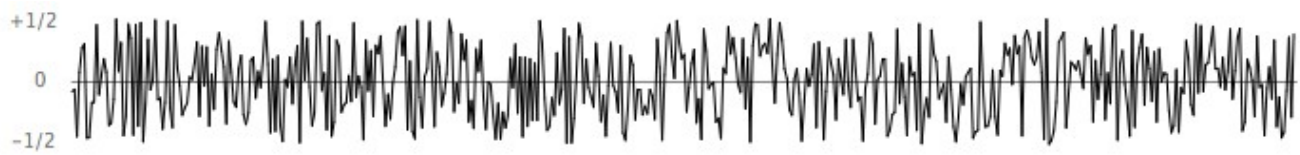
PS5b - Karplus-Strong String Simulation and Guitar Hero

In Part B, I implemented the Karplus-Strong guitar string simulation and generated a stream of string samples for audio playback using the keyboard. I declare a pointer to a RingBuffer in the class GuitarString as a private member variable instead of declaring a RingBuffer object itself. This was my first time using another class in another implementation in another class, so it was interesting figuring this out. I learned that you cannot instantiate the ring buffer until the GuitarString constructor is called at run time since we don't know how big the ring buffer will be until given the frequency of the string. This program simulates the plucking of a guitar string as shown in:

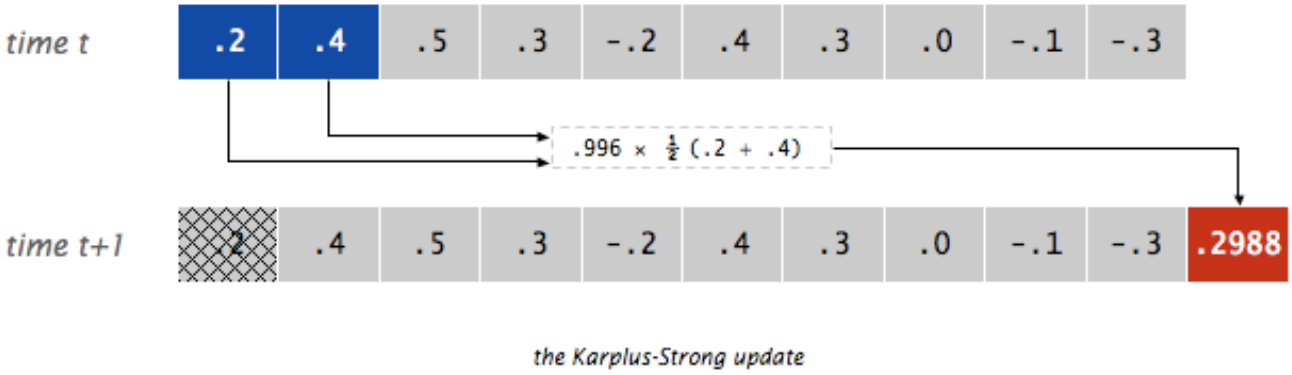
```
32: void GuitarString::pluck()
```

When a guitar is blocked, the string vibrates and creates sound. The pluck function fills the guitar string's ring buffer with random numbers over the int16_t range, which holds values from -32768 to 32767. Also when plucked, the ring buffer may be full so it empties the ring buffer by dequeuing the values until it is empty.



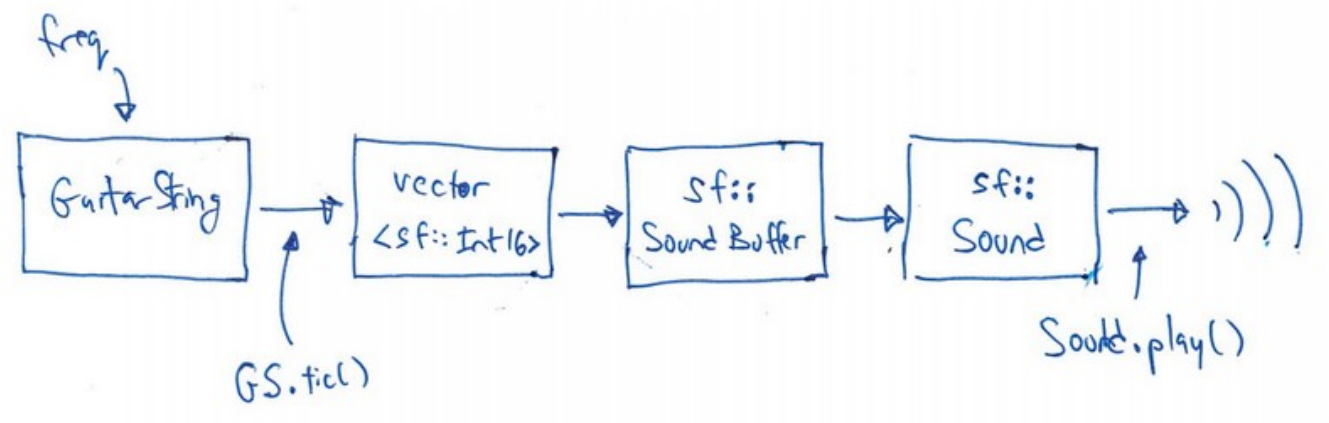
White noise created from plucking the string.

After the string is plucked, it vibrates which causes a displacement that spreads like a wave. The Karplus-Strong algorithm simulates vibrations by maintaining a ring buffer of N samples. The tic() function which pushes random numbers in the ring buffer are then used in the algorithm which finds the average of the front of the buffer to the buffer the second item in the buffer and then scaled by an energy decay factor of 0.996.



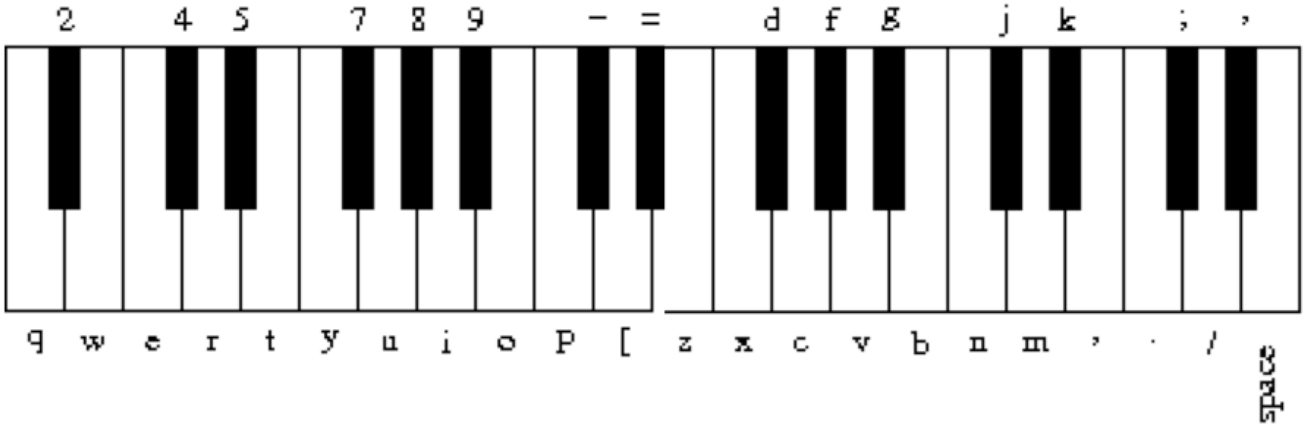
Example of the Karplus-Strong algorithm.

With the stream of values from the GuitarString object, each one is retrieved one value at at time and immediately given to an audio playback object. SFML proviveds an existing SF::SoundBuffer that is created with a vector of sound samples, which is created from a vector of sf::Int16s. From there I create an sf::Sound object from the sf::SoundBuffer which then, the sf::Sound object can then be played.

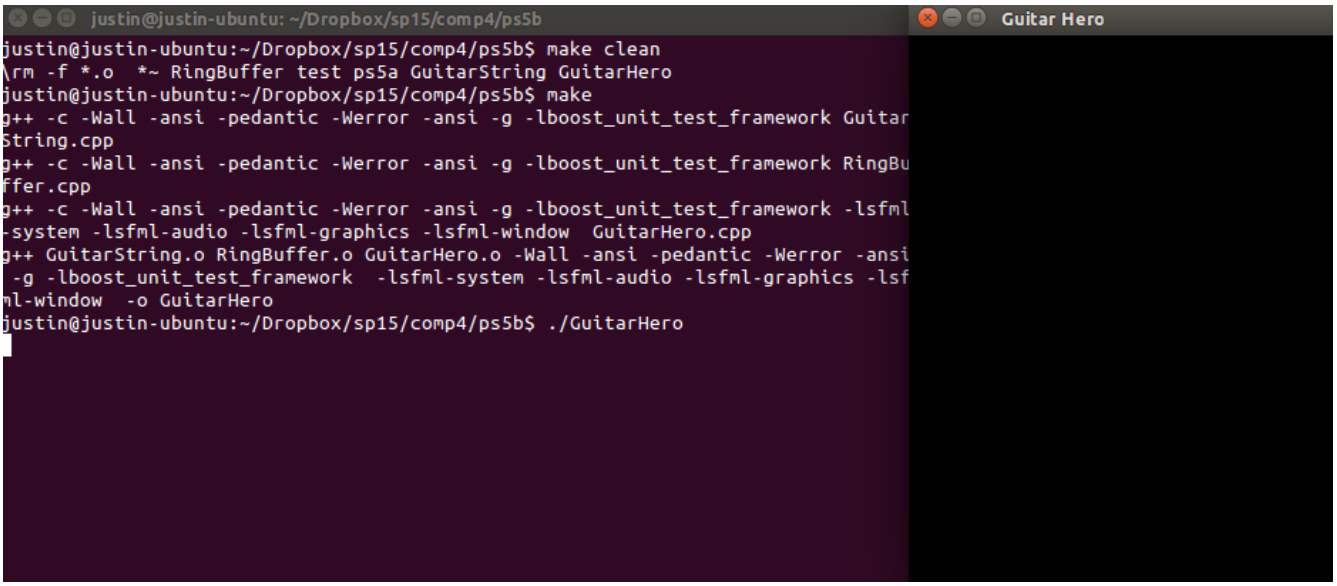


The whole sequence described above.

After all of that is set up, the next items to be done are turning the keyboard into an electronic keyboard to output a sound when pressed and implementing the frequency the sounds were to be played at. The keyboard was set up by using the sf::TextEntered event, which gives you the ascii value of that character.



An image of a electronic keyboard that would represent the keys being pressed on the keyboard.



Compiling and executing the program.

Note: if you want to see a demonstration of the running program, feel free to let me know and I can set it up.

```
CC = g++
OFLAGS = -c -Wall -ansi -pedantic -Werror -ansi -g -lboost_unit_test_framework
CFLAGS = -Wall -ansi -pedantic -Werror -ansi -g -lboost_unit_test_framework
LFLAGS = -lsfml-system -lsfml-audio -lsfml-graphics -lsfml-window

all:    GuitarHero

GuitarHero: GuitarString.o RingBuffer.o GuitarHero.o
        $(CC) GuitarString.o RingBuffer.o GuitarHero.o $(CFLAGS) $(LFLAGS) -o GuitarHero

GuitarHero.o: GuitarHero.cpp
        $(CC) $(OFLAGS) $(LFLAGS) GuitarHero.cpp

#GStest.o: GStest.cpp
#       $(CC) $(OFLAGS) $(LFLAGS) GStest.cpp

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

GuitarString.o: GuitarString.cpp GuitarString.hpp
        $(CC) $(OFLAGS) GuitarString.cpp

clean:
        \rm -f *.o  *~ RingBuffer test ps5a GuitarString GuitarHero
```

```
// Copyright [2015] Justin Nguyen  [legal/copyright]

#include <SFML/Graphics.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Window.hpp>

#include <math.h>
#include <limits.h>

#include <iostream>
#include <string>
#include <exception>
#include <stdexcept>
#include <vector>

#include "RingBuffer.hpp"
#include "GuitarString.hpp"

#define SAMPLES_PER_SEC 44100

double getFreq(int x) {
  double CONCERT_A = 440.0;
  double CONCERT_C = CONCERT_A * pow(2.0, ((x-24.0)/12.0));
  return CONCERT_C;
}

std::vector<sf::Int16> makeSamplesFromString(GuitarString gs) {
  std::vector<sf::Int16> samples;

  gs.pluck();
  int duration = 8;  // seconds
  int i;
  for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
    gs.tic();
    samples.push_back(gs.sample());
  }

  return samples;
}

int main() {
  sf::RenderWindow window(sf::VideoMode(300, 200), "Guitar Hero");
  sf::Event event;
  double freq;
  unsigned int index;
  std::vector <std::vector <sf::Int16> > samples(37);
  std::vector <sf::SoundBuffer> soundBuffers(37);
  std::vector <sf::Sound> sounds(37);
  for (int i = 0; i < 37; i++) {
    freq = getFreq(i);
    samples.at(i) = makeSamplesFromString(GuitarString(freq));
    if (!(soundBuffers.at(i)).loadFromSamples
       (&(samples.at(i))[0], (samples.at(i)).size(), 2, SAMPLES_PER_SEC))
      throw std::runtime_error("sf::SoundBuffer: failed to load from samples
.");
    (sounds.at(i)).setBuffer(soundBuffers.at(i));
  }
  std::string keyboard = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";
  while (window.isOpen()) {
    while (window.pollEvent(event)) {
```

```
      switch (event.type) {
      case sf::Event::Closed:
        window.close();
        break;
      case sf::Event::TextEntered:
        index = keyboard.find(event.text.unicode);
        if (index != std::string::npos)
         sounds[index].play();
        break;
      default:
        break;
      }
    }
    window.clear();
    window.display();
  }
  return 0;
}
```

```
// Copyright [2015] Justin Nguyen  [legal/copyright]

#include "GuitarString.hpp"
#include <stdint.h>
#include <math.h>
#include <SFML/Audio.hpp>
#include <vector>
#include <iostream>
#include "RingBuffer.hpp"

GuitarString::GuitarString(double frequency) {
  count = 0;
  N = ceil(frequency);
  ptrRB = new RingBuffer(N);
  while ((*ptrRB).isEmpty())
    (*ptrRB).enqueue(0);
}

GuitarString::GuitarString(std::vector<sf::Int16> init) {
  count = 0;
  N = init.size();
  ptrRB = new RingBuffer(N);
  for (std::vector<sf::Int16>::
          iterator it = init.begin(); it != init.end(); ++it)
    (*ptrRB).enqueue(*it);
}

GuitarString::~GuitarString() {
  delete ptrRB;
}

void GuitarString::pluck() {
  while (!(*ptrRB).isEmpty())
    (*ptrRB).dequeue();
  while (!(*ptrRB).isFull())
    (*ptrRB).enqueue((sf::Int16)(rand() & 0xffff)); //NOLINT
}

void GuitarString::tic() {
  int16_t front = (*ptrRB).dequeue();
  int16_t frontNext = (*ptrRB).peek();
  float result = ((front + frontNext)/2) * 0.996;
  (*ptrRB).enqueue(result);
}

sf::Int16 GuitarString::sample() {
  return (*ptrRB).peek();
}

int GuitarString::time() {
  return count++;
}
```

```
// Copyright [2015] Justin Nguyen  [legal/copyright]

#ifndef GS_HPP_
#define GS_HPP_

#include <stdint.h>
#include <SFML/Audio.hpp>
#include <vector>
#include "RingBuffer.hpp"

class GuitarString {
 public:
  explicit GuitarString(double frequency);
  explicit GuitarString(std::vector<sf::Int16> init);
  ~GuitarString();
  void pluck();
  void tic();
  sf::Int16 sample();
  int time();
  int count;
 private:
  RingBuffer *ptrRB;
  int N;
};

#endif
```

```
// Copyright [2015] Justin Nguyen  [legal/copyright]

#include "RingBuffer.hpp"
#include <stdint.h>
#include <vector>
#include <iostream>
#include <stdexcept>

RingBuffer::RingBuffer(int capacity) {
  if (capacity < 1) {
    throw std::invalid_argument
    ("RB constructor: capacity must be greater than zero.");
  }
    ringBuffer.reserve(capacity);
}

int RingBuffer::size() {
  return ringBuffer.size();
}

bool RingBuffer::isEmpty() {
  if (size() == 0)
    return 1;
  else
    return 0;
}

bool RingBuffer::isFull() {
  int cur_cap = ringBuffer.capacity();
  if (size() == cur_cap)
    return 1;
  else
    return 0;
}

void RingBuffer::enqueue(int16_t x) {
  if (!isFull())
    ringBuffer.push_back(x);
  else
    throw std::runtime_error("enqueue: can’t enqueue to a full ring.");
}

int16_t RingBuffer::dequeue() {
  int16_t temp;
  temp = ringBuffer.front();
  if (!isEmpty())
    ringBuffer.erase(ringBuffer.begin(), ringBuffer.begin() + 1);
  else
      throw std::runtime_error("dequeue: can’t dequeue to a empty ring.");
  return temp;
  }

int16_t RingBuffer::peek() {
  if (!isEmpty())
    return ringBuffer.front();
  else
    throw std::runtime_error("peek: can’t dequeue to a empty ring.");
}
```

```
// Copyright [2015] Justin Nguyen  [legal/copyright]

#ifndef RB_HPP_
#define RB_HPP_

#include <stdint.h>
#include <vector>

class RingBuffer {
 public:
  explicit RingBuffer(int capacity);
  int size();
  bool isEmpty();
  bool isFull();
  void enqueue(int16_t x);
  int16_t dequeue();
  int16_t peek();

 private:
  std::vector<int16_t> ringBuffer;
};

#endif
```