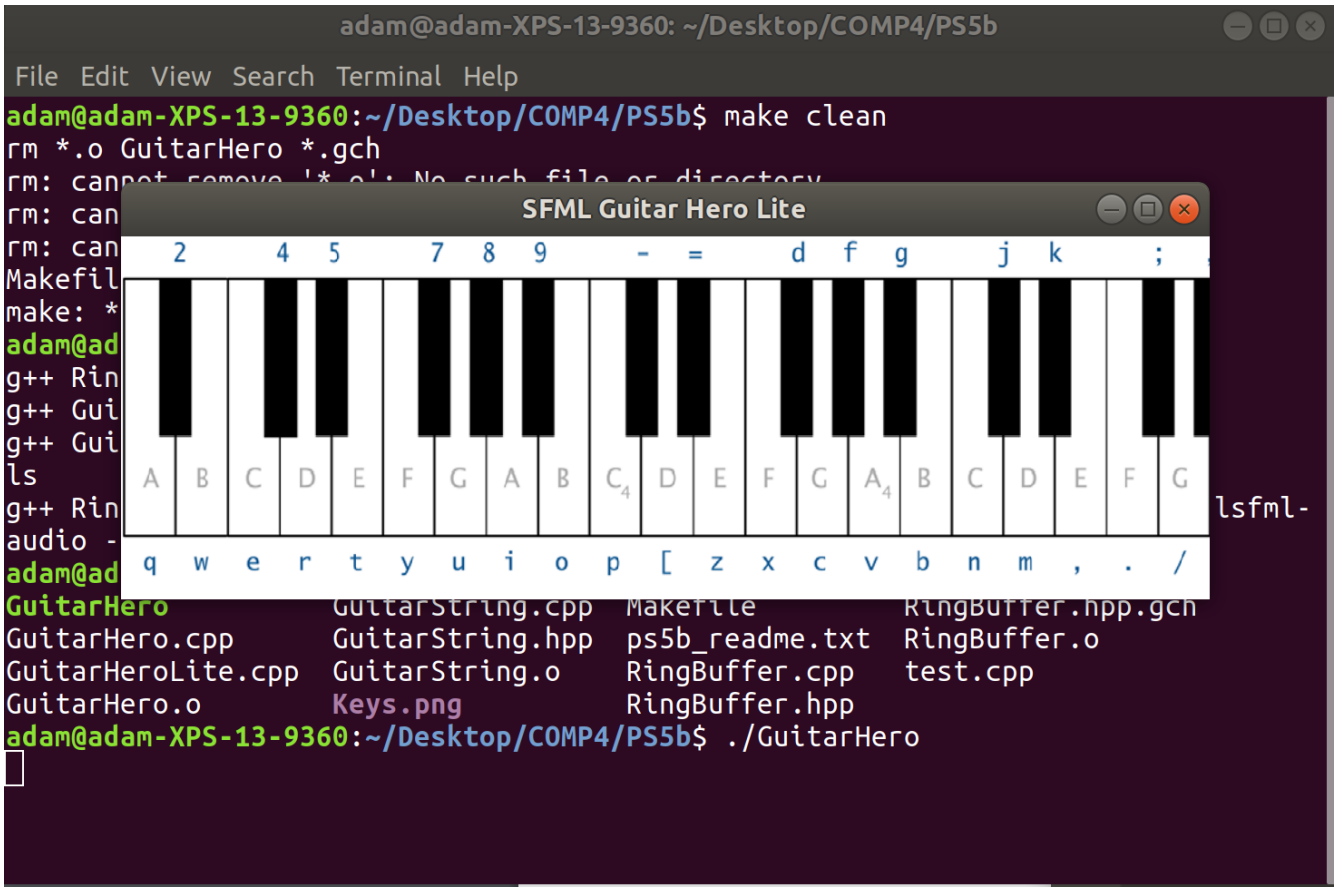
PS5 Guitar Hero

In this assignment, I had to use the SFML library and a ring buffer to play different sounds when different keys were pressed. OO methods that were used in this assignment were classes, and data structures that were central to the assignment were vectors. Another key algorithm that was central to this assignment was queues. In this assignment I created a stack for queue, then implemented enqueue, dequeue, isfull, isempty, and peek for the queue stack. I also implemented a few other functions to help me debug my code while I was writing it such as size, print, and get.

One issue I had with this assignment was that only two of the keys on the keyboard that were supposed to create sounds did not create any sound.



```
CC = g++
CFLAGS = -std=c++11 -c -g -Wall -Werror# -pedantic
LIBS = -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window

all: GuitarHero

GuitarHero: RingBuffer.o GuitarString.o GuitarHero.o
        $(CC) RingBuffer.o GuitarString.o GuitarHero.o -o GuitarHero $(LIBS)

GuitarHero.o: RingBuffer.hpp GuitarString.hpp GuitarHero.cpp
        $(CC) GuitarHero.cpp $(CFLAGS)

GuitarString.o: RingBuffer.hpp GuitarString.hpp GuitarString.cpp
        $(CC) GuitarString.cpp $(CFLAGS)

RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
        $(CC) RingBuffer.cpp RingBuffer.hpp $(CFLAGS)

clean:
        rm *.o GuitarHero *.gch
```

```
#include <SFML/Audio.hpp>
#include <SFML/System.hpp>

#include <vector>
#include <cstdlib>
#include <cmath>

#include "RingBuffer.hpp"

using namespace std;
using namespace sf;

class GuitarString {
public:
    GuitarString(double frequency);
    GuitarString(vector<Int16> init);

    void pluck();

    void tic();
    Int16 sample();
    int time();

private:
    RingBuffer rb;
    int count;

};
```

```
#include "GuitarString.hpp"

#define DECAY 0.996

GuitarString::GuitarString(double frequency) : rb(ceil(44100/frequency)) {
    count = 0;
}
GuitarString::GuitarString(vector<Int16> init) : rb(init.size()) {
    for (unsigned i = 0; i < init.size(); i++)
        rb.enqueue(init[i]);
    count = 0;
}

void GuitarString::pluck() {
    rb.empty();
    while (!rb.isFull())
        rb.enqueue((int16_t)(rand() & 0xffff));
}

void GuitarString::tic() {
    int N1 = rb.dequeue();
    int N2 = rb.peek();
    rb.enqueue(DECAY * 0.5 * (N1 + N2));
    count++;
}
Int16 GuitarString::sample() {
    return rb.peek();
}
int GuitarString::time() {
    return count;
}
```

```
#include <stdint.h>
#include <iostream>
#include <vector>
#include <stdexcept>

#define DECAY 0.996

using namespace std;

class RingBuffer {
public:
    RingBuffer(int capacity);

    int ringSize();
    bool isEmpty();
    bool isFull();
    void enqueue(int16_t x);
    int16_t dequeue();
    int16_t peek();
    int get(int x);
    //void print();

    void empty();

private:
    int capacity, size, first, last;
    int16_t temp_first;
    std::vector<int16_t> vector;

};
```

```
/*
Copyright 2019 Adam Baptista


*/

#include "RingBuffer.hpp"

RingBuffer::RingBuffer(int capacity) {
    try {
        if (capacity < 1)
            throw invalid_argument
                ("Capacity cannot be less than or equal to 1.");
    } catch(invalid_argument& e) {
        cerr << "RB constuctor: capacity cannot be less than or equal to 1."
;
        cerr << endl;
        throw e;
    }
    //sets a certain amount of space for items, like vector = new int[capaci
ty];
    vector.reserve(capacity);
    for (int i = 0; i < capacity; i++)
        vector.push_back(0);
    size = 0;
    first = 0;
    last = 0;
    this->capacity = capacity;
}

int RingBuffer::ringSize() {
    return size;
}

bool RingBuffer::isEmpty() {
    if (size > 0)
        return false;
    return true;
}

bool RingBuffer::isFull() {
    if (capacity > size)
        return false;
    return true;
}

void RingBuffer::enqueue(int16_t x) {
    try {
        if (isFull())
            throw runtime_error
                ("Enqueue: can’t enqueue an full ring");
    } catch (runtime_error& e) {
        cerr << "Enqueue: can’t enqueue an full ring";
        cerr << endl;
        throw e;
    }
    vector[last] = x;
    if (last == capacity - 1) {
        last = 0;
    } else {
        last++;
```

```
    }
    size++;
}

int16_t RingBuffer::dequeue() {
    try {
        if (isEmpty())
            throw runtime_error
                ("Dequeue: can’t dequeue en empty buffer");
    } catch (runtime_error& e) {
        cerr << "Dequeue: can’t dequeue en empty buffer";
        cerr << endl;
        throw e;
    }
    temp_first = peek();
    if (first == capacity - 1) {
        first = 0;
    } else {
        first++;
    }
    size--;
    return temp_first;
}

int16_t RingBuffer::peek() {
    try {
        if (isEmpty())
            throw runtime_error
                ("Peek: can’t peek at an empty ring");
    } catch (runtime_error& e) {
        cerr << "Peek: can’t peek at an empty ring";
        cerr << endl;
        throw e;
    }
    return vector[first];
}

/*
void RingBuffer::print() {
    for (int i = 0; i < capacity; i++)
        cout << vector[i] << endl;
    cout << endl;
}
*/

void RingBuffer::empty() {
    size = 0;
    first = 0;
    last = 0;
}
```

```
 1: #include <SFML/Graphics.hpp>
 2: #include <SFML/System.hpp>
 3: #include <SFML/Audio.hpp>
 4: #include <SFML/Window.hpp>
 5:
 6: #include <math.h>
 7: #include <limits.h>
 8: #include <stdint.h>
 9:
10: #include <iostream>
11: #include <string>
12: #include <exception>
13: #include <stdexcept>
14: #include <vector>
15:
16: #include "GuitarString.hpp"
17:
18: #define HZ 44100
19:
20: vector<int16_t> makeSampleFromString(GuitarString gs) {
21: vector<int16_t> samples;
22:
23:     gs.pluck();
24:     int duration = 8;
25:     for (int i = 0; i < HZ * duration; i++) {
26:         gs.tic();
27:         samples.push_back(gs.sample());
28:     }
29:     return samples;
30: }
31:
32: int main() {
33:     Sprite background;
34:     Texture texture;
35:     if (!texture.loadFromFile("Keys.png")) {
36:         cout << "Failed to load background" << endl;
37:         return EXIT_FAILURE;
38:     }
39:     Vector2f backSize(texture.getSize());
40:     RenderWindow window(VideoMode(1200, 400), "SFML Guitar Hero Lite");
41:     background.setTexture(texture);
42:
43:     Vector2f WINSIZE(window.getSize());
44:     double xScale = (double) WINSIZE.x / backSize.x;
45:     double yScale = (double) WINSIZE.y / backSize.y;
46:
47:     background.setScale(xScale, yScale);
48:
49:
50:     Event event;
51:     double frequency;
52:     string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/â\200\231 ";
53:     vector<vector<int16_t> > samples(37);
54:     vector<Sound> sounds(37);
55:     vector<SoundBuffer> soundBuffers(37);
56:
57:     for(int i = 0; i < 37; i++) {
58:         frequency = 440 * pow(2, (i - 24) / 12.0);
59:         GuitarString gs(frequency);
60:         samples[i] = makeSampleFromString(gs);
61:         if (!soundBuffers[i].loadFromSamples(&samples[i][0], samples[i].size
```

```
(), 2, HZ))
            throw std::runtime_error("SoundBuffer: failed to load from sampl
es.");
                sounds[i].setBuffer(soundBuffers[i]);
    }

    while (window.isOpen()) {
        while (window.pollEvent(event)) {
            switch (event.type) {
            case Event::Closed:
                window.close();
                break;
            case Event::TextEntered:
                if (event.text.unicode < 128) {
                    string temp;
                    temp += static_cast<char>(event.text.unicode);
                    int index = keyboard.find(temp);
                    sounds[index].play();
                }
                if (Keyboard::isKeyPressed(Keyboard::Escape)) {
                    window.close();
                    break;
                }
                break;
            default:
                break;
            }
            window.clear();
            window.draw(background);
            window.display();
        }
    }
    return 0;
}
```

```
/*
  Copyright 2015 Fred Martin, fredm@cs.uml.edu
  Wed Apr  1 09:43:12 2015
  test file for GuitarString class

  compile with
  g++ -c GStest.cpp -lboost_unit_test_framework
  g++ GStest.o GuitarString.o RingBuffer.o -o GStest -lboost_unit_test_frame
work
*/

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

#include <vector>
#include <exception>
#include <stdexcept>

#include "GuitarString.hpp"

BOOST_AUTO_TEST_CASE(GS) {
  vector<sf::Int16> v;

  v.push_back(0);
  v.push_back(2000);
  v.push_back(4000);
  v.push_back(-10000);

  BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));

  GuitarString gs = GuitarString(v);

  // GS is 0 2000 4000 -10000
  BOOST_REQUIRE(gs.sample() == 0);

  gs.tic();
  // it’s now 2000 4000 -10000 996
  BOOST_REQUIRE(gs.sample() == 2000);

  gs.tic();
  // it’s now 4000 -10000 996 2988
  BOOST_REQUIRE(gs.sample() == 4000);

  gs.tic();
  // it’s now -10000 996 2988 -2988
  BOOST_REQUIRE(gs.sample() == -10000);

  gs.tic();
  // it’s now 996 2988 -2988 -4483
  BOOST_REQUIRE(gs.sample() == 996);

  gs.tic();
  // it’s now 2988 -2988 -4483 1984
  BOOST_REQUIRE(gs.sample() == 2988);

  gs.tic();
  // it’s now -2988 -4483 1984 0
  BOOST_REQUIRE(gs.sample() == -2988);

  // a few more times
```

```
  gs.tic();
  BOOST_REQUIRE(gs.sample() == -4483);
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 1984);
  gs.tic();
  BOOST_REQUIRE(gs.sample() == 0);
}
```