

# Composição em Tempo Real

Carlos Guedes

## 1. Introdução

Robert Rowe (1993) define sistemas musicais interactivos como sistemas computacionais capazes de modificar o seu comportamento na presença de entrada musical. Ainda que possa haver grandes distinções qualitativas sobre a forma como estes sistemas processam e respondem à entrada da informação, uma característica comum a todo e qualquer sistema musical interactivo é que a sua resposta é dada em tempo real. As aplicações destes sistemas na prática musical são hoje bastante diversificadas, como por exemplo: em instalações interactivas que respondem musicalmente à presença e acções de humanos; em *performances* de música em que o computador é utilizado como parceiro musical, quer como improvisador quer como acompanhador; em *software* orientado para a formação musical capaz de corrigir um aluno enquanto realiza uma progressão de acordes ao teclado; e obviamente, na composição musical.

Hoje em dia, os sistemas musicais interactivos permitem a um compositor trabalhar de uma forma diferente daquela utilizando os processos tradicionais. No trabalho com sistemas interactivos, um compositor pode programar algoritmos que produzem um determinado resultado musical e alterar esse resultado durante a sua execução em tempo real. Pode, por isso, ir *refinando* o resultado musical que está a ser produzido. O trabalho com sistemas interactivos permite a um compositor trabalhar a um nível *metacomposicional*, ou seja, a *compor com processos composticionais* decorrentes. Não sendo estas formas de trabalho inteiramente novas em relação a processos já utilizados em certa música experimental nos anos 60 e seguintes, os sistemas musicais interactivos facilitam a sistematização de um certo tipo de pensamento compostacional.

### 1.1. Organização do texto

Este texto que serve de apoio à lição sobre composição em tempo real está dividido em quatro partes. A primeira é constituída por esta introdução.

A segunda parte é uma discussão relativamente extensa de carácter teórico e conceptual, onde se apresentam selectivamente desenvolvimentos históricos e abordagens conceptuais que informam a formulação do conceito de composição em tempo real. A definição do conceito é apresentada nesta secção como um *work-in-progress*. Nesta secção, são também apresentados trabalhos de outras pessoas que representam exemplos das várias formas de prática da composição em tempo real actualmente. Esta apresentação não é exaustiva nem crítica, pois isso seria, nos dias de hoje, motivo de escrita de um livro provavelmente.

Na terceira parte do texto, apresenta-se uma pequena demonstração de carácter prático, que será realizada na lição, sobre as possibilidades oferecidas por certos ambientes de programação (notavelmente o ambiente Max/MSP) na criação de sistemas musicais interactivos que permitem a prática da composição em tempo real.

Finalmente na quarta parte do texto, apresenta-se uma conclusão sobre o trabalho realizado.

## 2. Discussão de carácter teórico e conceptual

### 2.1. Composição algorítmica

A utilização de algoritmos — conjuntos de instruções precisas, ordenadas no tempo, que produzem um resultado determinado quando executadas correctamente — é um processo bastante antigo de composição musical. À volta de 1026, Guido d'Arezzo desenvolveu uma técnica formal de compor melodias para textos. O seu esquema atribuía uma altura para cada vogal, e a melodia variava de acordo com o conteúdo de vogais do texto (Rowe, 1993). Outro exemplo de utilização de algoritmos em composição é as técnicas isorrítmicas de Guillaume Machaut para composição de motetes (Roads, 1996).

Outros processos ainda, como rondós, hoquetos, canons, fugas e variações na música tradicional são processos formalizáveis algorítmicamente (Roads, 1996). O compositor David Cope, no seu projecto *Experiments in Musical Intelligence* (EMI), tem apresentado desde 1987 modelizações algorítmicas da música de Bach, Brahms, Beethoven e Bartók assim como de outros compositores. Neste projecto, um computador *recombina*<sup>1</sup> música destes compositores para criar novas versões, estilisticamente correctas, da sua música (Cope, 2005).

O processo de recombinação não é inteiramente novo, e já Haydn e Mozart trabalharam juntos para gerar música utilizando este princípio (DeMarco, 1998), sendo o exemplo mais conhecido deste trabalho, a peça *Musicalisches Würfelspiel* de Mozart (KV. 516f), um jogo de dados para criar minuetos a partir de compassos escritos previamente.<sup>2</sup> As sequências de compassos são determinadas pelos valores obtidos no lançamento dos dados, que servem de índices de procura em tabelas contendo os números de compasso em quatro páginas de fragmentos musicais. No século XVIII, muitas composições utilizando dados eram criadas como jogos de salão permitindo a pessoas sem conhecimento de música comporem uma variedade de peças em estilo de “faça você mesmo”<sup>3</sup> (Russcol, 1972). O que é interessante, é que este processo incorporava um elemento de acaso, uma característica de muitos algoritmos computacionais dos dias de hoje (Roads, 1996).

A utilização do acaso em composição musical é um aspecto que tem sido utilizado em música desde a idade média (Morgan, 1991). No século XX, é notável a exploração

<sup>1</sup> Cope (1996) define recombinação como um método para produzir nova música recombinando fragmentos de música existente de uma forma lógica.

<sup>2</sup> Para uma versão *online* desta peça, onde se podem gerar vários minuetos que são executados por um sintetizador MIDI ir a <http://sunsite.univie.ac.at/Mozart/dice/> (acedido a 24/3/2008).

<sup>3</sup> Outros exemplos famosos da época são também *O sempre-pronto compositor de Polonaises e Minuetos* de Johan Kirnberger (1<sup>a</sup> edição, 1757; 2<sup>a</sup> edição revista, 1783) e a *Anedota Filarmónica* de Joseph Haydn (1790) (Wikipedia, *Musikalisches Würfspiel*, s.d.). Não deixa de ser interessante notar como esta prática do “faça você mesmo a sua peça musical” é hoje em dia espelhada na míriade de programas de computador existentes, de acesso fácil e generalizado (e.g. *GarageBand* da Apple, aplicação distribuída com as versões mais recentes do sistema operativo MacOs X), ou em instalações de carácter lúdico utilizando sistemas digitais interactivos que produzem música em tempo real de acordo com a interacção entre o público e o sistema de geração musical.

do elemento do acaso na música de alguns compositores do período pós-guerra de 1945, tais como John Cage, Morton Feldman, Earle Brown, Pierre Boulez e Karlheinz Stockhausen (e.g., cf. Morgan, 1991, pp. 358-379).

## 2.2. Composição algorítmica por computador

A história, estética e principais técnicas da composição algorítmica estão hoje em dia bastante bem documentadas. Roads (1996, Capítulos 18 e 19), Dodge e Jerse (1985), e Chadabe (1997), constituem bons exemplos a consultar por um leitor mais curioso sobre este campo. Um sítio de consulta indispensável *on-line*, é a página do compositor Christopher Ariza (*Algorithmic.net: Algorithmic composition resources | Main*, s.d.) que tem uma coleção extensíssima de fontes bibliográficas (algumas com *links* para consulta), assim como referências a muitos programas de computador utilizados neste campo, fornecendo, quando possível, *links* para esses programas. Como mencionado na introdução a este texto, referem-se aqui somente aspectos de carácter histórico que suportam a presente formulação do conceito de composição em tempo real.

### 2.2.1. Lejaren Hiller

Lejaren Hiller (1924-1994) foi das primeiras pessoas a utilizar computadores na composição algorítmica. Entre 1955 e 1957, juntamente com Leonard Isaacson, criou a *Illiad Suite for String Quartet*, uma peça hoje considerada um marco na história da música (Roads, 1996). Para além de ter sido a primeira peça composta inteiramente aplicando processos formais em computador, a *Illiad Suite*, define aquela que viria a ser a tendência mais prolífica de exploração na música gerada computador: a utilização de algoritmos estocásticos na geração musical.

Nos três primeiros andamentos (“Experiências” I, II, e III). Hiller e Isaacson utilizaram geração de números aleatórios correspondendo a notas musicais que posteriormente eram confrontados com um sistema de regras que os aceitava ou não dependendo do facto de eles estarem em conformidade com um estilo composicional preexistente. Este método, designado por *generate and test* (Dodge e Jerse, 1985), é aplicado nos dois primeiros andamentos para gerar música de acordo com as regras para realizar *cantus firmus* e contraponto de espécies descritas por Johann Joseph Fux em *Gradus ad Parnassum* (1725/1965). No terceiro andamento, o mesmo método é aplicado para lidar com aspectos rítmicos, de dinâmica e instruções de *performance* numa textura musical cromática livremente dissonante. Na primeira secção, as escolhas de alturas são fixas, seguidamente, escolhidas aleatoriamente da escala cromática, posteriormente são controladas por um sistema simples de regras compostionais, e finalmente por processos dodecafónicos elementares (Hiller, 1989).

No quarto andamento (“Experiência IV”), Hiller e Isaacson aplicam regras que controlam a probabilidade de ocorrência de um evento musical que são ainda mais afastadas dos critérios estilísticos da composição tradicional (Dodge e Jerse, 1985). Os

compositores empregaram cadeias de Markov<sup>4</sup> para criar “sequências de eventos em que a escolha do novo evento pode ser dependente dos eventos precedentes; ou, em [falando] termos musicais, a escolha de cada nova nota ou intervalo numa linha melódica pode ser dependente nas notas ou intervalos prévios na mesma linha melódica”<sup>5</sup> (Hiller e Isaacson, 1959; citado em Dodge e Jerse, 1985, pp. 292-293). De acordo com Hiller (1989), embora o método estocástico empregue no quarto andamento seja o método “mais difícil de perceber nestas experiências, também pareceu ser o método mais lógico para desenvolvimentos futuros na composição assistida por computador. É a técnica composicional mais adequada e mais dependente no processamento matemático de alta velocidade” (p. 10).

Outros compositores rapidamente seguiram as pegadas de Hiller na aventura da música por computador. Herbert Brün e John Myhill, James Tenney, Pierre Barbaud, Michel Philipot, Iannis Xenakis, e Gottfried Michael Koenig são outros dos pioneiros no campo. Embora os seus trabalhos em música por computador fossem posteriores aos de Hiller e Isaacson, alguns deles, já tinham começado a compor utilizando processos formais há algum tempo. Por exemplo, *Metastasis* para orquestra, de Xenakis, uma peça composta utilizando fórmulas estocásticas calculadas “à mão” foi estreada em 1955 (Roads, 1996). Aliás, o trabalho de Xenakis utilizando processos estocásticos (equações de Poisson e Cadeias de Markov) e teoria dos jogos, teria de esperar até 1962 para começar a ser realizado em computador (Xenakis, 1992).

### **2.2.2. Iannis Xenakis**

Durante o ano de 1962, Iannis Xenakis (1922-2001) teve acesso ao computador IBM 7090 da IBM-France onde produziu uma série de peças, que à exceção de *Morisma-Amorisima* (1962) têm a indicação de ST. Estas peças utilizaram o algoritmo ST, um algoritmo estocástico escrito em FORTRAN. Este algoritmo é uma elaboração

<sup>4</sup> As cadeias de Markov são dos processos mais antigos de composição algorítmica por computador. Inicialmente formulada pelo matemático Andrey Markov, uma cadeia de Markov é um sistema de probabilidades, em que a probabilidade de um evento futuro é determinada pelo estado de um ou mais eventos no passado imediato (Roads, 1996). O número de eventos passados que contam para a determinação do evento futuro (a “memória” da cadeia) é chamada a *ordem* da cadeia — e.g. numa cadeia de Markov de ordem 1, o evento seguinte depende do anterior, numa cadeia de ordem 2, nos dois eventos anteriores, etc. Numa cadeia de Markov de ordem 0, nenhum evento precedente influencia a determinação do evento seguinte.

<sup>5</sup> Tradução do Autor, daqui para a frente indicado simplesmente como T. A.

das “fases fundamentais de uma obra musical”<sup>6</sup> utilizada para a composição de *Achorripsis* (1957) (Harley, 2004).

As funções probabilísticas são implementando várias limitações que tinham em conta as particularidades das fases do cálculo (instrumentação, âmbito, dinâmicas, etc.) Uma vez que o programa era compilado no computador, os resultados numéricos tinham que ser transcritos para música, permitindo a Xenakis fazer o seu próprio julgamento sobre os resultados, alterando pormenores ou mesmo reordenando os eventos como ele achava necessário (Harley, 2004).

O algoritmo ST permitiu a geração de várias peças através da alteração dos dados de entrada: *ST/48-1* para orquestra; *ST/10-1* para ensemble; *Amorsima-Morsima (ST/10-2)* para ensemble (complemento de *ST/10-1*); *ST/4-1* para quarteto de cordas; *Morsima-Amorsima* para quarteto misto; e *Atréees (ST/10-3)* para ensemble. Xenakis comparava o algoritmo ST a formas como a fuga, que consiste em conjuntos de regras que dão origem a um número de composições. Contudo, nem todas as fugas são música interessante, assim como não o seria a aplicação automática do algoritmo (Harley, 2004). Nas conclusões ao trabalho efectuado nas peças da série *ST*, Xenakis (1992) descreve as vantagens do uso de computadores na composição musical:

1. Os cálculos longos e laboriosos feitos à mão são reduzidos a nada. A velocidade de uma máquina como o IBM-7090 é tremenda — na ordem das 500.000 operações elementares por segundo.
2. Livre de cálculos entediantes, o compositor pode-se dedicar aos problemas gerais que a nova forma musical implica e a todos os aspectos desta forma ao modificar os valores de entrada. Por exemplo, ele pode testar todas as combinações instrumentais desde solistas, passando por orquestras de câmara, até grandes orquestras. Com a ajuda dos computadores, o compositor torna-se uma espécie de piloto: pressiona botões, introduz coordenadas, e supervisiona os

<sup>6</sup> De acordo com Xenakis (1992) estas fases são:

1. *Concepções iniciais ...;* 2. *Definição das entidades sonoras* e do seu simbolismo comunicável dentro dos limites de meios possíveis (sons dos instrumentos musicais, electrónicos, ruídos, conjuntos de eventos sonoros ordenados, formações contínuas ou granulares, etc.); 3. *Definição das transformações* que estas entidades sonoras irão sofrer no decurso da composição ...[de acordo com as definições em 2.]; 4- *Microcomposição* (escolha e fixação detalhada das relações funcionais ou estocásticas dos elementos de 2. ...; 5. *Programação sequencial* de 3. e 4. (o esquema e *padrão integral* da obra; 6. *Implementação dos cálculos*, verificações, *feedbacks*, e modificações definitivas [de 5.]; 7. *Resultado simbólico final* da programação (escrever a música em papel usando notação tradicional, expressões numéricas, gráficos, ou outras formas de solfejo); 8. *Realização sonora* do programa (execução orquestral directa, manipulações do tipo da música electromagnética, construção computadorizada das entidades sonoras e suas transformações) (p. 22, T. A.).

Para Xenakis, a ordem da lista não é rígida e permutações entre os vários passos são possíveis no decorrer da composição da peça.

controlos de um navio cósmico que veleja no espaço sonoro, através de constelações e galáxias que dantes só conseguia ver como um sonho distante. Agora ele pode explorá-las sem dificuldade sentado numa cadeira.

3. O programa, i.e., a lista de operações sequenciais que constituem a nova forma musical é uma manifestação objectiva desta forma. O programa pode ser consequentemente enviado para qualquer ponto do planeta que tenha computadores que o consigam ler, e pode ser explorado por qualquer compositor piloto.
4. Por causa do facto de que certas incertezas podem ser introduzidas no programa, o compositor-piloto pode afirmar a sua personalidade no resultado sonoro a obter.<sup>7</sup> (p. 144)

As conclusões de Xenakis acima transcritas são fulcrais para a presente discussão sobre composição em tempo real — principalmente os conceitos de “compositor-piloto,” e de música diversa que pode ser criada através do mesmo algoritmo estocástico que produz resultados diferentes, ainda que semelhantes pois são gerados pelo mesmo processo formal. Além disso, todo seu trabalho teve um impacto tremendo no desenvolvimento da música por computador, composição assistida por computador, novas formas de síntese sonora, assim como formas radicalmente diferentes de pensar a criação musical. Este aspecto é hoje reconhecido inquestionavelmente pela comunidade musical e os pressupostos avançados por ele ainda inspiram nova pesquisa em composição musical (cf. Roads, 2001).

### **2.2.3. Clarence Barlow**

Clarence Barlow tem produzido trabalho muito interessante na composição algorítmica por computador. Contrariamente a Xenakis ou Hiller que inicialmente aplicaram fórmulas gerais da teoria da estatística (e.g. distribuição de Poisson e cadeias de Markov), Barlow (n. 1947) desde muito cedo se preocupou na criação de algoritmos e fórmulas matemáticas que tivessem em conta princípios musicais gerais como harmonia e métrica. *Çoğluotobüsüşletmesi* (1978) é uma peça para piano de trinta minutos que foi possível através de uma extensa pesquisa que este compositor realizou nos campos da tonalidade e métrica (Barlow, 1980). Em *Çoğluotobüsüşletmesi*, Barlow formulou uma abordagem quantitativa à harmonicidade, coerência métrica, assim como encontrou princípios algorítmicos que calculam a dinâmica do ataque num ponto de um dado compasso (Barlow, 1980, 2005; Roads, 1996). Estes algoritmos, traduzidos em fórmulas

---

<sup>7</sup> T. A.

matemáticas de alguma complexidade,<sup>8</sup> que permitem a geração de estruturas rítmicas e de alturas com bastante coerência musical, estão na base de *AUTOBUSK* (Barlow, 2001) um programa desenvolvido para o computador Atari entre 1987 e 2000.

*AUTOBUSK* usa os conceitos de estabilidade, clareza, consonância e dissonância para afectar o comportamento rítmico e microtonal de texturas musicais. É um método generativo que usa material de base providenciado pelo utilizador para desenvolver a música de acordo com os princípios algorítmicos do programa (Rowe, 1993).

#### 2.2.4. Sever Tipei

Em 1989, na sua pesquisa em composição algorítmica por computador, compositor Sever Tipei definiu o conceito de *Manifold Compositions*:

O termo Manifold Composition é proposto para definir todas as variantes actuais e potenciais de uma peça gerada por computador que:

- a) corre um programa comprehensivo contendo elementos aleatórios e
- b) lê essencialmente os mesmos dados para cada variante

Uma vez que cada membro do Manifold (variante da peça) tem um carácter único e uma vez que, teoricamente, um número ilimitado [de variantes] pode ser criado, uma variante não pode ser apresentada em público mais do que uma vez.

Um programa de computador é comprehensivo se, uma vez executado, não requer a intervenção do compositor para produzir a saída final. (Tipei, 1989, p. 324)

<sup>8</sup> Abaixo, reproduz-se a fórmula para o cálculo da *indispensabilidade* (peso) de uma pulsação  $n$  pertencente a uma métrica cuja estratificação é expressa como produto de números primos (Barlow, 2005. Reproduzido com autorização). E.g., se quisermos calcular o peso de cada colcheia num compasso 3/4, os níveis de estratificação são 3x2 (correspondendo a 3 semínimas que se subdividem em 2 colcheias cada), o peso de cada colcheia nesta métrica calculado pela fórmula é respectivamente 5 0 3 1 4 2, o que acentua, por esta ordem o 1º tempo, 3º tempo, 2º, etc. Contudo, se quisermos calcular o peso de cada colcheia numa métrica de 6/8 (expressa como 2x3, 2 semínimas com ponto que se subdividem em três colcheias) o peso calculado pela fórmula é 5 0 2 4 1 3. Estes resultados são de uma grande coerência musical.

Formula for the Indispensability  $\Psi$  of the  $n^{\text{th}}$  Pulse of a Metre of Stratification  $p_1 \times p_2 \times p_3 \times \dots \times p_z$

$$\Psi_z(n) = \sum_{r=0}^{z-1} \left\{ \prod_{i=0}^{z-r-1} p_i \Psi_{p_{z-r}} \left( 1 + \left[ 1 + \frac{(n-2) \bmod \prod_{j=1}^z p_j}{\prod_{k=0}^r p_{z+1-k}} \right] \bmod p_{z-r} \right) \right\}$$

whereby (all variables being whole numbers):

1.  $p_0 = p_{z+1} = 1$
2.  $n$  is the position in the bar of the pulse in question, starting at 1
3.  $p_j$  is the stratification divisor on level  $j$
4.  $z$  is the number of levels in the stratification
5.  $\Psi_p(x)$  is the Indispensability of the  $x^{\text{th}}$  pulse of a first-order bar with the prime stratification  $p$
6.  $u \bmod v$  is the remainder of the division  $(u+mv)/v$ , by sufficiently large  $m$  never negative
7.  $[x]$  is the whole-number component of  $x$

Para produzir um *manifold*, o programa de computador tem que conter elementos estocásticos, ou características da música aleatória, que são responsáveis por pequenas modificações dados, tais como mudar a semente (*seed*) do gerador pseudo-aleatório de variante em variante para obter resultados diferentes. O *manifold*, permite produzir em massa os seus membros (as variantes), pois basta alterar a semente dos geradores aleatórios para produzir uma variante nova. A preferência de Tipei em não apresentar a mesma variante duas vezes, pretende enfatizar a natureza efêmera do contacto entre públicos e qualquer obra de arte temporal. “[Os] Manifolds tornam-se então colecções de peças “de usar e deitar fora,” só que, em vez de adquirirem uma qualidade pejorativa, reforçam a ideia de realidades alternativas ou paralelas” (Tipei, 1989, p. 325).

Tipei afirma que as abordagens anteriores para produzir variantes de uma peça a partir do mesmo algoritmo computacional (a série ST de Xenakis acima mencionada, *Algorithms* de Hiller, *Segmente* de Koenig, *Protoforms: Fractals for Computer Band* de Larry Austin) são diferentes da abordagem do *manifold*, pois nunca tentaram produzir em massa as variantes e também porque nem sempre os dados de entrada eram os mesmos. O conceito de *manifold composition* pode ser visto como um possível refinamento do trabalho de Xenakis na série ST e informa o conceito apresentado em composição em tempo real.

### **2.2.5. Três (ou quatro) programas pioneiros de composição musical**

Curtis Roads (1996) menciona que dos muitos dos programas de composição algorítmica que foram escritos na década de 60 só muito poucos é que foram utilizados por mais de um compositor. Na secção “Three Pioneering Composition Programs” de *Computer Music Tutorial* (pp. 836-844) ele descreve três programas que serão abaixo sumariados — *Programa de Música Estocástica* de Xenakis (SMP),<sup>9</sup> *Project 1* de Gottfried Michael Koenig e *POD* de Barry Truax. Estes programas representam a primeira geração de programas de composição automática escritos por compositores programadores que foram utilizados por outros. O SMP e o Project 1 foram escritos nos anos 60 e o POD no início da década de 70. Não sendo estritamente um programa de composição algorítmica e originalmente criado para síntese sonora, decidiu-se incluir o *Music 4* de Max Mathews por ser o programa mais importante das linguagens de síntese Music-n (Boulanger, 2000) e que inspira a criação do *Csound* por Barry Vercoe — um dos programas mais utilizados actualmente no campo da música por computador e com a genealogia mais ancestral.

#### *SMP*

O Programa de Música Estocástica (SMP) modela uma composição como uma sequência de secções caracterizadas por uma duração e densidade de notas nessa duração. O compositor interage com o SMP, estipulando atributos globais da partitura e executando o programa de seguida. Os atributos incluem: 1- duração média das secções;

---

<sup>9</sup> Apresenta-se a sigla mais comum deste programa (de *Stochastic Music Program*) por ser a utilizada nos livros de referência de música por computador (e.g. Dodge e Jerse, 1985 e Roads, 1996)

2- densidade mínima e máxima de notas numa secção; 3- classificação dos instrumentos em classes de timbres; 4- distribuição das classes de timbres como função da densidade; 5- probabilidade de cada instrumento pertencente a uma classe de timbres tocar; 6- duração mais longa executável por um instrumento. O SMP auxiliou Xenakis na composição de várias obras entre as quais *Eonta*(1963) e foi melhorado em 1979 por John Myhill estando desde então disponível para computadores pessoais (Roads, 1996).

### *Project 1*

O programa Project 1 de Gottfried Michael Koenig foi criado em 1964 com o intuito inicial de testar as regras composicionais da música serial, mas desde cedo ficou claro que as regras baseadas em listas de parâmetros e permutações de séries não podiam ser descritas sem planos concretos para uma composição (*Gottfried Michael Koenig*, s.d.).

O modelo em que o programa é baseado é o do princípio de alternância entre um par de situações opostas “regular/irregular” (princípio RI) inspirado na não-repetição de elementos seriais, permitindo sete princípios de selecção a cinco parâmetros musicais (instrumento, ritmo, harmonia, registo e dinâmica) que vão desde selecções completamente aleatórias (não-repetitivas) até selecções completamente determinísticas (repetitivas).

Ao usar Project 1, o compositor introduz os seguintes dados no programa: 1- um conjunto de pesos para acordes de vários tamanhos; 2- o número total de eventos a ser gerado; 3- um conjunto de tempos; e 4- um número que serve de semente a um gerador pseudo-aleatório para os processos estocásticos. O programa gera então secções (estruturas) em que cada um dos cinco parâmetros passa pelos sete princípios de selecção aleatoriamente. Os resultados são impressos sob a forma de uma lista de notas que podem ser transcritos para notação musical, para um sistema de síntese em tempo real que havia no Instituto de Sonologia, ou (mais recentemente) para MIDI (Roads, 1996). O programa Project 2 (1966) estende o trabalho de Project 1 com o intuito de dar ao compositor uma maior flexibilidade no processo composicional.

### *POD*

Enquanto SMP e Project 1 geravam “partituras” para serem transcritas para instrumentos tradicionais, a série de programas POD (*POisson Distribution*) de Barry Truax desenvolvida entre 1972 e 1999 foi expressamente escrita para a síntese de som digital. Originalmente criando síntese monofônica em tempo real implementando formas de onda fixas, modulação de amplitude e frequência, o programa foi-se desenvolvendo, implementando polifonia, controlo em tempo real de síntese granular (1986-87) e difusão sonora multicanal controlada por computador (Truax, *POD & PODX System chronology*, s.d.).

O POD gera eventos dentro dos limites de *máscaras tendenciais* (regiões de frequência vs. tempo) e a distribuição dos eventos no tempo e frequência segue a distribuição de Poisson (uma distribuição de probabilidade discreta). O compositor especifica o número de eventos dentro de cada máscara tendencial ajustando o parâmetro

de densidade de eventos. Além dos objectos sonoros, máscaras tendenciais, e densidades o compositor também estipula os princípios de selecção (semelhantes aos de Project 1 de Koenig) a serem empregues na escolha de objectos sonoros a serem postos na máscara tendencial. Uma vez que a partitura de síntese é gerada, os utilizadores podem variar a execução da partitura de várias formas alterando o tempo, direcção, grau de sobreposição de eventos, envolvente e outros parâmetros de síntese (Roads, 1996).

#### *Music 4*

“A síntese sonora por computador nasceu em 1957 quando um IBM704, em Nova York, lentamente criou uma ‘peça’ de 17 segundos composta naquilo que provavelmente foi o primeiro exemplo audível de uma nova escala no meu programa Music 1”<sup>10</sup> (Mathews, 2000, p. xxxi). O programa Music 4 para síntese sonora, um sucessor do Music 1, começou a ser distribuído em algumas universidades americanas no início dos anos 60 onde foi subsequentemente desenvolvido (Dodge e Jerse, 1985). O Music 5, a última versão do programa criado por Max Mathews e a sua equipa nos Bell Laboratories, conheceu um grande sucesso em parte por causa da publicação do livro *The Tecnhology of Computer Music* (Mathews, Miller, Pierce e Risset, 1969) onde a linguagem era explicada em detalhe.

Nos programas Music 4, o compositor cria uma “orquestra,” definindo instrumentos de acordo com a sintaxe da linguagem. Cada instrumento é desenhado para produzir um determinado som e tem parâmetros de entrada que controlam várias das suas características (duração, intensidade, altura, etc.) Para além disso, o compositor cria uma “partitura” que define a *forma* como os instrumentos vão ser tocados — i.e. quando vão ser tocados e quais os parâmetros que vão ser executados/variados. Depois de compilado, o programa “executa” então a partitura, tocando os instrumentos definidos na orquestra (Dodge e Jerse, 1985).

Esta organização de uma linguagem de síntese prevaleceu naquilo que hoje em dia se designa por programas da classe Music-n, que após a distribuição do Music 4 foram desenvolvidos por outros compositores-programadores, entre os quais Barry Vercoe que escreveu os programas Music 360 e Music 11 (1973). Em 1985, Barry Vercoe desenvolve o *Csound*, inteiramente escrito na linguagem de programação C, que continua a ser ainda hoje desenvolvido por uma grande comunidade de programadores, sendo talvez a linguagem de síntese sonora mais conhecida e utilizada, e que aplica os conceitos de “orquestra” e “partitura” que controla os parâmetros da orquestra gerando um ficheiro de som com a “execução” da “partitura” após compilado. A própria sintaxe do Csound é reminiscente da sintaxe do Music 11 (cf. Dodge e Jerse, 1985, Example 1.1., p. 15).

#### **2.2.6. Alguns exemplos actuais de programas de composição algorítmica**

*AC Toolbox* (Berg, 2007), *Open Music* (Assayag, Agon, et al., 2008), *Common Music* (Taube, 2007), e *PWGL* (Laurson, Kuuskankare e Norilo, 2008), constituem alguns dos programas que hoje em dia se podem utilizar em composição algorítmica. Todos

---

<sup>10</sup> T. A.

estes programas são codificados em LISP, uma linguagem muito popular em composição algorítmica. Todos eles são programas de utilização livre e permitem a um utilizador calcular fragmentos ou secções de uma peça utilizando algoritmos. Open Music e PWGL possuem um ambiente gráfico de programação em um utilizador liga objectos gráficos entre si, Common Music, utiliza um interpretador LISP que executa comandos de texto, e o *interface* do utilizador de AC Toolbox é mostrado na Figura 1. Outra característica comum a todos estes programas é que não permitem a modificação dos algoritmos enquanto correm, distinguindo-se por isso dos ambientes de programação interactivos.

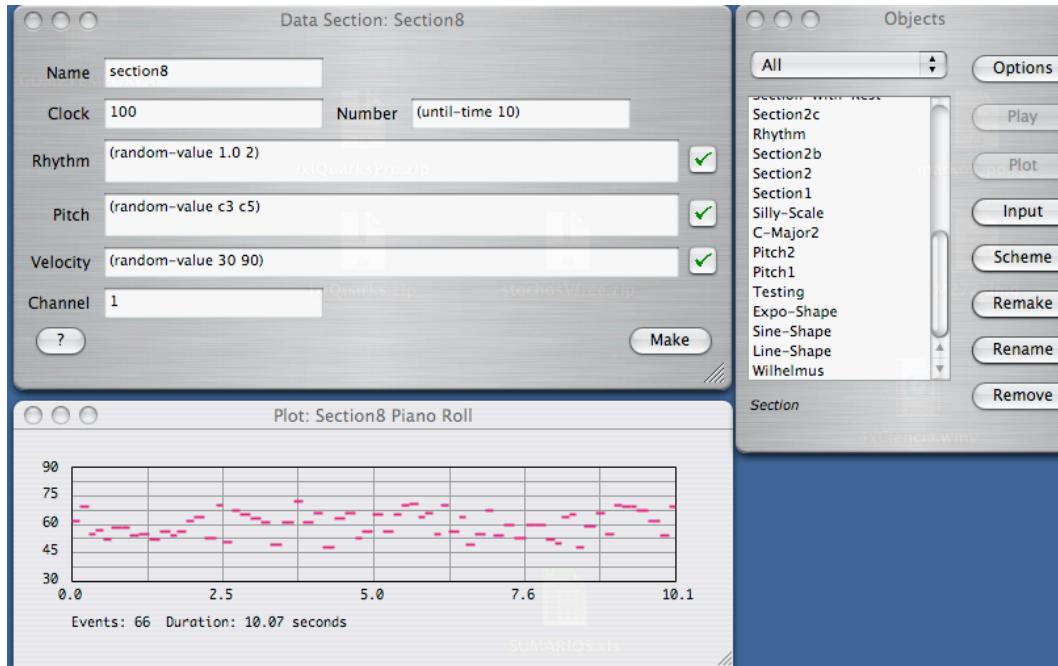


Figura 1. *Interface* do utilizador de AC Toolbox. Na imagem, uma secção de 10 segundos de música é calculada, contendo valores de alturas aleatórias entre Dó3 e Dó5 com durações entre 100 e 200 ms.

### 2.3. Sistemas musicais interactivos

Os sistemas musicais interactivos facultam novas formas de pensamento composicional (Rowe, 1993, 2001; Winkler, 1998). A música interactiva é um ramo da composição algorítmica por computador. Só que, ao contrário dos sistemas que foram descritos acima, os sistemas musicais interactivos permitem modificar o comportamento dos algoritmos durante a sua execução. Quando fazem parte de uma *performance* musical eles podem modificar o seu comportamento em função do que se passa à volta deles.

Para Robert Rowe (2001) esta versatilidade representa a essência da interacção e uma expansão fascinante da arte de composição. Outro aspecto motivante no uso de sistemas musicais interactivos é que eles requerem a participação dos humanos para fazer a música funcionar. Se os sistemas oferecem uma resposta musical suficientemente interessante como parceiros, eles podem encorajar pessoas a fazerem música a qualquer nível desejado. “Eu creio que um aspecto crítico para a viabilidade e vitalidade da música na nossa cultura é o facto de um número significativo de pessoas continuarem (ou

começarem) a praticar música activamente, em vez de simplesmente absorverem a música que é bombardeada por altifalantes em todo o lado”<sup>11</sup> (p. 4).

Tal como a história da composição algorítmica por computador, a história dos sistemas musicais interactivos está devidamente documentada (ver por exemplo, Rowe, 1993; Roads, 1996; Chadabe, 1997; ou Winkler, 1998;). Neste texto, apresentam-se apenas aspectos históricos relevantes à presente discussão sobre composição em tempo real.

### **2.3.1. Alguns aspectos históricos importantes no desenvolvimento de sistemas musicais interactivos**

#### *Primeiros desenvolvimentos*

O trabalho de John Cage do fim dos anos 30 e princípios dos anos 40 inspirou um número de compositores a experimentar com música electrónica “ao vivo” (*live-electronics*) nos anos 60 (entre os quais o próprio Cage). O seu interesse em improvisação e indeterminação naturalmente levou ao aparecimento dos primeiros sistemas electrónicos interactivos. Em meados dos anos 60 o aparecimento dos sintetizadores utilizando controlo de voltagem<sup>12</sup> abriu caminho para as técnicas da música interactiva (Winkler, 1998).

Max Mathews foi também um pioneiro no desenvolvimento de sistemas interactivos musicais em computador. Entre 1968 e 1979, juntamente com F. Richard Moore, desenvolveu o sistema GROOVE nos Bell Laboratories que tinha um programa de direcção musical (*Conductor Program*) que permitia ao utilizador controlar o tempo, dinâmica e equilíbrio de um conjunto de computadores que executava uma partitura musical previamente definida. O *Conductor Program* era uma ferramenta de grande valor para investigar o potencial de interacção entre músicos e máquinas na evolução da *performance* musical em tempo real. Além de Mathews, os intérpretes Paul Zukofsky e Gerard Schwartz utilizaram o sistema para dirigir interpretações de música tradicional como um meio de investigar a *nuance* em *performance* de música por computador. O sistema também foi utilizado pelos compositores Emmanuel Ghent e Laurie Spiegel, entre outros, para criar composições musicais (Dodge e Jerse, 1985).

Em 1966, Joel Chadabe desenhou um plano para um sintetizador completamente automatizado, discutiu a sua construção com Robert Moog, e conseguiu arranjar financiamento para produzir o Sistema CEMS (*Coordinated Electronic Music Studio*) que foi instalado no estúdio da Universidade Estadual de Nova York em Albany no fim de 1969. Naquela que era a maior concentração mundial de sintetizadores Moog numa sala, Chadabe era capaz de programar um processo pseudo-aleatório suficientemente complexo para automatizar uma peça musical (Chadabe, 1997). “Era o equivalente em tempo real à composição algorítmica. Eu simplesmente ouvia enquanto o sistema estava a

<sup>11</sup> T. A.

<sup>12</sup> Controlo de voltagem é um sinal eléctrico que pode ser utilizado para automatizar um sintetizador analógico. Quase tudo que pode ser alterado num módulo de um sintetizador analógico pode ser externamente controlado variações de voltagem (Winkler, 1998).

produzir som e enquanto estava a decidir que sons produzir.”<sup>13</sup> (p. 286). Na execução de *Ideas of Movement at Bolton Landing* (1971) Chadabe utilizava joysticks para controlar filtros, moduladores, amplificadores e vários sequenciadores.

Em 1972, Salvatore Martinaro com a ajuda de Sergio Franco conclui o SalMar Construction um instrumento utilizando circuitos digitais de computadores com 291 interruptores que eram utilizados para controlar quatro programas que corriam separadamente. Os programas controlavam quatro vozes num de quatro sistemas de afinação diferentes. Alguns interruptores criavam alterações nos quatro programas, enquanto outros só afectavam mudanças num programa de acordo com o nível de controlo desejado por Martinaro (Chadabe, 1997). Para Chadabe:

O Sistema CEMS e o SalMar Construction foram os primeiros instrumentos de composição interactivos, o que quer dizer que eles tomavam decisões musicais, ou pelo menos *pareciam* tomar decisões musicais.... Estes instrumentos eram interactivos no sentido em que o intérprete e o instrumento se influenciavam mutuamente. O intérprete era influenciado pela música produzida pelo instrumento, e o instrumento era influenciado pelos controlos accionados pelo intérprete. Estes instrumentos introduziram o conceito de controlo partilhado e simbiótico de um processo musical onde a geração de ideias pelo instrumento e o julgamento musical do intérprete trabalhavam juntos para dar forma ao fluxo musical.

.... Quando um instrumento é configurado ou construído para tocar uma composição, sejam quais forem os pormenores que se alterem de *performance* em *performance* e quando essa música é composta enquanto está a ser executada, as distinções entre instrumento música, compositor e intérprete esbatem-se. O instrumento é a música. O compositor é o intérprete.<sup>14</sup> (Chadabe, 1997, p. 291)

O trabalho de Joel Chadabe é seminal na definição do conceito de *composição interactiva* (como ele o intitula), ou de *composição em tempo real*, o termo que eu prefiro empregar para designar este tipo de prática musical com sistemas musicais interactivos.

Em meados dos anos 70, no Mills College, a *League of Automatic Music Composers* e posteriormente *The Hub*, desenvolveram ideias de interactividade no contexto de computadores ligados em rede, desenvolvendo ideias dos compositores Jim Horton, John Bischoff e Rich Gold (Chadabe, 1997). A *League* dissolveu-se em 1983 e John Bischoff e Tim Perkins (que tinha feito parte da *League*) fundaram *The Hub* em 1985 com os mesmos propósitos. Como procedimento habitual cada membro do *Hub* criava uma peça que era tocada cooperativamente por todos. A forma de interacção no grupo era moldada no estilo de grupo de improvisação tradicional, só que utilizando computadores ligados em rede e interactivamente.

<sup>13</sup> T. A.

<sup>14</sup> T. A.

### M e Jam Factory

Em 1987, os programas *M*<sup>15</sup> (Cycling'74a, 2007; Figura 2) e *Jam Factory* são os primeiros programas de composição utilizando sistemas interactivos a serem comercializados por uma firma chamada *Intelligent Music* fundada por Joel Chadabe, onde David Zicarelli trabalhava como colaborador (Rowe, 1993; Chadabe, 1997; Winkler, 1998). A *Intelligent Music* tinha sido criada com o intuito de desenvolver e distribuir *software* para composição interactiva. Entre as grandes novidades implementadas por estes programas, contam-se controladores gráficos que permitem acesso às variáveis de controlo de parâmetros musicais, afectando em tempo real o conteúdo musical a ser produzido. O conteúdo musical é audível sob a forma de eventos MIDI. *M* é uma colecção de algoritmos que manipulam um *padrão* (uma colecção de notas) gravadas em MIDI por um compositor e posteriormente transformada pelos controladores gráficos. Uma combinação de transformações cíclicas, aleatórias, ou executadas manualmente podem ser aplicadas aos parâmetros musicais de altura, ritmo ou timbre (por mensagens MIDI de *program change*) (Chadabe, 1997). O *Jam Factory* possuía quatro “executantes” cujo material era gerado usando tabelas de transição características das cadeias de Markov. O programa mantinha várias tabelas de ordens diferentes e a parte essencial do algoritmo de *Jam Factory* era a decisão probabilística feita por uma nota sobre qual a tabela a utilizar. Diferentes tabelas eram usadas para alturas e durações (Rowe, 1993).

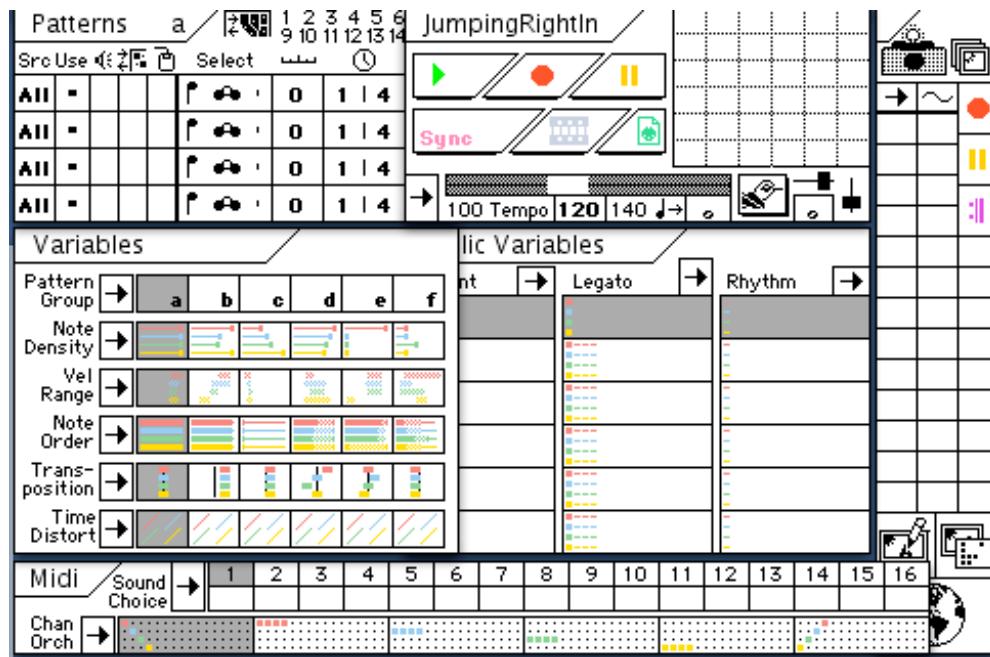


Figura 2. Imagem da versão actual de *M*.

<sup>15</sup> O *M* foi reeditado recentemente para o Mac OS X pela companhia Cycling'74.

### *Robert Rowe*

Em 1987, Robert Rowe começou a trabalhar em *Cypher*, um programa que era capaz de “ouvir” a entrada musical (Chadabe, 1997). A ideia de Rowe, era criar um instrumento interactivo que percebesse o que estava a “ouvir,” e produzir um resultado musical interessante baseado nessa audição. Ele definiu as funções básicas do programa como “ouvinte” (*Listener*) e “executante” (*Player*). “Ouvir” significa perceber a informação que entra no sistema sob a forma de mensagens MIDI, analisá-la, e enviar mensagens para o “executante” a comunicar o que estava a ouvir. As mensagens MIDI para o ouvinte podem vir de um executante humano, de outro computador, ou do próprio Cypher. O “executante” (ou secção de composição) gera e executa material musical. No Cypher, não há partituras armazenadas associadas com o programa. O “ouvinte” analisa, agrupa, e classifica os eventos de entrada à medida que eles chegam em tempo real e sem os comparar a alguma representação registada previamente. O “executante” usa vários estilos algorítmicos para produzir uma resposta musical. Alguns destes estilos podem usar pequenos fragmentos sequenciados mas nunca há qualquer execução de uma secção inteira da peça (Rowe 1993).

Rowe (1993) desenvolveu trabalho fulcral no desenvolvimento dos sistemas musicais interactivos. Não só apresentou pela primeira vez uma taxonomia para classificar sistemas musicais interactivos que será descrita brevemente, assim como, ao concentrar as suas atenções no desenvolvimento do “ouvinte,” realizou um trabalho sem par na implementação computacional de teorias musicais e cognitivas para análise da entrada musical (ver Rowe, 1993, 2001). Esta ideia de implementar teorias (musicais ou cognitivas) computacionalmente para melhorar a interpretação da entrada num sistema interactivo, serviu de modelo ao trabalho que eu próprio desenvolvi em dança interactiva (Guedes, 2005).

### *Max e Pd*

O Max (Cycling’74, 2007b), cujo nome presta homenagem a Max Mathews, foi desenvolvido no Institute de Recherche et Coordination Acoustique/Musique (IRCAM) em Paris a partir de 1986. O seu autor, Miller Puckette, tinha desenvolvido originalmente o Max para controlar o então poderosíssimo sintetizador 4X, que conseguia realizar síntese sonora em tempo real de forma muito sofisticada. A partir de 1990 o desenvolvimento do Max bifurca-se, tendo a primeira versão comercial aparecido pela *Opcode Systems* de David Oppenheim e David Zicarelli; e numa versão intitulada FTS (*Faster Than Sound*) desenvolvida no IRCAM por Miller Puckette e outros, entre os quais Corte Lippe e Zack Settel para controlar, sintetizar e processar som na placa ISPW (*IRCAM Sound Processing Workstation*) em computadores NeXT (Winkler, 1998). A vida da ISPW foi curta dado o facto de os computadores NeXT terem sido descontinuados em 1994. Miller Puckette entretanto sai do IRCAM em meados dos anos 90 e cria um novo sistema de software, Pd (*Pure Data*) (Puckette et al., s.d.), um ambiente com características idênticas às implementadas em Max e FTS, dedicado ao processamento e síntese sonora em tempo real e que aproveita a velocidade crescente de processamento

dos computadores pessoais, não necessitando por isso de *hardware* específico para o fazer.

Em 1997, aparece o *MSP* (*Max Signal Processing*, e também as iniciais de *Miller S. Puckette*) que é uma biblioteca de extensão para o Max implementando análise, síntese e processamento de som em tempo real. O MSP importa muitos dos algoritmos de Pd para o ambiente Max. Desde 1999 que o Max/MSP é comercializado pela Cycling'74<sup>16</sup>, uma companhia de David Zicarelli, que em 2003 lançou *Jitter*, uma biblioteca para o ambiente Max que permite processar vídeo em tempo real assim como imagens 3-D, e processamento matricial (Wikipedia, *Max (Software)*, s.d.).

Tanto o Max como o Pd são sistemas de programação gráfica que implementam o paradigma das linguagens de programação por objectos. Estes ambientes são hoje em dia extremamente populares na imensa comunidade internacional que trabalha em música electrónica, que os utiliza e desenvolve. A grande virtude destes sistemas é o facto de permitirem o trabalho interativo e integrado de análise e processamento de MIDI, áudio e vídeo em tempo real; e permitem que os seus utilizadores programem computadores a nível bastante elevado sem precisarem de aprender programação formalmente. Cada objecto é representado por uma “caixa” ou por um gráfico que é ligado a outros objectos para produzir um determinado resultado (e.g. tocar um ficheiro de áudio, variando a velocidade de leitura em tempo real). Por outro lado, a forma excelente de organização da sua documentação e a existência de *fora* de ajuda *on-line* bastante populares, permitem com que seja muito fácil começar a trabalhar nestes ambientes, mesmo autodidaticamente. O ambiente Max/MSP será o veículo de demonstração do conceito de composição em tempo real nesta lição.

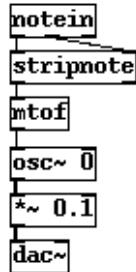


Figura 3. Exemplo de pequeno programa em Pd implementando um sintetizador simples. A entrada da nota MIDI (notein) é convertida para Hertz (mtof) que serve como valor de frequência para um oscilador de onda sinusoidal (osc~), que depois de atenuado (\*~) é enviado para a saída de áudio do computador (dac~).

### *Live coding*

Com o aparecimento de linguagens de programação dinâmica em meados de 90 (e.g. SuperCollider, que será brevemente descrito abaixo) começou a ser possível escrever programas cujo código pode ser modificado durante a execução do programa. Este aspecto, originou uma prática musical recente, chamada *live coding*, que consiste na

---

<sup>16</sup> <http://www.cycling74.com>, acedido a 28/03/2008.

programação e alteração de algoritmos musicais durante uma *performance*, onde o texto que está a ser introduzido pelos compositores programadores é projectado em grandes dimensões. O primeiro registo que há de uma *performance* com estas características é do grupo *slub* (Alex McLean e Adrian Ward) a 17 de Junho de 2000 na Hoxton Foundry em Londres (Toplap, *Historical performances — Toplap*, s.d.).

### **2.3.2. Classificação de sistemas musicais interactivos e definição de relações entre computadores e intérpretes**

Rowe (1993) propõe um sistema de classificação para sistemas musicais interactivos que é conceptualizado como uma combinação de três dimensões, cujas características ajudam a identificar as motivações musicais por detrás dos tipos de interpretação da entrada e os métodos de resposta. A primeira dimensão distingue sistemas *guiados por partitura* (*score-driven systems*) e por *performance* (*performance-driven systems*):

Os programas guiados por partitura utilizam colecções de eventos predeterminadas, ou fragmentos musicais armazenados, para serem comparados com a música que chega à entrada [do sistema]. Eles eventualmente organizam eventos utilizando as categorias tradicionais de pulsação, métrica, ou tempo. Estas categorias permitem a um compositor preservar e empregar formas familiares de pensar o fluxo temporal, tais como especificar que alguns eventos ocorrerão a tempo no próximo compasso ou no fim de cada quarto compasso.

Os programas guiados por *performance* não antecipam a realização de uma partitura. Por outras palavras, eles não têm uma representação musical armazenada que esperam encontrar à entrada. Além disso, os sistemas orientados para a performance não tendem a empregar as categorias tradicionais de métrica, utilizando mais frequentemente parâmetros mais gerais, envolvendo medidas perceptuais como densidade e regularidade, para descrever o comportamento da música a chegar.<sup>17</sup> (p. 7)

Outra dimensão agrupa as formas de resposta musical em três tipos — transformativa, generativa ou sequenciada:

Os métodos transformativos pegam em algum material musical existente e aplicam-lhe transformações para produzir variantes. De acordo com esta técnica, estas variantes podem ou não ser reconhecidas como estando relacionadas com o original. Para os algoritmos transformativos, o material de base é a entrada musical completa. Contudo, este material não precisa de estar armazenado — muitas vezes estas transformações são aplicadas directamente à entrada do material musical.

---

<sup>17</sup> T. A.

Para os algoritmos generativos, por outro lado, qualquer material de base que existe é elementar ou fragmentário — por exemplo, escalas ou conjuntos de durações armazenados. Métodos generativos usam conjuntos de regras para produzir a saída musical a partir do material de base armazenado, lendo estruturas de alturas de acordo com uma distribuição aleatória, por exemplo, ou aplicando processos seriais a valores de durações permitidos.

As técnicas de sequenciação utilizam fragmentos musicais pré-gravados em resposta a alguma entrada em tempo real. Alguns aspectos destes fragmentos podem ser variados em *performance*, tais como o tempo da reprodução, a dinâmica, pequenas variações rítmicas, etc.<sup>18</sup> (p. 7)

Finalmente, a terceira dimensão distingue os paradigmas de *instrumento* e *executante*:

Os sistemas que implementam o paradigma de instrumento preocupam-se em construir um instrumento musical estendido: os gestos performativos de um executante humano são analisados por um computador e conduzem a uma saída elaborada que excede a resposta instrumental normal. Imaginando um sistema destes a ser tocado por um executante, o resultado musical poderia ser pensado como um solo.

Sistemas seguindo o paradigma de executante tentam construir um executante artificial, uma presença musical com uma personalidade e comportamento próprios, embora o grau com que segue a liderança do parceiro humano possa variar. O paradigma do executante tocado por um humano produziria uma saída mais parecida com um dueto.<sup>19</sup> (p. 8)

Esta taxionomia tridimensional é importante pois aborda todos os aspectos a ter em conta na concepção de sistemas musicais interactivos — (1) a existência prévia ou não de material musical para ser executado interactivamente; (2) as formas de resposta musical do sistema; e (3) a sua utilização em *performance*, ou seja, se o sistema se comporta como um hiperinstrumento (Machover e Chung, 1989) ou como um parceiro executante com uma personalidade própria.

Todd Winkler (1998), define modelos de relação entre humanos e computadores na utilização de sistemas musicais interactivos todos eles baseados em práticas de execução musical tradicionais — o modelo do maestro e orquestra sinfónica; o modelo do quarteto de cordas; o modelo de improvisação do *combo* de Jazz; e o modelo de livre improvisação.

No modelo do maestro e orquestra sinfónica, um sistema interactivo é “dirigido” em função de uma partitura preexistente. Este modelo é dos mais antigos na exploração de sistemas digitais interactivos (cf. o sistema GROOVE acima descrito) e motivou toda a pesquisa do IRCAM nos anos 80 e 90 em sistemas de *score-following* (sistemas capazes

<sup>18</sup> T. A.

<sup>19</sup> T. A.

de seguir uma partitura durante a sua execução, um exemplo de sistemas guiados por partitura). O modelo do quarteto de cordas é um pouco mais complexo pois os vários músicos influenciam-se entre si durante uma *performance* e a responsabilidade musical é partilhada. No modelo de improvisação do *combo* de Jazz, a situação de improvisação sobre um modelo existente é utilizada, fazendo o computador reconhecer padrões rítmicos e melódicos e responder de forma improvisada a estes padrões. Finalmente, o modelo de improvisação livre, baseado no *free jazz* dos 60 oferece um modelo complexo do mais alto nível de interactividade. O computador pode interagir subtilmente ou de forma críptica com o intérprete criando música audível em si, aparentemente independente da do intérprete. Pode acontecer que nenhuma das partes esteja “em controlo,” mas cada uma delas pode influenciar a forma como a outra responde. Um bom exemplo desta ideia de *performer* virtual pode ser visto no trabalho do compositor/trombonista George Lewis (Winkler, 1998).

O que se apresentou acima são apenas modelos de concepção e prática musical com sistemas musicais interactivos. Estes modelos podem obviamente ser utilizados em combinação ou de forma híbrida no trabalho em música interactiva.

### **2.3.3. Composição por refinamento**

Os sistemas musicais interactivos contribuem para um processo de *composição por refinamento* [itálico adicionado]. Dado o facto de o programa reagir imediatamente a mudanças de configuração e entrada, o utilizador pode desenvolver aplicações compostionais refinando continuamente ideias iniciais e esboços até ao desenvolvimento de guiões para uma situação de *performance* em que o computador pode seguir a evolução e articulação de ideias musicais e contribuir para elas à medida que se desenvolvem.<sup>20</sup> (Rowe, 1993, p. 6)

O trabalho de compor uma peça é habitualmente um processo de refinamento. Na forma tradicional de compor, um compositor vai refinando as ideias, material musical, secções, e outros aspectos pertinentes da peça, desde os esquiços iniciais até à sua execução. Muitas vezes, após a primeira execução instrumental da peça, um compositor ainda a revê, corrigindo aqui e ali pormenores, ou mesmo secções inteiras da mesma. Os programas de notação musical em computador aceleraram esse processo. Hoje em dia, um compositor pode trabalhar uma peça instrumental tradicional e ser constantemente confrontado com o resultado sonoro do que está a fazer através de um sintetizador MIDI que simula os instrumentos para os quais a peça está a ser escrita.

O trabalho em composição algorítmica — por exemplo, utilizando um dos programas referidos em 2.2.6. — acelera esse processo vertiginosamente. Um músico pode gerar secções inteiras de uma peça que utiliza um dado algoritmo, ou conjunto de

---

<sup>20</sup> T. A.

algoritmos, e ir refinando os resultados a obter alterando partes desse algoritmo.<sup>21</sup> Contudo, não pode fazer essas alterações *durante* a execução do algoritmo. No trabalho com sistemas interactivos, é possível modificar os algoritmos e alterar a entrada de dados durante a execução do programa, fazendo muitas vezes com que o processo de refinamento seja parte integrante do processo musical. Este último aspecto torna possível a composição em tempo real.

#### 2.3.4. Metacomposição

Na introdução a *Notes from the Metalevel* (2004) Heinrich Taube questiona se existirá em música um nível de abstracção acima da informação de *performance* de uma partitura, e responde que sim, que poderá existir um *metanível (metalevel)*. Se uma partitura representa uma composição, então esse nível — que prefiro intitular *metacomposicional*<sup>22</sup> — representa a *composição da composição*. Para Taube:

A representação [metacomposicional] da música, lida com a representação da actividade, ou processo, de composição musical por oposição ao seu objecto, a partitura. Se [o nível metacomposicional] parece mais efémero que o nível da execução poderá dever-se ao facto que está mais relacionado com os processos cognitivos misteriosos que ocorrem no compositor. Além disso, não há nenhum objecto físico comum, como uma partitura ou CD, para representar o [nível metacomposicional].... Mas, ao contrário dos bits num CD ou das notas numa partitura, as declarações na representação [metacomposicional] são representações activas dos processos, métodos, algoritmos e técnicas que um compositor desenvolve para articular os sons nas suas composições.<sup>23</sup> (pp. 3-4)

Taube considera então que a música pode ser representada em três níveis diferentes de abstracção: acústico, interpretação e nível metacomposicional.

Rowe (2001), afirma que como os sistemas musicais interactivos derivam parâmetros de controlo a partir de uma análise em tempo real de uma *performance*, criam uma expansão no domínio da composição. “Ao delegar alguma responsabilidade criativa aos intérpretes, e outra a um programa de computador, o compositor empurra a composição para cima (a um [nível metacomposicional] captado nos processos

<sup>21</sup> A criação de um programa de computador também se faz habitualmente por um processo de refinamento. Por isso, talvez não seja de espantar a ligação que sempre existiu entre composição e programação, tendo sempre havido compositores que também são programadores excelentes. Como dizia Mark Coniglio (2002, conversa pessoal) “composers are, in general, good programmers.”

<sup>22</sup> O termo *metacomposicional*, ou *nível metacomposicional*, será utilizado daqui em diante para designar o que em inglês é habitualmente referido como *metalevel* em composição algorítmica.

<sup>23</sup> T.A.

executados pelo computador) e para fora (para os humanos que improvisam dentro da lógica da peça).”<sup>24</sup> (p. 6)

Eu próprio (Guedes, 2005; Cap. 7), ao referir a forma como a peça *Etude for Unstable Time* (2003) foi concebida composicionalmente, afirmo existirem dois níveis de composição na obra — um primeiro nível (composicional) definido pelos algoritmos composticionais estocásticos que geram um determinado resultado musical, e um segundo nível (metacompositonal) que consiste na (*re*)composição dos algoritmos durante a execução da peça a partir dos resultados musicais dados por eles durante a sua manipulação por mim e pelo bailarino durante a execução da peça.

#### **2.4. O conceito de composição em tempo real**

O conceito de composição em tempo real aqui apresentado surge, em parte, da minha prática recente com sistemas musicais interactivos. Nesse aspecto, a peça *Etude for Unstable Time* representa um ponto de viragem no meu pensamento composicional. Não é que eu não tivesse já experimentado alguns dos aspectos que levaram à formulação deste conceito anteriormente, mas, em *Etude*, a minha produção musical conheceu algumas “primeiras vezes:” (1) a utilização da biblioteca m-objects que desenvolvi para o ambiente Max/MSP, e que permite a um bailarino gerar estruturas rítmicas musicais e controlar o tempo musical em tempo real; (2) a criação de uma peça que foi concebida para ser composta durante uma *performance* através de uma colaboração entre três entidades distintas (um bailarino, um compositor e um computador); e (3) uma peça que era composta em tempo real em que algoritmos estocásticos que produziam um determinado resultado sonoro de acordo com regras previamente implementadas, estavam a servir de material “pré-compositonal.” Ou seja, estava(mos) a compor utilizando processos compostionais decorrentes, ou seja, a trabalhar a um nível metacompositonal.<sup>25</sup>

Por outro lado, observei que a prática musical com sistemas interactivos se tinha alterado substancialmente desde o começo do milénio e que a prática da “composição em tempo real” estava hoje em dia presente em várias actividades musicais que vão desde a simples utilização de *software* que permite compor em tempo real ao interagir com um *interface* ligado ao sistema (teclado alfanumérico ou MIDI, rato, sistemas utilizando sensores ou câmaras), passando por instalações interactivas em que o utilizador altera o conteúdo audiovisual em tempo real, até práticas musicais especializadas como a manipulação interactiva de algoritmos estocásticos (e.g. o trabalho de Joel Chadabe, de

<sup>24</sup> T.A.

<sup>25</sup> E a um nível *metacoreográfico*, uma vez que o bailarino/coreógrafo Maxime Iannarelli estava a utilizar um vocabulário coreográfico que ia articulando e transformando no tempo de acordo com o resultado musical que estava a ser produzido. Ou seja, a peça, embora seja diferente cada vez que é executada é reconhecível de *performance* para *performance*.

Karlheinz Essl, e algum do meu trabalho) ou até a criação dos próprios algoritmos durante uma *performance* (e.g. a prática do *live coding* e o movimento TOPLAP).<sup>26</sup>

A prática da composição em tempo real está, por isso, hoje em dia disseminada em várias formas e talvez mereça começar a ser analisada de uma forma mais sistemática — se fizermos uma pesquisa no *Google* utilizando os termos “Real-Time Composition” ou “Interactive Composition,” os resultados são decepcionantes.<sup>27</sup> Este texto, surge também como um esforço pessoal no sentido de começar a definir essa prática e analisá-la. Outros textos certamente o seguirão.

#### **2.4.1. Definição (*work-in-progress*)**

Composição em tempo real pode-se definir *como uma prática composicional utilizando sistemas musicais interactivos, em que algoritmos generativos com um comportamento estocástico são utilizados e não sendo transformados pelo compositor no decurso da execução de uma peça*. Este conceito está intimamente ligado à noção de composição interactiva avançada por Chadabe (1997), e pode ser considerado como uma elaboração do conceito de composição em tempo real de Karlheinz Essl (1998, 2007b):

[Composição em tempo real]: Música que não remete para qualquer texto notado, que não está ligada a qualquer tipo de duração, não é reproduzível e que é apresentada em tempo real, com a ajuda de programas de computador. Em vez de objectos preparados, que são executados, ou de forma móvel combinados uns com os outros, entram aqui processos fluidos, que estão em relações variadas uns com os outros: os denominados geradores de estruturas, que por causa de um modelo implementado, são capazes de produzir infinidáveis variantes de uma Meta-estrutura, só de forma abstracta possíveis de descrever. Os parâmetros dos geradores de estruturas deixam-se manipular em tempo real, pelos quais, de forma interactiva, podem intervir no processo generativo.<sup>28</sup> (Essl, 1998)

O conceito acima apresentado, também entra em ressonância com muitas das ideias acima apresentadas: nomeadamente os conceitos de “compositor-piloto” e de processo algorítmico formal que pode gerar várias composições de Xenakis, assim com o conceito de *manifold* de Tipei. Hiller, está também obviamente por detrás de tudo isto, na sua asserção de que os métodos estocásticos são o meio mais prolífico de exploração em composição algorítmica por computador.

A composição em tempo real é intrinsecamente interactiva. Ou seja, nesta prática estabelece-se um *diálogo* entre o compositor e o sistema na construção de uma peça. Daí

---

<sup>26</sup> (*Temporary|Transnational|Terrestrial|Transdimensional*) Organisation for the (*Promotion|Proliferation|Permanence|Purity*) of Live (*Algorithm|Audio|Art|Artistic*) Programming (Toplap, Main page—Toplap, s.d.)

<sup>27</sup> Excepto talvez o facto de se obter a *link* para a RTC Lib (*Real-Time Composition Library*) de Karlheinz Essl (Essl, 2007a)

<sup>28</sup> Tradução de Eugénio Amorim.

que a utilização de algoritmos generativos que produzem uma saída estocástica, não determinística, seja essencial para motivar o diálogo entre o compositor e o sistema — deve haver um ou mais elementos que fogem ao controlo do compositor. Este último aspecto é essencial na distinção entre composição em tempo real e improvisação.

Quando perguntei a Joel Chadabe (Lefkada, Grécia, 2007) como é que ele distinguia composição em tempo real de improvisação ele respondeu muito simplesmente: “na improvisação, uma pessoa está em controlo total do instrumento; quando se trabalha em composição em tempo real não se está controlo. O sistema tem uma personalidade própria. Contudo, a composição em tempo real é improvisatória.”

Esta subtileza de Chadabe, na distinção entre composição em tempo real e improvisação é importante, na medida em que define os sistemas de composição em tempo real como implementando o paradigma do *executante* na taxonomia de Rowe (1993). Enquanto que os sistemas interactivos para improvisação, implementam o paradigma do *instrumento*. Por outro lado, regressando ainda à classificação de Rowe, a resposta dos sistemas interactivos utilizados em composição em tempo real é essencialmente *generativa* e são sistemas guiados por *performance*.

Há também uma distinção, subtil mas importante, a fazer entre aquilo que é habitualmente designado por *música generativa* e composição em tempo real. A composição em tempo real, como eu a defino, é *generativa*, mas não é forçosamente *automática*, pois implica interacção com humanos no processo de realização de uma peça. No que habitualmente se designa por música generativa, a geração automática de música<sup>29</sup> é também incluída. A composição em tempo real pode então ser vista como uma subclasse da música generativa.

Contudo, por vezes, composição em tempo real e geração automática de música confundem-se. Um exemplo notável disso é a composição *Lexicon Sonate* (1992-) para piano Yamaha *Disklavier* controlado por computador, de Karlheinz Essl. Esta peça pode ser realizada pelo menos de três formas diferentes: numa instalação onde um computador gera a peça, em *performances* com o próprio Essl a interagir com os algoritmos da peça,<sup>30</sup> e numa versão que pode ser descarregada *on-line* que permite a um utilizador interagir directamente com os algoritmos (Figura 4).

---

<sup>29</sup> Um exemplo interessante de música generativa automática é *OM* (Gleetchplug Design, 2008) um programa de computador que gera *ambient music* em tempo real.

<sup>30</sup> Vídeo de Essl a interpretar a *Lexicon* pode ser visto em <http://www.essl.at/works/Lexikon-Sonate.html#vídeo> (acedido a 29/03/2008).

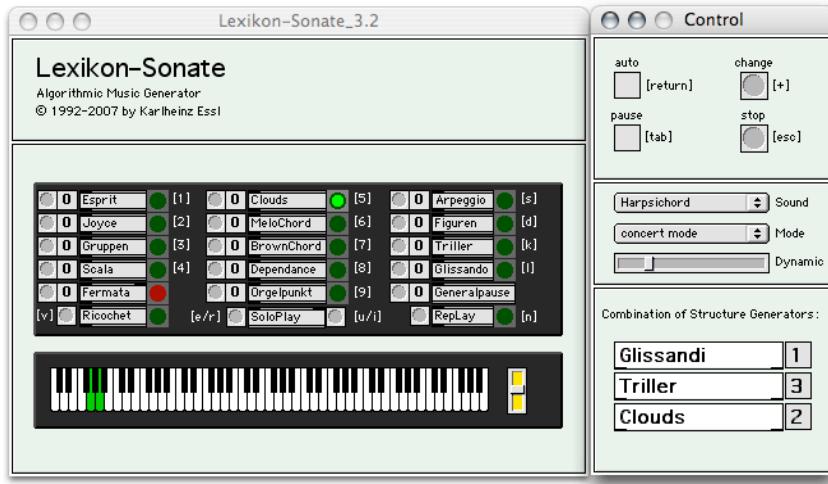


Figura 4. O interface do utilizador da versão *on-line* de *Lexicon Sonate* (Essl, 2007c)

Para acabar de definir a prática actual da composição em tempo real, resta definir os dois níveis de utilização para os quais os sistemas interactivos que facultam a composição em tempo real podem ser pensados — sistemas para utilizadores comuns e sistemas para especialistas. Os sistemas desenhados para utilizadores comuns usam geralmente um *interface* simples de operar, e podem aparecer sob a forma de instalações, jogos, ou *software*. Os sistemas para especialistas são geralmente constituídos por ambientes de programação dedicados ou bibliotecas que aumentam a funcionalidade dos ambientes de programação,<sup>31</sup> que permitem a criação de sistemas musicais interactivos para composição em tempo real.

#### 2.4.2. Alguns exemplos

No fim da secção anterior, distingui dois níveis de utilização de sistemas interactivos que facilitam a prática da composição em tempo real. Nesta secção serão dados alguns exemplos de *software* e sistemas musicais interactivos nos dois níveis de utilização identificados. Comecemos pelo nível do utilizador comum.

#### *Instalações*

Na instalação interactiva *Côr: um projecto audiovisual interactivo* (Guedes, Ula li e Woolford, 2003, Casa da Música, Festival em obra aberta) foi criada uma harpa (*l'harpe sans cordes*) em que os utilizadores, ao passarem a mão numa área da harpa, tinham uma experiência interactiva de criação audiovisual. A posição da mão na área de interacção espacializava o som em quatro canais e o movimento da mão controlava uma

<sup>31</sup> Exclui-se aqui propositadamente o facto de ser possível codificar um sistema nativamente utilizando uma linguagem de programação. Isto porque a comunidade de utilizadores especializados dos ambientes de programação mais utilizados é infinitamente maior que a comunidade de programadores destes sistemas. Por outro lado, há uma tendência maior, por parte da comunidade de utilizadores especializados, para se expandirem os ambientes de programação de utilização mais comuns do que propriamente codificar programas novos.

série de algoritmos estocásticos que produziam música dando aos seus utilizadores a *sensação* de estarem a criar música em tempo real. Havia 5 algoritmos diferentes que eram accionados por um processo de escolha estocástico que tinha em conta três factores: o tempo entre utilizadores, o tempo em que um utilizador utilizava o instrumento, o tipo de gesto do utilizador (tipo percussivo, ou tipo fluido). A música produzida ia-se alterando misteriosamente de acordo com estes três factores — e.g. quando o utilizador criava muitos gestos percussivos, um algoritmo com sons percussivos entrava em acção sendo depois preciso “acalmar” o instrumento para regressar aos sons mais fluidos, característica geral do “som” da harpa.

Visualmente, a *harpe sans cordes* permitia ao utilizador colorir toda a ”moldura” criada pela frente do grande auditório da Casa da Música controlando a côr de de vários projectores de grande potência em sincronia com a música. Além disso, combinações diferentes de cores correspondiam às secções musicais que iam sendo executadas.

### *Jogos de computador*

O jogo *Electroplankton* (Nintendo, 2005) de Toshio Iwai para a consola Nintendo DS (2005) permite aos seus utilizadores criar música de várias formas, tais como mover a caneta da Nintendo no seu ecrã táctil ou gravar sons a partir do microfone da consola.<sup>32</sup>

### *Software*

Existe hoje em dia um grande número de *software* que permite a criação musical interactivamente. Este *software* geralmente contém *interfaces do utilizador* bastante intuitivos e fáceis de operar. Dois excelentes exemplos nacionais são os programas *Polissonos* (2007a) e *Digital Jam* (2007b) do jovem compositor Rui Penha, actualmente em uso no projecto Digitópia na Casa da Música. Em Polissonos, um utilizador pode gerar várias melodias em escalas à escolha, ou ritmos, em *loop*, a partir de vértices de polígonos. Estes fragmentos musicais podem depois ser sequenciados e alterados durante a execução (Figura 5). Digital Jam é um programa para improvisação colectiva utilizando computadores ligados em rede (Penha, 2008).

---

<sup>32</sup> Ir a <http://www.youtube.com/watch?v=d3v6npP8OZk> (acedido a 30/03/2008) para ver uma demonstração do jogo.

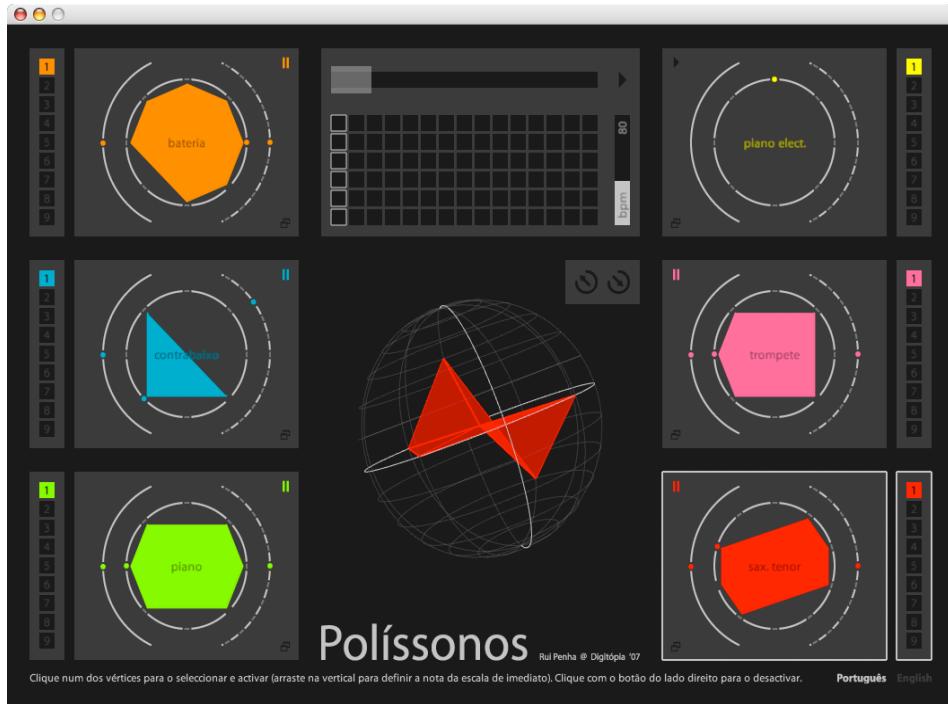


Figura 5. Interface do utilizador em Políssonos (Penha, 2007)

Outro exemplo de criação de *software* para fruição e composição musical através de um *interface* intuitivo é o programa *Stockwatch* do meu aluno Samuel van Ransbeeck (2007). Este programa sonifica para MIDI os dados dos índices de bolsa de valores de vários sítios do mundo (Figura 6). Tendo sido inicialmente concebido como um programa de geração automática de música, *Stockwatch* permite ao utilizador “recompor” a música que está a ser tocada, seleccionando índices de bolsas de valores diferentes (e.g. Dow Jones ou Kuala Lumpur Composite Index). Permite também alterar as durações das notas correspondentes ao valor das acções no mercado, a duração entre eventos, os sons do sintetizador MIDI. Além disso, permite ainda controlar a mistura entre a sonificação directa para MIDI (apresentada nos canais 1 e 2) com a mistura de uma reprodução de variantes das melodias que é feita através de uma análise por cadeias de Markov (canais 3 e 4) dos dados sonificados em tempo real. Em *Stockwatch*, não há nenhum controlo possível sobre a geração de alturas (a não ser que se mude de índice bolsista) ou mesmo sobre a articulação temporal dos eventos. A saída MIDI do programa pode ser exportada para um ficheiro para ser ouvido ou trabalhado posteriormente.

*Stochos* (Bökesoy, 2004) e *ixiQuarks* (ixi audio, 2007) constituem outros dois exemplos de *software* que produzem resultados sonoros fascinantes. *Stochos* é um gerador de eventos estocásticos baseado nos algoritmos de Xenakis e *ixiQuarks* é uma aplicação contendo várias ferramentas divididas nas categorias de utilitários de base, instrumentos, efeitos e filtros. Um aspecto interessante nos instrumentos de *ixiQuarks* é o seu interface do utilizador que muitas vezes lembra o de um jogo de computador (e.g. *Predators*. Ver Figura 7).

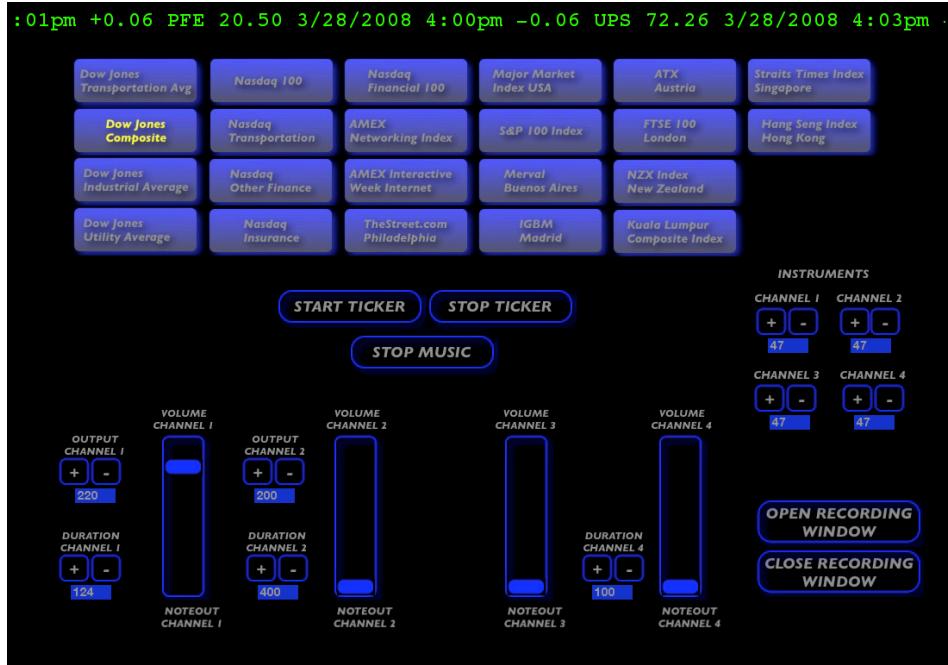


Figura 6. O interface do utilizador de *Stockwatch* (van Ransbeek, 2007)

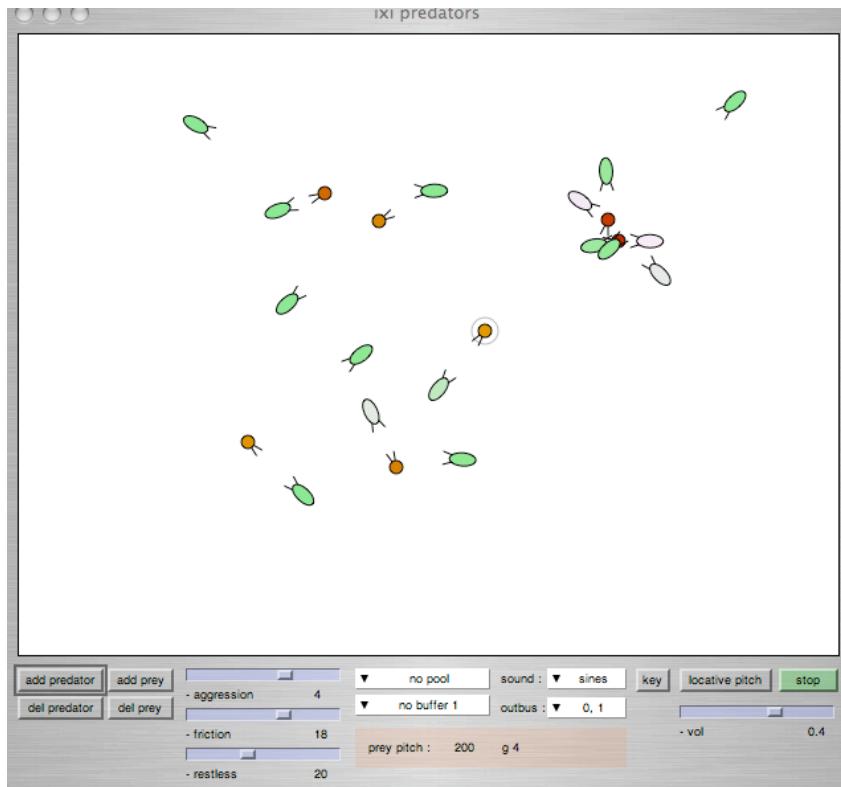


Figura 7. Interface do utilizador de *Predators* um “instrumento” de ixiQuarks.

### *Sistemas para especialistas*

Como se disse acima, os sistemas para especialistas são geralmente constituídos por ambientes de programação dedicados ou bibliotecas que aumentam a funcionalidade dos ambientes de programação. Nestes sistemas, com uma curva de aprendizagem bastante mais acentuada que a do *software* descrito na secção anterior, o utilizador habitualmente constrói programas para um determinado fim específico. Isto não quer dizer, porém, que o *software* mencionado acima — assim como uma série de outros programas do género actualmente disponíveis — possam servir os utilizadores especializados na prática da composição em tempo real e música interactiva.

Além dos já referidos Max/MSP/Jitter e Pd, existem hoje em dia outros ambientes de programação de utilização generalizada. Contudo, antes de observarmos esses ambientes de programação, convém referir a existência de uma biblioteca de objectos para Max/MSP dedicada para a prática da composição em tempo real: a *RTC-lib* (*Real-Time Composition Library*) de Essl (2007a) é um conjunto de *patches*<sup>33</sup> e objectos externos<sup>34</sup> que permitem experimentar uma série de técnicas compostionais, como processos seriais, permutações e acaso controlado em tempo real. A RTC-lib foi originalmente concebida para a *Lexicon Sonate* (Essl, 2007c).

O *SuperCollider* foi originalmente concebido por James McCartney em 1996 para síntese de áudio em tempo real e composição algorítmica. É agora um programa *open source* e de distribuição livre (*SuperCollider*, s.d.), desenvolvido por uma comunidade vasta de utilizadores e programadores. O SuperCollider combina a linguagem por objectos Smalltalk com elementos de linguagens funcionais de programação com uma sintaxe da família do C (Wikipedia, *SuperCollider*, s.d.). SuperCollider é uma linguagem de programação dinâmica<sup>35</sup> que é bastante utilizada na prática do *live coding*.

Finalmente, o *Chuck:Strongly-Timed Concurrent, and On-the-fly Audio Programming Language* (Wang e Cook, 2003, 2004) é uma nova linguagem para síntese, composição, *perfomance* e análise em tempo real que corre no sistemas, Mac OS X, Windows e Linux (*Chuck:Strongly-Timed Concurrent, and On-the-fly Audio Programming Language*, s.d.). Desenhada especialmente para se poder programar e alterar os algoritmos que se vão criando durante uma *performance*, Chuck tem ganho um grande número de adeptos no *live coding*, uma prática musical recente que consiste em programar, a partir do nada, algoritmos que geram música e vídeo durante a *performance*.

#### **2.4.3. Novas práticas**

Pelo que se tem vindo a apresentar nesta secção sobre composição em tempo real, pode-se perceber facilmente que os sistemas musicais interactivos originaram novas

<sup>33</sup> Patch é a designação habitual para um programa em Max/MSP.

<sup>34</sup> Objectos externos são objectos para utilizar no Max que não fazem parte da distribuição original. Estes objectos estão habitualmente programados em C, C++, ou Java.

<sup>35</sup> Linguagens de programação dinâmicas são linguagens de alto nível que permitem executar comandos e adicionar novo código durante a execução do programa.

práticas musicais que estão cada vez mais disseminadas por uma grande comunidade “compositores.” Estes “compositores” podem ir desde utilizadores comuns, sem grande experiência ou conhecimento musicais, até compositores e compositores-programadores com grande nível e experiência.

Nesse aspecto, os sistemas musicais interactivos, principalmente os que facultam a actividade de composição em tempo real, têm contribuído para aproximar as pessoas da prática musical. Curiosamente, ou talvez não, estas práticas têm-se desenvolvido fora do formato de concerto de música instrumental tradicional, abrindo e desenvolvendo novas perspectivas de criação musical que estimulam a continuação do desenvolvimento de sistemas musicais interactivos. Alguns exemplos:

1. A utilização de *software* que permite a criação de música em tempo real por uma pessoa sem conhecimentos musicais na intimidade do seu computador;
2. As instalações interactivas em espaços públicos em que as pessoas podem intervir dinamicamente no conteúdo que está a ser produzido;
3. As orquestras de *laptops* que vão florescendo um pouco por todo lado;
4. A prática do *live coding*, em que (geralmente) um grupo de compositores se junta para programar algoritmos durante uma *performance*, exibindo em projecções de grandes dimensões o código que estão a dactilografar;
5. A extensão interactiva a outras artes do palco como a dança e o teatro (ver, por exemplo, Dixon, 2007; Guedes, 2005).

Estas práticas, são talvez mais efémeras que a do concerto instrumental tradicional. Talvez o formato de concerto tradicional não seja o espaço correcto de apresentação destas práticas musicais. Quem já teve que esperar sentado, no meio de um concerto, que um compositor ou técnico tivesse que reiniciar o computador passados três minutos na execução de uma obra de quinze, e ter de ouvir esses três minutos novamente, consegue perceber muito bem o que estou a dizer — principalmente se a obra for daquelas de *audição menos fácil*.

Passemos então à demonstração do conceito de composição em tempo real.

### 3. Demonstração

Nesta secção vamos demonstrar o conceito de composição em tempo real utilizando a versão do 4.6.3. do Max/MSP/Jitter<sup>36</sup> (Cycling'74, 2007b). Este ambiente de programação é excelente para este tipo de trabalho por causa da sua flexibilidade. Como já foi dito acima, o Max permite programar um computador a um nível elevado sem se necessitar de grandes conhecimentos formais de programação; e também permite trabalhar interactivamente.

Vamos então desenvolver um sistema destinado a uso especializado. Com um fim puramente didáctico, vamos incorporar no exemplo a desenvolver, quatro aspectos importantes do trabalho em Max: (1) a implementação de um algoritmo conhecido; (2) a utilização de uma biblioteca que não faz parte da distribuição original; (3) a utilização e modificação de partes de *patches* que já são distribuídos com o Max; e (4) a utilização de um objecto externo, codificado em C por mim originalmente, que implementa uma versão simplificada da equação de indispensabilidade métrica de Barlow (cf. nota de rodapé 9). Nesta demonstração serão também abordados os conceitos de composição por refinamento e metacomposição.

Embora o Max tenha originalmente poucos algoritmos que produzam directamente um resultado interessante, há hoje em dia uma série de algoritmos publicados que podem ser transportados (mais ou menos facilmente) para este ambiente. Livros como o de Dodge e Jerse (1985), Roads (1996), e Rowe (1993, 2001), têm muitos algoritmos que se podem implementar no Max.<sup>37</sup> Por outro lado, Winkler (1998) apresenta de forma bastante acessível, bastantes *patches* que podem ser utilizados directamente na análise e/ou composição de eventos MIDI em tempo real.

Passemos agora à implementação do programa.

#### 3.1. O algoritmo gerador e sua implementação

Uma das premissas na definição que apresentei em 2.4.1. foi a da utilização de algoritmos estocásticos. Para os fins desta lição, vamos utilizar como base um algoritmo clássico de geração de caótica que Curtis Roads apresenta em *Computer Music Tutorial* (1996). A função logística formulada por R. May (1976) pode ser definida muito simplesmente pela equação:<sup>38</sup>

$$(1) \quad x_{n+1} = l \times x_n \times (1 - x_n)$$

em que  $0 < l \leq 4$ . Para cada novo valor de  $x$  uma pessoa utiliza o valor anterior multiplicado por alguns factores. De acordo com Roads (1996) a função comporta-se da seguinte forma:

Para  $0 \leq l \leq 1$ , todas as iterações convergem para ... 0

<sup>36</sup> Daqui para frente, para simplificar, designado unicamente por Max.

<sup>37</sup> Rowe (2001), dá inclusivamente um exemplo de como codificar um dos algoritmos em C++ do seu livro como um externo para Max.

<sup>38</sup> No original a letra a variável  $l$  aparece como a letra grega *lambda*.

Para  $1 < l < 3$ , um limite fixo com o valor de  $(1 - 1/l)$  atrai todos os valores de  $x$

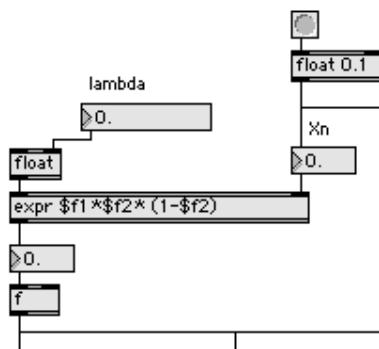
Para  $l \geq 3$ , o ponto fixo continua inicialmente mas fica instável de forma a que os pontos na sua vizinhança não são atraídos mas sim repelidos. Este mesmo âmbito assiste ao nascimento de um ciclo de *limites* entre dois membros (uma alternância entre os valores),

A  $l = 3.449499$ , o ciclo de dois membros bifurca para um ciclo de quatro.

De  $l = 3.54409$  a  $3.569946$  o ciclo de quatro desdobra-se para um ciclo de oito e assim indefinidamente, criando uma cascata harmónica; isto corresponde a fenómenos físicos como o começo de turbulência nos fluidos.

Entre 3.8 e 3.86 uma região de estabilidade aparece só com algumas alternâncias, até ser desestabilizada para um caos ainda maior. (p. 888)

No Exemplo 1 podemos ver a implementação da função logística em Max:



Exemplo 1. Função logística implementada no objecto **expr**.<sup>39</sup> O objecto **float** acima à direita serve para inicializar  $x_0$  com o valor 0.1.

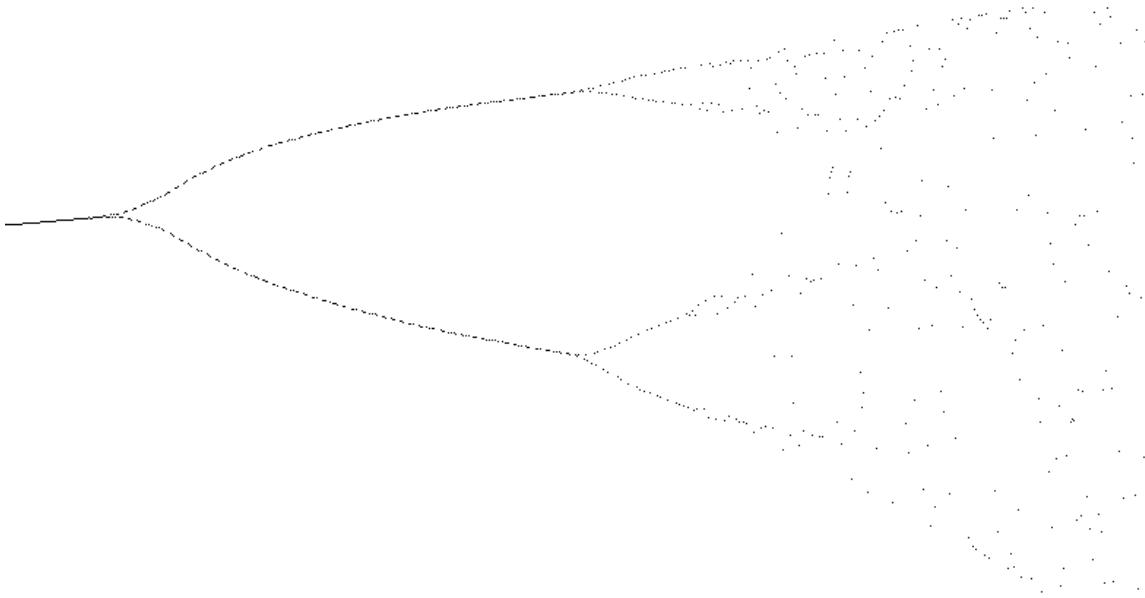
No Exemplo 2, podemos ver o gráfico da função, calculado pela fórmula com os valores de  $l$  entre 2.8 e 4 onde se podem notar as tais bifurcações mencionadas por Roads (1996). Num workshop na *International Computer Music Conference* (Copenhaga, 2007), Heinrich Taube mostrou esta função como um exemplo interessante para geração *caótica* de melodias,<sup>40</sup> utilizando os valores de  $l$  entre 3,... e 4. Vamos então explorar musicalmente este gráfico, convertendo os valores dados pela função para notas MIDI. Para isso, ligamos a saída da função ao objecto **zmap** que converte os valores de saída (entre 0. e 1.) para as notas MIDI 20 a 100. O Exemplo 3 mostra o *patch*. O objecto **metro** envia mensagens de *bang*<sup>41</sup> repetidamente para a função para podermos explorar musicalmente a função alterando os valores de  $l$  (*lambda*).

---

<sup>39</sup> Para maior clareza, os objectos do Max aparecem no texto em **negrito**.

<sup>40</sup> Roads (1996) menciona que Pressing (1988), Bidlack (1992), Ames (1992), Di Cipio(1990) e outros exploraram inicialmente esta função. Contudo, não nunca tive acesso a essas fontes.

<sup>41</sup> Ordens de execução.



Exemplo 2. Gráfico da função logística calculada pela equação do da função logística. Podem-se ver as bifurcações mencionadas por Roads (1996).

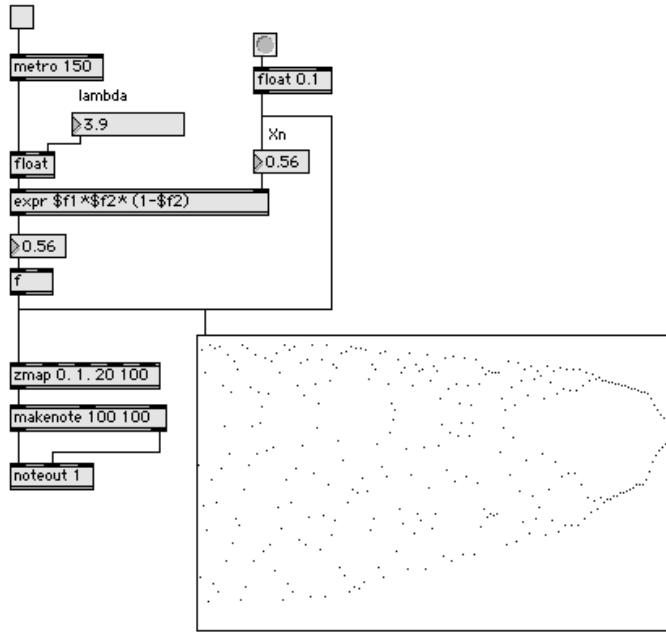
Podemos então *viajar* neste espaço musical e explorar musicalmente a função, experimentando vários valores de  $l$ , dando valores diferentes ao âmbito das notas MIDI, ou ligando a saída a um instrumento virtual para explorar timbres mais interessantes. Os resultados musicais mais interessantes situam-se a partir do momento em que a função começa a alternar valores entre quatro membros ( $l$  maior que 3.5 aprox.). Se dermos a  $l$  um valor entre 3.8 e 3.86 podemos verificar a tal zona de estabilidade mencionada por Roads (1996).

### 3.2. Refinamentos até uma possível versão final

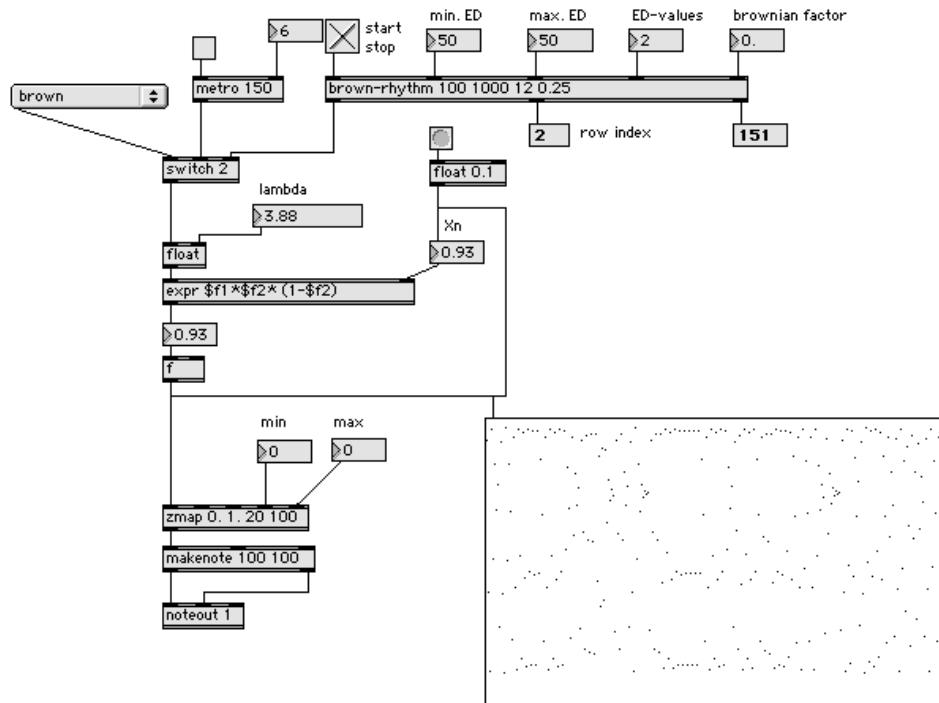
Façamos então um primeiro *refinamento* ao programa, tornando o ritmo menos monótono através da utilização do gerador de ritmo *browniano*<sup>42</sup> da RTC-lib de Essl (**brown-rhythm**), aproveitando o objecto **switch** para podermos alternar entre pulsação regular e ritmo browniano. O *patch* fica agora como no Exemplo 4.

---

<sup>42</sup> O movimento Browniano é um dos processos estocásticos contínuos no tempo mais simples, e é o termo utilizado para descrever o movimento aleatório de partículas num líquido ou gás, ou o modelo matemático para descrever esses movimentos, muitas vezes chamado *teoria das partículas* (Wikipedia, *Brownian Motion*, s.d.). Neste objecto, este tipo de movimento é utilizado para saltar entre índices de uma tabela contendo durações que evoluem exponencialmente entre dois limites. O ‘factor browniano’ é o grau de desordem (entre 0 e 1) com que esses índices são procurados. Quando está em 0, não há saltos nas iterações; quando o valor é 1, os saltos entre os índices são feitos aleatoriamente.

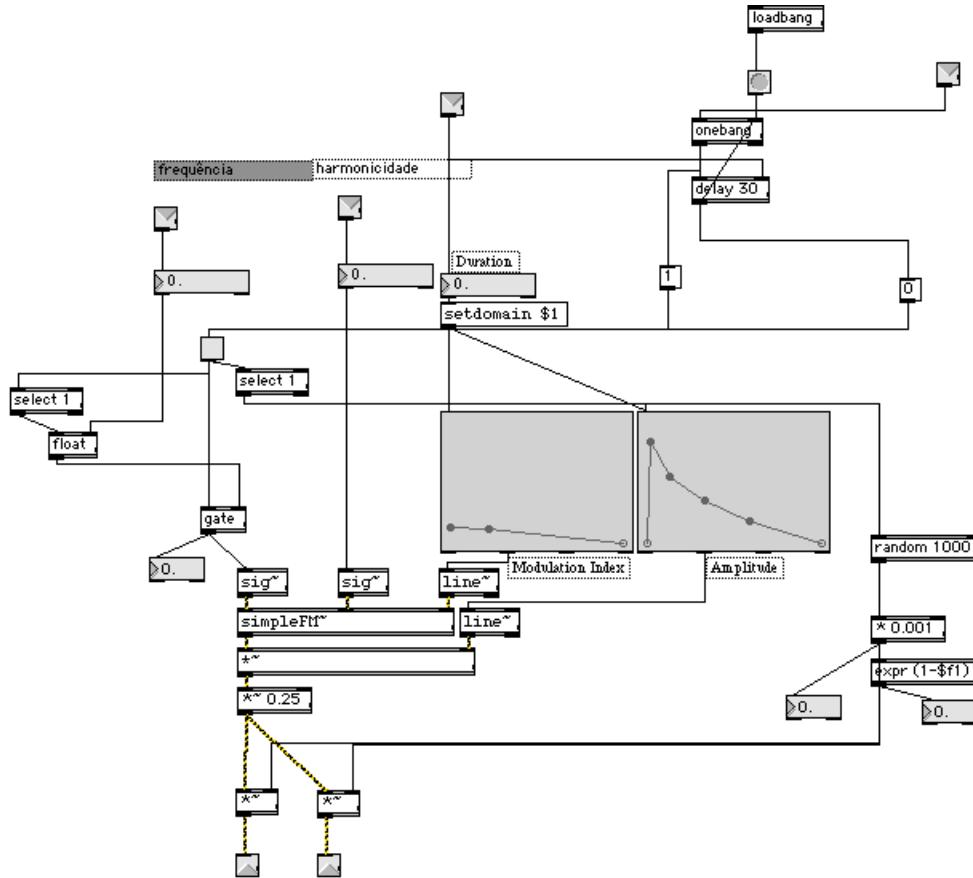


Exemplo 3. Patch para a sonificação inicial da função logística. O objecto **zmap** converte os valores entre 0. e 1. para notas MIDI entre 20 e 100 respectivamente, e está ligado ao objecto **makenote** para criar uma nota MIDI com o valor obtido com a velocidade 100 e duração de 100 ms. O objecto **noteout** envia a nota para MIDI para um *interface* ou para um sintetizador virtual interno.



Exemplo 4. Adição de ritmo browniano como um primeiro refinamento do *patch*.

Uma das grandes virtudes do Max é permitir converter facilmente a saída MIDI para áudio digital utilizando os objectos do MSP. Vamos agora converter os valores das notas MIDI para valores de frequência e aproveitar uma versão modificada do *patch* tutorial de síntese FM para “colorirmos” a nossa saída MIDI com sons gerados por FM. O *subpatch FMTone* é um reaproveitamento do *patch* tutorial “11. FM synthesis” distribuído com a aplicação, readaptado para este fim e com alguns melhoramentos — e.g. difusão em dois canais aleatoriamente (Ex. 5). Podemos agora ver o aspecto do nosso *patch* geral implementando este último refinamento no Exemplo 6.

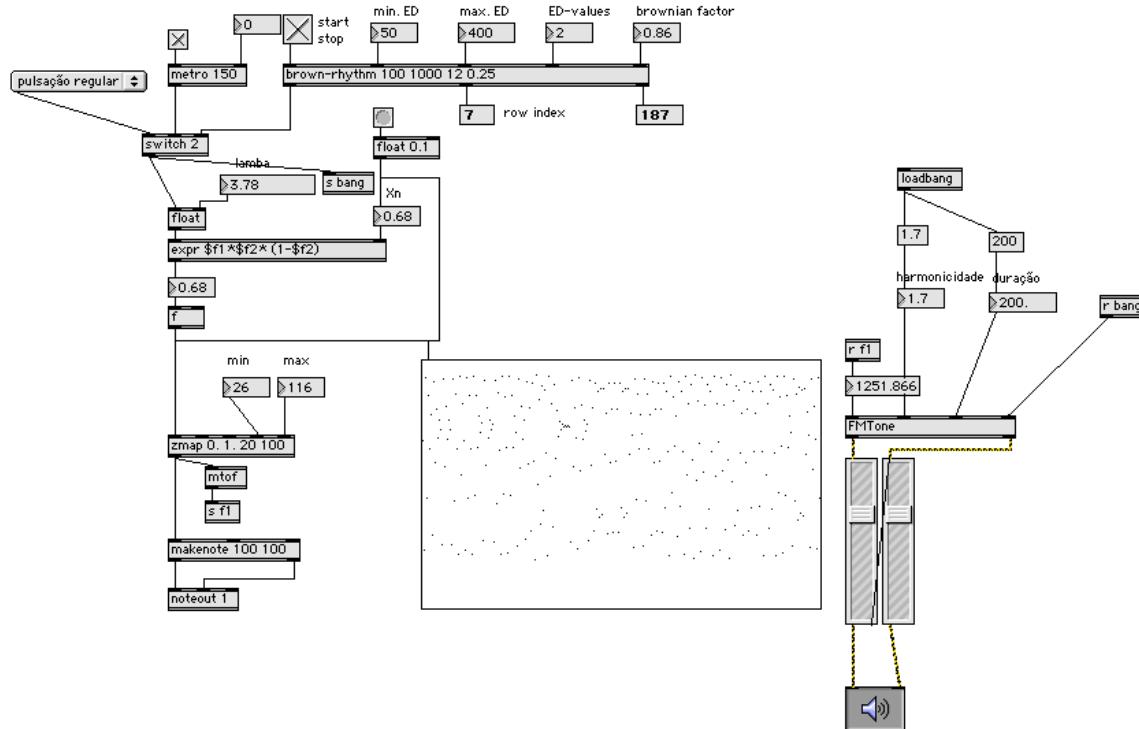


Exemplo 5. Re-aproveitamento do *patch* tutorial “11. FM synthesis.”

O objecto **mtof**, ligado à saída do **zmap** converte para Hertz o valor da nota MIDI e envia-o remotamente (**s f1**) para a entrada de frequência do **FMTone**. Como podemos ver também, conseguimos controlar externamente a harmonicidade e a duração dos sintetizador FM que implementamos. Quando a duração das notas FM é superior à dos intervalos temporais entre eventos, essas notas ficam “penduradas” dando um efeito adicional interessante.

Temos agora vários parâmetros de intervenção no nosso sistema que podemos manipular para começar a pensar numa secção de uma peça composta em tempo real. A saber: variação do valor do caos da função (*l*); duas formas de articulação rítmica (pulsação regular e ritmo browniano); variação das durações máxima e mínima e variação

do factor browniano no gerador (entre 0. e 1., sendo 1 o valor mais caótico); variação da harmonicidade e duração na síntese FM. Note-se que a nossa intervenção no sistema é de carácter *metacomposicional*, uma vez que estamos a agir sobre processos generativos decorrentes, em execução.



Exemplo 6. Inclusão de um sintetizador FM com objectos do MSP para diversificar a saída do sistema

Comecemos então a fazer um plano metacomposicional de uma secção que vai consistir em variações destes parâmetros no tempo. Um possível *algoritmo*:

- 1- Começar com um valor de  $l$  acima de 3 com pulsação regular usando somente a saída MIDI;
- 2- Ir aumentando o valor até haver uma grande alternância entre as notas;
- 3- Introduzir a saída FM com um valor qualquer de harmonicidade e uma duração bastante curta;
- 4- Variar a harmonicidade ou não e experimentar com valores *caóticos* de  $l$ ;
- 5- Ir progressivamente para valores de  $l$  entre 3.8 e 3.86 (zona de estabilidade);
- 6- Voltar a uma zona caótica e intensificar a saída aumentando a harmonicidade consideravelmente assim como a duração dos eventos de síntese FM para lá dos valores da pulsação;
- 7- Mudar bruscamente para o gerador de ritmo browniano com um intervalo grande de durações (50-400 ms) com cerca de 20 de iterações;
- 8- Voltar a ao ponto 4- ou a outro ponto acima de 4- sempre que desejar;
- 9- Parar quando estiver satisfeito.

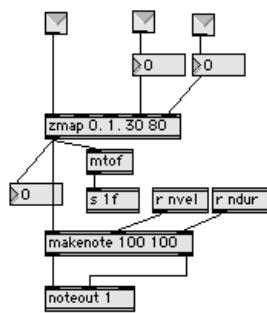
O algoritmo que se apresenta acima é só uma de várias possibilidades que o sistema oferece para a composição de uma secção em tempo real utilizando estes parâmetros. Todo este processo deve obviamente ser aperfeiçoado intuitivamente, experimentando valores, alterando os passos do algoritmo, e prosseguir por tentativa e erro até se chegar a resultados satisfatórios. Contudo, o sistema acima apresentado é um pouco rudimentar e talvez fosse melhor pensar em estratégias que possam tornar o resultado sonoro mais complexo, de forma a abrir outras possibilidades e, consequentemente, torná-lo mais apelativo à interacção.

Como vimos acima, os sistemas musicais interactivos promovem uma forma (interactiva) de composição por refinamento. Isto faz com que muitas vezes um compositor vá refinando o sistema de execução em execução podendo facilmente entrar num processo contínuo de refinação. Este aspecto, não sendo inteiramente novo como já vimos acima, é hoje em dia muito mais comum. Uma peça que existe sob a forma de programa de computador, e que se constitui como um conjunto de operações com comportamento estocástico que produzem música intrinsecamente mutante, é um campo fascinante de exploração. Esta música põe também problemas interessantes do ponto de vista musical e formal. Esta discussão ficará para a conclusão da lição. Regressemos agora ao desenvolvimento do nosso sistema interactivo de exploração da função logística.

Um refinamento seguinte, pode ser a criação de vários fluxos simultâneos de notas MIDI, o que se implementa muito facilmente dado o facto de o Max permitir a multiplicação dos objectos com os simples comandos de *copy/paste*. Para simplificar o aspecto gráfico do *patch*, decidiu-se encapsular os objectos que recebem a saída da função logística e a convertem para MIDI (**zmap**, **makenote** e **noteout**) num objecto chamado **NotaMIDI** que aceita como argumentos o número do canal MIDI a utilizar, e o âmbito dos intervalos de conversão (Ver Exemplos 7a) e b)).

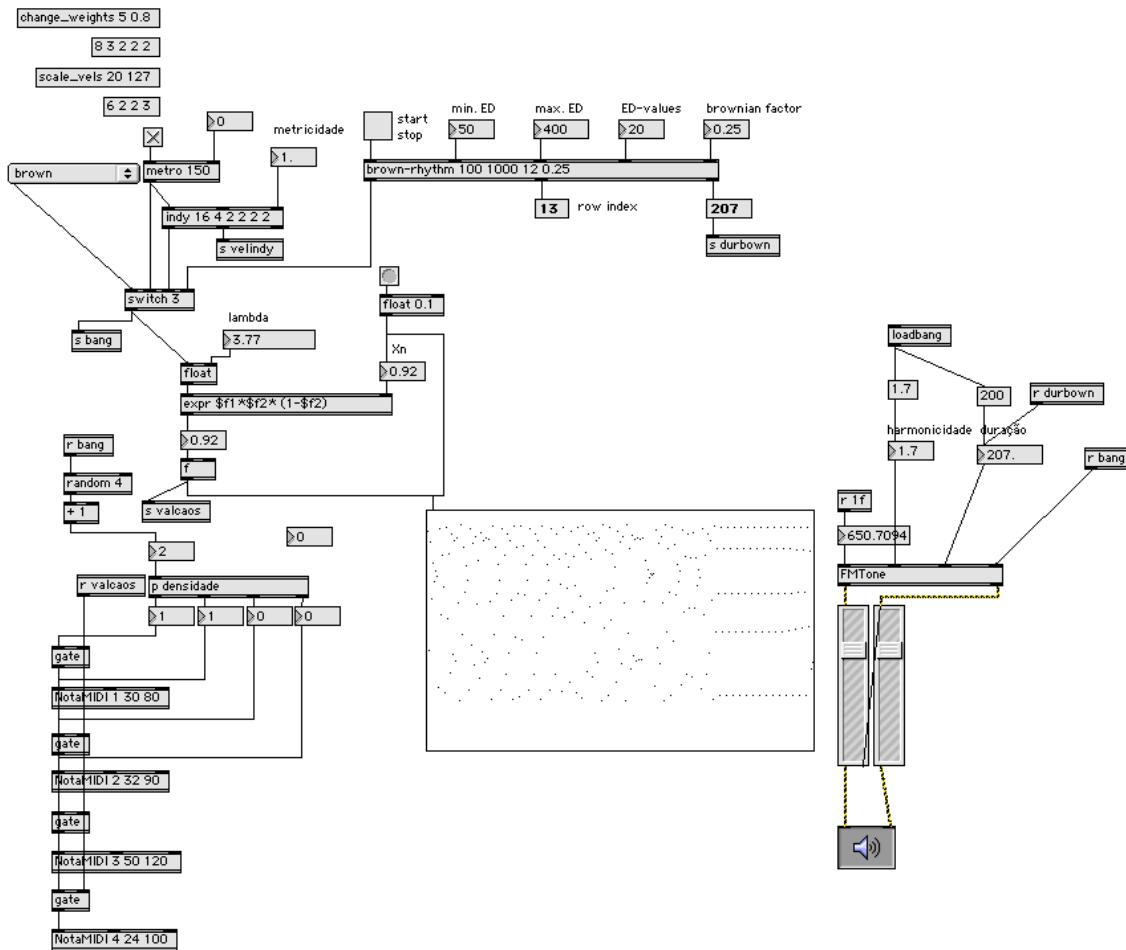
a)

b)



Exemplo 7 a). Objecto **NotaMIDI** contendo os argumentos para inicialização de canal MIDI (1), valor mínimo (30) máximo (80) de conversão. A entrada (*inlet*) mais à esquerda é o valor da função, as duas entradas seguintes são respectivamente os valores mínimos e máximos de conversão que podem ser alterados externamente. b) Interior do objecto. Reparar também os objectos **r nvel** e **r ndur**, que permitem a recepção de valores para a velocidade e duração enviados remotamente.

Na versão do *patch* apresentada no Exemplo 8, pode-se ver a inclusão de quatro objectos **NotaMIDI** a transmitirem para quatro canais MIDI independentes, assim como outros melhoramentos que serão discutidos em seguida: o controlo da densidade de notas aleatoriamente (**p densidade**) e a inclusão de um novo modo rítmico (o objecto **indy**) que permite a geração estocástica de ritmo numa dada métrica de acordo com uma simplificação da fórmula de indispensabilidade métrica de Barlow (1980).



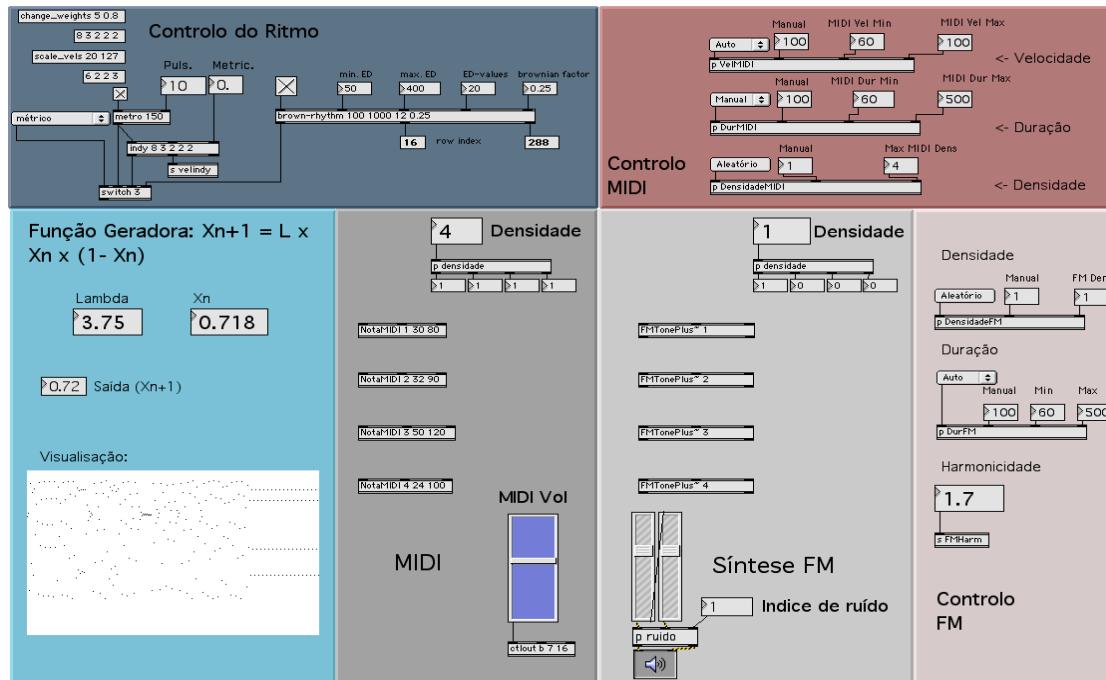
Exemplo 8. Refinamentos subsequentes ao *patch* apresentado no Exemplo 6. Inclusão de um modo métrico de articulação rítmica, quatro fluxos independentes de notas MIDI que convertem os valores de saída da função de forma diferente. Controlo aleatório de densidade de eventos.

Como se disse acima, o controlo aleatório da densidade é feito com um gerador de números aleatórios (**random**) ligado ao *subpatch* **p densidade** que deixa ou não passar o valor da função (objecto **gate**) para ser convertido. Repare-se também, acima, no objecto **indy**, que recebe mensagens *bang* do objecto **metro** e que articula ritmo estocasticamente numa dada métrica. O argumento 16 4 2 2 2 do objecto significa que há 16 pulsações factorizadas 4 vezes, e os números primos da factorização são 2 2 2 2. Isto é uma forma de representar o compasso 4/4 até à subdivisão da semicolcheia para ser calculado pela fórmula de indispensabilidade métrica de Barlow (1980). Esta função calcula os pesos

relativos da importância de cada pulsação na definição desta métrica. No canto superior esquerdo, podemos ver algumas mensagens que se podem enviar para o objecto. A mensagem '8 3 2 2 2' representa um compasso 4/4 estratificado até ao nível da colcheia (oito pulsasões) e a mensagem '6 2 2 3,' representa um compasso 6/8 estratificado até ao nível da colcheia também (2 x 3). A mensagem 'scale.vels 20 127' escala as velocidades das notas à saída do segundo *outlet* de **indy** — as velocidades das notas são escaladas à razão do peso na pulsação; ou seja, a pulsação com menor peso tem uma velocidade de 20 e a de maior peso, uma velocidade de 127. Esta correspondência proporcional entre peso e velocidade ajuda a definir a articulação da métrica. Finalmente, a mensagem 'change\_weights' permite mudar selectivamente o peso de cada pulsação na métrica para criar (por exemplo) efeitos de síncopa — no caso apresentado acima, a pulsação 5 fica um peso de 0.8 (valores entre 0. e 1.).

No Exemplo 9, e passados muitos refinamentos, podemos ver uma possível versão final do *patch* para ser utilizada em *performance*. Embora visualmente radicalmente diferente do *patch* anterior, esta versão segue a linha dos refinamentos implementados até aqui. Comecemos pela organização visual.

Em primeiro lugar, esta nova organização do programa agrupa visualmente as seis secções do programa: (1) função geradora; (2) saída MIDI; (3) secção de síntese FM; (4) controlo dos parâmetros da síntese FM; (5) controlo dos parâmetros MIDI; e (6) controlo da articulação rítmica. Cada secção oculta do utilizador os aspectos do programa que não são necessários observar durante a sua execução. Esta versão do programa dá ênfase ao que é necessário visualizar durante em *performance*, simplificando consideravelmente o aspecto visual do *patch* — aliás, uma prática bastante recomendada na programação em Max (cf. Winkler, 1998).



Exemplo 9. Uma possível versão final do *patch* para ser utilizada em *performance*, seguindo a linha de refinamento apresentada. Note-se, sobretudo, a reorganização visual do *patch*, ocultando muitos pormenores do utilizador e agrupando os controlos implementados para a síntese FM e para a saída MIDI.

Comecemos pelas “novidades” introduzidas. Como se pode ver na secção de Síntese FM, foi implementada a possibilidade de se gerarem quatro notas FM com densidade controlável, à imagem do que se fez para a geração MIDI (reparar nas semelhanças entre as secções). Quer a secção síntese FM, quer a secção MIDI possuem agora controlo da densidade que pode ser feito de duas formas — Aleatória, como exemplificado acima, e Manual, permitindo ao utilizador seleccionar manualmente a densidade de eventos. Foi também implementado um controlo sobre as durações dos eventos MIDI e Síntese FM com dois modos também acessíveis ao utilizador — Manual, e Aleatório entre limites de valores máximo e mínimo a definir pelo utilizador. Na secção de Síntese FM, há um terceiro modo, Auto, que entra em funcionamento quando o controlo do ritmo está em modo browniano. Implementou-se também a possibilidade de controlar a velocidade dos eventos MIDI (cf. Secção “Controlo MIDI” do *patch*). O controlo da velocidade MIDI, oferece ao utilizador a possibilidade de controlar a velocidade de dois modos — Manual, e Aleatório entre limites de valores máximo e mínimo, semelhante ao controlo da duração. Há também um modo “Auto” no controlo da velocidade MIDI, quando a articulação rítmica está no modo métrico. Finalmente, incluiu-se um “índice de ruído” na saída FM, que consiste na multiplicação do sinal de saída por um sinal aleatório limitado por uma banda cuja largura se pode definir na caixa numérica adjacente (os valores mais baixos geram menos ruído no sinal).

A secção de controlo do ritmo não sofreu nenhuma alteração de maior, à excepção de que os modos rítmicos “métrico” e “brown” alteram respectivamente a velocidade dos eventos MIDI e a duração dos eventos Síntese FM. Na secção da fórmula geradora, fizeram-se somente melhoramentos de ordem *estética* com vista a tornar clara a secção — a fórmula da função logística aparece agora em linguagem matemática (em vez da linguagem quase *críptica* em que tinha de ser expressa no objecto **expr**) e todo resto foi ocultado à excepção dos parâmetros *lambda*,  $x_n$ , saída da função ( $x_{n+1}$ ) e a sua representação gráfica.

Temos agora um sistema que produz um resultado sonoro algo complexo, que permite a manipulação de vários parâmetros de ordem metacomposicional para a criação de uma pequena peça (ou secção de uma peça) composta em tempo real. Precisaríamos agora de explorar bem o sistema para encontrar um possível algoritmo como o apresentado anteriormente (p. 36) que articulasse esta peça no tempo.

Haveria muitos mais refinamentos a fazer no programa se tal se entendesse. Ficam aqui algumas sugestões:

- 1- Em vez de *mapear* a função logística no espaço cromático, fazer corresponder a saída da função a índices de uma tabela contendo várias oitavas de uma escala ou uma coleção de alturas extensa (e.g. o espectro de um som);
- 2- Permitir ao utilizador do sistema alterar os gráficos de modulação e envolvente dos eventos de síntese FM (cf. Exemplo 5), facultando esses gráficos no *interface* do utilizador;
- 3- Tornar as alturas geradas na síntese FM independentes da saída MIDI, assim como criar uma articulação rítmica diferente para a síntese FM;
- 4- Fazer com que a função controle outras formas de síntese (e.g. síntese granular);

- 5- Expandir o programa para *performance* em dança interactiva, criando um *patch* que analisa um/a bailarino/a em movimento através de uma câmara, mapeando os parâmetros de análise do movimento para os parâmetros de controlo musical que foram definidos;
- 6- Permitir o controlo do *patch* externamente por MIDI, de forma a simplificar o acesso aos parâmetros;
- 7- Criar *presets* com combinações de parâmetros para poder alterar muitos parâmetros simultaneamente.

### 3.3. Síntese

Antes de passar à conclusão deste texto-lição seria bom relembrarmos agora, em jeito de síntese, como este trabalho operacionaliza alguns dos conceitos expressos na secção anterior:

*O compositor como piloto* (Xenakis, 1992) — neste sistema, um compositor viaja dentro de um universo sonoro, premindo “botões” e explorando lugares possíveis fornecidos pelo sistema;

*Manifold* (Tipei, 1989) — este programa é de certa forma um *manifold* pois não há modificação nos dados de entrada: o sistema produz eventos MIDI e de síntese sonora FM a partir de vários algoritmos generativos cujos parâmetros são alterados durante a sua execução. Em vez disso ser feito automaticamente por uma rotina de computador, é realizado por um humano durante a execução do programa em interacção com este. Contudo, o sistema é fechado, pois os algoritmos não podem ser reescritos durante a execução do programa. Outro aspecto é também importante na comparação ao conceito de manifold é o facto de ser impossível criar duas versões iguais da peça que possamos realizar com ele;

*Composição por refinamento* (Rowe, 1993) — o processo de desenvolvimento deste sistema é um processo interativo de refinamento que extravasa para a concepção e execução da peça;

*Metacomposição* (Taube, 2004) — o trabalho com o sistema desenvolve-se a um nível abstracto acima do nível composicional (nível metacomposicional), pois estamos a intervir sobre processos compostionais que estão a decorrer;

*Composição em tempo real/Interactiva* (Chadabe, 1997; Essl, 1998, 2007b) — Este sistema promove a interacção e não deverá por isso ser considerado um “instrumento.” Nos instrumentos/hiperinstrumentos, há sempre uma resposta determinística às acções do instrumentista. Caso contrário, era impossível desenvolver uma prática instrumental. Como mencionei acima na formulação do conceito, os sistemas musicais interactivos para composição em tempo real aplicam o *paradigma do executante* (Rowe, 1993). Ao responder não-deterministicamente, o sistema promove o diálogo, a *interacção*. Nós podemos ter uma *ideia* da resposta do sistema às nossas acções (aspecto fundamental para desenvolver um resultado minimamente previsível na sua articulação temporal). Contudo, há constantemente um aspecto de surpresa que promove o diálogo. .

#### 4. Conclusão

Nesta lição apresentou-se um conceito pessoal, em desenvolvimento, de composição em tempo real. Este conceito foi definido como sendo *uma prática composicional utilizando sistemas musicais interactivos, em que algoritmos generativos com um comportamento estocástico são utilizados e não sendo transformados pelo compositor no decurso da execução de uma peça*. O conceito de composição em tempo real foi formulado a partir de experiências pessoais recentes do domínio da composição, e também pela observação de práticas musicais que emergiram desde meados dos anos 80 e que se intensificaram a partir do começo do milénio.

Antes de formular a definição, contextualizei esta prática na história da composição algorítmica por computador, identificando selectivamente pessoas, conceitos e desenvolvimentos tecnológicos que, na minha opinião, contribuíram inegavelmente para o desenvolvimento desta prática na sua forma actual.

Distinguiram-se dois tipos de sistemas que promovem a actividade composicional em tempo real — sistemas orientados para utilizadores comuns e sistemas orientados para especialistas. Deram-se alguns exemplos possíveis dos sistemas actualmente existentes quer para utilizadores comuns quer para especialistas.

Exemplificou-se a criação de um destes sistemas para ser usado por especialistas utilizando o ambiente Max/MSP, e “forçando,” com um fim puramente didáctico, a utilização de quatro aspectos importantes no trabalho em composição algorítmica neste ambiente de programação: (1) a implementação de um algoritmo conhecido; (2) a utilização de uma biblioteca que não faz parte da sua distribuição original; (3) utilização e modificação de partes de *patches* que já são distribuídos com o Max; (4) a utilização de um objecto externo em codificado C originalmente por mim.

Finalmente, explicou-se sucintamente como este trabalho operacionaliza conceitos apresentados na secção de discussão teórica e conceptual.

Penso então que o conceito de composição em tempo real ficou claramente definido e contextualizado, através da discussão de carácter teórico e conceptual e do exemplo dado. Penso também que ficou claro como os sistemas musicais interactivos abrem novas perspectivas fascinantes no trabalho em composição.

A criação de uma obra que existe como um programa de computador que tem um carácter intrinsecamente mutante, uma vez que é irrepetível de apresentação em apresentação, além de formular problemas que têm a ver com o seu formato de apresentação, com a sua sustentabilidade para lá do facto de que a tecnologia se torna rapidamente obsoleta, põe outros que vão à essência da própria música e de como estamos habituados a ouvi-la e praticá-la. Há pelo menos dois aspectos merecem ser abordados aprofundadamente em futuros trabalhos neste campo.

Um deles é o problema da *passagem musical*. A nossa cultura desenvolveu de forma avançadíssima a *passagem musical* — os compositores passam horas, dias, ou meses a reescrever *aquela passagem musical* que só *daquela* forma faz sentido na peça; os intérpretes passam semanas, meses, ou anos a estudar *aquela passagem musical* que só executada *daquela* forma é que dá sentido à sua interpretação; os ouvintes, por sua vez, relembram a música e estabelecem com ela uma relação afectiva por causa *daquela passagem musical*, que adoram ouvir *naquele momento* da peça e até gostam de a comparar com outras interpretações. Numa música que não se inscreve no tempo desta

forma, como é que os três entes principais da prática musical tradicional (compositor, intérprete e público) se vão relacionar com ela?

O outro problema é o problema da forma musical. Como é que se dá *forma* (no sentido tradicional) a uma peça que é composta em tempo real? Será que conseguimos desenvolver estratégias de audição e de composição em tempo real que consigam articular formalmente uma peça com o mesmo sucesso de uma sonata ou sinfonia? Será que há estratégias de articulação formal de média, grande escala para descobrir? Será que vamos desenvolver novos hábitos de audição para esta música?

Todas estas perguntas que formulei, às quais me abstenho de tentar responder por agora, ficam no ar como uma sugestão para trabalho futuro nesta área. Estes problemas são pertinentes e requerem uma reflexão futura profunda. Fascinantes? Definitivamente!

### Referências:

- About Pure Data.* (s.d.). Acedido a 30/03/2008 em <http://puredata.info/>
- Ariza, C. (s.d.). *Algorithmic.net: Algorithmic composition resources | Main.* Acedido a 03/04/2008 em <http://www.flexatone.net/algoNet/>
- Assayag, G., Agon, C., et al. (2008). Open Music (versão 5.2.1) [programa de computador e manual]. Acedido a 30/03/2008 em <http://sourceforge.net/projects/ircam-openmusic/>
- Barlow, C. (1980). Bus journey to parametron. *Feedback Papers 21-23.* Colónia: Feedback-Studio-Verlag Köln.
- Barlow, C. (2001). AUTOBUSK: A real-time pitch & rhythm generator. *Report No. 44, Oktober 2001.* Mainz: Bereich Musikinformatik, Musikwissenschaftliches Institut Johannes Gutenberg – Universität Mainz.
- Barlow, C. (2005). *On Musicquantics.* (Trad. J. Schwartz, D. Richards e C. Barlow. Ed. C. Barlow). Manuscrito não publicado
- Berg, P. (2007). AC Toolbox (versão 4.4.2) [programa de computador e manual]. Acedido a 30/03/2008 em <http://www.koncon.nl/ACToolbox/>
- Bökesoy, S. (2004). Stochos [programa de computador e manual]. Acedido a 29/03/2008 em <http://www.doncrack.com/freeware/downloads.php?id=3746/software/Stochos/>
- Chadabe, J. (1997). *Electric sound: The past and promise of electronic music.* Englewood Cliffs, NJ: Prentice-Hall.
- Chuck:Strongly-Timed Concurrent, and On-the-fly Audio Programming Language.*(s.d.). Chuck:Strongly-Timed Concurrent, and On-the-fly Audio Programming Language (versão 1.2.2.1) [programa de computador e manual]. Acedido a 29/03/2008
- Cope, D. (1996). *Experiments in musical intelligence.* Madison, WI: A-R Editions.
- Cope, D. (2005). *Computer models of musical creativity.* Cambridge, MA: MIT Press.
- Cycling '74. (2007a). M (versão 2.7.2) [programa de computador e manual]. Acedido a 25/03/2008 em <http://cycling74.com/downloads/M>
- Cycling '74. (2007b). Max/MSP (versão 4.6.3) [programa de computador e manual]. Acedido a 31/03/2008 em <http://cycling74.com/downloads/maxmsp>
- DeMarco, D. (1998). *Experiments in Musical Intelligence: Back from the dead?.* Acedido a 30/03/2008 em <http://cem.colorado.edu/archives/sp1998/deanna.html>
- Dixon, S. (2007). *Digital Performance: New Technologies in Theatre, Dance, Performance Art and Installation.* Cambridge, MA: MIT Press

- Fux, J. J. (1965). *The study of counterpoint from Johann Joseph Fux's Gradus ad Parnassum*. (Trad. e ed. A. Mann). New York: Norton (Orginal publicado em 1725).
- Essl, K. (2007a). RTC-lib (versão 4.6.1)[biblioteca para Max/MSP/Jitter e manual]. Acedido a 28/03/2008 em <http://www.essl.at/works/rtc.html>
- Essl, K. (2007b). Algorithmic composition. Em Collins, N. e d'Escrivan, J. *Cambridge Companion to Electronic Music* (pp.107-125). Cambridge: Cambridge UP.
- Essl, K. (2007c). Lexicon Sonate (versão 3.2.) [obra musical para piano Yamaha Disklavier controlado por computador]. Acedido a 29/03/2008 em <http://www.essl.at/works/Lexikon-Sonate.html#ava>
- Essl, K. & Günther, B. (1998). Realtime Composition. Musik diesseits der Schrift. Em G. Nauck (Ed.) *Positionen. Beiträge zur neuen Musik Nr. 36*. Acedido a 29/03/2008 em <http://www.essl.at/bibliogr/realtme-comp.html>
- Gleetchplug Design. (2008). OM (versão 2.0) [programa de computador]. Acedido a 29/03/2008 em <http://nuke.gleetchplug.com/Products/OM/tabid/64/Default.aspx>
- Gottfried Michael Koenig. (s.d.). Acedido a 26/03/2008 em <http://www.koenigproject.nl/indexe.htm>
- Guedes, C. (2005). *Mapping movement to musical rhythm: A study in interactive dance*. Tese de doutoramento, New York University. (UMI nº 3166525)
- Guedes,C., Ula li & Woolford, K. (2003). Cor: um projecto audiovisual interactivo [instalação multimédia].
- Harley, J. (2004). *Xenakis: His life in music*. New York: Routledge.
- Hiller, L. & Isaacson, I. (1959). *Experimental music: Composition with an electronic computer*. New York: McGraw-Hill
- Hiller, L. (1989). Quartet No. 4 for Strings 'Illiac Suite'. *Lejaren Hiller—Computer Music Retrospective (1957-1985)* [Notas ao CD]. Mainz: Wergo WER 60128-50.
- ixi audio. (2007). IxiQuarks (versão 4) [programa de computador e manual]. Acedido a 29/03/2008 em <http://www.ixi-audio.net/content/software.html>
- Laurson, M., Kuuskankare, M. & Norilo, K. (2008). PWGL (versão1.0 beta(rc10)) [programa de computador e manual]. Acedido a 30/03/2008 em <http://www2.siba.fi/PWGL/>
- Machover, T., & Chung, J. (1989). HyperInstruments: Musically intelligent and interactive performance and creativity systems. *Proceedings of the International Computer Music Conference*, 186-190.
- Mathews, M. (2000). Preface. Em R. Boulanger (Ed.), *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. Cambridge, MA: MIT Press.

- Mathews, M., Miller, J., Moore, R. F., Pierce, J. & Risset, J.C. (1969). *The technology of computer music*. Cambridge, MA: MIT Press
- May, R. (1976). Simple mathematical models with very complicated dynamics. *Nature* 261, 459-467
- Morgan, R. (1991). *Twentieth century music*. New York: Norton.
- Mozart's Musikalisches Würfespiel*. (s.d.). Acedido a 24/03/2008 em <http://sunsite.univie.ac.at/Mozart/dice/>
- Nintendo (2005). Electroplankton [jogo de computador para a consola Nintendo DS]. Acedido a 29/03/2008 em <http://electroplankton.nintendods.com/flash.html>
- Penha, R. (2007a). Políssonos [programa de computador]
- Penha, R. (2007b). Digital Jam [programa de computador]
- Penha, R. (2008). Towards a free, open source and cross-platform software suite for approaching music and sound design. Manuscrito submetido para a *International Computer Music Conference*
- Pennycook, B. (2008). Who will turn the knobs when I die?. Manuscrito submetido para publicação na revista *Organized Sound*
- Puckette, M., et al (s.d.). Pure Data [programa de computador e manual]. Acedido a 30/03/2008 em <http://puredata.info/downloads>
- Roads, C. (1996). *The computer music tutorial*. Cambridge, MA: MIT Press.
- Roads, C. (2001). *Microsound*. Cambridge, MA: MIT Press.
- Rowe, R. (1993). *Interactive music systems: Machine listening and composing*. Cambridge, MA: MIT Press.
- Rowe, R. (2001). *Machine Musicianship*. Cambridge, MA: MIT Press.
- Russcol, H. (1972). *The liberation of sound: An Introduction to electronic music*. Englewood Cliffs, NJ: Prentice-Hall.
- SuperCollider* (s.d.). SuperCollider (versão 3.2, 2008)[programa de computador e manual]. Acedido a 29/03/2008 em <http://supercollider.sourceforge.net/>
- Taube, H. K. (2004). *Notes from the metalevel: Introduction to algorithmic music composition*. Londres: Taylor & Francis.
- Taube, H. K. (2007). Common Music (versão 2.10) [programa de computador e manual]. Acedido a 30/03/2008 em <http://commonmusic.sourceforge.net/doc/cm.html>
- Tipei, S. (1989). Manifold compositions: A (super)computer-assisted composition experiment in progress. *Proceedings of the International Computer Music Conference*, 324-327
- Toplap. (s.d.). *Historical performances — Toplap*. Acedido a 31/03/2008 em <http://toplaph.org/index.php/HistoricalPerformances>

- Toplap. (s.d.). *Main page— Toplap*. Acedido a 28/03/2008 em [http://toplap.org/index.php/Main\\_Page](http://toplap.org/index.php/Main_Page)
- Truax, B. (s.d.). *POD & PODX System chronology*. Acedido a 30/03/2008 em <http://www.sfu.ca/~truax/pod.html>
- Van Ransbeeck, S. (2007). Stockwatch [programa de computador e manual]. Acedido a 29/03/2008 em <http://www.filemojo.com/018385241811973/Stockwatch.zip>
- Wang, G. & Cook. P. (2003). ChucK: A Concurrent, On-the-fly, Audio Programming Language. *Proceedings of the International Computer Music Conference*
- Wang, G. & Cook. P. (2004). On-the-fly Programming: Using Code as an Expressive Musical Instrument. *Proceedings of the 2004 International Conference on New Interfaces for Musical Expression (NIME)*
- Wikipedia (s.d.). *Brownian Motion* Acedido a 02/04/2008 em [http://en.wikipedia.org/wiki/Brownian\\_motion#Intuitive\\_metaphor\\_for\\_Brownian\\_motion](http://en.wikipedia.org/wiki/Brownian_motion#Intuitive_metaphor_for_Brownian_motion)
- Wikipedia (s.d.). *Max (Software)*. Acedido a 27/03/2008 em [http://en.wikipedia.org/wiki/Max\\_%28software%29](http://en.wikipedia.org/wiki/Max_%28software%29)
- Wikipedia (s.d. ). *Musikalisches Würfespiel*. Acedido a 24/03/2008 em [http://en.wikipedia.org/wiki/Musikalisches\\_Würfespiel](http://en.wikipedia.org/wiki/Musikalisches_Würfespiel)
- Wikipedia (s.d.). *SuperCollider*. Acedido a 29/03/2008 em <http://en.wikipedia.org/wiki/SuperCollider>
- Winkler, T. (1998). *Composing interactive music: Techniques and ideas using Max*. Cambridge, MA: MIT Press.
- Xenakis, I. (1992). *Formalized music: Thought and mathematics in music* (Ed. Rev.). Hillsdale, NY: Pendragon Press