_LOW;
249             break;
250         default:
251             break;
252     }
253 
254 }
255 //DRAWS THE SHIP, WON'T DRAW THRUSTER IF EXPLODING
256 void ship::draw(sf::RenderWindow& rendWind) const{
257     rendWind.draw(_ship);
258     if(!_exploding){
259         rendWind.draw(_thrusters);
260     }
261 }
262 //MOVEMENT DEPENDING ON DIRECTION OF SHIP - BOUNDS ARE SET TO THE POSITION/
    BOUNDS OF THE GAMEBOY SCREEN
263 //IF STATEMENTS CHECK IF OUT OF BOUNDS AND RESETS VIOLATING DIMENSION TO
    EITHER BOUND, AND RANDOMIZES THE OTHER
264 void ship::moveShip(){
265     switch (_direction)
266     {
267         case UP:
268             if(_position.second - _speed <=  SCREEN_Y_OFFSET - SHIP_SIZE){
269                 _position.second = PLAYABLE_HEIGHT + SHIP_SIZE;
270                 _position.first = getRandX();
271             }
272             _position.second -= _speed;
273             break;
274         case DOWN:
275             if(_position.second + _speed >= SCREEN_Y_OFFSET +
    PLAYABLE_HEIGHT + SHIP_SIZE){
276             _position.second = SCREEN_Y_OFFSET - SHIP_SIZE;
277             _position.first = getRandX();
278             }
279             _position.second += _speed;
280             break;
281         case LEFT:
282             if(_position.first - _speed <= SCREEN_X_OFFSET - SHIP_SIZE){
283                 _position.first = PLAYABLE_WIDTH + SCREEN_X_OFFSET +
    SHIP_SIZE;
284                 _position.second = getRandY();
285             }
286             _position.first -= _speed;
287             break;
288         case RIGHT:
289             if(_position.first + _speed >= SCREEN_X_OFFSET + PLAYABLE_WIDTH
    + SHIP_SIZE){
290                 _position.first = SCREEN_X_OFFSET - SHIP_SIZE;
291                 _position.second = getRandY();
292             }
293             _position.first += _speed;
```

```
        default:
            break;
    }
    _ship.setPosition(_position.first, _position.second);
    _thrusters.setPosition(_position.first, _position.second);
}

void ship::setPosition(float x, float y){
    _ship.setPosition(x, y);
}

//RETURNS RANDOM Y VALUE (FOR INITIALIZING/OOB SHIP RESET)
const float getRandY(void){
    return rand() % int(PLAYABLE_HEIGHT) + SCREEN_Y_OFFSET;
}
//RETURNS RANDOM X VALUE (FOR INITIALIZING/OOB SHIP RESET)
const float getRandX(void){
    return (rand() % int(PLAYABLE_WIDTH)) + SCREEN_X_OFFSET;
}
//PROVIDES VARIABILITY IN SHIP SPEED
const float getRandSpeed(void){
    const float conversion = 0.01;
    return ((rand() % int(MAX_SHIP_SPEED)) + (MIN_SHIP_SPEED)) * conversion;
}


int main()
{
    srand(static_cast<unsigned int>(time(0)));

    sf::RenderWindow window(sf::VideoMode(WINDOW_LENGTH, WINDOW_HEIGHT), "
   Main Window");

    sf::Sprite window_bg;
    sf::Sprite gameboy_wrapper;
    sf::Texture gameboy_wrapper_texture;
    sf::Texture window_bg_texture;

    sf::Music background_music;
    sf::SoundBuffer lasersfx_buffer;
    sf::Sound lasersfx;
    sf::SoundBuffer laserhitsfx_buffer;
    sf::Sound laserhitsfx;


    crosshair sCrossHair;
    std::vector<ship*> vsShips;


    background_music.openFromFile("battle_music.wav"); //MAIN MUSIC
    background_music.setLoop(true);
    background_music.play();

    lasersfx.setVolume(70);
    laserhitsfx.setVolume(70);
    laserhitsfx_buffer.loadFromFile("hit_sfx.wav");//HIT SFX
    lasersfx_buffer.loadFromFile("laser_gun.wav");//SHOOT SFX

    laserhitsfx.setBuffer(laserhitsfx_buffer);
```

```
352     lasersfx.setBuffer(lasersfx_buffer);
353 
354     window.setMouseCursorVisible(false);
355 
356     gameboy_wrapper_texture.loadFromFile("gameboy_body.png");
357     window_bg_texture.loadFromFile("space_background.png");
358 
359     gameboy_wrapper.setTextureRect(sf::IntRect(0,0, GAMEBOY_SKIN_WIDTH,
    GAMEBOY_SKIN_HEIGHT));
360     gameboy_wrapper.setTexture(gameboy_wrapper_texture);
361     window_bg.setTextureRect(sf::IntRect(0, 0, PLAYABLE_WIDTH,
    PLAYABLE_HEIGHT));
362     window_bg.setTexture(window_bg_texture);
363     window_bg.setColor(sf::Color::White);
364     window_bg.setPosition(SCREEN_X_OFFSET, SCREEN_Y_OFFSET);
365 
366     //iINITIALIZES {SHIP_COUNT} SHIPS, ALTERNATES DIRECTION
367     for(int i = 0; i < SHIP_COUNT; i++){
368         ship* tShip = new ship(SHIP_DIRECTION(i % 4), getRandX(), getRandY()
    , getRandSpeed());
369         vsShips.push_back(tShip);
370     }
371 
372     while (window.isOpen())
373     {
374         sf::Event event;
375         while (window.pollEvent(event))
376         {
377             if (event.type == sf::Event::Closed)
378                 window.close();
379         }
380 
381         window.clear();
382         window.draw(window_bg);
383 
384         //IF SHOOT BUTTON/SPACE IS PRESSED, CHECK IF HIT (ADD FINE TUNE HIT
    MARKERS RELATIVE TO CROSSHAIR MIDDLE - WAS GUESS AND CHECK, SO COULD BE
    TIGHTER)
385         //SHOOT ALLOWANCE FOR CLIPPING, ETC
386         if(sf::Keyboard::isKeyPressed(sf::Keyboard::Space)){
387             lasersfx.play();
388             std::pair<float,float> temp_crosshair_pos = sCrossHair.
    getPosition();
389             for(auto& ship: vsShips){
390                 std::pair<float, float> temp_ship_pos = ship->getPosition();
391                 switch(ship->getDirection()){
392                     case UP:
393                         if(temp_ship_pos.first + UP_SHOOT_OFFSET_X >=
    temp_crosshair_pos.first - SHOOT_ALLOWANCE &&
394                         temp_ship_pos.first + UP_SHOOT_OFFSET_X <=
    temp_crosshair_pos.first + SHOOT_ALLOWANCE &&
395                         temp_ship_pos.second + UP_SHOOT_OFFSET_Y >=
    temp_crosshair_pos.second - SHOOT_ALLOWANCE &&
396                         temp_ship_pos.second + UP_SHOOT_OFFSET_Y <=
    temp_crosshair_pos.second + SHOOT_ALLOWANCE &&
397                         ship->isVisible())
398                         {
399                             ship->explode();
400                             laserhitsfx.play();
```

```
401                         }
402                         break;
403                     case DOWN:
404                         if(temp_ship_pos.first + DOWN_SHOOT_OFFSET_X >=
    temp_crosshair_pos.first - SHOOT_ALLOWANCE &&
405                             temp_ship_pos.first + DOWN_SHOOT_OFFSET_X <=
    temp_crosshair_pos.first + SHOOT_ALLOWANCE &&
406                             temp_ship_pos.second + DOWN_SHOOT_OFFSET_Y >=
    temp_crosshair_pos.second - SHOOT_ALLOWANCE &&
407                             temp_ship_pos.second + DOWN_SHOOT_OFFSET_Y <=
    temp_crosshair_pos.second + SHOOT_ALLOWANCE &&
408                             ship->isVisible())
409                         {
410                             ship->explode();
411                             laserhitsfx.play();
412                         }
413
414                         break;
415
416                     case LEFT:
417                         if(temp_ship_pos.first + LEFT_SHOOT_OFFSET_X >=
    temp_crosshair_pos.first - SHOOT_ALLOWANCE &&
418                             temp_ship_pos.first + LEFT_SHOOT_OFFSET_X <=
    temp_crosshair_pos.first + SHOOT_ALLOWANCE &&
419                             temp_ship_pos.second + LEFT_SHOOT_OFFSET_Y >=
    temp_crosshair_pos.second - SHOOT_ALLOWANCE &&
420                             temp_ship_pos.second + LEFT_SHOOT_OFFSET_Y <=
    temp_crosshair_pos.second + SHOOT_ALLOWANCE &&
421                             ship->isVisible())
422                         {
423                             ship->explode();
424                             laserhitsfx.play();
425                         }
426                         break;
427                     case RIGHT:
428
429                         if(temp_ship_pos.first + RIGHT_SHOOT_OFFSET_X >=
    temp_crosshair_pos.first - SHOOT_ALLOWANCE &&
430                             temp_ship_pos.first + RIGHT_SHOOT_OFFSET_X <=
    temp_crosshair_pos.first + SHOOT_ALLOWANCE &&
431                             temp_ship_pos.second + RIGHT_SHOOT_OFFSET_Y >=
    temp_crosshair_pos.second - SHOOT_ALLOWANCE &&
432                             temp_ship_pos.second + RIGHT_SHOOT_OFFSET_Y <=
    temp_crosshair_pos.second + SHOOT_ALLOWANCE &&
433                             ship->isVisible())
434                         {
435                             ship->explode();
436                             laserhitsfx.play();
437                         }
438                     }
439
440
441             }
442         }
443         //DRAW SHIPS - IF EXPLODING DO EXPLODING SEQUENCE, DON'T DRAW IF NOT
     VISIBLE
444         for(auto& ship: vsShips){
445             if(ship->isVisible()){
446                 if(!ship->isExploding()){
```

```
                    ship->moveShip();
                    ship->animateEngine();
                }
                else{
                    ship->animateExplosion();
                }
                ship->draw(window);
            }
        }

         if(sf::Keyboard::isKeyPressed(sf::Keyboard::W)){
             sCrossHair.setPositionKeys(K_UP);
         }
         else if(sf::Keyboard::isKeyPressed(sf::Keyboard::A)){
             sCrossHair.setPositionKeys(K_LEFT);
         }
         else if(sf::Keyboard::isKeyPressed(sf::Keyboard::S)){
             sCrossHair.setPositionKeys(K_DOWN);
         }
         else if(sf::Keyboard::isKeyPressed(sf::Keyboard::D)){
             sCrossHair.setPositionKeys(K_RIGHT);
         }
         sCrossHair.drawCrosshair(window);
         window.draw(gameboy_wrapper);

         window.display();
         sf::sleep(sf::Time(sf::milliseconds(DELAY_MS)));
    }
    //CLEAN UP SHIPS
    for(auto& ship : vsShips){
        delete ship;
    }
    return 0;
}
```

Chapter 2

PS1

2.1 PS1a: LFSR

2.1.1 Discussion

PS1a marked the beginning of development for what would become the main data structure used in an image encryption utility. As such, we were tasked with developing a linear feedback shift register, more specifically, a Fibonacci LFSR. The structure of this register involves an 1-Dimensional array of fixed size, filled with binary values. Two primary methods are used modify the values: a step function, which left shifts the values through the array, while setting the rightmost value equal to the XOR product of several binary values, initially held in 'tap' positions, and a generate function, which performs the step function, and returns a bit string representative of the binary generated at each step.

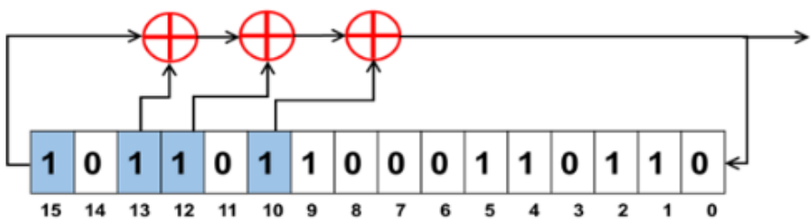


Figure 2.1: A LFSR with taps at positions 10, 12, and 13.
Credit: *Dr.Daly*

2.1.2 What I accomplished

My initial attempt at this assignment took place on a flight to California for a work conference. Admittedly, it was quite bad and was completed in under an hour using a vector to hold int values transcribed from an input string. I could not get the tests to compile, and submitted a largely unfinished product.

However, the second time around, with some rest and quiet, I determined the best implementation, as with really any data structure meant to hold binary values, was a bitfield object. As such, I implemented the data structure as an approximate allocation of required `int`s, and set/unset bits of the array to best simulate LFSR functionality. The implementation borrowed heavily from concepts of data structures learnt in Computing II, and it felt rewarding knowing past knowledge and effort came to serve me well in a future application.

I would say the accomplishment in this project is derived from making a very space efficient representation of the register, as well as creating a very close analog to physical register functionality (in the sense of modulating actual bit values), compared to the original implementation which was oversized, and only replicated functionality by performing similar arithmetic behavior. In some sense, a vector of `int`s could represent an LFSR and moving values through a vector could look like shifting values through a register, but the mechanics are fundamentally different, and the data could represented on a much lower, accurate level via direct bit manipulation.

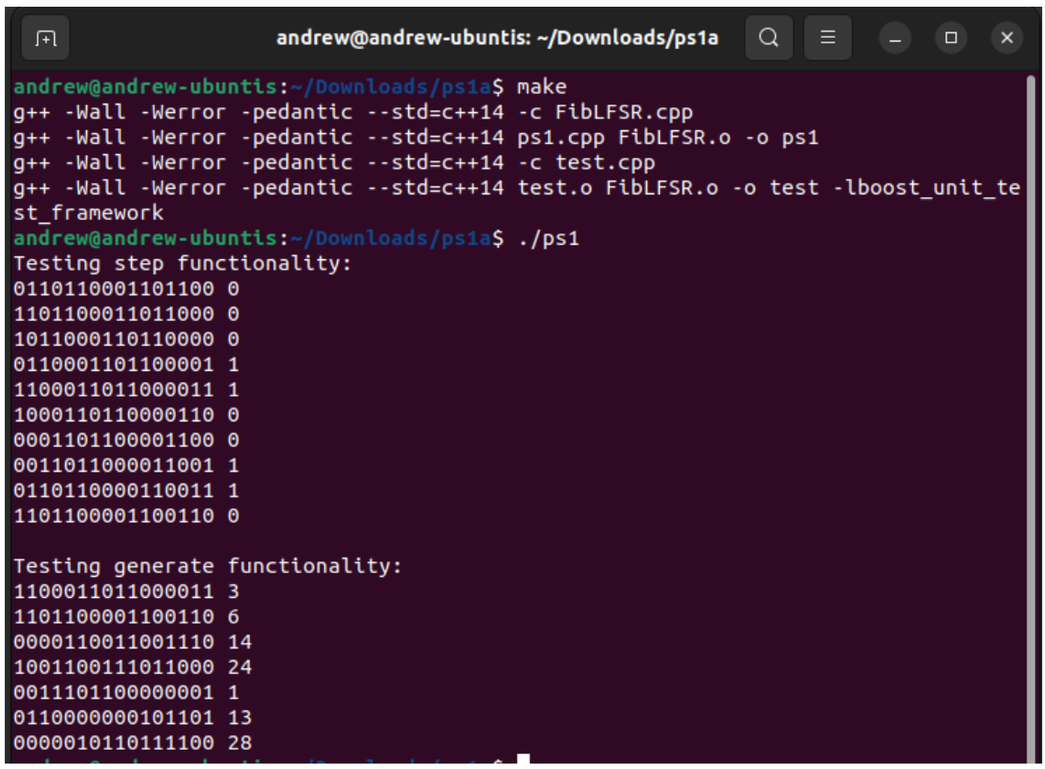


Figure 2.2: Output of the program, demonstrating functionality of methods `step()` and `generate()`

2.1.3 What I already knew

The relevant things I knew at the start of this project were bitfield objects, setting and un-setting binary values in a bitfield, shifting arithmetic, and class design. I knew that bit manipulation could be performed on a block of allocated memory with bitwise operators, to represent/manipulate bit strings. And I knew that this implementation, if done correctly and fit the use-case, was incredibly space efficient. I also knew that C++ class design allowed for easy maneuverability of this object as an `int*` member variable.

2.1.4 What I learned

To be precise, I learned about the functionality of a Fibonacci LFSR. Although I wouldn't quite know the extent of the use-case until part b of the project, I was confident there was some relevant encryption application for altering a sequence of bits via an unpredictable but 'known-to-me' scheme, such that data could be translated into obscurity.

Otherwise, I hadn't come across a relevant application for a bitfield object. If that seems obtuse, I am aware that there are an incredible amount of things that can be represented via a sequence of binary values, and that this principle is leverageable, to a significant benefit, across computing. However, with modern computing performance, and small scale applications, considering things like maximal space efficiency as a primary objective might needlessly stand in the way of development. This project served to remind me to expand my consideration, and temper my reflexes - that a little thought outside my usual bag of tricks may yield a simple, effective design.

2.1.5 Challenges

Again, there were few salient challenges throughout this project. The biggest challenge was not technical, performing the first attempt on a crowded airplane after a restless night and early morning. Once I knew the better implementation, there were few mechanical struggles that mostly resembled remembering the nuances of a proper bitfield implementation. Again, these were resolved in relatively short order.

However, there was one particular issue at first resulting from over-shifting the values in the register, as shifting involved the bitwise `«` operator, and over time would produce a value larger than allocation could accommodate. As such, after every shift, the bit string would be `AND`'d with a bit string of all 1's of length the size of the intended bitstring.

2.2 PS1b: PhotoMagic

2.2.1 Discussion

PS1b called for the utilization of our newly implemented data structure with the aim of encrypting and decrypting an image, provided input image file path, output image filepath, and a starting seed to initialize our LFSR.

Figure 2.3: Image used to test encryption/decryption. I generally use this image to test image applications. I'm also partial to a particular image of a sloth.

The implementation was rather straightforward, and involved reading each RGB color value for every pixel in the image, and `XOR`'ing this value with an 8-bit string returned from our LFSR's generate function. These values became the new pixel values in an image object we would export as our encrypted image file.

Figure 2.4: Demonstration of encryption. As you can see, pixel values are successfully obscured. Inputs given are the file paths and an alphanumeric passcode (to be discussed later.)

2.2.2 What I accomplished

This project, in essence, was very rewarding. I was happy with the implementation of the underlying data structure, and was put to use with great ease for an extraordinarily practical use-case. I was able to develop a Fibonacci LFSR that accurately simulated physical register functionality, and installed it in a program that iteratively performed a simple operation on a few values, and rendered data completely obfuscated, and, after performing the operation

again returned the data back to its original form. This project revealed the power of good, simple ideas.

Additionally, the extra credit portion of this assignment came with an easy implementation. Previous work on past projects planted the idea of interchangeability of data types, so when tasked with translating a passcode into a bit sequence seed for the register, I considered the password a numeric value, a non-unique sum of it's ASCII values, and generated the bit string that corresponded with this number. In retrospect, this isn't entirely secure, as any anagram should yield the same corresponding string, but that could be cured with some additional operation on a given character's ASCII value, with respect to it's position in the passcode.

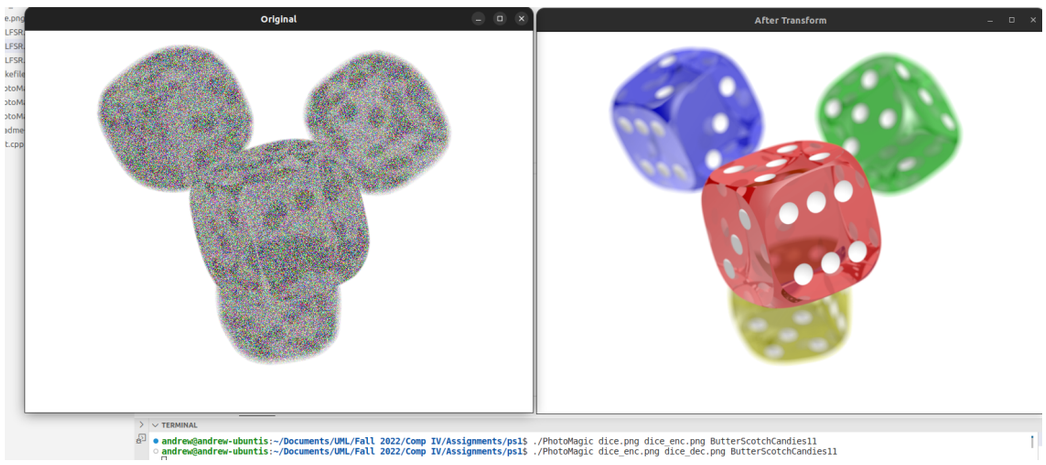


Figure 2.5: Demonstration of decryption with running code. Pixel values are returned to their original state. Inputs given are the same as for encryption.

2.2.3 What I already knew

The relevant things I knew at the start of this project mostly concerned understandings of the Fibonacci LFSR, as provided by the previous project. Developing the data structure provided some very useful intuitions regarding how it should be used. When given the project for image encryption, suddenly a more complete understanding fell into place. This understanding made installation of the LFSR into the project quite natural.

Additionally, with the passcode based implementation, having worked on projects in the past that needed to consider words in terms of numerical values, made the 'not-really-hashing' hashing function, that delivered a bit sequence from the code, fairly straightforward to write.

2.2.4 What I learned

What I learned throughout this project was mainly an extension of the lesson I learned in part a, which was to consider the properties of different data structures and objects to make implementations more efficient and easy to simplistic. In this case, a simple register-like data structure that modified internal values and generated usable values (as some byproduct of internal manipulation), lent itself perfectly to a data encryption application.

Additionally, I furthered my understanding of this register device, and the sequential bitwise operations it should perform in conjunction with data. I initially had an intuition

```
1 std::string returnEasyHash(std::string passcode){
2     int sum = 0;
3     for(char& ch : passcode){
4         sum += int(ch);
5     }
6     return std::bitset<16>(sum).to_string();
7 }
```

Figure 2.6

that data could be obscured with this mechanism, however (and a fairly obvious recognition now) hadn't realized that the same sequence of operations could be performed to return obscured data to its original state.

2.2.5 Challenges

I believe I initially encountered some difficulties with modifying the pixel values. The initial version of the provided code demonstrated the alteration of color values by subtracting starting values from 255 to generate a negative of the image. It wasn't until a little later I realized, to make the operation reversible, the operation performed on the color values should be an `XOR` with a generated bit string from the LFSR.

2.2.6 Codebase

Makefile

```
FLAGS= -Wall -Werror -pedantic --std=c++14
all: PhotoMagic test

PhotoMagic: PhotoMagic.o FibLFSR.o
    g++ ${FLAGS} PhotoMagic.o FibLFSR.o -o PhotoMagic -lsfml-graphics -lsfml
   -window -lsfml-system

PhotoMagic.o: PhotoMagic.cpp FibLFSR.cpp
    g++ ${FLAGS} -c PhotoMagic.cpp

FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
    g++ ${FLAGS} -c FibLFSR.cpp

test: test.o FibLFSR.o
    g++ ${FLAGS} test.o FibLFSR.o -o test -lboost_unit_test_framework

test.o: test.cpp FibLFSR.hpp
    g++ ${FLAGS} -c test.cpp
clean:
    rm *.o
    -rm PhotoMagic
    -rm test
```

FibLFSR.hpp

```
#include<iostream>
#include <SFML/Graphics.hpp>

//global consts for system independence 32/64
const int BITS_PER_BYTE = 8;
const int BYTES_PER_INT = sizeof(int);

class FibLFSR{
public:
    FibLFSR(std::string seed);
    int step(void);
    int generate(int k);
    void set(const int index);
    void unset(const int index);
    bool at(int index);
    const int getSize(void)const;
    const int getLimit(void)const;
    ~FibLFSR(){delete [] _bit_string;}
private:
    int* _bit_string;
```

```
    int _size;
    int _limit;
};

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

FibLFSR.cpp

```
#include<iostream>
#include<vector>
#include<iostream>
#include <SFML/Graphics.hpp>
#include"FibLFSR.hpp"

//class implementations

//helper function - to keep bitstring_data from scaling massively,
//each call to step limits the bitstring_data by ANDing it's value
//by the maximum value of that size 111...1
int generate_limit(int size){
    int k = 0;
    for(int i = 0; i < size; i++){
        k = (k << 1) + 1;
    }
    return k;
}



//Constructor, creates a bit field object for representing bit string
//The field is initialized to zero, and bits are flipped as necessary while reading in
//characters from the seed
FibLFSR::FibLFSR(std::string seed) {
    _bit_string = new int [(seed.size() / (BYTES_PER_INT * BITS_PER_BYTE)) + 1]{0};
    _size = seed.size();
    for(int i = _size - 1; i >= 0; i--){
        switch(seed[i]){
            case '1':
            _bit_string[(_size - i - 1) / (BYTES_PER_INT * BITS_PER_BYTE)] |= (1 << (_size - i - 1) % (BYTES_PER_INT * BITS_PER_BYTE));
            break;
            default:
            break;
        }
    }
    _limit = generate_limit(_size);
}

//For indexing field
bool FibLFSR::at(const int index){
    return ((_bit_string[index / (BYTES_PER_INT * BITS_PER_BYTE)]) & (1  << (index % (BYTES_PER_INT * BITS_PER_BYTE))));
}
//returns limit
const int FibLFSR::getLimit(void)const{
    return _limit;
}
//Peforms a step, left shifts the field by 1, and sets the 0th bit to the XOR'd values of the taps and exiting bit
```

```
49 int FibLFSR::step() {
50     int tProd = ((at(15) ^ at(13)) ^ at(12)) ^ at(10);
51     *_bit_string <<= 1;
52     *_bit_string &= _limit;
53     switch (tProd)
54     {
55     case 1:
56         set(0);
57         break;
58     default:
59         unset(0);
60         break;
61     }
62     return tProd;
63 }
64
65 //Built to set bits with bitwise operators - had I exercised more foresight
       I could've just subtracted one where necessary.
66 void FibLFSR::set(const int index){
67     _bit_string[(index) / BITS_PER_BYTE] |= (1 << (index) % BITS_PER_BYTE);
68 }
69 //Built to unset bits with bitwise operators
70 void FibLFSR::unset(const int index){
71     _bit_string[(index) / BITS_PER_BYTE] &= ~(1 << (index) % BITS_PER_BYTE);
72
73 }
74 //returns size
75 const int FibLFSR::getSize(void) const{
76     return _size;
77 }
78
79 std::ostream& operator<<(std::ostream& out, FibLFSR& rhFib){
80     for(int i = rhFib.getSize() - 1; i >= 0; i--){
81         out << rhFib.at(i);
82     }
83     return out;
84 }
85
86 // Simulate k steps and return a k-bit integer
87 int FibLFSR::generate(int k) {
88     int result = 0;
89     for (int i = 0; i < k; i++) {
90         result = (result << 1) + step();
91     }
92     return result;
93 }
```

PhotoMagic.cpp

```
1  #include<bitset>
2  #include <SFML/System.hpp>
3  #include <SFML/Window.hpp>
4  #include <SFML/Graphics.hpp>
5  #include<string>
6  #include "FibLFSR.hpp"
7
8
9  //generates simple hash of alphanumeric passcode
10 //by summing up char values of passcode, and using
11 //the std bitset library to represent value as a binary string
12
```

```
13 std::string returnEasyHash(std::string passcode){
14     int sum = 0;
15     for(char& ch : passcode){
16         sum += int(ch);
17     }
18     return std::bitset<16>(sum).to_string();
19 }
20
21
22 //uses the fibonacci register object's generate function,
23 //to return an 8 bit integer value, which is used to encrypt/decrypt
24 //pixel values via XOR
25 void transform(sf::Image& image, FibLFSR* fibReg)
26 {
27     sf::Color p;
28     for (unsigned int x = 0; x < image.getSize().x; x++) {
29         for (unsigned int y = 0; y < image.getSize().y; y++) {
30             p = image.getPixel(x, y);
31             p.r = p.r ^ fibReg->generate(8);
32             p.g = p.g ^ fibReg->generate(8);
33             p.b = p.b ^ fibReg->generate(8);
34             image.setPixel(x, y, p);
35         }
36     }
37 }
38
39 //The main process takes the following args from command line: input_file,
    outputfile, and alphanumeric passcode.
40 //The input image is transformed, and original and post-transformation
    images are displayed. The transformed image is
41 //saved to the output file name.
42 int main(int argc, char* argv[])
43 {
44     std::string input_file = argv[1], out_file = argv[2], seed =
    returnEasyHash(argv[3]);
45
46     FibLFSR flfsr(seed);
47     sf::Image image_unencrypted;
48
49     if (!image_unencrypted.loadFromFile(input_file))
50         return -1;
51
52     sf::Image image_encrypted(image_unencrypted);
53
54     transform(image_encrypted, &flfsr);
55
56     sf::Vector2u size = image_unencrypted.getSize();
57     sf::RenderWindow window(sf::VideoMode(size.x, size.y), "Original");
58     sf::RenderWindow window2(sf::VideoMode(size.x, size.y), "After Transform
    ");
59
60     sf::Texture unencrypted_img_texture, encrypted_img_texture;
61     sf::Sprite unencrypted_img_sprite, encrypted_img_sprite;
62
63     unencrypted_img_texture.loadFromImage(image_unencrypted);
64     encrypted_img_texture.loadFromImage(image_encrypted);
65     unencrypted_img_sprite.setTexture(unencrypted_img_texture);
66     encrypted_img_sprite.setTexture(encrypted_img_texture);
67
```

```
68      while (window.isOpen() && window2.isOpen())
69      {
70          sf::Event event;
71          while (window.pollEvent(event))
72          {
73              if (event.type == sf::Event::Closed)
74                  window.close();
75          }
76          while(window2.pollEvent(event))
77          {
78              if (event.type == sf::Event::Closed)
79                  window2.close();
80          }
81
82          window.clear(sf::Color::White);
83          window2.clear(sf::Color::White);
84
85          window.draw(unencrypted_img_sprite);
86          window2.draw(encrypted_img_sprite);
87
88          window.display();
89          window2.display();
90      }
91
92      if (!image_encrypted.saveToFile(out_file))
93          return -1;
94
95      return 0;
96  }
```

Chapter 3

PS2

3.1 Snowflake Fractal

3.1.1 Discussion

> "[The] Major decision was to make the triangles inherently dependent on a relative origin for setting position. This would turn out to be a bad idea."
>
> (Andrew Allman : *Readme-ps2.md*)

I largely consider PS2: Snowflake Fractal to be the dud among my projects. Unfortunately, as a rule, I would start my projects either the day of, or the day prior to submission, coding everything in one straight shot. There are obvious disadvantages to this strategy - it is often the case you realize a fundamentally better way to approach development after building something the first time. Had I spared more time, having understood this concept, my programs should have benefited for it. I believe this is the only case among all my projects where this style prevented me from creating a working product.

This project tasked us with developing a recursive method that would create and position a series of custom, triangle graphics objects. The triangles were to be fixed around differing center points, building outward, to create a snowflake-fractal design.

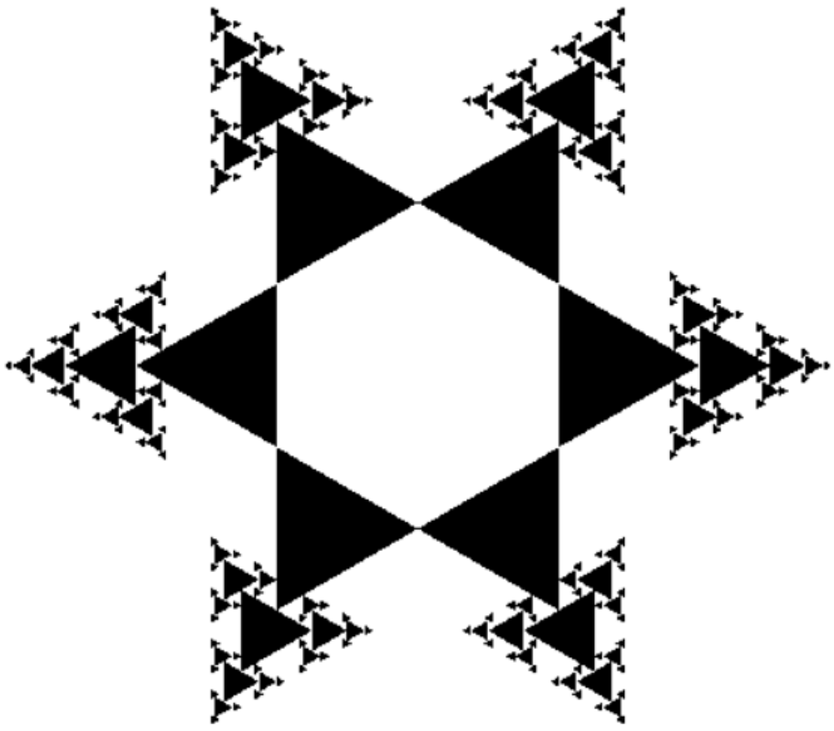


Figure 3.1: Intended final design

In figure 3.1, you can see the intended fractal design, with six large triangles rotated and spread, jointly inscribing a negative circle, their bases forming a hexagon. Smaller triangles

```
1 void snowflake(sf::RenderWindow& window, double L, int divisions, sf::
    Vector2f Center, double start_angle, double magnitude, int n, bool
    first_pass);
```

Figure 3.2: Declaration of recursive drawing method (Snowflake.cpp : 46 - 97)

layer outward, their bases planted on the tip of the previous layer's triangles. This pattern continues until the recursion depth is hit (a variable provided at input), at which point this layering pattern starts to decorate the lengthwise tips, presumably continuing on from the call stack built while creating the initial outward layers.

The concept of fractals, most notably the visualizations representing the Mandelbrot Set, are important in representing and identifying patterns and complexities deriving from, often, simple mathematical rules. I believe this was the intention of the project - develop internal rules for how our custom triangle objects to be positioned, and let this rule set be run recursively, modifying certain values (i.e. size, position), and populating the results into the final fractal design.

Forewarning, the code listed below in section 'Codebase' is not good code. It is largely the result from not adequately understanding how to fully implement this program.

3.1.2 What I accomplished

What I was able to build was about, I would say seventy-five percent of the program. I was able to develop a triangle class with some, seemingly, intuitive behaviors. I was able to get these objects to recursively populate outward (by modifying the size and distance from origin parameters (magnitude)), and I was in the early, experimental stages of writing the logic for the auxiliary/lengthwise triangles.

I suppose there is some accomplishment in terms of sequence, to say the triangles do build outward first, and (when the recursion depth is hit), attempts to start placing new triangles with different logic. It is this different logic that was never fully fleshed out though, and why the program fails to position these auxiliary triangles correctly.

The class design included a default constructor, although callable, shouldn't be used (one of the many residual mistakes), in addition to a value constructor which set the side length, distance from origin, origin point, and rotation value. The scheme of this idea was based on the realization that for any individual triangle, there existed a cluster of triangles with the same center point. In essence, all the triangles positioned while building outward could be rotated around the true center of the figure (only differing in magnitude and size). The auxiliary triangles could manipulated similarly, this time fixed around the centroid of the larger triangles. This actually proved to be somewhat difficult, presumably more difficult than making the origin of each triangle it's own centroid so rotation was independent of any common centerpoint of a cluster.

As you can see in Figure 3.3, the outward layers are placed, then an attempt to place the auxiliary triangles (colored blue) yields a series of stars composed of overlapping triangles.

3.1.3 What I already knew

Coming into this project, I had some grasp on recursion. And as beautiful of a concept as it can be, I find it terribly difficult to intuit recursive solutions. However, I did have a solid enough understanding to proceed. I knew you could reduce a solution to a problem by identifying what the solution to one case looks like, and describing a ruleset that handles variations in cases. In this project, the problem was to position and size a triangle, and my cases were:

- Position around the fractal center point.
- Position around an auxiliary triangle center point.

The recursive implementation to this was not elegant, but I do believe contained the correct elements to be successful given the rest of the design.

Otherwise, I had been acquainted enough at this point with the SFML library to feel comfortable expanding into intermediate-beginner concepts like custom sprite design and rotation around defined center points.

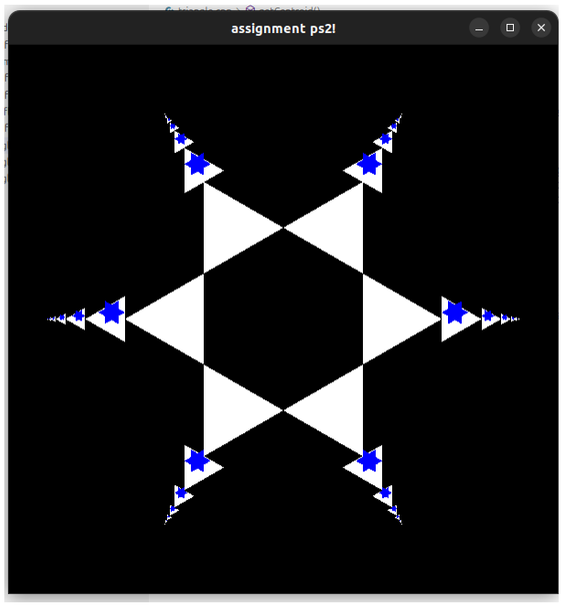


Figure 3.3: Final fractal design for my implementation.

3.1.4 What I learned

Although I can't quite stress enough that this project under-delivered, I did learn a bit about triangular geometry during development. Basic formulas such as

$$Centroid = (\frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3})$$

$$Height = \frac{\sqrt{3}}{2} \cdot L$$

were of great use, in addition to degree to radian approximations like:

$$rad = deg \cdot 0.017453$$

and calculating positions of transformed points, given initial location, point of rotation, and angle:

$$x' = x \cdot cos(-\theta) - y \cdot sin(-\theta)$$
$$y' = x \cdot sin(-\theta) + y \cdot cos(-\theta)$$

I have no doubt used these in the past, but even basic geometrical formulas prune over time, so I consider this refresher in *shape math* one of the primary lessons learned.

In addition to geometry, this project pointed us in the direction of the SFML documentation to learn about custom graphic class design. I learned to create a basic shape from a SF::VertexArray object, and how to define a private method that allowed a rendering window to natively draw my shape. This would be useful in several later projects.

3.1.5 Challenges

There were several problems in the final product, as well as many along the way. I believe, in my first attempts to gain a foothold in proper triangle manipulation, I simply initialized a lot of triangle objects, manually drawing some layers of the fractal design to try and identify patterns of differentiation. After finding success with placement, using a rather convoluted scheme that transformed triangles in space relative to some factor of their length and centroid distances, I decided to go back to the drawing board, revisiting class design.

From there, using the premise that the triangles could be managed more efficiently by cluster (around a shared center point), I found quick success in developing the outward layers. However, I couldn't properly identify the right values to transform the auxiliary triangles by, to produce the intended fractal design. At this point, I had spent a lot of time in initial, fruitless experimentation, and was beginning to realize I was making very little progress on the auxiliary triangles as well. The deadline had already passed, so I roughly

transformed the, then, iterative design into a recursive one, and made some of the triangles blue to shamelessly recoup points lost via cheap extra credit.

Given another chance at this one, I think I would find more success creating triangle objects independent of shared center points, with the origin set to the the centroid, and identify a much simpler recursive scheme. The basic challenges were as follows:

- Handling positioning of auxiliary triangles
- Handling recursion, namely the switch from fractal center point rotation to centroid rotation
- Determining correct class design (standalone triangle class vs. triangle class meant to facilitate manipulation of clusters)
- Determining correct reduction in size between layers of triangles (eventually solved)

3.1.6 Codebase

Makefile

```
FLAGS= -Wall -Werror -pedantic --std=c++14

all: Snowflake lint

Snowflake: Snowflake.o triangle.o
    g++ ${FLAGS} Snowflake.o triangle.o -o Snowflake -lsfml-graphics -lsfml-window -lsfml-system

Snowflake.o: Snowflake.cpp triangle.cpp
    g++ ${FLAGS} -c Snowflake.cpp

triangle.o: triangle.cpp triangle.hpp
    g++ ${FLAGS} -c triangle.cpp

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

triangle.hpp

```
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include<math.h>

class triangle : public sf::Drawable, public sf::Transformable{
public:
    triangle(){};
    triangle(double side_length, double origin_distance, sf::Vector2f origin, double rotation, sf::Color color);

    double getRotate()const {return _rotation;}
    sf::Vector2f getCentroid();

private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
    {
        states.transform *= getTransform();
        target.draw(_shape, states);
    }
```

```
19      sf::VertexArray _shape;
20      sf::CircleShape _center;
21      double _rotation;
22  };
```

triangle.cpp

```
1   #include <SFML/Window.hpp>
2   #include <SFML/Graphics.hpp>
3   #include<math.h>
4
5   #include "triangle.hpp"
6
7   triangle::triangle(double side_length, double origin_distance, sf::Vector2f
        origin, double rotation, sf::Color color) : _shape(sf::Triangles, 3){
8           double height = (sqrt(3)/ 2 * side_length);
9
10          _rotation = rotation;
11          _shape[0].position = sf::Vector2f(origin.x, origin.y -
        origin_distance - height);
12          _shape[1].position = sf::Vector2f(origin.x - (side_length / 2.0),
        origin.y - origin_distance);
13          _shape[2].position = sf::Vector2f(origin.x + (side_length / 2.0),
        origin.y - origin_distance);
14
15          _shape[0].color = color;
16          _shape[1].color = color;
17          _shape[2].color = color;
18      };
19
20  sf::Vector2f triangle::getCentroid(){
21          return sf::Vector2f((_shape[0].position.x + _shape[1].position.x +
        _shape[2].position.x) / 3, (_shape[0].position.y + _shape[1].position.y
        + _shape[2].position.y) / 3);
22      }
```

snowflake.cpp

```
1   #include <SFML/Window.hpp>
2   #include <SFML/Graphics.hpp>
3   #include<iostream>
4   #include<math.h>
5   #include"triangle.hpp"
6
7   //Copyright [2022] <Andrew Allman>
8
9
10  void snowflake(sf::RenderWindow& window, double L, int divisions, sf::
        Vector2f Center, double start_angle, double magnitude, int n, bool
        first_pass);
11
12  int main(int argc, char* argv[]){
13
14      double L = std::stoi(argv[1]), n = std::stoi(argv[2]);
15      double size = 3 * (L / 0.5);
16
17      sf::RenderWindow window(sf::VideoMode(size, size), "assignment ps2!");
18
19      double height = (sqrt(3)/ 2 * L);
20      snowflake(window, L, 6, sf::Vector2f(size/2, size/2), -30, height, n,
        true);
21      sf::Time t = sf::milliseconds(300);
```

```
22
23      while (window.isOpen())
24      {
25          sf::Event event;
26          while (window.pollEvent(event))
27          {
28              if (event.type == sf::Event::Closed)
29                  window.close();
30          }
31          window.display();
32          sf::sleep(t);
33      }
34  }
35
36  double degreesToRadians(double degrees){return degrees * 0.017453;}
37
38  sf::Vector2f getLocation(sf::Vector2f point, sf::Vector2f about, double
        angle){
39      sf::Vector2f tPoint(point - about);
40      double x = tPoint.x * cos(degreesToRadians(-angle)) - tPoint.y * sin(
        degreesToRadians(-angle));
41      double y = tPoint.x * sin(degreesToRadians(-angle)) + tPoint.y * cos(
        degreesToRadians(-angle));
42      return sf::Vector2f(x, y) + about;
43  }
44
45
46  void snowflake(sf::RenderWindow& window, double L, int divisions, sf::
        Vector2f Center, double start_angle, double magnitude, int n, bool
        first_pass){
47      sf::CircleShape center(1);
48      center.setPosition(Center);
49      center.setFillColor(sf::Color::Red);
50      sf::Time t(sf::milliseconds(2000));
51
52
53      double height = (sqrt(3)/ 2 * L);
54
55      std::vector<triangle> triangles{static_cast<long unsigned int>(divisions
        )};
56      std::vector<sf::Transform> rotations{static_cast<long unsigned int>(
        divisions)};
57
58      if(n == 0){
59          return;
60      }
61
62      for(int i = 0; i < divisions; i++){
63          rotations[i] = sf::Transform();
64          rotations[i].rotate(start_angle + (i * (360 / divisions)), Center);
65      }
66
67      for(long unsigned int i = 0; i < triangles.size(); i++){
68          triangles[i] = (triangle(L, magnitude, Center, start_angle + (i *
        (360 / divisions)), sf::Color::White));
69          window.draw(triangles[i], rotations[i]);
70      }
71      snowflake(window, L/2.0, 6, Center, start_angle, height + magnitude, n -
        1, false);
```

```
72
73      if(!first_pass){
74      for(long unsigned int i = 0; i < triangles.size(); i++){
75           for(int j = 0; j < 10; j++){
76                  if(j % 2){
77                      sf::Vector2f centroid_loc = getLocation(triangles[i].
    getCentroid(), Center, triangles[i].getRotate());
78                      triangle p1(L/2, 0, centroid_loc, 30, sf::Color::Blue);
79                      sf::Transform rotate;
80                      p1.setOrigin(p1.getCentroid());
81                      p1.setPosition(p1.getCentroid());
82                      p1.rotate(30 * j);
83                      window.draw(p1, rotate);
84                  }
85                  else {
86                      sf::Vector2f centroid_loc = getLocation(triangles[i].
    getCentroid(), Center, triangles[i].getRotate());
87                      triangle p1(L/2, 0, centroid_loc, -30, sf::Color::Blue);
88                      sf::Transform rotate;
89                      p1.setOrigin(p1.getCentroid());
90                      p1.setPosition(p1.getCentroid());
91                      p1.rotate(-30);
92                      window.draw(p1, rotate);
93                  }
94          }
95      }
96  }
97  }
```

Chapter 4

PS3

4.1 Circular Buffer

4.1.1 Discussion

PS3a: Circular buffer tasked us with the development of the data structure needed to implement the latter part of this assignment. A circular buffer is much like a queue, but functions like a circle, with the head and tail both initially pointing at the same index. Every enqueue operation advances the tail forward one space around the array, while every enqueue operation advances the head forward one space. The clever aspect of a circular buffer is that items aren't actually released until destruction, and so a long as valid positions are marked by the head and tail indices, new entries will overwrite existing data. Possible implementations include 'never-full' versions, wherein enqueing past capacity may overwrite non-dequeued values. This may or may not be the behavior you want, depending on the application, so in our case, we raise an exception if an enqueue is attempted on a full buffer, or a dequeue is called on an empty buffer.

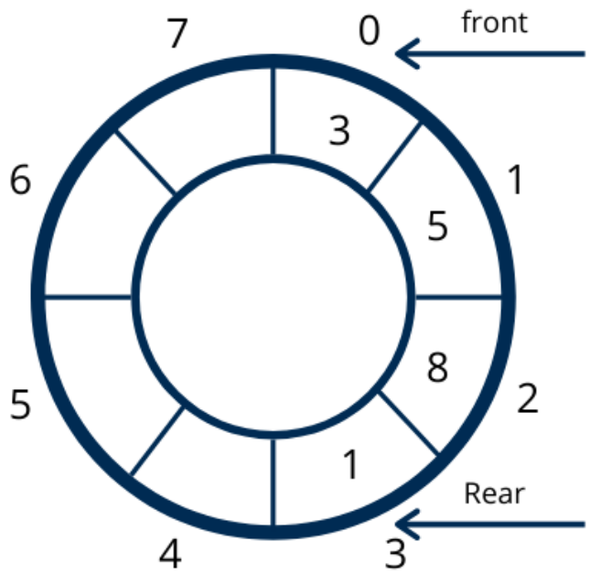


Figure 4.1: Diagram of a circular buffer containing numerical values. This could be the result of four enqueue operations and no dequeue operations.

My implementation keeps tracks of fullness by checking if after an enqueue operation (in which the tails is advanced one space forward in the buffer) the tail index is equal to the head index. If so, a flag is set, and any further enqueue operations will raise an exception.

4.1.2 What I accomplished

This assignment was very mechanically satisfying to implement. I believe initially, after too quickly putting pen to paper, I designed a circular stack, which worked similarly but

```
/* If is full, throws a runtime error, otherwise the tail is set to the new value,
   and the tail advances to the next space. If tail advances to
   the head's position, the buffer is full, and the full flag is set */

template<class T>
void CircularBuffer<T>::enqueue(T item) {
    if (isFull()) {
        throw std::runtime_error("Buffer Full");
    }

    _data[_tail] = item;
    _tail = (_tail + 1) % _capacity;
    _full = _tail == _head;
}

/* If buffer is empty, throws a runtime error, otherwise a temp item is assigned the value of head,
   and the head advances to the next item. Since an item was removed, the full flag is set
   to false. If the next item is the tail, the queue is empty, and since the full flag
   was explicitly set to false, we can reliably assess if the buffer is empty with the isEmpty function */
template<class T>
T CircularBuffer<T>::dequeue() {
    if (isEmpty()) {
        throw std::runtime_error("Queue Empty");
    }

    T retItem = _data[_head];
    _head = (_head + 1) % _capacity;
    _full = false;

    return retItem;
}
```

Figure 4.2: Implementation of enqueue and dequeue operations.

advanced the head and dequeued from the tail index. The fix, as you imagine, was quite straightforward. The enqueue and dequeue operations made use of simple modular arithmetic to keep updated index values in bounds.

Again, it was very rewarding designing a data structure, much like in C, but with the power of C++ templates to make variable storage generalized, in a convenient-to-write way.

This implementation also made use of a lambda expression, passed as a function pointer, to return the size. If the buffer was full, it would return the capacity, otherwise the difference between tail and head. The initial submission noted that I originally wrote the size calculation incorrectly, and was subtracting the tail from the head, then modding the value by the capacity. The unit tests didn't catch this, as I only checked the size when empty and when full.

```
auto evalSize = [](size_t head, size_t tail, size_t capacity, bool is_full)
{return is_full ? capacity : (tail - head) % capacity;};
```

Figure 4.3: Lambda expression used to return size

4.1.3 What I already knew

Having done a fair bit of work in implementing data structures in Computing II, I felt rather enabled to design a lower-level container that made use of rather explicit principles. Having some discrete mathematics background made the use of modular arithmetic very apparent given the name of the structure. Also, Computing III introduced the concept of template programming for generic implementations, which made this particular requirement easy to satisfy.

In effect, I had the all of the programming fundamentals needed to write this in a way that was sound and obvious.

4.1.4 What I learned

In the past, I had heard little of the circular buffer, and in coming into this project hadn't quite known how it should be implemented, or what its most practical use-case was. From a mechanical aspect, this, this initial assignment allowed me to discover proper implementation, with the latter assignment demonstrating a couple valid use cases.

Also, there were a few questions that needed to be addressed in implementation, such as, how do we differentiate the two different sets of conditions in which the head index can equal the tail index? And how does one determine size, when this variability exists? These questions were cut from the same cloth, and were solved in short order with a `bool` member variable that indicated fullness of the buffer. As shown in Figure 4.3, the size could be the capacity when the head index was equal to the tail index, as long as the `full` flag was set to `true`, following an enqueue operation that advanced the tail index to the head index (Figure 4.2). The size could also be zero, under these conditions, so long as the buffer flag `full` was set to false (either by initial conditions, or after any dequeue operation) (Figure 4.2).

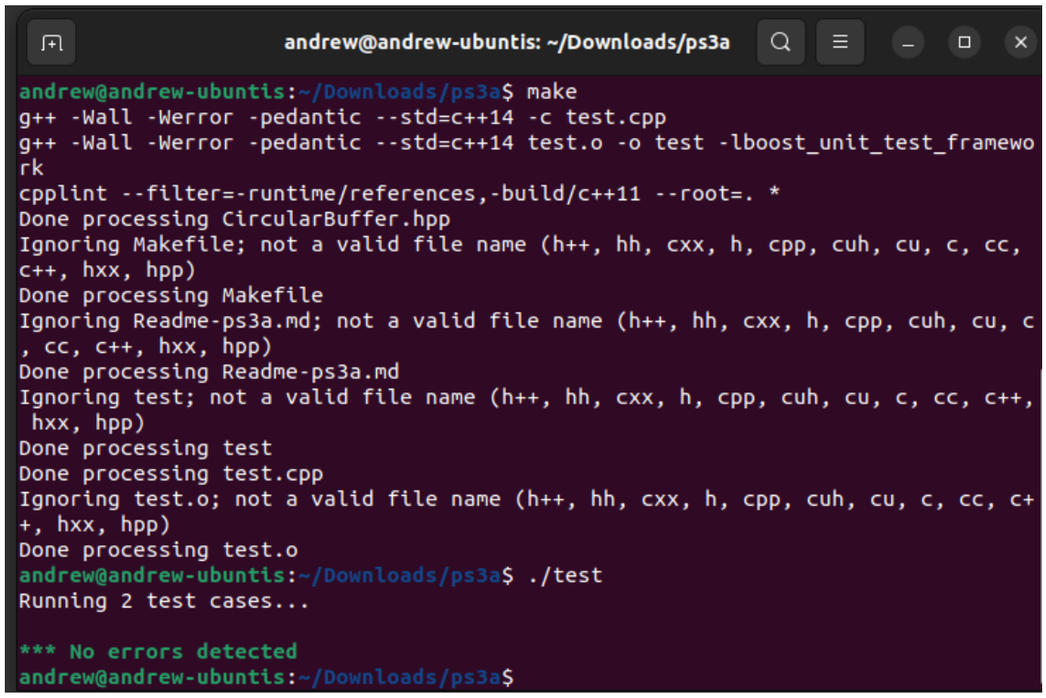


Figure 4.4: Output of successful unit tests, for functionality testing of data structure

4.1.5 Challenges

There were relatively few challenges in creating this data structure. Mid-development questions arose when determining how to calculate size with several cases of the head index equaling the tail index, as discussed in 4.1.4 - this, again, was solved with a `full` flag, that could be utilized in telling these two cases apart.

Otherwise, initially developing the buffer as a stack instead of a queue was also solved by swapping the behaviors of the head and tail indexes.

4.2 StringSound

4.2.1 Discussion

Project PS3b was perhaps my favorite assignment, and the one I worked on the longest. Luckily, this may have been the only assignment (save for PS0) I set aside a proper amount of time for, to pursue something more ambitious than assigned. The instructions tasked us with implementing our circular buffer data structures to hold integer values, the length of which would vary, depending on a desired frequency value.

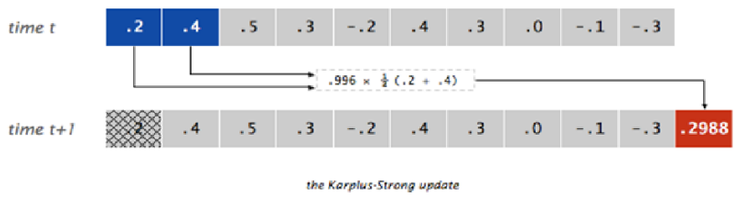


Figure 4.5: Representation of our averaging algorithm via Karplus-Strong
Credit: Dr. Daly

The length of the buffer would be set as the dividend of the frequency and sampling rate (44,100 Hz). The buffer would enqueue random values when member function `pluck()` was called, and update with every call of member function `tick()`. These values could be dequeued into a vector of samples, which we could load into `sf::SoundBuffer` objects, which would then be loaded into `sf::Sound` objects for playback.

Having a small background in music production, I found this process quite fascinating - and quickly implemented the string version to play with the concept a bit more. This lead to the creation of a separate instrument, a customizable, dual-oscillator FM synthesizer.

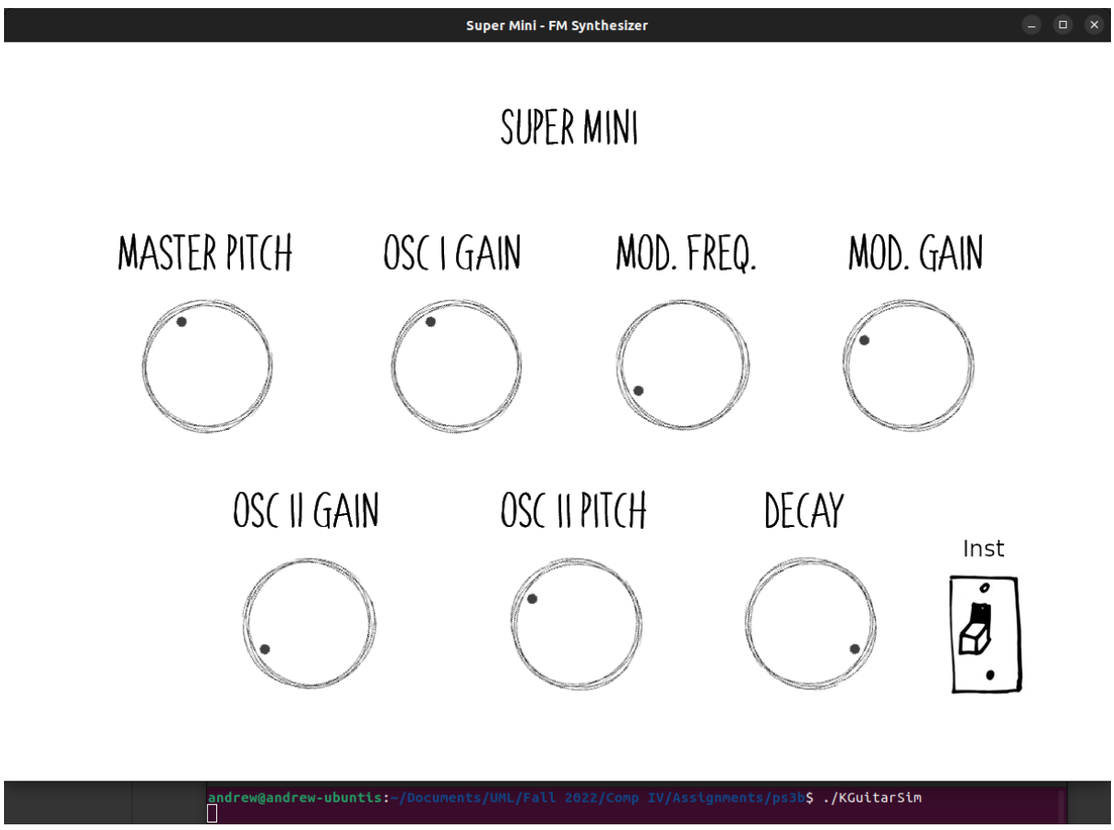


Figure 4.6: Diagram of a circular buffer containing numerical values. This could be the result of four enqueue operations and no dequeue operations.

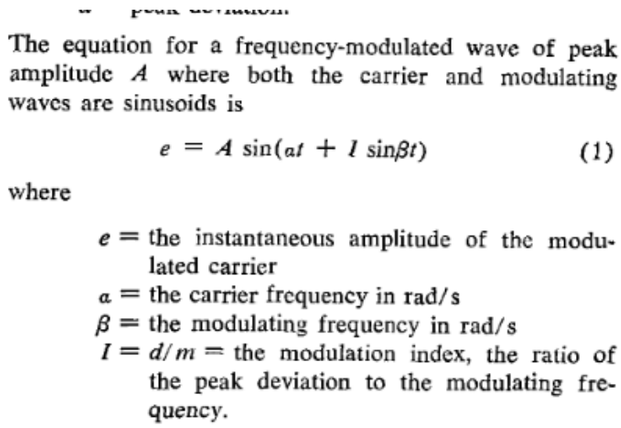
4.2.2 What I accomplished

Shown in Figure 4.6 is final product, an interface facilitating the customization of a number of effect and synth parameters.

My initial experimentation started with an attempt to swap out the guitar simulation with the production of a basic sine wave tone, following the same basic properties. After a fair amount of trial and error, I was able to produce a somewhat brighter/more jagged tone that somewhat resembled a smoother sawtooth wave.

I knew the tone should be smoother, but was certain the math was correct, and chalked the jagged sound up to inadequately large steps through the sine wave values. I somewhat liked the tone and decided to press on, researching a bit on the mechanics of sound design.

Some searching brought me to a concept I had previously heard of, the Frequency Modulation synthesizer. Pursuing this brought me to a 1973 article by composer and Stanford Professor John M. Chowning that described the necessary equations for generating the values of frequency modulated sine waves.



peak deviation.

The equation for a frequency-modulated wave of peak amplitude A where both the carrier and modulating waves are sinusoids is

$$e = A \sin(\alpha t + I \sin\beta t) \qquad (1)$$

where

e = the instantaneous amplitude of the modulated carrier
α = the carrier frequency in rad/s
β = the modulating frequency in rad/s
$I = d/m$ = the modulation index, the ratio of the peak deviation to the modulating frequency.

Figure 4.7: "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation" (1973)

These equations led to the follow implementation as follows Determining `I` value:

```
1  if (_mod_active) {
2      step_mod = X_MAX / (SAMPLING_RATE/(_mod_frequency / 2 * M_PI));
3      for (int i = 0; i < _buffersize; i++) {
4          double e = _mod_gain * sin(_mod_frequency * x_mod);
5          mod_max = e > mod_max ? e : mod_max;
6          mod_min = e < mod_min ? e : mod_min;
7          x_mod += step_mod;
8      }
9      I = (mod_max - mod_min) / _mod_frequency;
10     x_mod = 0;
11 }
```

Figure 4.8: Determining value I, by stepping through modulating wave's sine equation along for the length of the carrier buffer. Note, this wouldn't be the intended value per the formula, only the local max and min, contained to the values of the wave that would be contained to the buffer - the true period for the modulating frequency would typically be much longer than the carrier's

```
1      while (!_buffer->isFull()) {
2          _buffer->enqueue(ceil(
3              _gain * sin(_carrier_frequency * x_carrier +
4                  (I * ((sin(_mod_frequency * x_mod)))))));
5
6          x_carrier += step_carrier;
7          x_mod += step_mod;
8      }
```

Figure 4.9: Enqueuing instantaneous amplitudes generated from Chowning's formula

I then performed the same sequence again, creating another buffer to contain the samples of a second oscillator.

```
    if (_osc2_active) {
        if (_osc2buffer->isFull()) {
            _osc2buffer->empty();
        }

        while (!_osc2buffer->isFull()) {
            _osc2buffer->enqueue(ceil(
                _osc2_gain * sin(_carrier_frequency * x_carrier +
                (I * ((sin(_mod_frequency * x_mod)))))));

            x_carrier += step_carrier;
            x_mod += step_mod;
        }
    }
```

Figure 4.10: Enqueuing instantaneous amplitudes generated from Chowning's formula

This oscillator wasn't inherently independent, and instead, just used a shorter or longer buffer, with alterations to the pitch occurring (I believe) via the `tick()` function, with the OSC II buffer essentially acting like a stretched or condensed portion of OSC I's buffer. This binding mechanism, however, was the basis for creating an independent OSC II pitch control, and Master Pitch control - with the former controlling the length of the OSC II buffer, and the latter simply changing the OSC I carrier frequency. In hindsight, it would have made sense to make both oscillators fully independent, with a master pitch knob adjusting the independent carrier frequencies of both.

The `sample()` method enabled stereo playback, by returning two samples from the same oscillator in succession before returning two samples from the other. Otherwise, playback and stereo would have panned one oscillator entirely to the left and the other to the right.

```
sf::Int16 SynthSound::sample() {
    if (!_buffer->isEmpty()) {
        if (_osc2_active && !_osc2buffer->isEmpty()) {
            switch (_osc2_deq) {
                case 0:
                    _osc2_deq += 1;
                    return (_buffer->peek());
                case 1:
                    _osc2_deq += 1;
                    return (_buffer->peek());
                case 2:
                    _osc2_deq += 1;
                    return (_osc2buffer->peek());
                case 3:
                    _osc2_deq = 0;
                    return (_osc2buffer->peek());
           }
        } else { return (_buffer->peek()); }
    }
    return 0;
}
```

Figure 4.11: Method for enabling stereo playback via 2-pass alternation scheme

4.2.3 What I already knew

What I knew coming into this was relatively little. I did have somewhat of a fascination for and background in digital synthesizers, but was mostly contained fooling about in digital audio workstations. I had some basic understanding of trigonometric functions, and understanding of wave forms - as well as some intuition about values from these functions could translated into sound.

From a programming perspective, the relevant pieces I had were class design, graphics implementation (from earlier projects), and some mathematical coding. I understood the functionality, and in this context, intended usage of the circular buffer data structure.

Additionally, I had some previous education in music theory, when fresh out of high school, a family friend paid me one-hundred dollars to take his college music theory course for him. It was fairly interesting, and not much of it stuck, but, for the most part, the theory element of this assignment served as a refresher and not a mystery.

4.2.4 What I learned

I will say I learned quite a bit with this project. Some obvious examples include FM Synthesis, and broadly digital sound synthesis, useful contexts for circular buffers, and the mechanics behind digital audio playback. I have spent a good deal of time creating electronic music, and adjusting knobs on my analog synthesizer, but had little mechanical understanding behind the electronic production and modulation of sound.

This was also the first time making a real interactive GUI in C++. And although quite crude, developing technique for relating knob positions to internal values was somewhat of a trying experience.

4.2.5 Challenges

Initial challenges arose when trying to create any sound at all. I believe I started off by haphazardly loading random values into the buffer, to create a tone. After understanding the loading process, I implemented the StringSound class given the provided specs. Some of the major challenges came with

- Determining correct step size for moving through sine function in SynthSound class
- Implementing modulating frequency
- Get knob values to correlate to FX values and determine proper variance ranges
- Generating synth notes that could sustain over one another*

```
1     if (sInstr.getValue()) {
2     auto sf_pair =
3     std::pair<sf::Sound*, sf::SoundBuffer*> (new sf::Sound, new sf::
    SoundBuffer);
4     SynthSound note(
5         frequency,
6         fx_knobs[MOD_FREQ]->getValue(),
7         fx_knobs[OSC_I_GAIN]->getValue(),
8         fx_knobs[DECAY]->getValue(),
9         fx_knobs[MOD_GAIN]->getValue(),
10        fx_knobs[OSC_II_PITCH]->getValue(),
11        fx_knobs[OSC_II_GAIN]->getValue());
12 ...
13    if (poly_kill_queue.isFull()) {
14        auto pair = poly_kill_queue.dequeue();
15        delete pair.first;
16        delete pair.second; }
```

Figure 4.12: *Queue of synth notes to cap memory usage while allowing live modification

4.2.6 Codebase

Makefile (Circular Buffer)

```
FLAGS= -Wall -Werror -pedantic --std=c++14

all: test lint

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

test: test.o
    g++ ${FLAGS} test.o -o test -lboost_unit_test_framework

test.o: test.cpp CircularBuffer.hpp
    g++ ${FLAGS} -c test.cpp


clean:
    rm *.o
    -rm test
```

CircularBuffer.hpp

```
// Copyright [2022] Andrew Allman




#ifndef CIRCULARBUFFER_HPP_
#define CIRCULARBUFFER_HPP_

#include<stdlib.h>
#include <memory>
#include<stdexcept>

template<class T>

class CircularBuffer {
 public:
    explicit CircularBuffer(size_t capacity);
    size_t size(size_t(*evalSize)(size_t, size_t, size_t, bool)) const;
    bool isEmpty() const;  // Is the buffer is empty?
    bool isFull() const;  // Is the buffer full?
    void enqueue(T item);  // Add item to the end
    T dequeue();  // Delete and return item from the front
    T peek() const;  // Return (but do not delete) item from the front
 private:
    size_t _capacity;
    size_t _head;
    size_t _tail;
    std::unique_ptr<T[]> _data;
    bool _full;
};

// Lambda function for returning size
auto evalSize = [](size_t head, size_t tail, size_t capacity, bool is_full)
{return is_full ? capacity : (head - tail) % capacity;};

/* Initialize class with smart pointer to handle memory leaks,
   initialize data to capacity size, and head/tail to zero,
   with full flag to false.*/
```

```
39
40 template<class T>
41 CircularBuffer<T>::CircularBuffer(size_t capacity) :
42 _capacity(capacity), _head(0), _tail(0),
43 _data(std::unique_ptr<T[]>(new T[capacity])), _full(false) {
44     if (capacity < 1) {
45         throw std::invalid_argument("Capacity Cannot be Less than 1.");
46     }
47 }
48
49
50 /* Utilizes lambda function for calculating size.
51    I didn't see a good reason to use a lambda for this assignment,
52    as I'm not entirely sure what the utilization of the circular buffer
53    will entail in the next assignment */
54
55 template<class T>
56 size_t CircularBuffer<T>::size(size_t(*evalSize)(size_t, size_t, size_t,
       bool))
57 const {
58     return evalSize(_head, _tail, _capacity, isFull());
59 }
60
61 /* If head == tail, buffer is either full or empty,
62    so maintain 'is_full' status with flag*/
63 template<class T>
64 bool CircularBuffer<T>::isEmpty() const {return _head == _tail && !_full;}
65
66 // Returns the full flag value
67 template<class T>
68 bool CircularBuffer<T>::isFull() const {return _full;}
69
70 /* If is full, throws a runtime error, otherwise the tail is set to the new
       value,
71    and the tail advances to the next, presumably empty space. If tail
       advances to
72    the head's position, the buffer is full, and the full flag is set */
73
74 template<class T>
75 void CircularBuffer<T>::enqueue(T item) {
76     if (isFull()) {
77         throw std::runtime_error("Buffer Full");
78     }
79
80     _data[_tail] = item;
81     _tail = (_tail + 1) % _capacity;
82     _full = _tail == _head;
83 }
84
85 /* If is empty, throws a runtime error, otherwise a temp item is assigned
       the value of head,
86    and the head advances to the next item. Since an item was removed, the
       full flag is set
87    to false. If the next item is the tail, the queue is empty, and since the
        full flag
88    was explicitly set to false, we can reliably assess if the buffer is
       empty with the isEmpty function */
89 template<class T>
90 T CircularBuffer<T>::dequeue() {
```

```
    if (isEmpty()) {
        throw std::runtime_error("Queue Empty");
    }

    T retItem = _data[_head];
    _head = (_head + 1) % _capacity;
    _full = false;

    return retItem;
}

/*If is empty, throws a runtime error, otherwise
returns a copy of the head of the buffer, without altering items.*/
template<class T>
T CircularBuffer<T>::peek() const {
    if (isEmpty()) {
        throw std::runtime_error("Queue Empty");
    }
    return _data[_head];
}

#endif  // CIRCULARBUFFER_HPP_
```

test.cpp

```
// Copyright [2022] Andrew Allman




#include <iostream>
#include <string>

#include "./CircularBuffer.hpp"

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

/* Tests size after initialization, using function pointer to lambda
   Tests exception when attempting to peek, or dequeue an item from the
   empty buffer*/

BOOST_AUTO_TEST_CASE(testInitConditions) {
  size_t (*pEvalSize)(size_t, size_t, size_t, bool) = evalSize;
  CircularBuffer<int16_t> cBuf(10);

  // Extra credit implementation
  BOOST_REQUIRE_EQUAL(cBuf.size(pEvalSize), 0);

  BOOST_REQUIRE_THROW(cBuf.peek(), std::runtime_error);
  BOOST_REQUIRE_THROW(cBuf.dequeue(), std::runtime_error);
}

/* Tests invalid capacity value in constructor
   Test valid capacity value in constructor
   Tests size after initialization
   Tests functionality of isEmpty/isFull
   Tests size after adding elements
   Tests exception thrown trying to add item to full buffer
   Tests peek
```

```
   Tests dequeue return values while performing more queue operations
   Rechecks dequeue values after operations
   Tests no exception thrown adding item to buffer after operations
   Tests exceptions thrown peek or dequeue from empty buffer */

BOOST_AUTO_TEST_CASE(testFunctionality) {
  size_t (*pEvalSize)(size_t, size_t, size_t, bool) = evalSize;
  BOOST_REQUIRE_THROW(CircularBuffer<int16_t> cBuf(0), std::invalid_argument
   );
  BOOST_REQUIRE_NO_THROW(CircularBuffer<int16_t> cBuf(4));

  CircularBuffer<int16_t> cBuf(4);
  BOOST_REQUIRE_EQUAL(cBuf.size(pEvalSize), 0);
  cBuf.enqueue(1);
  cBuf.enqueue(2);
  cBuf.enqueue(3);
  cBuf.enqueue(4);
  BOOST_REQUIRE_EQUAL(cBuf.isEmpty(), false);
  BOOST_REQUIRE_EQUAL(cBuf.isFull(), true);

  // Extra credit implementation
  BOOST_REQUIRE_EQUAL(cBuf.size(pEvalSize), 4);

  BOOST_REQUIRE_THROW(cBuf.enqueue(5), std::runtime_error);
  BOOST_REQUIRE_EQUAL(cBuf.peek(), 1);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 1);
  cBuf.enqueue(1);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 2);
  cBuf.enqueue(2);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 3);
  cBuf.enqueue(3);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 4);
  cBuf.enqueue(4);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 1);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 2);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 3);
  BOOST_REQUIRE_EQUAL(cBuf.dequeue(), 4);
  BOOST_REQUIRE_NO_THROW(cBuf.enqueue(1));
  cBuf.dequeue();
  BOOST_REQUIRE_EQUAL(cBuf.isFull(), false);
  BOOST_REQUIRE_EQUAL(cBuf.isEmpty(), true);
  BOOST_REQUIRE_THROW(cBuf.peek(), std::runtime_error);
  BOOST_REQUIRE_THROW(cBuf.dequeue(), std::runtime_error);
}
```

StringSound.hpp

```
// Copyright [2022] <Andrew Allman>

#ifndef STRINGSOUND_HPP_
#define STRINGSOUND_HPP_

#include<math.h>
#include<random>
#include<memory>
#include<exception>
#include<vector>

#include"./CircularBuffer.hpp"
#include"./Definitions.hpp"

```

```
15 using std::vector;
16 using std::unique_ptr;
17
18
19 class StringSound {
20  public:
21     explicit StringSound(double frequency);
22     explicit StringSound(vector<sf::Int16> init);
23     StringSound(const StringSound& obj) = delete;
24     void pluck();
25     void tick();
26     sf::Int16 sample() const;
27     size_t time() const;
28  private:
29     unique_ptr<CircularBuffer<sf::Int16>> _buffer;
30     size_t _tick_count;
31 };
32
33 #endif  // STRINGSOUND_HPP_
```

StringSound.cpp

```
1  // Copyright [2022] <Andrew Allman>
2
3
4
5
6  #include "StringSound.hpp"
7  #include<exception>
8  #include<vector>
9
10 using std::vector;
11 using std::unique_ptr;
12
13
14 StringSound::StringSound(double frequency) : _tick_count(0) {
15     if (ceil(SAMPLING_RATE/frequency) <= 0) {
16         throw std::invalid_argument("Invalid Frequency");
17     }
18
19     _buffer = unique_ptr<CircularBuffer<sf::Int16>>(
20         new CircularBuffer<sf::Int16>(SAMPLING_RATE/frequency));
21 }
22
23 StringSound::StringSound(vector<sf::Int16> init): _tick_count(0) {
24     if (init.size() == 0) {
25         throw std::invalid_argument("Invalid Vector Size");
26     }
27     _buffer = unique_ptr<CircularBuffer<sf::Int16>>(
28         new CircularBuffer<sf::Int16>(init.size()));
29
30     for (size_t i = 0; i < _buffer->capacity(); i++) {
31         _buffer->enqueue(init[i]);
32     }
33 }
34
35 void StringSound::pluck() {
36     std::random_device rand_dev;
37     std::default_random_engine rand_engine(rand_dev());
38     std::uniform_int_distribution<int16_t> rand_dist(-32768, 32767);
39
```

```
    if (!_buffer->isEmpty()) {
        _buffer->empty();
    }

    while (!_buffer->isFull()) {
        _buffer->enqueue(rand_dist(rand_engine));
    }
}

void StringSound::tick() {
    sf::Int16 front(_buffer->dequeue());
    _buffer->enqueue(DECAY_RATE * 0.5 * (front + _buffer->peek()));
    _tick_count++;
}

sf::Int16 StringSound::sample() const {
    return _buffer->peek();
}

size_t StringSound::time() const {
    return _tick_count;
}
```

SynthSound.hpp

```
// Copyright [2022] <Andrew Allman>


#ifndef SYNTHSOUND_HPP_
#define SYNTHSOUND_HPP_

#include<vector>
#include<memory>

#include"./CircularBuffer.hpp"
#include"./Definitions.hpp"


using std::vector;
using std::unique_ptr;


class SynthSound {
 public:
    explicit SynthSound(
    double frequency,
    double mod_frequency,
    double gain,
    double decay,
    double mod_gain,
    double osc2_pitch,
    double osc2_gain);
    explicit SynthSound(vector<sf::Int16> init);
    SynthSound(const SynthSound& obj) = delete;
    sf::Int16 sample();
    void set();
    void tick();

 private:
    sf::Int16 _buffersize;
    unique_ptr<CircularBuffer<sf::Int16>> _buffer;
```

```
    double _carrier_frequency;
    double _mod_frequency;
    bool _mod_active;
    double _gain;
    double _mod_gain;
    double _decay;
    bool _osc2_active;
    double _osc2_pitch;
    double _osc2_gain;
    int _osc2_deq;
    unique_ptr<CircularBuffer<sf::Int16>> _osc2buffer;
};

#endif  // SYNTHSOUND_HPP_
```

SynthSound.cpp

```
// Copyright [2022] <Andrew Allman>




#include<vector>
#include<random>

#include"./SynthSound.hpp"

using std::vector;
using std::unique_ptr;

const double X_MAX = 2 * M_PI / SAMPLING_RATE;

SynthSound::SynthSound(double frequency, double mod_frequency, double gain,
 double decay, double mod_gain, double osc2_pitch, double osc2_gain) :
    _buffersize(SAMPLING_RATE/frequency),
    _buffer(unique_ptr<CircularBuffer<sf::Int16>>(
        new CircularBuffer<sf::Int16>(_buffersize))),
    _carrier_frequency(frequency * 2 * M_PI),
    _mod_frequency(mod_frequency * 2 * M_PI),
    _mod_active(_mod_frequency != 0),
    _gain(gain),
    _mod_gain(mod_gain),
    _decay(decay),
    _osc2_active(sf::Int16(osc2_pitch * _buffersize) != 0),
    _osc2_gain(osc2_gain),
    _osc2_deq(0) {
    if (_osc2_active) {
         _osc2buffer = unique_ptr<CircularBuffer<sf::Int16>>(
            new CircularBuffer<sf::Int16>(_buffersize * (2.0/osc2_pitch)));
    }
}


void SynthSound::set() {
    double x_carrier = 0,
    x_mod = 0,
    step_mod = 0,
    mod_max = 0,
    mod_min = 0,
    I = 0;

```

```
45      double step_carrier =  X_MAX / ((SAMPLING_RATE/_carrier_frequency));
46
47      if (_buffer->isFull()) {
48          _buffer->empty();
49      }
50
51      if (_mod_active) {
52          step_mod = X_MAX / (SAMPLING_RATE/(_mod_frequency / 2 * M_PI));
53          for (int i = 0; i < _buffersize; i++) {
54              double e = _mod_gain * sin(_mod_frequency * x_mod);
55              mod_max = e > mod_max ? e : mod_max;
56              mod_min = e < mod_min ? e : mod_min;
57              x_mod += step_mod;
58          }
59          I = (mod_max - mod_min) / _mod_frequency;
60          x_mod = 0;
61      }
62
63
64      while (!_buffer->isFull()) {
65          _buffer->enqueue(ceil(
66              _gain * sin(_carrier_frequency * x_carrier +
67                  (I * ((sin(_mod_frequency * x_mod)))))));
68
69          x_carrier += step_carrier;
70          x_mod += step_mod;
71      }
72
73      x_mod = x_carrier = 0;
74
75      if (_osc2_active) {
76          if (_osc2buffer->isFull()) {
77              _osc2buffer->empty();
78          }
79
80          while (!_osc2buffer->isFull()) {
81              _osc2buffer->enqueue(ceil(
82                  _osc2_gain * sin(_carrier_frequency * x_carrier +
83                  (I * ((sin(_mod_frequency * x_mod)))))));
84
85              x_carrier += step_carrier;
86              x_mod += step_mod;
87          }
88      }
89  }
90
91  void SynthSound::tick() {
92      sf::Int16 front(_buffer->dequeue());
93      _buffer->enqueue((
94          DECAY_RATE * _decay * (front + _buffer->peek())));
95
96      sf::Int16 osc2_front(_osc2buffer->dequeue());
97      _osc2buffer->enqueue(
98          DECAY_RATE * _decay * (osc2_front + _osc2buffer->peek()));
99  }
100
101 sf::Int16 SynthSound::sample() {
102     if (!_buffer->isEmpty()) {
103         if (_osc2_active && !_osc2buffer->isEmpty()) {
```

```
            switch (_osc2_deq) {
                case 0:
                    _osc2_deq += 1;
                    return (_buffer->peek());
                case 1:
                    _osc2_deq += 1;
                    return (_buffer->peek());
                case 2:
                    _osc2_deq += 1;
                    return (_osc2buffer->peek());
                case 3:
                    _osc2_deq = 0;
                    return (_osc2buffer->peek());
            }
        } else { return (_buffer->peek()); }
    }
    return 0;
}
```

sprites.hpp

```
// Copyright [2022] <Andrew Allman>




#ifndef SPRITES_HPP_
#define SPRITES_HPP_

#include<math.h>
#include<SFML/Graphics.hpp>
#include <SFML/Window.hpp>


class FX_KNOB : public sf::Drawable, public sf::Transformable {
 public:
    FX_KNOB(double min, double max, sf::Vector2f pos, double init);
    void adjust(sf::Vector2f);
    void update();

    double getValue() { return _min + _value * (_max - _min); }
    bool hasMouseHover(sf::Vector2f mPos) {
      return _skin.getGlobalBounds().contains(mPos);
    }

 private:
      virtual void draw(sf::RenderTarget& target,
      sf::RenderStates states) const {
        states.transform *= getTransform();
        target.draw(_skin, states);
    }
    double _min;
    double _max;
    double _value;
    sf::Texture _texture;
    sf::Sprite _skin;
};


FX_KNOB::FX_KNOB(double min, double max,
sf::Vector2f pos, double init) :_min(min), _max(max), _value(init) {
```

```
  _texture.loadFromFile("knob.png");
  _skin.setTexture(_texture);
  _skin.setOrigin(_skin.getGlobalBounds().width/2,
    _skin.getGlobalBounds().height/2);
  _skin.setPosition(pos);
  _skin.setRotation(0);
}

void FX_KNOB::update() {
  if (_value == 0) {
      _skin.rotate(240);
  } else if (_value <= 0.5) {
      _skin.rotate(360 - _value * 120);
  } else {
    _skin.rotate(_value * 120);
  }
}

void FX_KNOB::adjust(sf::Vector2f mPos) {
  double perf_rot;
  double current_rotation = _skin.getRotation();
  double rot_angle = (atan2(mPos.y - _skin.getPosition().y, mPos.x -
  _skin.getPosition().x) * 180 / M_PI) + 90;

  rot_angle = rot_angle < 0 ? rot_angle + 360 : rot_angle;

  if (rot_angle < 240 && rot_angle > 120) {
    if (rot_angle < 180) {
      perf_rot = 120 - current_rotation;
    } else {
      perf_rot = 240 - current_rotation;
    }
  } else {
    perf_rot = (rot_angle - current_rotation);
  }

  _skin.rotate(perf_rot);

  if (_skin.getRotation() >= 240 && _skin.getRotation() < 360) {
    _value = (120 - abs(_skin.getRotation() - 360)) / 240.0;
  } else {
    _value = 0.5 + _skin.getRotation() / 240.0;
  }
}

class SWITCH : public sf::Drawable, public sf::Transformable{
 public:
    explicit SWITCH(sf::Vector2f pos);
    void flip();
    bool hasMouseHover(sf::Vector2f mPos) {
      return _skin.getGlobalBounds().contains(mPos);
    }

    bool getValue() { return _val; }

 private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
   const {
        states.transform *= getTransform();
```

```
        target.draw(_skin, states);
    }
    sf::Texture _texture;
    sf::Sprite _skin;
    bool _val;
};

SWITCH::SWITCH(sf::Vector2f pos) : _val(false) {
  _texture.loadFromFile("switch_off.png");
  _skin.setTexture(_texture);
  _skin.setOrigin(_skin.getGlobalBounds().width/2,
  _skin.getGlobalBounds().height/2);
  _skin.setPosition(pos);
  _skin.setRotation(0);
}

void SWITCH::flip() {
  switch (_val) {
  case true:
    _texture.loadFromFile("switch_off.png");
    _skin.setTexture(_texture);
    _val = false;
    break;
  case false:
    _texture.loadFromFile("switch_on.png");
    _skin.setTexture(_texture);
    _val = true;
  }
}

#endif  // SPRITES_HPP_
```

Definitions.hpp

```
// Copyright [2022] <Andrew Allman>

#ifndef DEFINITIONS_HPP_
#define DEFINITIONS_HPP_

#ifndef SAMPLING_RATE
#define SAMPLING_RATE 44100
#endif

#ifndef CONCERT_A
#define CONCERT_A 440.0
#endif

#ifndef DECAY_RATE
#define DECAY_RATE 0.996
#endif
#endif  // DEFINITIONS_HPP_
```

Chapter 5

PS4

5.1 Sokoban UI

5.1.1 Discussion

Sokoban UI had us develop the interface to a Sokoban style game. For the unititiated (much like myself before this project), a Sokoban game (derived from the Japanese term for 'warehouse keeper') is a stage based game, with the completion of a stage resulting from moving crates through a map, placing them on marked tiles. Moving all crates to marked tiles results in the completion of the stage. The maps can sometimes be difficult to navigate through, and it is often the case that moving a crate into place too soon, or pushing it into an area with walls on either side, would result in the inability to continue.

This portion of the project strictly regarded the setup portion for the game. In our implementation, we would take a text file containing the dimensions of a grid, followed by the grid, with different characters representing different textures on the map, and build the stage representative of the grid.

```
10 10
##########
#....*...#
#....O...#
#........#
#...##...#
#...##...#
#..@..O..#
#.......*#
#........#
##########
```

Figure 5.1: Example of input for building a level

5.1.2 What I accomplished

This initial project went well. Though I didn't do much in the way of preemptive organization, some very concise and practical design choices came to mind that made development of the map both very legible in code, and efficient in implementation.

First, within the overloaded `operator»()` method, a matrix class was created to house the symbols read from the input file with linear complexity, while simultaneously allocating new sprites for each character, and assigning position. Then, when it came time to update the screen, the `draw()` iterated through each vector of elements, drawing each element, for another linearly complex sequence.

Additionally, the extra credit portion was implemented, adding a game clock that would later be used to track time elapsed while completing a level. Succintly, the project made solid use of enumerated types and an efficient setup procedure that bolstered performance, legibility, and ease of development.

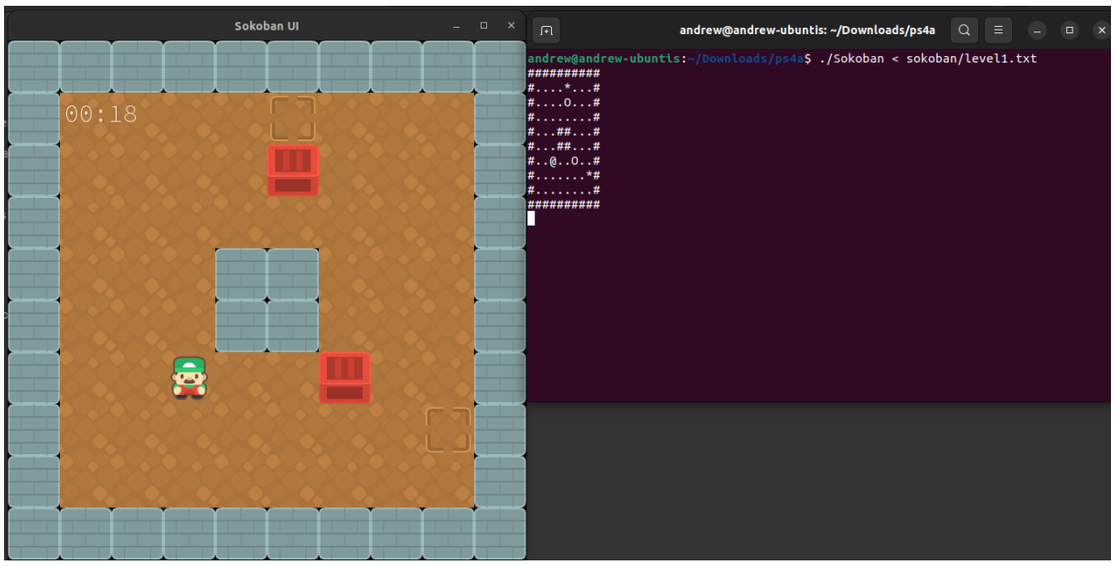


Figure 5.2: Picture of final build, showing UI implementation for input shown in Figure 5.1

5.1.3 What I already knew

Coming into this project, I had developed a useful level of experience with the SFML library that allowed me the comfort needed to be succinct in design. Knowing to represent the grid elements in a matrix class was rather helpful as well, and provided a very fitting data structure for both handling input (for later use in gameplay implementation) and outputting stage data via the overloaded `operator«()` method.

5.1.4 What I learned

In this project, I learned, or more accurately, was reminded that quality design choices don't only affect efficiency, but can rapidly expedite production, and enable smoother development as the codestack builds. This lesson will extend into the next segment of the project, where future-conscious decisions early (or those enabling gameplay over just UI implementation) would allow for continued development unimpeded by initial choices.'

Additionally, this project further familiarized me with UI implementation in the SFML library, allowing me to develop intuition about efficiently building and retaining state information about graphics in a given UI environment.

5.1.5 Challenges

This project, as I recall, went rather smoothly on the development side. Referencing my Readme for this assignment, it seems I struggled a great deal with redefinition errors in the Makefile. In looking at the Makefile, it seems like this was overcome by only defining a target for the Sokoban program, listing dependencies, and allowing make to handle the generation of the dependency objects.

5.2 Sokoban

5.2.1 Discussion

The second half of this project tasked us with creating the gameplay elements of our Sokoban game. Entering this project, I was still satisfied with how setup had gone, which kept me motivated to consider performance for the remaining development, without sacrificing the linear complexity of the UI setup process. Things to consider at this point were methods that would handle moving the crate/detecting placement into marked tiles, moving the character/preventing invalid moves, and keeping track of which crate was pushed (with efficient complexity) without modifying the `draw()` method. As you'll see, choosing the right data structure to hold crate information would be key to making this happen.

I will say in reflection, the fun of this project was the technical challenge of maintaining performance without sacrificing gameplay or design simplicity. Had I not been careful with

the initial half of this project, I might not have been so stubborn in design choices here, which I think paid off.

5.2.2 What I accomplished

The additions to this project included two new methods `movePlayer()`. `moveCrate()`, and a new member variable `unordered_map<size_t, std::pair<size_t, bool» _crates`. The map of crates was initially set with keys equal to the matrix position of the crate

`skb._crates[i ·columns + j]...`

with values set equal to the corresponding crate graphic vector position, in addition the bool value `false`, representing that the crate was not in a marked tile.

`= std::make_pair(skb._sprites[CRATE].size() - 1, false)`

This way, when a crate was pushed, the character's position in addition to the character's direction could provide a coordinate for the pushed crate, which would be used to retrieve the vector position of the specific crate that was pushed, allowing for constant time look up in the map and constant time addressing in the vector, without sacrificing linear time drawing.

After a crate had been moved, the map would copy the crate pair object into a new slot of the map with a key value of representing its update position. The object in the old position was then deleted. If the crate was pushed into position, its flag value would be set to true (disabling further movement) and its corresponding sprite would be updated to a green sprite to indicate that the crate had been set. Then member variable `_crate_count` would be decremented.

The rest of the elements were rather straightforward, with player direction setting the x and y modifier values that would be added to the player position to determine what texture of the map the player tried to advance into, with different cases handled in a relatively succinct switch statement. Players couldn't move into walls, and players could not push the crate through walls. As far I could tell, the final program was free of all bugs.

When the crate count was decremented to a value of zero, the Sokoban object's _win flag was set to true, which the main method would check with each iteration. At this point, the player would turn to face the direction opposite the last crate, victory music would play, and the screen would display a confetti-washed win sequence.

5.2.3 What I already knew

What I knew coming into this project mainly comprised of intermediate class design, and advanced-beginner graphics implementation within the SFML library. Knowledge of data structures and complexities allowed for optimized performance of the game without sacrificing simplicity or legibility in implementation. Having invested some effort into PS0 allowed me to approach this project with a useful level of awareness and intuition, as well as caution (in knowing that creativity and organization rarely mix well.) Being able to focus on this project as an expression of mechanical creativity changed the way I approached development.

5.2.4 What I learned

This project supplied me with a good deal of introductory experience with simple game design, especially allowing me to build a foundation in properly organizing graphical elements of a map. More than that, these two projects pushed granted me an opportunity to develop an efficient solution to the crate efficiency problem, which (as abstract as it may seem) will likely server me well in the future.

5.2.5 Challenges

The main challenge I ran into was keeping track of which crates were being pushed (relative to their order in a vector of sprite). Building the game, sprites were allocated and positioned, set into sprite vectors, and recorded to a matrix of symbols. However, this design initially failed to consider items that must be treated unique, specifically the crates. This issue, as

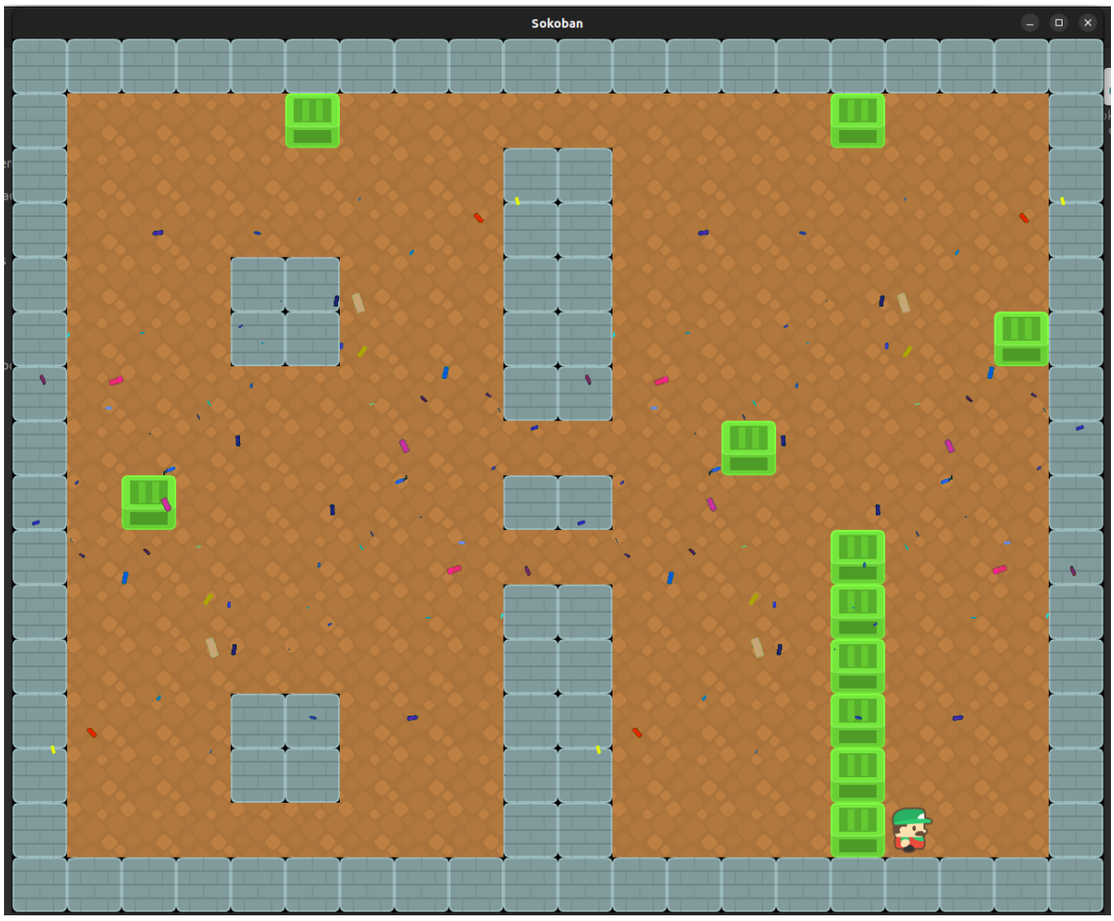


Figure 5.3: Picture of final build, showing a completed stage, complete with animated confetti and victory music

mentioned before, was solved using an unordered map that was indexed via matrix position values, returning a pair of relevant values, which made updating sprite values, and extracting gameplay-relevant information straightforward and performant.

A separate challenge I ran into, for some time, was making the key value of the crates in the map equal to

$$\texttt{i} \cdot \texttt{j} + \texttt{columns}$$

and not

$$\texttt{i} \cdot \texttt{columns} + \texttt{j}$$

which would cause some arbitrary crate to be moved, not the one that was being pushed.

Some other difficulty occured in using a `stringstream` object to read in the inputted stream data. It seems the workaround was converting the `stringstream` data into a string, and initializing a copy buffer `stringstream` with the string data. Otherwise, the rest of the project went rather smoothly.

5.2.6 Codebase

Makefile (Sokoban UI)

```
1  FLAGS= -Wall -pedantic -Werror --std=c++14
2
3  all: Sokoban
4
5  lint:
6      cpplint --filter=-runtime/references,-build/c++11 --root=. *
7
8  Sokoban: main.o Sokoban.o Matrix.o
9      g++ ${FLAGS} main.o Sokoban.o Matrix.o -o Sokoban -lsfml-system -lsfml-
       window -lsfml-graphics
10
11 clean:
```

```
    rm *.o
    -rm Sokoban
```

Makefile (gameplay)

```
FLAGS= -Wall -pedantic -Werror --std=c++14

all: Sokoban

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

Sokoban: main.o Sokoban.o Matrix.o
    g++ ${FLAGS} main.o Sokoban.o Matrix.o -o Sokoban -lsfml-system -lsfml-window -lsfml-graphics -lsfml-audio

clean:
    rm *.o
    -rm Sokoban
```

Matrix.hpp

```
// Copyright [2022] <Andrew Allman>




#ifndef MATRIX_HPP_
#define MATRIX_HPP_

#include<stdlib.h>
#include<memory>


using std::unique_ptr;

class Matrix {
 public:
    Matrix();
    Matrix(size_t height, size_t width);
    size_t size() const;
    size_t rSize() const;
    size_t cSize() const;
    char& operator()(size_t, size_t);
    const char& operator()(size_t, size_t) const;
    char& at(size_t, size_t);
    const char& at(size_t, size_t) const;

 private:
    size_t _rows;
    size_t _columns;
    unique_ptr<char[]> _state_matrix;
};

#endif  // MATRIX_HPP_
```

Matrix.cpp

```
// Copyright [2022] <Andrew Allman>




#include"./Matrix.hpp"
```

```
7
8  Matrix::Matrix(): _rows(0), _columns(0) {}
9
10 Matrix::Matrix(size_t height, size_t width):
11 _rows(height),
12 _columns(width),
13 _state_matrix(unique_ptr<char[]>(new char[height * width])) {
14     for (size_t i = 0, size = height * width; i < size; i++) {
15         _state_matrix[i] = '|';
16     }
17 }
18
19 char& Matrix::operator()(size_t row, size_t column) {
20     return _state_matrix[row * _columns + column];
21 }
22
23 const char& Matrix::operator()(size_t row, size_t column) const {
24     return _state_matrix[row * _columns + column];
25 }
26
27 char& Matrix::at(size_t row, size_t column) {
28     return _state_matrix[row * _columns + column];
29 }
30
31 const char& Matrix::at(size_t row, size_t column) const {
32     return _state_matrix[row * _columns + column];
33 }
34
35 size_t Matrix::size() const {
36     return _rows * _columns;
37 }
38
39 size_t Matrix::rSize() const {
40     return _rows;
41 }
42
43 size_t Matrix::cSize() const {
44     return _columns;
45 }
```

Sokoban.hpp (Sokoban UI)

```
1  // Copyright [2022] <Andrew Allman>
2
3  #ifndef SOKOBAN_HPP_
4  #define SOKOBAN_HPP_
5
6  #include<iostream>
7  #include<vector>
8  #include<memory>
9  #include<string>
10 #include<exception>
11 #include<utility>
12 #include<SFML/Graphics.hpp>
13 #include<SFML/Window.hpp>
14
15 #include"./Matrix.hpp"
16
17
18 using std::istream;
19 using std::vector;
```

```
20 using std::string;
21 using std::unique_ptr;
22 using std::ostream;
23
24 #define SPRITE_SIZE 64
25 #define NUM_SPRITES 8
26
27 enum DIRECTION {NORTH = 4, SOUTH, EAST, WEST};
28 enum TEXTURES { GROUND, STORAGE, CRATE, WALL, CHAR_N, CHAR_S, CHAR_E, CHAR_W
       };
29 const vector<string> TEXTURE_FP = {
30     "sokoban/ground_01.png",
31     "sokoban/ground_04.png",
32     "sokoban/crate_03.png",
33     "sokoban/block_06.png",
34     "sokoban/player_08.png",
35     "sokoban/player_05.png",
36     "sokoban/player_17.png",
37     "sokoban/player_20.png"
38 };
39
40
41 class Sokoban : public sf::Drawable, public sf::Transformable{
42  public:
43     Sokoban();
44     void movePlayer();
45     friend ostream& operator<<(ostream&, const Sokoban&);
46     friend void operator>>(istream&, Sokoban&);
47     std::pair<size_t, size_t>gridSize();
48     ~Sokoban();
49
50  private:
51     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
       const {
52         states.transform *= getTransform();
53         for (size_t i = 0; i < _sprites.size(); i++) {
54             if (i < CHAR_N) {
55                 for (sf::Sprite* sprite : _sprites[i]) {
56                     target.draw(*sprite, states);
57                 }
58             } else if (i == _play_direction) {
59                 target.draw(*_sprites[i][0], states);
60                 break;
61             }
62         }
63     }
64     vector<sf::Texture> _textures;
65     vector<vector<sf::Sprite*>> _sprites;
66     std::pair<size_t, size_t> _play_position;
67     DIRECTION _play_direction;
68     Matrix _gamestate;
69 };
70
71
72 #endif  // SOKOBAN_HPP_
```

Sokoban.cpp (Sokoban UI)

```
1 // Copyright [2022] <Andrew Allman>
2
3
```

```
4
5
6  #include "./Sokoban.hpp"
7  #include "./Matrix.hpp"
8
9  using std::string;
10 using std::vector;
11 using std::istream;
12 using std::ostream;
13
14
15 Sokoban::Sokoban() : _textures(vector<sf::Texture>(NUM_SPRITES)),
16  _sprites(vector<vector<sf::Sprite*>>(NUM_SPRITES)),
17  _play_direction(SOUTH),
18  _play_position(std::make_pair(0, 0)) {}
19
20 Sokoban::~Sokoban() {
21     if (_gamestate.size() != 0) {
22         for (vector<sf::Sprite*>& sVector : _sprites) {
23             for (sf::Sprite* sprite_ptr : sVector) {
24                 delete sprite_ptr;
25             }
26         }
27     }
28 }
29
30 void Sokoban::movePlayer() {}
31
32 ostream& operator<<(ostream& out, const Sokoban& skb) {
33     for (size_t i = 0; i < skb._gamestate.rSize(); i++) {
34         for (size_t j = 0; j < skb._gamestate.cSize(); j++) {
35             out << skb._gamestate(i, j);
36         }
37         out << '\n';
38     }
39     return out;
40 }
41
42 void operator>>(istream& initstream, Sokoban& skb) {
43     size_t rows, columns;
44     string buffer;
45     bool char_set = false;
46     if (!(initstream >> rows) || !(initstream >> columns)) {
47         throw std::runtime_error("Invalid Dimension Data reading from Stream
    .");
48     }
49
50     skb._gamestate = Matrix(rows, columns);
51
52     for (size_t texture_itr = 0;
53         texture_itr < TEXTURE_FP.size();
54         texture_itr++) {
55             skb._textures[texture_itr].loadFromFile(TEXTURE_FP[texture_itr])
    ;
56     }
57
58     for (size_t sprite_card_itr = static_cast<size_t>(CHAR_N);
59         sprite_card_itr <= static_cast<size_t>(CHAR_W);
60         sprite_card_itr++) {
```

```
61              skb._sprites[sprite_card_itr].push_back(new sf::Sprite);
62              skb._sprites[sprite_card_itr][0]->setTexture(
63                  skb._textures[sprite_card_itr]);
64      }
65
66      for (size_t i = 0; i < rows; i++) {
67          if (!(initstream >> buffer)) {
68              throw std::runtime_error(
69                  "Stream does not match provided dimensions.");
70          }
71          for (size_t j = 0; j < columns; j++) {
72              switch (buffer[j]) {
73              case '.':
74                  skb._sprites[GROUND].push_back(new sf::Sprite);
75                  skb._sprites[GROUND].back()->setTexture(
76                      skb._textures[GROUND]);
77                  skb._sprites[GROUND].back()->setPosition(
78                      SPRITE_SIZE * j, SPRITE_SIZE * i);
79                  skb._gamestate(i, j) = buffer[j];
80                  break;
81              case '#':
82                  skb._sprites[WALL].push_back(new sf::Sprite);
83                  skb._sprites[WALL].back()->setTexture(
84                      skb._textures[WALL]);
85                  skb._sprites[WALL].back()->setPosition(
86                      SPRITE_SIZE * j, SPRITE_SIZE * i);
87                  skb._gamestate(i, j) = buffer[j];
88                  break;
89              case 'O':
90                  skb._sprites[GROUND].push_back(new sf::Sprite);
91                  skb._sprites[GROUND].back()->setTexture(
92                      skb._textures[GROUND]);
93                  skb._sprites[GROUND].back()->setPosition(
94                      SPRITE_SIZE * j, SPRITE_SIZE * i);
95                  skb._sprites[CRATE].push_back(new sf::Sprite);
96                  skb._sprites[CRATE].back()->setTexture(
97                      skb._textures[CRATE]);
98                  skb._sprites[CRATE].back()->setPosition(
99                      SPRITE_SIZE * j, SPRITE_SIZE * i);
100                 skb._gamestate(i, j) = buffer[j];
101                 break;
102             case '*':
103                 skb._sprites[STORAGE].push_back(new sf::Sprite);
104                 skb._sprites[STORAGE].back()->setTexture(
105                     skb._textures[STORAGE]);
106                 skb._sprites[STORAGE].back()->setPosition(
107                     SPRITE_SIZE * j, SPRITE_SIZE * i);
108                 skb._gamestate(i, j) = buffer[j];
109                 break;
110             case '@':
111                 if (char_set) {
112                     throw std::runtime_error(
113                         "Attempted to initialize more than one character.");
114                 }
115                 skb._sprites[GROUND].push_back(new sf::Sprite);
116                 skb._sprites[GROUND].back()->setTexture(
117                     skb._textures[GROUND]);
118                 skb._sprites[GROUND].back()->setPosition(
119                     SPRITE_SIZE * j, SPRITE_SIZE * i);
```

```
                skb._sprites[skb._play_direction][0]->setPosition(
                    SPRITE_SIZE * j, SPRITE_SIZE * i);
                skb._gamestate(i, j) = buffer[j];
                skb._play_position = std::make_pair(i, j);
                char_set = true;
                break;
            default:
                break;
            }
        }
    }
}

std::pair<size_t, size_t>Sokoban::gridSize() {
    return std::make_pair(_gamestate.cSize(), _gamestate.rSize());
}
```

Sokoban.hpp (gameplay)

```
// Copyright [2022] <Andrew Allman>

#ifndef SOKOBAN_HPP_
#define SOKOBAN_HPP_

#include<iostream>
#include<vector>
#include<memory>
#include<unordered_map>
#include<string>
#include<exception>
#include<utility>
#include<SFML/Graphics.hpp>
#include<SFML/Window.hpp>

#include"./Matrix.hpp"

using std::istream;
using std::vector;
using std::string;
using std::unique_ptr;
using std::ostream;

#define SPRITE_SIZE 64
#define NUM_SPRITES 9

enum DIRECTION {NORTH = 4, SOUTH, EAST, WEST};
enum TEXTURES { GROUND, STORAGE, CRATE, WALL, CHAR_N,
CHAR_S, CHAR_E, CHAR_W, CRATE_SET };

const vector<string> TEXTURE_FP = {
    "sokoban/ground_01.png",
    "sokoban/ground_04.png",
    "sokoban/crate_03.png",
    "sokoban/block_06.png",
    "sokoban/player_08.png",
    "sokoban/player_05.png",
    "sokoban/player_17.png",
    "sokoban/player_20.png",
    "sokoban/crate_03_set.png"
};

```

```

class Sokoban : public sf::Drawable, public sf::Transformable{
 public:
    Sokoban();
    void movePlayer(const DIRECTION);
    void moveCrate(const DIRECTION, const bool, int8_t, int8_t);
    friend ostream& operator<<(ostream&, const Sokoban&);
    friend void operator>>(istream&, Sokoban&);
    std::pair<size_t, size_t> gridSize() const;
    void reset() {
            if (_gamestate.size() != 0) {
            for (vector<sf::Sprite*>& sVector : _sprites) {
                for (sf::Sprite* sprite_ptr : sVector) {
                    delete sprite_ptr;
                }
            }
        }
    }
    const std::pair<size_t, size_t> getPlayerPosition()
    { return _play_position; }
    bool isWin() { return _win ;}

    ~Sokoban();

 private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
   const {
        states.transform *= getTransform();
        for (size_t i = 0; i < _sprites.size(); i++) {
            if (i < CHAR_N) {
                for (sf::Sprite* sprite : _sprites[i]) {
                    target.draw(*sprite, states);
                }
            } else if (i == _play_direction) {
                target.draw(*_sprites[i][0], states);
                break;
            }
        }
    }
    vector<sf::Texture> _textures;
    vector<vector<sf::Sprite*>> _sprites;
    std::pair<size_t, size_t> _play_position;
    std::unordered_map<size_t, std::pair<size_t, bool>> _crates;
    DIRECTION _play_direction;
    Matrix _gamestate;
    char _trail;
    size_t _crate_count;
    bool _win;
};




#endif  // SOKOBAN_HPP_
```

Sokoban.cpp (gameplau)

```
// Copyright [2022] <Andrew Allman>


```

```
5
6  #include "./Sokoban.hpp"
7  #include "./Matrix.hpp"
8
9  using std::string;
10 using std::vector;
11 using std::istream;
12 using std::ostream;
13
14
15 Sokoban::Sokoban() : _textures(vector<sf::Texture>(NUM_SPRITES)),
16  _sprites(vector<vector<sf::Sprite*>>(NUM_SPRITES)),
17  _play_direction(SOUTH),
18  _play_position(std::make_pair(0, 0)),
19   _trail('.'),
20   _crate_count(0),
21   _crates(0),
22   _win(false) {}
23
24 Sokoban::~Sokoban() {
25     if (_gamestate.size() != 0) {
26         for (vector<sf::Sprite*>& sVector : _sprites) {
27             for (sf::Sprite* sprite_ptr : sVector) {
28                 delete sprite_ptr;
29             }
30         }
31     }
32 }
33
34 void Sokoban::moveCrate(const DIRECTION dMove,
35 const bool crate_set, int8_t x_mod, int8_t y_mod) {
36         if (crate_set) {
37             _sprites[CRATE][_crates[
38             _play_position.first *
39             _gamestate.cSize() +
40             _play_position.second].first]->setTexture(_textures[CRATE_SET]);
41             _crate_count--;
42         }
43         _sprites[CRATE][_crates[
44             _play_position.first *
45             _gamestate.cSize() +
46             _play_position.second
47             ].first
48         ]
49         ->setPosition(
50             SPRITE_SIZE * (_play_position.second + x_mod),
51             SPRITE_SIZE * (_play_position.first + y_mod));
52         _crates[
53             (_play_position.first + y_mod) *
54               _gamestate.cSize() +
55               (_play_position.second + x_mod)
56         ] = std::make_pair(
57             _crates[_play_position.first *
58             _gamestate.cSize() +
59             _play_position.second
60             ].first, crate_set);
61
62         _crates.erase(
63             _play_position.first *
```

```
64              _gamestate.cSize() +
65              (_play_position.second));
66          _gamestate(
67              _play_position.first + y_mod,
68              _play_position.second + x_mod) = 'O';
69  }
70
71  void Sokoban::movePlayer(const DIRECTION dMove) {
72      std::pair<size_t, size_t> start_position = _play_position;
73      char start_trail = _trail;
74      bool crate_set = false;
75
76      size_t y_mod = 0, x_mod = 0;
77
78      switch (dMove) {
79      case NORTH:
80          y_mod = -1;
81          break;
82      case WEST:
83          x_mod = -1;
84          break;
85      case SOUTH:
86          y_mod = 1;
87          break;
88      case EAST:
89          x_mod = 1;
90      default:
91          break;
92      }
93
94      switch (_gamestate.at(_play_position.first + y_mod,
95              _play_position.second + x_mod)) {
96          case '.':
97                  _trail = '.';
98                  _play_direction = dMove;
99                  _sprites[dMove][0]->setPosition(
100                     SPRITE_SIZE * (_play_position.second + x_mod),
101                     SPRITE_SIZE * (_play_position.first + y_mod));
102                 _play_position = std::make_pair(
103                     _play_position.first + y_mod,
104                     _play_position.second + x_mod);
105                 break;
106
107         case '*':
108                 _trail = '*';
109                 _play_direction = dMove;
110                 _sprites[dMove][0]->setPosition(
111                     SPRITE_SIZE * (_play_position.second + x_mod),
112                     SPRITE_SIZE * (_play_position.first + y_mod));
113                 _play_position = std::make_pair(
114                     _play_position.first + y_mod,
115                     _play_position.second + x_mod);
116                 break;
117
118         case 'O':
119                 switch (_gamestate.at(_play_position.first + (2 * y_mod),
120                 _play_position.second + (2 * x_mod))) {
121                 case '*':
122                     crate_set = true;
```

```
123                 case '.':
124                     if (_crates[(_play_position.first + y_mod) *
125                         _gamestate.cSize() +
126                     _play_position.second + x_mod].second) {
127                         break; }
128                 _trail = '.';
129                 _play_direction = dMove;
130                 _sprites[dMove][0]->setPosition(
131                     SPRITE_SIZE * (_play_position.second + x_mod),
132                     SPRITE_SIZE * (_play_position.first + y_mod));
133                 _play_position = std::make_pair(
134                     _play_position.first + y_mod,
135                     _play_position.second + x_mod);
136                 moveCrate(dMove, crate_set, x_mod, y_mod);
137                 break;
138                 default:
139                     break;
140             }
141         default:
142             break;
143     }
144 
145     if (start_position != _play_position) {
146         _gamestate(_play_position.first, _play_position.second) = '@';
147         _gamestate(start_position.first, start_position.second) =
    start_trail;
148     }
149 
150     if (_crate_count == 0) {
151         _play_direction =
152             dMove == NORTH ? SOUTH :
153             dMove == SOUTH ? NORTH :
154             dMove == WEST ? EAST :
155             WEST;
156         _sprites[_play_direction][0]
157         ->setPosition(
158             SPRITE_SIZE *
159             _play_position.second,
160             SPRITE_SIZE * _play_position.first);
161        _win = true;
162     }
163 }
164 
165 ostream& operator<<(ostream& out, const Sokoban& skb) {
166     out << std::to_string(skb.gridSize().second) << " " << skb.gridSize().
    first;
167     for (size_t i = 0; i <= skb._gamestate.rSize(); i++) {
168         out << "\n";
169         for (size_t j = 0; j < skb._gamestate.cSize(); j++) {
170             out << skb._gamestate(i, j);
171         }
172     }
173     return out;
174 }
175 
176 void operator>>(istream& initstream, Sokoban& skb) {
177     size_t rows, columns;
178     string buffer;
179     bool char_set = false;
```

```
180
181     if (!(initstream >> rows) || !(initstream >> columns)) {
182         throw std::runtime_error("Invalid Dimension Data reading from Stream
    .");
183     }
184
185     skb._gamestate = Matrix(rows, columns);
186
187     for (size_t texture_itr = 0;
188         texture_itr < TEXTURE_FP.size();
189         texture_itr++) {
190             skb._textures[texture_itr].loadFromFile(TEXTURE_FP[texture_itr])
    ;
191     }
192
193     for (size_t sprite_card_itr = static_cast<size_t>(CHAR_N);
194         sprite_card_itr <= static_cast<size_t>(CHAR_W);
195         sprite_card_itr++) {
196             skb._sprites[sprite_card_itr].push_back(new sf::Sprite);
197             skb._sprites[sprite_card_itr][0]->setTexture(
198                 skb._textures[sprite_card_itr]);
199     }
200
201     for (size_t i = 0; i < rows; i++) {
202         if (!(initstream >> buffer)) {
203             throw std::runtime_error(
204                 "Stream does not match provided dimensions.");
205         }
206         for (size_t j = 0; j < columns; j++) {
207             switch (buffer[j]) {
208             case '.':
209                 skb._sprites[GROUND].push_back(new sf::Sprite);
210                 skb._sprites[GROUND].back()->setTexture(
211                     skb._textures[GROUND]);
212                 skb._sprites[GROUND].back()->setPosition(
213                     SPRITE_SIZE * j, SPRITE_SIZE * i);
214                 skb._gamestate(i, j) = buffer[j];
215                 break;
216             case '#':
217                 skb._sprites[WALL].push_back(new sf::Sprite);
218                 skb._sprites[WALL].back()->setTexture(
219                     skb._textures[WALL]);
220                 skb._sprites[WALL].back()->setPosition(
221                     SPRITE_SIZE * j, SPRITE_SIZE * i);
222                 skb._gamestate(i, j) = buffer[j];
223                 break;
224             case 'O':
225                 skb._sprites[GROUND].push_back(new sf::Sprite);
226                 skb._sprites[GROUND].back()->setTexture(
227                     skb._textures[GROUND]);
228                 skb._sprites[GROUND].back()->setPosition(
229                     SPRITE_SIZE * j, SPRITE_SIZE * i);
230                 skb._sprites[CRATE].push_back(new sf::Sprite);
231                 skb._sprites[CRATE].back()->setTexture(
232                     skb._textures[CRATE]);
233                 skb._sprites[CRATE].back()->setPosition(
234                     SPRITE_SIZE * j, SPRITE_SIZE * i);
235                 skb._gamestate(i, j) = buffer[j];
236                 skb._crates[i * columns + j] =
```

```
                std::make_pair(skb._sprites[CRATE].size() - 1, false);
                skb._crate_count++;
                break;
            case '*':
                skb._sprites[STORAGE].push_back(new sf::Sprite);
                skb._sprites[STORAGE].back()->setTexture(
                    skb._textures[STORAGE]);
                skb._sprites[STORAGE].back()->setPosition(
                    SPRITE_SIZE * j, SPRITE_SIZE * i);
                skb._gamestate(i, j) = buffer[j];
                break;
            case '@':
                if (char_set) {
                    throw std::runtime_error(
                        "Attempted to initialize more than one character.");
                }
                skb._sprites[GROUND].push_back(new sf::Sprite);
                skb._sprites[GROUND].back()->setTexture(
                    skb._textures[GROUND]);
                skb._sprites[GROUND].back()->setPosition(
                    SPRITE_SIZE * j, SPRITE_SIZE * i);
                skb._sprites[skb._play_direction][0]->setPosition(
                    SPRITE_SIZE * j, SPRITE_SIZE * i);
                skb._gamestate(i, j) = buffer[j];
                skb._play_position = std::make_pair(i, j);
                char_set = true;
                break;
            default:
                break;
            }
        }
    }
}

std::pair<size_t, size_t>Sokoban::gridSize() const {
    return std::make_pair(_gamestate.cSize(), _gamestate.rSize());
}
```

main.hpp (gameplay)

```
// Copyright [2022] <Andrew Allman>
```

```
#ifndef MAIN_HPP_
#define MAIN_HPP_

#include<math.h>
#include<iostream>
#include<sstream>
#include<memory>

#include "./Sokoban.hpp"
#include "./Matrix.hpp"
#include "SFML/Audio.hpp"

using std::cin;
using std::cout;

const size_t WOW_WIDTH = 160,
```

```
WOW_HEIGHT = 50,
DELAY = 18,
CONFETTI_COUNT = 1615;

enum RETURN_STATUS { RESET, WIN };

RETURN_STATUS runGame(sf::RenderWindow& window,
Sokoban* sGame,
sf::Clock& clock,
sf::Font& font,
sf::Text& text);

RETURN_STATUS winSequence(sf::RenderWindow& window,
Sokoban* sGame);

#endif  // MAIN_HPP_
```

main.cpp (gameplay)

```
// Copyright [2022] <Andrew Allman>



#include"./main.hpp"
#include<algorithm>

auto convertGameClock = [](const sf::Clock& clock) {
    int total_seconds = clock.getElapsedTime().asSeconds();
    int minutes = total_seconds / 60, seconds = total_seconds % 60;

    return minutes < 10 ? seconds < 10 ?
            "0" + std::to_string(minutes) +
            ":" + "0" + std::to_string(seconds):
            "0" + std::to_string(minutes)
            + ":" + std::to_string(seconds):
        seconds < 10 ?
            std::to_string(minutes) +
            ":" + "0" + std::to_string(seconds):
            std::to_string(minutes)
            + ":" + std::to_string(seconds);
};


int main() {
    Sokoban* sGame = new Sokoban;
    std::string game_copy;
    std::stringstream copyBuffer;
    std::stringstream cBuf1;
    cin >> *sGame;
    copyBuffer << *sGame;

    std::string game_copy_str = copyBuffer.str();
    game_copy.resize(copyBuffer.str().size());
    std::copy(game_copy_str.begin(), game_copy_str.end(), game_copy.begin())
   ;

    sf::Font font;
    sf::Clock clock;
    sf::Text text;

    text.setFont(font);
```

```
42      text.setPosition(70, 70);
43      text.setFillColor(sf::Color::White);
44      font.loadFromFile("SourceCodePro-ExtraLight.ttf");
45
46      std::pair<size_t, size_t> gameGrid = sGame->gridSize();
47
48      sf::RenderWindow window(
49          sf::VideoMode(
50              gameGrid.first * SPRITE_SIZE, gameGrid.second * SPRITE_SIZE),
51                  "Sokoban");
52      while (!runGame(window, sGame, clock, font, text)) {
53          delete sGame;
54          sGame = new Sokoban;
55          copyBuffer.clear();
56          copyBuffer = std::stringstream(game_copy);
57          copyBuffer >> *sGame;
58      }
59      delete sGame;
60  }
61
62  RETURN_STATUS runGame(sf::RenderWindow& window, Sokoban* sGame,
63  sf::Clock& clock, sf::Font& font, sf::Text& text) {
64      clock.restart();
65      window.setKeyRepeatEnabled(false);
66      while (window.isOpen()) {
67          sf::Event event;
68          while (window.pollEvent(event)) {
69              switch (event.type) {
70                  case sf::Event::Closed:
71                      window.close();
72                      break;
73
74                  case sf::Event::KeyPressed:
75                      switch (event.key.code) {
76                      case sf::Keyboard::W:
77                          sGame->movePlayer(NORTH);
78                          break;
79                      case sf::Keyboard::A:
80                          sGame->movePlayer(WEST);
81                          break;
82                      case sf::Keyboard::S:
83                          sGame->movePlayer(SOUTH);
84                          break;
85                      case sf::Keyboard::D:
86                          sGame->movePlayer(EAST);
87                          break;
88                      case sf::Keyboard::R:
89                          return RESET;
90                          break;
91                      default:
92                          break;
93                      }
94                  default:
95                      break;
96              }
97          }
98
99          text.setString(convertGameClock(clock));
100         window.clear();
```

```
        window.draw(*sGame);
        window.draw(text);
        window.display();

        if (sGame->isWin()) {
            return winSequence(window, sGame);
        }
    }
    return WIN;
}


RETURN_STATUS winSequence(sf::RenderWindow& window, Sokoban* sGame) {
    size_t i = 0;
    sf::SoundBuffer buffer;
    sf::Sound sound;
    sf::Texture confetti_texture;
    sf::Sprite confetti_sprite;
    sf::Sprite confetti_sprite_2;
    sf::Sprite confetti_sprite_3;
    sf::Sprite confetti_sprite_4;
    sf::Time t = (sf::milliseconds(DELAY));
    std::string confetti_fp;

    confetti_sprite.setPosition(0, window.getSize().y / 2);
    confetti_sprite_2.setRotation(180);
    confetti_sprite_2.setPosition(window.getSize().x, window.getSize().y /
   2);
    confetti_sprite_3.setRotation(180);
    confetti_sprite_3.setPosition(window.getSize().x/2, window.getSize().y
   /2);
    confetti_sprite_4.setPosition(window.getSize().x/2, window.getSize().y
   /2);

    buffer.loadFromFile("RoyaltyFreeVictoryTheme.wav");
    sound.setBuffer(buffer);
    sound.setLoop(true);
    sound.play();

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            switch (event.type) {
                case sf::Event::Closed:
                    window.close();
                    break;
                case sf::Event::KeyPressed:
                    if (event.key.code == sf::Keyboard::R)
                        return RESET;
            }
        }

        i = i % CONFETTI_COUNT;
        confetti_fp = "./confetti/" + std::to_string(i) + ".png";
        confetti_texture.loadFromFile(confetti_fp);
        confetti_sprite.setTexture(confetti_texture);
        confetti_sprite_2.setTexture(confetti_texture);
        confetti_sprite_3.setTexture(confetti_texture);
        confetti_sprite_4.setTexture(confetti_texture);
```

```
157         window.clear();
158         window.draw(*sGame);
159         window.draw(confetti_sprite);
160         window.draw(confetti_sprite_2);
161         window.draw(confetti_sprite_3);
162         window.draw(confetti_sprite_4);
163         window.display();
164         i++;
165         sf::sleep(t);
166     }
167     return WIN;
168 }
```

Chapter 6

PS5

6.1 DNA Sequence Alignment

6.1.1 Discussion

The DNA sequence alignment project tasked us with creating a dynamic programming algorithm to efficiently determine the optimal sequence alignment of two strands of gene sequences. The concept of dynamic programming was one that I studied earlier in the semester so this project went perhaps the smoothest of the lot. The implementation required the use of a matrix (sized MxN) corresponding to the lengths of input sequences. Then, using a set of cost values, and a set of comparison rules, we would populate a matrix with optimal costs, and return the first position of the matrix as the globally optimal alignment cost.

In this case, the cost of a gap insert was equal to two, the cost of an alignment (two mismatched character) was one, and the cost of an alignment (with matched characters) was zero. Using this logic, we could generate positions in the table, using the minimum of some previously generated values and the operation cost. Dynamic programming was a new concept for me this semester, and already I'm finding tremendous utility in terms of creating efficient solution to problems that duplicate work.

x\y		0	1	2	3	4	5	6	7	8
		T	A	A	G	G	T	C	A	–
0	A	7	8	10	12	13	15	16	18	20
1	A	6	6	8	10	11	13	14	16	18
2	C	6	5	6	8	9	11	12	14	16
3	A	7	5	4	6	7	9	11	12	14
4	G	9	7	5	4	5	7	9	10	12
5	T	8	8	6	4	4	5	7	8	10
6	T	9	8	7	5	3	3	5	6	8
7	A	11	9	7	6	4	2	3	4	6
8	C	13	11	9	7	5	3	1	3	4
9	C	14	12	10	8	6	4	2	1	2
10	–	16	14	12	10	8	6	4	2	0

Figure 6.1: Example of DP table showing generated, relative costs throughout the table, following the optimal choice, usually in proximity to the diagonal, produces a trail of choices for aligning the sequences

6.1.2 What I accomplished

This project enabled me to put some of my theoretical knowledge of dynamic programming into practice. Given a variety of sequences used for comparison, I was able to generate the correct alignments with the intended measure of efficiency. The project included a

benchmarking section, which allowed us to compare performance of our algorithm (on our machine) against sample standards performed on some other machine. I was actually rather confident going in, but my numbers actually came in rather high. I experimented a bit with a version that used a matrix class that didn't zero out all the indices - which seemed to have shed some time off my implementation. I will likely continue to experiment with my implementation, as there could be additional latency by iteratively dereferencing the class object's dynamically allocated matrix.

```
1  int EDistance::optDistance() {
2      int source_length = _source.length(), target_length = _target.length();
3      for (int i = source_length; i >= 0; i--) {
4          for (int j = target_length; j >= 0; j--) {
5              if (i >= source_length && j >= target_length) {
6                  (*_dTable)(i, j) = 0;
7              } else if (j >= target_length) {
8                  (*_dTable)(i, j) = (*_dTable)(i + 1, j) + 2;
9              } else if (i >= source_length) {
10                 (*_dTable)(i, j) = (*_dTable)(i, j + 1) + 2;
11             } else {
12                 (*_dTable)(i, j) = min(
13                     (*_dTable)(i + 1, j + 1) + penalty(_source[i], _target[j
       ]),
14                     (*_dTable)(i + 1, j) + 2,
15                     (*_dTable)(i, j+ 1) + 2);
16             }
17         }
18     }
19     return (*_dTable)(0, 0);
20 }
```

Figure 6.2: `optDistance()` implementation for `EDistance` class

At any rate, I was able to successfully implement that algorithm with the intended complexity, and quickly retrace the steps through the table to generate a sequence alignment. Also, per the assignment instructions, I was also able to analyze memory usage, develop equations for expressing memory usage given sequence lengths, and generate theoretical estimated for work performed in a given timespan.

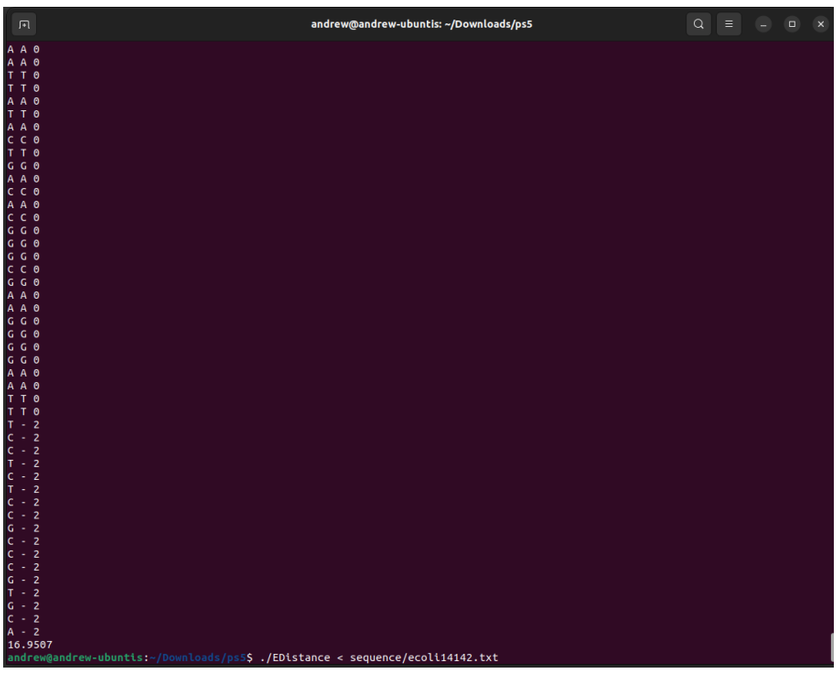


Figure 6.3: Program output for generating optimal alignment for E. Coli sequences of length 14,142

6.1.3 What I already knew

What was most relevant coming into this project was having some understanding of dynamic programming. The concept, very simply, utilizing answers from previously solved problems to solve new ones more efficiently. Examples given in class, the likes of maximal sub array and others, expanded my understanding of potential use cases, replacing traditional brute force solutions with dynamic algorithms often reducing exponentially complex problems to polynomial complexity.

6.1.4 What I learned

In this project, I learned how to fully implement this dynamic solution, in addition to generating a literal (sequence) solution via DP table retracing. Additionally, this projected enabled me to gain an in-depth look into memory and time analysis features of valgrind, which will no doubt serve me well in analyzing memory and time performance in future development.

6.1.5 Challenges

The largest challenge I had with developing this program was generating the alignment sequence, and retracing the path through the table. Although overcome somewhat quickly, it took some consideration of the logic implemented in the `optDistance()` method to fully uncover the necessary conditional statements.

6.1.6 Codebase

Makefile

```
FLAGS=  -g -Wall -Werror -pedantic --std=c++14

all: EDistance

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

EDistance: main.o EDistance.o Matrix.o
    g++ ${FLAGS} main.o EDistance.o Matrix.o -o EDistance -lsfml-system


clean:
    rm *.o
    -rm EDistance
```

Matrix.hpp

```
// Copyright [2022] <Andrew Allman>



#ifndef MATRIX_HPP_
#define MATRIX_HPP_

#include<stdlib.h>
#include<memory>
#include <iostream>
#include<algorithm>

using std::unique_ptr;


class Matrix {
 public:
```

```
    Matrix();
    Matrix(size_t height, size_t width);
    size_t size() const;
    size_t rSize() const;
    size_t cSize() const;
    int& operator()(int, int);
    const int operator()(int, int) const;
    int& at(int, int);
    const int& at(int, int) const;
 private:
    int _rows;
    int _columns;
    unique_ptr<int[]> _state_matrix;
};

#endif  // MATRIX_HPP_
```

Matrix.cpp

```
// Copyright [2022] <Andrew Allman>




#include"./Matrix.hpp"

Matrix::Matrix(): _rows(0), _columns(0) {}

Matrix::Matrix(size_t height, size_t width):
_rows(height),
_columns(width),
_state_matrix(unique_ptr<int[]>(new int[height * width])) {
    for (int i = 0, size = height * width; i < size; i++) {
        _state_matrix[i] = '0';
    }
}

int& Matrix::operator()(int row, int column) {
    return _state_matrix[row * _columns + column];
}

const int Matrix::operator()(int row, int column) const {
    return _state_matrix[row * _columns + column];
}

int& Matrix::at(int row, int column) {
    return _state_matrix[row * _columns + column];
}

const int& Matrix::at(int row, int column) const {
    return _state_matrix[row * _columns + column];
}

size_t Matrix::size() const {
    return _rows * _columns;
}

size_t Matrix::rSize() const {
    return _rows;
}

```

```
size_t Matrix::cSize() const {
    return _columns;
}
```

EDistance.hpp

```
// Copyright [2022] <Andrew Allman>

#ifndef EDISTANCE_HPP_
#define EDISTANCE_HPP_


#include <stdlib.h>

#include<iostream>
#include<string>
#include<cstddef>
#include<algorithm>

#include "./Matrix.hpp"

using std::string;
using std::size_t;

class EDistance {
 public:
    EDistance(const string& source, const string& target);
    static int penalty(char a, char b);
    static int min(int a, int b, int c);
    int optDistance();
    const string alignment(
      string (*formatReturnString)(char, char, char)) const;
    ~EDistance();
 private:
    string _source;
    string _target;
    Matrix* _dTable;
};

#endif  // EDISTANCE_HPP_
```

EDistance.cpp

```
// Copyright [2022] <Andrew Allman>



#include "./EDistance.hpp"


EDistance::EDistance(const string& source, const string& target) :
_source(source),
_target(target),
 _dTable(new Matrix(_source.length() + 1, _target.length() + 1)) {}


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

int EDistance::min(int a, int b, int c) {
    return a < b ? a < c ? a : c : b < c ? b : c;
```

```
}

int EDistance::optDistance() {
    int source_length = _source.length(), target_length = _target.length();
    for (int i = source_length; i >= 0; i--) {
        for (int j = target_length; j >= 0; j--) {
            if (i >= source_length && j >= target_length) {
                (*_dTable)(i, j) = 0;
            } else if (j >= target_length) {
                (*_dTable)(i, j) = (*_dTable)(i + 1, j) + 2;
            } else if (i >= source_length) {
                (*_dTable)(i, j) = (*_dTable)(i, j + 1) + 2;
            } else {
                (*_dTable)(i, j) = min(
                    (*_dTable)(i + 1, j + 1) + penalty(_source[i], _target[j
   ]),
                    (*_dTable)(i + 1, j) + 2,
                    (*_dTable)(i, j+ 1) + 2);
            }
        }
    }
    return (*_dTable)(0, 0);
}


const string EDistance::alignment(
    string (*formatReturnString)(char, char, char)) const {
    int i = 0, j = 0, mininimum;
    string retString = "";
    char src, trg, pen;

    while (i < _source.length() || j < _target.length()) {
        if (_source[i] == _target[j]) {
            src = _source[i];
            trg = _target[j];
            i += 1;
            j += 1;
            pen = '0';
        } else if ((*_dTable)(i + 1, j + 1) == (*_dTable)(i, j) - 1) {
            src = _source[i];
            trg = _target[j];
            i += 1;
            j += 1;
            pen = '1';
        } else if ((*_dTable)(i + 1, j)  == (*_dTable)(i, j) - 2) {
            src = _source[i];
            trg = '-';
            i += 1;
            pen = '2';
        } else {
            src = '-';
            trg = _target[j];
            j += 1;
            pen = '2';
        }
        retString += formatReturnString(src, trg, pen);
    }
    return retString;
}
```

```
78
79 EDistance::~EDistance() { delete _dTable; }
```

Main.cpp

```
1  // Copyright [2022] <Andrew Allman>
2
3
4
5
6
7  #include<string>
8  #include<iostream>
9  #include<SFML/System.hpp>
10
11 #include "./EDistance.hpp"
12
13 int main() {
14     string source, target;
15     if (!(std::cin >> source)) {
16         throw std::runtime_error("Invalid file data.");
17     }
18     if (!(std::cin >> target)) {
19         throw std::runtime_error("Invalid file data.");
20     }
21     sf::Clock c;
22     EDistance e(source, target);
23     std::cout << "Edit distance = " << e.optDistance() << std::endl;
24     std::cout << e.alignment([](char src, char trg, char pen) {
25             string retString = "";
26             retString += src;
27             retString += " ";
28             retString += trg;
29             retString += " ";
30             retString += pen;
31             retString += "\n";
32             return retString;
33         });
34     std::cout << c.getElapsedTime().asSeconds() << std::endl;
35 }
```

Chapter 7

PS6

7.1 RandWriter

7.1.1 Discussion

PS6 had us implement a program which would ingest a corpus to analyze, and then generate random text in the style of the corpus, by utilizing Markovian probabilities. I would say this project also went over rather well, and it's a concept that I enjoyed implementing a lot. Seeing how meaningfulness developed as we extended our analysis to consider corpus strings of greater length was quite rewarding. Also, gaining insight into predictive application we take for granted, such as auto-suggest typing or querying, was also an interesting component.

On the mechanical side of things, developing a good way to track relative frequencies of letters adjacent to recorded kgrams posed another engaging design challenge, which, in the end I would like to have improved upon in some respect.

7.1.2 What I accomplished

I initially had a difficult time deciding on data structures. Before I fully understood what the implementation would be per the instructions, I started capturing frequency of all phrases of length less than or equal to our *kgram* length, assigning them to vectors of maps, then vectors of sorted vectors. When I realized the look-ahead string-building process baked in the Markovian probabilities, the rest of the implementation was smooth.

However, the class initialization was a bit heavy handed - when recording adjacent letter frequencies for recorded string, I created an ASCII table for every ingested *k-gram*, each containing 127 valid ASCII spaces. This wastes quite a lot of space, and I planned on going back for it, unfortunately ran out of time.

The data structure I implemented was an `unordered_map` with a `string` key, and a vector<pair<char, int» value. The vector is the symbol table that represents frequencies of letters adjacent to the key. I wasted a lot of time trying to define a custom comparator for a different data structure `unordered_map<string, map<char, int»` that would act map of heaps,but couldn't identify a good way to get the map to order by value. As such, I defined a custom sort that would sort the vectors by frequency. The cost is greater up-front time for ingestion, but relatively fast generation of text, using constant time lookup for the phrase and iterating through the sorted vector, subtracting off sequential frequencies.

In retrospect, it definitely could be far more performant for ingestion, but in practice, I think it made sense to focus on efficient generation of text.

7.1.3 What I already knew

Coming into the project, I knew a bit about Markov chains, their utility for generating a succession of probabilities, and how they could be implemented to power applications like speech recognition, auto-suggest, and Google pagerank (in addition to an infinite number of probability based applications). However, I had never implemented a Markov chain in an application before.

```
andrew@andrew-ubuntis:~/Documents/UML/Fall 2022/Comp IV/Assignments/PS6$ ./TextWriter 10 5000 < tomsawyer.tx
t
THE ADVENTURES OF TOM SAWYER BY MARK TWAIN (Samuel Langhorne Clemens) PREFACE Most of the true gush, for the
 towel with his sleeve, and said reluctantly, with both eyes shut and groping down the road. Presently Tom s
eized him and said: "Hallo, who's that?" "Nothing. Nothing offered for a reconciliation. He took another mar
k, and branched off in high feather, and as they sped by some outlying cottages that lay in the village. Jus
t as they got their marbles and played "knucks" and "ringtaw" and "keeps" till that was exalted into the vil
lage; they had not noticed whether Tom and Huck was making thirty or forty years ago you drove me away from
here." "No, Tom, it'll take a dare will steal sheep." The new boy stepped on a spreading cravat which was we
llnigh obliterated. Daily Muff Potter washing in the sunny walks of fancy's Eden by the romantic enough, in
his black thread and threatening the schools, and it is not well bear the light! Becky cried, and Tom was ov
er and over again, and she went." "It happened along monotonously drove the knife in Potter's knife, and exc
laimed: "Boys, I know what it is. You just wait till we get in the world; it's what makes you ask?" "Well, I
 reckon hanging'll come of it." "Oh, may I come?" said Sally Rogers. "Yes." "And me?" said Sally Rogers. "Ye
s." "And me?" said Sally Rogers. "Yes." "And me, too?" said Susy Harper. "And Joe?" "Yes." "What was it, he
reasoned. Tom's whole being applauded this idea. It was discovered. When the sermon was finished, she gazed
a moment, assumed a listening to the precious box. Tom and Joe Harper, for playing hookey this evening, * an
d [* Southwestern for "afternoon"] I'll just take and split the bean, and cut the wart so as to be hardly wo
rth the saving. Tom counted the "deaf and dumb Spaniard might bear fruit; but to sit down and worship him, i
f it's pretty stiff. Where'd you get around so since-" "Oh, go 'long to school and had to miss the fun; Mary
 remained discreet. "Huck, have you seen my Tom this morning?" "No'm." "When did you sit?" "I was hid." "Whe
re?" "Behind the other course, and sprang into its master's wife. The lady was bending down and worship him,
 if it's dark, Huck. Why, he might at least mean nobody but Tom, since only Tom had told them at first, but
as he warmed to his satisfactory, for they would have given worlds, now, to have that swarthy sun-tanned ski
n of his, and his brace of tall sons speedily dressed their spirits. They were not to be put down by the mir
th. He felt that way, Huck." "Why, I know it. How can he tell?" "What's that for? I bet you Johnny Miller's
house, seized these titles, from his face. The barrow was got ready to rest. They gradually approaching the
alley; fearing all sorts of provision against rain. Here was a long, narrow, wooded island, far from hating
Tom the less there is. Can't learn an old dog new tricks, as the widow's woodshed and the key thrown away, a
nd deafen every creature. The odd superstitions touched upon the end of the roughest thing you know. No-we'l
l wait till I catch you out! I'll just take my sheer of it along with you. I'd made sure you'd played hookey
 and been a-swimming, I am. Don't you be afraid of me. I wouldn't shet me out, would you, if you want to go
an' git dis water an' not stop at the Widow Douglas, and noticed that the quarry. They passed by the resumpt
ion of fights and other declamatory gems. Then the order followed: "Now, Becky." And he fell to gazing longi
ngly across them devils, even if they had remained at home in a dreary mood, and the story could not see to
the end of half an hour older. I'd like to, honest injun, now-is it fun, or earnest?" "Earnest, Huck-just as
 easy! If I'd a knowed this was a request which Joe had just been going to shake hands with him! By jings, I
 wish I never, never, poor abused boy!" And Mrs. Harper and Ben Rogers hove in sight, and another five, or h
ere she stays." "That's the row there? Who's banging? What do you want?" "Let me in-quick! I'll tell you." "
Were you anywhere near Horse Williams' grave and stooping, round and round about that Spaniard entered or le
ft the alley or left it; nobody resembling together over the matter with you-what is the matter is! What a b
lamed lot of fools we are! You got to talk so properly that speech was marred by the spring and the boys. Th
e Widow Douglas' drawing-room. Mr. Jones was bound to its place. Then she left. CHAPTER XVII BUT there was r
eason in what Tom said he could not break the charm of life was but a small, plain affair, with a shaky voic
e: "Tom, I am so hungry!" Tom took Huck to a lonely place, and an hour made solemn by old tradition, and he
and Becky soon discovered a grim chapter of their lives at present. CHAPTER VIII TOM dodged hither and thith
er with hate. But neither could he on Saturday holiday seemed a weary long time and get laughed at. Oh, you'
re a mind to, Tom." And then what? Go on, Tom!" "Then I thought I'd got sober. I'd no business that galled h
im. He trotted along the dismal
```

Figure 7.1: Program output for generating 5000 random words using k-gram length 10

7.1.4 What I learned

As such, with perhaps too much preconceived, I learned in this project, by extending tje number of elements captured prior to an additional element (i.e a kgram next to a character) instead of two individual elements (a character next to a character), you could simulate the extension of a Markov chain. This meant, to generate the predictive strings you needn't start with one letter, determine adjacent frequencies to that letter, then determine adjacent frequencies to that 2-gram, then 3-gram, and so on, but you could capture larger strings and their overall frequency data as well as adjacent letters, and this could simulate extended Markov chains.

7.1.5 Challenges

I admit I had some trouble understanding the concept at first. Initially, I tried to build it without the assignment instructions, experimenting with different data structures and implementations, based on the premise that I would collect frequencies on every string of length less than or equal to the input *kgram* length, then build the L-sized string starting with one random character using frequency for 1-grams to build a 2-gram, so on and so forth.

Also, initially i had some trouble with 0-order grams. In the end, it had to do with the template argument in random device for generating random numbers. Finally, I had some trouble generating text if the corpus included ASCII values outside of range(0, 127), so the the TextWriter omits any delimited words containing illegal characters.

7.1.6 Codebase

Makefile

```
1 FLAGS= -g -Wall -Werror -pedantic --std=c++14
2
```

```
all: TextWriter

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

TextWriter: TextWriter.o RandWriter.o
    g++ ${FLAGS} TextWriter.o RandWriter.o -o TextWriter

test: test.o RandWriter.o
    g++ ${FLAGS} test.o RandWriter.o -o test -lboost_unit_test_framework

clean:
    -rm Main
    -rm test
    -rm *.o
```

RandWriter.hpp

```
// Copyright 2022 Andrew Allman



#ifndef RANDWRITER_HPP_
#define RANDWRITER_HPP_

#include<string>
#include<unordered_map>
#include<vector>
#include<numeric>
#include<algorithm>
#include<iostream>
#include<utility>
#include<random>

using std::string;
using std::ostream;
using std::unordered_map;
using std::pair;
using std::cout;
using std::copy;
using std::vector;
using std::make_pair;
using std::sort;
using std::random_device;
using std::default_random_engine;
using std::uniform_int_distribution;
using std::endl;

const char ASCII_MAX = 127;
vector<pair<char, int>>* initSymbolTable();

class RandWriter {
 public:
    RandWriter(string text, int k);
    int orderK() const;
    int freq(string kgram) const;
    int freq(string kgram, char c) const;
    char kRand(string kgram);
    friend ostream& operator<<(ostream& out, RandWriter& r);
    string generate(string kgram, int L);
    ~RandWriter();
```

```
 private:
    std::random_device rand_dev;
    unordered_map<string, vector<pair<char, int>>*> SymbolTableMap;
    int _k;
};

#endif  // RANDWRITER_HPP_
```

RandWriter.cpp

```
// Copyright 2022 Andrew Allman



#include "./RandWriter.hpp"

vector<pair<char, int>>* initSymbolTable() {
    vector<pair<char, int>>* symbol_table =
    new vector<pair<char, int>>(ASCII_MAX);

    for (char i = 0; i < ASCII_MAX; ++i) {
        symbol_table->at(i) = make_pair(i, 0);
    }
    return symbol_table;
}

RandWriter::RandWriter(string text, int k) : _k(k) {
    char c;
    string tsub;
    for (size_t i = 0; i < text.length() - _k - 1; i++) {
        c = text[i + _k];
        tsub = text.substr(i, _k);
        if (SymbolTableMap.find(tsub) == SymbolTableMap.end()) {
            SymbolTableMap[tsub] = initSymbolTable();
        }
        ++SymbolTableMap[tsub]->at(c).second;
        tsub.clear();
    }
    for (auto& itm : SymbolTableMap) {
        sort((*itm.second).begin(), (*itm.second).end(), [](
            pair<char, int> a, pair<char, int> b) {
            return a.second > b.second;
        });
    }
}

int RandWriter::orderK() const {
    return _k;
}

int RandWriter::freq(string kgram) const {
    if (SymbolTableMap.find(kgram) == SymbolTableMap.end()) {
        throw std::runtime_error("No such kgram found: " + kgram);
    }

    return std::accumulate((*SymbolTableMap.at(kgram)).begin(),
    (*SymbolTableMap.at(kgram)).end(), 0, [](
        int sum, pair<char, int> a) { return sum + a.second; });
}

int RandWriter::freq(string kgram, char c) const {
```

```
52      if (kgram.length() != _k) {
53          throw std::runtime_error("Provided kgram incorrect size.");
54      }
55
56      if (SymbolTableMap.find(kgram) == SymbolTableMap.end()) {
57          throw std::runtime_error("No such kgram found: " + kgram);
58      }
59
60      for (size_t i = 0; i < (*SymbolTableMap.at(kgram)).size(); ++i) {
61          if ((*SymbolTableMap.at(kgram)).at(i).first == c) {
62              return (*SymbolTableMap.at(kgram)).at(i).second;
63          }
64      }
65      return 0;
66  }
67
68  char RandWriter::kRand(string kgram) {
69      if (kgram.length() != _k) {
70          throw("Invalid kgram length. Does not match k");
71      }
72      std::default_random_engine rand_engine(rand_dev());
73      std::uniform_int_distribution<int> rand_dist(0, freq(kgram));
74      int rnd_input = rand_dist(rand_engine);
75      for (auto& pr : (*SymbolTableMap[kgram])) {
76          rnd_input -= pr.second;
77          if (rnd_input <= 0) {
78              return pr.first;
79          }
80      }
81      return ' ';
82  }
83
84  ostream& operator<<(ostream& out, RandWriter& r) {
85      for (auto& itm : r.SymbolTableMap) {
86          out << "K-Gram: " << itm.first << endl;
87          for (auto& pr : *itm.second) {
88              out << "Char: "
89              << pr.first
90              << " | Frequency: "
91              << pr.second
92              << std::endl;
93          }
94      } return out;
95  }
96  string RandWriter::generate(string kgram, int L) {
97      if (kgram.length() != _k) {
98          throw std::runtime_error("Provided seed doesn't match k-length.");
99      }
100     while (kgram.length() != L) {
101         if (_k > 0)
102             kgram += kRand(kgram.substr(kgram.length() - _k));
103         else
104             kgram += kRand("");
105     }
106     return  kgram;
107 }
108
109 RandWriter::~RandWriter() {
110     for (auto& itm : SymbolTableMap) {
```

```
        delete itm.second;
    }
}
```

TextWriter.cpp

```
// Copyright 2022 Andrew Allman




#include<fstream>
#include<sstream>

#include "./RandWriter.hpp"

using std::ifstream;
using std::stringstream;
using std::cin;
using std::stoi;

int main(int argc, char* argv[]) {
    string inp_text;
    stringstream buffer;
    int k = stoi(argv[1]), L = stoi(argv[2]);

    while (cin >> inp_text) {
        for (char c : inp_text) {
            if (c < 0 || c > ASCII_MAX) { continue; }
        }
        buffer << string(inp_text + " ");
    }
    string seed = buffer.str().substr(0, k);
    RandWriter rw(buffer.str(), k);
    std::cout << rw.generate(seed, L) << endl;
}
```

test.cpp

```
// Copyright 2022 Andrew Allman



#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE Main

#include <iostream>
#include <string>
#include<fstream>
#include<sstream>
#include <boost/test/unit_test.hpp>

#include"./RandWriter.hpp"

using std::ifstream;
using std::stringstream;
using std::cin;


BOOST_AUTO_TEST_CASE(testInit) {
  std::ifstream fp("tomsawyer.txt");
  string inp_text;
```

```
24    stringstream buffer;
25    buffer << fp.rdbuf();
26    BOOST_REQUIRE_NO_THROW(RandWriter(buffer.str(), 10));
27    RandWriter rw(buffer.str(), 0);
28    BOOST_REQUIRE_EQUAL(rw.orderK(), 0);
29    BOOST_REQUIRE_GT(rw.freq("", ' '), rw.freq("", 'e'));
30    BOOST_REQUIRE_GT(rw.freq(""), 380000);
31    BOOST_REQUIRE_NO_THROW(rw.kRand(""));
32    BOOST_REQUIRE_NO_THROW(rw.generate("", 0));
33    BOOST_REQUIRE_THROW(rw.generate("I", 10), std::runtime_error);
34  }
```

Chapter 8

PS7

8.1 Kronos Time Clock

8.1.1 Discussion

This program tasked us with the incredibly practical goal of analyzing text logs. Empowered by knowledge of regular expression, and the application of to identify strings of relevant patterns, we could parse the entries of a Kronos InTouch time clock log, to discern information about device startups in addition to service behavior.

8.1.2 What I accomplished

The program uses two class objects, one (`BootSequence`) to represent a boot sequence, and another (`Service`) to represent a service object. The `BootSequence`s for a file are stored in a `vector`, and are built carrying a map of service objects. The `BootSequence` stores the startup information (dates, times, successful, linenumbers, etc.), and the service objects store the the service object information (elapsed time, linenumbers, etc).

Performance of this implementation makes it so the file only need be read once, but lines that match certain patterns will be searched several times while building different objects. Performance will resemble linear with `n` operations for all lines, `m` operations for server specific lines, and `k` operations for service specific lines.

Initially, getting my bearings in regex/data parsing in C++, I wanted to identify schemes comprehensive enough to identify the correct lines without being too strict as to omit target lines/phrases. After that, building the service analysis came with noticing a complete boot would be signaled before the last service had signaled completion, so the logic needed to be changed. The structure is essentially:

- Identify when a boot attempt occured
- Capture all service relevant lines inbetween in vectors.
- Transition the data to class objects once a new attempt occurs, record success or failure based on appearance of 'AbstractConnector' line

From there, there was some basic logic included in outputting to the files - with the exception of device4, which I'll speak to in challenges.

8.1.3 What I already knew

With much brevity, the things I knew coming into the project regarded effective class design, effective procedural programming (such that we avoid duplicate work where possible), and some regular expression syntax.

8.1.4 What I learned

What I learned in this project mostly involved further development of proper regular expression syntax, in addition use cases for practical inmplementation.

```
Device Boot Report

InTouch log file: device1_intouch.log
Lines Scanned: 443838
Device boot count: initiated = 6, Completed: 6

=== Device boot ===
435369(device1_intouch.log): 2014-03-25 19:11:59 Boot Start
435759(device1_intouch.log): 2014-03-25 19:15:02 Boot Completed
    Boot Time: 183000 ms

Services
    Logging
        Start: 435386(device1_intouch.log)
        Completed: 435387(device1_intouch.log)
        Elapsed Time: 346 ms

    DatabaseInitialize
        Start: 435388(device1_intouch.log)
        Completed: 435412(device1_intouch.log)
        Elapsed Time: 33139 ms

    MessagingService
        Start: 435413(device1_intouch.log)
        Completed: 435415(device1_intouch.log)
        Elapsed Time: 8663 ms
...
```

Figure 8.1: Excerpt from generated log report file

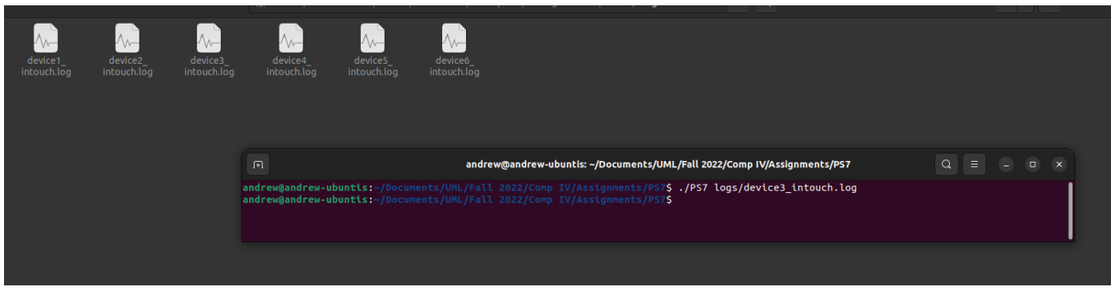


Figure 8.2: Example of programming running, and producing report for inputted log file

8.1.5 Challenges

In general, there were the mechanical difficulties of building a project. I spent about 2 hours trying to identify why dynamic allocation of BootSequence objects triggered segmentation error. As it turned out, there were old build files using different data structures being included in the build. Upon deletion things went according to plan.

Additionally, I wrote the program according to the results given with device5_intouch.log_BOOT.rpt, and unfortunately, immediately moved on to prepping my code for export, which makes it somewhat costly to change / debug. Upon testing device4, I received an segmentation error when attempting to index a specific service which wasn't contained in the service map, meaning it did not get added either in the file or by the logic. I didn't exhaustively investigate this, since successful startup logic seems to produce quality reports for all other files, so my theory was exceptions for these programs were occurring, but they did not prevent successful startup. As such, I patched this by including a condition at export that noted when expected services did not start correctly (either by identifying when no success line occurred, or when the service was not present between server boot signals), and recorded them in services not successfully started portion of the log.

As it turns out, I believe my theory was correct, and this patch wasn't a terrible solution to the issue I had encountered.

In any case, all log files were handled, and no faulty service detail information should exist in the reports.

8.1.6 Codebase

Makefile

```
FLAGS= -Wall -Werror -pedantic --std=c++14

all: PS7

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

PS7: main.o BootSequence.o Service.o
    g++ ${FLAGS} main.o BootSequence.o Service.o -o PS7 -lboost_regex

clean:
    -rm PS7
    -rm *.o
```

BootSequence.hpp

```
// Copyright [2022] <Andrew Allman>

#ifndef BOOTSEQUENCE_HPP_
#define BOOTSEQUENCE_HPP_

#include<string>
#include<map>

#include "boost/date_time/gregorian/gregorian.hpp"
#include "boost/date_time/posix_time/posix_time.hpp"
#include <boost/date_time/posix_time/posix_time_io.hpp>
#include "./Service.hpp"


using std::map;
using boost::posix_time::ptime;


class BootSequence {
 public:
    BootSequence();
    void setInitLine(const int);
    void setEndLine(const int);
    void setBootStampStart(const ptime&);
    void setBootStampComplete(const ptime&);
    void setSuccessful(const bool);
    const bool Successful() const;
    void setServiceMap(map<string, Service>*);
    const map<string, Service>* getServiceMap() const;
    const ptime& getBootStampStart() const;
    const ptime& getBootStampComplete() const;
    const int getInitLine() const;
    const int getEndLine() const;
    ~BootSequence();
 private:
    int _initline;
    int _endline;
    ptime _bootstampstart;
```

```
    ptime _bootstampcomplete;
    bool _successfulboot;
    map<string, Service>* _services;
};

#endif  // BOOTSEQUENCE_HPP_
```

Bootsequence.cpp

```
// Copyright [2022] <Andrew Allman>



#include "./BootSequence.hpp"


BootSequence::BootSequence() {}

void BootSequence::setInitLine(const int initline) { _initline = initline; }

void BootSequence::setEndLine(const int endline) { _endline = endline; }

void BootSequence::setBootStampStart(const ptime& bootstampstart) {
_bootstampstart = bootstampstart; }

void BootSequence::setBootStampComplete(const ptime& bootstampcomplete) {
_bootstampcomplete = bootstampcomplete; }

void BootSequence::setSuccessful(const bool successfulboot) {
_successfulboot = successfulboot; }

void BootSequence::setServiceMap(map<string, Service>* services) {
_services = services; }

const int BootSequence::getInitLine() const { return _initline; }

const int BootSequence::getEndLine() const { return _endline; }

const bool BootSequence::Successful() const { return _successfulboot; }

const ptime& BootSequence::getBootStampStart() const {
return _bootstampstart; }

const ptime& BootSequence::getBootStampComplete() const {
return _bootstampcomplete; }

const map<string, Service>* BootSequence::getServiceMap() const {
return _services; }

BootSequence::~BootSequence() {
if (~_services->empty()) { delete _services; } }
```

Service.hpp

```
// Copyright [2022] <Andrew Allman>

#ifndef SERVICE_HPP_
#define SERVICE_HPP_

#include<string>

using std::string;
```

```
9
10 class Service {
11  public:
12     Service();
13     Service(const string&, const int, const int, const int, const bool);
14     void setName(const string&);
15     void setInitLine(const int);
16     void setEndLine(const int);
17     void setElapsed(const int);
18     void setSuccessful(const bool);
19     const string& getName() const;
20     const int getInitLine() const;
21     const int getEndLine() const;
22     const int getElapsed() const;
23     const bool getSuccessful() const;
24  private:
25     string _servicename;
26     int _initline;
27     int _endline;
28     int _elapsed;
29     bool _successful;
30 };
31
32 #endif  // SERVICE_HPP_
```

Service.cpp

```
1  // Copyright [2022] <Andrew Allman>
2
3  #include "./Service.hpp"
4
5
6  Service::Service(): _servicename("undefined"), _initline(-1),
7  _endline(-1), _elapsed(-1), _successful(false) {}
8
9  Service::Service(const string& servicename,
10 const int initline, const int endline,
11 const int elapsed, const bool successful):
12 _initline(initline),
13 _endline(endline),
14 _elapsed(elapsed),
15 _successful(successful) {}
16
17 void Service::setName(const string& servicename) {
18 _servicename = servicename; }
19
20 void Service::setInitLine(const int initline) {
21 _initline =  initline; }
22
23 void Service::setEndLine(const int endline) {
24 _endline = endline; }
25
26 void Service::setElapsed(const int elapsed) {
27 _elapsed = elapsed; }
28
29 void Service::setSuccessful(const bool successful) {
30 _successful = successful; }
31
32 const string& Service::getName() const {
33 return _servicename; }
34
```

```
35 const int Service::getInitLine() const {
36 return _initline; }
37
38 const int Service::getEndLine() const {
39 return _endline; }
40
41 const int Service::getElapsed() const {
42 return _elapsed; }
43
44 const bool Service::getSuccessful() const {
45 return _successful; }
```

main.hpp

```
1  // Copyright [2022] <Andrew Allman>
2
3  #ifndef MAIN_HPP_
4  #define MAIN_HPP_
5
6  #include<map>
7  #include<iostream>
8  #include<fstream>
9  #include<vector>
10 #include<string>
11 #include<utility>
12 #include<algorithm>
13 #include<numeric>
14 #include<locale>
15 #include <boost/date_time/posix_time/posix_time_io.hpp>
16 #include <boost/regex.hpp>
17
18 #include "./BootSequence.hpp"
19
20 using std::vector;
21 using std::string;
22 using std::stoi;
23 using std::pair;
24 using std::cin;
25 using std::cout;
26 using std::endl;
27 using boost::posix_time::time_facet;
28 using boost::gregorian::date;
29 using boost::gregorian::from_simple_string;
30 using boost::posix_time::ptime;
31 using boost::posix_time::time_duration;
32 using boost::regex;
33 using boost::smatch;
34
35 std::map<string, Service>* parseServicesLines(
36     const vector<std::pair<string, int>> init_lines,
37     const vector<pair<string, int>> end_lines, bool successful_boot);
38
39 const int generateBootSequences(std::ifstream&, vector<BootSequence*>&);
40 void generateLogs(const int line_count,
41 const vector<BootSequence*>&, const string& fp);
42
43 const regex INIT_BOOT_RS("\\(log\\.c\\.[0-9]+\\) server started");
44 const regex COMP_BOOT_RS(
45     "AbstractConnector:Started SelectChannelConnector@0\\.0\\.0\\.0:9080");
46
47 const regex INIT_SERVICE_RS("Starting Service\\.  [a-zA-Z]+ [0-9]*\\.[0-9]+"
```

```
    );
const regex COMP_SERVICE_RS("Service started successfully.");
const regex DATE_FRM_RS("[0-9]{4}-[0-9]{2}-[0-9]{2}");
const regex TIME_FRM_RS("[0-9]{2}:[0-9]{2}:[0-9]{2}(\\.[0-9]{1,3})?");
const regex ALPHA_STRING("[a-zA-Z]+");
const regex MS_FRM_NUMBER_RS("\\([0-9]+");

const size_t SERVICE_INITLINE_SPLIT_INDEX = 19;
const size_t SERVICE_ENDLINE_SPLIT_INDEX = 31;

const time_facet* T_FACET(new time_facet("%Y-%m-%d %H:%M:%S"));

const std::vector<string> SERVICES = {
    "Logging", "DatabaseInitialize", "MessagingService",
    "HealthMonitorService", "Persistence", "ConfigurationService",
    "LandingPadService", "PortConfigurationService", "CacheService",
    "ThemingService", "StagingService", "DeviceIOService", "BellService",
    "GateService", "ReaderDataService", "BiometricService",
    "StateManager", "OfflineSmartviewService", "AVFeedbackService",
    "DatabaseThreads", "SoftLoadService", "WATCHDOG",
    "ProtocolService", "DiagnosticsService"
};

#endif  // MAIN_HPP_
```

main.cpp

```
// Copyright [2022] <Andrew Allman>



#include "./main.hpp"



int main(int argc, char* argv[]) {
    string log_filename = argv[1];
    std::ifstream logfile(log_filename);
    log_filename.substr(log_filename.find("/") + 1);
    std::vector<BootSequence*> Sequences;
    int lines = generateBootSequences(logfile, Sequences);
    generateLogs(lines, Sequences,
        log_filename.substr(log_filename.find("/") + 1));
    for (auto& boot_sequence : Sequences) {
        delete boot_sequence;
    }
}

void generateLogs(const int line_count,
const vector<BootSequence*>& Sequences,
const string& filename) {
    std::ofstream boot_log;
    vector<string> exception_cases;
    boot_log.open("reports/" + filename + ".rpt");
    boot_log.imbue(std::locale(cout.getloc(), T_FACET));

    int completed = std::accumulate(Sequences.begin(), Sequences.end(), 0,
    [](int sum, const BootSequence* itr)
    {return sum + static_cast<int>(itr->Successful());});

    boot_log << "Device Boot Report\n" << endl;
```

```
    boot_log << "InTouch log file: " + filename << endl;
    boot_log << "Lines Scanned: " << line_count << endl;

    boot_log << "Device boot count: initiated = "
    << Sequences.size()
    << ", Completed: "
    << completed
    << "\n"
    << endl;

    for (auto& boot_sequence : Sequences) {
        boot_log << "=== Device boot ===" << endl;
        if (boot_sequence->Successful()) {
            boot_log << boot_sequence->getInitLine()
            << "(" + filename + "): "
            << boot_sequence->getBootStampStart()
            << " Boot Start"
            << endl;

            boot_log << boot_sequence->getEndLine()
            << "(" + filename + "): "
            << boot_sequence->getBootStampComplete()
            << " Boot Completed"
            << endl;

            boot_log << "\tBoot Time: " << (
                boot_sequence->getBootStampComplete() -
                boot_sequence->getBootStampStart()).total_milliseconds()
            << " ms"
            << "\n"
            << endl;

            boot_log << "Services" << endl;

            auto service_map_ptr = boot_sequence->getServiceMap();
            for (auto& service : SERVICES) {
                if ((*service_map_ptr).find(service)
                != (*service_map_ptr).end()) {
                    boot_log << "\t" + service + "\n";

                    if ((*service_map_ptr).at(service).getEndLine() == -1) {
                        boot_log << "\t\tStart: "
                        << "Not Started"
                        << "(" + filename + ")"
                        << endl;
                    } else {
                        boot_log << "\t\tStart: "
                        << (*service_map_ptr).at(service).getInitLine()
                        << "(" + filename + ")"
                        << endl;
                    }

                    if ((*service_map_ptr).at(service).getEndLine() == -1) {
                    boot_log << "\t\tCompleted: "
                    << "Not Completed"
                    << "(" + filename + ")"
                    << endl
                    << "\t\tElapsed Time: \n"
                    << endl;
```

```
                    exception_cases.push_back(service);
                    } else {
                    boot_log << "\t\tCompleted: "
                        << (*service_map_ptr).at(service).getEndLine()
                        << "(" + filename + ")"
                        << endl
                        << "\t\tElapsed Time: "
                        << (*service_map_ptr).at(service).getElapsed()
                        << " ms\n"
                        << endl;
                    }
                } else {
                    exception_cases.push_back(service);
                }
            }
            if (!exception_cases.empty()) {
                boot_log << "\t*** Services not successfully started: ";
                for (auto str_itr = exception_cases.begin();
                    str_itr != exception_cases.end() - 1; ++str_itr) {
                        boot_log << *str_itr << ", ";
                }
            boot_log << exception_cases[exception_cases.size() - 1]
            << "\n"
            << endl;
            exception_cases.clear();
            }
        } else {
            boot_log << boot_sequence->getInitLine()
            << "(" + filename + "): "
            << boot_sequence->getBootStampStart()
            << " Boot Start"
            << endl
            << "**** Incomplete boot ****\n"
            << endl;

            boot_log << "Services" << endl;

            for (auto& service : SERVICES) {
                boot_log << "\t" + service + "\n"
                << "\t\tStart: Not started"
                << "(" + filename + ")"
                << endl
                << "\t\tCompleted: Not completed"
                << "(" + filename + ")"
                << endl
                << "\t\tElapsed Time:"
                << endl;
            }

            boot_log << "\n\t*** Services not successfully started: ";

            for (auto str_itr = SERVICES.begin();
            str_itr != SERVICES.end() - 1; ++str_itr) {
                boot_log << *str_itr << ", ";
            }
            boot_log << SERVICES[SERVICES.size() - 1]
            << "\n"
            << endl;
        }
```

```
153     }
154     boot_log.close();
155 }
156 
157 const int generateBootSequences(std::ifstream& logfile,
158 vector<BootSequence*>& Sequences) {
159     bool first_pass = true, boot_complete = false;
160     int line_count = 0, bt_init_line = -1, bt_end_line = -1;
161     time_duration starttime, endtime;
162     date start_date;
163     smatch rs_match;
164     std::string t_line;
165 
166     std::vector<std::pair<std::string, int>>
167     service_init_lines,
168     service_end_lines;
169 
170     while (!logfile.eof()) {
171         line_count++;
172 
173         std::getline(logfile, t_line);
174 
175         if (regex_search(t_line, INIT_BOOT_RS)) {
176             if (!first_pass) {
177                 if (boot_complete) {
178                     Sequences.push_back(new BootSequence);
179                     Sequences[Sequences.size() - 1]->
180                     setServiceMap(parseServicesLines(
181                         service_init_lines, service_end_lines, true));
182                     Sequences[Sequences.size() - 1]->
183                     setBootStampStart(ptime(start_date, starttime));
184                     Sequences[Sequences.size() - 1]->
185                     setBootStampComplete(ptime(start_date, endtime));
186                     Sequences[Sequences.size() - 1]->
187                     setInitLine(bt_init_line);
188                     Sequences[Sequences.size() - 1]->
189                     setEndLine(bt_end_line);
190                     Sequences[Sequences.size() - 1]->
191                     setSuccessful(true);
192                     service_init_lines.clear();
193                     service_end_lines.clear();
194                     boot_complete = false;
195                 } else {
196                     Sequences.push_back(new BootSequence);
197                     Sequences[Sequences.size() - 1]->
198                     setServiceMap(parseServicesLines(
199                         service_init_lines, service_end_lines, false));
200                     Sequences[Sequences.size() - 1]->
201                     setBootStampStart(ptime(start_date, starttime));
202                     Sequences[Sequences.size() - 1]->
203                     setInitLine(bt_init_line);
204                     Sequences[Sequences.size() - 1]->
205                     setSuccessful(false);
206                     service_init_lines.clear();
207                     service_end_lines.clear();
208                 }
209 
210             } else { first_pass = false; }
211 
```

```
            bt_init_line = line_count;

            if (regex_search(t_line, rs_match, DATE_FRM_RS)) {
                start_date = date(from_simple_string(rs_match.str()));
            }

            if (regex_search(t_line, rs_match, TIME_FRM_RS)) {
                starttime = time_duration(
                    std::stoi(rs_match.str().substr(0, 2)),
                    std::stoi(rs_match.str().substr(3, 2)),
                    std::stoi(rs_match.str().substr(6, 2)), 0);
            }
        }

        if (regex_search(t_line, COMP_BOOT_RS) && !first_pass) {
            if (regex_search(t_line, rs_match, TIME_FRM_RS)) {
                endtime = time_duration(
                    std::stoi(rs_match.str().substr(0, 2)),
                    std::stoi(rs_match.str().substr(3, 2)),
                    std::stoi(rs_match.str().substr(6, 2)), 0);
                boot_complete = true;
                bt_end_line = line_count;
            }
        }

        if (regex_search(t_line, INIT_SERVICE_RS) && !first_pass) {
            service_init_lines.push_back(
                std::pair<string, int>(t_line, line_count));
        }

        if (regex_search(t_line, COMP_SERVICE_RS) && !first_pass) {
            service_end_lines.push_back(
                std::pair<string, int>(t_line, line_count));
        }
    }

    if (!service_init_lines.empty() && boot_complete) {
        Sequences.push_back(new BootSequence);
        Sequences[Sequences.size() - 1]->
        setServiceMap(parseServicesLines(
            service_init_lines, service_end_lines, true));
        Sequences[Sequences.size() - 1]->
        setBootStampStart(ptime(start_date, starttime));
        Sequences[Sequences.size() - 1]->
        setBootStampComplete(ptime(start_date, endtime));
        Sequences[Sequences.size() - 1]->
        setInitLine(bt_init_line);
        Sequences[Sequences.size() - 1]->
        setEndLine(bt_end_line);
        Sequences[Sequences.size() - 1]->
        setSuccessful(true);
        service_init_lines.clear();
        service_end_lines.clear();
    } else if (!service_init_lines.empty()) {
        Sequences.push_back(new BootSequence);
        Sequences[Sequences.size() - 1]->
        setServiceMap(parseServicesLines(
            service_init_lines, service_end_lines, false));
        Sequences[Sequences.size() - 1]->
```

```
271         setBootStampStart(ptime(start_date, starttime));
272         Sequences[Sequences.size() - 1]->
273         setInitLine(bt_init_line);
274         Sequences[Sequences.size() - 1]->
275         setSuccessful(false);
276         service_init_lines.clear();
277         service_end_lines.clear();
278     }
279     return t_line == "" ? --line_count : line_count;
280 }
281
282 std::map<std::string, Service>* parseServicesLines(
283 const vector<std::pair<std::string, int>> init_lines,
284 const vector<std::pair<std::string, int>> end_lines, bool successful_boot) {
285     smatch rs_match;
286     std::map<std::string, Service>* service_map =
287         new std::map<std::string, Service>;
288
289     if (successful_boot) {
290         for (auto& line : init_lines) {
291             string temp_split = line.first.substr(
    SERVICE_INITLINE_SPLIT_INDEX);
292             if (regex_search(temp_split, rs_match, ALPHA_STRING)) {
293                 (*service_map)[rs_match.str()].setName(rs_match.str());
294                 (*service_map)[rs_match.str()].setInitLine(line.second);
295                 (*service_map)[rs_match.str()].setSuccessful(true);
296             }
297         }
298
299         for (auto& line : end_lines) {
300             string temp_split = line.first.substr(
    SERVICE_ENDLINE_SPLIT_INDEX);
301             if (regex_search(temp_split, rs_match, ALPHA_STRING)) {
302                 string tServiceName(rs_match.str());
303                 (*service_map)[tServiceName].setEndLine(line.second);
304                 if (regex_search(temp_split, rs_match, MS_FRM_NUMBER_RS)) {
305                     (*service_map)[tServiceName].setElapsed(
306                         stoi(rs_match.str().substr(1)));
307                 }
308             }
309         }
310     } else {
311         for (auto& service : SERVICES) {
312             (*service_map)[service] = Service();
313             (*service_map)[service].setSuccessful(false);
314         }
315     }
316     return service_map;
317 }
```