Streams I

Mike Winter (2006, revised 2007)

Streams I

Michael Benjamin Winter (2006, revised 2007)

In Streams I, glissandi pass through a set of 54 very selective and finely tuned resonators. The center frequency for each of the resonators is drawn from an available frequency range that expands and contracts from a set, starting frequency. The algorithm that determines these available ranges is outlined in the notes along with performance instructions. The input of the computer programs that implements the aforementioned algorithm is a signal from the live instruments that is then sent to a resonator bank.

Performance Instructions

Streams Ia – (original version)
for glissandi and resonators
performed by two or more instruments that, together, can glissando from C2 to C7

Performers play a continuous glissando at a steady dynamic throughout the piece that is slightly softer than the output of the resonators. The glissandi may change direction at any time. Also, the average pitch distance between instruments should constantly change so that the interaction between instruments ranges from playing close (in pitch) in respect to the other instruments with frequent crossings – to – independent more separated (in pitch) lines. The performers should focus playing between C2 to C3, C4 to C5, and C6 to C7, which are the ranges that the resonators are between. However, there should be glissandi throughout the entire range from slightly below C2 to slightly above C7. Within a twenty-second time period, all the glissandi of the entire ensemble should cover a distance in pitch from 9 to 15 octaves. Thus, the average speed in semitones per second for each instrument is 9 to 15 octaves divided by the number of instruments in the ensemble divided by twenty and multiplied by twelve. So the minimum and maximum semitones per second for the glissandi can be calculated as ((9/20/ensemble_size)*12) and ((15/20/ensemble_size)*12), respectively.
For example:

Ensemble Size	Minimum Gliss. Speed (semitones per second)	Maximum Gliss. Speed (semitones per second)
2	2.70	4.50
3	1.80	3.00
4	1.35	2.25
5	1.08	1.80
6	0.90	1.50
etc.		

Streams Ib
for glissandi and resonators
performed by one or more instruments that can glissando from C2 to C3, C4 to C5, C6 to C7, C4 to C7, C2 to C5, or C2 to C7

Performers play glissandi at a soft dynamic throughout the piece that is slightly softer than the output of the resonators. The glissandi may change direction at any time. Also, if multiple performers are playing, the average pitch distance between instruments should constantly change so that the interaction between instruments ranges from playing close (in pitch) in respect to the other instruments with frequent crossings – to – independent more separated (in pitch) lines. The performers should focus playing between C2 to C3, C4 to C5, and C6 to C7, which are the ranges that the resonators are between. The speed of the glissandi may vary so long as the piece does not become about gesture. For a performance with more than one player, performers may pause so long as there is always one person playing. Performers should explore the sonic-scape attempting to find the points of resonance, which are often changing over time.

The conceptual clarity of the process does not necessarily need to be audible.

Streams Ic
for noise and resonators
performed by one or more instruments that produce broadband noise (e.g. tam tam)

A tremelo or roll on the instrument producing constant soft noise that is slightly softer than the output of the resonators. For a performance with more than one player, performers may pause so long as there is always one person playing.

The Score

The included score is a graph of the available ranges for the resonators. The algorithm for these ranges uses a system/part/voice paradigm that is labeled in the score.

Computer Application

The piece runs with a computer application that is either included in the score package or can be obtained. The application takes a signal from the live instruments and passes them through a resonant filter bank. The center frequency of each resonator is determined by a random frequency chosen every three to five seconds from an available frequency range that changes over time (see **Algorithm** below). Though the application is included, the provided algorithm makes it possible to reprogram an equally functional application. As a glissando passes through the center frequency of one of the resonators, only the resonation of that pitch is sounded through loud speakers. The resonation should last between 5 and 10 seconds and should sound slightly louder than the glissandi from the instruments. When a resonator changes center frequency for resonation, the resonator should cross-fade quickly between the former center frequency and the new center frequency to eliminate any unwanted artifacts.

Algorithm

The algorithm given below uses a starting index of 0 as opposed to the documentation in the score, which shows from a starting index of 1. For example, systems 1-3 in the score are 0-2 in the algorithm. This also applies to the parts and the voices. In the algorithm, x = current time (in seconds) and is the timestamp for the start time of the resonation of that pitch, s = system, P[] = number of parts per system, p = part, v = voice, T = total time (in seconds), r = pitch (in cents from C2) of resonation, d = duration of resonation for that pitch. Also included in the following three pages, is java code that will print the output.

```
P[3] = {8, 6, 4};
T = 1200;
For[s = 0; s < 3; s++]{
    For[p = 0; p < P[s]; p++]{
        For[v = 0; v < 3; v++]{
            x = 0;
            While[x < T]{
```

$$u = \begin{cases} 1200 \log_2\left(-\frac{(p-2P[s]+2)\,2^{4-2s}}{P[s]-1}\right) & x < \frac{T\left(p+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+1\right)}{2(P[s]+2)(2s+1)} \\ 1200 \log_2\left(2^{4-2s}\left(\frac{p\left((p+1)T+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor T-2(P[s]+2)(2s+1)x\right)^2}{(P[s]-1)T^2}+\frac{-p+P[s]-1}{P[s]-1}+1\right)\right) & x < \frac{T\left(p+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+2\right)}{2(P[s]+2)(2s+1)} \\ 1200 \log_2\left(2^{4-2s}\right) & x < \frac{T\left(-p+2P[s]+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+2\right)}{2(P[s]+2)(2s+1)}; \\ 1200 \log_2\left(2^{4-2s}\left(\frac{p\left((p-2P[s]-3)T-2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor T+2(P[s]+2)(2s+1)x\right)^2}{(P[s]-1)T^2}+\frac{-p+P[s]-1}{P[s]-1}+1\right)\right) & x < \frac{T\left(-p+2P[s]+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+3\right)}{2(P[s]+2)(2s+1)} \\ 1200 \log_2\left(-\frac{(p-2P[s]+2)\,2^{4-2s}}{P[s]-1}\right) & x < \frac{T\left(\left\lfloor\frac{2sx+x}{T}\right\rfloor+1\right)}{2s+1} \end{cases}$$

$$l = \begin{cases} 1200 \log_2\left(-\frac{(p-2P[s]+2)\,2^{4-2s}}{P[s]-1}\right) & x < \frac{T\left(p+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+1\right)}{2(P[s]+2)(2s+1)} \\ 1200 \log_2\left(2^{4-2s}\left(\frac{(p-P[s]+1)\left((p+1)T+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor T-2(P[s]+2)(2s+1)x\right)^2}{(P[s]-1)T^2}+\frac{-p+P[s]-1}{P[s]-1}+1\right)\right) & x < \frac{T\left(p+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+2\right)}{2(P[s]+2)(2s+1)} \\ 1200 \log_2\left(2^{4-2s}\right)-1200 & x < \frac{T\left(-p+2P[s]+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+2\right)}{2(P[s]+2)(2s+1)}; \\ 1200 \log_2\left(2^{4-2s}\left(\frac{(p-P[s]+1)\left((p-2P[s]-3)T-2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor T+2(P[s]+2)(2s+1)x\right)^2}{(P[s]-1)T^2}+\frac{-p+P[s]-1}{P[s]-1}+1\right)\right) & x < \frac{T\left(-p+2P[s]+2(P[s]+2)\left\lfloor\frac{2sx+x}{T}\right\rfloor+3\right)}{2(P[s]+2)(2s+1)} \\ 1200 \log_2\left(-\frac{(p-2P[s]+2)\,2^{4-2s}}{P[s]-1}\right) & x < \frac{T\left(\left\lfloor\frac{2sx+x}{T}\right\rfloor+1\right)}{2s+1} \end{cases}$$

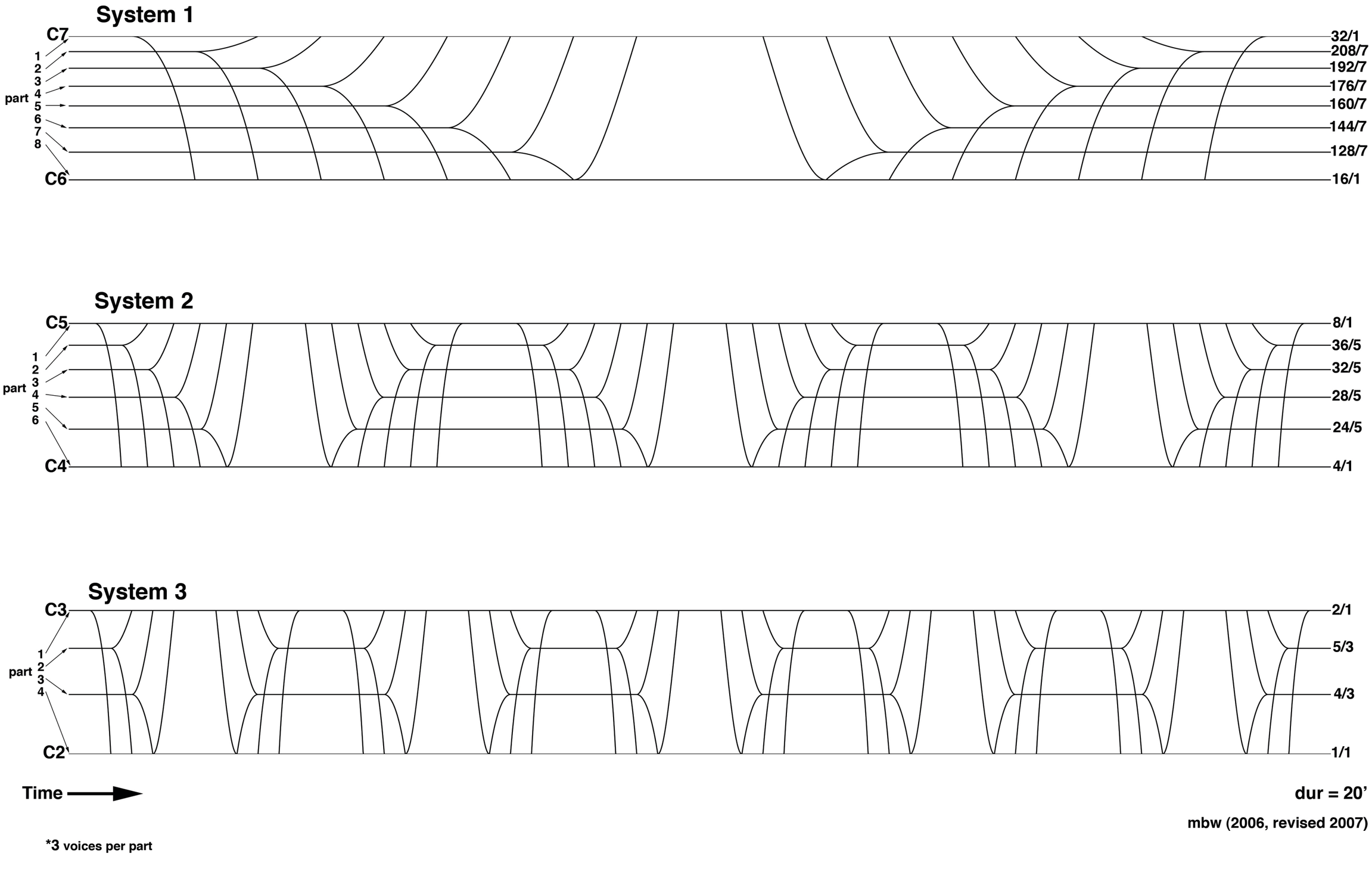
```
            r = l < randomNo < u;
            d = 3 < randomNo < 5;
            x = x + d;
            }
        }
    }
}
```

```
import java.util.ArrayList;

public class Generator {

    double[] noPartsPerSystem = {8, 6, 4};
    double totalTime = 1200; // (in seconds)
    double noSystems = 3;
    double noVoicesPerPart = 3;
    double x; // (current time in seconds)
    double upperLimit;
    double lowerLimit;
    double pitchToResonate; // (in cents from C2)
    double resonantFrequency;
    double duration; // (in seconds)
    ArrayList resonationInfo;
    int bangCount;
    int voiceIndex;
    int resonationIndex;

    public Generator() {
        render();
    }

    public void render() {

        bangCount = 0;
        voiceIndex = 0;
        resonationIndex = 0;
        resonationInfo = new ArrayList();

        for (int system = 0; system < noSystems; system++) {
            for (double part = 0; part < noPartsPerSystem[system]; part++) {
                for (int voice = 0; voice < noVoicesPerPart; voice++) {

                    x = 0;

                    while (x < totalTime) {

                        if (x < (totalTime*(part+2*(noPartsPerSystem[system]+2)*Math.floor((2*system*x+x)/totalTime)+1))/
                                (2*(noPartsPerSystem[system]+2)*(2*system+1))) {
                            upperLimit = (1200/Math.log(2))*Math.log(-1*(part-2*noPartsPerSystem[system]+2)*
                                    Math.pow(2,4-2*system)/(noPartsPerSystem[system]-1));
                            }
                        else if (x < (totalTime*(part+2*(noPartsPerSystem[system]+2)*Math.floor((2*system*x+x)/totalTime)+2))/
                                (2*(noPartsPerSystem[system]+2)*(2*system+1))) {
```

```
    upperLimit = (1200/Math.log(2))*Math.log(Math.pow(2,4-2*system)*(part*Math.pow((part+1)*totalTime+
            2*(noPartsPerSystem[system]+2)*Math.floor((2*system*x+x)/totalTime)*totalTime-
            2*(noPartsPerSystem[system]+2)*(2*system+1)*x,2)/
            ((noPartsPerSystem[system]-1)*Math.pow(totalTime,2))+(-part+noPartsPerSystem[system]-1)/
            (noPartsPerSystem[system]-1)+1));
}
else if (x < (totalTime*(-1*part+2*noPartsPerSystem[system]+2*(noPartsPerSystem[system]+2)*
        Math.floor((2*system*x+x)/totalTime)+2))/(2*(noPartsPerSystem[system]+2)*(2*system+1))) {
    upperLimit = (1200/Math.log(2))*Math.log(2*Math.pow(2,4-2*system));
}
else if (x < (totalTime*(-1*part+2*noPartsPerSystem[system]+2*(noPartsPerSystem[system]+2)*
        Math.floor((2*system*x+x)/totalTime)+3))/(2*(noPartsPerSystem[system]+2)*(2*system+1))) {
    upperLimit = (1200/Math.log(2))*Math.log(Math.pow(2,4-2*system)*(part*
            Math.pow((part-2*noPartsPerSystem[system]-3)*totalTime-2*(noPartsPerSystem[system]+2)
                    *Math.floor((2*system*x+x)/totalTime)*totalTime+2*(noPartsPerSystem[system]+2)
                    *(2*system+1)*x,2)/((noPartsPerSystem[system]-1)*Math.pow(totalTime,2))+
                    (-part+noPartsPerSystem[system]-1)/(noPartsPerSystem[system]-1)+1));
}
else if (x < totalTime*(Math.floor((2*system*x+x)/totalTime)+1)/(2*system+1)) {
    upperLimit = (1200/Math.log(2))*Math.log(-1*(part-2*noPartsPerSystem[system]+2)*
            Math.pow(2,4-2*system)/(noPartsPerSystem[system]-1));
}

if (x < (totalTime*(part+2*(noPartsPerSystem[system]+2)*Math.floor((2*system*x+x)/totalTime)+1))/
        (2*(noPartsPerSystem[system]+2)*(2*system+1))) {
    lowerLimit = (1200/Math.log(2))*Math.log(-1*(part-2*noPartsPerSystem[system]+2)*
            Math.pow(2,4-2*system)/(noPartsPerSystem[system]-1));
    }
else if (x < (totalTime*(part+2*(noPartsPerSystem[system]+2)*Math.floor((2*system*x+x)/totalTime)+2))/
        (2*(noPartsPerSystem[system]+2)*(2*system+1))) {
    lowerLimit = (1200/Math.log(2))*Math.log(Math.pow(2,4-2*system)*((part-noPartsPerSystem[system]+1)
            *Math.pow((part+1)*totalTime+2*(noPartsPerSystem[system]+2)*
                    Math.floor((2*system*x+x)/totalTime)*totalTime-2*(noPartsPerSystem[system]+2)*
                    (2*system+1)*x,2)/((noPartsPerSystem[system]-1)*Math.pow(totalTime,2))+
                    (-part+noPartsPerSystem[system]-1)/(noPartsPerSystem[system]-1)+1));
}
else if (x < (totalTime*(-1*part+2*noPartsPerSystem[system]+2*(noPartsPerSystem[system]+2)*
        Math.floor((2*system*x+x)/totalTime)+2))/(2*(noPartsPerSystem[system]+2)*(2*system+1))) {
    lowerLimit = (1200/Math.log(2))*Math.log(2*Math.pow(2,4-2*system))-1200;
}
else if (x < (totalTime*(-1*part+2*noPartsPerSystem[system]+2*(noPartsPerSystem[system]+2)*
        Math.floor((2*system*x+x)/totalTime)+3))/(2*(noPartsPerSystem[system]+2)*(2*system+1))) {
    lowerLimit = (1200/Math.log(2))*Math.log(Math.pow(2,4-2*system)*((part-noPartsPerSystem[system]+1)*
            Math.pow((part-2*noPartsPerSystem[system]-3)*totalTime-2*(noPartsPerSystem[system]+2)*
                    Math.floor((2*system*x+x)/totalTime)*totalTime+2*(noPartsPerSystem[system]+2)*
```

```
                                    (2*system+1)*x,2)/((noPartsPerSystem[system]-1)*Math.pow(totalTime,2))+
                                    (-part+noPartsPerSystem[system]-1)/(noPartsPerSystem[system]-1)+1));
                        }
                        else if (x < totalTime*(Math.floor((2*system*x+x)/totalTime)+1)/(2*system+1)) {
                            lowerLimit = (1200/Math.log(2))*Math.log(-1*(part-2*noPartsPerSystem[system]+2)*Math.pow(2,4-2*system)/
                                    (noPartsPerSystem[system]-1));
                            }

                        pitchToResonate = Math.random()*(upperLimit-lowerLimit)+lowerLimit;
                        duration = Math.random()*2+3;
                        resonantFrequency = 65.406395*Math.exp(pitchToResonate*Math.log(2)/1200);

                        //filter out repeats
                        if (resonationIndex == 0 ||
                                voiceIndex != ((ResonationInfo) resonationInfo.get(resonationIndex-1)).voiceIndex ||
                                (voiceIndex == ((ResonationInfo) resonationInfo.get(resonationIndex-1)).voiceIndex &&
                                    resonantFrequency != ((ResonationInfo) resonationInfo.get(resonationIndex-1)).resonantFrequency))
{
                            resonationInfo.add(new ResonationInfo(x, voiceIndex, resonantFrequency));
                            resonationIndex++;
                        }

                        System.out.println("System = " + (system+1) + "; Part = " + (part+1) + "; Voice = " + (voice+1) +
                                "; Timestamp = " + x + "; Pitch  = " + pitchToResonate +
                                "; Resonant Frequncy  = " + resonantFrequency + ";");

                        x = x+duration;
                    }
                    voiceIndex++;
                }
            }
        }
    }

    public static void main(String args[]) {
        Generator generator = new Generator();
    }
}
```

Streams I


System 1
C7
part
1
2
3
4
5
6
7
8
C6
32/1
208/7
192/7
176/7
160/7
144/7
128/7
16/1
System 2
C5
part
1
2
3
4
5
6
C4
8/1
36/5
32/5
28/5
24/5
4/1
Frequency Ratios
System 3
C3
part
1
2
3
4
C2
2/1
5/3
4/3
1/1
Pitch
Time
dur = 20’
mbw (2006, revised 2007)
*3 voices per part