// WAVE FORM GENERATOR

var WaveformGenerator = {

// The generateWaveform function takes 4 parameters:

// - type, the type of waveform to be generated

// - frequency, the frequency of the waveform to be generated

// - amp, the maximum amplitude of the waveform to be generated

// - duration, the length (in seconds) of the waveform to be generated

generateWaveform: function(type, frequency, amp, duration) {

var nyquistFrequency = sampleRate / 2; // Nyquist frequency

var totalSamples = Math.floor(sampleRate \* duration); // Number of samples to generate

var result = []; // The temporary array for storing the generated samples

if(frequency> nyquistFrequency){

console.log("ERROR: Frequency can not by higher than: ", nyquistFrequency);

frequency = nyquistFrequency;

return;

}

switch(type) {

case "sine-time": // Sine wave, time domain

for (var i = 0; i < totalSamples; ++i) {

var currentTime = i / sampleRate;

result.push(amp \* Math.sin(2.0 \* Math.PI \* frequency \* currentTime));

}

break;

case "square-time": // Square wave, time domain

var oneCycle = sampleRate/frequency;

for(var i =0; i<totalSamples; i++){

if((i%oneCycle)<(oneCycle/2))//first half of cycle

result.push(amp\*1);

else//second half of cycle

result.push(amp\*-1);

}

break;

case "square-additive": // Square wave, additive synthesis

for (var i = 0; i < totalSamples; ++i) {

var currentTime = i / sampleRate;

var sampleValue = 0;

// Add the sine waves, until the nyquist frequency is reached

var wave =1;

while(wave\*frequency < nyquistFrequency){

sampleValue += (1.0 / wave)\* Math.sin(2.0 \* Math.PI \* frequency \* wave \* currentTime);

wave+=2;

}

result.push(amp \* sampleValue);

}

break;

case "sawtooth-time": // Sawtooth wave, time domain

var oneCycle = sampleRate/frequency;

for(var i=0; i<totalSamples;i++){

var cycleFraction = (i%oneCycle)/oneCycle;

result.push((2\*amp\*(1-cycleFraction))-amp);

}

break;

case "sawtooth-additive": // Sawtooth wave, additive synthesis

for(var i=0; i<totalSamples;i++){

var currentTime = i/sampleRate;

var sampleValue =0;

var wave=1;

while(wave\*frequency <nyquistFrequency){

sampleValue += (1.0/wave) \* Math.sin(2.0 \* Math.PI \* frequency \* wave \* currentTime);

wave++;

}

result.push(amp\*sampleValue);

}

break;

case "triangle-additive": // Triangle wave, additive synthesis

for(var i =0; i<totalSamples; i++){

var currentTime = i/sampleRate;

var sampleValue =0;

var wave =1;

while (wave\*frequency < nyquistFrequency){

sampleValue +=(1/ (wave\*wave) ) \* Math.cos(2.0 \* Math.PI \* frequency \* wave \* currentTime);

wave+=2;

}

result.push(amp\* sampleValue);

}

break;

case "fm": // FM

var modFreq = $("#fm-modulation-frequency").val();

if(modFreq > nyquistFrequency){

console.log("Modulation frequency can not be higher than:", nyquistFrequency);

modFreq = nyquistFrequency;

}

var modAmp = $("#fm-modulation-amplitude").val();

console.log(modAmp);

for (var i = 0; i < totalSamples; ++i) {

var currentTime = i / sampleRate;

var modulator = modAmp \* Math.sin(2.0 \* Math.PI \* modFreq \* currentTime);

result.push(amp \* Math.sin(2.0 \* Math.PI \* frequency \* currentTime+modulator));

}

break;

case "karplus-strong": // Karplus-Strong algorithm

var kBase = $("#karplus-base").val();

var p = $("#karplus-p").val();

var b = $("#karplus-b").val();

if((b<0) || (b>1)){

console.log("b value must be within range [0,1]");

if(b<0)

b=0;

if(b>1)

b=1;

}

//fill 6 sec with sound

if(kBase == "256hz-sine"){//make sine wave

for (var i = 0; i < totalSamples; ++i) {

var currentTime = i / sampleRate;

result.push(amp \* Math.sin(2.0 \* Math.PI \* frequency \* currentTime));

}

}

else if (kBase == "white-noise"){//make white noise

for(var i=0; i<totalSamples; i++){

result.push(amp\* (Math.floor(Math.random() \* (1 +1 + 1)) -1));

}

}

for(var i = p+1; i<totalSamples; i++){

var t = Math.random();

if(t >= b)

result[i] = 0.5 \* (result[i-p] + result[i-p-1]);

else

result[i] = -0.5 \* (result[i-p] + result[i-p-1]);

}

break;

case "white-noise": // White noise

for(var i=0; i<totalSamples; i++){

result.push(amp\*((Math.random()\*2)-1));

}

break;

case "repeating-narrow-pulse": // Repeating narrow pulse

var cycle = Math.floor(sampleRate / frequency);

for (var i = 0; i < totalSamples; ++i) {

if(i % cycle === 0) {

result.push(amp \* 1.0);

} else if(i % cycle === 1) {

result.push(amp \* -1.0);

} else {

result.push(0.0);

}

}

break;

default:

break;

}

return result;