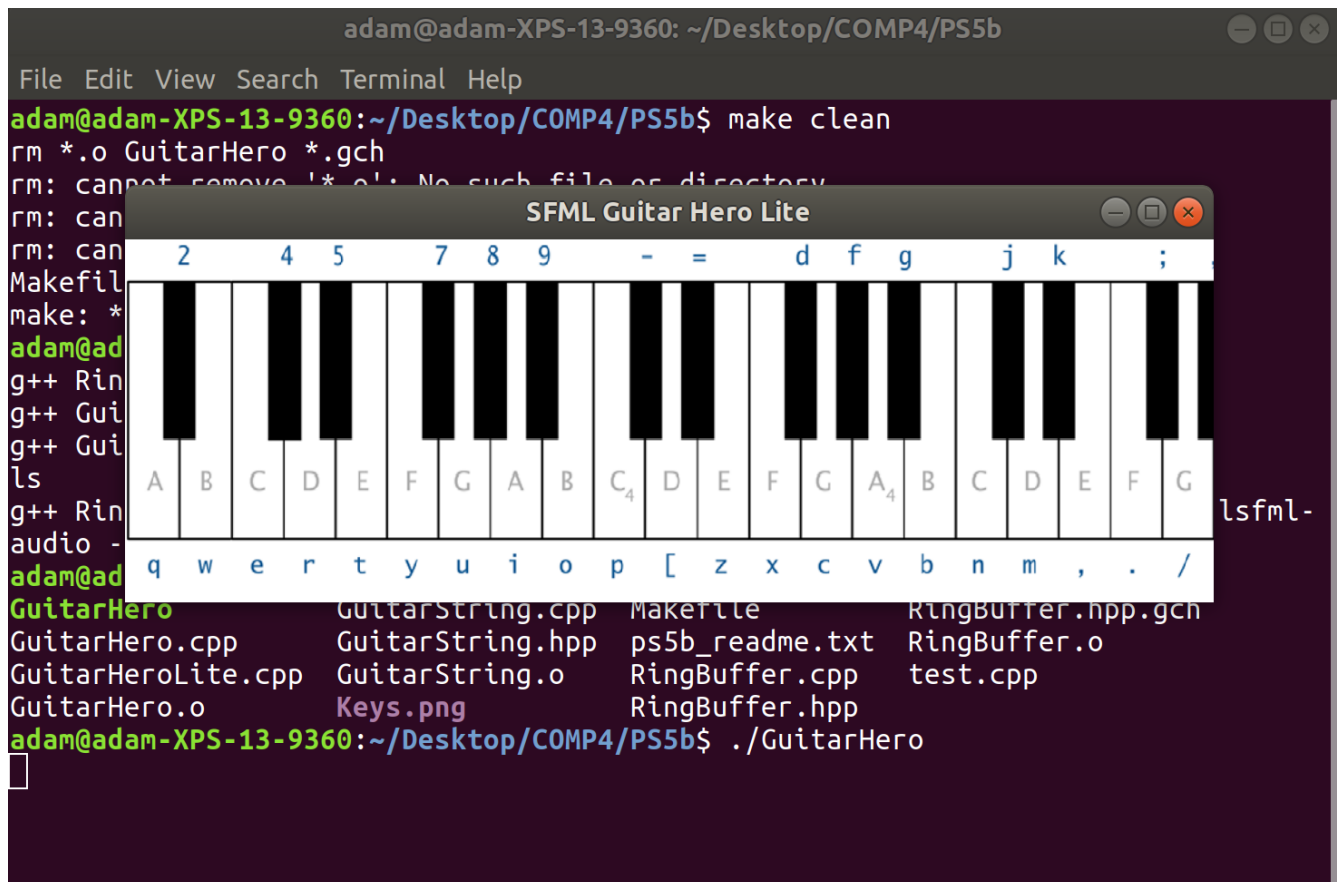


PS5 Guitar Hero

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.



The image shows a terminal window and an application window. The terminal window is titled "adam@adam-XPS-13-9360: ~/Desktop/COMP4/PS5b" and contains the following text:

```
File Edit View Search Terminal Help
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS5b$ make clean
rm *.o GuitarHero *.gch
rm: cannot remove '*.o': No such file or directory
rm: can
rm: can
Makefile
make: *
adam@ad
g++ Rin
g++ Gui
g++ Gui
ls
g++ Rin
audio -
adam@ad
GuitarHero    GuitarString.cpp  Makefile      RingBuffer.hpp.gch
GuitarHero.cpp  GuitarString.hpp  ps5b_readme.txt RingBuffer.o
GuitarHeroLite.cpp  GuitarString.o    RingBuffer.cpp test.cpp
GuitarHero.o      Keys.png          RingBuffer.hpp
adam@adam-XPS-13-9360:~/Desktop/COMP4/PS5b$ ./GuitarHero
```

The application window is titled "SFML Guitar Hero Lite" and displays a virtual guitar fretboard. The fretboard has 12 frets and 6 strings. The strings are labeled A, B, C, D, E, F from left to right. The frets are labeled 2, 4, 5, 7, 8, 9, -, =, d, f, g, j, k, ; from left to right. The keys on the keyboard are shown below the fretboard: q, w, e, r, t, y, u, i, o, p, [, z, x, c, v, b, n, m, ,, ., /.

```
1: CC = g++
2: CFLAGS = -std=c++11 -c -g -Wall -Werror# -pedantic
3: LIBS = -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window
4:
5: all: GuitarHero
6:
7: GuitarHero: RingBuffer.o GuitarString.o GuitarHero.o
8:     $(CC) RingBuffer.o GuitarString.o GuitarHero.o -o GuitarHero $(LIBS)
9:
10: GuitarHero.o: RingBuffer.hpp GuitarString.hpp GuitarHero.cpp
11:     $(CC) GuitarHero.cpp $(CFLAGS)
12:
13: GuitarString.o: RingBuffer.hpp GuitarString.hpp GuitarString.cpp
14:     $(CC) GuitarString.cpp $(CFLAGS)
15:
16: RingBuffer.o: RingBuffer.cpp RingBuffer.hpp
17:     $(CC) RingBuffer.cpp RingBuffer.hpp $(CFLAGS)
18:
19: clean:
20:     rm *.o GuitarHero *.gch
```

```
1: #include <SFML/Audio.hpp>
2: #include <SFML/System.hpp>
3:
4: #include <vector>
5: #include <cstdlib>
6: #include <cmath>
7:
8: #include "RingBuffer.hpp"
9:
10: using namespace std;
11: using namespace sf;
12:
13: class GuitarString {
14: public:
15:     GuitarString(double frequency);
16:     GuitarString(vector<Int16> init);
17:
18:     void pluck();
19:
20:     void tic();
21:     Int16 sample();
22:     int time();
23:
24: private:
25:     RingBuffer rb;
26:     int count;
27:
28: };
```

```
1: #include "GuitarString.hpp"
2:
3: #define DECAY 0.996
4:
5: GuitarString::GuitarString(double frequency) : rb(ceil(44100/frequency)) {
6:     count = 0;
7: }
8: GuitarString::GuitarString(vector<Int16> init) : rb(init.size()) {
9:     for (unsigned i = 0; i < init.size(); i++)
10:         rb.enqueue(init[i]);
11:     count = 0;
12: }
13:
14: void GuitarString::pluck() {
15:     rb.empty();
16:     while (!rb.isFull())
17:         rb.enqueue((int16_t)(rand() & 0xffff));
18: }
19:
20: void GuitarString::tic() {
21:     int N1 = rb.dequeue();
22:     int N2 = rb.peek();
23:     rb.enqueue(DECAY * 0.5 * (N1 + N2));
24:     count++;
25: }
26: Int16 GuitarString::sample() {
27:     return rb.peek();
28: }
29: int GuitarString::time() {
30:     return count;
31: }
```

```
1: #include <stdint.h>
2: #include <iostream>
3: #include <vector>
4: #include <stdexcept>
5:
6: #define DECAY 0.996
7:
8: using namespace std;
9:
10: class RingBuffer {
11: public:
12:     RingBuffer(int capacity);
13:
14:     int ringSize();
15:     bool isEmpty();
16:     bool isFull();
17:     void enqueue(int16_t x);
18:     int16_t dequeue();
19:     int16_t peek();
20:     int get(int x);
21:     //void print();
22:
23:     void empty();
24:
25: private:
26:     int capacity, size, first, last;
27:     int16_t temp_first;
28:     std::vector<int16_t> vector;
29:
30: };
```

```
1: /*
2: Copyright 2019 Adam Baptista
3:
4:
5: */
6:
7: #include "RingBuffer.hpp"
8:
9: RingBuffer::RingBuffer(int capacity) {
10:     try {
11:         if (capacity < 1)
12:             throw invalid_argument
13:                 ("Capacity cannot be less than or equal to 1.");
14:     } catch(invalid_argument& e) {
15:         cerr << "RB constructor: capacity cannot be less than or equal to 1."
16:         ;
17:         cerr << endl;
18:         throw e;
19:     }
20:     //sets a certain amount of space for items, like vector = new int[capaci
21: ty];
22:     vector.reserve(capacity);
23:     for (int i = 0; i < capacity; i++)
24:         vector.push_back(0);
25:     size = 0;
26:     first = 0;
27:     last = 0;
28:     this->capacity = capacity;
29: }
30:
31: int RingBuffer::ringSize() {
32:     return size;
33: }
34:
35: bool RingBuffer::isEmpty() {
36:     if (size > 0)
37:         return false;
38:     return true;
39: }
40:
41: bool RingBuffer::isFull() {
42:     if (capacity > size)
43:         return false;
44:     return true;
45: }
46:
47: void RingBuffer::enqueue(int16_t x) {
48:     try {
49:         if (isFull())
50:             throw runtime_error
51:                 ("Enqueue: can't enqueue an full ring");
52:     } catch (runtime_error& e) {
53:         cerr << "Enqueue: can't enqueue an full ring";
54:         cerr << endl;
55:         throw e;
56:     }
57:     vector[last] = x;
58:     if (last == capacity - 1) {
59:         last = 0;
60:     } else {
61:         last++;
62:     }
63:     size++;
64: }
```

```
60:     }
61:     size++;
62: }
63:
64: int16_t RingBuffer::dequeue() {
65:     try {
66:         if (isEmpty())
67:             throw runtime_error
68:                 ("Dequeue: can't dequeue en empty buffer");
69:     } catch (runtime_error& e) {
70:         cerr << "Dequeue: can't dequeue en empty buffer";
71:         cerr << endl;
72:         throw e;
73:     }
74:     temp_first = peek();
75:     if (first == capacity - 1) {
76:         first = 0;
77:     } else {
78:         first++;
79:     }
80:     size--;
81:     return temp_first;
82: }
83:
84: int16_t RingBuffer::peek() {
85:     try {
86:         if (isEmpty())
87:             throw runtime_error
88:                 ("Peek: can't peek at an empty ring");
89:     } catch (runtime_error& e) {
90:         cerr << "Peek: can't peek at an empty ring";
91:         cerr << endl;
92:         throw e;
93:     }
94:     return vector[first];
95: }
96:
97: /*
98: void RingBuffer::print() {
99:     for (int i = 0; i < capacity; i++)
100:         cout << vector[i] << endl;
101:     cout << endl;
102: }
103: */
104:
105: void RingBuffer::empty() {
106:     size = 0;
107:     first = 0;
108:     last = 0;
109: }
```

```
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))
62:         throw std::runtime_error("SoundBuffer: failed to load from sampl
es.");
63:         sounds[i].setBuffer(soundBuffers[i]);
64:     }
65:
66:     while (window.isOpen()) {
67:         while (window.pollEvent(event)) {
68:             switch (event.type) {
69:                 case Event::Closed:
70:                     window.close();
71:                     break;
72:                 case Event::TextEntered:
73:                     if (event.text.unicode < 128) {
74:                         string temp;
75:                         temp += static_cast<char>(event.text.unicode);
76:                         int index = keyboard.find(temp);
77:                         sounds[index].play();
78:                     }
79:                     if (Keyboard::isKeyPressed(Keyboard::Escape)) {
80:                         window.close();
81:                         break;
82:                     }
83:                     break;
84:                 default:
85:                     break;
86:             }
87:             window.clear();
88:             window.draw(background);
89:             window.display();
90:         }
91:     }
92:     return 0;
93: }
```

```
1: /*
2:  Copyright 2015 Fred Martin, fredm@cs.uml.edu
3:  Wed Apr 1 09:43:12 2015
4:  test file for GuitarString class
5:
6:  compile with
7:  g++ -c GStest.cpp -lboost_unit_test_framework
8:  g++ GStest.o GuitarString.o RingBuffer.o -o GStest -lboost_unit_test_frame
work
9: */
10:
11: #define BOOST_TEST_DYN_LINK
12: #define BOOST_TEST_MODULE Main
13: #include <boost/test/unit_test.hpp>
14:
15: #include <vector>
16: #include <exception>
17: #include <stdexcept>
18:
19: #include "GuitarString.hpp"
20:
21: BOOST_AUTO_TEST_CASE(GS) {
22:     vector<sf::Int16> v;
23:
24:     v.push_back(0);
25:     v.push_back(2000);
26:     v.push_back(4000);
27:     v.push_back(-10000);
28:
29:     BOOST_REQUIRE_NO_THROW(GuitarString gs = GuitarString(v));
30:
31:     GuitarString gs = GuitarString(v);
32:
33:     // GS is 0 2000 4000 -10000
34:     BOOST_REQUIRE(gs.sample() == 0);
35:
36:     gs.tic();
37:     // it's now 2000 4000 -10000 996
38:     BOOST_REQUIRE(gs.sample() == 2000);
39:
40:     gs.tic();
41:     // it's now 4000 -10000 996 2988
42:     BOOST_REQUIRE(gs.sample() == 4000);
43:
44:     gs.tic();
45:     // it's now -10000 996 2988 -2988
46:     BOOST_REQUIRE(gs.sample() == -10000);
47:
48:     gs.tic();
49:     // it's now 996 2988 -2988 -4483
50:     BOOST_REQUIRE(gs.sample() == 996);
51:
52:     gs.tic();
53:     // it's now 2988 -2988 -4483 1984
54:     BOOST_REQUIRE(gs.sample() == 2988);
55:
56:     gs.tic();
57:     // it's now -2988 -4483 1984 0
58:     BOOST_REQUIRE(gs.sample() == -2988);
59:
60:     // a few more times
```

```
61:  gs.tic();
62:  BOOST_REQUIRE(gs.sample() == -4483);
63:  gs.tic();
64:  BOOST_REQUIRE(gs.sample() == 1984);
65:  gs.tic();
66:  BOOST_REQUIRE(gs.sample() == 0);
67: }
```