Epitaffio di Silicio.
Traccia per improvvisazione in live coding

Nicola Buso

Ricevuto il 31 marzo 2017
Revisione dell'8 luglio 2017

1 Introduzione

In questo lavoro viene presentata una proposta di improvvisazione musicale al calcolatore basata su interfaccia testuale, secondo un paradigma esecutivo/compositivo consolidatosi progressivamente a partire dal volgere del millennio e impostosi, per lo più, sotto il nome di "live coding": il codice, che presiede alla generazione e trasformazione del suono come anche alla gestione dell'articolazione formale, viene scritto dal vivo e contestualmente esposto al pubblico.

Da un lato la gestualità della scrittura (la scrittura del codice in tempo reale) implica una significativa contrazione dei tempi decisionali, poiché programmare dal vivo, per dir così 'al volo', non concede al programmatore (esecutore/compositore[1]) l'agio della riflessione che può distendersi nel tempo differito, richiedendo al contrario decisioni rapide e intuizioni fulminee – oltre che una certa abilità dattilografica, virtuosistica.

D'altro lato il gesto della scrittura implica anche un'altrettanto significativa dilatazione dei tempi realizzativi: tipicamente, nel paradigma del live coding, per ottenere un singolo suono non basta pigiare un singolo tasto (come avviene per esempio al pianoforte), ma è necessaria la stesura di codice relativamente complesso (detto in modo informale: "si devono pigiare molti tasti..."); il controllo di ogni singolo suono attraverso il codice non può, verosimilmente, produrre risultati musicalmente rilevanti: il codice, pertanto, non dovrà, per l'appunto, essere vòlto a definire ogni singolo evento sonoro, ma una complessità di eventi; il gesto scrittorio non pretende (non può pretendere) di emulare l'immediatezza del gesto strumentale (che insiste sull'hic et nunc), ma impone piuttosto un approccio processuale (che abbracci un orizzonte temporale di più ampio respiro): la scrittura del codice, lungi dal proporsi il controllo diretto e immediato di ogni singolo evento, mirerà al controllo dei processi che determinano tutti gli eventi, inclinando fortemente verso un approccio algoritmico.

[1] Si veda l'accostamento tra le figure del pianista-compositore (XIX sec.) e quella di programmatore-compositore nel panorama del live coding in Click (2007).

Musica/Tecnologia, 11-12 (2017-2018), ISSN online 1974-0050, ISSN print 1974-0042, pp. 15-27
DOI: 10.13128/Music_Tec-23797 – http://www.fupress.com/mt

Il contrappunto dei tempi (dove la 'contrazione del tempo decisionale', imposto dalla programmazione dal vivo, si accompagna alla 'dilatazione del tempo realizzativo', dovuto all'abbandono dell'immediatezza del gesto strumentale per l'adozione, nel gesto scrittorio, della mediazione di una progettualità processuale algoritmica) il contrappunto dei tempi, dicevo, si manifesta sensibilmente nell'ostensione del codice, entrando in una sorta di dimensione teatrale, dove il corpo è insieme negato e affermato: negato perché cede la scena alla scrittura, affermato perché la scrittura è una scrittura in atto, l'atto dello scrivere, ovvero gesto[2]. Nell'ostensione del codice si assiste al contrappunto del corpo: sua presenza-assenza.

L'assenza del corpo è legata all'idea del suo sottrarsi e svanire, decomporsi forse, e dunque perire, come accade al codice, scritto sul momento: costruito, sviluppato, modificato, cancellato – giocoforza cancellato, per lasciare spazio a nuove stringhe (nuovi processi, nuovi suoni, etc.): il codice, per essere vivo, deve morire.

'Dead coding'?

1.1 Setup

Il setup audio è stereofonico, con rinforzo delle frequenze gravi sia sul canale destro che sinistro.

Può essere utilizzato un mixer con almeno due ingressi e quattro uscite indipendenti: le due uscite stereo principali e le due uscite dedicate alle frequenze gravi devono poter essere controllate separatamente, per agevolare la taratura preliminare del sistema.

Per tarare il sistema, prima dell'esecuzione, può essere utilizzato un frammento di codice basato sui diagrammi descritti in 'partitura'[3], che verrà presumibilmente utilizzato in concerto (particolarmente indicati, in quanto potenzialmente problematici, sono gli algoritmi descritti dai diagrammi 6 e 7).

Durante l'esecuzione si incoraggia il controllo del suono attraverso la sola interfaccia testuale (codice), senza ausilio di controller esterni (faders, motion capture, ecc.).

Il codice scritto dal vivo deve essere proiettato su uno schermo di fronte al pubblico, sulla scena.

Altoparlanti e subwoofer dedicati ai canali sinistro e destro saranno posti ai lati dello schermo.

L'esecutore si troverà nella posizione di ascolto più confortevole, circa in centro sala (tra il pubblico)[4], oppure, se questa collocazione non è possibile, al centro della scena[5].

[2] Ugo di San Vittore afferma: "gestus est modus et figuratio membrorum corporis, ad omnem agendi et habendi modum". Cit. in: J. C. Schmitt, (1990). *Il gesto nel medioevo*. Bari: Laterza.

[3] Con il termine 'partitura' (cfr. 3) viene qui indicato il complesso dei diagrammi di flusso (cfr. 3.1.2) che descrivono gli algoritmi da implementare, unitamente alla rappresentazione grafica (cfr. 3.3.2) di una ipotetica distribuzione temporale degli eventi generati dagli algoritmi medesimi.

[4] Si veda, infra: 2 Scheda tecnica, Stage plan A.

[5] Si veda, infra: 2 Scheda tecnica, Stage plan B.

Nessuna luce sull'esecutore, se non quella generata dal monitor del calcolatore: l'attenzione deve cadere sull'ascolto – e sulla proiezione del codice.

1.2 Algoritmi

Gli algoritmi descritti dai diagrammi (3.1) indicati in 'partitura' (3) devono essere implementati durante l'esecuzione scrivendo il codice 'al volo', utilizzando il linguaggio di programmazione ritenuto più opportuno[6].

Il Codice può essere scritto 'ex nihilo', oppure a partire da (piccoli) frammenti scritti prima dell'esecuzione, da completare e modificare (riscrivere) durante l'esecuzione stessa.

Non vi sono materiali preregistrati.

I diagrammi descrivono otto algoritmi; il primo diagramma (il più semplice) costituisce la base di tutti gli altri, in un processo che va dal semplice al complesso.

Il primo diagramma descrive un filtro passa banda del secondo ordine Butterworth con un offset costante sulla retroazione.

Nel secondo diagramma i filtri con retroazione diventano due, con frequenze di taglio differenti, con offset differente e distorsione.

Nel terzo diagramma la retroazione dei filtri è incrociata: l'uscita del primo filtro entra nell'ingresso del secondo, e viceversa.

Nel quarto diagramma i filtri sono quattro, nel percorso di retroazione viene introdotto un meccanismo di autoregolazione; l'uscita del primo e del secondo filtro, riscalate, vengono immesse nelle linee di retroazione del terzo e quarto filtro.

Il quinto diagramma ripropone lo schema del quarto, variando casualmente le frequenze di taglio dei quattro filtri.

Nel sesto diagramma un solo filtro passa banda (anche in questo caso di tipo Butterworth) con distorsione, offset e meccanismo di autoregolazione nella linea di retroazione cambia frequenza casualmente nel registro acuto ogni volta che il segnale supera la soglia dello 0; il segnale così ottenuto entra in un banco di filtri Allpass quando il suo valore RMS supera una determinata soglia; ogni filtro Allpass viene distorto e mandato in uscita.

Nel settimo diagramma il filtro passabanda descritto nel diagramma precedente varia la sua frequenza casualmente in un registro più ampio, dal grave all'acuto, e, quando il valore RMS del segnale filtrato supera una determinata soglia, il medesimo segnale entra in un banco di ritardi (con tempi di ritardo nell'ordine di centesimi di secondo) con filtro passa basso del secondo ordine Butterworth in retroazione e distorsione prima di essere inviato in uscita.

Nell'ultimo diagramma il filtro passabanda descritto nel sesto diagramma cambia frequenza in un registro ancora più ampio, ad intervalli regolari (con frequenza sub audio); quando il suo valore RMS supera una determinata soglia entra in un riverbero[7] di 18 sec e distribuito nell'uscita stereo.

[6] E.g.: SuperCollider, Chuck; si veda: implementazione degli algoritmi.

[7] Si veda e.g.: GVerb based on the "GVerb" LADSPA effect by Juhana Sadeharju (https://github.com/highfidelity/gverb).

L'ordine di complessità degli algoritmi suggerisce l'ordine di implementazione nel codice[8], e dunque l'ordine dell'esecuzione: l'algoritmo più complesso viene scritto modificando l'algoritmo più semplice (oppure una sua copia, al fine di ottenere un grado maggiore di polifonia).

L'uscita stereofonica può rimanere statica oppure resa dinamica.

1.3 Prospettive di sviluppo – frammenti

La scrittura del codice si allontana dall'immediatezza del gesto strumentale, per approdare ad una prospettiva algoritmica (automazione di processi e decisioni).

Automi.

L'ostensione del codice, nella sua metamorfosi, mostra i meccanismi che presiedono alla generazione e trasformazione del suono: come la prassi anatomica, è un processo di svelamento.

Anatomia del codice.

La modifica al volo del codice può comportare la perdita del codice precedente nell'affermazione del codice nuovo: emerge un senso di perdita, di un venir meno, di uno svanire[9].

Nascita e morte del codice.

La nascita, modificazione e morte del codice rese visibili attraverso la proiezione su schermo affermano il valore iconico-figurale e la valenza teatrale della parola: un teatro senza corpo, senza gesto – se non il gesto della scrittura[10].

Anatomia del codice nel teatro – disincarnato – della scrittura.

1.4 Bibliografia essenziale

N. Collins, (2007) *Towards Autonomous Agents for Live Computer Music: Realtime Machine Listening and Interactive Music Systems* (PhD Thesis). University of Cambridge: Centre for Science and Music, Faculty of Music.

[8] Anche se questo non è strettamente vincolante: è possibile seguire un ordine diverso.

[9] Sulla fugacità della vita: Epitaffio di Sicilo.

[10] Code as an Expressive Musical Instrument (Ge Wang 2004); Live Coding Practice Click Nilson (NIME, 2007).

N. Click, (2007) Live Coding Practice, *Proceedings of the International Conference on New Interfaces for Musical Expression.*

T. Magnusson, (2009) *Epistemic Tools The Phenomenology of Digital Musical Instruments,* University of Sussex, Submitted for the degree of Doctor of Philosophy.

A. McLean, (2011) *Artist-Programmers and Programming Languages for the Arts* Goldsmiths, University of London, Submitted for the degree of Doctor of Philosophy.

G. Wang, (2004) On-the-fly Programming: Using Code as an Expressive Musical Instrument, *Proceedings of the International Conference on New Interfaces for Musical Expression.*

G. Wang, (2008) *The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality*. (PhD Thesis), Princeton University.

Risorse in rete:

<http://supercollider.github.io> (ultima consultazione: 09/2017)

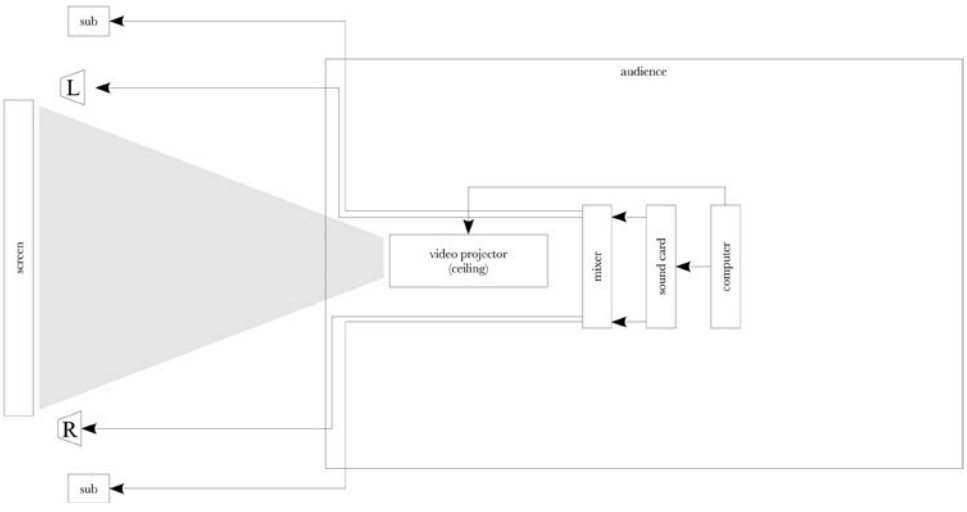
<http://toplap.org> (ultima consultazione: 09/2017.)

<https://www.bgc-jena.mpg.de/bgc-theory/index.php/Group/NathanielVirgo> (ultima consultazione: 09/2017.)

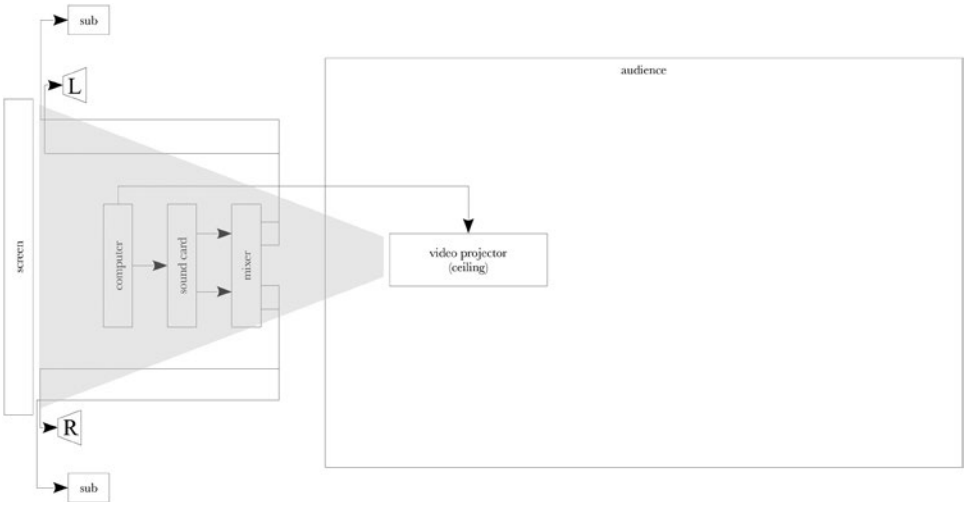
2 Scheda tecnica

Setup:
- computer
- stereo sound card
- mixer 2 in, 4 indipendent out
- 2 loudspeakers
- 2 subwoofer
- video projector

Stage plan A (better) :



Stage plan B:



3 Partitura

Siano implementati i seguenti algoritmi, secondo un ordine libero[11], utilizzando un'interfaccia testuale[12] – da rendere visibile al pubblico.

Sono possibili ripetizioni e sovrapposizioni, anche multiple, con fade in e fade out lenti oppure rapidi/percussivi (cfr. 3.3.1, 3.3.2).

3.1 Diagrammi

3.1.1 legenda

BPF = second order Butterworth bandpassfilter
freq = centre frequency (Hertz)
rq = bandwidth / frequency
RMS = root mean square
Allpass = Allpass delay line (delay time and decay time in seconds)
LPF = second order Butterworth lowpassfilter
Reverb, e.g.: GVerb based on the "GVerb" LADSPA effect by Juhana Sadeharju[13]

* = moltiplicazione

[11] L'ordine suggerito, anche se non vincolante, va dal semplice al complesso.

[12] Linguaggio utilizzato per l'implementazione degli algoritmi descritti nei diagrammi: distribuzione standard di SuperCollider 3.8 (<http://supercollider.github.io>).

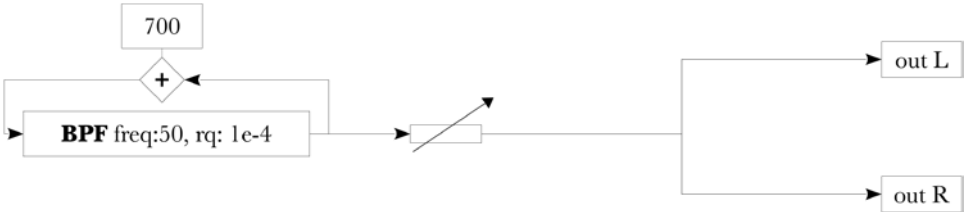
[13] < https://github.com/highfidelity/gverb> (ultima consultazione: 09/2017).

^ = elevamento a potenza
+ = somma
– = sottrazione

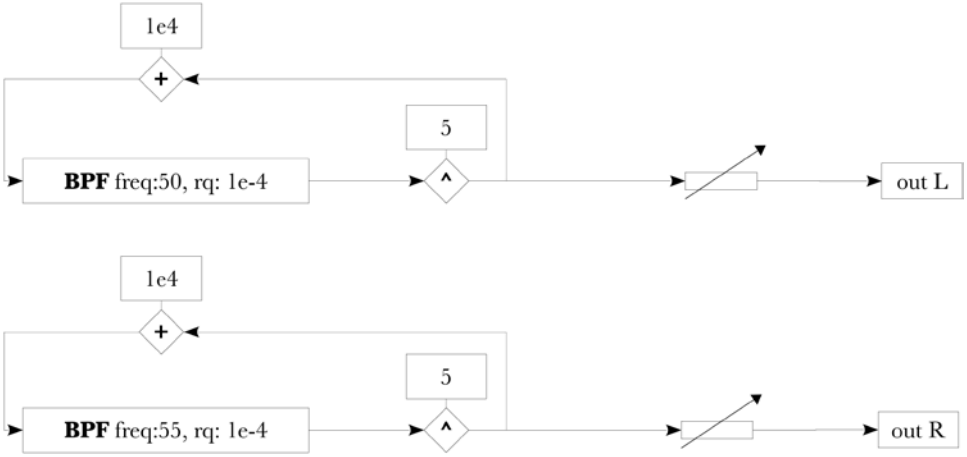
XeY =X*(10^Y); e.g.: 3e4=3*(10^4)=30000
Xe-Y =X*(10^(-Y)); e.g.: 3e-4=3*(10^(-4))=0.0003

 = potenziometro

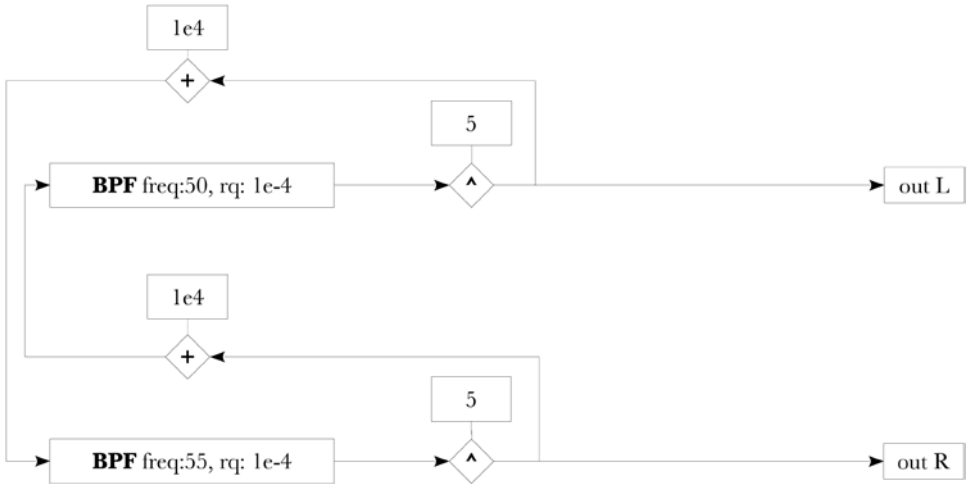
3.1.2 Diagramma 1



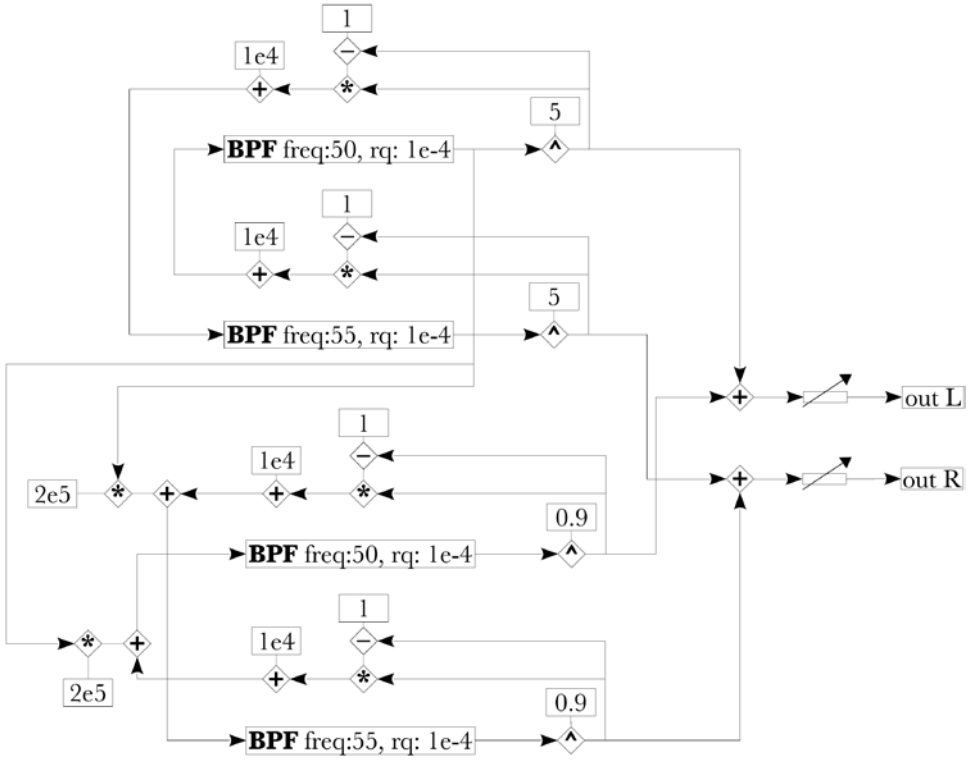
3.1.3 Diagramma 2



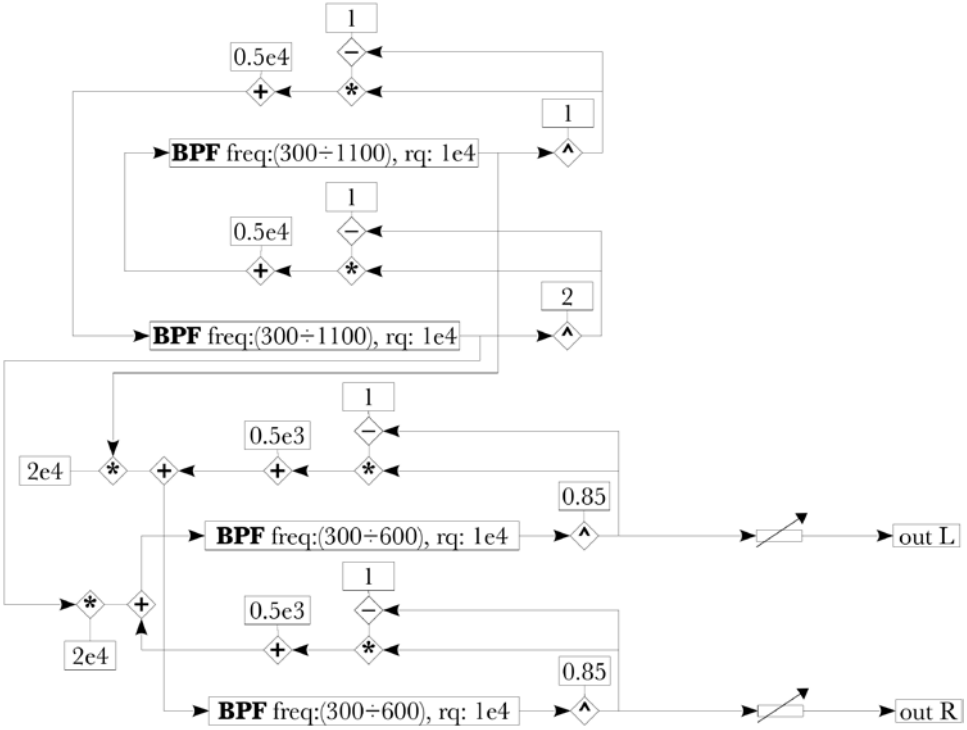
3.1.4 Diagramma 3



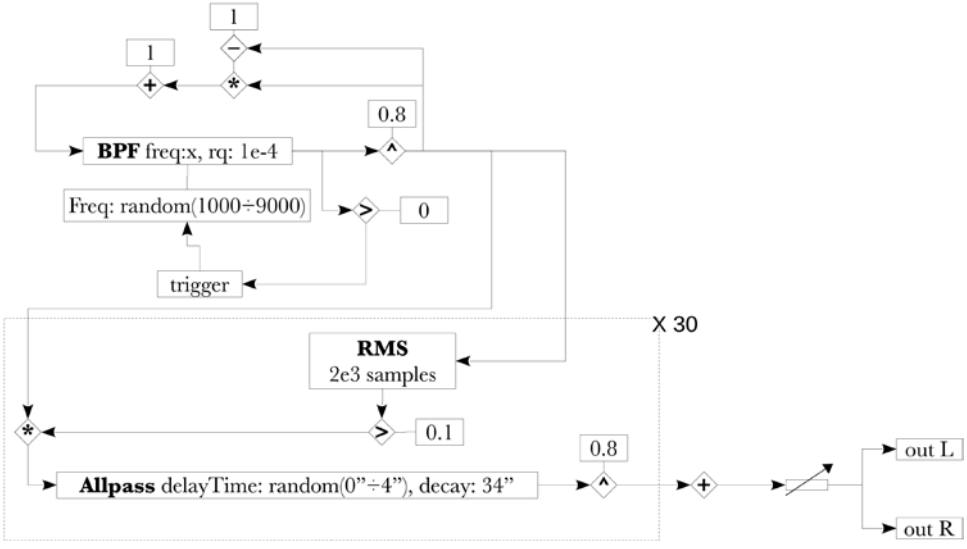
3.1.5 Diagramma 4



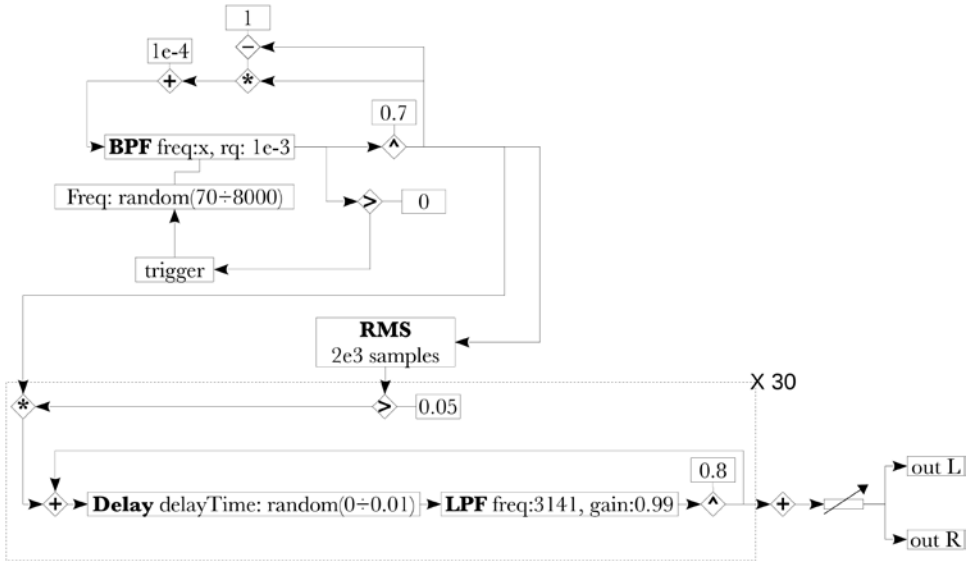
3.1.6 Diagramma 5



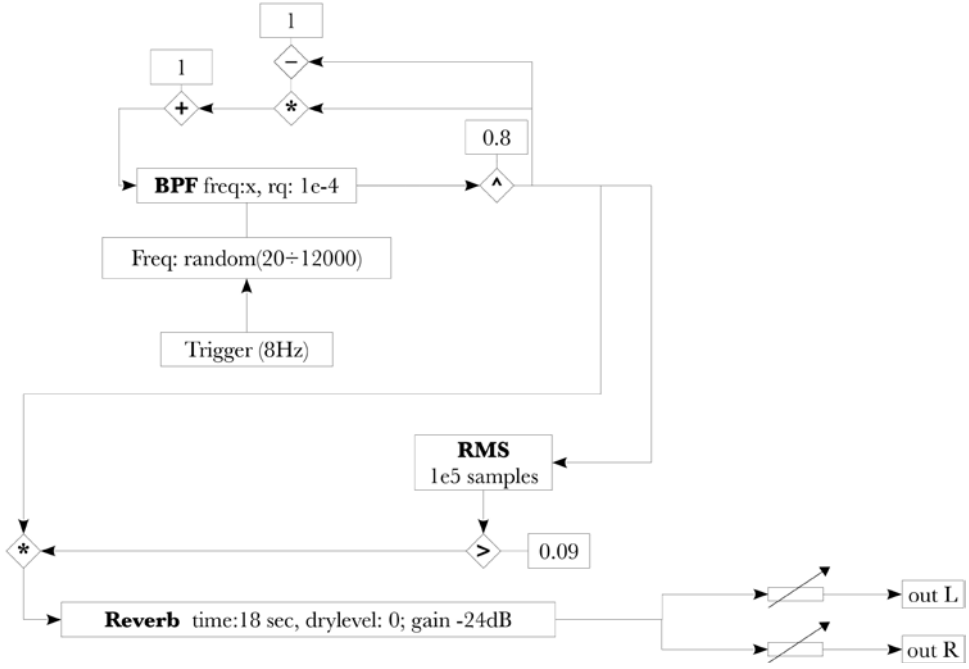
3.1.7 Diagramma 6



3.1.8 Diagramma 7



3.1.9 Diagramma 8



3.2 Implementazione degli algoritmi: esempio in SC3[14]

```
(
s=Server.local;
s.options.memSize=2**18;
s.boot;
p=ProxySpace.push(s);
)
p.fadeTime=0.24

////////////////////////// DIAGRAMMA 1 ////////////////////////////

~a.play;
~a={BPF.ar(~a.ar+700, 50, 1e-4)};

////////////////////////// DIAGRAMMA 2 ////////////////////////////

~b.play;
~b={BPF.ar(~b.ar+1e4, [50,55], 1e-4)**5};

////////////////////////// DIAGRAMMA 3 ////////////////////////////

~b={BPF.ar(~b.ar.reverse+1e4, [50,55], 1e-4)**5};

////////////////////////// DIAGRAMMA 4 ////////////////////////////

~b={BPF.ar(~b.ar.reverse*(1-~b.ar)+1e4, [50,55], 1e-4)**5};
~d={BPF.ar(~d.ar.reverse*(1-~d.ar)+0.5e3+(~b.ar*2e5), [600,1526], 1e-
4,0.07)**0.9};
~d.play;

////////////////////////// DIAGRAMMA 5 ////////////////////////////

~c={BPF.ar(~c.ar.reverse*(1-~c.ar)+0.5e4, {rrand(300,1100)}!2, 1e-
4)**[1,2]}; ~d={BPF.ar(~d.ar.reverse*(1-~d.ar)+0.5e3+(~c.ar*2e4),
{rrand(300,600)}!2, 1e-4,0.1)**0.85};

////////////////////////// DIAGRAMMA 6 ////////////////////////////

(
~e={BPF.ar(~e.ar*(1-~e.ar)+1, TRand.kr(1000,9000,~e>0), 1e-4)**0.8};
~f={(RunningSum.rms(~e.ar,2e3)<0.1)};
~g.ar(30); ~g={AllpassN.ar(~e.ar*~f,4,{4.0.rand}!30, 34)**0.95}; ~outg.
play;
~outg={Splay.ar(Mix(~g.ar),1,-28.dbamp)};
)
```

[14] Distribuzione standard di SuperCollider 3.8 (<http://supercollider.github.io>).

////////////////////////// DIAGRAMMA 7 ////////////////////////////

```
(
~h={BPF.ar(~h.ar*(1-~h.ar)+1e-4, TRand.ar(20,18000,~h>0), 1e-3)**0.7};
~i={(RunningSum.rms(~h.ar,2e3)<0.25)};
~l.ar(30); ~l={DelayN.ar(Mix(~h.ar*~i)+LPF.ar(~l.ar.reverse,8141,0.99),
0.2, {0.01.rand}!30)}; ~outl.play;
~outl={Splay.ar(Mix(~l.ar),1,-12.dbamp)};
)
```

////////////////////////// DIAGRAMMA 8 ////////////////////////////

```
(
~m={BPF.ar(~m.ar+1e-1, LFNoise0.ar(8).range(20,12000), 2e-4)**0.6};
~n={(RunningSum.rms(Mix(~m.ar),1e5)<0.09)}; ~o={GVerb.ar(Mix(~m.
ar*~n),50,18,drylevel:0,mul:-18.dbamp)};
~o.play;
)

p.free(30)
```

3.3 Distribuzione degli eventi

Viene qui indicata a puro titolo esemplificativo una ipotetica distribuzione temporale degli eventi generati dagli algoritmi descritti dagli otto diagrammi (cfr. 3.1).

Sono ammesse altre distribuzioni.

La densità degli eventi qui proposta si riferisce ad una durata approssimativa di 11 minuti circa.

La durata proposta è puramente indicativa: è possibile discostarsi da essa anche significativamente.

La densità degli eventi deve comunque ispirarsi ad un principio di estrema rarefazione. È da ricercare una densità minima: il minimo indispensabile per garantire coesione formale.

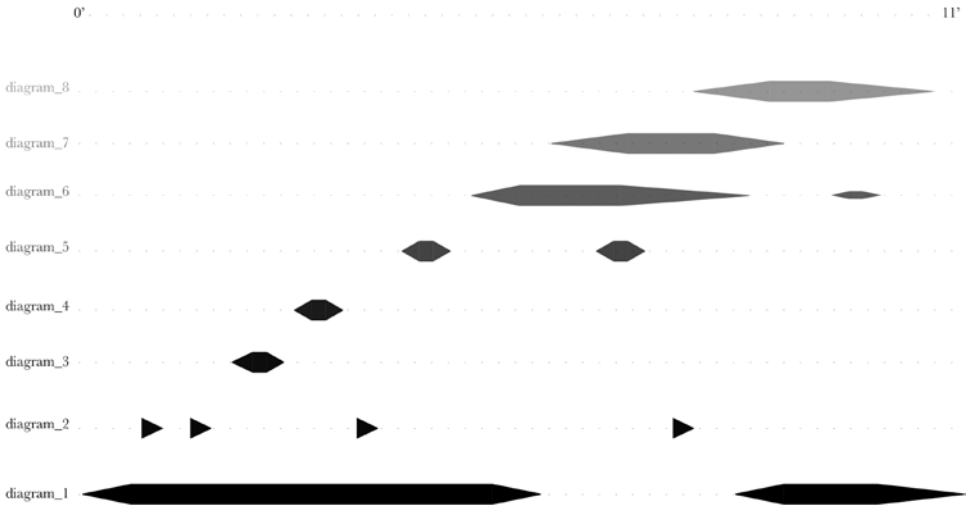
3.3.1 legenda

 = fade in

 = suono tenuto

 = fade out (se non preceduto da suono tenuto: attacco percussivo e fade out)

3.3.2 score


0'
11'
diagram_8
diagram_7
diagram_6
diagram_5
diagram_4
diagram_3
diagram_2
diagram_1