

```
1: #include "StringSound.hpp"
2: #include <vector>
3:
4:
5: StringSound::StringSound(double frequency):
6:     buff(ceil(SAMPLING_RATE / frequency)) {
7:     num = ceil(SAMPLING_RATE / frequency);
8:
9:     for (int i = 0; i < num; i++) {
10:         buff.enqueue((int16_t)0);
11:     }
12:     tictic = 0;
13: }
14:
15:
16: StringSound::StringSound(std::vector<sf::Int16> init):
17:     buff(init.size()) {
18:     num = init.size();
19:
20:     std::vector<sf::Int16>::iterator it;
21:
22:     for (it = init.begin(); it < init.end(); it++) {
23:         buff.enqueue((int16_t)*it);
24:     }
25:     tictic = 0;
26: }
27:
28: void StringSound::pluck() {
29:     for (int i = 0; i < num; i++) {
30:         buff.dequeue();
31:     }
32:
33:     for (int i = 0; i < num; i++) {
34:         buff.enqueue((sf::Int16)(rand() & 0xffff));
35:     }
36:
37:     return;
38: }
39:
40:
41: void StringSound::tic() {
42:     int16_t first = buff.dequeue();
43:     int16_t second = buff.peek();
44:
45:     int16_t avg = (first + second) / 2;
46:     int16_t karplus = avg * ENERGY_DECAY_FACTOR;
47:
48:     buff.enqueue((sf::Int16)karplus);
49:
50:     tictic++;
51:
52:     return;
53: }
54:
55:
56: // return current sample
57: sf::Int16 StringSound::sample() {
58:
59:     sf::Int16 sample = (sf::Int16)buff.peek();
60:
61:     return sample;
62: }
63:
64:
65: // number of tics called
```

```
66: int StringSound::time() {  
67:     return tictic;  
68: }
```