```
    1: /*Copyright Sam Pickell 2017*/
    2: #include <limits.h>
    3: #include <cmath>
    4: #include <iostream>
    5: #include <string>
    6: #include <exception>
    7: #include <stdexcept>
    8: #include <vector>
    9:
   10: #include "GuitarString.hpp"
   11:
   12: #define CONCERT_A 440.0
   13: #define SAMPLES_PER_SEC 44100
   14:
   15: std::vector<sf::Int16> makeSamplesFromString(GuitarString& gs);
   16:
   17: int main() {
   18:   sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar Hero");
   19:   sf::Event event;
   20:   std::vector< std::vector<sf::Int16> > my_samples;
   21:   std::vector<sf::SoundBuffer*> my_buffers;
   22:   std::vector<sf::Sound> my_sounds;
   23:
   24:
   25:   for (int i = 0; i < 37; i++) {
   26:       // Step 1
   27:       double freq = CONCERT_A * pow(2, (static_cast<double>(i - 24.0) / 12.0
));
   28:       GuitarString gs = GuitarString(freq);
   29:       my_samples.push_back(makeSamplesFromString(gs));
   30:
   31:       // Step 2
   32:       sf::SoundBuffer* my_buf = new sf::SoundBuffer;
   33:       if (!my_buf->loadFromSamples(
   34:         &(my_samples.at(i))[0], my_samples.at(i).size(), 2, SAMPLES_PER_SEC)
) {
   35:           throw std::runtime_error(
   36:         "sf::SoundBuffer: failed to load from samples.");
   37:         }
   38:       my_buffers.push_back(my_buf);
   39:
   40:       // Step 3
   41:       sf::Sound my_sound;
   42:       my_sound.setBuffer(*my_buf);
   43:       my_sounds.push_back(my_sound);
   44:     }
   45:
   46:   while (window.isOpen()) {
   47:     while (window.pollEvent(event)) {
   48:       switch (event.type) {
   49:       case sf::Event::Closed:
   50:         window.close();
   51:         break;
   52:
   53:       case sf::Event::KeyPressed:
   54:         switch (event.key.code) {
   55:         case sf::Keyboard::Q:
   56:           my_sounds.at(0).play();
   57:           break;
   58:           case sf::Keyboard::Num2:
   59:           my_sounds.at(1).play();
```

```
   60:           break;
   61:           case sf::Keyboard::W:
   62:           my_sounds.at(2).play();
   63:           break;
   64:           case sf::Keyboard::E:
   65:           my_sounds.at(3).play();
   66:           break;
   67:           case sf::Keyboard::Num4:
   68:           my_sounds.at(4).play();
   69:           break;
   70:           case sf::Keyboard::R:
   71:           my_sounds.at(5).play();
   72:           break;
   73:           case sf::Keyboard::Num5:
   74:           my_sounds.at(6).play();
   75:           break;
   76:           case sf::Keyboard::T:
   77:           my_sounds.at(7).play();
   78:           break;
   79:           case sf::Keyboard::Y:
   80:           my_sounds.at(8).play();
   81:           break;
   82:           case sf::Keyboard::Num7:
   83:           my_sounds.at(9).play();
   84:           break;
   85:           case sf::Keyboard::U:
   86:           my_sounds.at(10).play();
   87:           break;
   88:           case sf::Keyboard::Num8:
   89:           my_sounds.at(11).play();
   90:           break;
   91:           case sf::Keyboard::I:
   92:           my_sounds.at(12).play();
   93:           break;
   94:           case sf::Keyboard::Num9:
   95:           my_sounds.at(13).play();
   96:           break;
   97:           case sf::Keyboard::O:
   98:           my_sounds.at(14).play();
   99:           break;
  100:           case sf::Keyboard::P:
  101:           my_sounds.at(15).play();
  102:           break;
  103:           case sf::Keyboard::Dash:
  104:           my_sounds.at(16).play();
  105:           break;
  106:           case sf::Keyboard::LBracket:
  107:           my_sounds.at(17).play();
  108:           break;
  109:           case sf::Keyboard::Equal:
  110:           my_sounds.at(18).play();
  111:           break;
  112:           case sf::Keyboard::Z:
  113:           my_sounds.at(19).play();
  114:           break;
  115:           case sf::Keyboard::X:
  116:           my_sounds.at(20).play();
  117:           break;
  118:           case sf::Keyboard::D:
  119:           my_sounds.at(21).play();
  120:           break;
```

```
  121:           case sf::Keyboard::C:
  122:           my_sounds.at(22).play();
  123:           break;
  124:           case sf::Keyboard::F:
  125:           my_sounds.at(23).play();
  126:           break;
  127:           case sf::Keyboard::V:
  128:           my_sounds.at(24).play();
  129:           break;
  130:           case sf::Keyboard::G:
  131:           my_sounds.at(25).play();
  132:           break;
  133:           case sf::Keyboard::B:
  134:           my_sounds.at(26).play();
  135:           break;
  136:           case sf::Keyboard::N:
  137:           my_sounds.at(27).play();
  138:           break;
  139:           case sf::Keyboard::J:
  140:           my_sounds.at(28).play();
  141:           break;
  142:           case sf::Keyboard::M:
  143:           my_sounds.at(29).play();
  144:           break;
  145:           case sf::Keyboard::K:
  146:           my_sounds.at(30).play();
  147:           break;
  148:           case sf::Keyboard::Comma:
  149:           my_sounds.at(31).play();
  150:           break;
  151:           case sf::Keyboard::Period:
  152:           my_sounds.at(32).play();
  153:           break;
  154:           case sf::Keyboard::SemiColon:
  155:           my_sounds.at(33).play();
  156:           break;
  157:           case sf::Keyboard::Slash:
  158:           my_sounds.at(34).play();
  159:           break;
  160:           case sf::Keyboard::Quote:
  161:           my_sounds.at(35).play();
  162:           break;
  163:           case sf::Keyboard::Space:
  164:           my_sounds.at(36).play();
  165:           break;
  166:         default:
  167:           break;
  168:         }
  169:
  170:       default:
  171:         break;
  172:       }
  173:
  174:       window.clear();
  175:       window.display();
  176:     }
  177:   }
  178:
  179:   // Clean up after the vector of pointers
  180:   for (unsigned int i = 0; i < my_buffers.size(); i++) {
  181:       delete my_buffers.at(i);
```

```
182:     }
183:   return 0;
184: }
185:
186: std::vector<sf::Int16> makeSamplesFromString(GuitarString& gs) {
187:   std::vector<sf::Int16> samples;
188:
189:   gs.pluck();
190:   int duration = 8;
191:   int i;
192:   for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
193:     gs.tic();
194:     samples.push_back(gs.sample());
195:   }
196:
197:   return samples;
198: }
```

```
 1: /*Copyright Sam Pickell 2017*/
 2:
 3: #ifndef GUITARSTRING_HPP
 4: #define GUITARSTRING_HPP
 5:
 6: #include <SFML/Graphics.hpp>
 7: #include <SFML/System.hpp>
 8: #include <SFML/Window.hpp>
 9: #include <SFML/Audio.hpp>
10: #include <vector>
11: #include "RingBuffer.hpp"
12:
13: class GuitarString {
14:  public:
15:   GuitarString();
16:   explicit GuitarString(double frequency);
17:   explicit GuitarString(std::vector<sf::Int16> init);
18:   ~GuitarString();
19:   void pluck();
20:   void tic();
21:   sf::Int16 sample();
22:   int time();
23:
24:  private:
25:   RingBuffer* data;
26:   int tic_tracker;
27: };
28: #endif
```

```
 1: /*Copyright Sam Pickell 2017*/
 2: #include "GuitarString.hpp"
 3: #include <cmath>
 4: #include <cstdlib>
 5: #include <vector>
 6:
 7: GuitarString::GuitarString() {
 8: }
 9:
10: GuitarString::GuitarString(double frequency) {
11:   int N = ceil(44100/frequency);
12:   data = new RingBuffer(N);
13:   for (int i = 0; i < N; i++) {
14:       data->enqueue(0);
15:     }
16: }
17:
18: GuitarString::GuitarString(std::vector<sf::Int16> init) {
19:   data = new RingBuffer(init.size());
20:   for (unsigned int i = 0; i < init.size(); i++) {
21:       data->enqueue(init.at(i));
22:     }
23: }
24:
25: GuitarString::~GuitarString() {
26:   delete data;
27: }
28:
29: void GuitarString::pluck() {
30:   unsigned int r = 123;
31:   for (int i = 0; i < data->size(); i++) {
32:       int16_t random_var = (int16_t)(rand_r(&r) % 0xffff);
33:       data->dequeue();
34:       data->enqueue(random_var);
35:     }
36: }
37:
38: void GuitarString::tic() {
39:   int16_t KS_update = data->peek();
40:   data->dequeue();
41:   KS_update = ((KS_update + data->peek()) / 2) * .996;
42:   data->enqueue(KS_update);
43:
44:   tic_tracker++;
45: }
46:
47: sf::Int16 GuitarString::sample() {
48:   return data->peek();
49: }
50:
51: int GuitarString::time() {
52:   return tic_tracker;
53: }
```

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=GuitarHero.o RingBuffer.o GuitarString.o
 4: P=GuitarHero
 5: SFML= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 6: all: $(P)
 7: $(P):$(O)
 8:         $(C) -o $(P) $(O) $(SFML)
 9:
10: $(E).o:
11:         $(C) -c $<  -o $@
12:
13: clean:
14:         rm $(O) $(P)
```