UNIVERSIDADE DE BRASÍLIA

INSTITUTO DE ARTES

DEPARTAMENTO DE ARTES VISUAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM ARTE

ALEXANDRE GALVÃO DE QUEIROZ RANGEL

O artista como desenvolvedor de sistemas computacionais: experiências audiovisuais

Brasília

2019

ALEXANDRE GALVÃO DE QUEIROZ RANGEL

O artista como desenvolvedor de sistemas computacionais: experiências audiovisuais

Tese apresentada ao Programa de Pós-Graduação em Arte do Instituto de Artes da Universidade de Brasília, para obtenção do título de Doutor em Arte.

Área de concentração: Arte e Tecnologia.

Orientador: Prof. Dr. Marcus Santos Mota

Brasília

2019

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

ATA DA DEFESA DE TESE
CURSO DE DOUTORADO EM ARTE /UNB – 2019.

Ao vigésimo oitavo dia do mês de outubro de dois mil e dezenove, às nove horas, realizou-se no Escola de Informática, sala 426 - ICC Central a Sessão Pública de Defesa de Tese do (a) aluno (a) **Alexandre Galvão de Queiroz Rangel**, matrícula nº 16/0162157, intitulada: "O artista como desenvolvedor de sistemas computacionais : experiências audiovisuais". A comissão examinadora composta pelos professores: Marcus Santos Mota (VIS/UnB) - Suzete Venturelli (VIS/UnB) – Membro Interno, - Lúcio França Teles (FE/UnB) – Membro externo – e – André de Souza Parente – Membro externo. Antenor Ferreira Côrrea (VIS/UnB) – Suplente interno e Célia Kinuko Matsunaga Higawa (DIN/UnB) – Suplente externo. Após arguir a banca deliberou pela aprovação.

ASSINATURAS:

x Marcus Santos Mota
Professor (a) Dr. (a). Marcus Santos Mota

Suzete Venturelli
Professor (a) Dr. (a). Suzete Venturelli

Lucio Teles
Professor (a) Dr. (a). Lúcio França Teles

Professor (a) Dr. (a). André de Souza Parente

Proclamados os resultados pelo (a) Professor (a) Dr. (a). Marcus Santos Mota, Presidente da Banca Examinadora, foram encerrados os trabalhos, e para constar eu, Marcus Santos Mota, lavrei a presente ata, que assino.

Brasília-DF, **segunda-feira, outubro 28, 2019**

Marcus Santos Mota
Professor (a) Dr. (a). Marcus Santos Mota
PRESIDENTE DA BANCA EXAMINADORA

Dedicatória

Às mulheres das novas gerações Janaína, Gabriela, Flora e Sol.

Aos pais Arlete e Rangel e aos avós Luzia e Galvão, pelo incentivo aos estudos e ao desenvolvimento pessoal.

A Giovana Ungarelli, pelo companheirismo durante toda minha formação acadêmica.

A Claudio Delamare e Ilya Eric Lee, grandes mestres (*in memoriam*).

Agradecimentos

A Wagner Barja, Arthur Cordeiro e Jay Barros pelas longevas parcerias.

A Ton Roosendaal e Sam Aaron, pelas iniciativas e esforços continuados de desenvolvimento de *softwares* criativos (*Blender* e *Sonic Pi*).

A Prof. André Parente, Prof. Lúcio Teles, Prof. Marcus Mota, Profa. Suzete Venturelli, Ariel Barja, e Glória Marques pelas valiosas contribuições a esta pesquisa.

“Eu passo a me conhecer através do que faço, porque na realidade eu não sei o que eu sou. Porque se é invenção, eu não posso saber! Se eu já soubesse o que seriam essas coisas, elas não seriam invenção.”

Hélio Oiticica

Resumo

Esta tese tem por objeto apresentar e investigar ligações tecnológicas digitais entre ambientes de desenvolvimento multimídia e a transmissão do conhecimento por meio da educação envolvendo a produção visual e sonora. Pretende demonstrar métodos de inserção poética nas engrenagens da criação de arte computacional, com foco em uso e desenvolvimento de *software* livre.

A pesquisa trata da minha trajetória de criação poética, desenvolvimento de *software*, e estudo e ensino de sistemas computacionais como forma de expressão artística. O que venho a chamar de sistemas são programas de computador com ênfase em técnicas de *remix* e geração processual imagética e sonora. Tenho sempre em vista o uso subversivo de *softwares* e tecnologias para a produção de conexões imprevistas.

Compreendo o processo e as ferramentas de criação de sistemas de arte eletrônica no âmbito do seu potencial técnico na programação de computadores, tendo como fio condutor a criação artística.

O trabalho é apresentado por meio de exemplos de sistemas de produção audiovisual e suas aplicações no conjunto minha da obra relacionada ao contexto da arte generativa, nos últimos dez anos. Detalha também uma seleção de realizações de outros artistas considerados seminais para o desenvolvimento do campo da arte computacional.

Com vistas a potencializar construções computacionais, aprofundo esta investigação sobre a própria produção usada como estudo de caso, numa contribuição para os processos criativos da arte-educação e formulação de projetos formativos no ambiente da cultura digital contemporânea.

Palavras-chave: sistemas computacionais, programação, *live coding*, acaso, novas mídias, audiovisual, videoarte, música experimental, interatividade, educação, *software* livre, *Blender*, *Sonic Pi*.

Abstract

This thesis aims to present and investigate digital technological links between multimedia development environments and the transmission of knowledge through education using experiences of visual and sound art production. It demonstrates methods of poetic insertion in the gears of computational art creation, focusing on the use and development of free software.

The research deals with my trajectory of poetic creation, software development, study and teaching of computer systems as a form of artistic expression. What I have come to call systems are computer programs with an emphasis on remix techniques and image and sound procedural generation, always with a view to subversive use of software and technologies in the production of unforeseen connections.

I understand the process and tools of creating electronic art systems as part of their technical potential in computer programming, with the artistic creation as a guiding thread.

The work is presented through examples of audiovisual production systems and their application in my body of work related to the context of generative art, in the last ten years. It also details a selection of achievements of other artists considered seminal for the development of the field of computational art.

With a view to enhancing computational constructions, I deepen this investigation into the production itself used as a case study, in a contribution to the creative processes of art education and the formulation of formative projects in the environment of contemporary digital culture.

Keywords: computer systems, programming, live coding, chance, new media, audiovisual, video art, experimental music, interactivity, education, free software, *Blender*, *Sonic Pi*.

Sumário

Lista de Imagens

Lista de Gráficos

Lista de Tabelas

Lista Cronológica de Obras

Título Ano / Link		Técnica / Formato	Pág.
Floresta		Vídeo generativo	
2009	< www.youtube.com/watch?v=uI36dl6y_CI >		127
Bichos Impossíveis		Sistema audiovisual interativo	
2006 2014	< www.youtube.com/watch?v=8rRNG1Ua9Mw > < www.youtube.com/watch?v=a67ExTPhfjY >		141
Eixo X		Sistema audiovisual interativo	
2010	< www.youtube.com/watch?v=k4uxH5QdagU >		138
Homenagem a Wlademir Dias-Pino		Live Coding / Vídeo	
2013	< www.youtube.com/watch?v=S32Z1j0Mg1E >		48
muSEU feito na unha		Videoescultura generativa	
2013	< www.youtube.com/watch?v=0go0Bf7wgmo >		117
MeditaMáquina		Live Coding / Performance	
2013	< www.youtube.com/watch?v=qyvysqdWoKE >		50
Weekly Beats 2014		52 Obras audiovisuais	
2014	< www.alexandrerangel.art.br/musica-wb2014.html >		70
Traquitana Audiovisual Interativa		Sistema audiovisual interativo	
2015	< www.youtube.com/watch?v=ZD0Ln-9dKLo >		132
Sábio ao Contrário		Performance audiovisual	
2015	< www.alexandrerangel.art.br/sabioaocontrario.html >		75
CODE MUZIK		6 oficinas de live coding	
2016	< www.quasecinema.org/sonicpi.html >		59

Weekly Beats 2016	52 composições sonoras	
2016	< www.alexandrerangel.art.br/musica-wb2016.html >	74
Hipertexto da Corrupção	Sistema audiovisual online	
2017	< www.alexandrerangel.art.br/hipertextodacorrupcao.html >	113
Brasília 1960-2160	Performance audiovisual	
2017	< www.youtube.com/watch?v=YirEN_mUOjc >	77
ALGORAVE	Performances audiovisuais online	
2017 2018	< www.youtube.com/watch?v=Z6-eC1bEWzk > < www.youtube.com/watch?v=updb3O6zOIY >	84
Emoção Artificial	Sistema audiovisual	
2018	< www.alexandrerangel.art.br/emocaoartificial.html >	78
Memórias Corrompidas	Sistema audiovisual interativo	
2018	< www.alexandrerangel.art.br/memoriascorrompidas.html >	118
Esculturas Quânticas	Sistema audiovisual interativo	
2018	< www.alexandrerangel.art.br/esculturasquanticas.html >	100
TOPLAP 15 - Slicing sonic thoughts on a Turing machine	Performance audiovisual online	
2019	< www.youtube.com/watch?v=_r0PLWJyRDk >	81
Tocando Brasília	Sistema audiovisual interativo	
2019	< www.alexandrerangel.art.br/tocandobrasilia.html >	86
Cubos em Dança	Sistema audiovisual dinâmico	
2019	< www.alexandrerangel.art.br/cubedance.html >	107

Tabela 1. Lista cronológica de obras citadas na tese.

Introdução

Esta tese descreve e aprofunda uma discussão sobre os processos criativos e o ferramental de desenvolvimento de sistemas audiovisuais computacionais generativos. A metodologia utilizada para a pesquisa foi autoetnográfica, pautada por indagações sobre os procedimentos de construção de obras de minha autoria. A tese apresentada é resultado da investigação e produção artística dos últimos dez anos (de 2009 a 2019); nela serão detalhadas técnicas do universo de sistemas computacionais e do *live coding* (a performance de programação de *softwares* de música e artes visuais em tempo real). Foram escolhidos obras e ambientes de produção audiovisual com base em filtros teóricos e artísticos, num esforço de compreender a programação de *software* como uma extensão das habilidades humanas de argumentação e organização de material audiovisual.

O problema de pesquisa abordado é a compreensão e expressão de como funcionam os processos criativos de obras computacionais multimídia, traçando-se caminhos entre conceito e execução. Apresento uma pesquisa sobre a hipótese de que os sistemas computacionais servem como extensões do pensamento e do corpo expressivo do artista em experiências audiovisuais. Faço uma abordagem mais ampla do que o conceito tradicional de programador, abrangendo o que denominei de sistemas: formas de tradução poética e de pensamento lógico baseado em regras que usam variados tipos de programação de computador. Interesso-me na produção de *software* como matéria-prima para experimentações artísticas, e no que constitui os métodos criativos do artista programador: pensamento não tradicional, estrutura do pensamento, diferencial do pensamento; com foco nos processos de ideação e realização.

O escopo da discussão abrange a expressão musical, visual e poética da criação de arte enquanto código. A inovação da pesquisa é preencher a lacuna de conhecimento sobre os processos de criação do artista programador. A tese não é só auto-referenciada, mas possui objetivo educativo, com criação e difusão de informação e conhecimento, e conexões presenciais e em rede (Capítulo 2, Seção 2.3.1).

Destaco técnicas e sistemas computacionais e o momento mundial de interesse na prática de *live coding*, assim como o aparecimento constante de novos sistemas de prática de programação ao vivo, que usam linguagens textuais ou interfaces

gráficas de criação. Num discurso sobre o fazer artístico usando o computador e suas especificidades, esta pesquisa detalha o processo criativo em meios audiovisuais digitais, com foco na produção por meio de criação de programas de computador. A metodologia escolhida foi a realização e análise de oficinas presenciais abertas ao público e a análise de obras criadas no período entre os anos de 2013 e 2019. Chamo atenção para o poder das práticas ao vivo, presenciais, para a interação em tempo real com o receptor da mensagem, para a múltipla dimensionalidade do aqui e agora e da possibilidade de transmissão ao vivo. Alguns dos aspectos do processo artístico observados pela pesquisa são a produção poética e a abordagem prática da produção da música eletrônica, arte visual e computação gráfica.

Minha relação com o tema de criação de arte com código com computador vem desde a minha graduação em Artes Plásticas pela Universidade de Brasília (2009), quando desenvolvi um *software* livre para edição de vídeo ao vivo, no contexto de *VJ (Visual Jockey*, o criador de visuais eletrônicos em tempo real). Batizei o *software* de *Quase-Cinema*: uma homenagem aos artistas Hélio Oiticica e Neville D'Almeida, que nos anos 1960 criaram uma série de obras multimídia intituladas de *Quasi-Cinemas*. O projeto *Quase-Cinema* foi contemplado com a Bolsa Funarte de Desenvolvimento Artístico, ocasião em que tive a oportunidade de desenvolver a segunda versão do programa, distribuído como *software* livre. Já no meu mestrado em Artes, com com foco em Arte-Educação, desenvolvi métodos de aplicação do *Quase-Cinema* para capacitação e autonomia técnica em criação audiovisual ao vivo. As oficinas do *Quase-Cinema* foram ministradas no Brasil, Argentina, Espanha, Estados Unidos, Dinamarca e Taiwan. Desde então, venho me dedicando à prática de composição audiovisual improvisada – interessam-me o formato de criação e as possibilidades de derivações a partir sistemas abertos ao *remix* e ao acaso –, abraçando a contradição entre as regras e o inesperado, entre o previsto e o imprevisto; e transgressão da ordem do sistema, em busca da emergência poética do erro.

Apresento uma forma de subversão da precisão dos algoritmos com a inserção da inevitabilidade do acaso. Pretendo discutir os aspectos do *live coding* de compreensão da linguagem pelo computador e compreensão do computador por parte do público. Também será levantada a questão da criação paralela em três áreas: música, arte visual e sistema de *software*. Levantarei questões a respeito da necessidade de saber programar *software* para explorar o processo de criação com *live coding*, levando em consideração que algumas linguagens de *live coding* têm, inclusive, um caráter didático, objetivando, além da criação artística, o ensino de conceitos de programação de computadores. O sistema *Sonic*

Pi, a linguagem estudada mais profundamente nesta tese, é direcionada para crianças a partir de 10 anos de idade, sem conhecimento prévio de construção de *software*.

O que é um programa que não entrega um resultado completo, quando o processo é o mais importante – criando diálogo com o conceito de obra aberta da arte contemporânea? Qual a articulação poética e artística com criadores que jogam com o acaso, como John Cage e Brian Eno? Como é processo criativo do artista nessa interface com a arte generativa? Como entra a computação? Como ser, ao invés de limitado, criativo com a formatação da máquina?

"Como resultado da adoção da interface gráfica nos computadores, nos anos 1980, o *software* substituiu outras ferramentas e tecnologias para os profissionais criativos e também deu para centenas de milhões de pessoas as habilidades de criar, de manipular, de sequenciar e de compartilhar mídias – mas, será que isso levou à invenção de formas de culturas fundamentalmente novas?" (MANOVICH, p.3) Tento exemplificar exemplos dessa novidade histórica com obras abertas nos sentidos de interpretação, técnica e formato.

A articulação sobre como o processo de *live coding* proporciona a inserção do acaso no sistema, além do sentido mais estrito da própria palavra acaso (números aleatórios), mas com o desenvolvimento do raciocínio exposto na tela e com a possibilidade do erro, tanto da máquina quanto do ser humano. A partir disso, a tese será dividida em três capítulos:

Capítulo 1, que apresenta uma visão do campo da arte computacional, com ênfase em obras de arte generativa e conceitual – campos que trabalham com o potencial das ideias e da aplicação de regras para o direcionamento à obra de arte ou performance artística.

Capítulos 2 e 3, que colocam o foco da pesquisa em técnicas de criação de obras sonoras e visuais, respectivamente – associando métodos e ferramentas de produção com obras de arte de minha autoria. A maioria dos processos explorados é baseada na produção de sistemas, ou seja, conjuntos de regras ou códigos de computador que direcionam a máquina a cocriar com o artista.

Capítulo 4, que embarca em detalhamentos e discussões sobre a possibilidade de interação do performer e do público com sistemas dinâmicos, vistos como obras abertas e dentro de uma proposta de tornar a interação com código tátil e, posteriormente, etéreo.

Os relatos das experiências de criação estão organizados a partir da perspectiva e possibilidades de cada ferramenta. Cada ambiente criativo é apresentado e seguido de obras desenvolvidas essencialmente com este sistema.

Capítulo 1. Arte, Tecnologia e Processo

"Eu estava tentando fazer pinturas que se movessem
e música que ficasse parada."
Brian Eno

Este capítulo apresenta e contextualiza o processo de programação ao vivo *(live coding)* enquanto método de criação artística, tanto musical quanto visual, sob a perspectiva de quem cria obras nesse contexto. Uso como exemplo a minha produção audiovisual realizada na década de 2009 a 2019, na qual utilizei diversas técnicas de programação direcionada à criação de arte generativa computacional. Apresento e questiono o papel dos objetivos e regras, visando o artista enquanto criador de opções e de caminhos de desenvolvimento.

Charles Hutchins, no seu artigo *Live Patch / Live Code*, traça um histórico da definição de programação ao vivo, que inicialmente só englobava o uso de programação em computadores, e agora já engloba outros sistemas onde se podem criar processos de fluxo e decisões. Hutchins argumenta que podem ser incluídos instrumentos musicais, tais como sintetizadores modulares, na lista de ferramentas para programação ao vivo. (HUTCHINS, p.1) Partindo dessa premissa, incluo na minha definição também o uso de ferramentas de programação visual com fluxograma, principalmente o ambiente de criação visual Blender com seu software acessório Animation Nodes e sua conexão com outros programas.

Sobre a formulação de ideias artísticas por meio de código e das novas formas de pensamento e solução de questões derivadas desta prática, aqui concordando com Pierre Lévy, quando percebe que "[...] o pensamento é profundamente histórico, datado e situado, não apenas em seu propósito, mas também em seus procedimentos e modos de ação" (LÉVY, p.95). Expressar-se artisticamente por meio de código é uma tarefa que explicita habilidades de solução de problemas e de questionamento da ferramenta. A obra e a ferramenta de criação são hoje normalmente contemporâneas entre si.

Segundo o artista de novas mídias e educador americano Golan Levin, "Um preceito para mim é que os artistas deveriam ser obrigados a usar os meios de seu tempo, que no nosso caso são o *software* e os circuitos eletrônicos." (LEVIN, 2012). Ou à frente de

seu tempo: em 2001 (era anterior à disseminação dos telefones celulares, hoje onipresentes), Levin criou a obra *Phone Pixel*, com uma interface em um cinema para telefonar para a plateia, criando uma sinfonia de toques especiais e piscadas das telas dos telefones. A obra, vista hoje, remete-nos a temas extremamente atuais, como o abuso de ligações indesejadas para telefones celulares.

Meu histórico de junção da produção de música com as artes visuais vem desde o meu projeto de graduação em artes plásticas, quando desenvolvi o *software* de edição de vídeo ao vivo *Quase-Cinema*. O objetivo do desenvolvimento desse sistema criativo, além de possibilitar a realização da minha própria expressão artística com performances audiovisuais, é produzir arte como processo de troca e capacitação técnica, conforme descrito no relato das Oficinas *CODE MUZIK*, no capítulo 2, item 2.3.1.

Desenvolvo técnicas e processos de *live coding* como ferramentas de composição e performance e transmissão de conhecimento. Nesse formato de produção, a música não é vista como o produto final a ser burilado e refinado, e sim como um processo. Até mesmo o fim do processo pode ser considerado como obra aberta, porque costumeiramente o resultado de uma sessão de programação ao vivo é um sistema vivo, constituído de algoritmos e passível de ser executado e modificado *ad infinitum* – pelo autor ou por outras pessoas, se distribuído juntamente com o seu código-fonte.

Compreendo o código-fonte de programação como um desdobramento da partitura musical. Uma diferença em relação à partitura convencional é a simultaneidade dos processos de composição e execução musical de uma performance. “Vindo de partituras estáticas e formatos de armazenamento determinísticos, movemo-nos para partituras que são escritas e executadas em tempo real, resultando em uma nova forma de música ao vivo: a música generativa.” (MAGNUSSON, 2011, p.23)

Por que a tendência na criação das obras em relação aos algoritmos e à programação ao vivo? “A teoria da composição algorítmica normalmente distingue entre a síntese de partitura (criação de uma composição com a ajuda do computador, normalmente para instrumentos tradicionais) e a síntese sonora (criação de sons computacionais que só podem ser ouvidos por meio de auto-falantes). Essa distinção tem suas raízes na diferenciação tradicional entre partitura e instrumento, mas o ato contínuo gerado pelo computador entre dois sons diferentes é tanto partitura quanto síntese.” (SUPPER, p.1)

A composição e a distribuição da música vem distanciando-se cada vez mais do suporte da partitura, graças aos avanços das tecnologias digitais. Os experimentos de criação com partituras visuais foram etapa importante, que culminou no processo estudado

nesta tese: o da criação, descrição e distribuição de música como linhas de código de programação de software. A descrição de sons por meio de desenhos é um nível de abstração intermediário: nem mais simples, nem mais complexo – somente diferente – do que a descrição de sons com notas em uma partitura, ou no algoritmo de programação computacional.

A apropriação e o *remix* já estão presentes desde os *Bichos* de Lygia Clark e os *Parangolés* de Hélio Oiticica, que só ganhavam vida quando vestidos e mesclados com os corpos do público. "Essas rupturas tropicalistas na arte trazem o foco para o indivíduo, para a vida cotidiana, para o fragmento, para os processos de apropriação, para a reciclagem, para a indústria cultural e para um novo repertório originado das mais diversas mídias de massa." (MELLO, p.78)

Também reconheço como influência os sistemas com desdobramentos em cadeia dos artistas Fischli & Weiss, tal como o filme *The Way things go (O Jeito em que as coisas andam)*, de 1985, no qual uma série de eventos aparentemente caóticos desenrola-se em forma de narrativa, e a magia das máquinas de Rube Goldberg, máquinas hipnotizantes que, sempre nos lembrando da regra da causa e efeito, desencadeiam uma série de eventos preconcebidos em narrativas – lineares –, em direção a um fim certo e sem objetivo prático evidente – uma metáfora da vida?

Desde o advento dos computadores pessoais e *softwares* de criação sonora digital, as fronteiras entre composição, registro de peça musical e performance vêm-se fragmentando, e estes campos entremeiam-se cada vez mais. "Se o termo 'música' é sagrado e reservado para instrumentos dos séculos dezoito e dezenove, podemos substituí-lo por um termo mais significativo: organização de sons." (CAGE, 1961, p.3) Cage, no livro que editou chamado *Notations* (Partituras), mostra exemplos de artistas que desafiam as normas de tempo e espaço físico das notações musicais, como pesquisa e produção além das fronteiras musicais estabelecidas. Como outro exemplo de novos procedimentos e compreensões, chamo atenção para a obra *Solitude*[1] *(Solidão)*, na qual Hans-Christoph Steiner desenvolve uma partitura não convencional que é resultado de processo criativo e fórmula para execução precisa por um *software* específico (o sistema de criação sonora *Pure Data*[2]).

---

[1] Página da obra Solitude, de Hans-Christoph Steiner: < https://at.or.at/hans/solitude >

[2] Site do software Pure Data: < https://puredata.info >

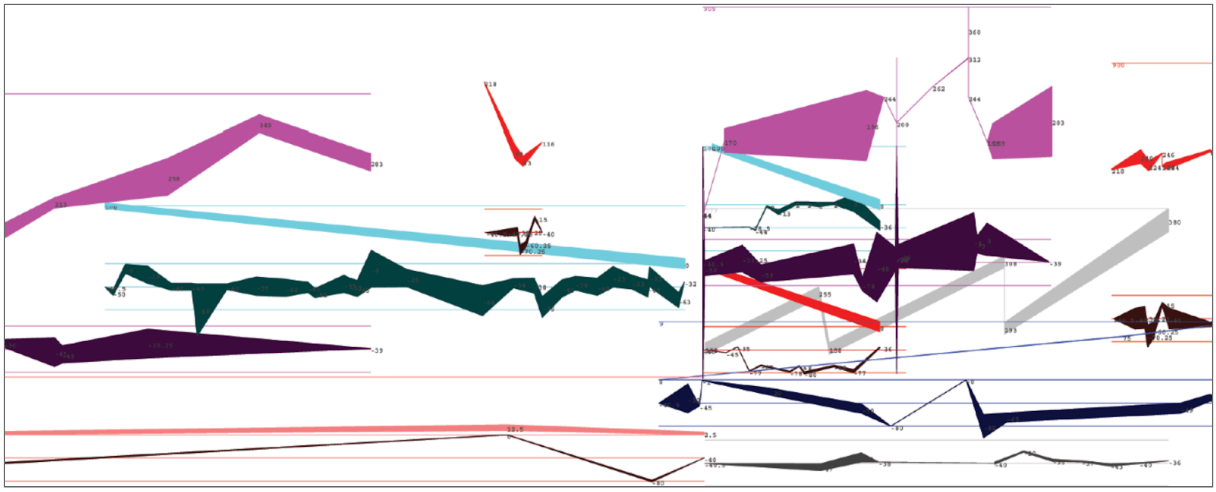


Imagem 1. Trecho da partitura de Solitude, de Hans-Christoph Steiner (2004).

A partitura acima controla a criação de todos os aspectos sonoros da obra. As linhas funcionam como as instruções necessárias para o *software Pure Data* controlar a duração e volume das gravações sonoras.

Observando alguns exemplos de obras e sistemas contemporâneos, podemos perceber que a interface dos *softwares* tem-se misturado com a partitura, cruzando os limites entre composição e performance. Um exemplo claro de sistema que mistura aspectos de registro e performance é o sintetizador *ANS*, que é baseado em leitura sonora de imagens, como ocorre no processo das trilhas sonoras em películas de cinema. O desenvolvedor de *software* Alexander Zolotov criou um *software* livre baseado nessa técnica, chamado de *Virtual ANS*[3]. No sistema *Virtual ANS,* o compositor / *performer* pode desenhar linhas que controlam a imediata síntese sonora. O *software* permite também a sonificação de imagens preexistentes, na forma de arquivos do tipo JPG. O processo é uma releitura, ou desdobramento, do equipamento físico com funções similares de leitura sonora de imagens chamado Sintetizador *ANS*[4], desenvolvido pelo engenheiro russo Alexander Scriabin entre 1937 e 1957.

[3] Página do *software* Virtual ANS: < http://www.warmplace.ru/soft/ans >

[4] Histórico do sintetizador ANS, batizado com as iniciais do seu criador, Alexander Nikolayevich Scriabin: < https://en.wikipedia.org/wiki/ANS_synthesizer >

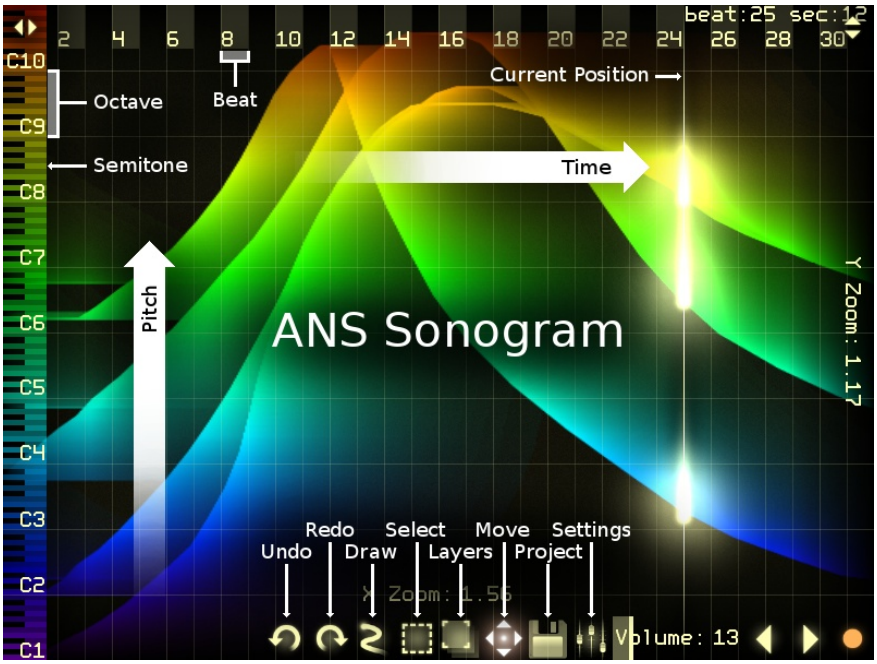


Imagem 2. Interface do software Virtual ANS, mostrando leitura e sonificação de uma imagem.

O *software* livre *online PIXELSYNTH* [5] (2016), da artista Olivia Jack, é outra releitura do sistema *ANS*, desenvolvido com a linguagem *JavaScript*. Nesse tipo de projeto de instrumento, o que chama a atenção é a possibilidade de "manuseio" do som, trazendo o tato para o jogo entre sonoridade e visualidade, e introduzindo dimensões poéticas no contato com a máquina (nesse caso, a superfície de vidro da tela do *tablet*).

"Quando celebramos um artefato tal como uma composição musical, uma pintura, um teorema ou um poema, também estamos celebrando o ato criativo que o trouxe à existência. O artefato resultante de um ato criativo deve ser visto como um convite para o engajamento em um diálogo com o artefato e/ou com o criador e/ou com a cultura e/ou com você mesmo." (COLTON, p.5)

1.2 Arte Generativa e Obra Aberta

Este item descreve o contexto da arte contemporânea no tocante ao campo de interseção entre produção de arte computacional e produção de arte generativa, cuja produção é baseada em regras. Apresento o fazer da programação como um desdobramento do conjunto de práticas de arte baseadas em fórmulas. A pesquisa se enquadra no contexto de criação e expressão artística com *remix* audiovisual. A discussão será sobre traquitanas e engenhocas em criadas código-fonte computacional.

---

[5] *Site* do sistema *online PIXELSYNTH:* < https://ojack.github.io/PIXELSYNTH >

"A arte generativa funciona criando um processo. O processo executável na forma de *software* é a obra generativa. O objetivo do processo é a representação de uma ideia peculiar, definida pela visão criativa subjetiva, no nível abstrato, com a ajuda de algoritmos simbólicos. Os resultados desse processo mostram a ideia através de múltiplos cenários." (SODDU, p. 1) Encaro o processo de criação generativo como um diálogo conceitual entre a regra e o subjetivo – entre o programado, o inesperado e a emergência da invenção. O pesquisador Roy Ascott introduziu nas artes o termo *moist media* (mídia úmida), que significa o território entre o humano e o computacional. Tentou descrever a simbiose entre as práticas relacionadas – a princípio – com o ser humano (sensibilidade, percepção, bom senso) e as práticas relacionadas com o universo digital do *software* (precisão, cálculos rápidos, armazenamento de dados audiovisuais). É nessa área fértil que tento posicionar o processo criativo (diálogo com a máquina) de uma obra computacional: onde os algoritmos podem complementar – ou incentivar – os processos mentais biológicos? Reproduzo a seguir um trecho do Manifesto Generativo[6], escrito por Alex McLean e Ade Ward em 2000.

Manifesto Generativo

1. Atenção aos detalhes que só a música generativa feita à mão pode permitir (você pode se aprofundar em estruturas usando código). Código permite aprofundamento em estruturas criativas.

2. Resultado instantâneo e controle composicional. Nós odiamos esperar. (É inconcebível esperar que sistemas não instantâneos exibam sinais de vida.)

3. Construa e explore ambientes sonoros com ecos do nosso próprio ambiente. (A arte reflete a narrativa humana, o código reflete a atividade humana.)

4. Processo aberto, mentes abertas. (O código é não-ambíguo, não pode ser escondido atrás de obscuridade; nós procuramos abolir a obscuridade das artes.)

5. Usar somente *software* escrito por nós mesmos. O *software* dita o resultado, nós ditamos o *software*. (Autoria não pode concedida para aqueles que não criaram!)

[6] Registro de Manifesto Generativo (2000):
< https://slab.org/the-generative-manifesto-august-2000/ >

A arte generativa é uma ode à obra aberta, conceito fundamental da arte contemporânea e sinal dos tempos em que vivemos, quando o efêmero é presente em todos os meios e fazeres artísticos. “É a evolução do trabalho, as mutações e as transformações que sofre no caminho para lugar nenhum (já que não existe 'resultado final'), é que devemos relacionar a uma obra generativa, sem esperar um estado final ou conclusão. O processo é a conclusão.” (MAGNUSSON, 2002, p.61)

Nos primórdios da automatização do processo de composição musical foram criados vários jogos de dados, espécies de passatempos em eventos sociais, para criação musical em grupos, seja de músicos, seja de leigos. No final do século XVIII, a Europa viu o aparecimento de mais de vinte jogos para criação musical por leigos; alguns deles usavam dados, outros simplesmente pediam aos participantes que inventassem números. Em 1789, Mozart criou um jogo, com dados, para composição musical, hoje existe uma versão desse jogo para sistema operacional *iOS* e uma versão para navegadores de internet[7]. A receita é simples: tacar dois dados 16 vezes e copiar notas de uma tabela, mas podem ser criados mais de 45 milhões de bilhões de combinações de composições com as regras de Mozart!

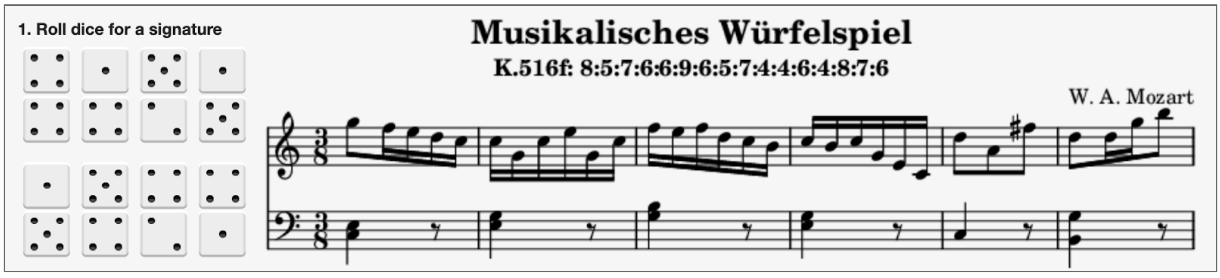


Imagem 3. Jogo de composição de Mozart em versão rodando na internet.

Como artistas que trabalharam com sistemas de regras para a realização dos seus trabalhos, cito Sol LeWitt e Athos Bulcão. O artista norueguês Marius Watz sintetiza “de forma arrogante” – como ele próprio rotula – que “não há forma sem geometria. Não há geometria sem algoritmo. Não há algoritmo sem código”.

Nos primeiros tempos da minha curiosidade como artista e como programador de computador, com 13 anos de idade, fazia programas na linguagem *BASIC* para randomizar caracteres e pontos coloridos de baixa resolução numa televisão ligada a um computador tipo *Apple II*. Pude reviver esse tipo de programa numa videoarte de 2017,

[7] O sistema *Mozart’s Musical Dice* é detalhado e pode ser executado *no site*:
< https://mozart.qvwx.de/index.en.html >

ego.print[8], graças à tecnologia de emulação do *Apple II BASIC* rodando em um navegador de internet atual – sem a necessidade de um computador de 30 anos atrás! Essa obra entra então diretamente no diálogo de preservação das obras de *software* e no de novas mídias.

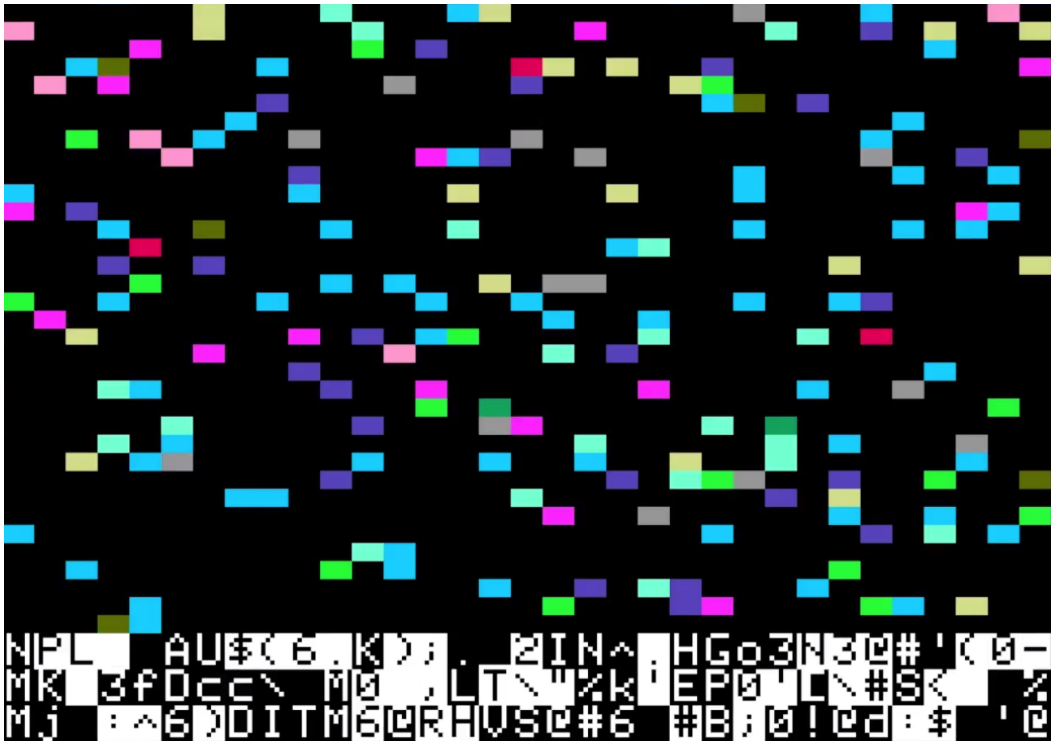


Imagem 4. Frame da obra ego.print.

A obra *ego.print* foi exposta no *#16.ART Congresso Internacional de Arte e Tecnologia*, no Museu Nacional da República, em Brasília, na forma de uma gravação de tela da saída do programa, em *loop* com seis horas de duração. Fascina-me a criação de algumas dezenas de linhas de código como instruções para a criação de vídeos de longas durações. Esses vídeos nunca serão assistidos na íntegra, criando uma conexão temporal única com o ambiente de exposição.

Nos sistemas de *live coding*, uma grande quebra de paradigma de composição é a fuga do padrão vigente em composição musical assistida por computador, no qual se pode ver a obra visualmente organizada em forma de linha do tempo, normalmente com várias trilhas, uma para cada instrumento.

Um algoritmo nada mais é do que uma ou mais instruções de procedimentos, como uma receita de bolo. O mundo da arte contemporânea já apresentou muitos sistemas dessa natureza, tais como obras do grupo Fluxus. A obra *Composição 1960 #10*, do artista

[8] Vídeo completo da obra *ego.print*: < www.youtube.com/watch?v=D4dCMPkuas0 >

minimalista americano Bob Morris (1931-2018), é um exemplo emblemático, resumindo-se a uma frase somente: “Desenhe uma linha reta e siga-a”. Outro exemplo de arte por instruções marcante pela simplicidade é a obra *Xerox Book #1*, realizada pelo artista australiano Ian Burn (1939-1993), em 1968, que indica que uma folha em branco deve ser copiada em uma máquina fotocopiadora 100 vezes. A cada passagem pelo sistema, a cópia capta mais interferência, fazendo o ruído óptico ficar cada vez mais evidente.

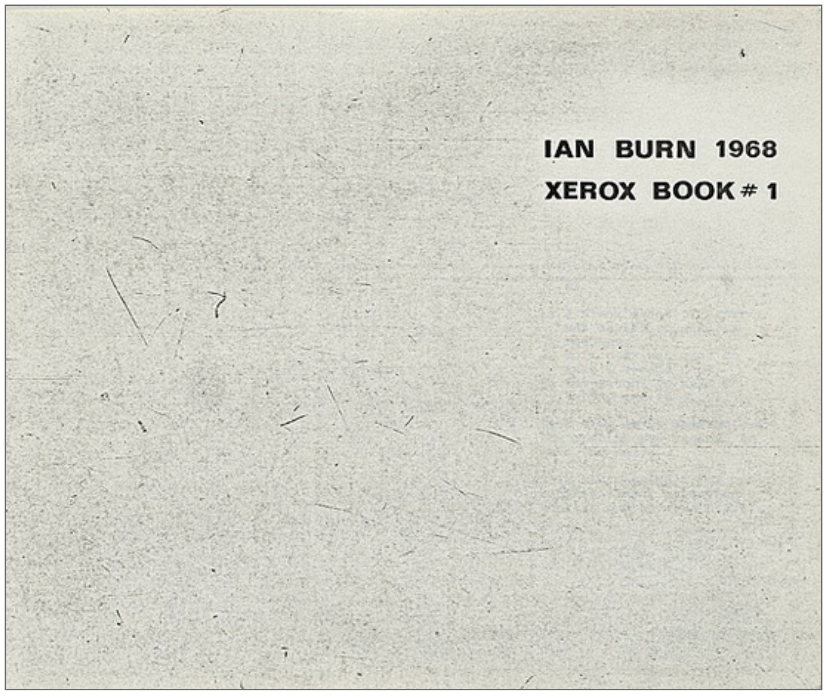


Imagem 5. Página do Xerox Book, Ian Burn (1969). Imagem: Centre For Australian Art.

Segundo Gunnarson (2019), os algoritmos são caixas pretas, mas passíveis de modificações por meio de interrupções e intervenções. A complexidade é alcançada por meio da fusão de partes mais simples, variáveis, e por meio da modificação constante de trechos de código-fonte. Trabalhando diretamente com programação de código, é comum seguir uma abordagem experimental, na qual os procedimentos são tão elusivos quanto o resultado a ser alcançado. O *performer* torna-se parte de uma engrenagem de *feedback* com a máquina.

> “Fronteiras flutuantes e o desenvolvimento contextualizado podem dar forma às condições de manifestação musical e criar um espaço compartilhado entre o compositor e os processos técnicos autônomos. Ambos tornam-se interconectados e se sobrepõem por meio de diferentes funções, desde a tomada de decisões até a criação de

> materiais. Ainda que inseridos numa situação em comum, eles contribuem e dão suporte um ao outro. Essa troca transforma os processos generativos em condutores da atividade criativa, ao invés de simplesmente consequências dela." (GUNNARSON, 2019)

Uma questão importante na prática de *live coding*, onde a partitura tem a forma de texto, é a compreensão desses textos. Ao se observar uma sessão de programação ao vivo, pode-se pensar, em um primeiro momento, que o código é – somente – um conjunto de instruções a serem seguidas pela máquina. Ao se perceberem estilos de construção e palavras-chave conhecidas (sejam elas em inglês ou na língua nativa do público), vai-se tomando consciência do processo criativo do artista, vai-se ganhando percepção do fluxo de processo de pensamento e argumentação com a máquina.

Podemos observar a programação ao vivo como um desdobramento das experiências musicais do século XX, tal como o desenvolvimento das criações do compositor alemão Karlheinz Stockhausen (1928-2007) em direção à composição por processos e fórmulas. Entre as décadas de 1950 e 1960, as obras de Stockhausen passaram por música pontilhista, composição por grupos, forma-momento, composição por processo, e finalmente, composição por fórmula (ASSIS, p.23). Acredito que esse seja um ponto de interseção entre a arte sonora e as artes visuais.

Em seu texto *Parágrafos sobre a Arte Conceitual*, escrito em 1967, o artista americano Sol LeWitt (1928-2007) postula que, na arte conceitual, a idéia, após sua concepção, transforma-se na máquina criadora da obra; também afirma que o trabalho, uma vez fora das mãos do artista, não será controlado a respeito de como será percebido e compreendido por pessoas diferentes. Essa consciência é muito presente em mim quando crio ou apresento minhas obras, inclusive abrindo espaço para essas múltiplas – e ricas – experiências e pontos de vista sobre as obras.

A não previsibilidade é um ponto essencial no tipo de interação que acontece entre seres humanos e máquinas durante o processo criativo de programação ao vivo. É uma mescla entre trabalho intelectual e emocional, uma mescla entre aquisição de controle e deixar-se levar pelo fluxo; oposição entre uma visão instrumentalista, em que o *performer* deve ser virtuoso com um instrumento, e uma visão mentalista, em que o performer tenta fazer a máquina funcionar criativamente, domando as dificuldades técnicas das mais abrangentes, como a escolha dos *softwares* e equipamentos para o seu ecossistema pessoal, dos mais minuciosos detalhes técnicos e de comandos e variáveis que trabalharão dentro de um algoritmo específico. Observa-se aqui um diálogo com a música

concreta, que trouxe de liberdade ao artista criador sonoro. Nele, o compositor usa mais de atributos de inventividade do que da teoria musical.

Voltemos a atenção para composições generativas anteriores ao computador, baseadas em regras ou *scripts* de processos.

John Cage (1912-1992) foi um dos compositores mais influentes do século XX, tanto pela sua produção musical quanto pela conceitual, que aproximou o fazer musical ocidental de práticas meditativas da cultura oriental. Cage dizia não fazer objetos (artísticos) e sim demonstrar processos de criação, sem objetivar a qualidade da peça em desenvolvimento. "Eu não estou interessado em mim mesmo, em meu trabalho ser uma comunicação de mim para o ouvinte, mas dos sons propriamente para os ouvintes. Então eu faço música da qual eu não sou somente o compositor, mas o ouvinte também." (SCHEFFER, 2012) Em 1939, Cage criou a composição *Imaginary Landscape Number 1*. Entre os instrumentos utilizados, havia dois toca-discos com diferentes velocidades de rotação. Essa técnica foi explorada primeiramente por Paul Hindemith e Ernst Toch em 1930.

Desde os anos 1950, Cage interessou-se por conceitos *zen* budistas sobre deixar-se levar pelo fluxo da vida e dos acontecimentos cotidianos. Esse conceito é presente em suas obras por meio de operações de chance e acaso. A composição *Imaginary Landscape Number 4* foi criada para doze aparelhos de rádio, e realizada pela primeira vez em 1951, na Universidade de Columbia, no Estados Unidos. A premonição sobre o uso e a invenção de novos timbres sonoros levou Cage a construir várias versões de seu "piano preparado": um piano com materiais colocados no seu interior para modificar os sons naturais do instrumento, tornando-o uma espécie de orquestra percussiva sob o comando de um só músico. Para Cage, muitos sons interessantes para o processo criativo estariam fora do alcance da harmonia. A obra de Cage *Music of Chances (Música de Chances)* foi inspirada e composta usando o livro chinês do *I Ching*. Uma das obras seminais na carreira de Cage é *4:33* ; composta – aparentemente – só de silêncio. Quase silêncio, uma vez que a experiência sonora vivida pelo público é repleta de sons do ambiente. O trabalho não visava à estética de uma peça pronta, e sim à proposta de tornar os sons do dia a dia perceptíveis como arte. Nesse sentido, a obra dialoga diretamente com os objetos *ready made* de Marcel Duchamp.

Eno editou, em 1975, um conjunto de cartões denominado *Estratégias Oblíquas* [9]. Cada carta contém uma instrução criativa, com o objetivo de trazer direções

---

[9] Versão *online* com possibilidade de escolha randômica das cartas de "Estratégias Oblíquas" disponível em: < http://stoney.sb.org/eno/oblique.html >

inesperadas ao processo criativo (musical). As frases passam tanto pelo terreno dos termos musicais quanto pelo terreno da autoajuda: “Abandone instrumentos normais”, “Use menos notas”, “Confie no você de agora”, ou “O que o seu amigo mais próximo faria?”. Aqui, mais do que nunca, o fazer musical aproxima-se de uma brincadeira. Até mesmo o conceito de música é implicitamente questionado, em oposição à exploração de universos sonoros.

Outro artista importante de menção no contexto da inovação e inventividade musical é o grego Iánnis Xenákis (1922-2001), conhecido como um dos compositores de *avant-garde* mais importantes. Xenákis foi um dos primeiros artistas a usar modelos matemáticos na criação musical, constituindo uma grande influência na história da música eletrônica. Pioneiro na criação de partituras geométricas, seu trabalho sonoro realizou um intenso diálogo com as artes visuais e, principalmente, com a arquitetura. Xenákis estudou e elaborou trabalhos de sonificação de dados e notações musicais não tradicionais. Fazia suas partituras inicialmente sem o auxílio de computadores. Nas suas composições que utilizam os meios digitais, fica evidente que computadores e programas não são somente mecanismos, mas sim mídias extremamente ricas e que possuem agenciamentos próprios, delegados a eles e distribuídos então para outros atores do processo de criação (BERRY, 2012). Aqui podemos ver duas raízes das quais, por meio da prática de música aumentada pelo computador, desenvolveram-se posteriormente: 1) a intercalação metafórica do sonoro com elementos visuais e 2) a colaboração do inesperado no momento da execução e apreciação da obra.

Xenákis desenvolveu e utilizou programas de computador para cálculos de equações de distribuição de dados – no caso, eventos sonoros. Além de processos aleatórios, utilizava máscaras de tendência, controlando os limites de escolhas do sistema, desde a estrutura da peça até os parâmetros para a criação de ondas sonoras.

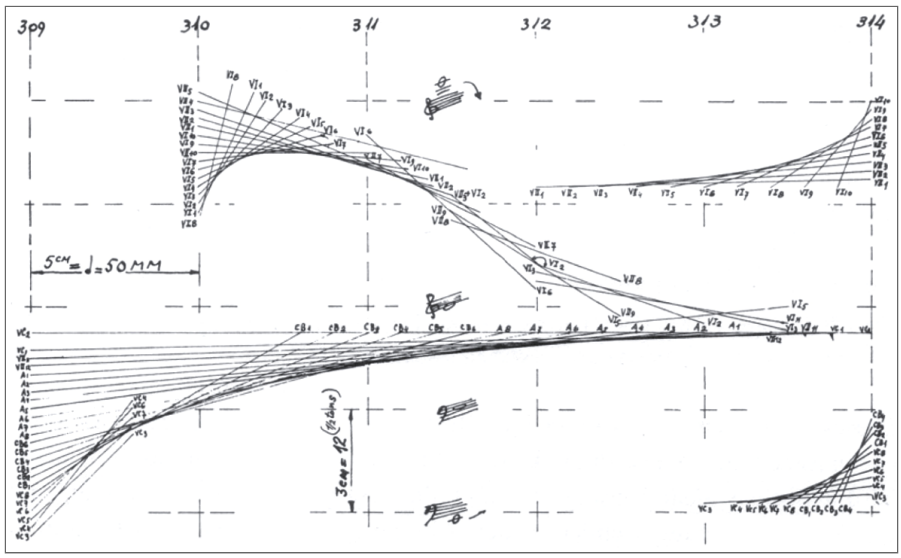


Imagem 6. Trecho da partitura visual de Metastasis (Iánnis Xenákis, 1954).

Nos anos 1970, Xenakis desenvolveu um novo instrumento: um computador e *software* para criação musical chamado *UPIC (Unité Polyagogique Informatique du CEMAMu - Centre d'Etudes de Mathématique et Automatique Musicales de Paris*), um precursor dos *softwares* de produção musical em plataformas *desktop* e equipamentos sensíveis ao toque (*tablets*). Um dos trabalhos mais importantes de Xenákis foi a composição *GENDY3*[10], cujos sons e organização estrutural foram ambos criados com seu método algorítmico.

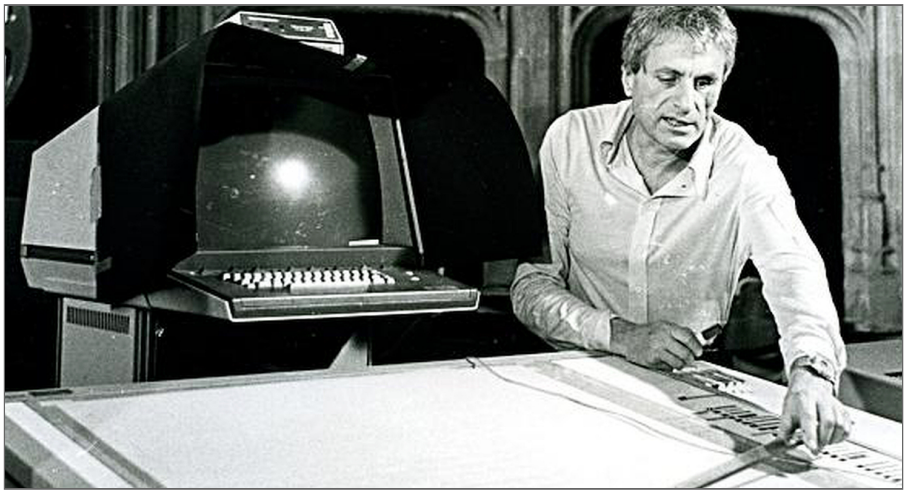


Imagem 7. Xenákis com a mesa de desenho do sistema UPIC. (foto: Joel Chadabe)

O pesquisador Peter Hoffman, em seu artigo "*Music Out of Nothing? A Rigorous Approach to Algorithmic Composition by Iánnis Xenákis*", relata que o artista "não tinha nenhuma ambição em emular instrumentos tradicionais [...] Sua busca era por fenômenos

[10] *GENDY3* (abreviação de "geração dinâmica", *GENeration DInamic*), além do nome da composição, é o nome do *software*, criado na linguagem *BASIC* por Xenakis.

emergentes, tratando a música eletroacústica algorítmica e a possibilidade de computação dos sons como a matéria prima da sua criação artística". (HOFFMAN, p. 7) Entendo que encontramos aqui um ponto crucial: quando um artista decide tratar o som como uma matéria plástica ou até escultórica, prevendo e moldando a pressão do ar do ambiente de reprodução.

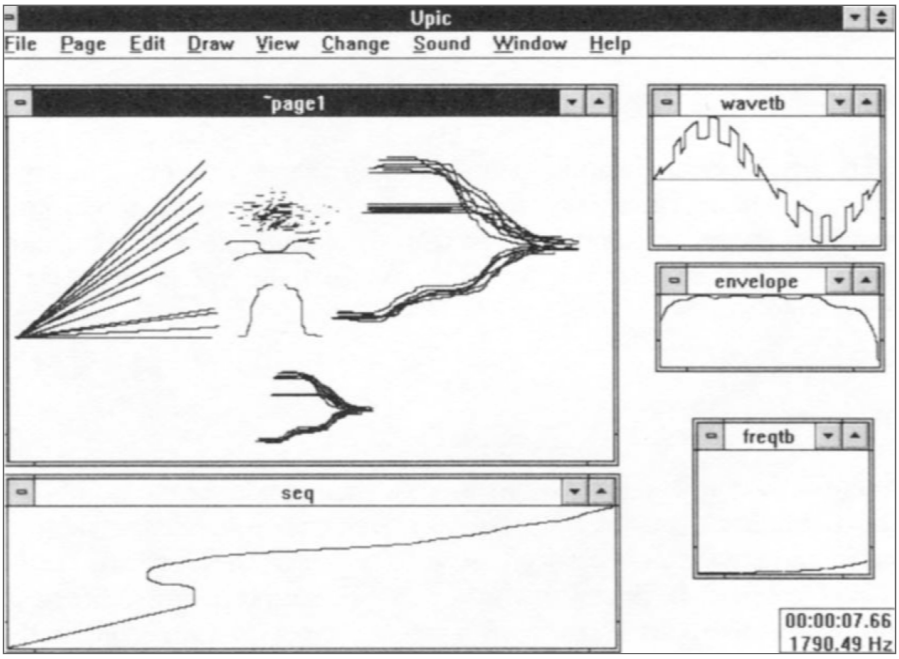


Imagem 8. Interface do software UPIC, em sua versão dos anos 1990.

Em trabalhos de Xenakis como o *software GENDY*, o resultado sonoro é o resultado de processos programados previamente. Já nos seus últimos trabalhos instrumentais, o compositor não usava mais *softwares*, pois já havia absorvido os algoritmos em sua intuição. (HARLEY, 2018) No sistema *UPIC*, o compositor podia desenhar com uma caneta óptica ondas sonoras e parâmetros musicais para execução musical instantânea, como frequência e amplitude.

O *software* livre de composição com partituras visuais *Iannix*[11], desenvolvido desde 2002, é uma adaptação dos procedimentos criativos de Xenákis. O programa roda atualmente em sistemas *Linux*, *macOS* e *Windows*. Indo além do conceito de representação de partituras bidimensionais do *UPIC* (no qual foi inspirado), o sistema *Iannix* permite a criação e a execução de partituras representadas por gráficos em três dimensões. Por meio dos sistema de conexão OSC, pode comunicar-se com vários outros *softwares*, permitindo uma ampla gama de possibilidades de síntese sonora. Para tornar as possibilidades criativas

[11] O *software Iannix* encontra-se disponível para *download* gratuito em: < https://www.iannix.org >

ainda mais poderosas do que com o uso de partituras gráficas, o *Iannix* permite o uso da linguagem de programação *JavaScript* para que se definam parâmetros e estruturas das composições musicais.

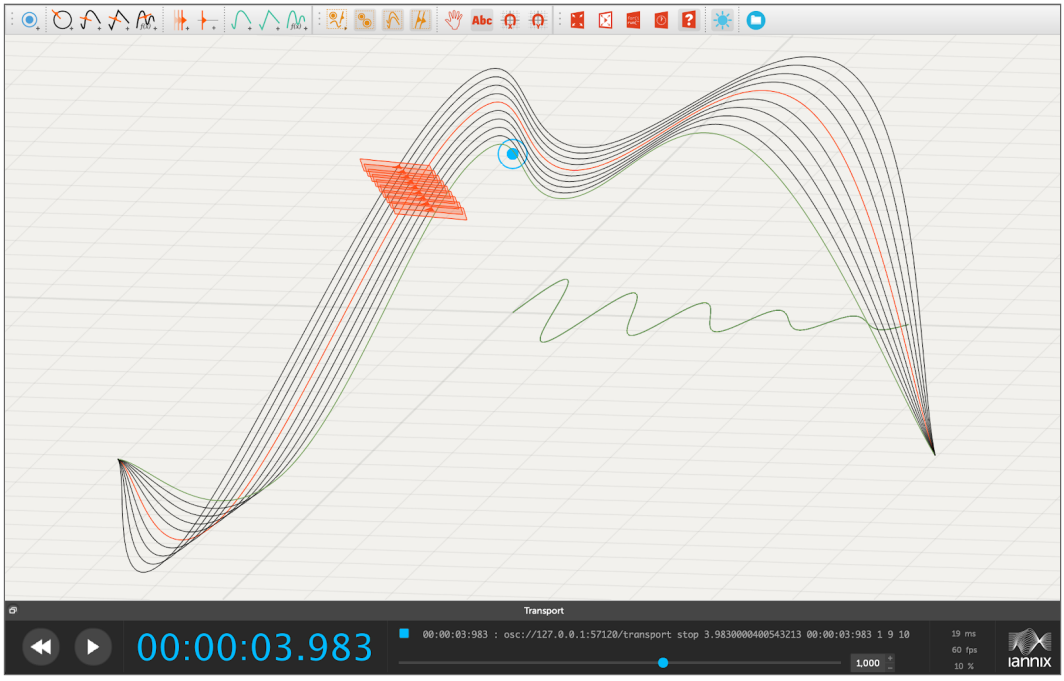


Imagem 9. Módulo de composição com esquema tridimensional do programa Iannix.

Xenákis cruzava então várias fronteiras entre música, desenho e arquitetura, provando que o artista não precisa se ater a uma só técnica, a um só suporte, ou a uma só área de atuação: ele pode misturar esses elementos numa obra.

Outro compositor que se vale é mencionar Steve Reich[12] (1936-), pioneiro da música minimalista. Sua obra usava regras de restrição de uso de elementos musicais, *loops* de fita de rolo, gravações de voz, foco nos sons e não no sentido das palavras com expectativa de criação de novidades sonoras. Suas principais obras foram compostas com técnicas de manipulação de velocidade. Em *Clapping Music* (Música de Palmas), são ouvidos dois padrões desencontrados de palmas com as mãos, que se encontram cerca de 5 minutos após o início da peça. A obra foi livremente inspirada no ritmo e sonoridade da música flamenca. A obra *It's gonna rain (Começará a chover)* explora o conceito de fases, executando duas gravações simultâneas em velocidades diferentes.

[12] A obra de Reich pode ser explorada no site do artista, disponível em: < www.stevereich.com >

Brian Eno[13] (1948-) é o artista multimídia criador do que foi chamado desde então de "música ambiente": paisagens sonoras com modificações suaves e constantes durante o tempo. Lançou em 1978 o álbum primordial para os campos da música eletrônica e da música ambiente, *Ambient 1: Music for Airports*. Um dos objetivos aparentes desse tipo de composição é permitir diferentes níveis de atenção do público. Tais peças podem ser ouvidas com atenção ou ignoradas, assim como o artista de videoarte Nam June Paik entendia a exposição de obras de videoarte – que têm outro "tempo" e intensidade de atenção de observação em relação a uma obra cinematográfica.

A fim de possibilitar a entrada de elementos aleatórios e inesperados dentro de composições musicais, Eno criou um sistema em forma de baralho de cartas, chamado de *Estratégias Oblíquas*: dezenas de cartões de regras, tais como: "Vire a composição de cabeça para baixo". Explora, desde 2009, a programação e a distribuição de obras sonoras com algoritmos generativos por meio de aplicativos *(apps)* para dispositivos móveis: *Bloom* (2009), *Trope* (2010), *Scape* (2013) e *Brian Eno: Reflection* (2017). Os aplicativos *Bloom*, *Trope* e *Scape* foram criados em conjunto com o desenvolvedor de *software* Peter Chilvers. Os programas operam no campo misto entre arte sonora e visual, assim como tem sido a carreira artística de Eno, que vem oscilando, durante as últimas quatro décadas, entre música e móbiles de luz. Suas esculturas cinéticas movimentam blocos de cor com lentos movimentos ritmados. "Eu gosto de ideia de uma música que continua a ser criada, comigo presente ou não." (ENO, 2017)

Hanna Davis[14] é uma compositora que vem trabalhando com sonificação de dados como forma de arte. A artista emprega um tratamento interpretativo do texto, uso de dados subjetivos e coleta artesanal de dados. Hanna distingue teoricamente as áreas correlatas de:

- Audificação: conversão direta de dados em sons (ondas sonoras), interpretando-se os dados como amplitude distribuída no tempo;

- Mapeamento de parâmetros: relaciona certos valores dos dados com componentes sonoros, focando em certos aspectos dos dados;

- Geração de música: mapeamento de trechos dos dados em componentes sonoros focado em padrões musicais interessantes para o artista, podendo conter componentes que não são derivados dos dados.

---

[13] *Site* do artista multimídia Brian Eno: < https://brian-eno.net >

[14] *Site* da artista Hanna Davis: < www.hannahishere.com >

Desenvolveu o projeto *TransProse*[15] (2013), criando composições sonoras que relacionam trechos de obras literárias a sentimentos e, posteriormente, a elementos musicais. O sistema funciona conectado a um banco de dados que relaciona mais de 14 mil palavras a sentimentos, o *NRC Lexicon*[16], desenvolvido pelo cientista de computação Saif Mohammad. Esse banco de dados existe em versões traduzidas para mais de 100 línguas. A artista utiliza, por exemplo, os seguintes grupos de regras para suas composições: se o sentimento é negativo, coloca o trecho em acordes menores; se é positivo, são utilizados acordes maiores. (DAVIS, 2018)

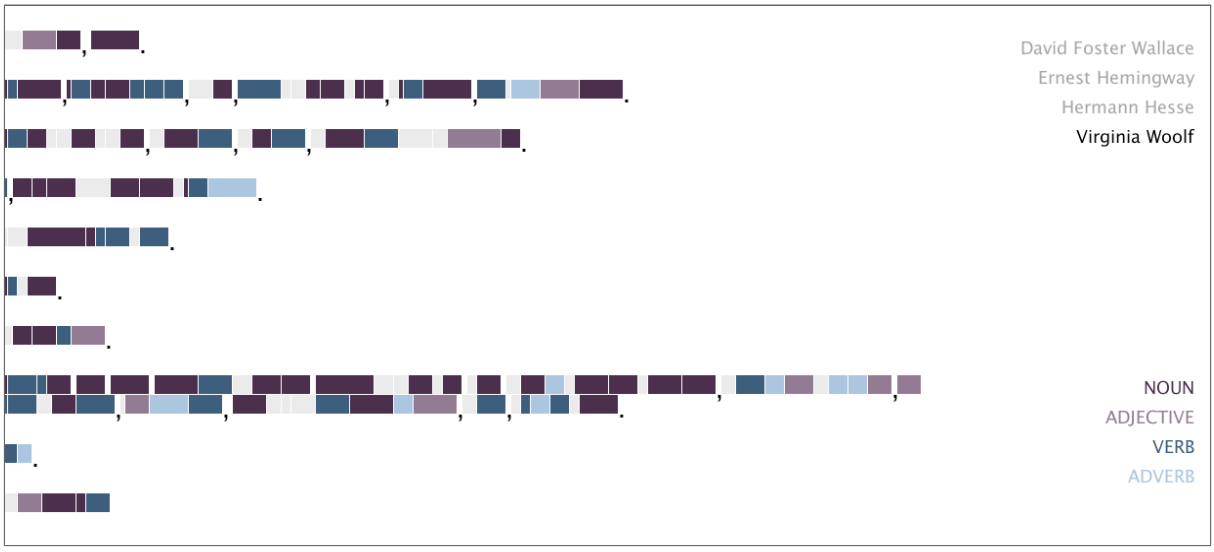


Imagem 10. Mapeamento gramatical de trecho do livro As ondas, de Virginia Woolf.

Davis apresentou, no Museu do Louvre de Paris, em 2018, a obra comissionada *Symphonologie, the Music of Business* [17], em parceria com o compositor francês Mathieu Lamboley. A sinfonia de 8 minutos foi executada por uma orquestra de cinquenta músicos e composta a partir da interpretação de 220 mil palavras de textos sobre como a tecnologia tem modificado os ambientes de trabalho. Neste caso, o *software TransProse* encontrou principalmente conexões de palavras com os sentimentos de confiança, ansiedade e medo.

Outro realizador que trabalhou de forma interessante com a composição algorítmica é James Murphy, um músico *pop* e líder da banda *LCD Sound System*. Em 2014 recebeu um convite da empresa de computadores e *software* IBM, por meio da agência de publicidade Ogilvy & Mather, para criar composições sonoras a partir de dados obtidos de

---

[15] *Site* do projeto *TransProse:* < www.musicfromtext.com >

[16] Página do banco de dados de palavras e significados emocionais subjetivos *NCR Emotion Lexicon*: < http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm >

[17] Matéria *This symphony had both human and computer composers*, no jornal Washington Post: < https://wapo.st/2dpcSPm >

partidas de tênis da competição *U.S. Open*. Com a assistência de engenheiros de *software*, Murphy criou vários temas musicais baseados em vários dados das partidas, tais como tempo entre raquetadas e evolução do placar de pontos (MURPHY). Foi feito então um trabalho de *remix* em estúdio de gravação para misturas e equalização das partes geradas, assim como criação arranjos musicais, entrelaçando as partes sonificadas pelos processos algorítmicos. A obra *Making Music with Tennis Data*[18] foi distribuída em forma de website de ação publicitária da empresa IBM.

1.3 *Live coding*

A programação ao vivo (*live coding*) traz consigo o erro, inerente do improviso. Como o pensamento orgânico lida com o erro, diferentemente da máquina? E com as interferências? A relação entre algoritmo e organismo é a base da hibridização. Não só aglomera duas coisas estranhas, mas pretende juntar dois sistemas que conversam entre si, que se transformam. O cérebro e máquina juntos trazem uma complexidade: a da relação entre a consciência humana e a do sistema, que é inorgânica e – agregando uma memória, uma dimensão de camadas e uma complexidade emergente. A máquina aproxima-se de um estado de consciência, aproxima-se dos processos mentais humanos.

Um dos aspectos mais importantes da criação em *live coding* é a improvisação. Daí o aspecto emergente de obra aberta – processo lúdico de aprender brincando. Como essa prática depende de computadores e sistemas de síntese de som, as primeiras experiências datam de não mais de uma década. Tudo é novo e tem um ritmo de reciclagem também muito rápido. As linguagens que foram inventadas e funcionavam em sistemas de 5 anos atrás podem já não funcionar tão facilmente.

Isso levanta uma discussão sobre a questão de da preservação da arte digital. Uma das possibilidades que vislumbro serem viáveis é o investimento em ferramentas de emulação de sistemas antigos em sistemas novos. Pude recentemente, por exemplo, rodar dentro de um navegador da internet o *Apple BASIC*, linguagem por meio da qual tive o primeiro contato com a informática, no início da década de 1980. Por que faço utilização – e a entusiasmo – do *software* livre? O pesquisador do *Massachusetts Institute of Technology* Michel Fischer, especializado em de circuitos de mídia, argumenta que "o código-fonte portátil tornou-se o esperanto dos humanos que criaram sua própria Babel de máquinas tribais." (FISCHER, p.125)

---

[18] Vídeo de registro de criação da obra *Making Music with Tennis Data*:
< https://vimeo.com/136780902 >

No que difere o *live coding*, como ferramenta de composição e performance, dos sistemas de programação de *software* convencionais? Principalmente, na imediaticidade de execução das ideias.

Sobre a mesclagem dos processos de composição e performance, podemos observar que uma criação em *live coding* pode englobar, simultaneamente, os processos de composição e de performance de uma obra sonora, visual ou audiovisual. Durante o processo de criação, o público pode perceber algumas palavras-chave da língua inglesa em destaque, o que permite uma compreensão geral de trechos da programação. Podem-se perceber estruturas de lógica e bifurcação com estruturas do tipo *IF* (indicam condição, hipótese), *THEN* (indicam consequência, reação).

As fronteiras da ciência da computação, assim como as fronteiras entre as práticas artísticas, são fluidas e subjetivas. Até mesmo a definição do que é arte e do que é design é uma discussão longa – que não é objeto desta pesquisa. Mas uma definição da qual sou entusiasta é a de que o design e a ciência da computação resolvem problemas, já a arte seleciona e propõe problemas e temáticas de foco voltado para o pensamento e para a reflexão. Nesse sentido a arte aproxima-se mais do domínio do conhecimento filosófico e espiritual – no sentido da postura de indagação. Dentro dessa perspectiva, considero a programação ao vivo, a programação aberta e o compartilhamento de trechos de código convites à colaboração e ao desenvolvimento. O código funciona como uma partitura de representações de processos computacionais, que não só conduz ao resultado sonoro, mas também fica disponível para reflexão, reuso e modificação (BROWN, 2006, p.1). Entendo aqui os sistemas de criação artística por meio de código assim como Pierre Lévy, que diz ser o virtual um real fantasmático, latente. Esse mundo é constituído de espaços e tempos específicos na sua realidade simulada; o computador é cocriador que ajuda a calcular um número imenso de possibilidades visuais e sonoras e desdobramentos narrativos de cada sistema-obra.

> "A prática da programação ao vivo consegue confundir os conceitos estabelecidos do discurso musical, tais como compositor, executor e público; instrumento, partitura e obra; composição, performance e improvisação; palco e auditório; instrumento e ferramenta. Ela apresenta uma perspectiva interessante quanto à questão modernista relativa à forma e ao conteúdo." (MAGNUSSON, 2014, p.7)

Uma das principais ofertas da técnica de programação ao vivo, para mim, foi a possibilidade de continuidade na busca de um hibridismo transmidiático, o que seria a fusão de arte visual, música, poesia e código-fonte; o código de computador funciona como suporte e mensagem ao mesmo tempo.

Sobre regras e limites: artistas como o compositor John Cage já apregoavam os benefícios da estimulação criativa por meio da imposição (ou sugestão) de regras ao processo de criação artística. Comparo essas práticas com a receita dadaísta para criação poética:

> "Pegue um jornal. Pegue uma tesoura. Escolha no jornal um artigo com o comprimento que pensa dar ao seu poema. Recorte o artigo. Depois, recorte cuidadosamente todas as palavras que formam o artigo e meta-as num saco. Agite suavemente. Seguidamente, tire os recortes um por um. Copie conscienciosamente pela ordem em que saem do saco. O poema será parecido consigo. E pronto: será um escritor infinitamente original e duma adorável sensibilidade, embora incompreendido pelo vulgo." (TZARA, 1920)

Como exemplo referenciado à minha própria obra, detalho no Apêndice I o projeto *Weekly Beats 2016*, quando pude fazer uso de arquivos de áudio escolhidos aleatoriamente com a função de escolha de página randômica do *site Freesound*, um vastíssimo banco de dados sonoro *online*.

1.4 Criatividade no *design* de linguagens de programação

As etapas anteriores ao uso de uma linguagem de programação são os desenhos conceituais e técnicos da linguagem propriamente dita. É um campo no qual a engenhosidade e estilo dos inventores definem quais os padrões de pensamento e de expressão os programadores poderão utilizar. Algumas implementações de sistemas são puramente brincadeiras, não tendo objetivos práticos de solução de problemas; mas chamam atenção para as decisões de desenho de um ambiente de programação.

A linguagem *Piet*[19] funciona não com comandos de texto, mas recebendo do programador uma imagem semelhante aos quadros do pintor Piet Mondrian, que nesse

---

[19] Página do sistema de programação Piet:
< http://www.dangermouse.net/esoteric/piet.html >

sistema não é o resultado do programa, e sim o formato de seus comandos. A imagem abaixo é um programa criado por Thomas Schoch[20], que escreve na tela a palavra "Piet". É realmente espantoso olhar para uma obra "de arte visual" e saber que ela é (também) um programa de computador. Essa fascinação acompanha-me desde os tempos da programação *BASIC*, quando pontos coloridos lentamente e imprevisivelmente misturavam-se com as linhas de código, em um espaço geométrico colaborado e culminou recentemente na criação da obra *Tocando Brasília*, criada com a linguagem de programação ORCA.

> "A cultura contemporânea está impregnada de práticas em que a produção de sentido resulta da combinação de fragmentos [...] O *remix* é a forma mais contemporânea de polifonia e, por se tratar de processo apenas possível em mídias eletrônicas e digitais, é mais fluido." (BASTOS, p.27-28)

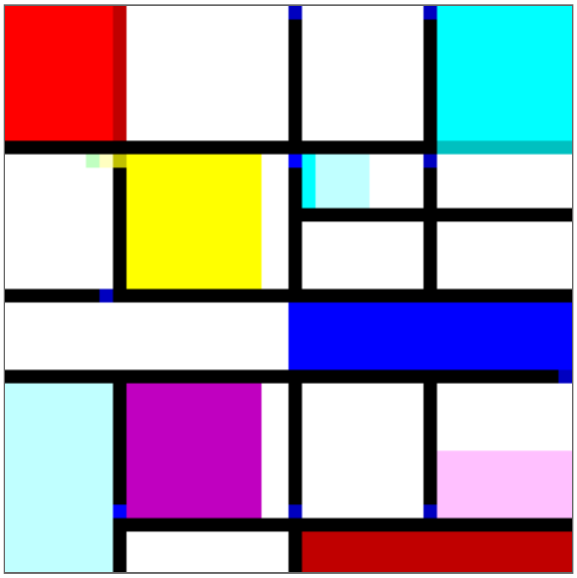


Imagem 11. Software no formato Piet: cada cor, linha e ponto é uma função de programação.

[20] Página de exemplos de programas na linguagem Piet:
< http://www.retas.de/thomas/computer/programs/useless/piet/Piet/index.html >

A linguagem de programação *Gertrud* [21] funciona da seguinte forma: o programa pode ser escrito com frases em qualquer língua (português, inglês, francês) e para cada frase é calculada uma média do tamanho das palavras. As palavras que não se encaixam na média de tamanho de cada frase são usadas como índices relacionados às funções de processamento do programa (cálculos matemáticos, operações de fluxo do programa, criação de variáveis, etc). O resultado estético do programa é um texto que o autor da linguagem considera ter estilo similar ao da escritora americana Gertrud Stein. Aos olhos de quem não sabe o que está examinando, o programa é um texto em prosa, sem nenhuma semelhança com o que se associa normalmente a um programa de computador, como a presença constante de números, sinais matemáticos e pedaços isolados da língua inglesa (*IF*, *THEN*, *ELSE*...).

Na linguagem *Cow*[22], cujo nome significa vaca, todos os comandos são variações de onomatopeias de sons produzidos pelo animal (Moo, MOo, moO, mOo, etc). O formato de programação *Arnoldc* utiliza frases de filmes com o ator Arnold Schwarzenegger ("*Hasta la vista, baby!*") e o sistema *Shakespeare*[23] exige que a programação seja feita utilizando vocábulos das peças teatrais de William Shakespeare. Já a linguagem *Pikachu*[24] tem comandos baseados nos sons emitidos pelo personagem do desenho animado Pokémon (pi, ka, pika, chu, pikapi, etc).

O visual do código-fonte desses sistemas de desenvolvimento, classificados pela comunidade de programação como "linguagens esotéricas", acaba-se assemelhando a uma poesia concreta ou a uma partitura musical. Um dos exemplos mais significativos, no tocante à inovação e uso "subversivo" de ferramentas de programação, é a prática da Poesia Perl, que engloba a criação de programas na linguagem *Perl*. Os programas, quando lidos, soam como poesia, como no programa de Angie Winterbottom, traduzido a seguir. (COX, 2001, p.4).

---

[21] Página da linguagem de programação *Gertrude*:
< http://www.p-nand-q.com/programming/languages/gplz/gertrude.html >

[22] Página da linguagem de programação *COW*: < https://bigzaphod.github.io/COW >

[23] Página da linguagem de programação *Shakespeare*:
< http://shakespearelang.sourceforge.net >

[24] Página da linguagem de programação *Pikachu*:
< https://github.com/groteworld/pikalang >

Perl	if ((light eq dark) && (dark eq light)
	&& ($blaze_of_night{moon} == black_hole)
	&& ($ravens_wing{bright} == $tin{bright})){
	my $love = $you = $sin{darkness} + 1; };

Inglês	*If light were dark and dark were light* *The moon a black hole in the blaze of night* *A raven's wing as bright as tin* *Then you, my love, would be darker than sin.*
Português	Se a luz fosse escura e escura fosse a luz, A lua, um buraco negro no brilho da noite, A asa de um corvo brilhante como estanho, Então você, meu amor, seria mais negra que o pecado.

Tabela 2. Poesia na linguagem Perl escrita por Angie Winterbottom.

O ponto mais importante desse desenvolvimento de linguagens exóticas, que primeiramente não tinham um objetivo além da sua própria existência, foi a invenção da linguagem *ORCA*, que utiliza uma abordagem completamente fora do tradicional, porém deixando uma porta aberta à coexistência com outros programas: a possibilidade de comunicação inter-*software* e inter-*hardware* utilizando-se os protocolos MIDI, OSC e/ou UDP[25]. O sistema *ORCA* é descrito detalhadamente no Capítulo 2, Seção 2.4.

Como pensa o artista ao vislumbrar a obra completa, sem se ater aos caminhos a percorrer? De acordo com o teórico sobre design e inovação John Maeda, os artistas conseguem inovar usando recursos restritos, lidar com a ambiguidade, com múltiplas perspectivas e também com a adversidade e a volatilidade (MAEDA); daí o meu interesse pelo *software* livre e pelo aprofundamento nas possibilidades não imaginadas previamente para os programas e equipamentos eletrônicos. Uma das técnicas que julgo mais frutíferas para a ideação e criação de trabalhos autênticos e diferenciados é a busca de novas formas de expressão por meio da junção de programas diferentes e uma certa subversão dos propósitos originais de cada *software* / *hardware*. Ao utilizar na minha produção o *Sonic Pi* para o controle do *software* de criação visual *Blender 3D,* estou subvertendo de maneira

[25] O protocolo *UDP - User Datagram Protocol* (Protocolo de Dados entre Usuários) permite o envio de longas mensagens de texto entre programas ou computadores diferentes. Os formatos *MIDI* e *OSC* serão detalhados no Capítulo 4.

radical o uso inicialmente pensado pelos seus desenvolvedores. Isso traz desafios novos e resultados inesperados muito interessantes à exploração artística, com atenção aos processos de emergência de resultados.

Como exemplo de artista empenhado em modificações e usos criativos de programas de computador, cito Robin Newman, um músico adepto da programação ao vivo. É criador de inúmeros projetos interessantes com o sistema *Sonic Pi*, dentre eles:

1) *Sonic Pi Sequencer*[26], um sequenciador para *Sonic Pi* rodando em dispositivos *iOS* ou *Android*, enviando comandos do *tablet* para o *Sonic Pi*;

2) *Sonic Pi Spirograph*[27], um sistema que cria ilustrações geométricas por meio de código musical do *Sonic Pi*.

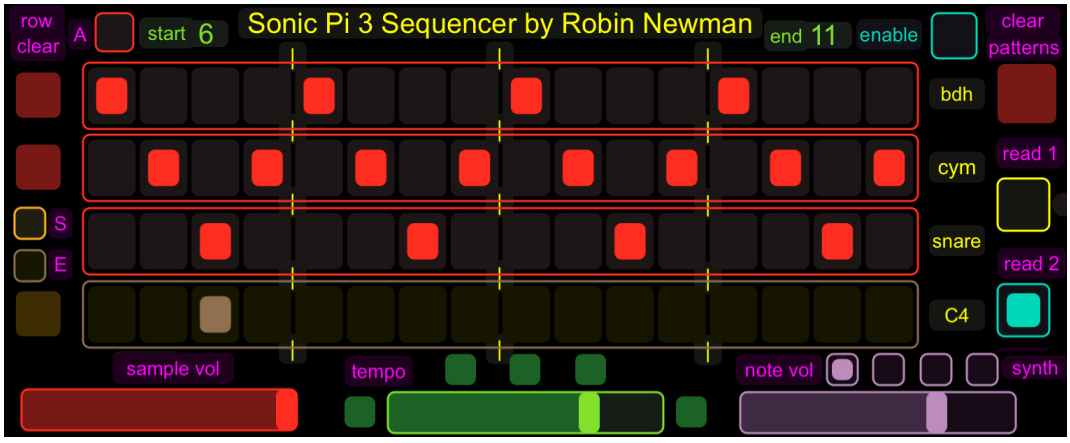


Imagem 12. Detalhe da interface do aplicativo Sonic Pi 3 Sequencer.

Uma interessante síntese do estado da arte da prática de *live coding* é o movimento descentralizado *Algorave*, que produz performances e transmissões pela internet de sessões de programação ao vivo. Uma vez por ano é realizada uma transmissão coletiva – a de 2019 com mais de 12 horas de duração – em que cada participante inscrito pode transmitir a sua sessão de *live coding* por cerca de meia hora. Nestas longas transmissões ao vivo, pode-se ver a vibrância dos estilos de criação com código ao vivo, tanto na variedade de ferramentas sendo exploradas pelos artistas quanto nas possibilidades de estilos musicais que são criados – apesar da clara tendência em direção a vertentes da música eletrônica, música serial e música eletroacústica.

---

[26] Página do sistema Sonic Pi Sequencer:
< https://rbnrpi.wordpress.com/sonic-pi-and-touchosc-sequencer >

[27] Página do sistema Sonic Pi Spirograph:
< https://rbnrpi.wordpress.com/2018/12/17/sonic-pi-controls-spirograph-version-2 >

A organização *TOPLAP*[28] (*Temporal Organisation for the Purity of Live Algorithm Programming* / *Organização Temporal para a Pureza da Programação de Algoritmos ao Vivo*) foi criada em 2004, com o objetivo de promover a prática da programação ao vivo. A *TOPLAP* organiza anualmente uma performance telemática coletiva de artistas de *live coding* via internet. A edição de 2019, que comemorou os 15 anos da organização, contou com a participação de 168 artistas, totalizando mais de 84 horas consecutivas de performances transmitidas ao vivo; a minha está descrita no Capítulo 2, na Seção 2.3.8.

O grupo *TOPLAP* foi formado no evento *Changing Grammars*, realizado em 2004, em Hamburgo, na Alemanha. Segue transcrição de trecho do seu manifesto constituinte:

> **Manifesto TOPLAP**
>
> **Nós demandamos:**
> • Dê-nos acesso à mente do artista, ao instrumento humano completo.
> • O obscurantismo é perigoso. Mostrem-nos as suas telas.
> • Programas são instrumentos que se podem modificar.
> • O programa deve ser transcendido – a inteligência artificial é o caminho.
> • O código deve ser visto, assim como ouvido – o fundo de algoritmos deve ser visto, assim como o seu resultado visual.
> • Live Coding não é sobre ferramentas. Algoritmos são pensamentos. Serras elétricas são ferramentas. É por isso que algoritmos são, algumas vezes, mais difíceis de se perceber do que serras elétricas.
>
> **Nós reconhecemos momentos contínuos de interação e profundidade, mas preferimos:**
> • Observação aprofundada nos algoritmos.
> • A hábil expressão de algoritmos como forma de sagacidade mental expressiva / impressionante.
>
> **Nós reconhecemos que:**
> • Não é necessário que o público entenda o código para poder apreciá-lo, assim como não é necessário saber tocar guitarra para se apreciar uma performance de um guitarrista.

[28] Site da organização TOPLAP: < https://toplap.org >

• Programação ao vivo pode ser acompanhada de demonstrações impressionantes de destreza manual e de glorificação da interface de digitação.

Entre os aspectos que definem a programação ao vivo, um dos mais importantes é a visualização do processo de programação. “O princípio explícito – Mostre sua tela – marca a diferença fundamental entre o *live coding* e a música eletrônica em geral.” (PRÓSPERO, p.46) Acredito, porém, que a classificação pode ser mais abrangente que o manifesto e englobar mais sistemas (no tocante às técnicas e procedimentos) de código adaptável ou modificável pelo *performer*. De acordo com o pesquisador Nick Collins (2011), sobre a programação ao vivo, parece não haver fim para as permutações potenciais entre os humanos e os algoritmos.

1.5 Sobre a escolha dos ambientes de criação

Escolhas técnicas, estéticas e políticas: a fim de responder a pergunta de quais escolhas delimitam o potencial criativo, dentre as opções de suporte e de técnicas, procuro demonstrar, pelos exemplos de processos criativos, que um dos pontos mais importantes de posicionamento poético é a subversão dos usos típicos de cada *software*, principalmente quando são feitas conexões inusitadas entre os sistemas. A conexão técnica entre os *softwares* de som e vídeo acontece por meio do protocolo de comunicação MIDI, que permite ligar programas dentro do mesmo computador ou instrumentos musicais eletrônicos conectados via cabo MIDI e/ou USB.

Podemos ver na tabela a seguir que a minha tendência tem sido usar um número maior de *softwares* e especialmente *softwares* livres, principalmente pela sua tendência de maior longevidade técnica.

	1995	1997	1999	2001	2003	2005	2007	2009	2011	2013	2015	2017	2019
Softwares com ênfase em criação visual													
Director													
Blender													
openFrameworks													
Processing													
VDMX													
Unity													
Three.js													

Softwares com ênfase em criação sonora													
ixi-lang													
reNoise													
Ableton Live													
Sonic Pi													
ORCA													

Tabela 3. Histórico de estudo e produção com ferramentas audiovisuais.

Uma das desvantagens intrínsecas aos ambientes de desenvolvimento baseados em modelos pagos é a vulnerabilidade da empresa às circunstâncias de mercado. Como exemplo disso, cito o software *Quase-Cinema 1*[29], que desenvolvi em 2006, no ambiente de criação *Macromedia Director*[30]. Comprado pela empresa Adobe, hoje o ambiente de desenvolvimento Director não existe mais no mercado e nem funciona em sistemas operacionais atuais.

Pude testemunhar outro exemplo com o *software* de criação tridimensional *Maya*, criado com a fusão das empresas Alias e Wavefront. Dediquei grande quantidade de tempo – e recursos financeiros – fazendo cursos e leituras sobre a operação do *software* entre os anos de 2002 e 2004. Por razões mercadológicas, o *Maya* foi adquirido pela empresa Autodesk, criadora do programa concorrente *3D Max*, um dos líderes de mercado entre os *softwares* proprietários. Uma filosofia de criação diferente da empresa antiga ocasionou uma grande guinada em direção à dissolução dos conceitos originais do programa, o que diluiu sua essência.

Mais um caso de ambiente profissional de criação tridimensional foi a história do programa canadense *Softimage 3D*, um dos mais venerados sistemas 3D; criado no ano de 1988, deixou completamente de existir no ano de 2014, após a sua compra também pela gigante do mercado Autodesk. Os *softwares* livres, embora precisem também buscar recursos financeiros para custear o seu desenvolvimento, não estão tão sujeitos a flutuações mercadológicas derivadas de brigas por fatias de mercado, e não deixam de existir em janelas de tempo tão curtas.

Essa análise incentiva muito o estudo, o uso e o ensino do sistema Blender, cuja longevidade e disponibilidade vêm sido garantidas já há duas décadas.

---

[29] Entrevista sobre o lançamento do *software Quase-Cinema 1*, meu projeto de graduação em Artes Plásticas na UnB:< www.youtube.com/watch?v=dg8hXzC47EY >

[30] História do software de criação multimídia Director:
< https://en.wikipedia.org/wiki/Adobe_Director >

Software	Data de lançamento	Data de obsolescência	Observações
Macromedia Director	1985	2013	Vendido da Macromedia para a Adobe e então descontinuado.
Softimage 3D	1988	2015	Negociado entre as empresas Softimage, Microsoft, Avid e finalmente para a Autodesk.
Macromedia Flash	1996	2020	Vendido da Macromedia para a Adobe. Será descontinuado em 2020.
Apple Final Cut Pro 7	1998	2011	Substituído por um software diferente, com o nome ligeiramente diferente (Final Cut Pro X).
Blender 3D	1998	*	Futuro promissor para a Fundação Blender, que acaba de lançar uma importante versão do seu software livre, o Blender 2.80.

Tabela 4. Ciclo de vida útil de alguns softwares multimídia.

Neste capítulo vimos o estado atual do campo de criação artística algorítmica, principalmente no que diz respeito à programação de código ao vivo, performática. Também pudemos contemplar a diversidade de expressão em um campo híbrido, que tangencia os campos das ciências humanas e exatas: o código como arte, em nível semântico, visual e sonoro.

Capítulo 2. Programação Sonora

"A ideia se torna a máquina que faz a arte."
Sol LeWitt

Neste capítulo falaremos sobre linguagens de programação de computador que fizeram – e fazem – parte do meu universo de experimentação sonora. Será meu chapéu de mágico ou caixa-preta, com a finalidade de contribuição para a cultura de desenvolvimento.

Minhas raízes como artista de arte generativa vieram do campo do vídeo, da imagem em movimento. Advindo da fruição de obras audiovisuais contemporâneas, meu interesse pela música emergiu da percepção de obra completa, no sentido de preenchimento do espaço sensorial.

Sobre a desmistificação do ato de programar computadores, cito o pesquisador Douglas Rushkoff, em sua obra clássica *Programe ou Seja Programado*:

> "Quando os seres humanos adquiriram a linguagem, aprendemos não só a ouvir, mas também a falar. Quando fomos alfabetizados, aprendemos a ler e a escrever. E agora, ao nos movermos para uma realidade cada vez mais digital, devemos aprender não só a usar os programas, mas também a fazê-los." (RUSHKOFF, p.8)

2.1 Live coding textual

As linguagens de *live coding* são sistemas de programação de *software* com uma característica em comum: os resultados da criação de algoritmos de programação podem ser ouvidos e vistos pelo autor e pelo público em tempo real, ou seja, enquanto estão sendo concebidos e digitados no teclado do computador.

Historicamente, a primeira performance conhecida de programação ao vivo foi realizada pelo artista Ron Kuivila, em 1985, no instituto de pesquisa STEIM, em Amsterdã. Já prevendo o quão arriscadas tais práticas são – artística e tecnicamente –, a apresentação acabou, após meia hora, com o computador travado (BLACKWELL, p.123).

Sobre os sistemas de produção sonora computacional, Blackwell classifica três grupos:

1) Os *softwares* tradicionais, chamados de *Digital Audio Workstations (Estações de Trabalho com Áudio Digital)*, tais como *Ableton Live* e *Apple Logic Pro*, que têm interfaces e rígidos fluxos de trabalho preestabelecidos em torno do conceito de linha do tempo, permitindo pouca flexibilidade de programação de lógica;

2) Programas de criação com fluxograma gráfico, tais como *MaX/MSP* e *PD* (*Pure Data*), que permitem maiores graus de manobras algorítmicas e composicionais;

3) Programas de composição com programação textual, que, embora mais difíceis de se dominar, trazem maiores possibilidades de compensação por meio da exploração das possibilidades programacionais.

Atualmente existem dezenas de sistemas, cada um com atributos técnicos e capacidades criativas diferentes. Uma característica em comum entre a maioria das linguagens de programação ao vivo é a sua distribuição como *software* livre, garantindo direitos especiais aos usuários: além da possibilidade de uso, a possibilidade de análise e modificação de funcionalidades do sistema original. A questão do *software* livre vai além da gratuidade, interferindo em questões sociais de reorientação de poder e conhecimento. (FISCHER, 2008)

2.2 *ixi lang*

O *ixi lang* é uma linguagem desenvolvida pelo pesquisador Thor Magnusson, da Universidade de Sussex, na Inglaterra, em 2011. A iniciativa e seus desdobramentos são relatados na pesquisa acadêmica *ixi lang: um parasita para live coding no SuperCollider*. A principal característica do sistema *ixi* é a possibilidade de se trabalhar com um código mutante, ou seja, uma espécie de programação na qual o código pode ter uma espécie de vida própria, podendo se transformar durante a execução da obra sonora.

Minhas primeiras experiências com programação ao vivo integraram o sistema ixi lang ao meu *software* de edição de vídeo ao vivo *Quase-Cinema 2*. A versão do ixi lang que funcionava entre 2011 e 2013 permitia a mesclagem do texto de programação do ixi com outros programas do *macOS* sem a necessidade de outro programa agenciando a conexão. Além do sistema de programação ainda textual, o projeto *ixi* disponibiliza um conjunto de *softwares* com interfaces visuais para criação musical aliada ao *ixi lang*: o *ixi Quarks*.

Os projetos de *software* livre também estão sujeitos à obsolescência: o sistema *ixi lang* não tem mais sido desenvolvido desde outubro de 2012, data de lançamento da versão 4 do programa. Os autores dos softwares *ixi lang* e *Sonic Pi* – Thor Magnusson e Sam Aaron – anunciaram, na conferência *Live Coding Madrid 2019*, seu plano de integração de alguns conceitos e comandos do *ixi lang* a futuras versões do *Sonic Pi*.

Imagem 13. Integração dos sistemas ixi lang e Quase-Cinema 2 na obra MeditaMáquina.

2.2.1 *Homenagem a Wlademir Dias-Pino*

Como exemplo de trabalho criado com o sistema *ixi lang*, apresento minha obra *Homenagem a Wlademir Dias-Pino*[31], uma peça audiovisual não interativa de 5 minutos de duração. Nessa obra, programei um mapeamento das letras constituintes de trecho da poesia *A Ave*, escrita pelo poeta brasileiro Wlademir Dias-Pino (1927-2018). *A Ave*, livro lançado em 1957, foi um marco inicial da poesia visual no Brasil. O vídeo – um registro do processo de *live coding* com o sistema *ixi lang* – se vale da possibilidade de "embaralhamento" dos elementos constituintes da programação, misturando as letras, mas respeitando os espaços em branco do texto original.

[31] Vídeo da obra "Homenagem a Wlademir Dias-Pino" disponível em: < www.youtube.com/watch?v=S32Z1j0Mg1E >

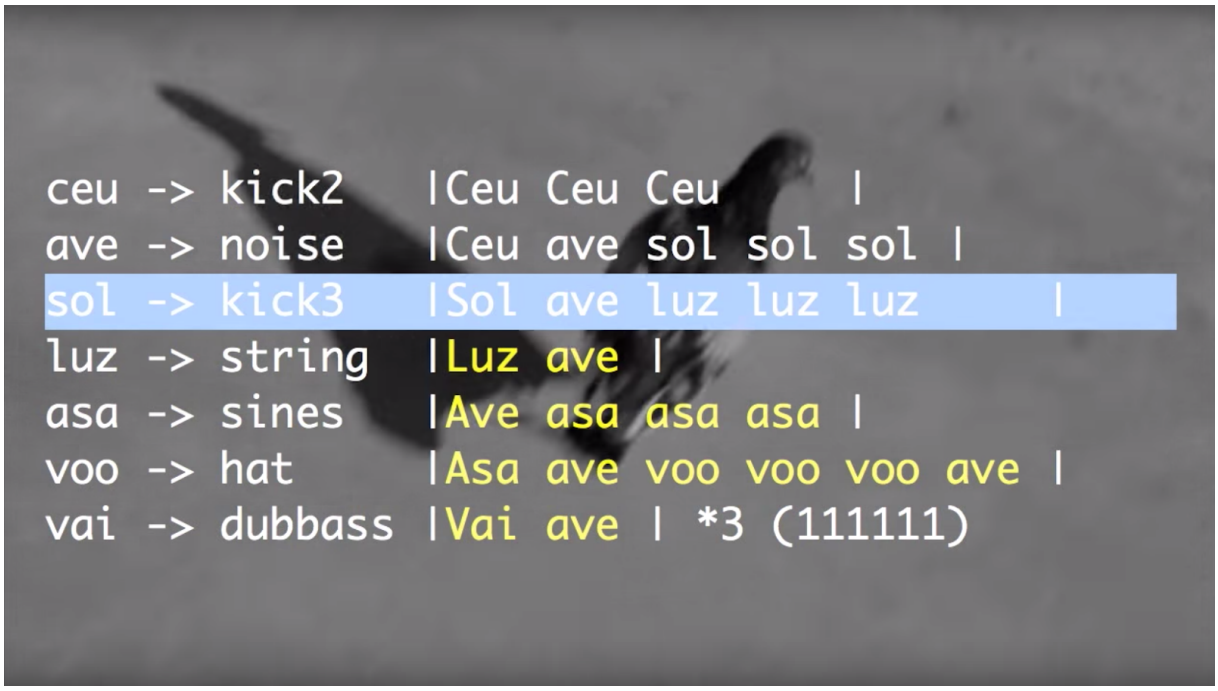


Imagem 14. Quadro do vídeo Homenagem a Wlademir Dias-Pino.

A mutação da programação é uma das características mais marcantes do sistema ixi lang: o código-fonte, propriamente dito, escrito pelo músico, é modificado como resultado da execução do código. Como exemplo palpitante de coautoria entre o artista, o computador e o acaso, vejamos como a seguinte linha se pode transformar dentro do próprio código.

```
ceu -> kick2 |Ceu Ceu Ceu     |
```

Após a execução do comando **swap ceu**, o sistema *ixi* troca o conteúdo do agente **ceu**, transformando a linha em, por exemplo:

```
ceu -> kick2 |CCu eeu ueC    |
```

O resultado da troca pode ser visto na tela do *performer*, assim como percebido na mudança do padrão musical sendo executado; no exemplo acima, uma sequência de tons de bumbo de bateria (*kick*). Caso o *performer* queira guardar estágios das mutações textuais, que constituintes das performances com o *ixi lang*, podem-se salvar vários arquivos de texto a cada estágio desejado.

Essa breve indicação por parte da máquina de que um dia poderá se programar sozinha é um vislumbre de um pós-maquínico, um computador realmente pensante, para além dos comandos pré-imaginados. Aqui os conceitos de humano e máquina começariam a se misturar. Sendo visto como uma intromissão ou como uma

colaboração, é fascinante e assustador, ao mesmo tempo, ver o seu código sendo modificado. A pesquisadora Lucia Santaella em seu artigo *O corpo biocibernético e o advento do pós-humano* já tenta descrever o que seria um estágio mesclado entre homem e máquina, pós-biológico, citando a abertura de uma nova fronteira entre mente e matéria: "[...] nossa visão daquilo que constitui o ser humano está passando por profundas transformações". (SANTAELLA, p.130)

Para melhor ou pior, neste momento histórico, o resultado é sempre surpreendente e desperta novas atitudes do *performer*, induzindo um real e dinâmico diálogo entre homem e máquina.

2.2.2 *MeditaMáquina*

MeditaMáquina é uma composição audiovisual e performática com duração de 23 minutos[32], exibida no encontro *Arranjos Experimentais: Cultura Numérica Audiovisual*, realizado no Paço das Artes pela Escola de Comunicação e Artes da USP, em 2013. A curadoria do evento foi feita pela pesquisadora Patrícia Moran e pelo VJ Spetto – artista pioneiro brasileiro da criação audiovisual ao vivo. O tema da performance foi uma visão fragmentada do urbanismo geometricamente regrado de Brasília, continuando a pesquisa visual sobre a as formas da cidade iniciada com a obra *Eixo X*.

Conceitualmente, a obra nasceu de uma ideia de expressão da cidade como sistema informático, da cidade como estúdio, da cidade como musa. Como capturar e processar os sons da cidade? Aqui já se esboçava o meu interesse teórico pelas questões sonoras e imagéticas proporcionadas pela experiência urbana de Brasília.

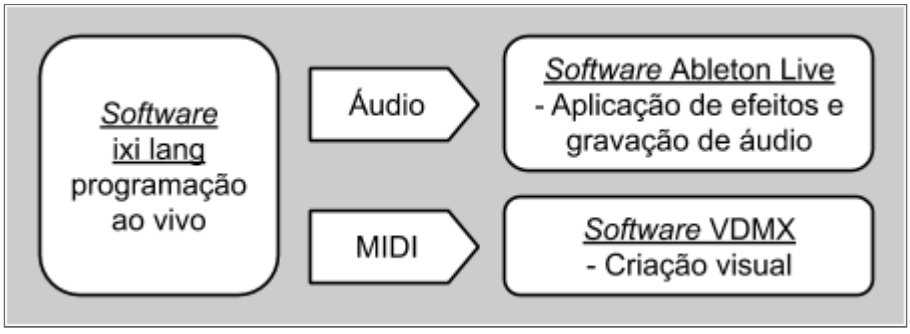


Gráfico 1. Esquema técnico da performance MeditaMáquina.

[32] Vídeo com registro completo da performance *MeditaMáquina*:
< www.youtube.com/watch?v=qyvysqdWoKE >

O software de criação visual *VDMX* teve papel importante na realização desta performance. Ao receber comandos *MIDI* do sistema *ixi lang*, sincronizava os vídeos de cada trecho da performance de acordo com as batidas da música. Além disso, um dos pontos fortes do *VDMX* é a capacidade de capturar a tela de outros programas e misturá-la com outros vídeos. Graças a isso, pude compor diversos vídeos com a interface do *ixi lang*, mostrando o texto da programação ao vivo sendo digitado.

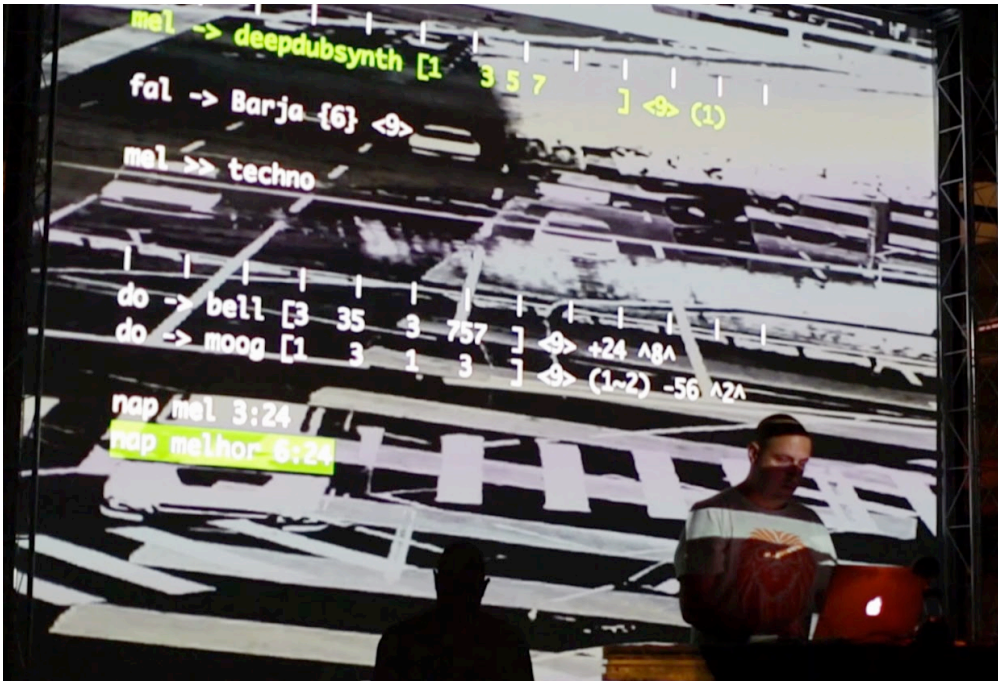


Imagem 15. Performance de programação ao vivo da obra MeditaMáquina.

Em uma performance de programação ao vivo, analogamente a um concerto de música ao vivo presencial, onde se vê o virtuosismo dos músicos, podem-se perceber as ações de escrita textual do artista transformando-se em sons e ritmos. Chamo a atenção para uma percepção do compositor Steve Reich: “Eu estou interessado nos processos perceptíveis. Quero poder ouvir o processo acontecendo através do som da música.” (REICH, p.1) É visível certa tensão na performance, tendo em vista a minha consciência da fragilidade do processo de programação ao vivo, com tantas variáveis e conexões que podem não funcionar a cada apertar de tecla. O erro é tão protagonista quanto o acerto; é visível e iminente.

2.3 *Sonic Pi*

O sistema de programação musical *Sonic Pi* foi desenvolvido pelo pesquisador e artista sonoro Sam Aaron, no Laboratório de Computação da Universidade de Cambridge, na Inglaterra. O objetivo inicial do projeto foi ensinar conceitos de programação e conceitos musicais ao mesmo tempo, tendo como público-alvo crianças a partir de 8 anos de idade. Como na Inglaterra o ensino de computação é obrigatório desde o ensino elementar, é interessante criar ferramentas lúdicas de aprendizado. Nos sistemas de *live coding* – não somente no *Sonic Pi* – a recompensa por realizar tarefas simples de programação é instantânea: a criação de sons experimentais ou a reprodução de peças musicais.

A primeira versão do *Sonic Pi* foi desenvolvida em 2012, sob encomenda da Fundação Raspberry Pi[33]. A Fundação criou a linha de computadores em miniatura Raspberry Pi, com foco em ensino financeiramente acessível de informática. Os computadores precisavam de mais softwares para ensino de programação (e música). Desta demanda com apoio financeiro nasceu o *Sonic Pi*, e dela o sistema herdou seu nome. Dadas a simplicidade de programação e a conexão direta entre comandos e realização de saídas sonoras, o *Sonic Pi* é uma opção prática e viável de aprendizado como primeira linguagem de criação de *software*. (AARON, 2014)

Vale notar que o *Sonic Pi* foi o segundo sistema de *live coding* criado por Sam Aaron. O pesquisador já havia desenvolvido o sistema *Overtone* [34], que utiliza o *software SuperCollider* para síntese sonora e a linguagem *Clojure* para realização da programação musical. O sistema mostrou-se muito complicado para ser compreendido pelo público e também para ser usado casualmente. Ao se realizarem processos de criação sonora no *Sonic Pi*, acabam-se usando – e aprendendo – conceitos que são úteis em quase todas as outras linguagens de programação fora do universo de *live coding*.

O motivo da minha pesquisa e minha produção terem um foco maior no sistema *Sonic Pi* é, principalmente, sua grande facilidade de uso e, consequentemente, de ensino. Seu sistema de programação é especialmente fácil; em nenhum outro sistema de programação é tão simples tocar uma nota musical. Por exemplo, o comando isolado **play**

---

[33] Site da Fundação Raspberry Pi, criadora dos micro-computadores Raspberry Pi: < www.raspberrypi.org >

[34] Site do sistema Overtone: < https://overtone.github.io >

:C já toca uma nota Dó. O *Sonic Pi* adota a convenção de notação musical em que cada letra corresponde a uma nota musical.

Dó	Ré	Mi	Fá	Sol	Lá	Si
C	D	E	F	G	A	B

Tabela 5. Convenção de correspondência de letras com notas musicais.

Uma vez escolhido um instrumento (**use_synth :piano**, por exemplo), podem-se tocar, com o comando **play**, notas musicais, representadas por letras ou números.

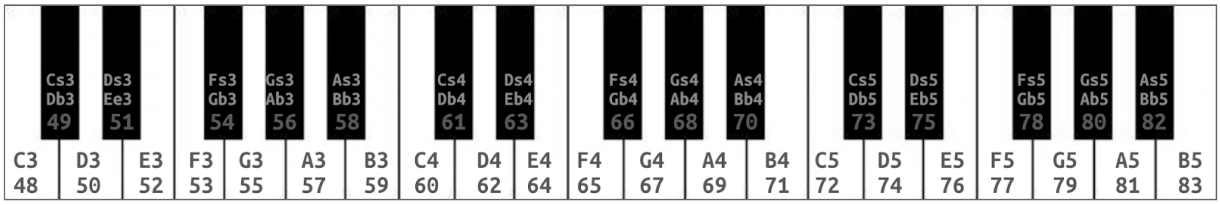


Imagem 16. Parte da correspondência entre notas e numeração pelo padrão MIDI.

O *Sonic Pi* trabalha com notação musical por números ou letras. Podemos tocar as notas pelo mapeamento do teclado de um piano pelo padrão MIDI. Nesse padrão, 12 números correspondem a uma oitava. Diferentemente de um piano, podemos também tocar teclas "intermediárias", inexistentes em um teclado (tais como 60,5 – entre o Dó e o Ré), sintetizando tonalidades diferentes, impossíveis para um piano normal. Observo aqui uma relação com os "pianos preparados" de John Cage, nos quais o artista inseria objetos dentro dos instrumentos para modificar a sonoridade resultante das cordas ou dos martelos.

```
1  use_synth :piano
2  # as 3 próximas linhas criam o mesmo som (Dó,Ré,Mí,Fá,Sol,Lá,Sí)
3  play_pattern [:C, :D, :E, :F, :G, :A, :B ]; sleep 7
4  play_pattern [:C4,:D4,:E4,:F4,:G4,:A4,:B4]; sleep 7
5  play_pattern [60,  62, 64, 65, 67, 69, 71]; sleep 7
```

Tabela 6. Código tocando sequências de notas musicais com o Sonic Pi.

As letras :C, :D, :E, :F, :G, :A, :B correspondem às notas musicais Dó, Ré, Mi, Fá, Sol, Lá, Si. As notas sustenidas são representadas com a letra **s** (:Cs4). As notas bemol são

representadas com a letra **b** (:Cb4). Ao representar as notas como letras, podemos especificar a oitava da nota:

- C é igual a C4.
- C3 é uma oitava abaixo (mais grave) do que C4.
- C5 é uma oitava acima (mais aguda) do que C4.

Interface do *Sonic Pi*

O *Sonic Pi* é um programa extremamente fácil de se baixar, instalar e, finalmente, usar, para apresentações ao vivo ou gravando o resultado sonoro. A interface gráfica do Sonic Pi é dividida em:

- Painel de programação: aqui são escritos os programas;
- Painel de log: janela de histórico, registro de erros e saída de texto;
- Painel de preferências e painel de ajuda;
- Painel de botões: comandos de tocar, parar, gravar, salvar o arquivo de texto ou de áudio;
- Abas: 9 áreas para programas diferentes.

Na figura abaixo vemos as áreas de programação, análise visual da geração sonora e retorno de informações (*log*).

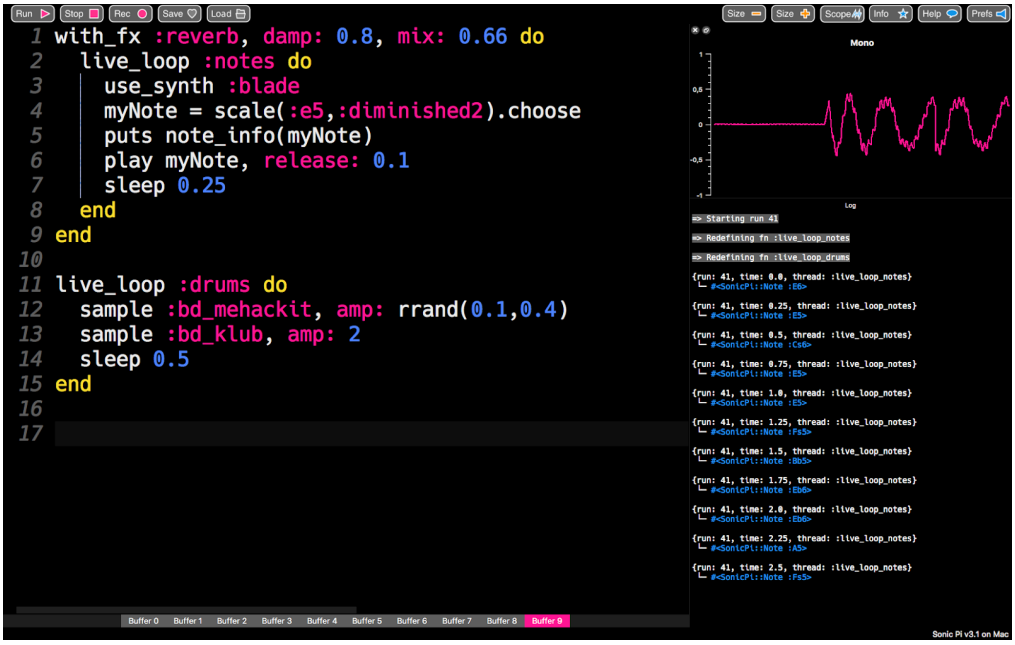


Figura 17. Interface do software de programação musical Sonic Pi.

Geração de sons

Os sons podem podem ser criados com as funções **play** e **sample**. O comando **play** toca notas musicais com o instrumento escolhido previamente; a versão 3.1 do Sonic Pi conta com 40 instrumentos (sintetizadores, ou *synths*, na nomenclatura do *Sonic Pi*). Já O comando **sample** toca arquivos de áudio (WAV, AIFF, FLAC) disponíveis dentro do Sonic Pi ou localizados no seu computador. Todos os sons produzidos pelo Sonic Pi são derivados dessas duas funções, com a exceção da função **sound_in**, que insere na composição o som captado pelo microfone ou placa de som do computador.

Tocando notas de uma escala em arpejo

Esse tipo de programação realiza uma improvisação de performance musical escolhendo notas a partir de uma escala pré-determinada.

```
1  live_loop :notas do
2    use_synth :pluck
3    play (:c3, :spanish).choose
4    sleep 1
5  end
```

Tabela 6. Código tocando notas escolhidas aleatoriamente a partir de uma escala musical.

Linha **1**: Inicia o *loop* (trecho a ser repetido).

Linha **2**: Seleciona o instrumento de síntese de violão.

Linha **3**: Escolhe uma nota da escala espanhola, com raiz na nota Dó da 3ª oitava (C3).

Linha **4**: Espera 1 compasso. No padrão de 60 BPM, equivale a 1 segundo.

Linha **5**: Retorna ao início do *loop*.

Opções do comando play

amp: define o volume da nota tocada;
amp_slide: duração (medido em *beats* - batidas) da mudança de volume;
pan: posição no campo estéreo - esquerda/direita (-1 a 1);
pan_slide: duração (em *beats*) da mudança de pan (0 a 1);
attack: duração (em *beats*) do início da nota;
sustain: duração (em *beats*) da nota em seu volume total;
release: duração (em *beats*) do final da nota.

Jogando dados

Considero o acaso um dos pilares da criação de obras generativas, um dos temperos que torna a obra, a cada execução, uma surpresa. Vejamos como os números aleatórios são gerados normalmente por computadores: ao criarmos um programa para escolher números entre zero e cem, ao ser executado, o *software* gera os número 42, 11, 7, e assim por diante. Quando pararmos o programa e pedimos para que se execute o *software* novamente, aparecem os números 42, 11, 7 – novamente! Ou seja, normalmente o computador gera números aleatórios, mas, ao fazê-lo mais uma vez, cria a mesma sequência numérica. Há formas de usar criativamente ou de quebrar esse ciclo de repetições. A forma que uso com maior frequência é fazer com que o *software*, sempre que executado, mude sua origem (semente, *seed*) de randomização, baseado no segundo do horário em que o programa for iniciado (o horário 22:45:12 utilizaria como semente o número 12).

Outra forma de conseguir números realmente randômicos é utilizar variáveis que sabemos que são caóticas. O *site* random.org[35] disponibiliza números aleatórios e gravações sonoras de ruído baseados em coleta de ruídos atmosféricos do planeta. E se perguntarmos ao computador qual o horário atual (somente os segundos) e usarmos esse número para tocar notas? É simples: após descobrirmos tal número (que vai de 0 a 60 e depois se reinicia), adicionamos a ele o número 40; assim tocaremos as notas de 40 a 100. Se não somássemos 40, tocaríamos notas muito graves (algumas inaudíveis), de 0 a 60.

```
1  use_bpm 60
2  use_synth :fm
3  128.times do
4    t = Time.new
5    puts t.sec, ' -> ', t.sec + 40 #mostra a nota escolhida
6    play t.sec + 40, amp: 2.2
7    sleep 1
8  end
```

Tabela 7. Transformando cada segundo, de 0 a 59, em uma nota musical.

Uma variação do código anterior, com o ponteiro do segundo modificando também parâmetros do sintetizador (*divisor* e *depth*):

[35] *Site* para geração de números, ruídos sonoros randômicos e outras combinações, tal como sequências de números para jogar na loteria: < www.random.org >

```
1  use_bpm 120
2  use_synth :fm
3  with_fx :normaliser, mix: 0.5 do
4    128.times do
5      t = Time.new
6      y = ((t.sec+1)/61.0) #gera fração entre 0.0 e 1.0
7      play t.sec + 40, amp: 1.6, divisor: 2.0 * y, depth: 32.0 * y
8      sleep 0.5
9    end
10 end
```

Tabela 8. Modificando parâmetros de um sintetizador do Sonic Pi.

Dessas combinações e experimentações com o código-fonte, matéria prima tão maleável que dá lugar ao acaso e ao erro, materializa-se o fazer da obra de arte. Citando o pesquisador de novas mídias Lev Manovich, a arte encontra-se entre o claro e o escuro, entre a técnica e o acidente. (MANOVICH, 2018, 31'20")

Outra forma de inserção de imprevisibilidade nos resultados audiovisuais é a conexão dos dados e algoritmos existentes com a interação humana, seja no ato da programação, seja no ato de fruição da obra.

Fiat vox: fazendo o computador falar

O áudio dessa ação é executado diretamente pelo sistema operacional; pode ser ouvido com outros sons do *Sonic Pi*, mas não pode receber efeitos sonoros nem ser gravado pelo *Sonic Pi*. Uma maneira de gravar o som de voz juntamente com a composição do *Sonic Pi* é usar programas de captura interna de áudio, tais como o *software* comercial *Loopback Audio*[36] ou o *software* livre SoundFlower[37], ambos para o sistema *macOS*. Essa função só funciona com um recurso de síntese do terminal do sistema operacional, que não existe em sistemas *Linux* e *Windows*, mas apenas em sistemas *macOS*.

```
1  system("say Sonic Pi")
2  sleep 2
3  system("say -v luciana -r 300 i am Sonic Pi")
4  sleep 2
5  system("say -v felipe -r 100 i am Sonic Pi")
```

Tabela 9. Utilizando a síntese do voz nativa do sistema macOS.

[36] Página do programa *Loopback Audio:* < www.rogueamoeba.com/loopback >

[37] Página do programa *Soundflower:* < https://github.com/mattingalls/Soundflower >

Linha **1**: Solicita ao sistema *macOS* que pronuncie a frase "Eu sou o *Sonic Pi*";

Linha **2**: Aguarda dois segundos até o próximo comando;

Linha **3**: Solicita ao sistema macOS que pronuncie a frase "Eu sou o *Sonic Pi*" com a voz de sonoridade brasileira "Luciana", com a velocidade de execução 300 (rápida);

Linha **4**: Aguarda dois segundos até a execução do próximo comando;

Linha **5**: Solicita ao sistema *macOS* que pronuncie a frase "Eu sou o Sonic Pi" com a voz de sonoridade brasileira "Felipe", com a velocidade de execução 100 (lenta).

Sound Design com *Sonic Pi*

O Sonic Pi dispõe de vários tipos de sintetizadores que podem tocar notas musicais ou receber um sinal sonoro de microfone ou instrumento externo (**sound_in**).

Sonoridade	**Sintetizadores do Sonic Pi**
Acústicos	piano, pluck
Inspiração em sintetizadores reais	*prophet, tb303*
Ondas sonoras básicas	*pulse, sine, saw, square, supersaw, tri*
Ondas modificadas com oscilação	*mod_beep, mod_fm, mod_pulse, mod_sine, mod_saw, mod_tri*
Ondas modificadas com oscilação	*dpulse, dsaw, dtri*
Inspirada em sons de *video games*	*chipbass, chiplead*
Sons ambientes	*dark_ambience, growl, hollow, hoover*
Sinos	*pretty_bell, dull_bell*
Ruídos	*noise, bnoise, chipnoise, cnoise, gnoise, pnoise*
Entrada de som via interface de áudio	*sound_in, sound_in_stereo*
Outros	*beep, fm, subpulse, tech_saws, zawa*

Tabela 10. Tipos de sintetizadores inclusos no software Sonic Pi.

Podem ser aplicados efeitos em cada som emitido pelo *Sonic Pi*, sejam notas musicais sintetizadas ou a reprodução de trechos de gravações sonoras (*samples*). O que

torna o Sonic Pi um sistema extremamente poderoso é a possibilidade de encadeamento de efeitos em cascata – o resultado de um filtro alimentando a entrada de som de outros filtros.

2.3.1 Oficinas *CODE MUZIK*

O primeiro objeto de estudo desta pesquisa de doutorado foram as pesquisas realizadas entre junho e novembro de 2016, cujo foco foi a realização de um estudo sobre as possibilidades criativas e educacionais da prática de programação de *software* para criação sonora. O estudo visa a mapear a abrangência da prática artística e as potencialidades do ensino de criação musical por meio de desenvolvimento de *software* para computadores. A possibilidade de replicação das experiências de ensino realizadas é potencializada pelo *website* do projeto, que conta com extenso material didático de sons e trechos de código de programação sonora com o *Sonic Pi*.

Sobre a contribuição das oficinas para a pesquisa, pude observar um crescimento da minha autonomia como programador, adquirida com o desembaraçar de questões técnicas de grupos com participantes de perfis de experiência diferenciados com relação a programação e criação musical, que também trouxeram a vivência de como ativar o entusiasmo dos participantes em criar programas de computador.

Imagem 18. Oficina CODE MUZIK no Anexo do Museu Nacional da República.

Foram realizados 6 módulos de oficinas e performances audiovisuais, que contaram com a participação de um músico convidado, a fim de propiciar um diálogo e interação entre música com instrumentos tradicionais e música com programação de *software*. Esses mesmos músicos participaram de análises sobre o material criado nas oficinas, permitindo um mapeamento para multiplicação das experiências por outros grupos. Foram criados, durante o projeto, relatório de pesquisa, apostila didática e videoaulas para acompanhar o processo de aprendizado nas oficinas e nos desdobramentos do projeto. O projeto vislumbrou a mistura e o crescimento mútuo das ciências humanas e exatas, uma vez que as práticas desenvolvidas e os profissionais envolvidos são multidisciplinares (desenvolvimento de *software*, música e artes visuais). O computador já intermedia o ensino, a produção e, principalmente, o consumo de música.

O projeto propôs uma importante mudança de paradigma: transformar a postura de consumo em investigação, exploração e criação. Essa mudança também abre portas para uma ampliação do escopo ferramental de estudantes de ciências humanas e exatas, trabalhando, ao mesmo tempo, capacitação para as Artes e para Tecnologias da Informação. A música foi utilizada como foco das atividades, que caminharam em confluência com as artes visuais, com resultado de expressão audiovisual em novas mídias.

Foram latentes as sensações de empoderamento e apropriação tecnológica vivenciadas pelos participantes. Durante a prática de ensino das oficinas, pude prever, a partir de experiências relatadas por Sam Aaron e Alexandra Cárdenas, que a experiência de aprendizado poderia ser usufruída inclusive por crianças. As práticas similares já realizadas por mim (oficinas, performances artísticas, o próprio mestrado em arte-educação – usando meu *software Quase-Cinema VJ*) indicaram um grande potencial de valorização do processo lúdico da criação musical no ensino de programação e vice-versa – uma vez que os conceitos e processos computacionais tradicionais podem dar inesperados e surpreendentes frutos musicais. A proposta baseou-se completamente em uso de *software* livre, o que possibilitou a multiplicação dos recursos criativos com o menor ônus financeiro possível.

As atividades do ciclo de oficinas incluíram:

- Criação de material didático, visando à multiplicação das ações de capacitação;
- Realização de oficinas presenciais;
- Realização de performances;
- Análise do material apresentado nas performances;

- Avaliação do ferramental proposto como ferramentas educacionais multidisciplinares.

As oficinas foram oferecidas gratuitamente e patrocinadas pelo primeiro edital de Pesquisa Musical do Fundo de Apoio à Cultura (FAC) da Secretaria de Cultura do Governo de Brasília (2016). O projeto também contou com a parceria do Museu Nacional do Conjunto Cultural da República de Brasília, instituição que apresentou o projeto e contribuiu com apoio técnico e disponibilização de espaços físicos para a realização das oficinas presenciais (Auditório Kalunga e Anexo do Museu). Foram locados os seguintes equipamentos para a realização das oficinas:

- 2 projetores de vídeo de 6.000 *lumens*;

- 2 TV's de 44 polegadas;

- Mesa de mixagem e gravação de áudio multipistas de 8 canais;

- Cabeamentos de áudio para os participantes;

- 2 Microfones sem fio;

- 10 computadores para uso dos participantes.

Ramiro Galas foi o primeiro convidado do ciclo de seis oficinas presenciais. Ramiro é um músico e pesquisador com trabalhos na área de música ao vivo e DJ e produtor musical do dueto Forró Red Light.

Imagem 19. Oficina CODE MUZIK 1 com o músico convidado Ramiro Galas.

Uma das maiores descobertas/invenções da primeira oficina foi a de como programar um sequenciador de bateria eletrônica diretamente pelo *software Sonic Pi* – e

somente com ele. Essa técnica mostrou-se de extrema facilidade e utilidade para as criações de todas as outras oficinas já realizadas durante esta pesquisa. Os objetivos principais da oficina com Ramiro foram realizar uma introdução geral ao processo de programação ao vivo e a traduzir os ritmos e procedimentos de criação de música eletrônica para o formato de criação do *Sonic Pi*.

A segunda oficina foi realizada com Vavá Afiouni[38], eclético multi-instrumentista e compositor, atuante em carreira solo e com a banda Toró de Palpite. Buscaram-se durante a oficina a produção de códigos musicais de estilos diversificados e o estudo de metodologia de transcrição de melodias para a linguagem do *Sonic Pi*. Vavá mostrou-se interessado em participar de outros módulos da oficina, com o intuito de realizar um programa de ensino de teoria musical para um grupo já iniciado em programação musical.

Imagem 20. O multi-instrumentista Vavá Afiouni durante a segunda oficina.

No exemplo de código a seguir, composto durante a oficina de baixo, está a linha de baixo da música *Billie Jean* (1982), do cantor e compositor Michael Jackson. A forma de colocar as notas na programação seguem o padrão do sequenciador de notas musicais desenvolvido na oficina anterior, feita com Ramiro Galas. O objetivo foi criar, além de uma reprodução da música em questão, variações sobre o mesmo tema e ritmo. Um efeito desse processo de criação baseado em estilos existentes é uma compreensão mais profunda e uma contribuição à análise do gênero musical em questão (COLLINS, 2003).

[38] Site do músico Vavá Afiouni: < www.vavaafiouni.com >

```
live_loop :billieJean do
  use_synth :beep
  play :G4,  amp: ring(1,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0)[y]
  play :A4,  amp: ring(0,0,0,1, 0,0,0,0, 0,0,0,0, 0,0,0,0)[y]
  play :Bb4, amp: ring(0,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0)[y]
  play :A4,  amp: ring(0,0,0,0, 0,0,0,0, 0,0,0,1, 0,0,0,0)[y]
  play chord(:G4,:m7),amp:ring(1,0,0,0, 0,0,0,0)[y]*0.5,release: 6, if one_in(2)
  sleep 1
  y = y + 1
end
```

Tabela 11. Código Sonic Pi de trecho da música Billie Jean.

O convidado seguinte foi Gérson Deveras: poeta, compositor e cantor que lançou discos solo e com a banda Os Cachorros das Cachorras. Gérson falou sobre o processo de musicalização da palavra e da colocação da palavra falada e cantada por cima de batidas musicais. Os participantes da oficina foram incentivados a criar dois tipos de materiais musicais: bases musicais com ritmos brasileiros para uso em posterior improvisação vocal, e textos para leitura, gravação e posterior manipulação experimental com a metodologia de de criação de *software*. Foi explorada a experimentação e produção de *software* para manipulação de vozes gravadas, por meio do uso de filtros e recortes com o sistema de *samples* (amostras musicais e trechos de áudio gravados) do *Sonic Pi*.

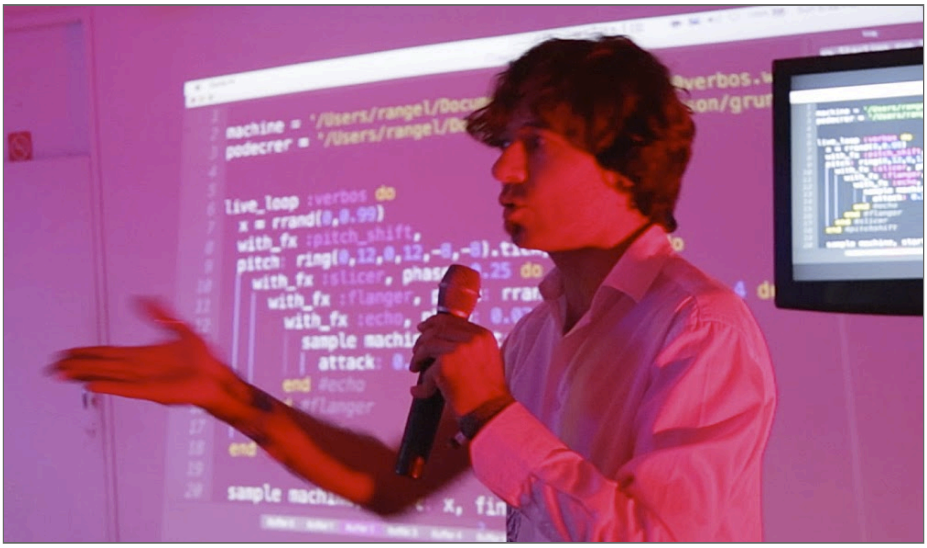


Imagem 21. Gérson Deveras em improvisação vocal com música de código.

Graças à fluente criatividade, flexibilidade e controle vocal do artista convidado, ficou muito claro na oficina o potencial de contribuição entre homem e máquina. Nas horas finais da oficina, quando estimulada a improvisação e a mesclagem de

ideias, os participantes puderam executar suas composições em *software* enquanto o cantor encaixava sobre as batidas letras de canções e poesias.

Sérgio "Cepa" Azevedo é guitarrista e foi o convidado da oficina seguinte. Graças à familiaridade do público com instrumentos como a guitarra e o violão elétrico, essa foi uma das oficinas mais demandadas. Também foi a oficina na qual vimos os participantes trazerem seus próprios instrumentos musicais e participarem ativamente das performances de final de dia – tanto com suas guitarras quanto com código.

Imagem 22. Oficina com participação de Cepa no Museu Nacional da República.

A oficina com o guitarrista Cepa teve como foco o uso do *Sonic Pi* como um processador de efeitos sonoros em tempo real. Isto é, criamos composições / sistemas capazes de capturar o áudio proveniente dos instrumentos (guitarras e violões elétricos) e aplicar efeitos imediatamente, tais como pedais de guitarra. A comparação com pedais de guitarra foi inevitável, e constatamos a flexibilidade e um grande potencial criativo com o sistema programático, uma vez que pedais de guitarra costumam fazer um só efeito, com poucos parâmetros configuráveis. Trabalhando os efeitos com o uso do *Sonic Pi*, pudemos atribuir variáveis aleatórias, estáticas ou passíveis de mudanças dinâmicas, às configurações, vinculando os efeitos às ações do performer, em tempo real. Um dos desafios técnicos da oficina foi o de acostumar-nos com as possibilidades dos efeitos em tempo real, especialmente as possibilidades de controle intuitivo de efeitos de eco e retorno (*feedback*).

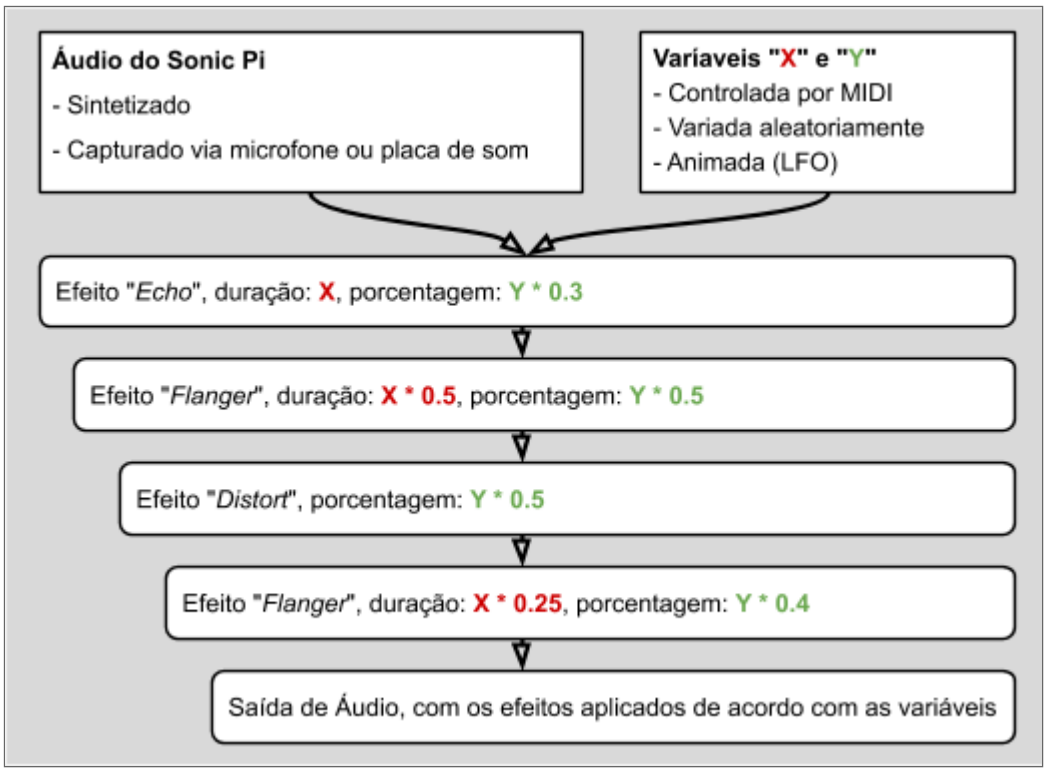


Gráfico 2. Fluxograma de áudio no Sonic Pi com efeitos dinâmicos ao vivo.

A próxima oficina foi com Rodrigo Barata, baterista e DJ atuante nos projetos Criolina e Muntchako. O foco foi a criação de batidas com programação, que em diálogo com as batidas orgânicas do músico convidado pudessem produzir resultados inusitados e frases musicais passíveis de posterior desenvolvimento.

Imagem 23. Rodrigo Barata durante oficina de percussão com Sonic Pi.

Um dos maiores desafios da oficina foi a sincronização entre os *softwares* desenvolvidos pelos participantes da oficina e o músico convidado, com sua bateria "real". Os ritmos explorados foram baseados em Música Popular Brasileira e *rock*. A questão mais discutida durante a oficina de percussão foi: o computador deve acompanhar o músico humano ou vice-versa?

Acabamos percebendo que cada formato tem seu potencial criativo próprio. Porém, se fôssemos fazer uma prova, o ser humano ganharia, devido a sua flexibilidade e capacidade orgânica de adaptação. De acordo com a sua programação, a máquina nunca erra. Mas – ainda – não foi programada para ter sensibilidade ou bom senso para discernir entre o certo e o errado – o que, nós humanos, por muitas vezes, também não temos. Aqui podemos voltar ao referencial de Ferreira Gullar, quando diz que o homem é, por natureza, incompleto, que sempre precisou inventar as suas ferramentas (VIANA, 2005).

Na oficina final, convidei Luiz Oliviéri: músico, artista visual e pesquisador acadêmico, autor de projetos nas áreas de educação e sonificação. A oficina com Oliviéri teve o intuito de reunir todo o conhecimento e técnicas desenvolvidas nas outras oficinas, objetivando práticas de composição e performance musicais. Oliviéri trouxe para os participantes seus conhecimentos teóricos e práticos de arranjamento, traduzindo os preceitos tradicionais para o terreno da programação de *software*. Aprendemos como desmembrar uma composição em várias partes: introdução, desenvolvimento, pontes, e *loops* principais. Foi a oficina em que mais pudemos debater e experimentar o conceito de imaginação, composição e performance musical sem o uso de partituras convencionais. Também foram realizados experimentos de criação sonora com instrumentos caseiros, tais como caneta marcadora laser e utensílios de cozinha, ou um espremedor de laranjas, que quando rodado, criava um efeito intermitente na luz laser.

Introduzimos o conceito de programação de ritmos com algoritmos euclidianos de distribuição de elementos (sonoros). O pesquisador musical Godfried Toussaint, em sua pesquisa The Euclidean Algorithm Generates Traditional Musical Rhythms, fez uma "redescoberta" do tratado de elementos geométricos[39] do matemático grego Euclides (300 A.C.), relacionando-o a ritmos de música tradicional (*world music*) de todo o mundo (TOUSSAINT, 2005), à exceção da indiana. Euclides desenvolveu um cálculo de distribuição de elementos em espaços determinados (que podem ser torres em uma muralha medieval ou notas musicais em um medida temporal!). O *Sonic Pi* possui uma

[39] História e influência nas ciências do tratado Elementos de Euclides:
< https://en.wikipedia.org/wiki/Euclid's_Elements >

função que distribui notas ou *samples* em espaços de tempo determinados: o comando **spread** (espalhar), que recebe dois parâmetros: itens e vagas.

Imagem 24. Luiz Oliviéri demonstrando sonificação de frequências de raio laser.

Um fluxo de trabalho que se repetiu – por mostrar-se frutífero técnica e criativamente – foi um ciclo de:

1) Explicação teórica e técnica sobre composição com programação ao vivo;

2) Demonstração de pequenas composições;

3) Período de experimentação pelos participantes;

4) Rodadas de apresentações, atenção individual e troca de idéias entre os participantes sobre caminhos possíveis para cada criação sonora.

Alguns participantes do público destacaram-se na realização das propostas das oficinas. Fernando Mazoni, percussionista do grupo musical performático Patubatê, ao participar da oficina com o artista Gérson Deveras, criou batidas com ritmos brasileiros no sistema Sonic Pi, disponíveis no material didático online *Os Ritmos de Coco e Maracatu*. A atriz e cantora Ingrid Galvão realizou uma série de improvisos vocais, acompanhada de códigos executados por outros participantes da oficina de Luiz Oliviéri. O artista Phil Jones, praticante de música com tecnologia de longa data, mostrou a todos suas composições e técnicas avançadas de programação no *Sonic Pi*.

Que aprendemos com os sistemas desenvolvidos e processos criados? A exploração dos sistemas individualmente, em grupo, e juntamente com os músicos

convidados levou a uma série de conclusões – e inúmeras outras indagações técnicas e criativas a serem desenvolvidas no curso dos estudos de doutorado, iniciados em 2016.

Em que se transforma a pessoa após a oficina? Com certeza em alguém com menos medo de empreender projetos que incorporem código de computador. Os principais objetivos das oficinas foram alcançados: desmistificação do processo de criação com programação de código e capacitação para criação em diversos estilos musicais com o sistema *Sonic Pi*.

Material didático

Fez parte da proposta de pesquisa e desenvolvimento do doutorado em andamento a criação de material didático online sobre a prática artística com a técnica de *live coding*, disponível em uma parte dedicada do *site Quase-Cinema Lab* [40]. Mais do que uma tradução do manual original do *Sonic Pi*, o material explora detalhadamente todos os instrumentos virtuais disponíveis no *software* e apresenta exemplos de código que podem ser executados imediatamente no *Sonic Pi*, incentivando a compreensão e a experimentação por meio da combinação de vários pedaços de código e modificação dos valores atribuídos a variáveis.

A organização do material didático acompanhou a ordem de transmissão de conhecimentos seguida durante as oficinas presenciais, misturando referências teóricas e técnicas com práticas de criação musical via *live coding*.

---

[40] *Link* do material didático sobre *Sonic Pi* produzido para as oficinas:
< www.quasecinema.org/sonicpi.html >

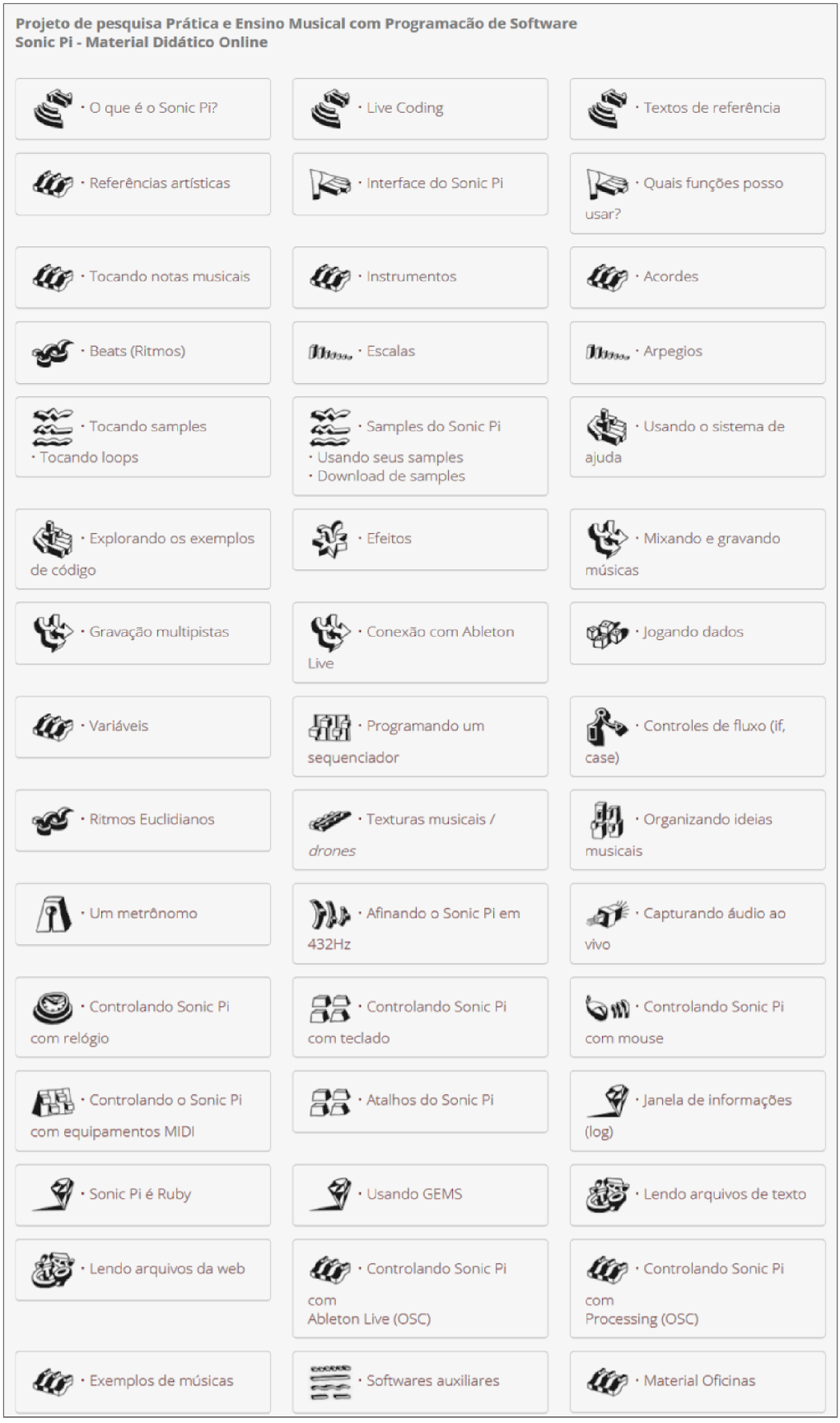


Imagem 25. Website com material didático sobre composição musical com o Sonic Pi.

O material didático online também conta com registros em vídeo das seis oficinas. Duas matérias da Rede Globo de televisão também estão disponibilizadas na página *web* do projeto. Uma das entrevistas foi gravada na segunda oficina CODE MUZIK e outra na demonstração feita no evento *Campus Party Brasília*.

2.3.3 Projeto *Weekly Beats 2014*

É interessante relatar o projeto/desafio *Weekly Beats* (Batidas Semanais) – do qual participei em 2014 e 2016 – para compreender o desenvolvimento da minha obra audiovisual. Durante o ano de 2014, participei de um desafio de criar uma música por semana, totalizando 52 obras[41] durante o ano (feito repetido com regras similares em 2016).

O desafio bianual Weekly Beats não tem prêmios, somente a satisfação de missão cumprida e, principalmente, a evolução das práticas criativas dos participantes. Além de produção musical, propus a mim mesmo que cada uma das minhas obras deveria ser uma música com um vídeo, ou seja, criei 52 obras audiovisuais. A pressão (ou vontade de estar em dia com o "jogo") de ter de pensar, criar e finalizar uma vídeo-arte, entre segunda-feira e domingo, é um incentivo que alavancou minha capacidade criativa como em nenhum ano anterior!

Ao delimitar as características principais das obras, procurei fazer com que os vídeos tivessem sons e visuais conectados com as temáticas da espiritualidade e da conexão do ser humano com o meio ambiente – no caso, mais urbano que bucólico, ainda que Brasília possibilite as duas explorações visuais. Apesar de muitos carros, vemos nas obras – e na cidade – muitas árvores. Os títulos dos vídeos procuram trazer ganchos para a temática, como podemos observar na obra *How to levitate (Como levitar)*. No vídeo, podemos ver meus pés em um plano contínuo, em primeira pessoa, andando sobre a água (em um truque de fotografia e composição de pós-produção).

Durante o desenvolvimento da estética visual das peças, procurei uma coerência por meio de elementos como a edição de vídeo, adotando uma linguagem que fizesse presentes elementos de câmera lenta, repetição e uso de planos únicos ou poucos planos. Outros aspectos que conectaram visualmente as 52 obras foram a paleta reduzida e com uso de cores muito saturadas e o efeito incidental de *pixels* grandes: quadrados de diferentes tamanhos, animados de acordo com as imagens do vídeo.

---

[41] Lista de reprodução com as 52 obras completas:
< www.youtube.com/playlist?list=PLEqi0nIP3VnpEKGTjMRcH001akEaIXTfi >

1 rANGEL "sweatingHeart" Alexandre Rangel 10:14
2 rANGEL "Tasting you: illumination" Alexandre Rangel 5:35
3 rANGEL "bataille" Alexandre Rangel 4:25
4 rANGEL "animalElectricity" Alexandre Rangel 6:05
5 rANGEL "theDawn" Alexandre Rangel 7:11
6 rANGEL "upsideDown" Alexandre Rangel 4:12
7 rANGEL "36hoursDay" Alexandre Rangel 5:46
8 rEVOLUTIONAIF + rANGEL "spiderBlues" Alexandre Rangel 3:10
9 rANGEL "suuba" Alexandre Rangel 4:29
10 rANGEL "OuterSpaceCarnival" Alexandre Rangel 4:36
11 rANGEL "mu-dança" Alexandre Rangel 6:47
12 rANGEL "primeiraMissa" Alexandre Rangel 4:13
13 rEVOLUTIONAIF + rANGEL "jogos - que ñ preciso ganhar" Alexandre Rangel 4:56
14 rANGEL "iemanjah" Alexandre Rangel 4:41
15 rANGEL "slowDrive inside myBrain" Alexandre Rangel 2:54
16 rANGEL + Fernanda Jacob "semba (sambar é rezar)" Alexandre Rangel 4:31
17 rANGEL "Santo Guerreiro" Alexandre Rangel 6:11
18 rANGEL "Toto Tzara" Alexandre Rangel 4:05
19 rANGEL "curtindo" Alexandre Rangel 6:38
20 rANGEL "almafuerte" Alexandre Rangel 4:03
21 rANGEL + Naura Timm "tarot" Alexandre Rangel 4:09
22 rANGEL + Madame Quin Quin "meditation: pink quartz" Alexandre Rangel 5:21
23 rANGEL "one infinite drive" Alexandre Rangel 4:18
24 rANGEL "interConnected" Alexandre Rangel 2:02
25 rANGEL "sãoJorge" Alexandre Rangel 8:16
26 rANGEL "countryDrive" Alexandre Rangel 3:31
27 rANGEL "Saci" Alexandre Rangel 6:49
28 rANGEL "break" Alexandre Rangel 4:27
29 rANGEL "1x7" Alexandre Rangel 3:12
30 rANGEL "visão" Alexandre Rangel 4:38
31 rANGEL "enta" Alexandre Rangel 12:31
32 rANGEL "how to levitate" Alexandre Rangel 5:13
33 rANGEL "staying, leaving, shaking" Alexandre Rangel 7:03
34 rANGEL "unlocking" Alexandre Rangel 4:53
35 rANGEL "machine for trance'n'dance" Alexandre Rangel 7:34
36 rANGEL "macro sky" Alexandre Rangel 5:38
37 rANGEL "motherLode" Alexandre Rangel 4:34
38 rANGEL "cavalo branco" Alexandre Rangel 8:15
39 rANGEL "um canto" Alexandre Rangel 6:19
40 rANGEL "janaína" Alexandre Rangel 6:21
41 rANGEL "materialize" Alexandre Rangel 6:22
42 rANGEL "noise to music ratio" Alexandre Rangel 5:03
43 rANGEL "welcome to the sky" Alexandre Rangel 8:43
44 rANGEL "dama da noite" Alexandre Rangel 11:07
45 rANGEL "I brought my body and soul here" Alexandre Rangel 10:30
46 rANGEL "therapy" Alexandre Rangel 4:21
47 rANGEL "salamandra" Alexandre Rangel 3:23
48 rANGEL "green and blue horizon" Alexandre Rangel 5:07
49 rANGEL "W3 flow" Alexandre Rangel 27:53
50 rANGEL "following" Alexandre Rangel 4:07
51 rANGEL "brasília paris" Alexandre Rangel 10:16
52 rANGEL "current skin / future of the skin" Alexandre Rangel 4:18

Imagem 26. Lista de obras do projeto Weekly Beats 2014.

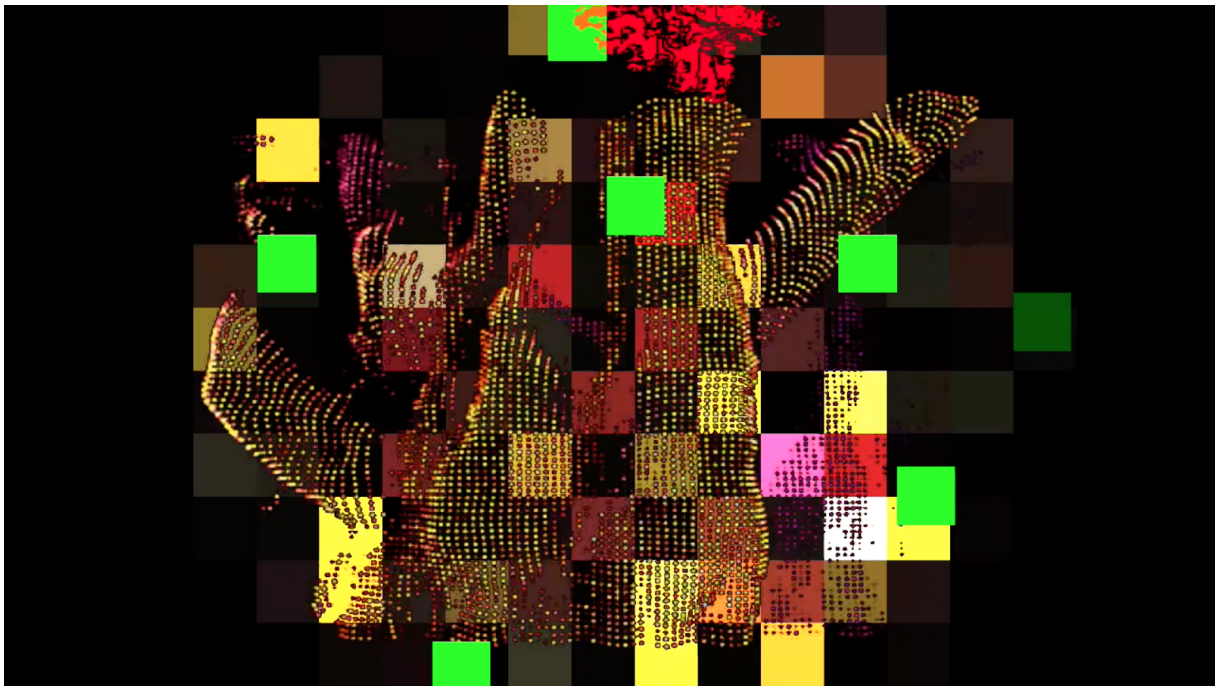


Imagem 27. Frame da obra Sweating Heart, com nuvem de pontos de profundidade coloridos.

Na imagem anterior, vemos uma das obras criadas nesta série – a mais significativa, a meu ver, intitulada *Sweating Heart (Coração Suando),* criada com equipamento de gravação tridimensional. A captação das imagens foi feita por meio de um sensor *Kinect 1*, registrando nuvens de pontos de profundidade (*point cloud*, na terminologia de criação 3D). Para fazer a captação das informações de profundidade, embarcamos em um carro um computador e um sensor *Kinect*, conectado via cabo USB, gravando sequências de imagens em tons de cinza. Cada tom de cinza entre preto e branco identifica a distância entre os pontos da imagem e a câmera do sensor. Para funcionar com o carro em movimento, sem uma tomada elétrica, o sensor teve o sistema de alimentação adaptado para funcionar com conector de eletricidade de interior automotivo (antigo aquecedor de acendedor de cigarro).

A parte sonora da obra foi toda composta com gravações de campo que registravam cantos de cigarras, muito características da paisagem sonora de Brasília. Tudo o que ouvimos na composição, soando como um baixo ou como elementos percussivos, são sons derivados das gravações de cigarras. Para a composição foi utilizado o *software reNoise*[42]. O *reNoise* é um programa de produção musical baseado nos ambientes de produção musicais digitais dos anos 1990, os *trackers,* que funcionavam normalmente em computadores das linhas *Commodore Amiga* e *Atari ST*. Uma das características específicas dos *trackers* é ter linhas do tempo que viajam no sentido vertical, diferentemente do

[42] *Site* do software de produção musical *reNoise:* < www.renoise.com >

sentido horizontal que quase a totalidade dos softwares de edição e criação audiovisual adotam hoje em dia, o que prova que características das interfaces de operação gráfica são só convenções/tendências.

Sistemas de regras também são possíveis de serem criados e executados em softwares de edição de vídeo, como o popular Adobe Premiere e o Blender (o programa de criação 3D Blender também edita vídeos). A construção visual das obras do desafio criativo foi realizada quase que totalmente com o software de edição Apple Final Cut Pro X [43].

É possível criar um "sistema lógico" dentro de um software de edição de vídeo. Um dos pontos-chave, que opera como uma bifurcação do fluxo lógico, é a função de recorte (ou key), que pode excluir ou incluir partes da imagem, baseando-se em cores ou no brilho de cada ponto (color key e luminance key). Os vídeos dessa série são, na sua grande maioria, compostos de planos contínuos – sem cortes – e em câmera lenta. A camada de inserção poética encontrou-se na lógica de colorização e composição de efeitos e camadas. Podemos ver a seguir um esquema do pensamento de construção como sistema/fluxograma e a imagem resultante:

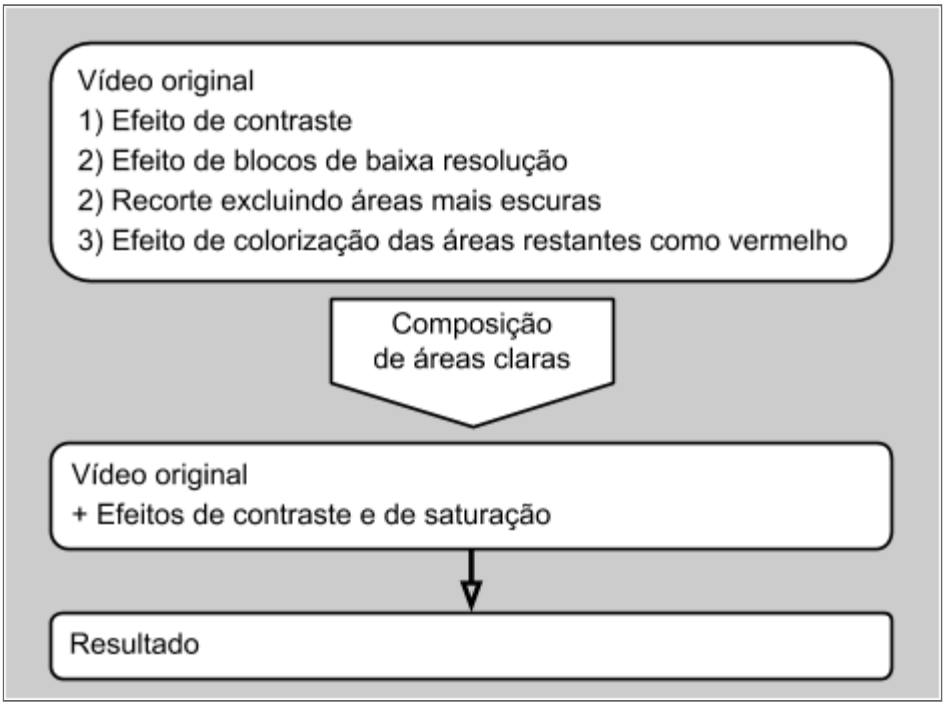


Gráfico 3. Esquema de construção com fluxo lógico aplicado em sistema de edição de vídeo.

[43] Página do *software* de edição de vídeo *Apple Final Cut Pro X:*
< www.apple.com/final-cut-pro >

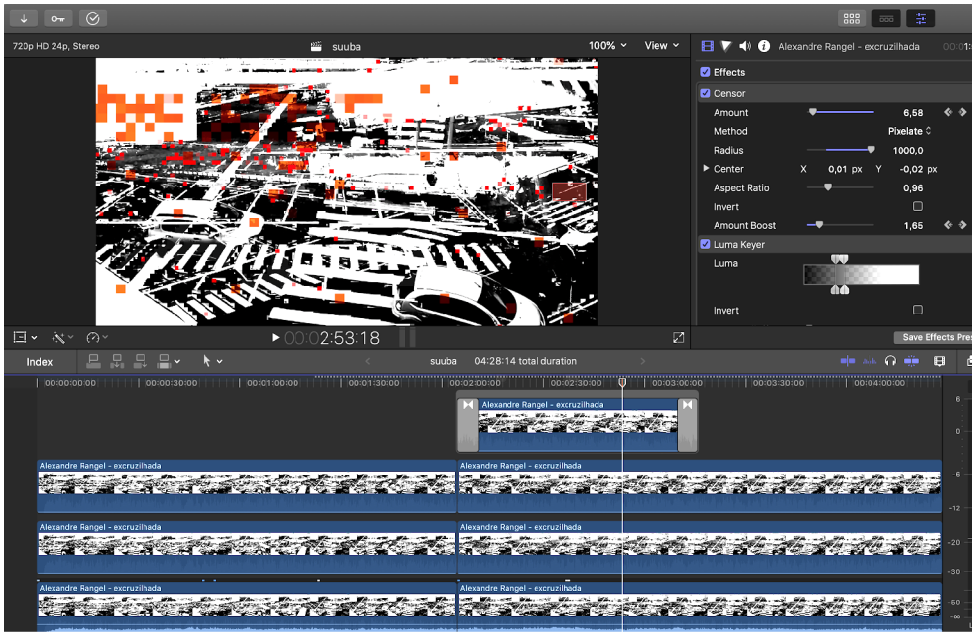


Imagem 28. Configuração da linha do tempo (timeline) da obra "Suuba".

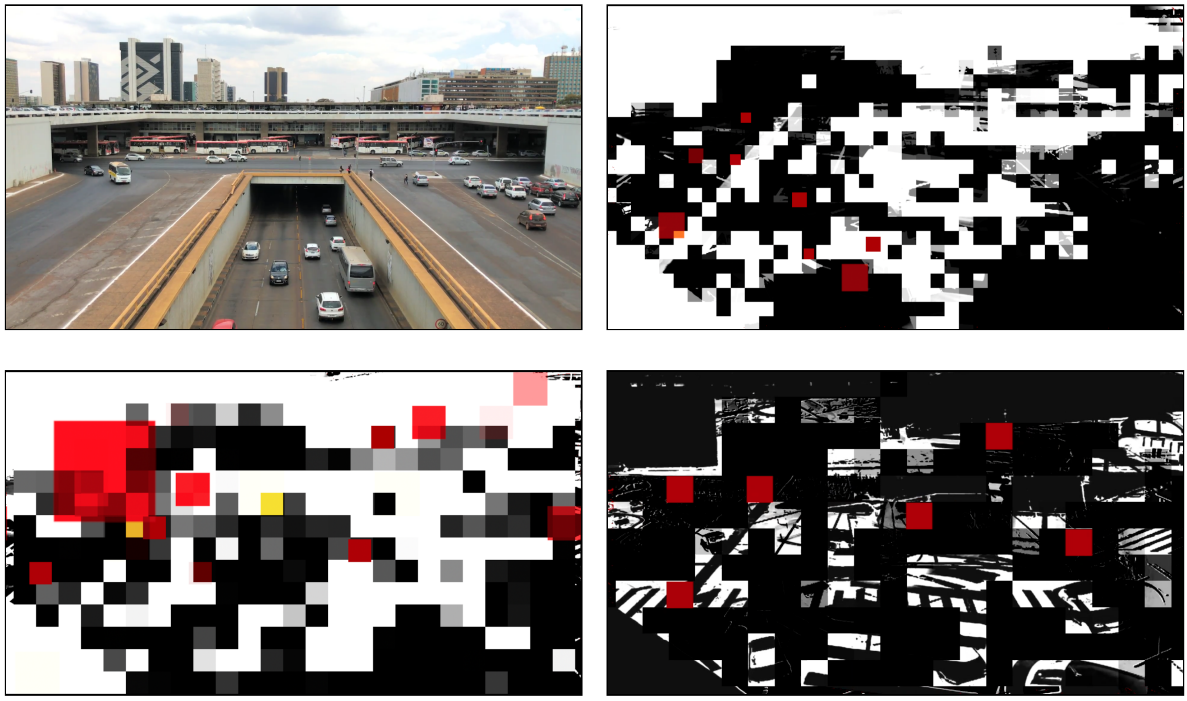


Imagem 29. Etapas de composição da obra Suuba, parte do Weekly Beats 2014.

2.3.4 Projeto *Weekly Beats 2016*

Em 2016, já com esta pesquisa de doutorado em andamento, me propus a criar 52 músicas (uma por semana), usando o ambiente de programação musical do *Sonic Pi*. No caso das 52 obras com programação musical, as práticas de reuso e *remix* foram incentivadas, uma vez que o código-fonte completo de todas as composições foi

disponibilizado na internet em formato executável e – o mais importante – modificável pelo "usuário final". Até mesmo o conceito de público é questionado, uma vez que, com pouco esforço, podem-se gerar obras derivadas, em diálogos com as produções originais.

Criei as seguintes regras para o processo, como estímulo criativo:

1) Utilizar somente o *Sonic Pi* para a criação das composições.

2) Utilizar dois sample sonoros colhidos aleatoriamente do banco de sons online Freesound. O *site* tem uma funcionalidade de escolha de sons inesperados, por meio da funcionalidade "*Give me a random sound"* (Me dê um som randômico).

Can Ince destaca o processo de auto-regragem nos processos artísticos em sua dissertação de mestrado em música: "Artistas impondo limites arbitrários a si próprios para encorajar a criatividade é uma prática comum em vários campos criativos. Talvez exista aqui um ponto de convergência, onde o caminho mais produtivo para essa forma de criatividade seja os artistas se empoderando com o conhecimento e as ferramentas para imaginar e criar seus próprios sistemas fechados para experimentação musical e artística." (INCE, p.37)

O código-fonte comentado das 52 composições realizadas com o sistema *Sonic Pi* constitui o Apêndice I desta tese.

2.3.5 *Sábio ao Contrário*

"Alexandre rANGEL encarna a persona do *Sábio ao Contrário* em uma performance-ritual de *remix* de filosofia e religião do Ocidente e do Oriente e música eletrônica experimental – uma leitura de tradução livre da obra de filosofia chinesa Tao Te Ching e animações de ilustrações religiosas do século XVII." (*release* da performance *Sábio ao Contrário*)

As ilustrações usadas nesse *remix* audiovisual fazem parte de iniciativa de uso livre do banco de imagens do Museu Nacional da Holanda (*Rijksmuseum*[44]). O extenso arquivo de imagens pode ser consultado por meio de palavras-chave. Para essa obra, busquei ilustrações em nanquim (e técnicas similares, com acabamento em preto e branco) com os temas religiosos de anjos e seus antagonistas. As imagens monocromáticas funcionam melhor para projeção de vídeo com colorização posterior, ao vivo.

---

[44] *Site* com banco de dados aberto de imagens do *Rijksmuseum:* < www.rijksmuseum.nl/en/rijksstudio >

Em uma obra como essa, onde é claro o momento de estreia, de dia de performance, fica muito evidente para mim a diferença entre os momentos de programar e performar: da solidão à tensão eufórica da apresentação em público.

Realizei a performance de abertura do *Taichung Soft Power Forum (Fórum Taichung de Poder Cultural)*, na cidade de Taichung, em Taiwan, no ano de 2015. Publiquei uma pequena tiragem de 300 livretos com poesia, ilustrações e código da trilha sonora da performance de abertura do fórum. O código-fonte do livreto é no sistema Sonic Pi, e pode ser reproduzido facilmente: basta que se copiem e colem os trechos de programação para o ambiente do *Sonic Pi*.

A performance também foi realizada duas vezes no *Festival SESC de Inverno*, em Petrópolis e Teresópolis, no estado do Rio de Janeiro, e duas vezes em Brasília, no Museu Nacional da República e no centro da cidade (Setor Comercial Sul), para a gravação de trecho do documentário *Intervenções Urbanas,* da diretora Lorena Figueiredo.

Imagem 30. Performance Sábio ao Contrário no Museu Nacional da República.

Trecho da performance em Taiwan e oficina:

< www.youtube.com/watch?v=t2gcFI4fe40 >

Performance no Museu Nacional da República, em Brasília:

< www.youtube.com/watch?v=bpEOgs9EplA >

2.3.6 *Brasília 1960-2160*

A performance audiovisual *Brasília 1960-2160* [45] apresenta uma visão de futuro com estética dos anos 1950/1960, quando Brasília foi pensada, criada e inaugurada. A performance mistura imagens da construção de Brasília com figuras antigas de discos voadores, e possui trilha sonora criada com o *Sonic Pi* e visuais criados com o *software VDMX*.

Visualmente, a obra tira do lugar comum as imagens de *drones*, fazendo composições de imagens aéreas de Brasília com imagens de arquivo da construção da Capital e desenhos de naves futuristas e discos voadores extraterrestres.

Apresentado como projeto para o aniversário de Brasília, a obra foi contemplada em edital da Secretaria de Cultura do Distrito Federal para realização de performance audiovisual nas comemorações do ano de 2017.

Imagem 31. Performance Brasília 1960-2160 no Museu Nacional da República.

[45] Registro em vídeo de trechos da performance *Brasília 1960-2160:*
< www.youtube.com/watch?v=YirEN_mUOjc >

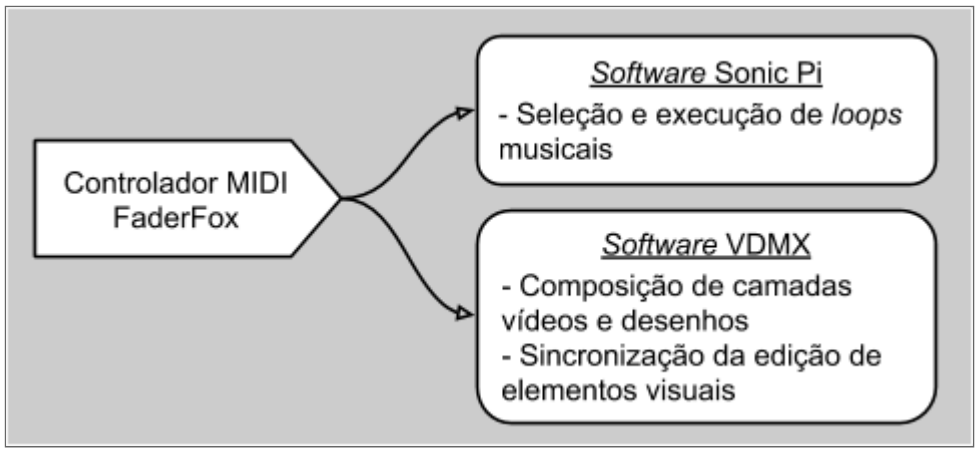


Gráfico 4. Esquema técnico da performance Brasília 1960-2160.

2.3.7 Emoção Artificial

Emoção Artificial é um sistema computacional[46] que trata da dualidade entre a criação de significado objetivo e subjetivo (com visualização e sonificação) e a desconstrução do significado por meio da superexposição. Propõe uma interpretação do conteúdo sonoro desvinculada do significado textual e uma interpretação geométrica e matemática das letras, dos espaços e da pontuação gramatical. Cria um diálogo com a poesia concreta por meio da objetificação e descontextualização de frases das manchetes jornalísticas. O jogo das palavras com os sons imprevisíveis chama a atenção para o papel da arte de levantar questões ao invés de dar respostas prontas; de escolher questões ao invés de respondê-las. O nome da obra é um trocadilho com a expressão "inteligência artificial", quase que um cálice sagrado da pesquisa em computação na década atual.

Duas características marcantes do sistema foram a surpresa em relação algumas matérias jornalísticas encontradas pelo programa e a presença de uma espécie de sarcasmo ou até mesmo ironia por parte do *software* – se é que o programa consegue expressar personalidade. Sem se preocupar com a sobreposição dos temas musicais ao o conteúdo das matérias, muitas vezes o sistema apresentava notícias de conteúdo extremamente trágico acompanhadas de temas sonoros alegres e festivos. Essas edições traziam certamente uma sensação de estranhamento quando assistidas pelo público.

[46] Página da obra, com registro de exposição e cerca de dez horas de trechos de de notícias lidas e sonorizadas pelo sistema: < www.alexandrerangel.art.br/emocaoartificial.html >

Imagem 32. Obra Emoção Artificial na exposição A/RISCADO: Arte, Ciência e Tecnologia.

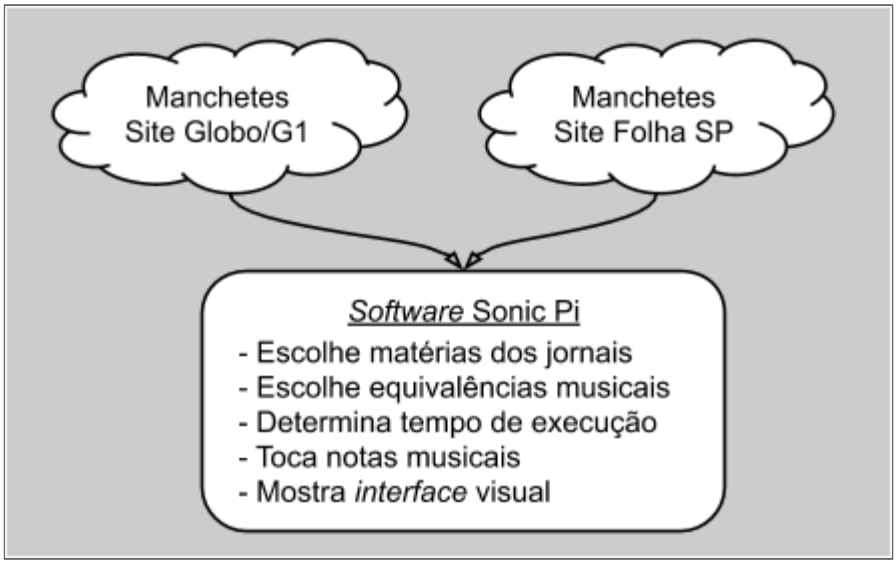


Gráfico 5. Esquema técnico da obra Emoção Artificial.

O processo que o *software* realiza para a criação audiovisual é:

1) Escolha e leitura de fluxos de informação do tipo RSS[47] a partir de dezenas de *links* de seções de jornais *online*;

2) Escolha do mapeamento entre letras e eventos sonoros (notas musicais ou *samples*, tais como sons de peças percussivas) de acordo com a Figura 38;

3) Escolha aleatória de sintetizador de som entre os 42 instrumentos padrão do Sonic Pi;

4) Determinação de velocidade da performance sonora (BPM, Batidas Por Minuto) a partir do tamanho da manchete escolhida (notícias com mais texto são executadas mais rapidamente do que notícias curtas);

5) Execução sonora, acompanhada de detalhamento de interpretação da manchete escolhida.

Nessa obra, as matérias primas são o tempo e o espaço, relativizados pelo acaso inserido pela programação. A regra é bagunçada pelo acaso.

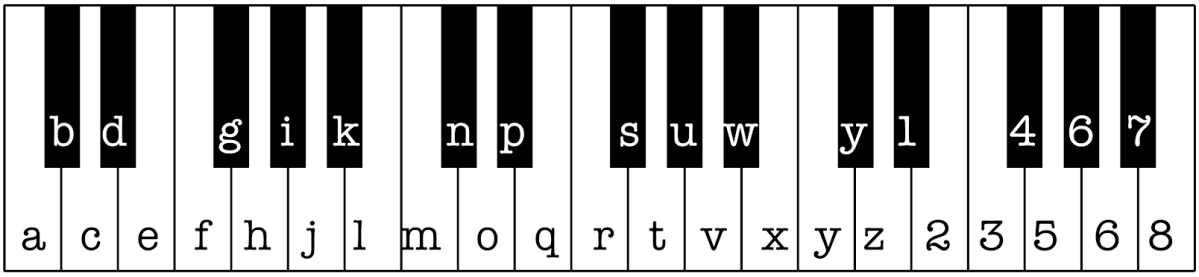


Imagem 33. Mapeamento de letras em notas musicais na obra Emoção Artificial.

A inovação tecnológica realizada neste projeto foi a capacitação do sistema *Sonic Pi* para ler manchetes de jornais a partir da internet. Descobri a possibilidade técnica de expandir as funções do sistema *Sonic Pi* por meio de extensões da sua linguagem padrão, o Ruby. Com uma "gema" *Ruby* (as extensões do *Ruby* são chamadas, de *GEMS*) adicionada a seu sistema, o *Sonic Pi* ganhou o poder de ler arquivos de texto a partir da internet. Os arquivos que considerei mais estáveis e interessantes, do ponto de vista de fluxo de informações, foram os resumos de notícias no formato RSS dos jornais online Folha de São Paulo e Globo/G1. Os arquivos dinâmicos RSS são mais adequados para captura de texto do que uma página normal de internet (HTML), pois não contém

[47] Um fluxo de notícias RSS é um endereço fixo, dentro de um *site*, onde notícias, ou outro fluxo de dados, são atualizadas constantemente. Exemplo de um endereço com fluxo de texto RSS: < http://feeds.folha.uol.com.br/emcimadahora/rss091.xml >

informações de diagramação ou imagens, facilitando a leitura por meio de software automatizado.

2.3.8 Performance telemática TOPLAP 15

"TOPLAP 15 - Slicing sonic thoughts on a Turing machine"[48] (Fatiando pensamentos sonoros através de uma máquina de Turing) foi uma performance que realizei em 2019 como parte da transmissão ao vivo de comemoração dos 15 anos da TOPLAP, uma organização internacional dedicada ao *live coding*. É interessante acompanhar um agrupamento tão diverso, com dezenas de performances usando – abertamente – a mesma técnica: a programação ao vivo de diferentes programas. Krzysztof Szlifirski, diretor do Estúdio de Música Experimental da Rádio Polonesa, indagou, em 1970: "A aplicação de novas tecnologias influencia e condiciona o valor e a originalidade de uma obra de arte?"

Para a parte visual da performance foi criada uma animação de elementos gráficos com o sistema *Blender* complementado pelo programa de modelagem e animação paramétrica *Sverchok*[49]. Essa animação, uma vez finalizada em formato MP4, foi importada para ser visualizada pelo *software* de edição ao vivo *VDMX*. Em uma camada visual na frente do grafismo, pode-se ver a interface textual de programação ao vivo do *Sonic Pi*. Um recurso do *software Sonic Pi*, acessível pelo painel de preferências do programa, permite uma configuração de transparência, desvendando quaisquer outras imagens que estejam na tela do computador. Essa obra serviu como desenvolvimento técnico para o que veio a tornar-se a obra *Esculturas Quânticas*, explorando as possibilidades técnicas da ligação dinâmica entre os programas *Blender* e *Sonic Pi*.

---

[48] Vídeo da performance *TOPLAP 15 - Slicing sonic thoughts on a Turing machine:* < www.youtube.com/watch?v=_r0PLWJyRDk >

[49] *Site* do sistema *Sverchok*, *software* complementar ao *Blender:* < http://nikitron.cc.ua/sverchok_en.html >

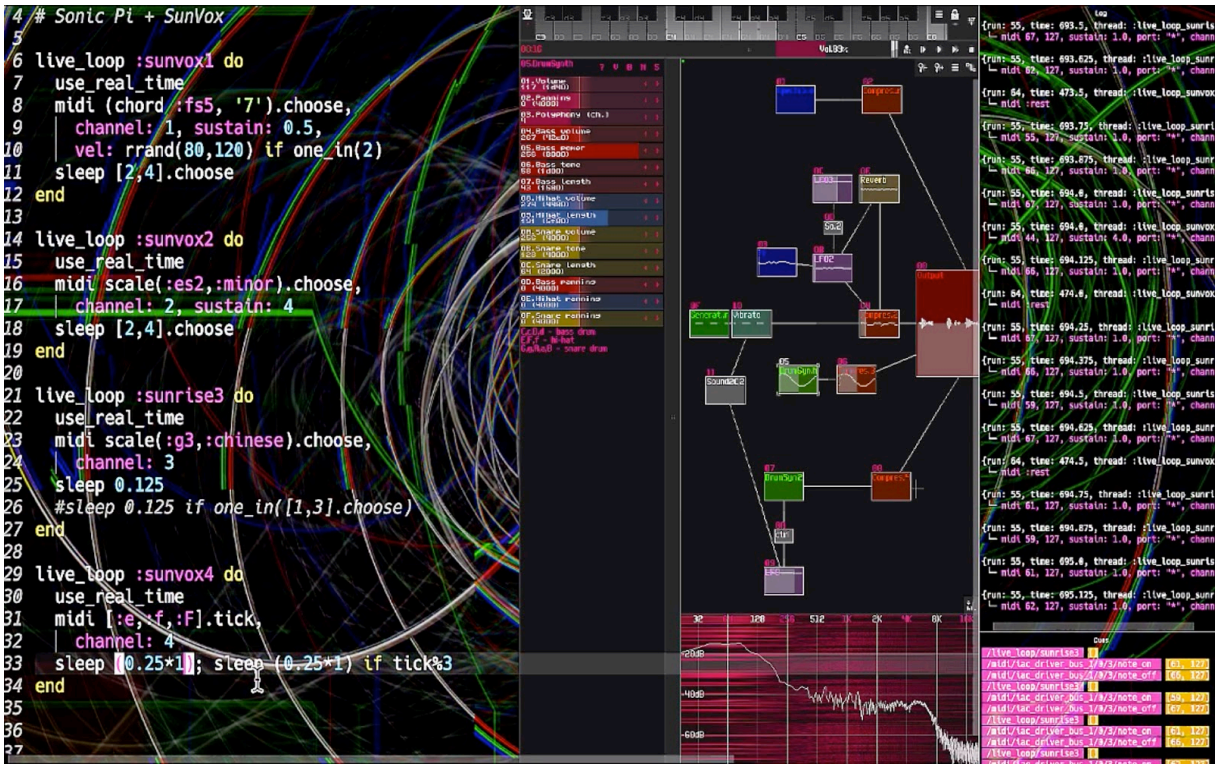


Imagem 34. Tela da performance, mostrando simultaneamente Sonic Pi, SunVox e VDMX.

A composição musical foi criada com o *Sonic Pi* juntamente com o *software* livre de produção musical *SunVox*[50], ambos rodando em sistema *macOS* (a mesma combinação sonora também funcionaria em sistemas *Linux* ou *Windows*). A programação, ao ser criada ao vivo no *Sonic Pi*, gerava notas musicais no padrão MIDI, direcionadas ao *SunVox*. O *Sonic Pi* não sintetizava nenhum som, permanecendo sempre em silêncio e enviando notas (comandos) MIDI. Cada *loop* (trecho de programação musical em repetição) do *Sonic Pi* foi direcionado para canais diferentes do *SunVox*: dois sintetizadores e duas baterias eletrônicas. Vale ser notado que o *SunVox* é um programa extremamente leve, o que não traz prejuízo de velocidade de processamento para o computador executando o *Sonic Pi*.

O SunVox é a simplificação máxima do sistema de programação por fluxograma. Sua simplicidade tecnológica e de interface permite que se brinque com ele, inclusive em celulares e equipamentos obsoletos (em outras circunstâncias de processamento, com *softwares* mais tradicionais). A flexibilidade e a leveza de manuseio do SunVox vêm do fato do seu desenvolvedor, o russo Alexander Zolotov, ter desenvolvido todo o programa, incluindo a *interface* gráfica de usuário (GUI – *Graphical User Interface*),

[50] *Site* do sistema SunVox: < www.warmplace.ru/soft/sunvox >

sem o uso de bibliotecas de programação – tornando o *software* independente de outros desenvolvedores, podendo funcionar facilmente em vários sistemas operacionais.

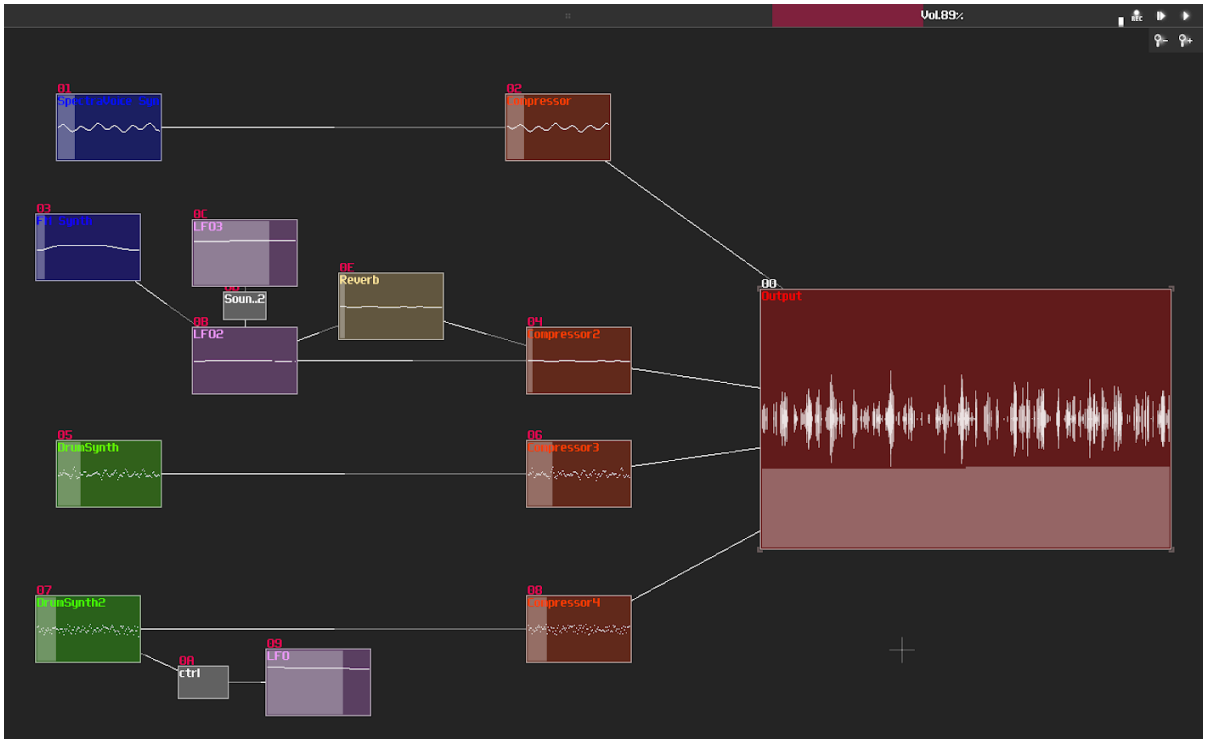


Imagem 35. Software SunVox recebendo notas MIDI em quatro canais diferentes.

O sistema *SunVox* permite também a reprodução dinâmica de música generativa *online* para execução em computador de mesa, *tablet*, *smartphone* ou em navegador de internet[51].

Ao comentar sobre o resultado dessa performance, o desenvolvedor americano David Lublin, criador do *software VDMX*, relatou-me, em e-mail pessoal, planos de criação de um sistema de *live coding* interno para o seu sistema de *remix* audiovisual. O sistema permitiria controlar os parâmetros visuais disponíveis na *interface* gráfica do VDMX. A programação poderia também o acesso a *sites* de bancos de imagens na internet, tais como o repositório de animações GIF Giphy[52]. Sugeri ao desenvolvedor a possibilidade de pesquisa de sons do banco do sons *Freesound*, para uso no ambiente do sistema *VDMX*, que já tem possibilidades avançadas de manipulação de imagem e som ao vivo.

---

[51] Biblioteca *SunVox* para execução musical dinâmica *online:* < http://warmplace.ru/soft/sunvox/jsplay >

[52] Repositório *online* de animações no padrão GIF: < https://giphy.com >

2.3.9 Performances telemáticas Algorave

O termo *Algorave* foi criado a partir da fusão de duas palavras em inglês: *algorithm* (algoritmo – equação ou função matemática, trecho de um programa de computador) e *rave* (festa de música eletrônica). Ou seja, uma festa de programação.

• Performance *Algorave 5th Birthday*. Duração: 29 minutos; data: 18-março-2017:
< www.youtube.com/watch?v=Z6-eC1bEWzk >
• Performance *Algorave 6th Birthday*. Duração: 29 minutos; data: 18-março-2018:
< www.youtube.com/watch?v=updb3O6zOIY >

2.4 *ORCA*

O sistema *ORCA* [53] é uma das maiores novidades na cena de sistema de programação ao vivo dos últimos anos. É desenvolvido pelo casal Rekka e Devine Lu Linvega – enquanto navegam em um veleiro pelo planeta! O *software* é uma mistura de sistema de programação e tabuleiro de jogo, com elementos sonoros. Sua natureza gráfica bidimensional permite a criação e visualização de vários pontos de geração de sons (que podem ser referidos com pulsos, cursores, ou cabeças de *playback*).

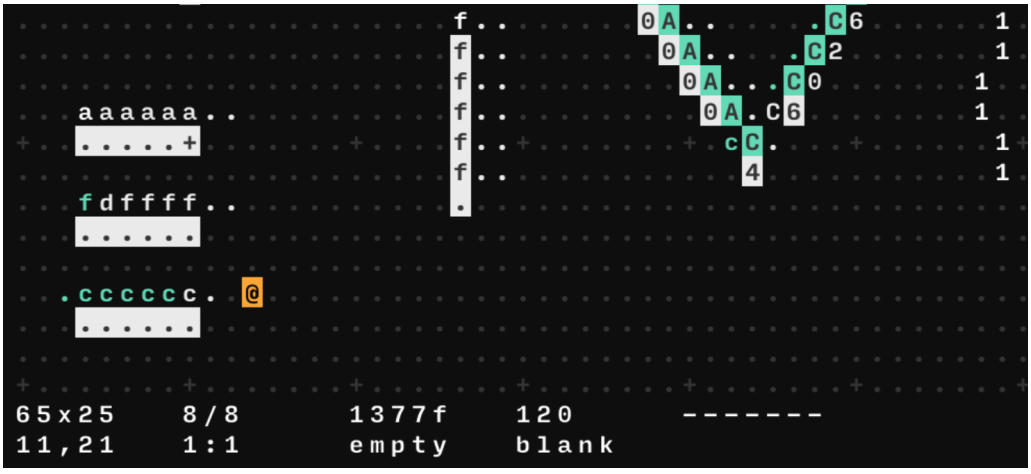


Imagem 36. Interface do sistema de programação ORCA.

O *software ORCA* não produz sons sozinho, mas pode enviar comandos (nos formatos MIDI, OSC ou UDP) para síntese sonora em outros *softwares*, tais como o *Sonic Pi* ou o *Ableton Live*.

[53] Site do sistema *ORCA*: < https://wiki.xxiivv.com/#orca >

Observo o campo de experimentação do sistema *ORCA* como uma ligação com as produções e experimentos com partituras visuais, permitindo uma espécie de desenho com o código e interpretações sonoras e visuais ao mesmo tempo. Sistemas de live coding como o *ORCA* são a concretização de prática que transformou profundamente, durante aproximadamente os últimos 100 anos, o conceito e o papel da partitura musical. Tornaram real a utopia de fusão entre composição, execução e improvisação audiovisual.

No sistema *ORCA*, cada letra maiúscula é uma função de programação. Como exemplos, a letra A adiciona, a letra **C** gera uma sequência de números, a letra **R** gera números randômicos, etc. Como são feitas operações aritméticas de programação, onde cada função é expressada (e resolvida) em apenas uma letra, como é o caso do *ORCA* ? Vejamos os seguintes quatro exemplos, ilustrados na figura seguinte:

Imagem 37. Exemplos de funções básicas do sistema ORCA.

1) Na primeira coluna da imagem, vemos o uso da função **A**, que adiciona os dois números que estão aos seus lados e apresenta o resultado embaixo;

2) Na segunda coluna, a função **C** realiza um contador, iniciando no número da esquerda e terminando no número da direita, apresentando o valor atual na linha de baixo;

3) Na terceira coluna, a função **R** gera números aleatórios contidos entre os números que estão aos seus lados. Aqui vemos uma novidade: gerar números entre 0 e Z?! Sim. Como o *ORCA* precisa expressar suas funções e resultados com apenas um caractere, foi adotado um sistema de contagem com base 36. Ou seja, podem-se expressar 36 números com os algarismos de **0** a **9** e as letras de **A** a **Z**;

4) Na quarta coluna, vemos a utilização da função **R,** em cima de um comando de nota MIDI, para modificar a nota sendo escolhida aleatoriamente. O símbolo de dois pontos (**:**) pode ser acompanhado de 5 parâmetros, de acordo com os caracteres à sua direita: 1) o canal MIDI, especificando o instrumento sendo tocado; 2) a oitava da nota; 3) a nota; 4) a velocidade (intensidade) da nota e, finalmente, 5) a duração da nota. Os parâmetros de intensidade e duração são opcionais.

0	1	2	3	4	5	6	7	8	9	A	B
0	1	2	3	4	5	6	7	8	9	10	11
C	D	E	F	G	H	I	J	K	L	M	N
12	13	14	15	16	17	18	19	20	21	22	23
O	P	Q	R	S	T	U	V	W	X	Y	Z
24	25	26	27	28	29	30	31	32	33	34	35

Tabela 12. Tabela de contagem com numeração base 36.

2.4.1 *Tocando Brasília*

Um jogo? Um instrumento? Ou uma cidade – inteligente? A obra *Tocando Brasília* explora a relação entre os papéis de jogador e cidadão, propondo momentos de sensibilização e atuação política, em dias de vigilância, de controle de dados, e da internet das coisas (*IoT - internet of Things*). Quem manipula a quem: a cidade ao ser humano ou o contrário? O *software* interativo mistura imagens de satélite e textos do urbanista Lúcio Costa, permitindo que o público manipule os mapas e toque a cidade como uma partitura.

Software Art ? Jogo Contemplativo? Talvez os dois ao mesmo tempo; além de fazer diálogos com a Videoarte e a Arte Sonora. Um dos pontos mais emblemáticos dessa obra é justamente o fato da sua natureza ser baseada em suportes híbridos. A pesquisadora Donna Cox, pioneira nos campos de arte computacional e visualização científica, afirma que a atividade multidisciplinar tornou-se um fator chave para a era pós-moderna e que proíbe a classificação das obras meramente pelo meio (COX, 1989, p.9). Traço também uma relação com o conceito de invenção do artista Hélio Oiticica, que evita a classificação e definição de formato e suporte da obra durante a sua concepção.

Procurei construir metáforas sonoras da cidade sem cair no lugar-comum do arquétipo urbano moderno, como o som de automóveis, pássaros ou pessoas, assim descrevendo a paisagem sonora de forma afetiva e menos como registro acústico realista.

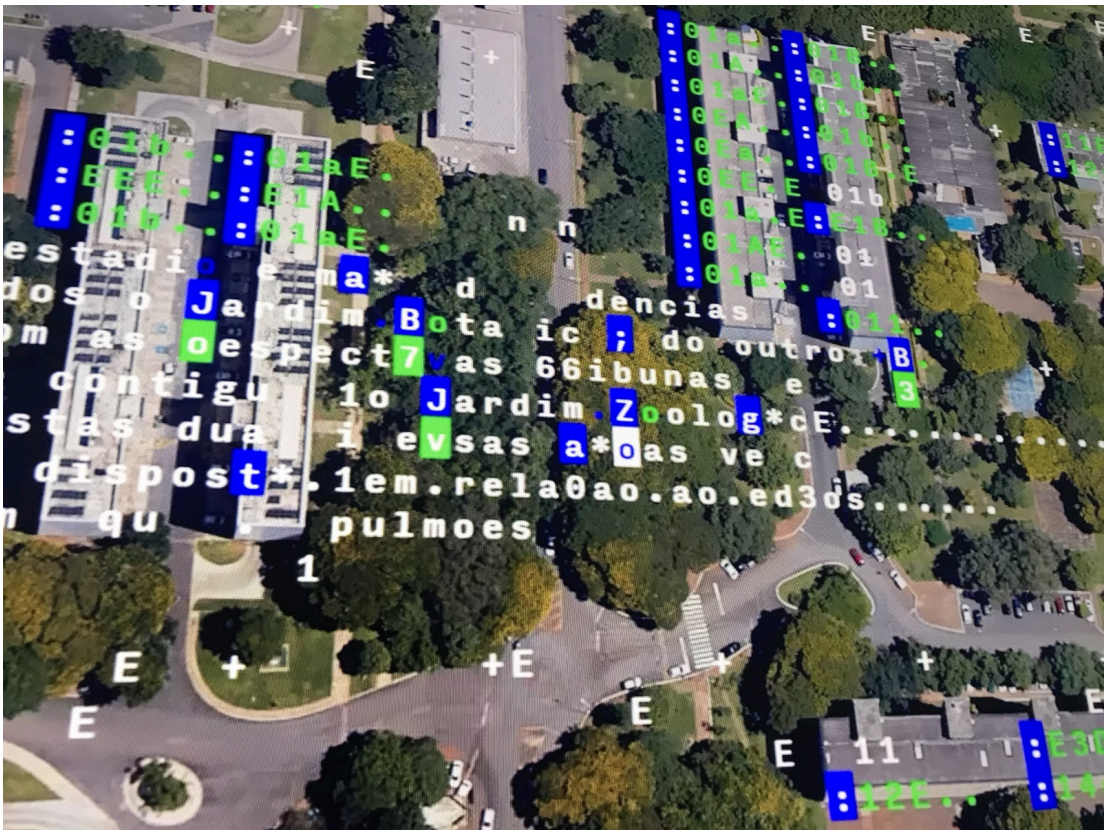


Imagem 38. Detalhe da obra Tocando Brasília.

Eu já alimentava a ideia de tocar Brasília há alguns anos, mas nunca havia concebido uma forma técnica que pudesse viabilizar a execução técnica do projeto. O "momento de Eureka" que agora seria possível veio quando eu me familiarizei com o sistema de produção musical *ORCA*. Com ele, pude:

1) Utilizar imagens de fundo (mapas) no ambiente de composição e exibição;

2) "Desenhar" os elementos musicais em cima das imagens de fundo, colocando manualmente as notas e comando em cima dos elementos urbanísticos de Brasília, tais como blocos residenciais e avenidas;

3) Receber informações da interação dos usuários por meio de um controle de *videogame* (o *gamepad* do sistema Super Nintendo). Usando uma programação feita no sistema *Processing*, pude orquestrar toda a experiência do público com a obra, tal como a interpretação da interatividade com o controle e a troca periódica dos mapas e elementos de criação sonora.

Para a escolha das notas musicais de cada momento da obra, criei uma correspondência com letras dos blocos de cada quadra de Brasília (A, B, C... acionando as notas Lá, Sí, Dó...).

Imagem 39. Controle tipo gamepad Super Nintendo.

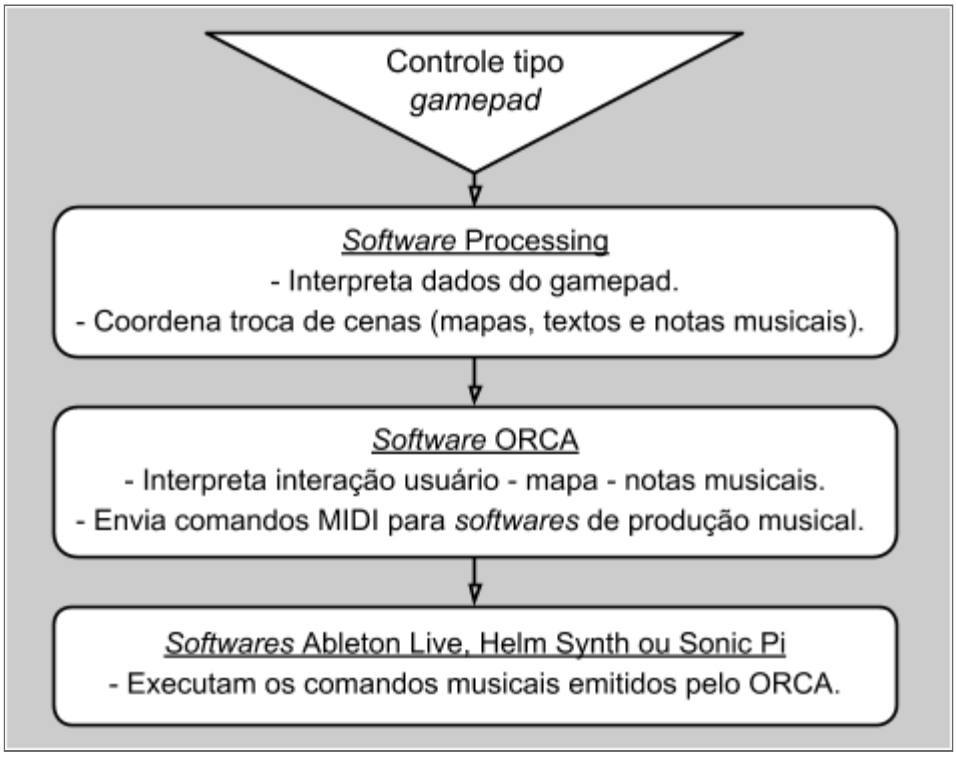


Gráfico 6. Esquema técnico da obra Tocando Brasília na versão construída no sistema ORCA.

A obra foi criada conceitualmente durante disciplina do doutorado em artes *Métodos de Deslocamento e Deriva*, ministrada pela professora Karina Dias, na Universidade de Brasília, no primeiro semestre de 2018. Um projeto relevante para o desenvolvimento do obra *Tocando Brasília*, realizado por mim em colaboração com o desenvolvedor Arthur Cordeiro, em 2003. O sistema interativo foi apresentado na *Conferência Internacional Blender 2004*, em Amsterdã. Para a criação sonora foi usado o *software* comercial alemão de criação e performance sonora *Ableton Live*[54]. Também foram feitos experimentos

[54] *Site* da empresa alemã Ableton, desenvolvedora do software de produção musical Ableton Live: < www.ableton.com >

bem-sucedidos de sonorização a partir dos programas *Helm Synth* [55] e *Sonic Pi*. As vantagens do *Ableton Live* foram a facilidade de programação de canais *MIDI* específicos para eventos diferentes e a riqueza de timbres disponíveis no programa.

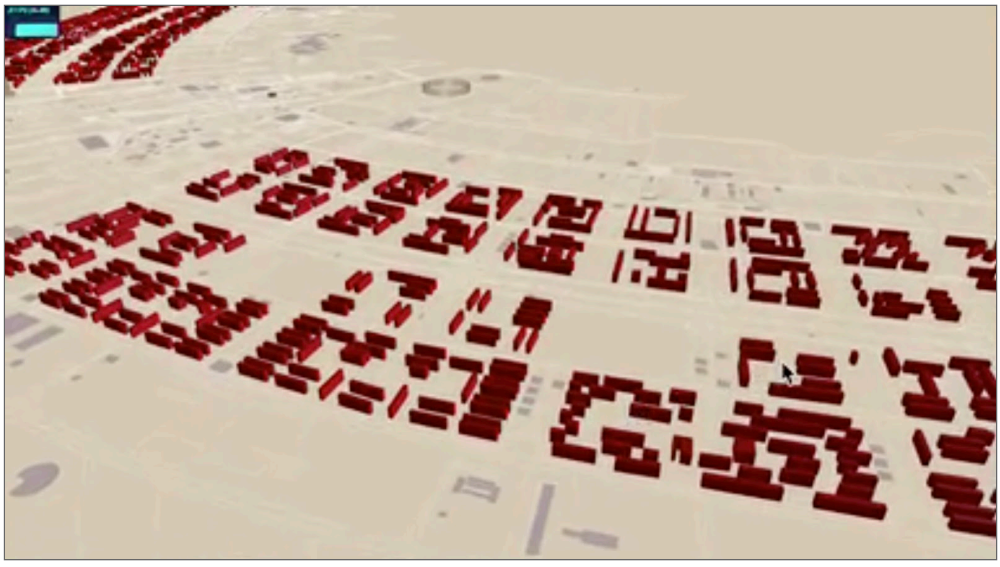


Imagem 40. Frame da obra Tocando Brasília em versão no sistema de programação Three.js.

Sobre o comportamento autônomo do agente de criação artística, Harold Cohen (1928-2016), um artista pioneiro da arte digital, constata no documentário *A Era da Máquina Inteligente:* "Eu surpreendo quando digo que um grupo de cerca de mil regras desenvolvidas pode criar uma obra de artes plásticas. Eu mesmo me pego olhando para os quadros criados e digo – Sério!? – Sim, um computador fez isso comigo aqui, são, ao seu lado; mas um dia, quando eu não estiver mais aqui, o computador poderá continuar criando obras inéditas!" (KURZWEIL). Cohen escreveu, nos anos 1970, o programa autônomo de pintura AARON[56], partindo da seguinte questão conceitual: "Quais as condições mínimas sob as quais um grupo de marcas funciona como uma imagem?"

[55] O *Helm* é um sintetizador sonoro que pode ser controlado pelo protocolo MIDI ou integrado programaticamente a projetos dentro do ambiente de desenvolvimento *Unity*. Encontra-se disponível para *download* gratuito em: < https://tytel.org/helm/ >

[56] Registros da produção imagética do sistema *AARON:* < http://aaronshome.com >

2.5 Outras ferramentas de programação sonora

Gibber

O *Gibber*[57] é um sistema de *live coding* criado pelo artista e desenvolvedor de *software* inglês Charlie Roberts[58]. Uma das principais características do sistema Gibber é o funcionamento na internet, via *web browser – Google Chrome*, preferencialmente. É interessante notar a predominância técnica e de vanguarda experimental do navegador Chrome. Nos últimos anos, foram realizadas dezenas de obras audiovisuais com tecnologias que apareceram primeiramente nesta plataforma. Para rodar o *Gibber*, não é necessária a instalação de nenhum *software* adicional na máquina do usuário.

Uma característica única entre os sistemas de *live coding* e muito interessante do Gibber é a integração com o banco de sons Freesound (FONT, 2013), *website* criado e mantido pelo Grupo de Tecnologia Musical da Universidade Pompeu Fabra, em Barcelona. Podem-se usar *links* de sons do banco de dados diretamente na programação, sem a necessidade de se fazer o *download* dos sons previamente. Mais de 400 mil arquivos sonoros (amostras de sons de instrumentos, gravações de campo, ruídos, etc) são disponibilizados com a licença de uso livre *Creative Commons*[59]. Pode-se fazer a busca por sons diretamente do ambiente de programação ao vivo. O *Gibber* consegue trabalhar dinamicamente com o conteúdo do Freesound graças ao uso da biblioteca *freesound.js*[60], disponível gratuitamente para a criação de outros projetos com a linguagem *JavaScript*.

Aqui se concretiza o anseio futurista de Luigi Russolo, que em seu livro *A Arte do Ruído* preconiza: "Com a multiplicação sem fim do maquinário, um dia seremos capazes de distinguir entre dez, vinte, ou trinta mil diferentes ruídos. Nós não teremos mais que imitar esses ruídos, e poderemos combiná-los de acordo com nossa fantasia artística" (RUSSOLO, p.12). Acredito que essa capacidade de integração com banco de dados online coloca o sistema em outro patamar, de autonomia mais ampla, criativamente; cito aqui a artista e pesquisadora Maria Beatriz (Bia) Medeiros, sobre a comunicação da internet: "um

---

[57] *Site* do projeto *Gibber:* < https://gibber.cc >

[58] *Site* do desenvolvedor Charlie Roberts: < http://charlie-roberts.com >

[59] *Site* da Organização Não-Governamental Creative Commons, dedicada pesquisa de criação e difusão de licenças de compartilhamento livre de software e mídias digitais: < https://creativecommons.org >

[60] *Site* da biblioteca *JavaScript FreeSound.js:* < https://github.com/g-roma/freesound.js >

lugar onde as subjetividades se encontram e burlam o interminável desejo de controle da própria máquina" (MEDEIROS, p.11).

O medo sempre esteve ligado à ideia de desconhecido, daí a noção presente na minha pesquisa de desmistificação e perda de medo do contato mais direto com a máquina, na forma da programação de sistemas.

Chamo a atenção para uma tendência de criação de software com integração a bancos de sons *online*, como o *Freesound*, com vistas a práticas de *remix*. O *MScaper* [61], fruto do mestrado em engenharia do português Paulo Teixeira, permite a integração de buscas no banco de sons Freesound ao *software* de produção musical Ableton Live (TEIXEIRA, 2019). Já o projeto *Playsound.Space* [62], programa especializado em remix sonoro disponível na internet, permite a busca e a reprodução de sons do *Freesound*. O conceito de uso instantâneo de sons inesperados é algo muito excitante para produções ao vivo, constituindo uma funcionalidade que pretendo integrar nas minhas próximas criações direcionadas ao remix audiovisual.

Um dos aspectos criativos mais específicos e interessantes do sistema Gibber é a possibilidade visualização de ondas juntamente com o código-fonte, o que possibilita melhor compreensão da estrutura sendo desenvolvida, tanto por parte do performer quanto por parte do público. Outra característica que facilita o uso e a adaptação do Gibber a variados tipos de projetos é a integração – usando a biblioteca *Gibberish* [63] – com o ambiente de programação *Processing P5.js* [64], direcionado a produções artísticas de *software art*. Ademais, outra opção de trabalho com o sistema *Gibber* é o *Gibberwocky* [65], um sistema de live coding baseado no projeto *Gibber* e integrado ao *software* Ableton Live. O sistema *Gibber* foi desenvolvido pelos artistas-desenvolvedores Charllie Roberts e Graham Wakefield, que entendem que prática de *live coding* "tem demonstrado o valor do código maleável durante a sua execução, trazendo conceitos antes associados com a composição em estúdio e inserindo o desenvolvimento de *software* no universo da performance ao vivo." (WAKEFIELD, p.1) A colheita inesperada de *samples* sonoros, escolhidos randomicamente no momento da performance, dialoga com a pesquisa e a produção de John Cage que faz uso de aparelhos de rádio sintonizados em várias estações pelos músicos ou pelo público.

---

[61] *Site* do sistema *MScaper:* < https://sites.google.com/view/soundscapeusingwebaudioarchive >

[62] *Site* do sistema *Playsound.Space:* < http://playsound.space/ >

[63] *Site* da biblioteca para *P5.js Gibbersish:* < http://charlie-roberts.com/gibberish >

[64] *Site* do sistema de programação *P5.js:* < https://p5js.org >

[65] *Site* do *software Gibberwocky:* < http://gibberwocky.cc >

```
1  k=Freesound({query:'preach',pick:'random'});
2  k.note(0.6); k.amp=1.5;
3
4  m=Freesound({query:'french', pick:'random'});
5  m.note(0.6); m.amp=1.5;
6  n=Freesound({query:'portugues', pick:'random'});
7  n.note(0.6); n.amp=1.5;
8
9  e=Freesound({query:'kick 120bpm',pick:'random'});
10 e.note(1); e.loops=true; e.amp=1.5;
11 f=Freesound({query:'bass 120bpm',pick:'random'});
12 f.note(1); f.loops=true; f.amp=1.5;
```

Tabela 13. Código Gibber escolhendo sons do site FreeSound usando palavras-chave.

O trecho de código precedente pode ser copiado e colado por meio de um navegador de internet, no site < https://gibber.cc >.

Slang

O *Slang* [66] é um sistema de programação sonora desenvolvido em JavaScript na forma de *website*. O autor do projeto pode criar a sua própria gramática de funções musicais com a biblioteca, também compatível com o padrão JavaScript Ohm.js[67]. O sistema *Slang* roda dentro de um navegador de internet (*Chrome*, *Safari* ou *Firefox*) e é um dos únicos sistemas de programação ao vivo que podem ser utilizados em dispositivos móveis, com sistemas *Android* ou *iOS*.

```
1  @synth (adsr (osc tri) 64n)
2        + (pan (lerp -1 1 2)) + (gain 0.6)
3  play @synth (notes (random (chord minorblues (random [d3 c5]) 16)))
4
5  @bass (adsr (osc saw) 32n)
6  play @bass (notes (random (chord majorblues (random [e1 e2 c1]))))
```

Tabela 14. Código do sistema Slang escolhendo notas a partir de escalas musicais.

[66] *Site* do projeto *Slang:* < http://slang.kylestetz.com >

[67] *Site* do sistema *Ohm.js:* < https://github.com/harc/ohm >

Linha **1**: Define um sintetizador como instrumento, usando uma onda do tipo triangular (*tri*).

Linha **2**: Define o lado do instrumento (**pan**): cada nota será tocada em uma caixa de som de um lado (esquerda/direita).

Linha **3**: Toca notas aleatórias, escolhidas a partir da escala blues menor.

Linha **5**: Define um instrumento com o nome "*bass*", com onda do tipo serrote (*saw*).

Linha **6**: Toca notas aleatórias, escolhidas a partir da escala de *blues* maior.

Imagem 41. Interface do sistema online Slang.

SuperCollider

O *SuperCollider* [68] é um *software* livre de síntese sonora muito poderoso, que, além de poder ser usado isoladamente, trabalha em conjunto, transparentemente, por detrás de vários *softwares* do *live coding*, tais como o *FoxDot*, o *TidalCycles* e o próprio *Sonic Pi*. Sua linguagem de programação, *sclang*, tem sintaxe relativamente complexa se comparada a outros *softwares* de *live coding*. Juntamente com sistema *ChucK*, é um dos projetos de programação sonora mais longevos.

[68] *Site* do projeto *SuperCollider:* < https://supercollider.github.io >

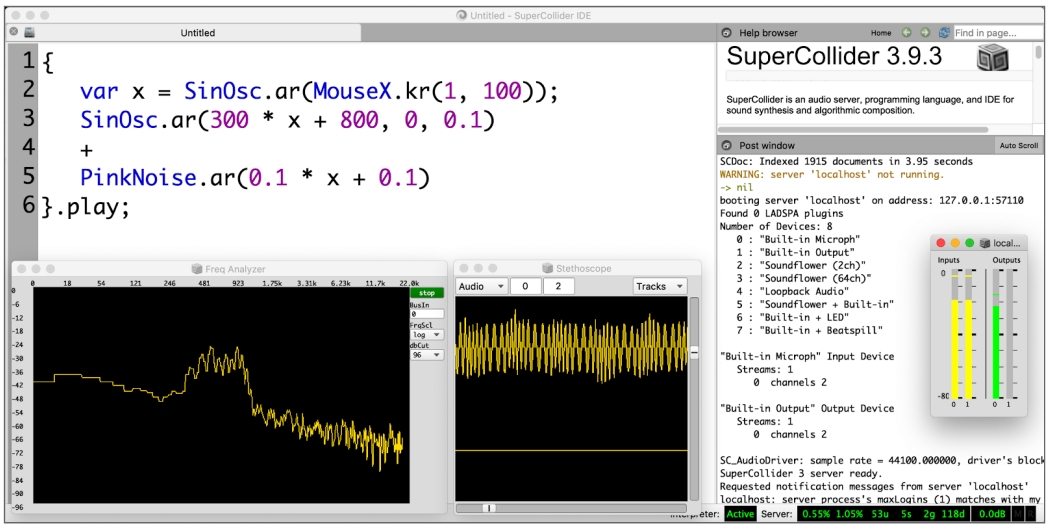


Imagem 42. Interface de programação sclang do SuperCollider.

ChucK

O sistema *ChucK*[69] foi desenvolvido em 2002 pelo artista e pesquisador chinês Ge Wang, na Universidade de Princeton, nos Estados Unidos. É um dos mais antigos programas de *live coding* com desenvolvimento ativo. É compatível com os sistemas operacionais *Linux*, *macOS* e *Windows*.

Como um exemplo raro de integração direta entre sistemas de *live coding* e programa de edição de áudio (*DAW – Digital Audio Workstation)*, foi desenvolvido um plug-in VST chamado *Chuck Rack*[70], que permite a criação e a execução de código Chuck dentro desses programas (HOCHENBAUM, 2017). O sistema *ChucK Rack* permite: 1) processar efeitos no áudio vindo do programa hospedeiro; 2) processar notas MIDI vindo do programa hospedeiro; 3) gerar comandos MIDI ou OSC com programação *ChucK*; 4) utilizar o sistema Chuck como um sintetizador, para ser tocado pelo programa hospedeiro; 5) sincronizar o tempo (BPM - Batidas Por Minuto) do *ChucK* com o programa hospedeiro, e 6) modificar variáveis do *ChucK* usando canais de automação do programa hospedeiro.

O Professor Ge Wang faz a seminal declaração em sua tese de doutorado em ciências da computação: "***code == instrument***" (código é igual a instrumento). (WANG, 2008, p.5)

---

[69] Site do projeto *ChucK*, mantido pela Universidade de Princeton:
< http://chuck.cs.princeton.edu >

[70] Site para download do *ChucK-Racks:* < https://github.com/mtiid/chuck-racks >

Tone.js

O Tone.js é uma biblioteca de programação musical com a linguagem JavaScript, própria para criação de programas para internet. Não é exatamente um sistema para performances de *live coding* – apesar de ter características em comum com essa categoria de programas – mas possibilita a criação de peças generativas para execução via navegador de internet. O *Tone.js* é uma boa companhia ao *Three.js*, biblioteca de programação visual, pois ambos são sistemas *JavaScript*, conhecidos por serem de fácil integração uns aos outros.

2.6 Comparação entre sistemas de live coding

Na tabela abaixo, podemos comparar a compatibilidade de alguns sistemas de programação ao vivo com os sistemas operacionais mais comuns.

	Android (mobile)	*iOS (mobile)*	*Linux (desktop)*	*macOS (desktop)*	*Windows (desktop)*
ChucK			✓	✓	✓
Extempore			✓	✓	✓
FoxDot					
*Gibber**			✓	✓	✓
ORCA			✓	✓	✓
Overtone					
*Slang**	✓	✓	✓	✓	✓
Sonic Pi			✓	✓	✓
TidalCycles			✓	✓	✓

Tabela 15. Compatibilidade dos sistemas de live coding com sistemas operacionais.

Os sistemas *Gibber* e *Slang* funcionam por meio da internet, rodando dentro de um navegador (*Google Chrome*, preferencialmente). Apesar dos dois sistemas funcionarem na *web*, somente o *Slang* é compatível com dispositivos móveis *Android* e *iOS*.

Neste capítulo, vimos as principais ferramentas para produção sonora com foco em técnicas de programação textual. Compreendemos, por meio dos exemplos de obras e seus processos constituintes, a força criativa da subjetividade aplicada a cada escolha de linha de programação e na abertura aos valores aleatórios. Observamos as potencialidades poéticas e as implicações técnicas da escolha de cada sistema de produção audiovisual aumentada pelo computador.

Capítulo 3. Programação Visual

"Nós não vemos as coisas como elas são, nós as vemos como nós somos."
Anaïs Nin

Este capítulo apresenta as ferramentas de programação ao vivo de arte visual e minha busca de uma interface com possibilidades criativas e programação de regras de comportamentos, dando à obra elementos generativos. Também mostro, exemplificando com obras de minha autoria, o uso do tempo de gravação e de reprodução das imagens como material criativo, expandindo *loops* e criando mixagens com adição de camadas.

Ao citar os exemplos de ambientes de criação que já utilizei e ainda utilizo, abordo questões sobre a obsolescência dos *softwares* e movimentos mercadológicos relevantes à produção artística.

Inicio o tema da programação visual com minha primeira experiência de criação de arte computacional contemplando a aleatoriedade, que foi a videoarte *Floresta*[71] , exibida na exposição *OBRANOME II*, no Parque Lage, Rio de Janeiro (2011) e no Museu Nacional da República, em 2010. O vídeo foi gerado a partir de um *software* próprio, desenvolvido no ambiente *Director*, que executava as seguintes regras:

1) Acessar um grupo de 40 fotos do centro de Brasília, perto do Restaurante Floresta[72] (daí o nome da obra e a analogia com a selva de concreto da cidade);

2) Escolher 4 imagens aleatoriamente;

3) Mixar as fotos, variando aleatoriamente nos parâmetros de posição, tamanho e modo de composição;

4) Aguardar o resultado (cerca de 1 minuto de espera por quadro) e salvar um *frame* do vídeo;

5) Repetir o processo, variando os parâmetros, ora suavemente, ora abruptamente, resultando em um ritmo constante dos desdobramentos imagéticos.

---

[71] Vídeo *Floresta (10'45")*: < www.youtube.com/watch?v=uI36dl6y_CI >

[72] O ponto central de Brasília retratado nas fotos é um viaduto que, sem manutenção, desabou em 2018.

Imagem 43. Frame do vídeo generativo Floresta.

3.1 *Blender*

O *Blender* [73] é um *software* livre de criação 3D completo que pode ser usado para produções visuais ao vivo ou tradicionais, quando quadros de animação são modelados, animados e preparados previamente ao momento da exibição. Possibilita a programação por meio de programação textual (com a linguagem *Python*) ou conexões de blocos visuais (*nodes*), que se assemelham a fluxogramas, tanto na funcionalidade quanto no aspecto visual. Um fluxograma pode informar – a um ser humano ou a um computador – quais algoritmos executar e que caminhos seguir, de acordo com o resultado de cada etapa.

Ter conhecido pessoalmente Ton Roosendaal, desenvolvedor holandês que criou o *Blender*, foi um grande marco psicológico para mim, pois percebi que era possível criar *softwares* profissionais sem a estrutura de grandes empresas. Tal encontro ocorreu em 2004, quando apresentava, na *Conferência Internacional Blender* 2004, em Amsterdã, o projeto *Brasília 3D*.

Ressalto que o *Blender* é o programa que estudo mais profundamente e que utilizo com maior frequência em meus projetos, desde o começo dos anos 2000. Graças ao seu design aberto a modificações, o *Blender* vem sendo palco de experimentações no tocante à integração com outros *softwares*.

---

[73] *Site* do Instituto Blender: < www.blender.org >

Aprofundei minha relação com a ativa comunidade do *Blender* ao participar do *Blender CAVE Development Sprint*[74] (um esforço coletivo de desenvolvimento do *software 3D* para o sistema imersivo de realidade virtual *CAVE*[75]), realizado no *Zukunftszentrum Tirol* (Centro Tirolês de Estudos do Futuro), na Áustria, em 2006.

Como uma das primeiras iniciativas de integração inter-programas, foi desenvolvido o sistema Blendnik (PORCARO, 2009), que integra o *Blender* à plataforma de criação sonora Pure Data – essa foi uma de minhas inspirações para continuar a pesquisa de integração do *Blender* a outros programas de criação audiovisual.

Controle MIDI e OSC

Os *softwares* livres *AddMIDI* e *AddOSC*[76] permitem que o *Blender* receba informações externas com o objetivo de controlar parâmetros de modelagem, texturização e animação de objetos em cenas tridimensionais. Os dois sistemas são distribuídos no formato de *add-ons*, sendo instalados e operados dentro do *Blender*. Cada um dos sistemas tem características próprias, como observado na tabela a seguir:

	AddMIDI	***AddOSC***
Mensagens	Permite mensagens numéricas com valores entre 0 e 127.	Permite mensagens numéricas com valores inteiros ou fracionais, negativos ou positivos, de praticamente qualquer magnitude.
Conectividade	Permite a conexão de outros programas e instrumentos musicais.	Permite a conexão de outros programas rodando no mesmo computador, em redes locais ou na internet.

Tabela 16. Comparação entre os protocolos de comunicação MIDI e OSC.

Foi lançada em setembro de 2019 uma nova versão dos *softwares AddMIDI* e *AddOSC*, reunidos em um *software* livre só, chamado de *MOM*[77]. Atestando a velocidade de desenvolvimento, tanto da área de arte computacional quanto na área de software livre

[74] Vídeo de registro do evento: < www.youtube.com/watch?v=mHCes6bDBLQ >
[75] Detalhamento do sistema *CAVE* de imersão em realidade virtual: < https://en.wikipedia.org/wiki/Cave_automatic_virtual_environment >
[76] Páginas dos sistemas *AddMIDI* e *AddOSC*: < www.jpfep.net/pages/addmidi > e < www.jpfep.net/pages/addosc >
[77] Página do *software MOM:* < www.jpfep.net/en-us/pages/addons/midi-osc-and-more >

(não só vinculado ao *Blender*). Os sistemas *AddOSC*, *AddMIDI* e *MOM* foram criados pelo desenvolvedor francês Jean-Philippe Pailler.

Blender com *Sverchok*

Sverchok é um *software* livre russo que funciona dentro do *Blender* como uma adição (*add-on*). O Sverchok foi inspirado no sistema de modelagem paramétrica *Rhino*[78], cuja linguagem de programação visual com algoritmos se chama *Grasshopper*[79] (grilo em inglês). *Sverchok* significa grilo em russo: uma referência ao sistema *Grasshopper*, um dos primeiros sistemas de criação 3D por meio de fluxogramas visuais.

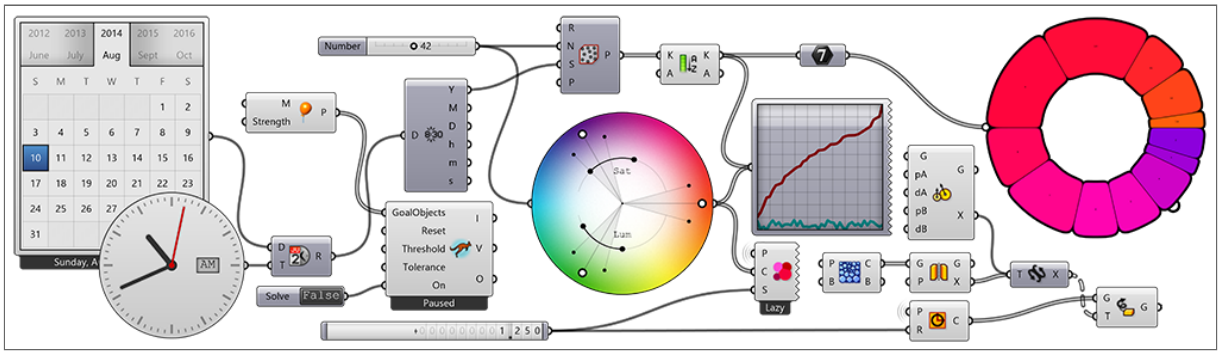


Imagem 44. Interface visual de programação do software Rhino / Grasshopper.

Blender com *Animation Nodes*

Animation Nodes[80] é um *software* livre no formato de *script* adicional ao *Blender* (na terminologia do *Blender*, um *add-on*) que permite a criação de objetos e animações paramétricas. O projeto for criado e é mantido pelo desenvolvedor Jacques Lucke. Desde 2019, Lucke vem trabalhando com o *Instituto Blender* na implementação do projeto *Everything Nodes* (fluxograma total), com o objetivo de criar um sistema paramétrico que seja abrangente a todas as funções do *Blender*.

3.1.1 *Esculturas Quânticas*

Esculturas Quânticas é uma obra interativa que explora infinitas possibilidades de interação corpo-matéria escultórica virtual. Cada participante, ao interagir com o sistema usando o corpo, cria infinitas possibilidades de transformações sonoras e visuais. As formas desenvolvidas pelo sistema fazem referência à Teoria Quântica de Campos, que

---

[78] *Site* do software *Rhino 3D*: < www.rhino3d.com >

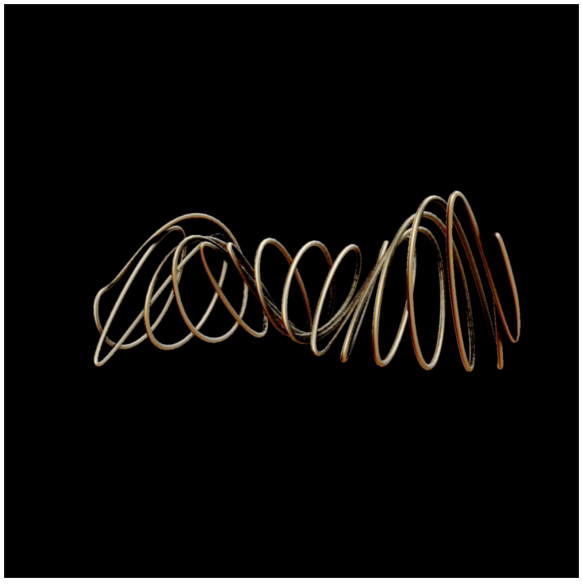
[79] *Página* sobre o sistema Grasshopper: < www.rhino3d.com/6/new/grasshopper >

[80] *Site* do projeto *Animation Nodes*, software complementar ao *Blender*: < https://github.com/JacquesLucke/animation_nodes >

descreve o posicionamento virtual de partículas a partir da influência do observador. Faço na obra uma referência visual e conceitual com o tema do entrelaçamento quântico – também referido por “movimento fantasma” –, que descreve uma relação entre as características atômicas (posição, momento, *spin* e polarização) de duas partículas diferentes, porém em sincronia. Estendo a metáfora para o público agente e a obra reagente.

Propus-me a criar “sistemas escultóricos” com propriedades sonoras e visuais, em forma de móbiles digitais mutantes animados. Ao ver os primeiros resultados da fusão entre modelagem, efeitos geométricos e *live coding*, imaginei determinados momentos das animações das formas resultantes como possíveis congelamentos de existência independente da imagem em movimento. Nesse sentido, o processo de criação de cada sistema busca a característica de possíveis materialidades. O espaço e o tempo desdobrado constituem um campo multidimensional de existência da obra. A temática já havia sido explorada na obra *Bichos Impossíveis*.

Uma inspiração estética sempre pairando no ar durante as etapas de preparação das obras foi o trabalho de Alexander Calder com seus móbiles. Por que nomeá-las de esculturas? O processo de construção é uma investigação quadridimensional (3 dimensões, mais o tempo) a respeito da materialidade possível de cada peça. Por que nomeá-las de quânticas? Porque existem infinitas possibilidades latentes, derivadas dos algoritmos de modelagem, dos processos randômicos, e das possíveis interações com o artista e com o público.



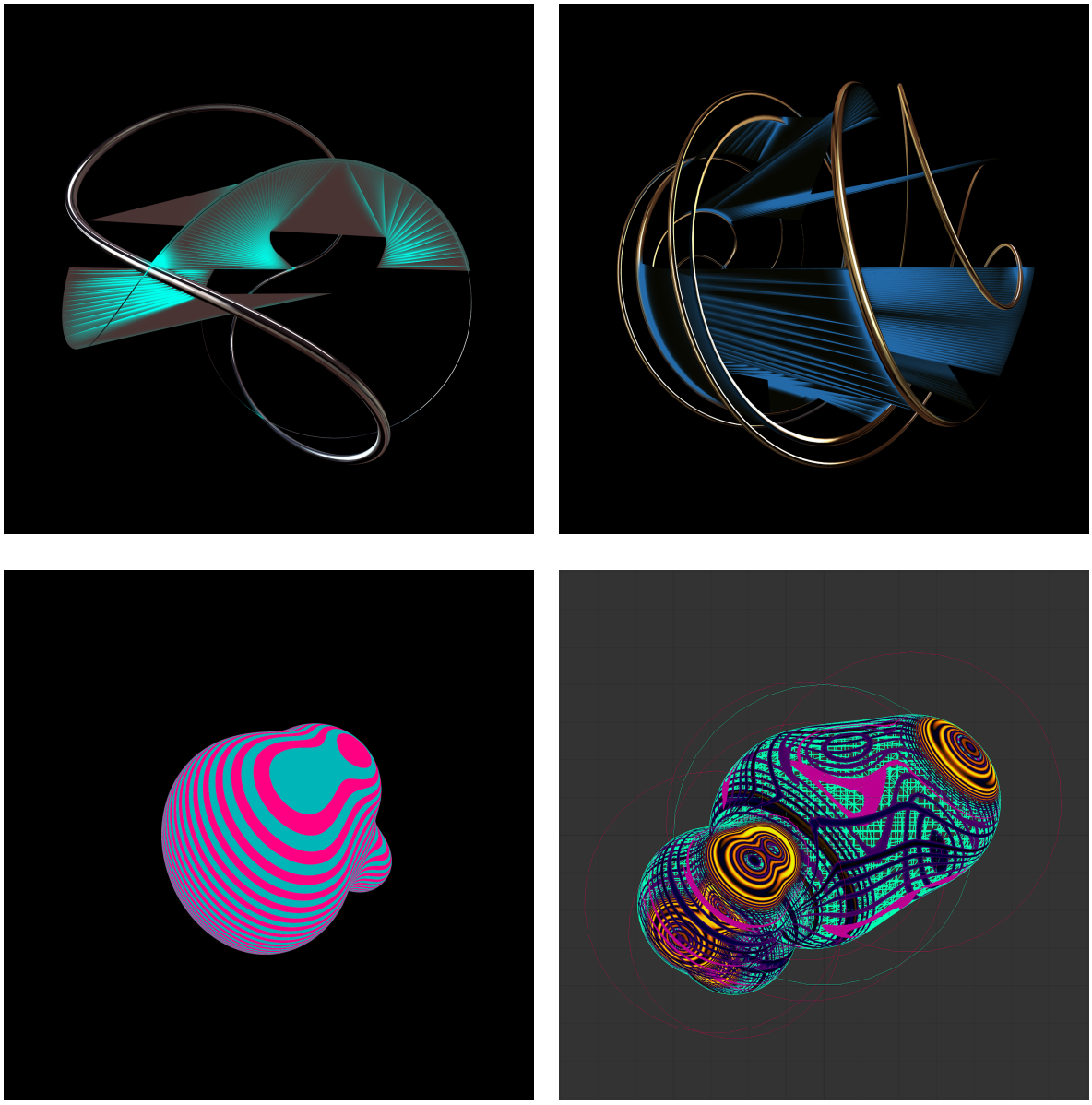


Imagem 45. Estados diferentes do sistema Esculturas Quânticas.

Procurei, ao desenvolver a obra, fazer um *hacking* (no sentido de uso não previsto) do sistema, trilhando caminhos técnicos e conceituais heterodoxos. Para a criação da obra, desenvolvi um ecossistema de ferramentas que consistiu de um *software* de gerenciamento de interatividade (*Processing*), de um sistema de *live coding* (*Sonic Pi*) e do *software* de criação 3D (Blender). Foi integrado à obra um sensor de posicionamento tridimensional, o *Microsoft Kinect*, abordado em detalhes no Capítulo 4, Seção 4.3.1.

A base da interação do trabalho é a descoberta, por parte de público, dos desdobramentos audiovisuais de seus movimentos corporais. Dentro do computador, o que está acontecendo é uma interpretação da posição dos usuários. Se mais de uma pessoa se

posicionar na frente do sensor, o sistema calcula a posição média dos participantes, tornando a interação coletiva.

Para a composição musical, o sistema, munido do posicionamento horizontal e vertical dos participantes, envia os dados como duas variáveis para o *Sonic Pi*. A programação sonora desdobra-se utilizando esses valores, primeiramente escolhendo uma dentre cerca de 20 cenas sonoras diferentes. Cada cena sonora utiliza os valores do sensor de forma diferente, seja para a produção de notas musicais isoladas ou acordes, seja para a escolha de instrumentos do *Sonic Pi* ; a maioria das cenas tem o posicionamento horizontal como referência para a escolha de escalas e de notas musicais dentro das escalas. O posicionamento vertical é normalmente usado para orientar o volume das notas e a porcentagem de aplicação de efeitos sonoros, tais como eco e reverberação.

No tocante às modificações visuais agenciadas pelo *software*, o sistema consiste de aproximadamente 20 cenas criadas no ambiente *Blender*. Os elementos visuais, prioritariamente, seguem os movimentos horizontais e verticais dos participantes. A fim do sistema nunca ficar parado, animações semi-aleatórias são constantemente executadas. Quando um usuário é detectado na área de percepção do sensor, são adicionadas animações ao estado padrão de movimentações. Busquei um meio-termo entre as reações derivadas diretamente dos movimentos dos participantes e as animações incontroláveis, com o objetivo de manter uma aura de mistério e de funcionalidades a serem descobertas por meio da exploração intuitiva.

Imagem 46. Interação do público com a obra Esculturas Quânticas em suporte de videowall.

É interessante notar o alto nível de conexão com a obra advinda da interação na escala humana de tamanho. É como se as esculturas fossem extensões dos corpos a dançar. Quando projetadas em escalas maiores que a humana, ocorre uma troca de ponto de vista subjetivo; deixa-se de controlar uma forma do tamanho de um monitor para se controlar algo que nos coloca como pequenos e à mercê das luzes ao nosso redor.

Uma versão interativa da obra foi exposta no evento corporativo *Fenae Inspira: Transformações*, em Belo Horizonte, em 2019. Outra versão foi exposta no *Festival Brasília Mapping*, no Museu Nacional da República, em Brasília, também em 2019. Nessa projeção mapeada, o conteúdo audiovisual podia ser controlado com o uso do sensor *Kinect 2*, mais rápido e preciso que a versão anterior do equipamento. Quanto ao desenvolvimento do sistema, vale notar que o *Kinect 2* é compatível somente com o sistema operacional *Windows*.

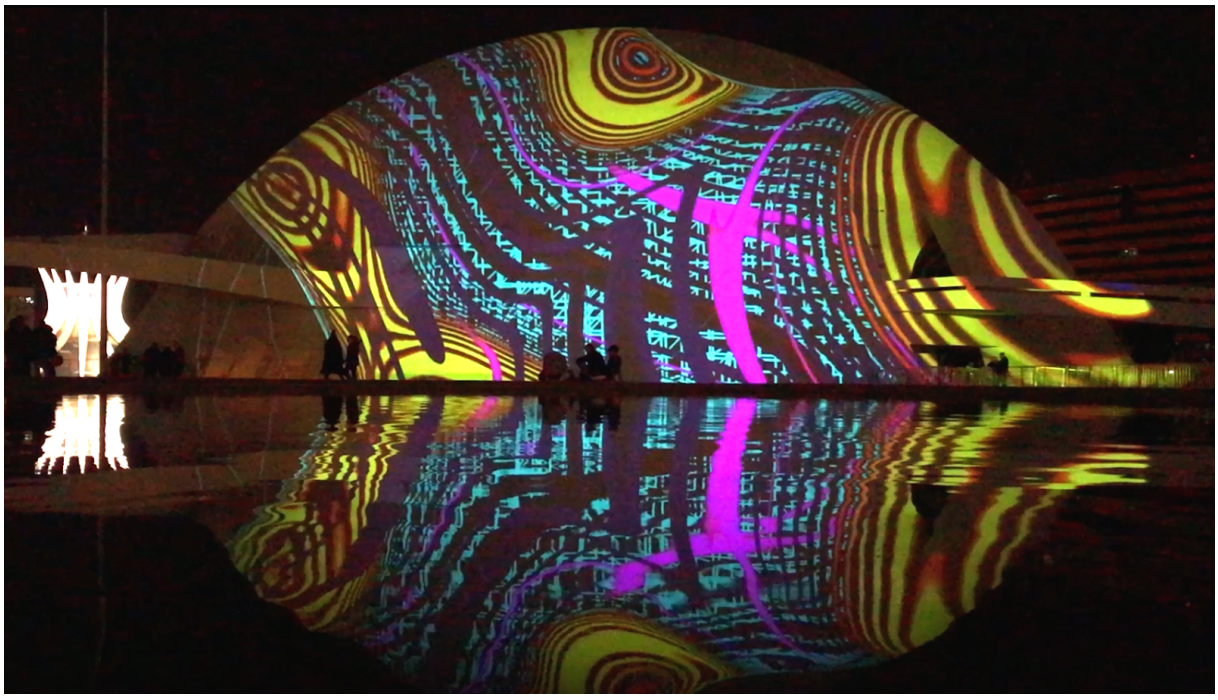


Imagem 47. Esculturas Quânticas projetadas no Museu Nacional da República.

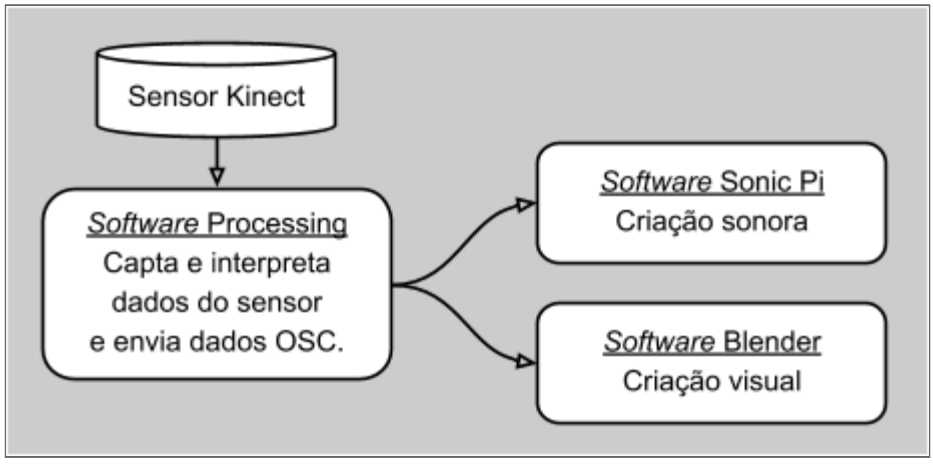


Gráfico 7. Fluxo de dados da obra Esculturas Quânticas.

Durante os testes dessa obra, a fim de liberar o desenvolvimento da presença do sensor e do espaço físico necessário para a interação, programei uma forma de interação com o mouse do computador, simulando a posição do público diante da obra. O uso do *mouse* permitia a interação audiovisual com a obra, desempenhando o papel de um sensor. Mesmo sem conhecê-la previamente, acabei desenvolvendo um mecanismo de funcionamento similar ao da obra *Music Mouse – An Intelligent Instrument* [81], criada pela compositora americana Laurie Spiegel em 1986. O *software* original de Spiegel não

[81] Site com detalhamento do software Music Mouse, de Laurie Spiegel:
< http://retiary.org/ls/progs/mm_manual/mouse_manual.html >

funciona em computadores recentes, mas uma versão *online* foi desenvolvida pelo pesquisador e desenvolvedor de *software* italiano Tero Parviainen[82].

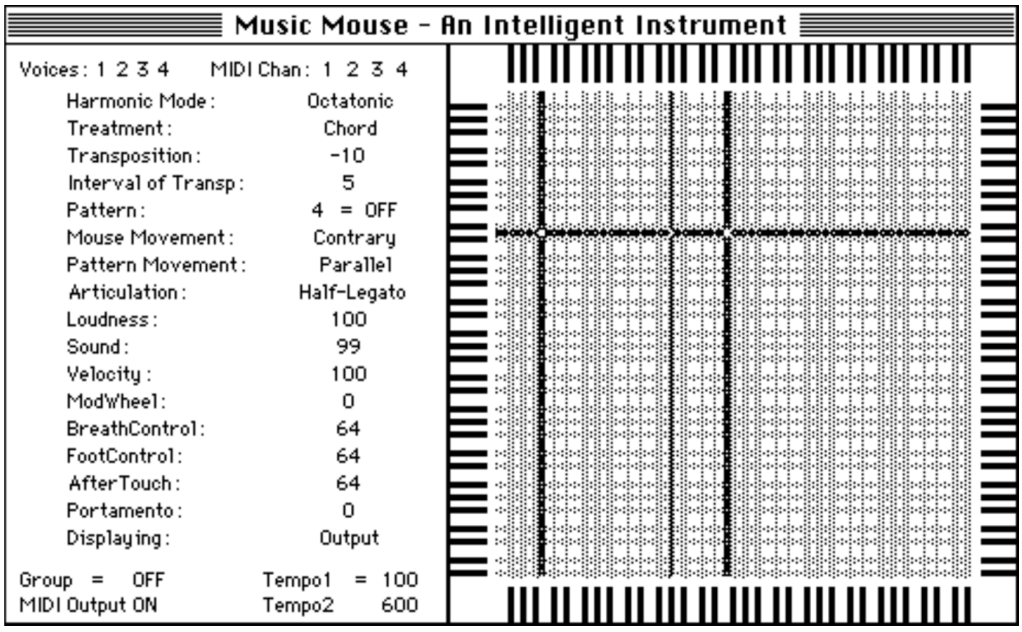


Imagem 48. Interface do software Music Mouse.

Para o desenvolvimento estético da obra, segui a estética *vaporwave*, que adota uma paleta de cores específica (rosa, azul e tons de verde, brilhantes como letreiros de neon) que remete ao passado próximo da década de 1980, tempo da invenção dos primeiros computadores e jogos eletrônicos caseiros. Existe uma relação com o trabalho, uma vez que este é baseado no sistema simples de programação do *Sonic Pi* – que comparo à linguagem *BASIC*, amplamente usada nas décadas de 1970 e 1980. Essa época vivia uma visão utópica do futuro da computação, talvez como hoje vemos os avanços recentes em programação de inteligência artificial.

O processo de criação pode ser padronizado seguindo as seguintes etapas:

1) Criação de uma cena tridimensional dentro do *software Blender*, trabalhando em conjunto com os sistemas de criação paramétrica *Sverchok* e *Animation Nodes;*

2) Escolha de parâmetros a serem conectados ao *Sonic Pi*. Normalmente entre um e dez parâmetros, tais como posição, cores, deformações e outras características geométricas dos objetos;

3) Composição musical com programação do *software Sonic Pi* ;

---

[82] Versão *online* do *software Music Mouse:* < https://teropa.info/musicmouse >

4) Preparação e envio de comandos do *Sonic Pi* para o *Blender*, em sincronia com a execução musical. Foram utilizados os scripts *AddOSC* e *AddMIDI* para envio de mensagens *OSC* e *MIDI*, respectivamente. Encontrei mais versatilidade e robustez ao trabalhar com o formato de comandos *OSC*, adotado na versão mais recente da obra.

O trabalho de "lapidação" das obras da série consistiu de uma exploração do ciclo de criação de regras e percepção de resultados. A emergência nasce da exploração das simetrias e contrastes entre a produção imagética, a partir de composição sonora e o caminho inverso (imagem gerando sons).

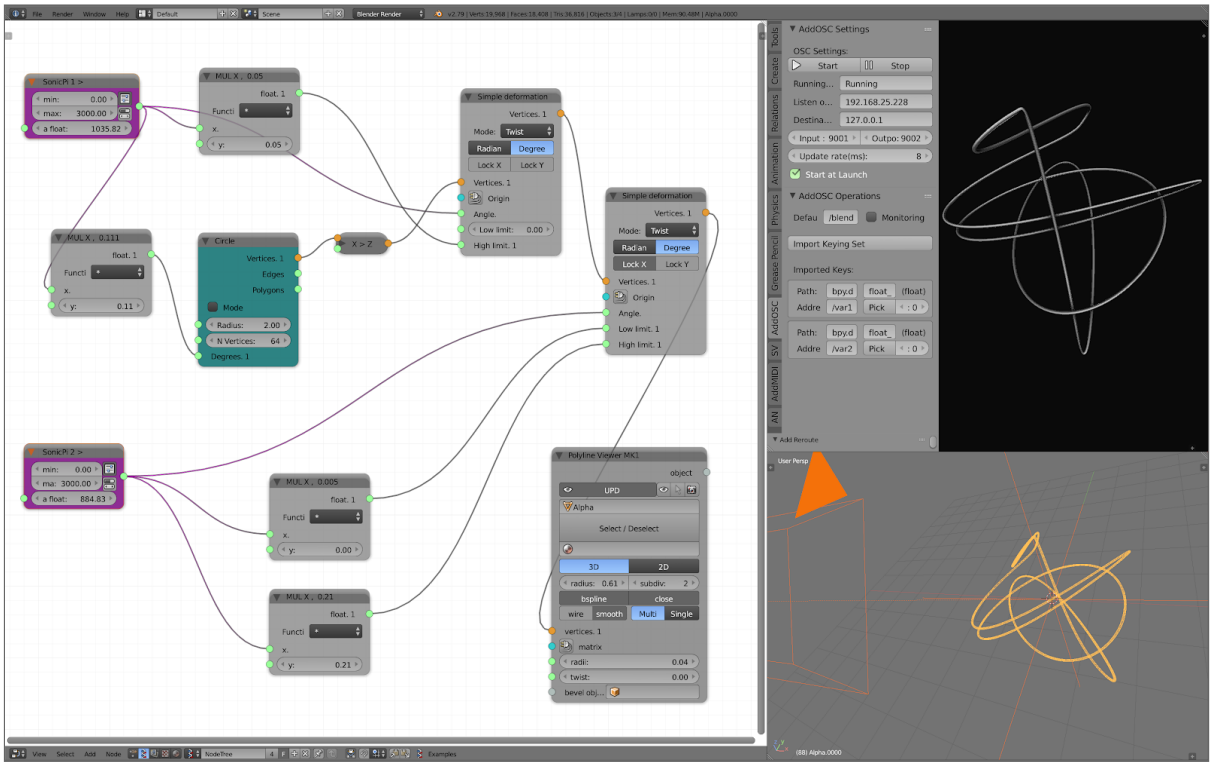


Imagem 49. Objeto animado sendo criado no sistema Sverchok com dados do Sonic Pi.

3.1.2 *Cubos em Dança*

A obra é uma homenagem ao artista americano Sol LeWitt (1928-2007), e constitui uma releitura, uma reinterpretação livre da obra escultórica *Incomplete Open Cubes* (Cubos Abertos Incompletos), série realizada nos anos 1970.

Detalho nesse trecho o processo de ideação e criação de *software* da obra *Cubos em Dança: Homenagem a Sol LeWitt*. A obra foi realizada com o artista e pesquisador de conservação de obras de novas mídias Teófilo Augusto da Silva.

O processo começou com uma pesquisa de adequação conceitual. O trabalho de Sol LeWitt orbita em torno de repetições e variações sobre temas simples, como a forma de um cubo – tema relacionado aos princípios da arte generativa – que explora combinações e o desdobramento de formas no espaço e no tempo (tratando-se de animações ou obras interativas). Sol LeWitt chamava as suas obras escultóricas de "estruturas", pela natureza matemática e algorítmica de suas suas criações. Em entrevista durante a exposição de LeWitt, no Museu de Arte Moderna de São Francisco, realizada em 1999, o curador Gary Garrels salienta que o artista preocupava-se com a geometria, com a percepção e com a forma como o olho, a mente e o espaço físico interagem entre si. Na obra *Incomplete Open Cubes,* jogava com o equilíbrio entre ordem e caos visual, dependendo do ponto de vista a partir do qual a obra é observada. (LEWITT)

O passo seguinte foi o estudo das possibilidades técnicas para a realização da obra. Uma das possibilidades técnicas vislumbradas para o desenvolvimento da obra foi o uso de algoritmos de *L-System* (Sistema-L), um sistema de regras desenvolvido pelo biólogo Aristid Lindenmayer em 1968 para descrever o comportamento de crescimento de células de plantas (LINDENMAYER, 1990). O Sistema-L funciona como se estivéssemos "movendo uma tartaruga", tal qual a técnica da linguagem de programação para aprendizado infantil LOGO[83], inventada em 1967.

Quanto ao formato final da obra, as possibilidades especuladas foram a criação de uma animação, de um *software* interativo ou de uma experiência de realidade aumentada. A fim de escolher a tecnologia mais adequada ao desenvolvimento da obra, examinamos as possibilidades técnicas de cada uma das plataformas de criação disponíveis, principalmente as relacionadas aos sistemas de distribuição e exibição da obra.

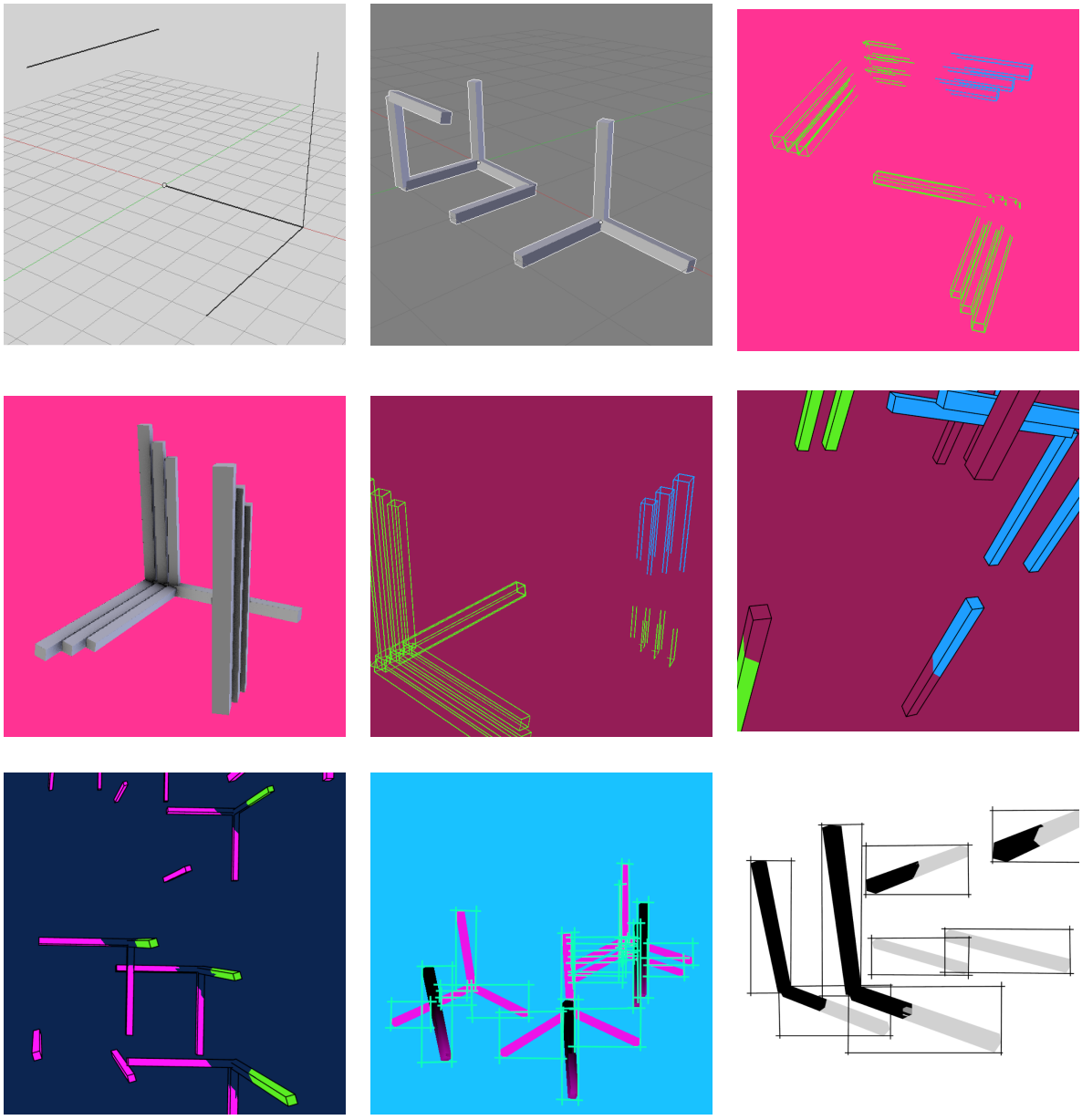
Ambiente de desenvolvimento	**Características possíveis da obra realizada**
Blender com *Animation Nodes*	A obra realizada pode tomar a forma de uma animação ou de um sistema aberto, controlado por sensores ou por outros *softwares*, como por exemplo o *Sonic Pi* ou o *Processing* (ambos por meio de comandos no padrão *OSC*).
Unity	A obra realizada pode tomar as mesmas formas das possíveis através do sistema *Blender*, além da possibilidade de poder virar um aplicativo para sistemas móveis ou de mesa, ou rodar na internet como aplicativo *WebGL* (somente para *desktop*).

[83] História da linguagem *LOGO:*
< https://en.wikipedia.org/wiki/Logo_(programming_language) >

Three.js	A obra realizada poderia ser exibida interativamente na internet como *JavaScript*, compatível com dispositivos *mobile* ou *desktop*.

Tabela 17. Possibilidades de uso de softwares para a obra Cubos em Dança.

Acredito que essas definições definem quais são os pontos de sensibilidade artística na produção: pontos-chave para modificação da programação para inserção de interações profícuas ao aprofundamento dos processos conceituais e estéticos em curso.



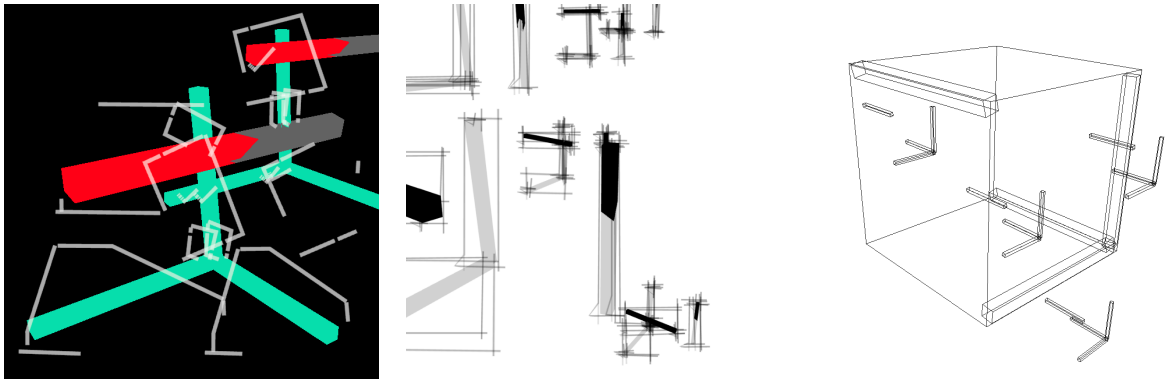


Imagem 50. Quadros de desdobramentos da obra Cubos em Dança.

Cada canto do cubo pode emanar linhas em três direções para gerar geometrias que venham a se concatenar em cubos completos (ou incompletos), de acordo com a gramática formal do Sistema-L escolhida (cada letra dirige a linha seguinte para uma direção diferente). No sistema de regras criado, toda a programação de criação das variações da peça e todas as animações das peças e do ponto de vista são gerenciados pelo fluxograma a seguir:

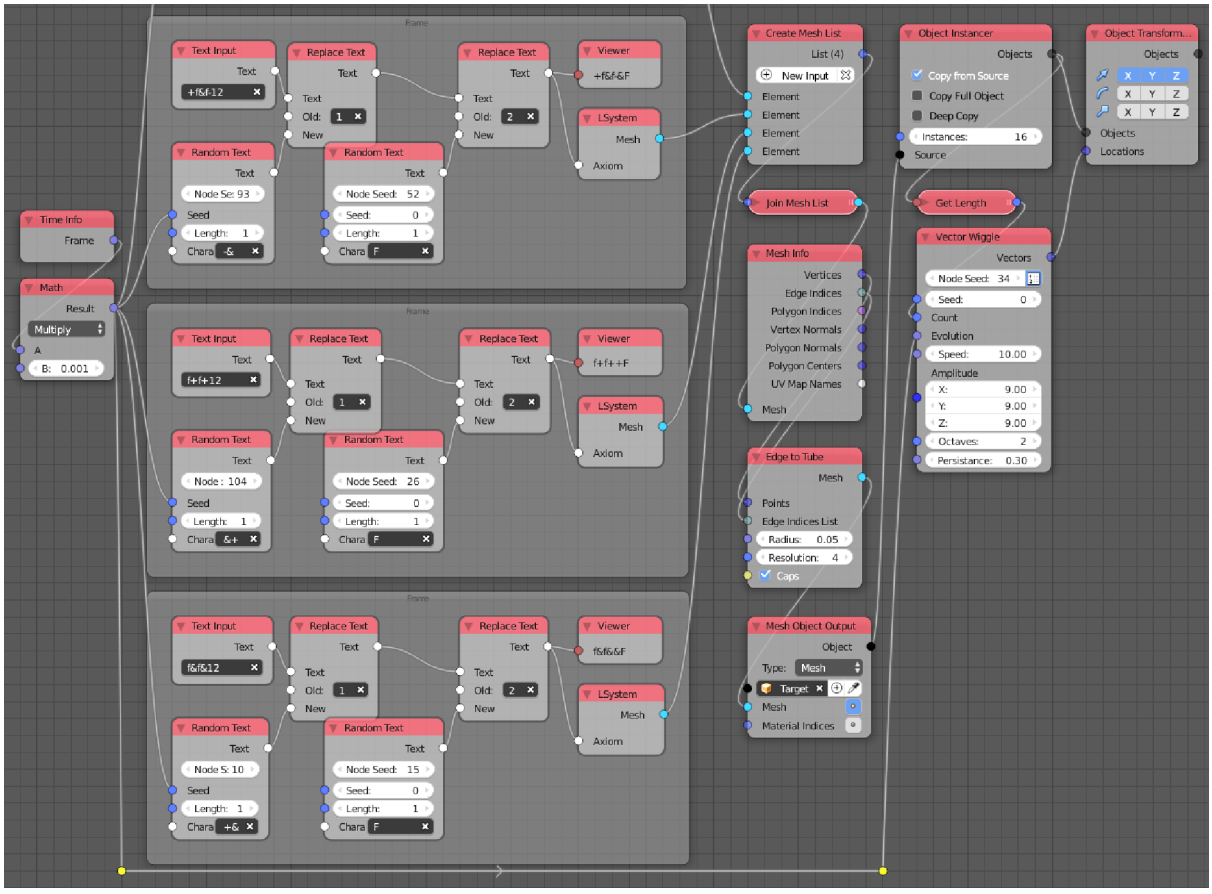


Imagem 51. Programação tipo fluxograma Blender / Animation Nodes da obra Cubos em Dança.

3.2 *Processing*

O *Processing* [84] é um ambiente de programação simplificado capaz de reduzir de semanas para dias ou horas o ciclo de criação de um programa. A programação no processing é feita textualmente, com uma sintaxe (simplificada) derivada da linguagem *Java*. Considero o *Processing* uma poderosa ferramenta de integração entre outros programas, como veremos nos detalhamentos a seguir. Ou seja, o *Processing* é a grande cola que agrega capacidades de *softwares* diferente e permite a criação de sistemas computacionais híbridos.

Sistema estável, é muito bem documentado, tanto pela Fundação Processing quanto pela comunidade internacional de desenvolvedores e educadores e desenvolvedores de *softwares* criativos – que como prática vem sido chamada de *creative coding* (programação criativa). Entende-se por *software* criativo qualquer programa de computador com finalidades artísticas, seja uma animação interativa, seja um *software* de edição de vídeo. O que diferencia um *software* criativo de uma videoarte ou animação computadorizada é a possibilidade de interação do espectador/usuário com a obra. O produto distribuído, vendido ou exposto é um programa sendo executado em um computador. Algoritmos presentes no código são como regras indicando o que o programa deve fazer em cada circunstância prevista pelo seu criador. Um programa pode ter uma resposta predefinida para movimentos do *mouse*, teclas do teclado, sons captados no ambiente ou até para imagens captadas por câmeras conectadas ao sistema. A maioria das interações é codificada com estruturas do tipo *SE [...], ENTÃO [...] – IF [...], THEN [...]*: "*Se acontecer tal evento, execute tal ação*". É uma estrutura de fluxograma lógico que pode ser representado por código textual ou em forma de fluxograma visual.

O ambiente de desenvolvimento completo do *Processing* facilita a codificação de gráficos, vídeos e sons, usando programação textual. Sua capacidade é aumentada com o uso de bibliotecas também de *software* livre, criadas ou adaptadas para uso dentro do ambiente de programação do Processing. As principais bibliotecas utilizadas foram *GLVideo* (para otimização da reprodução de arquivos de vídeo), The *MIDI Bus* (para possibilitar a comunicação do Processing com equipamentos *MIDI*), *Processing PostFX* (para a aplicação de efeitos de vídeo), e *ControlP5* (para a criação de elementos de interface interativos). Estes *softwares* gratuitos, quando adicionados ao ambiente de programação *Processing*, adicionam diversas facilidades à programação do *software*.

---

[84] *Site* do sistema de programação audiovisual *Processing:* < www.processing.org >

Uma importante contribuição para a comunidade é o canal do *YouTube Coding Train*[85], no qual Daniel Shiffman[86] ministra aulas sobre tópicos técnicos e execução de projetos multimídia com o sistema *Processing*.

3.2.3 *PiControl*

Desenvolvi o *software PiControl* como um sistema acessório ao *Sonic Pi*, com a finalidade de tornar as performances de *live coding* mais dinâmicas expressivas e as composições mais interessantes, controlando partes dos códigos musicais com controladores MIDI. Controladores MIDI são equipamentos eletrônicos que permitem interação com *softwares* de criação musical; podem ter a forma de teclados de piano, baterias eletrônicas com *pads* sensíveis à pressão e/ou a combinações de *knobs* (botões de rotação) e *sliders* (botões deslizantes verticais ou horizontais). Os equipamentos MIDI pertinentes à pesquisa são detalhados no Capítulo 4, Seção 4.2.2.

Para o funcionamento deste *software*, concebi o seguinte caminho, "ingênuo, porém eficiente" (Sam Aaron, em *e-mail* pessoal):

1) Um controlador *MIDI* é conectado via cabo *USB* a um computador rodando sistema operacional *Linux*, *macOS* ou *Windows*;

2) O código desenvolvido no *Processing* (*Java*) lê as informações enviadas pelo controlador MIDI;

3) O parâmetro do controlador sendo modificado é gravado em um arquivo de texto (no formato TXT);

4) O código desenvolvido no *Sonic Pi* (*Ruby*) lê cada arquivo de texto e vincula o valor lido a uma variável musical definida pelo músico / *performer* / programador.

3.3 *Three.js*

O *Three.js* (três, de 3 dimensões) é um sistema de criação multimídia muito interessante, pois é totalmente baseado na linguagem *JavaScript*. Isso permite que ele se comunique com centenas de outros projetos realizados com programação *JavaScript*, na sua grande maioria disponibilizados como *softwares* livres, com código-fonte disponível e documentado na internet.

---

[85] Canal *Coding Train:* < www.youtube.com/channel/UCvjgXvBlbQiydffZU7m1_aw >

[86] *Site* do educador Daniel Shiffman: < https://shiffman.net >

Na produção de um estágio inicial da obra *Tocando Brasília*, utilizei a programação *JavaScript* do sistema *Three.js* para utilizar o sistema de mapas online *openStreetMap*[87] com relativo sucesso. O obstáculo não transposto foi a ligação do conteúdo de mapas a algum um sistema de criação sonora.

3.3.1 *Hipertexto da Corrupção*

Hipertexto da Corrupção é um sistema de *web-art*[88] criado em 2017, que realiza uma composição audiovisual a partir de buscas na internet por vídeos com a palavra-chave "corrupção".

Imagem 52. Frame do obra Hipertexto da Corrupção.

O desenvolvimento do *software* com o sistema *Three.js* permite que a obra seja disponibilizada em formato dinâmico na internet, permitindo sua execução de forma única a cada acesso à página. A obra foi exibida na exposição *IMMERSPHERE International Fulldome Festival Computer Art Exhibition*, realizada no Planetário de Brasília, em 2017.

Podemos observar aqui uma realização do conceito de *remix* regenerativo, cunhado pelo pesquisador de novas mídias Eduardo Navas, "que consiste na justaposição de dois ou mais elementos constantemente atualizados, o que significa que eles são criados para se modificarem de acordo com o fluxo de dados." (NAVAS, 2009, p.8)

---

[87] *Site* do projeto *openStreetMap*: < www.openstreetmap.org >
[88] Página da obra *Hipertexto da Corrupção*, com versões dinâmica e em vídeo: < www.alexandrerangel.art.br/hipertextodacorrupcao.html >

3.3.2 *Diálogo entre a Bíblia e o Tao*

Diálogo entre a Bíblia e o Tao é um sistema dinâmico em forma de *website*[89] programado com a linguagem *JavaScript*. Dois agentes digitais interagem entre si por meio de voz sintetizada e reconhecimento de voz. O resultado ora faz sentido, ora é completamente absurdo, em uma visão de que as religiões falam as mesmas coisas, mas de formas diferentes. Internamente, estão esforçando-se para conversar dois algoritmos: a biblioteca de síntese de fala e a biblioteca de compreensão de fala, ambas disponibilizadas pelo Google. A obra segue uma estética minimalista com foco no conteúdo conceitual. Esse caminho visual foi seguido novamente na obra *Emoção Artificial*, em 2018.

Levando em consideração os quesitos de potencialidades técnicas e criativas, facilidade de desenvolvimento e facilidade de preservação / manutenção, foi escolhida para esse projeto a linguagem de programação *JavaScript*[90].

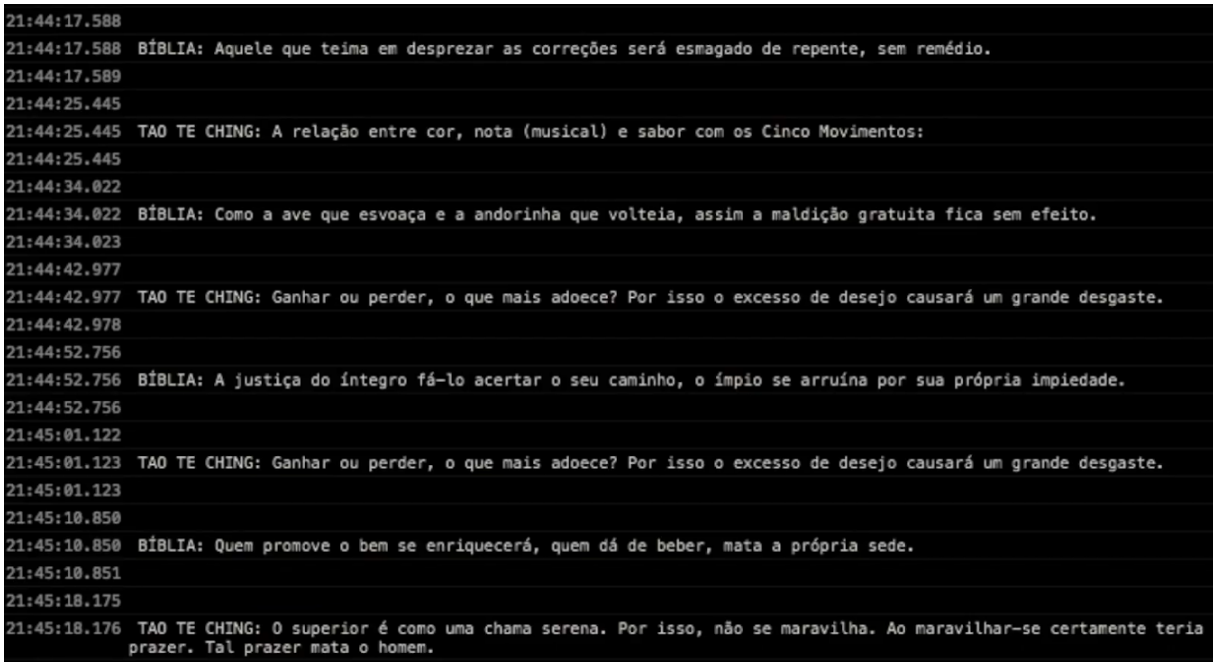


Imagem 53. Frame da obra Diálogo entre a Bíblia e o Tao.

---

[89] Registro em vídeo da obra *Diálogo entre a Bíblia e o Tao*: < www.youtube.com/watch?v=QooZ41ZXIlQ >

[90] História da linguagem *JavaScript:* < https://pt.wikipedia.org/wiki/JavaScript >

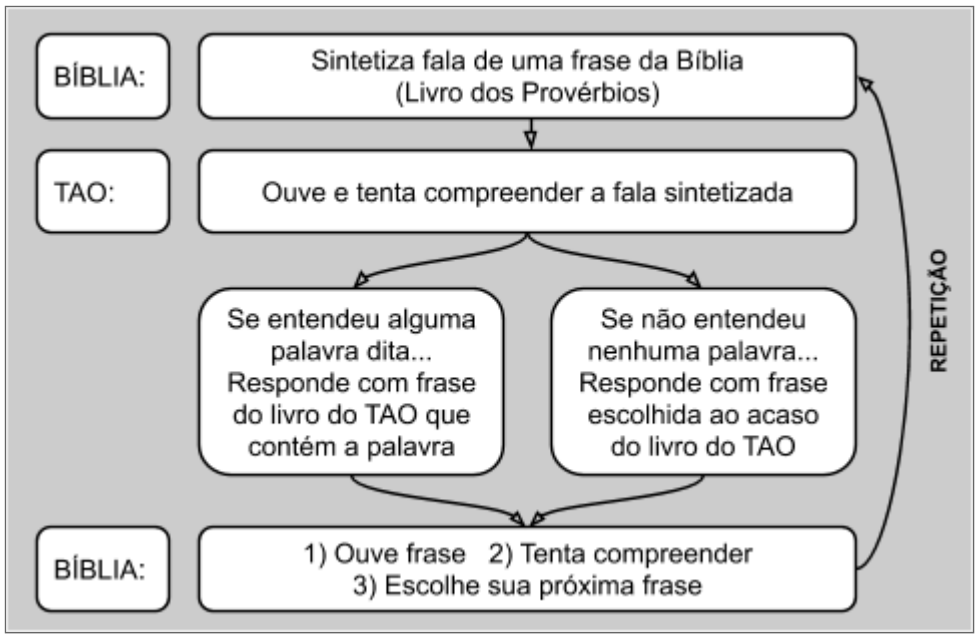


Gráfico 8. Esquema técnico da obra Diálogo entre a Bíblia e o Tao.

3.4 *VDMX / ISF*

O *VDMX* [91] é um *software* comercial de composição visual em tempo real. Ao lado do próprio *Quase-Cinema*, o *VDMX* é o meu *software* de escolha para a parte visual de performances audiovisuais. Apesar de funcionar somente no sistema macOS, ele também é uma das minhas escolhas para oficinas de capacitação em produção audiovisual. Dentre os *softwares* profissionais de criação visual, o *VDMX* é o que possui a licença de uso mais generosa, ou permissiva: pode-se usar o *software* completo no modo de teste, gratuitamente, perdendo-se somente a função de salvar o projeto sendo trabalhado.

A empresa desenvolvedora do *VDMX*, a americana Vidvox, desenvolveu um formato/linguagem de programação visual chamado ISF (*Interactive Shader Format*). O *ISF* é compatível com o padrão de programação visual *GLSL* (*Open Graphics Language Shading Format*). A combinação *VDMX / ISF* foi um dos primeiros sistemas que experimentei para criação de gráficos com a execução imediata característica de sistemas de *live coding*.

Na figura a seguir podemos ver o *VDMX* criando o componente visual da minha performance de *live coding MeditaMáquina* (2013). O papel do *VDMX* na performance era receber sinais *MIDI* do *software ixi lang*, e de acordo com as notas recebidas ativar clipes de vídeo e controlar quais efeitos e outros parâmetros visuais seriam aplicados. Dessa forma, a parte visual da apresentação, principalmente os cortes de imagem, era muito bem

[91] *Site* do *software VDMX*: < http://vidvox.net >

sincronizada com os eventos musicais engatilhados pela linguagem ixi lang. Uma vez iniciada a execução da composição pelo sistema *ixi lang*, o *VDMX* tem responde de forma automatizada de acordo com as regras estabelecidas. Além dessas ações pré-programadas, o *VDMX* podia ser controlado manualmente, trocando os bancos de imagem de acordo com os trechos mais amplos da performance. Para essa segunda camada de interação, foi ligado um controlador *MIDI* a um pequeno equipamento físico com botões mapeados em pontos-chave do *VDMX*. Outro método de controle rápido e confiável é o teclado com algumas funções de controle de velocidade de reprodução dos vídeos e com o importante controle de troca de sentido, ou seja, de fazer um vídeo tocar de frente para trás ou de trás para frente. É uma função que costumo mapear na barra de espaço do teclado, o que permite uma interação física mais energética com a máquina.

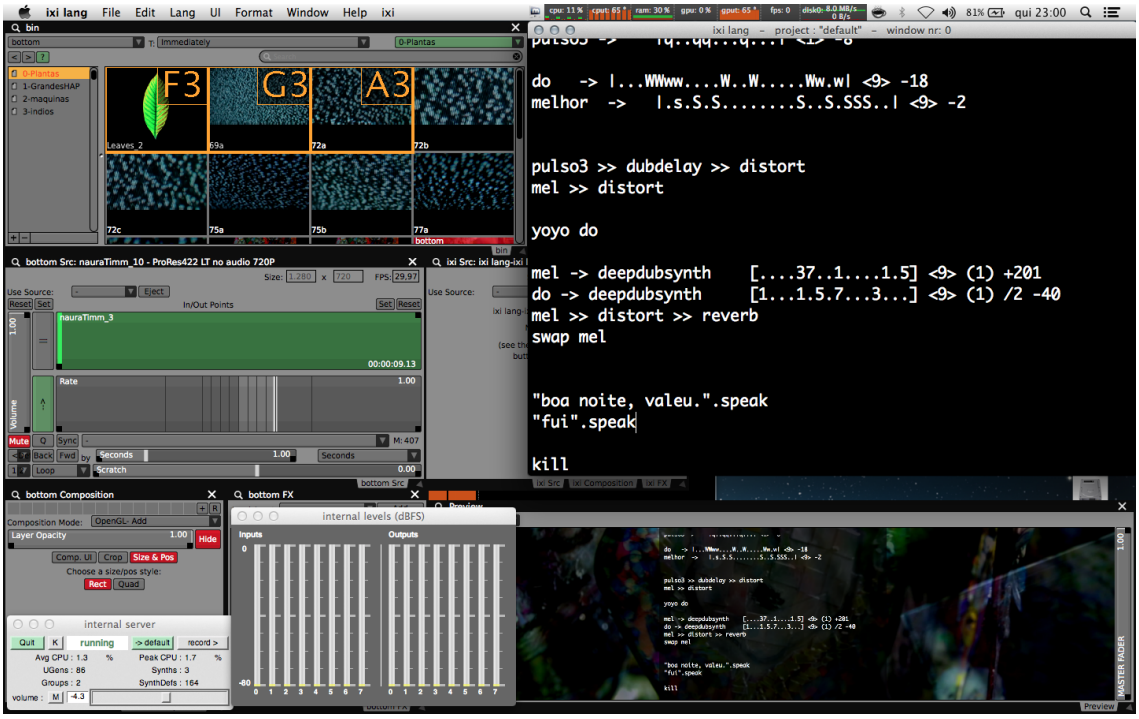


Imagem 54. VDMX gerenciando visual da performance MeditaMáquina.

Uma função de construção multimídia fantástica presente no *VDMX* é a possibilidade de capturar o conteúdo visual de janelas de qualquer outro programa, mesmo que esse programa não seja preparados para isso. Isto é, o *VDMX* pode incluir nas suas composições visuais uma janela de um navegador da internet com qualquer conteúdo ou no caso dessa performance, pode capturar a janela de programação do sistema *ixi lang* e misturá-la em tempo real a meus vídeos pré-selecionados para performance. Na imagem acima, que retrata a interface do *VDMX*, podemos ver a configuração de saída de três monitores, cada um com um pedaço da composição final dos vídeos e a interface do *ixi lang*

sobreposta, com transparência e fundos controlados para dar maior visibilidade à performance textual de *live coding*.

3.4.1 *muSEU feito na unha*

Ao ser convidado para participar da exposição *SEU muSEU (2013)*, com curadoria de Wagner Barja, criei a obra *muSEU feito na unha*[92]. O tema da exposição foi o a a ideia de pertencimento mútuo entre o Museu Nacional da República e a população de Brasília. A videoescultura é constituída de uma placa de madeirite, com um furo no formato do Museu e de um sistema que capta a imagem do público e aplica-a em uma televisão posicionada atrás da máscara do Museu. A idéia foi dar uma sensação de pertencimento mútuo entre o público e a instituição.

O processo de captura de vídeo por meio de *webcam* foi realizado com a aplicação de efeitos visuais em tempo real, programada no sistema *VDMX*. O som da obra, que se transformava generativamente durante o dia, foi programado no sistema Ableton Live, com uso da funcionalidade de ativação de diferentes trechos de camadas sonoras em ordem aleatória, criando combinações e mixagens inesperadas dentro do tema sonoro da obra.

A busca e objetivo durante a programação da obra foi atingir um equilíbrio entre a quantidade de efeitos e a capacidade do público de se reconhecer na obra e – conceitualmente – no Museu. Apliquei uma quantidade tal de efeitos de distorção que tornava necessário certo tempo de observação da imagem para que as pessoas pudessem reconhecer suas silhuetas ou faces. Esse era o momento de *eureka!* do público: "Ah, sou eu!"

---

[92] Registro da interação do público com a obra *muSEU feito na unha:*
< www.youtube.com/watch?v=0go0Bf7wgmo >

Imagem 55. Foto da obra muSEU feito na unha.

3.5 Unity

Software comercial muito popular para o desenvolvimento de jogos e experiências interativas, lançado em 2005, o *Unity* é direcionado para o desenvolvimento de jogos ou outros formatos de aplicativos multimídia com gráficos 2D ou 3D. Possui uma versão para uso gratuito (*Personal Version*), cuja única diferença é a apresentação da logomarca do Unity no início da execução de cada projeto.

A forma de programação principal no ambiente do *Unity* é a criação de código textual na linguagem *C#*, porém o *Unity* possui poderosas ferramentas de criação gráfica, incluindo um editor de sistemas gráficos por meio de fluxograma.

3.5.1 *Memórias Corrompidas*

Sistema computacional[93] de exploração lúdica de um espaço virtual, sonoro e visual, com síntese e *remix* de representações espaciais tridimensionais, *Memórias Corrompidas* foi criado parceria com o artista e programador Arthur Cordeiro.

A obra apresenta um universo virtual para a exploração por parte do público. Ao fundo, ouvimos trechos de missas em latim distorcidos por meio de efeitos de áudio e

[93] Página da obra *Memórias Corrompidas:*
< www.alexandrerangel.art.br/memoriascorrompidas.html >

de reprodução lenta granulada. A dinâmica da exploração do espaço tridimensional segue a linha de jogo contemplativo, ou seja, uma aventura de descobrimento sem a tensão da competição ou até mesmo da procura por alcançar algum objetivo predeterminado.

Imagem 56. Instalação da obra Memórias Corrompidas (foto: Diego Bressani).

Para digitalização (*scan*) de cenários tridimensionais foi utilizado o sensor de profundidade com tecnologia de digitalização com raios infravermelhos *Structure Sensor* [94]. O *Structure* é um *scanner* muito prático que funciona acoplado a um *tablet* do tipo *Apple iPad*.

[94] *Site* do *scanner Structure:* < https://structure.io >

Imagem 57. Scanner 3D Structure Sensor (foto: divulgação).

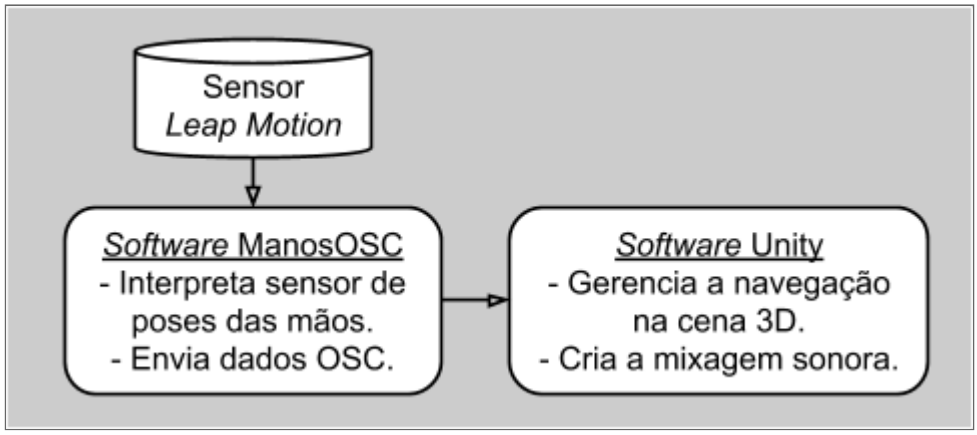


Gráfico 9. Esquema técnico da obra Memórias Corrompidas.

Um desdobramento dessa obra será a programação de um sistema de mixagem dinâmica de poesia. O sistema reproduziria os áudios com efeitos de espacialização tridimensional e trechos de poesias gravadas por artistas parceiros, nas localizações reais dos cenários 3D virtuais. Os poetas Antônio Miranda, Gérson Deveras e Sid Magnus são convidados para interações futuras com o sistema.

A obra *Memórias Corrompidas* foi exibida na exposição *A/RISCADO: Arte, Ciência e Tecnologia*, realizada no Museu Nacional da República, em Brasília, durante os meses de dezembro de 2018 e janeiro de 2019, com curadoria de Wagner Barja e Gilberto Lacerda. A obra foi apresentada no formato de projeção interativa em meio domo, construído com MDF, cortado a laser e montado por meio de encaixes, expandindo ainda mais os conceitos de imersão e construção de um todo a partir de partes interconectadas.

3.6 Outras ferramentas de criação visual

Outras ferramentas de criação visual notáveis de menção no contexto de criação de obras dinâmica e programação de regras de comportamento interativo.

3.6.1 *Hydra*

O *Hydra* é um ambiente de programação visual interativo baseado em programação textual. Consiste em duas versões: uma para execução na internet[95], outra para funcionar diretamente do computador "local". Sistema de criação visual colaborativa em rede desenvolvido pela artista americana, baseada na Colômbia, Olivia Jack[96], funciona por meio da internet ou localmente, em sistemas desktop (*Linux*, *macOS* e *Windows*). Integra, na mesma janela do navegador, o código sendo digitado ao resultado visual do algoritmo.

Como projeto precursor ao Hydra, a artista criou o sistema *Parche* (expressão em espanhol que significa, na Colômbia, um grupo de amigos) em uma residência artística, no Espaço Platoedro, em Medelim, na Colômbia, em 2017.

Imagem 58. Interface do sistema Hydra, funcionando via internet.

---

[95] Versão *online* do sistema *Hydra:* < https://hydra-editor-v1.glitch.me >

[96] *Site* da artista Olivia Jack: < https://ojack.github.io >

3.6.2 *Unreal Engine*

Software comercial para a criação de aplicativos 3D, lançado em 1998, o *Unreal Engine* é um sistema gratuito para uso pessoal, desde uma mudança no seu licenciamento realizada em 2015. O modelo de negócios da *Unreal Engine* baseia-se em uma cobrança de 5% do lucro de cada projeto desenvolvido com o *software*, caso o projeto venda mais que 3 mil dólares. Tecnicamente, o *Unreal* é conhecido pela sua capacidade de criação de gráficos 3D avançados e pela sua forma de programação, que é usualmente feita por conexões de fluxograma.

A Epic Games, desenvolvedora da *Unreal Engine*, é uma empresa de mentalidade aberta com relação ao *software* livre (*open source*). Todo o *software* é disponibilizado na forma de código-fonte, permitindo seu estudo e modificação. Sobre o envolvimento da empresa com o público e o mercado: Epic Mega Grants são doações de recursos financeiros em forma de doações, sem compromisso comercial algum, para empresas de áreas acadêmicas e de desenvolvimento de *software* livre, tais como a Fundação Blender, agraciada com uma doação de 1,2 milhão de dólares em julho de 2019[97]. Por meio da iniciativa lançada em março de 2019, a Epic distribuirá 100 milhões de dólares para criadores de conteúdo 3D, estudantes, profissionais, educadores e desenvolvedores de ferramentas com produção relacionada ao Unreal ou relacionada à melhoria de ferramentas de *software* livre para a comunidade de criação 3D. Existe a perspectiva futura das equipes de desenvolvimento de *software* da Epic Games trabalharem em novas funções e melhorias para o Blender voltadas à produção de obras interativas.

[97] Notícia disponível em: <www.blender.org/press/epic-games-supports-blender-foundation-with-1-2-million-epic-megagrant >

Capítulo 4. Tornando o código tátil – e etéreo

"No tempo da magia, um elemento explica o outro."
Vilém Flusser

Tempo e interatividade são dois conceitos complementares na arte computacional. Com a interatividade, chamamos a atenção ao tempo de resposta, de diálogo. "A programação é uma prática de melhoria cibernética da mente, na qual o programador usa de uma linguagem simbólica para entrar num sistema de trocas com o computador – com a finalidade de expandir suas capacidades de imaginação, pensamento, previsão, exploração de hipóteses, etc." (CARVER, 2019) Na interatividade, a gramática da criação dá-se em aberto, esperando por um clique ou por um movimento do público. Ressalto a necessidade de reflexão sobre a hibridização entre o pensar e o executar audiovisual, e sobre a dicotomia entre a certeza e o acaso. Enxergo a interatividade como elo entre o tempo criação e o tempo de fruição da obra.

Meu primeiro contato com sistemas de programação de computadores foi em 1986, quando tinha 12 anos de idade. A lógica que aprendi na época, o fluxo de um *software* e os comandos básicos de programação são a base de tudo que desenvolvo nesta área desde então. O aprendizado era feito com a linguagem *BASIC*[98], disponível na maioria dos computadores pessoais da época, como o *Apple II* que eu possuía[99]. Minha trajetória de capacitação de uso de linguagens de computador acabou sendo um excelente estudo de caso, uma vez que já se passaram 30 anos do meu contato inicial com essa área e ainda continuo pesquisando a área e praticando o desenvolvimento – criativo – de *software*. Durante esses anos me apaixonei pelos ambientes de programação *Macromedia Director* (1994), *Blender Game Engine* (2004) e *Processing* (2010), graças ao potencial dessas ferramentas de transformar ideias em sistemas. Todos esses programas foram criados e direcionados para o uso por profissionais de campos criativos (áudio, vídeo, 3D, interatividade), sem o pré-requisito de uma formação acadêmica em informática. Mesmo

---

[98] Página com história da linguagem *BASIC:* < https://pt.wikipedia.org/wiki/BASIC >

[99] Um *TK-3000 IIe*, clone brasileiro do *Apple IIe* que era ligado à televisão, não possuía mouse e gravava os programas em fita cassete! Uma vantagem dos computadores *Apple II* caseiros da época, se comparados aos caros *IBM-PC* empresariais, era a possibilidade de visualização de cores na tela.

sendo voltados principalmente para o público vindo das ciências humanas, nenhum desses ambientes se compara com a simplicidade inicial do *Sonic Pi*, criado para ser usado por crianças da faixa etária em torno dos 8 anos.

A programação de *software* é o elo que permite a delimitação e o controle do tempo, por parte do artista e por parte do receptor / interator. "A estética que a música pode ensinar aos artistas visuais é a do tempo." (DEWITT, p.118)

4.1 *openFrameworks*

O ambiente de programação openFrameworks está para a linguagem *C* assim como o *Processing* está para o *Java*. Ambos facilitam a realização de programação de multimídia por artistas sem capacitação prévia necessária em desenvolvimento de *software*, valendo-se de projetos de exemplo, comunidades e fóruns de discussão ativos.

O openFrameworks, diferentemente do ambiente de criação Processing, não possui um editor de código próprio, funcionando normalmente como biblioteca dentro dos ambientes de programação *XCode* (*macOS*) e *CodeBlocks* (*Windows* e *Linux*).

4.1.1 *Quase-Cinema*

Vislumbrando a concretização da mutação dos meios audiovisuais prevista por Gene Youngblood em seu livro *Cinema Expandido*, de 1970, onde começa o capítulo *Curto Circuito Global: A Terra como Software*, com a frase "Assim como cada fato é metafísico, toda máquina implica um software: a informação sobre a sua existência." Cérebro eletrônico, expandido. Se a máquina é análoga ao corpo, o *software* é análogo à alma.

Por que criar um *software*? O *software* pronto é massificador, nivelador de produções. Fazendo uma comparação com a pintura, imaginemos que a indústria criativa determinasse o formato das telas, ou as cores que poderiam ser usadas pelos pintores. Quando criei a primeira versão do sistema *Quase-Cinema,* meu objetivo era me libertar dos procedimentos sugeridos pelas ferramentas presentes no mercado daquela época, sendo elas soluções em *software* livre ou *software* comercial. Procurei inclusive não conhecer outros programas com objetivos semelhantes, para não condicionar o meu desenvolvimento das ideias realizadas previamente. Sabia que meu objetivo era a criação de um *software* de edição de vídeo dinâmico que permitisse a criação de edições, com ritmo e texturas de acordo com a intenção do criador. Desde a primeira versão do *software,* a recompensa foi uma sensação muito libertadora de poder criar cada botão e cada

funcionalidade de acordo com o objetivo de criação poética, de criação artística. Era um código se transformando em poesia.

Imagem 59. Detalhe da interface do software Quase-Cinema 2.

Sobre o processo de criação e apreensão dos sistemas e da cultura de *remix* e sua releitura, Christine Mello chama a atenção para "mudanças culturais perceptivas no nosso modo de captar a instância sensória, já que cada época e cada cultura produz um organismo com aptidões perceptivas diferentes." (MELLO, p.101).

O desenvolvimento do *software Quase-Cinema* acompanhou a evolução das ferramentas de criação de *software* multimídia, assim como a minha habilidade e familiaridade com os sistemas de desenvolvimento de obras audiovisuais, principalmente no tocante à habilidade de reproduzir e remixar (editar, misturar, compor) vídeos.

"Roy Ascott acredita que a arte é um sistema que transforma o comportamento e a consciência, assumindo então que o princípio cibernético do retorno (*feedback*) tem uma dimensão estética relacionada com o papel do observador, que está em transformação de um simples *voyeur* para um participante ativo. O usuário torna-se engajado pelas escolhas disponibilizadas e é então 'empoderado' ao invés de ser manipulado." (FERNEDING, p.7)

Versão	Tecnologia de desenvolvimento
Quase-Cinema 1	Desenvolvido no ambiente Macromedia Director, com a sua própria linguagem, o Lingo.
Quase-Cinema 2	Desenvolvido no ambiente openFrameworks / C++.
Quase-Cinema Feijoada	Desenvolvido no ambiente Processing / Java.

Tabela 18. Tecnologias de desenvolvimento do software Quase-Cinema.

Usando o openFrameworks, na época que estava desenvolvendo a versão 2 do *Quase-Cinema*, programei em alguns dias uma função de pintura com luz[100], durante oficina no *Festival Liquid Borderland Art and Music Festival* [101], realizado em 2009, em Aalborg, na Dinamarca. Participei do evento como convidado, fazendo performance de projeção, oficina de projeção de vídeo com *Quase-Cinema,* e a imprevista programação do sistema de pintura com luz integrado ao *software Quase-Cinema 2*.

A função de pintura com luz foi o ponto de partida para a criação da performance *Éon / Aeon*[102], em parceria com o artista plástico Wagner Hermusche e o músico Ramiro Galas, realizada na Galeria Fortress to Solitude, Nova York, em 2010, sob a curadoria do argentino Guillermo Creus. Nessa obra, realizei desenhos com elementos luminosos, editando o vídeo ao vivo com gravações prévias de desenhos de Hermusche e combinando o resultado visual das camadas com o *software Quase-Cinema 2*.

[100] Vídeo de demonstração do sistema de pintura com luz:
< www.youtube.com/watch?v=BaHwYxh7vWs >
[101] Vídeo de registro do *Festival Liquid Borderland Art and Music:*
< www.youtube.com/watch?v=TY1KSqLm69A >
[102] Vídeo de registro da performance *Éon / Aeon:*
< www.youtube.com/watch?v=uFLLuxSRFZ4 >

Imagem 60. Fotografia da performance Éon / Aeon.

Oficinas *Quase-Cinema*

A convite do pesquisador acadêmico e agitador cultural taiwanês Ilya Eric Lee, foram realizadas duas oficinas do sistema *Quase-Cinema* em Taiwan[103], em 2010 e 2012, com foco no uso e no estudo de perspectivas de desenvolvimento continuado do sistema *Quase-Cinema*. Comuniquei-me com os participantes em inglês com a ajuda de dois tradutores de chinês, que assistiram os alunos que não falavam inglês.

A oficina realizada na Universidade Nacional de Artes de Taipei[104], em 2012, seguiu uma programação didática direcionada ao grupo de participantes, em maioria jovens de 17 a 22 anos, alunos dos cursos de Cinema, Artes e Comunicação. As atividades foram realizadas na seguinte ordem:

1) Palestra sobre as origens da técnica de VJ e suas raízes no Cinema Experimental e na Videoarte; foram apresentados os artistas Bill Viola (exemplificando seu ritmo de edição, em peças de longa duração) e Maya Deren (mostrando seu trabalho de filmagem com movimentações de câmera, como que em um dança);

2) Apresentação do *software Quase-Cinema*, contextualizando sua produção e obras derivadas no contexto da Arte Contemporânea;

3) Apresentação de materiais preexistentes dos participantes da oficina;

---

[103] Vídeo de registro da oficina *Audiovisual Art Movement*, realizada na Universidade Nacional de Artes de Taipei, Taiwan, 2012: < www.youtube.com/watch?v=pUfNCLLvG2M >

[104] *Site* da *Taipei National University of the Arts:* < https://w3.tnua.edu.tw >

4) Saída em campo para gravação de material audiovisual e demonstração de técnicas de uso de câmera em movimento, câmera na mão e uso de planos cinematográficos tradicionais (*zoom*, *pan*, *travelling* etc);

5) Atividades em sala de aula, apresentando conceitos e técnicas de montagem de imagens em movimento e de conversão de imagens para performance ao vivo;

6) Projeção coletiva em formato de festa, com música dançante, ressaltando o aspecto de obra aberta que estava sendo criada; para a projeção foram usados 8 projetores de vídeo e 4 computadores conectados simultaneamente e misturados com a mesa de edição *Edirol V-4*.

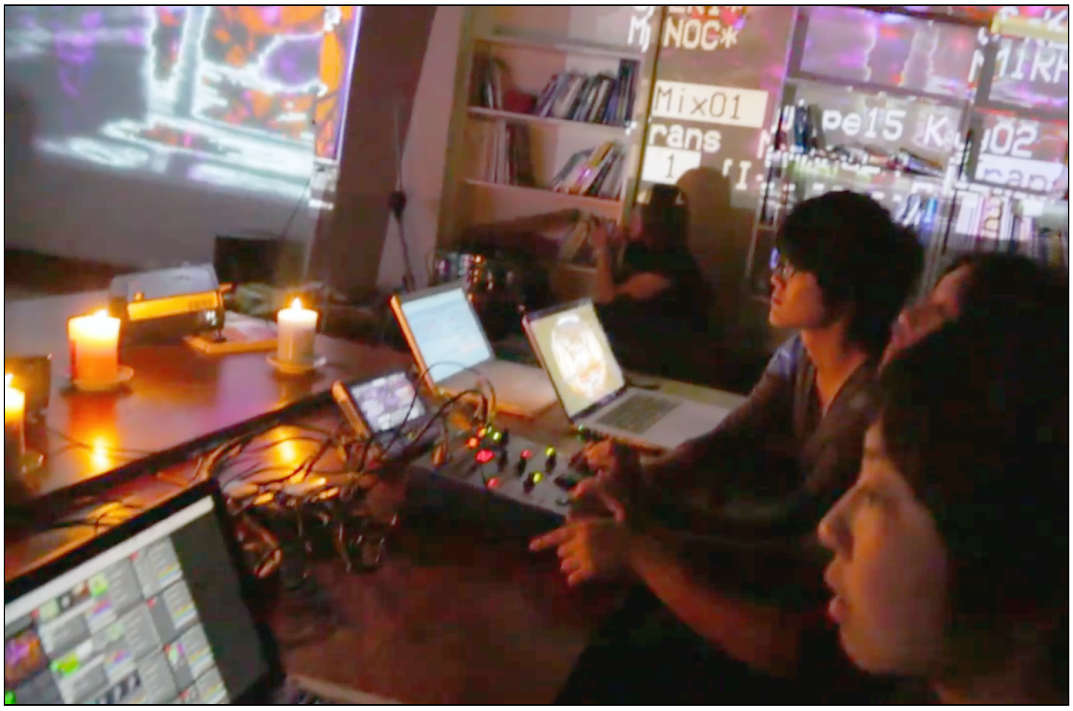


Imagem 61. Performance coletiva dos participantes da oficina.

Imagem 62. Cartaz da oficina Quase-Cinema em Taiwan (2012).

Por meio de uso das possibilidades criativas do programa e debates a respeito, chegamos a uma lista de desejos de funcionalidades futuras do *software Quase-Cinema*:

Interface

- Interface escalonável – ajuste do tamanho da interface ao tamanho do monitor do computador do usuário;

- Interface com cores customizáveis – útil para uso em ambientes com iluminações diferentes;
- Texto da interface em português e inglês – possibilidade de tradução da interface pelos usuários.
- Interface tridimensional de busca, edição e composição de imagens, mesclando os conceitos de interface de criação e produto final.

Conexões

- Conexão com outros *softwares* (via protocolo *MIDI* e/ou *OSC*), tais como programas de síntese sonora (*Sonic Pi*, *SunVox*, *Ableton Live*), o que permite que os parâmetros visuais criem composições sonoras e vice-versa.

Efeitos de imagem

- Utilização de efeitos de imagens com processamento na placa gráfica do computador – filtros programados com a linguagem GLSL (*Open Graphics Library Shading Language*[105]).

Áudio

- Reprodutor de áudio nos formatos MP3 e WAV;
- Análise de áudio reproduzido pelo sistema;
- Análise de som ambiente via microfone;
- Conexão das análises de áudio com parâmetros visuais variados.

Video mapping

- Possibilidade de distorção individual das camadas de vídeo para ajuste em superfícies de projeção irregulares (não retangulares);
- Controle de mapeamento em superfícies orgânicas por meio de manipulação de curvas (para corte e/ou distorção de camadas de imagens).

Registro de performances

- Possibilidade de gravação de arquivos audiovisuais com o resultado de uma performance;
- Possibilidade de transmissão ao vivo pela internet.

---

[105] Referência da linguagem OpenGL:
< https://en.wikipedia.org/wiki/OpenGL_Shading_Language >

4.2 Controladores físicos

Experiências e desenvolvimentos com controladores físicos: um dos desafios artísticos que me tenho proposto durante os anos mais recentes de produção é a possibilidade de tornar o código tátil, ou seja, manipulável com expressividade gestual e corporal. Para isso, venho pesquisando e desenvolvendo interfaces de comunicação entre sensores, controladores musicais e dispositivos de jogos, e o código propriamente dito. Procuro maneiras de fazer gestos controlarem o conteúdo de variáveis sonoras e visuais, de modo que se integrem como ferramentas composicionais e de performance criativa.

Experiência de fazer o sistema *Sonic Pi* controlar sintetizadores externos, tais como os pequenos *Korg Volca Beats* e o *Korg Volca Bass* (medindo 15 x 10 x 3 centímetros cada um), as programações rítmicas e melódicas executadas pelo *Sonic Pi* podem ser sintetizadas por esses sequenciadores / sintetizadores. Um grande diferencial desse procedimento de criação sonora é a possibilidade de manusear os parâmetros sonoros por meio dos controles físicos presentes nas máquinas. Podem-se alterar vários parâmetros ao mesmo tempo; tarefa difícil de se fazer com o *mouse*.

Imagem 63. O sequenciador de bateria Volca Beats e o sintetizador de baixo Volca Bass.

4.2.1 *Traquitana Audiovisual Interativa*

Desenvolvi um sistema de performance audiovisual, um ambiente criativo com o intuito de poder fazer performances dinâmicas audiovisuais, principalmente do material criado para o projeto Weekly Beats 2014, chamado *Traquitana Visual Interativa (TAI*).

Como material de base que motivou a criação da sistema *TAI*, eu acabara de criar as 52 composições audiovisuais criadas em 2014, cada uma contendo as trilhas de áudio e de vídeo separadas – ideal para a criação de versões derivadas e misturas.

Visando a simplificação visual e operacional do sistema, realizei inicialmente uma simplificação da mixagem das músicas. Cada música do projeto continha inicialmente de 4 a 16 canais de áudio. Sem nenhuma modificação na sonoridade das músicas, agrupei os canais semelhantes em 4 canais (dois grupos de elementos percussivos, um grupo de texturas sonoras, e um grupo para os sons de sintetizadores).

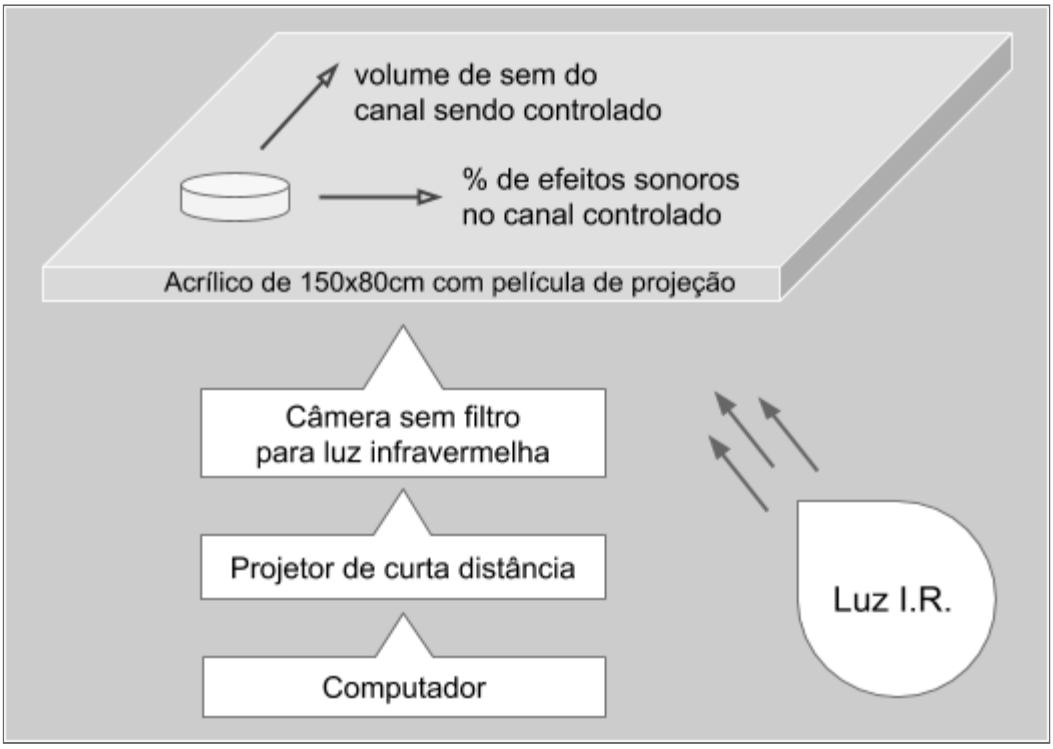


Gráfico 10. Esquema da estrutura física do sistema TAI.

Cada disco de acrílico controlava um dos quatro canais sonoros; a posição vertical de cada cilindro configurava o volume da cada canal.

Figura 64. Performance com o sistema TAI.

Tecnicamente, o sistema todo funciona baseado em código escrito no ambiente Processing, o que mostra mais uma vez sua versatilidade e potencial criativo.

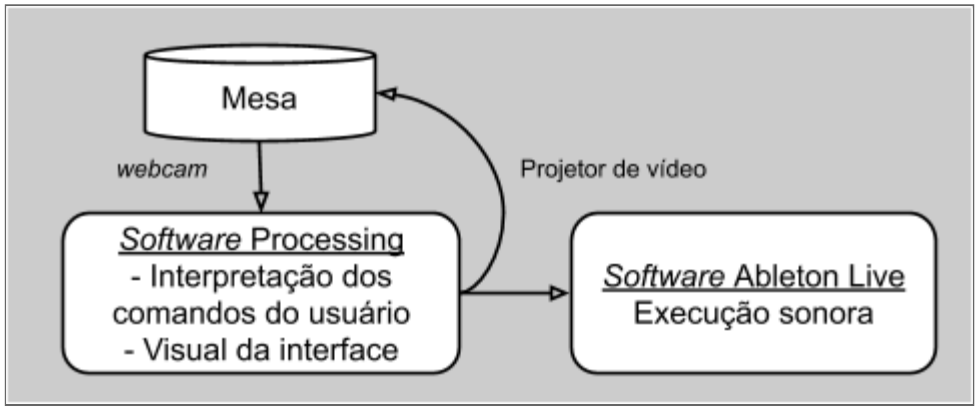


Gráfico 11. Esquema técnico do sistema TAI.

4.2.2 Controladores *MIDI*

O controlador *MIDI Faderfox UC-3* tem 8 *knobs* (controles rotatórios) e 9 *faders* (controles deslizantes verticais). Fabricado pela empresa alemã Faderfox[106], conecta-se ao computador via USB e permite o controle de *softwares* que reconhecem o protocolo de comunicação MIDI, como o *Sonic Pi* e o *Blender* – no caso do *Blender*, utilizando o *software*

[106] *Site* de empresa FaderFox, criadora de controladores *MIDI* especializados: < www.faderfox.de >

adicional AddMIDI. Cabe ao usuário de cada programa especificar quais parâmetros estarão vinculados a cada controle. É um equipamento muito prático, devido ao seu pequeno tamanho de 18 x 10 x 7 cm.

O equipamento *Edirol V-4* é, na realidade, uma mesa de edição de vídeo de quatro canais que, enquanto edita sinais de vídeo, também envia comandos *MIDI* via *USB*. Isso permite certa integração entre as práticas de edição de vídeo em tempo real (VJ) e *live coding* musical. Foi criado pela empresa japonesa Roland[107].

Imagem 65. Os controladores MIDI Faderfox UC-3 e Edirol V-4.

Experiência *Code Push* no evento *Campus Party Brasília*

Também realizei oficinas de criação musical com programação de *software* nos eventos *Campus Party* e *Campus Day* (uma prévia reduzida do evento principal). O *Campus Party* é um dos maiores encontros de tecnologia da América Latina, que reúne milhares de estudantes e palestrantes em uma semana de apresentações e acampamento.

[107] *Site* da empresa de equipamentos multimídia Roland: < www.roland.com/global >

Imagem 66. Entrevista para a Rede Globo durante a Campus Party Brasília 2016.

Os participantes do evento puderam ter uma experiência física de manipulação de trechos de código-fonte usando os botões dos equipamentos de DJ. Além de apresentar o projeto de oficinas presenciais de experimentação e de ensino, conduzi demonstrações da integração do sistema de programação *Sonic Pi* ao equipamento de produção musical profissional: o controlador musical *MIDI Ableton Push 2*.

Cada botão do controlador mandava uma mensagem MIDI para o *Sonic Pi*. O *Sonic Pi* então esperava o fim do compasso e trocava o trecho de programação sendo executado. Botões nas duas inferiores da controladora funcionavam como uma pequena partitura (ou uma linha do tempo) para sequenciamento de notas musicais ou gravações de trechos de bateria eletrônica.

Este experimento mostrou-se extremamente eficaz ao conseguir entusiasmar os participantes a controlarem pedacinhos de código por meio de botões. Esse sistema, que fundiu um controlador físico e alguns blocos de código é, na realidade, uma grande simplificação de todas as interações que temos com sistemas digitais: uma interface com botões e códigos que são executados conforme a demanda do usuário. Entretanto, essa simplificação e musicalização do processo mostrou-se lúdica e misteriosa, iniciando várias conversas sobre potencialidades de sistemas táteis de acionamento de programas. Nas palavras da repórter Bruna Roma, “Agora todo mundo pode ser DJ!” – DJ de código . . .

Imagem 67. Controlador MIDI Ableton Push 2. (foto: divulgação)

4.3 Sensor corporal *Kinect*

Uma busca por fluxos de criação utilizando equipamentos sem conexões cabeadas. Uma das referências primordiais que tenho de música eletrônica e de interfaces sem fio – um aspecto importante para uma interação com fluidez – foi ter visto vídeos de registro do músico francês Jean Michel Jarre tocando sua harpa de raio laser[108] nos anos 1980. A partir dos anos 2000, dispositivos de jogos têm-se tornado técnicas financeiramente acessíveis para produções artísticas.

O *Microsoft Kinect* é um sensor de posicionamento corporal que permite o envio de dados via cabo USB para *softwares* habilitados compatíveis, tais como o *Processing*. O sensor foi desenvolvido originalmente para o console de jogos *Xbox*. A partir do *Processing*, pode-se desenvolver a integração com vários *softwares* por meio do envio de mensagens *MIDI* ou *OSC*, tais como *Blender* e *Sonic Pi*.

Os sensores *Kinect* contam com duas câmeras: uma câmera normal, que capta a cor de cada ponto (*pixel*) e uma câmera de raios infravermelhos, que pode captar a distância de cada ponto. Graças à informação de distância (profundidade), pode-se facilmente isolar um elemento do fundo da imagem, permitindo assim seu acompanhamento no espaço tridimensional. Existem duas versões do *Kinect:* a versão 2

[108] Registro em vídeo de performance de Jean Michel Jarre com sua *Harpa Laser:* < https://youtu.be/Fiw2KlGf3hA?t=213 >

possui mais resolução de imagem que a versão 1 e permite o acompanhamento (*tracking*) de mais usuários simultaneamente. A área de captação dos sensores *Kinect* é de cerca de 9 metros quadrados. Os equipamentos *Kinect* não são mais fabricados atualmente. Seu possível sucessor no mercado de sensores 3D é o equipamento similar *Intel RealSense*.

Imagem 68. Sensores Microsoft Kinect 1 e Kinect 2.

Uma das primeiras obras que desenvolvi usando o *Kinect* foi o sistema *corpo-orquestra*[109], em parceria com o artista Luiz Oliviéri. Nesse trabalho é evidente a presença do corpo como agente de interação com a máquina, que aqui é invisível[110]. Aqui, o esforço foi para tornar o computador invisível e a interface transparente, a fim de estreitar os laços emocionais entre o participante e o conteúdo sonoro. Estava dada a capacidade de misturar e remixar a música com a dança, com o corpo livre no espaço. O que se deu na experiência foi uma interessante retroalimentação entre os movimentos corporais e a percepção sonora do resultado de cada ação. Era o corpo controlando a música, e subitamente de volta ao padrão do corpo responder à música.

---

[109] Desenvolvimento da obra *corpo-orquestra:* < www.youtube.com/watch?v=YMxC2V7CGp8 >

[110] Exibição da obra *corpo-orquestra* no *12º Fórum Internacional do Software Livre*, em Porto Alegre-RS, 2011: < www.youtube.com/watch?v=JYIjt7Wf3Lc >

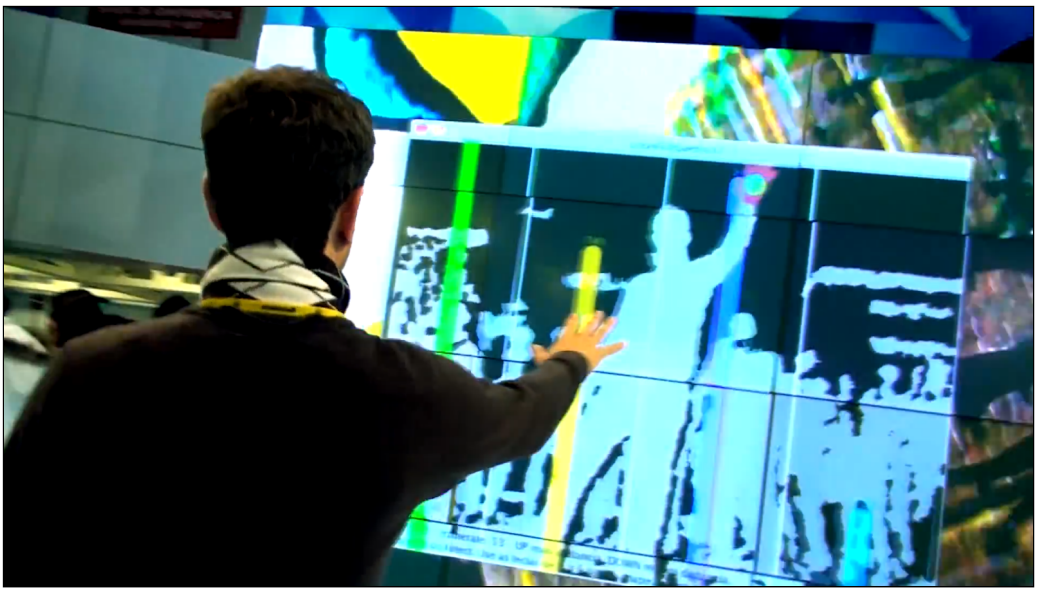


Imagem 69. Interação corporal, uma dança com a obra corpo-orquestra.

Outra obra que se valeu da tecnologia do *Kinect* foi a instalação interativa *Esculturas Quânticas*, na qual a posição do corpo e as posições dos braços são analisadas e enviadas, via *OSC*, para controlar o visual e o som em tempo real.

4.3.1 *Eixo X*

Como última obra citada, retorno à obra *Eixo X*, um sistema computacional em forma de instalação interativa, realizada em parceria com o artista chileno Rodrigo Paglieri. A obra foi exposta nas exposições: *Brasília Síntese das Artes*, com curadoria de Denise Mattar, no Centro Cultural Banco do Brasil de Brasília, e *Tékhne*[111], no Museu da Arte Brasileira da FAAP, sob curadoria de Denise Mattar e Christine Mello, em São Paulo; ambas em 2010.

O sistema, por meio de uma *webcam* embutida no meio de uma parede, capta a posição do público na sala e, a partir disso, rotaciona dois vídeos de Brasília, das Avenidas Eixo Monumental e Eixo Rodoviário, formando graficamente o traçado de um "X". Esse X foi pensado como uma metáfora de uma utopia não realizada, de centro urbano proibido para pessoas, só acessível para carros.

---

[111] *Site* e vídeo de registro da exposição Tékhne:
< www.faap.br/hotsites/memorias_reveladas_tekhne/tekhne.asp > e
< www.youtube.com/watch?v=tL5IiB_OkRY >

Imagem 70. Interação corporal com a obra Eixo X.

"Sempre mudamos a paisagem quando nos deslocamos nela?" – pergunta a crítica de arte Marília Panitz, em texto sobre o trabalho no catálogo da exposição (MATTAR). O mistério faz-se presente na pergunta e na resposta, respira com o interagir com o programa.

Esta foi a minha primeira obra com a técnica de sensoriamento do público. O sistema realiza uma detecção facial das pessoas presentes na sala, dentro do campo de visão da câmera. A pessoa mais perto da câmera (ou a pessoa que tiver a cabeça maior) assume o controle da interação ao mover-se à frente da câmera.

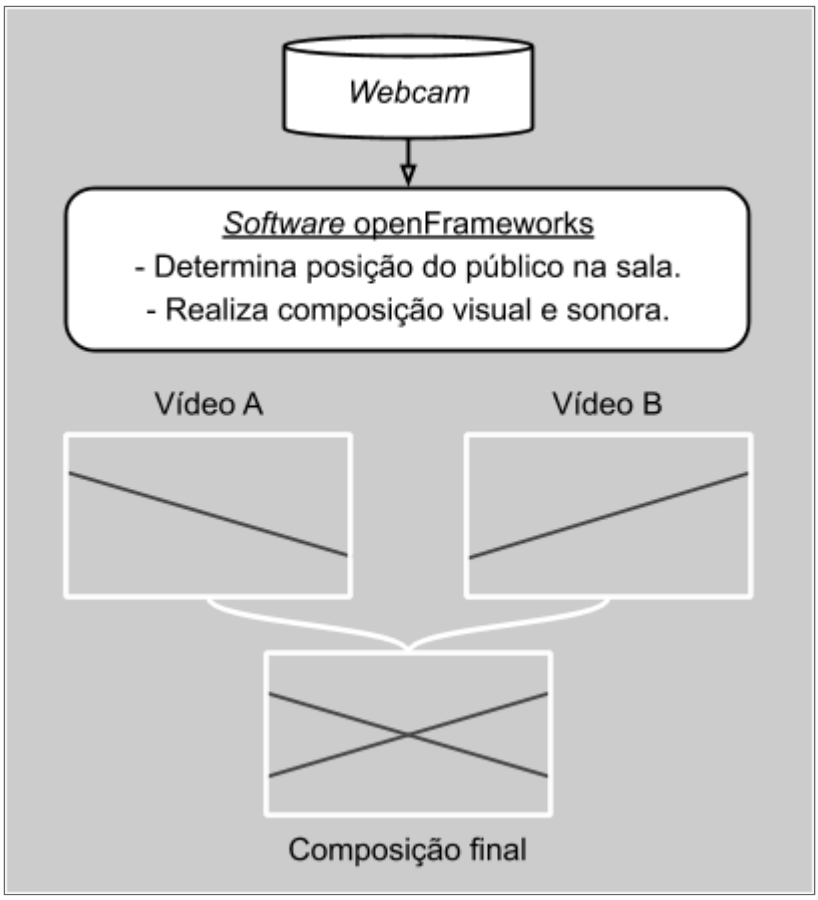


Gráfico 12. Esquema técnico da obra Eixo X.

4.4 Sensor manual *Leap Motion*

Leap Motion é um sensor de profundidade com reconhecimento de gestos das mãos e dos dedos. Criado pela empresa americana *Leap Motion*[112], consegue examinar um volume de cerca de meio metro cúbico, usando duas câmeras e um emissor de raios infravermelhos. Nesse espaço, ele consegue identificar até duas mãos simultaneamente, conseguindo precisar a posição e rotação das mãos, assim como a rotação de cada articulação de cada dedo, permitindo a execução de controles com grande velocidade e precisão gestual.

Pela minha experiência de desenvolvimento, um dos programas mais confiáveis para se obterem dados de leitura do *Leap Motion* e enviá-los para outro *software* é o Manos OSC[113]. Utilizei o sistema *Manos OSC* na instalação interativa *Memórias Corrompidas*, como base técnica da interação gestual com o público. Já na obra *Bichos*

---

[112] *Site* da empresa *Leap Motion:* < www.leapmotion.com >

[113] Página do programa *Manos OSC:*
< https://lm-s-apps-amnesia.leapmotion.com/apps/manososc/osx >

Impossíveis, usei o sensor Leap Motion para controlar o sistema desenvolvido no ambiente Processing.

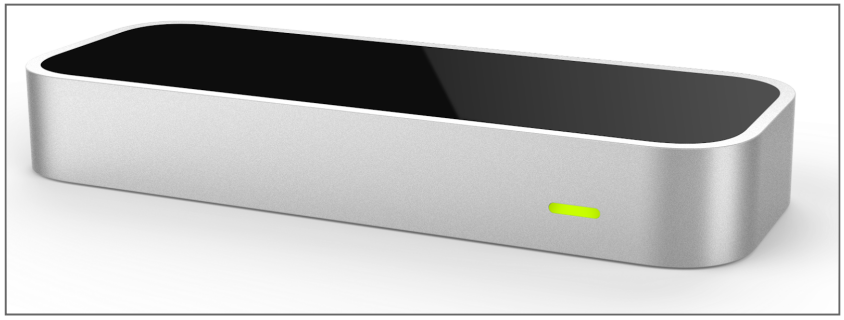


Imagem 71. O sensor para percepção de mãos Leap Motion (7 x 3 x 2 cm).

4.4.1 *Bichos Impossíveis*

A obra *Bichos Impossíveis*[114] é um sistema interativo que dialoga com a série de esculturas da artista brasileira Lygia Clark. Os *Bichos* de Clark eram interativos, compostos de peças metálicas com dobraduras que permitiam várias configurações geométricas. Hoje, quando expostas em museus, podem raramente ser manuseadas, devido a questões de conservação.

Criei um sistema computacional que permitia o controle de peças inspiradas nos *Bichos*, porém com possibilidades de movimentação fisicamente impossíveis – como a interpenetração entre as peças – daí o título *Bichos Impossíveis*. Uma versão da obra foi exibida na exposição *ENTRECOPAS*, em 2014, no Museu Nacional da República, com curadoria de Wagner Barja.

[114] Vídeo de registro de interação com a obra *Bichos Impossíveis:*
< www.youtube.com/watch?v=a67ExTPhfjY >

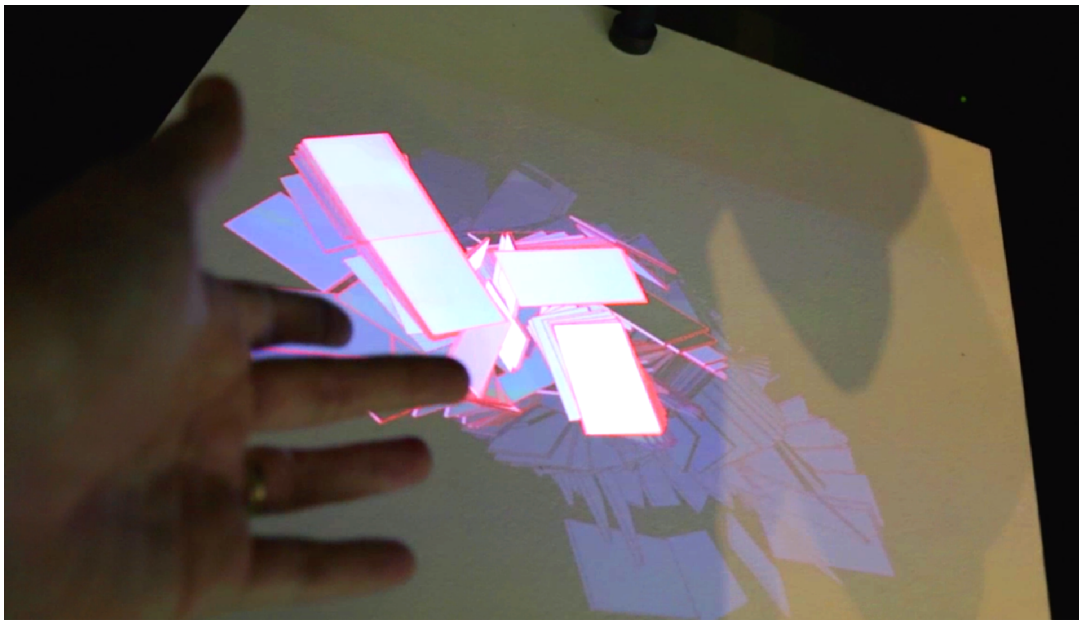


Imagem 72. Interação manual com a obra Bichos Impossíveis.

Este capítulo descreveu ferramentas de *hardware* e *software* que podem ser utilizadas, sozinhas ou em conjunto, para o desenvolvimento de obras audiovisuais generativas baseadas em algoritmos computacionais.

Foi mostrada ênfase nas vantagens da pesquisa e no uso continuado de ferramentas baseadas em *software* livre, por questões de longevidade e custo de manutenção dos projetos de desenvolvimento de arte computacional.

Considerações finais

O desenvolvimento continuado, desde 2009, de várias versões do *software* livre *Quase-Cinema* possibilitou a compreensão do processo de criação de um ambiente de criação digital completo, desde o desenho da interface até a programação das regras e do código-fonte do programa. A primeira interação desses sistemas computacionais é a interação entre a máquina, que precisa reconhecer os comandos humanos, e o artista criador, que nela se reconhece.

Sobre a fusão entre teoria e prática artística, o compositor Iannis Xenakis indaga, no prefácio da versão estendida do seu livro *Música Formalizada*, se as proposições teóricas por ele postuladas nos trinta e cinco anos anteriores teriam sobrevivido em sua música, e se foram esteticamente eficientes. Essa questão ficou em aberto, mas é válida como uma máxima gnóstica "Conhece-te a ti mesmo"[115] , valiosa a qualquer artista que ambicione o crescimento por meio do conhecimento da sua produção – e consequentemente, de si mesmo.

A pesquisa descreveu meus esforços para a criação de obras artísticas baseadas em sistemas computacionais dinâmicos. As principais questões levantadas foram: 1) a mesclagem entre os paradigmas de partitura visual e de código-fonte como partitura; 2) o conceito de obra aberta em relação com o ato de compor como musicar, o de criar como brincar; e 3) a atividade de programar como expansão das possibilidades do pensamento humano.

Um desafio das etapas de transmissão de conhecimento do trabalho foi lidar com o medo generalizado do contato com as entranhas do computador. "Indício de uma relação ambígua do século XX com a tecnologia das máquinas é a espera, por um lado, de que a tecnologia venha a libertar a humanidade dos trabalhos serviçais, e por outro lado, o temor ao enorme potencial da tecnologia para controlar a vida e os valores humanos." (GARNETT, p.21)

---

[115] "Conhece-te a ti mesmo" é uma das 147 máximas délficas, inscritas no Templo de Apolo, em Delfos, na Grécia. A Suda, enciclopédia grega do século X, interpreta o provérbio como direcionado àqueles que tentam ir para além do que são – uma advertência ou um convite à transformação?

A sociedade já foi muito condicionada, desde o advento da televisão – com sua comunicação de mão única –, a compreender os equipamentos eletrônicos telemáticos como uma forma de consumo de conteúdo e experiências predeterminadas, pasteurizadas.

O que definiu a produção artística associada? Experiências no domínio virtual da criação de *software* com conexões improváveis e modo de pensamento e condução de experiências do campo das artes; tentativa e erro, análise, curadoria e ensino. Dispositivo como obra aberta. O público, o criador e o aprendiz misturam seus papéis durante os processos. Em outra camada de possibilidades, o espectador cria o seu próprio caminho narrativo e a sua própria experiência audiovisual. Considero que a quantidade de controle e a quantidade de surpresas presentes na experiência são partes constituintes do mais importante ponto focal de inserção poética e de intervenção na obra de arte. "A partir do hipertexto, toda a leitura tornou-se um ato de escrita." (LÉVY, p.46)

Dentre as contribuições da pesquisa, ressalto também os programas desenvolvidos durante o projeto e a disponibilização de programas como *softwares* livres para uso e criação de derivações. Dentre as inovações apresentadas, destaco as experiências de conexões entre programas, tais como *Processing*, *Blender* e *Sonic Pi*.

Percebi e exemplifiquei, por meio do detalhamento de processos criativos, a possibilidade de fluidez no processo de desenvolvimento e experimentação (ao vivo ou não) com o código, tanto como um instrumento quanto como uma extensão dos processos cerebrais de criação de caminhos lógicos. Uma das principais conclusões, observando o detalhamento das obras trazidas para a pesquisa, é o poder da invenção, do inesperado, do incompreendido, e do mistério. "Essa mistura particular entre significado e mistério age como uma fonte recompensadora de interação humano-maquínica; a própria ideia da interação é estendida até uma experiência profunda de estética mediada pela máquina." (BEYLS, p.11) Considero que aqui está o ponto-chave, o ponto de inserção poética, de destilação do diálogo entre homem e máquina.

Vislumbro possibilidades de aprofundamento nas seguintes áreas de estudo e produção:

1) *Blender*: exploração continuada de criação por meio de fluxogramas (*Sverchok* e *Animation Nodes*) e por meio da linguagem *Python* embutida no *Blender*;

2) Sonic Pi: novas técnicas de programação (integração com a linguagem *ixi lang)*;

3) Criação de novas versões do ambiente de criação *Quase-Cinema*, incorporando novas ideias e o *feedback* dos participantes das oficinas presenciais.

A contribuição desta tese foi a apresentação de experiências de criação audiovisual que abraçam a objetividade e a subjetividade de forma complementar ao invés de antagônica, mostrando o fazer artístico com detalhamento original.

Apresentei caminhos de produção com ênfase em interconexões entre técnicas, suportes e ferramentas. Muitas outras possibilidades abrem-se agora, dada uma abertura técnica e conceitual da caixa-preta do artista.

Faço minhas as palavras de Miroslav Milovic, professor do Programa de Pós-Graduação em Metafísica da UnB, sobre a subjetividade: “Reinventar o pensamento, reinventar o jeito como a gente vê o mundo e, claro, pensar as alternativas. Como os cristãos falam: o novo no mundo. Então vamos ver se ainda é possível ser resistente, criticar, criar algo *novo*. Ainda essa palavra me fascina”. (MILOVIC, 2013)

Após realizar esta pesquisa, pude compreender com mais clareza a utilidade dos processos de aprendizado e sua transmissão pelo ensino e colaboração em torno de projetos de *software* livre, com vistas na concretização do potencial da cultura como agente de transformação.

Referências Bibliográficas

AARON, Samuel et al. ***Temporal Semantics for a Live Coding Language. Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional art, music, modeling & design***. p. 37-47. Gothenburg: 2014.

ASSIS, Gustavo Oliveira Alfaix. **Em Busca do Som: A Música de Karlheinz Stockhausen nos Anos 1950**. São Paulo: Editora UNESP, 2011.

BASTOS, Marcus. **Remix como polifonia e agenciamentos coletivos**. Territórios Recombinantes: Arte e tecnologia / Debates e Laboratórios. Camila Duprat Martins et al (org.). São Paulo: Imprensa Oficial do Estado de São Paulo, 2007.

BERRY, David M. (ed.). **Life in Code and Software: Mediated Life in a Complex Computational Ecology**. Open Humanities Press: 2012.

BEYLS, Peter. **Autonomy, Influence and Emergence in an Audiovisual Ecosystem**. Proceedings of the XV Generative Arts Conference. Lucca, Italy, 2012

BLACKWELL, Alan; COLLINS, Nick. **The Programming Language as a Musical Instrument**. Sussex University: Proceedings of Psychology of Programming Interest Group, 2005.

BOGOST, Ian. **Procedural Literacy: Problem Solving with Programming, Systems & Play**. Journal of Media Literacy, v. 52, n.1, 2005.

BROWN, Andrew R. **Code Jamming**. Media Culture Journal. Queensland University of Technology, 2006.

_____. **Generative Music in Live Performance**. Media Culture Journal. Queensland University of Technology, 2006.

BROWN, Andrew R.; SORENSEN, Andrew. **Interacting with generative music through live coding**. Queensland University of Technology e Australasian CRC for Interaction Design, 2009.

CAGE, John. **Notations**. Nova York: Something Else Press, 1969.

_____. **Silence: Lectures and Writings by John Cage**. 1961.

CADENA, Francisco Alfonso Alba. **Generating music by fractals and grammars**. Dissertação (Mestrado em Ciências Aplicadas) - San Luis Potosí: Universidad Autónoma de San Luis Potosí, Facultad de Ciencias, 2001.

CAPRA, Fritjof. **O Tao da Física: Um Paralelo Entre a Física Moderna e o Misticismo Oriental**. São Paulo: Editora Cultrix: 1995.

CÁRDENAS, Alexandra. **Live Coding: A New Approach to Musical Composition**. Dancecult: Journal of Electronic Dance Music Culture, v. 10, n. 1. 2018.

CARVALHO, Rodrigo Guerreiro Vaz Guedes de. **S+V+M: Relationships Between the Sound, Visual and Movement Domains in Interactive Systems**. Tese (Doutorado em Mídia Digital) - Porto: Universidade do Porto, 2018.

COLLINS, Nick. **Algorithmic Composition Methods for Breakbeat Science**. ARiADA, n. 3, 2003.

_____. **Live Coding and Machine Listening**. Proceedings of First International Conference on Live Coding. 2015.

_____. **Live Coding of Consequence**. Leonardo, v. 44, n. 3, p. 207-211, 2011.

COLTON, Simon e WIGGINS, Geraint A. **Computational Creativity: The Final Frontier?** Proceedings of the 20th European Conference on Artificial Intelligence. Amsterdam: IOS Press, p. 21-26, 2012.

COPE, David. **Experiments in Music Intelligence**. Santa Cruz: University of California, 1987.

COX, Donna J. **The Tao of Postmodernism: Computer Art, Scientific Visualization and Other Paradoxes**. Leonardo Supplemental Issue, v. 2, Computer Art in Context: SIGGRAPH '89 Art Show Catalog, p. 7-12, 1989.

COX, Geoff et al. **The Aesthetics of Generative Code**. Journal Proc. of Generative Art, 2001.

COX, Geoff. **What Does Live Coding Know?**

DEMERS, Joanna. **Listening Through The Noise: The Aesthetics of Experimental Electronic Music**. Nova York: Oxford University Press, 2010.

DEWITT, Tom. **Visual Music: Searching for an Aesthetic**. Leonardo, 1987, v. 20, n. 2, p. 115-122.

DOMINGUES, Diana. **A Caixa de Pandora e as tramas da Vida nas rede telemáticas.** in Arte e Tecnologia na Cultura Contemporânea. Brasília: Dupligráfica Editora Ltda, 2002.

_____. **A Humanização das Tecnologias pela Arte**. In: DOMINGUES, Diana (Org.). A Arte no Século XXI: a humanização das tecnologias. São Paulo: Editora UNESP, 2003.

ENO, Brian e SCHIMIDT, Peter. **Oblique Strategies: Over One Hundred Worthwhile Dilemmas**. 1975.

FERNÁNDEZ, Jose David e FICO, Francisco. **AI Methods in Algorithmic Composition**. Journal of Artificial Intelligence Research, n. 48, p. 513-582, 2013.

FERNEDING, Karen. **Embracing the Telematic: A Techno-Utopian Vision of Art and Pedagogy for the Post-Human Age of Control**. Journal of the American Association for the Advancement of Curriculum Studies, v.1, 2005.

FISCHER, Michael M. J. e DUMIT, Joseph. **Sharing Source Code** em Two Bits: The Cultural Significance of Free Software. Duke University Press, 2008.

FONT, Frederic et al. **Freesound technical demo**. Proceedings of the 21st ACM international conference on Multimedia. Barcelona: ACM, 2013.

FULLER, Matthew, ed. **Software Studies**. MIT Press, 2008.

GHISI, Daniele. **Music Across Music: Towards a Corpus-Based, Interactive Computer-Aided Composition**. (Tese de Doutorado em Pesquisa Musical e Composição) - Paris: Université Pierre et Marie Curie / Université Paris-Sorbonne, 2017.

GARNETT, Guy E. **The Aesthetics of Interactive Computer Music**. Computer Music Journal, v. 25, n. 1, p. 21-33. The MIT Press: 2001.

GOULART, Antonio José Homsi; ANTAR, Miguel Diaz. **Live Coding the computer as part of a free improvisation orchestra of acoustic instruments**. Proceedings of First International Conference on Live Coding. 2015.

HAMMEN, John. **Bitscript: A domain-specific scripting language for interactive music**. Proceedings of the 17th Linux Audio Conference, Stanford University, 2019.

HARLEY, James. **The Creative Compositional Legacy of Iannis Xenakis**. Definitive Proceedings of the “International Symposium Iannis Xenakis”, Atenas: 2005.

HOCHENBAUM, Jordan et al. **ChucK Racks: Text-based Music Programming For The Digital Audio Workstation**. 2017.

HOFFMANN, Peter. **Music Out of Nothing? A Rigorous Approach to Algorithmic Composition by Iannis Xenakis**. Berlim: 2009.

HORA, Daniel De Souza Neves, **Arte Hackeamento: Diferença, Dissenso E Reprogramabilidade Tecnológica**. Dissertação (Mestrado em Arte) - Brasília: Universidade de Brasília, 2010.

HUTCHINS, Charles Céleste. **Live Patch / Live Code**.

INCE, Can. **Programming For Music: Explorations in Abstraction**. Dissertação (Mestrado de Arte em Música) - Huddersfield: University of Huddersfield, 2018.

JÄRVELÄINEN, Hanna. **Algorithmic Musical Composition**. Helsinki: 2000.

KOENIG, Gottfried. **The Use of Computer Programmes in Creating Music**. UNESCO Music and Technology Meeting, Estocolmo, 1970.

LANGSTON, Peter S. **Six Techniques for Algorithmic Music Compositio**n. 15th International Computer Music Conference, 1989.

LAWSON, Shawn. **Performative Code: Strategies for Live Coding Graphics**.

LEVIN, Thomas Y. **Tones from out of Nowhere: Rudolph Pfenninger and the Archaeology of Synthetic Sound**. Massachusetts Institute of Technology, Grey Room n. 12, p. 32-79, 2003.

LEWITT, Sol. **Paragraphs on Conceptual Art**. Revista Artforum, junho de 1967.

MAKELA, Mia. **Live Cinema: Language and Elements**. Dissertação (Mestrado Novas Mídias) - Helsinki: Helsinki University of Art and Design, 2006.

MAGNUSSON, Thor. **Algorithms as Scores: Coding Live Music**. Leonardo Music Journal, vol. 21, 2011.

_____. **Herding Cats: Observing Live Coding in the Wild**. Computer Music Journal, n. 38, vol. 1, p. 8-16. Massachusetts Institute of Technology, 2014.

_____. **Code Scores in Live Coding Practice**. Brigthon: University of Sussex.

_____. **ixi lang: A Constraint System for Live Coding**. ISEA2010 RUHR Conference Proceedings. 2010.

_____. **The Ixi Lang: A Supercollider Parasite For Live Coding**. Proceedings of the International Computer Music Conference. 2011.

_____. **Processor Art: Currents in the Process Oriented Works of Generative and Software Art**. Tese (Doutorado). 2002.

_____. **Scoring with Code: Composing with Algorithmic Notation**. Organised Sound, n. 19, p. 268-275, 2014.

_____. **Sonic Writing: Technologies of Material, Symbolic, and Signal Inscriptions**. Bloomsbury Academic. Nova York: 2019.

MANOVICH, Lev. **Avant-garde as Software**.

_____. **Media After Software**. 2012.

MATTAR, Denise. **Brasília Síntese das Artes**. Catálogo da Exposição no Centro Cultural Banco do Brasil Brasília. Brasília: 2010.

MCLEAN, Alex. **Artist-Programmers and Programming Languages for the Arts**. Tese (Doutorado) - Londres: Goldsmiths University of London, 2011.

_____. **Reflections on live coding collaboration**.

MCLEAN, Alex; WIGGINS, Geraint. **Texture: Visual Notation for Live Coding**. Londres: Goldsmiths, University of London.

MEDEIROS, Maria Beatriz de. **Introdução** in Arte e Tecnologia na Cultura Contemporânea. Brasília: Dupligráfica Editora Ltda, 2002.

MELLO, Christine. **Extremidades do Vídeo**. Editora SENAC. São Paulo: 2008.

MENDES, Daniel de Souza. **O Cálculo E A Invenção Na Poética De Stockhausen**. Dissertação (Mestrado em Música). 2009.

MOONEY, James. **Hugh Davies's Electroacoustic Musical Instruments and their Relation to Present-Day Live Coding Practice: Some Historic Precedents and Similarities**. 2015.

NAUR, Peter. **Programming as Theory Building**. Selected Writings From 1951 To 1990 - ACM Press/Addison-Wesley, 1992.

NAVAS, Eduardo. **Regressive and Reflexive Mashups in Sampling Culture** in SONVILLA-WEISS, Stefan (Ed.) Mashup Cultures. Nova York: Springer Wien, 2010.

NEAL, Alexander Blair. **A Visualist's Practice: Explorations of Interfaces and Exercises for Live Video Performance**. Dissertação (Mestrado em Artes) - Nova York: Rensselaer Polytechnic Institute, 2010.

NILSON, Click. **Six Live Coding Works for Ensemble**. Proceedings of the 7th International Conference on New interfaces for Musical Expression, p. 112-117. Nova York, 2007.

OLDENBURG, Aaron. **Sonic Mechanics: Audio as Gameplay**. The International Journal of Computer Game Research, v. 13, n. 1, 2013.

OLMO, F. Javier Ruiz-del et al. **Creación sonora y nuevas tendencias artísticas en el siglo XXI: Algoritmos, música electrónica y Algorave**. Ediciones Complutense, 2018.

PARKINSON, Adam e BELL, Renick. **Deadmau5, Derek Bailey, and the Laptop Instrument: Improvisation, Composition, and Liveness in Live Coding**. Proceedings of First International Conference on Live Coding, 2005.

PATTESON, Thomas. **Instruments for New Music: Sound, Technology, and Modernism**. Oakland: University of California Press, 2016.

PAYLING, David. **Visual Music Composition with Electronic Sound and Video**. Tese (Doutorado em Tecnologia Musical) - Stoke-on-Trent: Staffordshire University, 2014.

PORCARO, Nicholas J.; LEVY, Ellen. **Blendnik: A Real-Time Performance System Using Blender and Pure Data**. 2009.

PRÓSPERO, Carolina Di. **Live coding. Arte computacional en proceso**. Contenido. Arte, Cultura y Ciencias Sociales, n. 5. Buenos Aires: Univetrsidad de San Martín / IDAES, 2015.

PUIG, Sergi Jordà. **Digital Lutherie: Crafting musical computers for new musics' performance and improvisation**. Tese (Doutorado em Informática e Comunicação Digital) - Barcelona: Universitat Pompeu Fabra, 2005.

REICH, Steve. **Music as a gradual process**. 1968.

ROBERTS, Charles; WAKEFIELD, Graham. **Live coding the digital audio workstation**.

ROBERTS, Charles et al. **Gibber: Abstractions for Creative Multimedia Programming**.

_____. **Music Programming in Gibber**. International Computer Music Conference, 2015.

ROBERTS, Charles et al. **Sharing Time and Code in a Browser-Based Live Coding Environment**.

RODDY, Stephen. **Ambient Data Monitoring w/ Generative Music Systems using EC & ML Techniques**. Proceedings of the 3rd Computer Simulation of Musical Creativity Conference, 2018.

RUSHKOFF, Douglas. **Program or Be Programmed: Ten Commands for a Digital Age**. Nova York: OR Books, 2010.

RUSSOLO, Luigi. **The Art of Noise**. Something Else Press, 1967.

SANTAELLA, Lucia. **O corpo biocibernético e o advento do pós-humano** in Arte e Tecnologia na Cultura Contemporânea. Brasília: Dupligráfica Editora Ltda, 2002.

SANT'ANNA, Hugo C. et al. **Da Arte Generativa Ao Pensamento Computacional: Uma análise comparativa das plataformas de aprendizagem**.

SCHAFFER, Pierre. **Music and Computers**. UNESCO Music and Technology Meeting, Estocolmo, 1970.

SHANKEN, Edward A. **Art in the Information Age: Technology and Conceptual Art**. Leonardo, v. 35, n. 4, p. 433-438, 2002.

_____. **Technology and Intuition: A Love Story? Roy Ascott's Telematic Embrace**. Leonardo, v. 30, n. 1, p. 60, 1997.

SODDU, Celestino. **AI Organic Complexity in Generative Art**. Proceedings of the XXI Generative Art International Conference. Roma: Domus Argenia, 2018.

SORENSEN, Andrew. **Impromptu: An interactive programming environment for composition and performance**. Australia.

SORENSEN, Andrew e GARDNER, Henry. **Programming With Time: Cyber-physical programming with Impromptu**. Proceedings of OOPSLA10: ACM International Conference on Object Oriented Programming Systems Languages and Applications, Nova York, p. 822-834, 2010.

SPONT, Marya Helen. **Looking Beyond the Visual: Considering Multi-Sensory Experience and Education with Video Art in Installation**. Dissertação (Mestrado em Artes) - Austin: The University of Texas, 2016.

SRIDHARAN, Harini et al. **Computational models for experiences in the arts, and multimedia**. Proceedings of the 2003 ACM SIGMM Workshop on Experiential Telepresence, p. 31-44, Berkeley, 2003.

STEWART, Jeremy; LAWSON, Shawn. **Cibo: An Autonomous TidalCycles Performer**.

STOWELL, Dan; MCLEAN, Alex. **Live music-making: a rich open task requires a rich open interface**.

SUPPER, Martin. **A Few Remarks on Algorithmic Composition**. Computer Music Journal, v. 25, n. 1, p. 48-53, 2001.

SUSLOV, Nikolai; SOSHENINA, Tatiana. **From Live Coding to Virtual Being**. Proceedings of First International Conference on Live Coding. Leeds: 2015.

SZLIFIRSKI, Krzysztof. **New Technology And The Training Of Composers In Experimental Music**. 1970.

TEIXEIRA, Paulo e BERNARDES, G. **The Online Musical Database in/as Performance**. Hidden Archives, Hidden Practices: Debates About Music-Making. Aveiro: 2019.

TERUGGI, Daniel. **Technology and musique concrète: the technical developments of the Groupe de Recherches Musicales and their implication in musical composition**.

TOUSSAINT, Godfried. **The Euclidean Algorithm Generates Traditional Musical Rhythms**. Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science, p. 47-56, 2005.

WAKEFIELD, Graham; ROBERTS, Charlie. **Collaborative Live-Coding Virtual Worlds with an Immersive Instrument**. Proceedings of the International Conference on New Interfaces for Musical Expression. 2014.

WANG, Ge. **The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality**. Tese (Doutorado em Ciências da Computação) - Princeton: Princeton University, 2006.

WANG, Ge; COOK, Perry R. **On-the-fly Programming: Using Code as an Expressive Musical Instrument**. Proceedings of 2004 International Conference on New Interfaces for Musical Expression. 2004.

XENAKIS, Iannis. **Formalized Music: Thought and Mathematics in Composition**. Stuyvesant: Pendragon Press, 1992.

YOUNG, La Monte; LOW, Jackson Mac. **An Anthology of Chance Operations**. Nova York: 1963.

YOUNGBLOOD, Gene. **Expanded Cinema**. Nova York: E. P. Dutton & Co, 1970.

Referências Multimídia

CARVER, Scott. **Ideomotoric Programming and Code as Psychedelic Practice**. Heretical Sound Synthesis mini-symposium, 2019. University of the Arts Helsinki, Sibelius Academy, Centre for Music & Technology.
Disponível em < www.youtube.com/watch?v=DhwmtwITjwc >

ENO, Brian. **Brian Eno: Behind The Reflection - BBC Click**. Vídeo (30'50"). Londres, BBC: 2017. Disponível em < www.youtube.com/watch?v=cv7epY75Wa0 >

KURZWEIL, Ray. **The Age of Intelligent Machines**. Vídeo (29'08"). The Massachusetts Institute of Technology Press: 1987. Disponível em: < www.youtube.com/watch?v=subiSt2Mf4Y >

LEVIN, Golan. **New-media arts inside and outside the research laboratory**. Microsoft Research Talks, 2012. Disponível em < www.microsoft.com/en-us/research/video/new-media-arts-inside-and-outside-the-research-laboratory >

LEWITT, Sol. **Untangling the puzzle of Sol LeWitt's open cubes**. Vídeo (2'42"). São Francisco: San Francisco Museum of Modern Art, 1999.
Disponível em < www.youtube.com/watch?v=w9ROCnWMPww >

LINDENMAYER, Aristid e PRUSINKIEWICZ, Przemyslaw. **The Algorithmic Beauty of Plants**. Springer-Verlag, Nova York, 1990.
Disponível em < http://algorithmicbotany.org/papers/#abop >

MANOVICH, Lev. **Algorithms of Culture?** Entrevista para a emissora Russia Today.
Disponínel em < www.youtube.com/watch?v=kaO4V1kcC7s >

MILOVIC, Miroslav e GONTIJO, Pedro. **Diálogos: subjetividade na Filosofia**. Vídeo (20'21"). UnB TV, 2013. Disponível em < www.youtube.com/watch?v=psMvrsX4pl0 >

MURPHY, James. **Making Music with Tennis Data (IBM x James Murphy)**. Vídeo (1'45"). 2015. Disponível em: < https://vimeo.com/136780902 >

VIANA, Zelito. **Ferreira Gullar - A Necessidade da Arte**. 2005. Vídeo (18').
Disponível em < www.youtube.com/watch?v=yRLDFOjxRWc >

Todos os *links* checados em 18 de setembro de 2019.

Apêndice I. Músicas do projeto Weekly Beats 2016

"O meio é a mensagem"
Marshall McLuhan

Neste anexo apresento o código-fonte para estudo e execução das 52 músicas compostas exclusivamente com o sistema *Sonic Pi*, durante o desafio *Weekly Beats 2016*.

Cada composição traz *links* para materiais multimídia, indicados por ícones, conforme mostra tabela a seguir:

	Link de *download* de arquivos de áudio MP3 das composições prontas, conforme executadas e gravadas no sistema *Sonic Pi*.
	Link de *download* de *samples* (arquivos sonoros usados como matéria prima), caso necessários para a execução dos programas no ambiente do *Sonic Pi*. Todos os *samples* são disponibilizados no banco de sons *Freesound*, normalmente em formato de áudio WAV.
	Link de visualização de vídeo da obra, caso existente (via *site YouTube*).

Tabela 19. Detalhamento das legendas do Apêndice I.

Para a leitura dos códigos-fonte a seguir: recomendo fazê-lo ouvindo cada composição associada.

Para executar as músicas no sistema *Sonic Pi* [116] sem a necessidade de modificação alguma do código-fonte, cada composição recomenda o uso do de uma versão específica do *Sonic Pi*, pois alguns comandos mudam a cada versão.

Link para audição de todas as composições:

< www.alexandrerangel.art.br/musica-wb2016.html >

[116] *Site* para *download* do *Sonic Pi:* < http://sonic-pi.net >

01. "they tried to capture the light"

Linhas **07** a **13**: Mostram um metrônomo visual (textual) na janela de texto do *Sonic Pi;*

Linhas **16** a **24**: Técnica de aplicação de efeitos sonoros em cascata (nove efeitos foram adicionados neste trecho);

Linhas **39** a **49**: Quatro variações do mesmo bumbo de bateria, cada uma com opções diferentes de velocidade de execução, modificando assim os timbres.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-they-tried-to-capture-the-light.mp3

```
01  # Alexandre rANGEL "they tried to capture the light"
02  # 4-Jan-2016 / Sonic Pi 2.9
03
04  set_volume! 0.5
05  use_bpm 120
06
07  live_loop :metronome do
08    clock = tick
09    bar = clock / 4
10    puts "bar : #{bar}"
11    puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
12    sleep 1
13  end #metronome
14  ##########################################################################################
15  live_loop :audio do
16    with_fx :compressor, amp: 1.5 do
17      with_fx :pitch_shift, pitch: [-8,-4,4,8,12].choose, window_size: rrand(0.001,0.1) do
18        with_fx :slicer, phase: (ring 0.25, 0.15, 0.25, 1.0/3)[tick] do
19          with_fx :compressor do
20            with_fx :distortion, distort: rrand(0.2,0.8) do
21              with_fx :flanger, phase: [0.1,0.25,1.0/3,0.5,1,2,4,8].choose do
22                with_fx :bitcrusher, bits: rrand_i(4,16) do
23                  with_fx :echo, phase: [2,2,2,2,4,4,4,8,16,32].choose do
24                    with_fx :echo, phase: [0.1,0.2,0.5,1,2].choose, mix: rrand(0.2,0.8) do
25                      synth :sound_in, attack: 2, sustain: 0, release: 2, amp: 0.1
26                    end
27                  end
28                end
29              end
30            end
31          end
32        end
33      end
34    end
35    sleep 4
36  end
37  ##########################################################################################
38  live_loop :kick do
39    sample :bd_haus, amp: 2, rate: rrand(0.98,1.0)
40    sleep 1
41    sample :bd_haus, amp: 2, rate: rrand(0.92,0.94)
42    sleep 1
43    sample :bd_haus, amp: 2, rate: rrand(0.92,0.96)
44    sleep 1
```

```
  if one_in(6)
    sample :bd_haus, amp: 2, rate: rrand(-0.98,-1.0)
  else
    sample :bd_haus, amp: 2, rate: rrand(0.98,1.0)
  end
  sleep 1
end
##########################################################################################
live_loop :hat do
  use_synth :pnoise
  play [:C6, :C4].choose, attack: 0.01, sustain: 0.1, release: 0.01, amp: 0.7
  use_synth :pulse
  with_fx :echo, mix: [0,0.25,0.5,1.0].choose do
    with_fx :pitch_shift, pitch: [-12,-8,-4,0,4,8,12].choose,
    window_size: [0.01,0.01,0.25,1.0/3,0.05,0.75,0.1].choose do
      with_fx :echo, mix: [0,0.5,1.0].choose do
        play [:C6, :C4].choose, attack: 0.01, sustain: 0.1, release: 0.01, amp: rrand(0.3,0.92)
      end
    end
  end
  sleep 0.5
end
```

02. "things do not exist independent of causation"

Linha **11**: Uma vez realizado o *download* do arquivo de *sample* especificado na tabela, nesta linha são indicados o nome e o caminho do arquivo baixado.

Linhas **73** a **88**: Trecho comentado, ou seja, não será executado. Os indicadores de comentário podem ser retirados para que o trecho seja executado, criando variações ao vivo ou no arranjo da composição.

Linhas **105** e **114**: Trechos executados de acordo com o andamento da música (controlado pela variável *bar*).

	www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-things-do-not-exist-independent-of-causation.mp3
	www.freesound.org/people/ngarvey/sounds/47964/dowload/47964__ngarvey__waves-on-to-rocks.aif

```
01 # Alexandre rANGEL "things do not exist independent of causation"
02 # 17-Jan-2015 / Sonic Pi 2.12
03
04 use_bpm 136
05 startClock = 0 # 0 to start song at intro
06 clock = 0; bar = 0 # global vars
07 set_volume! 1.0
08 t = Time.new
09 use_random_seed = (t.year + t.month + t.day + t.hour + t.min + t.sec)
10
11 myWave = 'C:/samples/waves.aif' # your sample path
12 load_sample myWave
13 ##########################################################################################
14 live_loop :metro do
15   cue :metro
16   sleep 0.5
17   clock = startClock + tick
18   bar = clock / 4
19   puts "bar : #{bar}"
20   puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
21 end #metro
22 ##########################################################################################
23 live_loop :track5 do #boom
24   tick
25   use_synth :growl
26   with_fx :pitch_shift, pitch: (ring -12,-12,-12,-6)[look], window_size: rrand(0.0001,0.055) do
27     play 80, detune: rrand(-100,100), sustain: (ring 1,1,1,2)[look] * 2.5,
28       pan: rrand(-0.3,0.3), pan_slide: 3, amp: rrand(3,4) * 0.8
29   end
30   sleep 3
31   with_fx :pitch_shift, pitch: (ring -24,-12,-12,-6)[look],
32     window_size: rrand(0.0001,0.01), time_dis: rrand(0.0001,0.001),
33     time_dis_slide: rrand(0.01,0.02), _slide_shape: [3,4].choose do
34     play 80, detune: rrand(-100,100), sustain: (ring 1,1,1,2)[look], amp: rrand(2,4) * 0.8
35   end
36   sleep 3
37   with_fx :pitch_shift, pitch: (ring -12,-6,0,12,24)[look],
38   window_size: rrand(0.0001,0.001), time_dis: rrand(0.0001,0.0005) do
39     play 90, detune: rrand(-100,100), sustain: (ring 1,1,1,2)[look],
```

```
      pan: rrand(-0.3,0.3), pan_slide: 3, amp: rrand(0,2) * 0.8
  end
  sleep 3
end
##########################################################################################
sleep 1
##########################################################################################
live_loop :track6 do #boom2
  use_synth :hollow
  with_fx :flanger, phase: rrand(0.01,1), mix: 0.333 do
    with_fx :echo, phase: 1.0/3 do
    with_fx :pitch_shift,pitch: (ring 16,-4,18,-8)[tick],window_size: rrand(0.005,0.025) do
    play 80, detune:rrand(-100,100), sustain:(ring 1,1,1,2)[tick], release:16, amp:rrand(1.0,1.333)
      end #fx
    end #fx
  end #fx
  sleep 1.5 * rand(10)
end
##########################################################################################
live_loop :track7 do
  with_fx :slicer, phase: ring(1,0.5,0.25,0.1)[bar] do
  with_fx :pitch_shift, pitch: (ring -16,-8,-4)[tick], window_size: rrand(0.0001,0.0005) do
      sample myWave, amp: 1.5
    end
  end
  with_fx :slicer, phase: ring(1,0.5,0.25,0.1)[bar-1] do
  with_fx :pitch_shift, pitch: (ring -4,-8,-16)[tick], window_size: rrand(0.0001,0.0004) do
      sample myWave, rate: -1, amp: 0.75
    end
  end
  sleep sample_duration myWave
end
##########################################################################################
#live_loop :track8 do #waves
#
#  with_fx :pitch_shift, pitch: (ring -16,-8,-4)[tick],
#  window_size: rrand(0.0001,0.0005) do
#    sample :mySamples__waves, amp: 1.5 * 1.25 * ([0,0,0,1].choose)
#  end
#
#  with_fx :pitch_shift, pitch: (ring -4,-8,-16)[tick],
#  window_size: rrand(0.0001,0.0004) do
#    sample :mySamples__waves,
#      rate: -1, amp: 0.75 * 1.25 * ([1,0,0,0].choose)
#  end
#
#  sleep sample_duration :mySamples__waves
#
#end
##########################################################################################
live_loop :track1 do #kick
  if bar > 24
    with_fx :bitcrusher, bits: (ring 10,10,8)[tick], mix: 0.3  do
      with_fx :flanger, mix: 0.3 do
        sample :bd_klub, amp: rrand(0.9,1.0) * 0.9
        sample :bd_haus, amp: rrand(0.9,1.0) * 0.9
      end
    end
  end #if
  sleep 1.5 / (ring 2,2,2,2,2,3)[tick]
end
```

```
101 ##########################################################################################
102 sleep 0.5
103 ##########################################################################################
104 live_loop :track2 do #tss
105   if bar > 36
106     sleep (0.5/3)
107     with_fx :flanger do
108       with_fx :slicer, phase: [0.1,0.2,0.5,0.75].choose do
109         use_synth :pnoise
110         play 120, attack: 0, sustain: 0.1, release: (ring 0.25,0.1,0.1,0.1)[tick],
111           detune: rrand(-36,22), amp: (ring 1,1.2,1,0.5)[tick] * 1.5
112       end #fx
113     end #fx
114   end #if bar
115   sleep (0.5/3) * 2
116 end
```

03. "Leo and Aquarius"

Linhas **9** e **10**: Esse trecho de código especifica uma raiz para a criação de números aleatórios para a composição baseada em um número que indica a hora de execução do programa. O número é gerado de acordo com a seguinte regra algorítmica: ano + mês + dia + hora + minuto + segundo.

Linha **26**: O *sample* está fora de qualquer loop, portanto, só será executado uma vez, quando iniciada – ou reiniciada – a composição.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-LeoAndAquarius.mp3

www.freesound.org/people/ivanbailey/sounds/41343

```
01 # Alexandre rANGEL "Leo and Aquarius"
02 # 24-Jan-2016 / Sonic Pi 2.12
03
04 use_bpm 136
05 startClock = 0 # 0 to start song at intro
06 clock = 0; bar = 0
07 set_volume! 1.0
08
09 t = Time.new
10 use_random_seed = (t.year + t.month + t.day + t.hour + t.min + t.sec)
11
12 sampleBell = '/Users/rangel/Documents/SamplesPi/bell.wav' # your sample path
13
14 ##########################################################################################
15 live_loop :metro do
16   cue :metro
17   sleep 0.5
18   clock = startClock + tick
19   bar = clock / 4
20   puts "bar : #{bar}"
21   puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
22 end #metro
23 ##########################################################################################
24 with_fx :compressor, pre_amp: 1.1, threshold: 0.5, slope_below: 1.5, mix: 0.75 do
25
26 sample sampleBell
27 ##########################################################################################
28 live_loop :track5 do #ambient
29   with_fx :pitch_shift, pitch: [-24,-12,-12,0,0,0,0,8,12,12,16,24].choose,
30     window_size: rrand(0.001,0.2), time_dis: (rrand(0.01,0.1) / 2) do
31     with_fx :distortion, distort: rrand(0.3,0.666), mix: rrand(0.5,0.9) do
32       with_fx :echo, phase: [1,2,4,4,4,8,8,16].choose,reps: [1,2,3].choose, mix: rrand(0.4,0.99) do
33         with_fx :compressor, pre_amp: (rrand(1.0,1.5) * 1.0),
34           threshold: 0.4, slope_above: 0.266, relax_time: 2 do
35           sample sampleBell, start: rrand(0.025,0.086), pitch: [-12,-8,-4,-2,0,2,4,8,12].choose,
36             window_size: rrand(0.01,0.2), amp: rrand(0.9,1.1) * 1.1, norm: [0,0].choose,
37             pan: rrand(-0.8,0.8), pan_slide: [1,2,4,8,16].choose
38             beat_stretch: rrand(4,160), attack: [1,2,4,4,8,8,16].choose
39           end
40         end
41       end
```

```
    end
    sleep [2,4,4,4,4,4,8,8,8,16].choose
  end
  sleep 8
  sample sampleBell, rate: -2
  sleep 4
##########################################################################################
live_loop :track1 do #kick
  sample :bd_klub, amp: ring(0.8,1.2,0.8,1.6)[tick] * [1.2,1.3].choose
  sample :bd_haus, amp: ring(0.8,0.3,0.8,0.5)[tick] * [1.2,1.3].choose

  if one_in(24)
    sample sampleBell, rate: [-4,-2,-2,-1,-1,1,2,4].choose, amp: 0.75
  end # if

  sleep 1
  sleep 15 if one_in(256)
  end
##########################################################################################
live_loop :track2 do #hat
 use_synth [:pnoise].choose
 with_fx :pitch_shift,pitch:16,window_size:[rrand(0.0001,0.001),rrand(0.0001,0.1)].choose do
   play :C3, amp: ring(0.3,0.2,0.3,0.33)[tick] * rrand(0,0.3), release: [1,2,3,4].choose
 end
 with_fx :pitch_shift, window_size: [rrand(0.0001,0.001),rrand(0.0001,0.1)].choose do
   play :C3, amp: ring(0.3,0.2,0.3,0.33)[tick] * 0.4
 end
 sleep 1.5
 sleep 1.5 * 4 if one_in(128)
end

end #compressor
```

04. "church-of-men-of-love"

Linha **51**: A cada 32 passagens por esse trecho de código, o sistema seleciona o próximo sintetizador listado.

Linhas **65** a **69**: Duas opções de tempo de espera para repetição do *loop* escolhidas com uma bifurcação lógica ***if***.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-church-of-men-of-love.mp3

www.freesound.org/people/vmgraw/sounds/235615

```
01 # Alexandre rANGEL "church-of-men-of-love"
02 # feeling weird about physicality
03 # 30-Jan-2016 / Sonic Pi 2.12
04
05 #use_bpm 96 # for intro
06 use_bpm 136
07 startClock = 0 # 0 to start song at intro
08 clock = 0; bar = 0
09 set_volume! 0.85
10 t = Time.new
11 use_random_seed = 4 #(t.year + t.month + t.day + t.hour + t.min + t.sec)
12 sampleMetal = 'C:/samples/metal.wav' # your sample path
13 load_sample sampleMetal
14 ##########################################################################################
15 live_loop :metro do
16   cue :metro
17   sleep 0.5
18   clock = startClock + tick; bar = clock / 4
19   puts "bar : #{bar}"; puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
20 end #metro
21 ##########################################################################################
22 with_fx :compressor,threshold: 0.5,slope_below:1.2,slope_above:0.77,mix:0.7 do
23 ##########################################################################################
24 live_loop :ambienceBeat do
25   sample sampleMetal, beat_stretch: 36, start: 0.38, pitch: -0.2,
26     window_size: rrand(0.0001,0.01), attack: 0.5, release: 3.5, amp: 4
27   sleep 18/2
28 end # :ambienceBeat
29 sleep (16)-2 if bar < 16 # delay start if intro
30 ##########################################################################################
31 live_loop :synth1 do
32   sync :metro
33   with_fx :pitch_shift, window_size:rrand(0.01,0.05), pitch:(([-12,-8,-4,0,4,8,12,16].choose)-8) do
34   with_fx :slicer, phase:(ring 1,0.75,0.5,0.2,0.1)[tick/32], mix:(ring 0,0.2,1,0.5,0,1)[tick/8] do
35     for i in 1..4
36       with_fx :slicer, phase: (0.25 * i) do
37 sample [:bass_voxy_c,:bass_trance_c].choose, rate: (0.1*i), attack: (ring 0,0.4,0,0.8)[tick], amp: 2
38 sample :bass_voxy_c, rate: -(0.1*i), attack: (ring 0,0.4,0,0.8)[tick], amp: 2
39       sleep 4 * i
40       end #slicer
41     end #for
42     end #slicer
43     end #pitch
44     sleep 4
```

```
  end # :synth1
sleep 4 #delay start
sleep 36 / 3  #delay start
##########################################################################################
live_loop :bass do
  sync :metro
 use_synth (ring :tb303,:pulse,:prophet)[tick/32]
 with_fx :ixi_techno, res: (ring 0.2,0.6,0.8)[tick/2], mix: (ring 0,0.5,1,1)[tick/2] do
      with_fx :echo, phase: (ring 2.5,0.25,2.5,0.5,1.5)[tick/16] do
play chord(:c2, :minor7 ).choose, amp: 0.6 *1.2, attack: 0.1,release: (ring 0.2,0.4,0.1,0.8)[tick/32]
      end #echo
    end # :ixi_techno
    sleep [0.5,1,1,2].choose
  end # :bass
  sleep 8 #delay start
##########################################################################################
live_loop :hat do
  sync :metro
  use_synth [:cnoise,:cnoise,:pnoise,:noise].choose
  play 60, release: (ring 2,2,2,0.6,2,2,2,0.8)[tick], amp: (ring 0.07,0.09)[tick]
  if one_in(12)
    sleep 1
  else
    sleep 2
  end #if
end #hat
##########################################################################################
sleep 2 #offbeat from hat
##########################################################################################
live_loop :kick do
  sync :metro
  sleep 0.5
  with_fx :ixi_techno, res: (ring 0.2,0.6,0.8)[tick/2] do
   8.times do
     sample :bd_klub, rate: (ring 0.9,0.8,0.9,0.7)[tick/2], amp: 4 * 1.05
     sleep 2
   end
   128.times do
    sample :bd_klub, rate: (ring 0.9,0.8,0.9,0.7)[tick/2], amp:3.5*1.05
    sample :bd_haus, rate: (ring 0.9,0.8,0.9,0.7)[tick/4], amp:rrand(0.4,0.5) * rrand(1.03,1.08)
    sleep 1
   end
  end # :ixi_techno
  sleep 8
end # :kick
##########################################################################################
end #compressor
```

05. "sunday morning worship code"

Linhas **25** a **27**: Definem a variável *myBPM* de acordo com o andamento da composição.

Linhas **30** a **33**: Agregam chances de definição aleatória da variável *myBPM*.

Linhas **67** a **76**: Trecho executado com a velocidade estabelecida pelo valor armazenado pela variável *myBPM*.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-SundayMorningWorshipCode.mp3

www.freesound.org/people/garysanford/sounds/178111
www.freesound.org/people/ERH/sounds/163465

```
01 # Alexandre rANGEL "sunday morning worship code"
02 # 06-Feb-2016 / Sonic Pi 2.12
03
04 mybpm = 144
05 use_bpm mybpm
06 startClock = 0 # 0 to start song at intro
07 clock = 0 # global var
08 bar = 0 # global var
09 set_volume! 0.6
10
11 t = Time.new
12 x = (t.year - t.month - t.day - t.hour - t.min - t.sec)
13 puts "x = #{x}" #1931
14 use_random_seed = x
15
16 sampleHarley = 'C:/samples/harley.wav'; load_sample sampleHarley
17 sampleMachinery = 'C:/samples/machinery.wav'; load_sample sampleMachinery
18 ##########################################################################################
19 live_loop :metro do
20   cue :metro
21   sleep 0.5
22   clock = startClock + tick
23   bar = clock / 4
24
25   mybpm = 144 / 2 if bar < 24
26   mybpm = 144 if bar == 0
27   mybpm = 144 if bar > 24
28
29   #ramdom bpms:
30   mybpm = 144 / 4 if one_in(36*1.1)
31   mybpm = 144 / 2 if one_in(72*1.1)
32   mybpm = 144 / 8 if one_in(48*1.1)
33   mybpm = 144 * 2 if one_in(24*1.1)
34
35   puts "let's go!" if clock < 1
36   puts "bar : #{bar}, bpm: #{mybpm}"; puts (ring "1 !","2 ! !","3 ! ! !","4 ! ! ! !")[clock]
37 end #metro
38 ##########################################################################################
39 sleep 0.5 #offset metro2
40 ##########################################################################################
41 live_loop :metro2 do
42   sync :metro
43   sleep 0.5
44   cue :metro2
```

```
  sleep 0.5
end #metro2
##########################################################################################
with_fx :compressor, slope_above: 0.75, pre_amp: 0.8, mix: 0.8 do
##########################################################################################
  live_loop :harley do
    with_bpm mybpm do
      with_fx :slicer, phase: [0.1,0.2,0.25,0.5,1].choose, mix: [0.01,1,1,1,1,1,1].choose do
        with_fx :compressor do
          sample sampleHarley, beat_stretch: 128, amp: 4
        end #:compressor
      end #:slicer
      sleep 128
    end #with_bpm
  end #harley
##########################################################################################
  sleep 8 #delay start
##########################################################################################
live_loop :bass1 do
  sync :metro
  use_synth :pulse

  with_bpm mybpm do
    with_fx :pitch_shift, pitch: [0,2,4,8].choose, window_size: 0.05 do
      with_fx :slicer, phase: (ring 0.5,0.25,0.5,0.95,0.1)[tick/16] do
        with_fx :distortion, distort: 0.5, mix: 0.5 do
          play chord(:c0, :minor).choose, attack: 0.01, sustain: 2, release: 2, pan: -0.2, amp: 0.66
        end #krush
      end #slicer
    end #pitch
    sleep (ring 2,1,4)[clock/16]
  end #mybpm

end #bass
##########################################################################################
sleep 32 # delay start

live_loop :bass2 do
  sync :metro
  use_synth [:tb303,:prophet].choose
  with_bpm mybpm do
    with_fx :compressor do
      with_fx :flanger, decay: [0.25,0.5,1,2,4,8,12].choose, mix: 0.5 do
        with_fx :slicer,phase:[0.25,0.5,0.75,1.0].choose,mix:(ring 0.25,0.5,0.75,1.0)[tick] do
          if one_in(2)
            play chord(:c0, :maj11).choose, attack: 2.0, sustain: 1, release: 6,
              pan: 0.2, amp: rrand(5,7) * 0.4
          else
          play chord(:c1,:dom7).choose, attack:4, sustain:2, release:4, pan 0.2, amp:rrand(5,7) * 0.4
          end #if
        end #slicer
      end #flanger
    end #compressor
    sleep [1,2,3,4,4,6,6,8].choose
  end #mybpm
end #bass
##########################################################################################
sleep 24 # delay start
##########################################################################################
live_loop :kick do
  sync :metro
```

```
    with_bpm mybpm do
    with_fx :echo, phase: (ring 0.25,0.5,0.2,0.25,0.5,3)[tick/32], mix: (ring 0.4,0.8)[tick/2] do
      sample :bd_haus, amp: rrand(3.3,3.5) * 1.1 * 0.66
      use_synth :pulse
      play (ring 40,40,40,32)[tick/1], amp: 0.7 * 0.66
    end #echo
    sleep (ring 4,2,1,1,1,1,1,1,  4,1,1,1,1,1,1,1)[tick/16]
  end #mybpm
end #kick
##########################################################################################
  sleep 16.5 # delay start
##########################################################################################
  live_loop :hat do
    sync :metro2
    use_synth :cnoise
    with_bpm mybpm do
      with_fx :echo, phase: 1.0/3, mix: [0,0,0,0.9][tick/12] do
       with_fx :echo, phase: 0.25, mix: [1,1,1,0][tick/12] do
       with_fx :pitch_shift, pitch: (ring 8,8,12,12)[tick], window_size: (ring 0.040,0.08)[tick/8] do
          play 48, attack: 0.1, release: 0.1, amp: (ring 2.25,2.6,4.0,2.5)[tick] + 3.0 * 0.6
          play rrand_i(42,54), attack: 0.1, release: 0.1, amp: (ring 4.25,2.6,2.6,2.5)[tick]+3*0.6
          end #pitch
        end #echo2
      end #echo1
    sleep 1
    end #mybpm
  end #hat
##########################################################################################
  sleep 16 #delay start
##########################################################################################
  live_loop :ride do
    sync :metro2
    use_synth (ring :noise,:cnoise)[tick/2]
    with_bpm mybpm do
      play 60, attack: (ring 0.02,0.03,0.04)[tick/32], release: (ring 0.5,0.8,0.3,0.75,0.1)[tick/8],
        amp: rrand(0.14,0.16) + (ring 0.4,0.45)[tick/2] * 1.333
      sleep (ring 1,1,1,1,4,1,1,1,  4,1,1,1,2,1,1,1)[tick/64
    end #mybpm
  end #ride
##########################################################################################
  #comment do
live_loop :beep do
  sync :metro

  with_bpm mybpm do
    if one_in(2)
        use_synth [:sine,:saw,:beep,:pulse].choose

with_fx :pan, pan: [-0.8,-0.5,0,0.5,0.8].choose, pan_slide: [0.25,0.5,1,2,4,8].choose do
with_fx :normaliser, mix: [0.2,0.4,0.6,0.7,0.8].choose do
with_fx :pitch_shift, pitch: [-2,-4,-8,-12,-16,-24].choose,
        window_size:  [0.01,0.02,0.04,0.1,0.2].choose do
with_fx :ixi_techno, phase: [1,2,3,4].choose, mix: rrand(0.5,0.75) do
with_fx :flanger, phase:[0.25,0.5,1,2,4].choose,
        res:rrand(0.0,0.5), res_slide:1, mix:[0.05,0.5,1].choose do
with_fx :echo, phase: [0.25,0.5,1.5,3].choose, mix:(ring 0.4,0.7,0.4,0.9)[clock/4] do
     play [60,60,64,90].choose, amp: 1.0 *1.1, attack: 0.1, release: 2
     play [60,60,64,90].choose, amp: 1.2 *1.1, attack: 0.1, release: 4
     play [60,62,64,90].choose, amp: 1.4 *1.1, attack: 0.5, release: 8
     end #echo
    end #flanger
```

```
    end #ixi
   end #pitch
  end #normalizer
 end #pan

     end #if

       sleep 4
     end #mybpm

   end #beep
   #end #comment
##########################################################################################
live_loop :machinery do
 with_bpm mybpm do
  with_fx :compressor, pre_amp: 1.3, slope_below: 2.4, mix: 0.8 do
with_fx:pitch_shift,pitch:[-2,-4,-8,-12,-16,-24].choose,window_size:[0.001,0.004,0.01,0.02].choose do
sample sampleMachinery, beat_stretch: [16,32,64].choose, amp: rrand(3,7) * 1.5, rate: [-1,1,1].choose
      end #:pitch_shift
    end #:compressor
    sleep [2,4,8,8,16,16,16,32].choos
 end #mybpm
end #machinery
##########################################################################################
end #:compressor
```

06. "atomic garden of gravitational waves"

Linha **27**: O efeito **pitch_shift**, conjugado com o parâmetro **window_size**, obtém sonoridades de granulação sonora, técnica de desconstrução dos sons em trechos individuais com pequenas frações de segundo de duração.

Linha **53**: Várias opções de tempo de espera às repetições do *loop*.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-atomic-garden-of-gravitational-waves.mp3

www.freesound.org/people/Digitopia_CdM/sounds/261681

www.youtube.com/watch?v=BEyHi5LkQWk

```
1  # Alexandre rANGEL "atomic garden of gravitational waves" v10
2  # 14-Feb-2016 / Sonic Pi 2.12
3
4  mybpm = 66; use_bpm mybpm
5  startClock = 0 # 0 to start song at intro
6  clock = startClock # global var
7  bar = 0 # global var
8  set_volume! 0.8
9
10 t = Time.new; x = (t.year - t.month - t.day - t.hour - t.min - t.sec); puts "x = #{x}" #1951
11 use_random_seed = x
12 sampleGamelan = 'C:/samples/gamelan.wav'; load_sample sampleGamelan
13
14 ##########################################################################################
15 live_loop :metro do
16   cue :metro
17   sleep 0.5
18   clock = startClock + tick; bar = clock / 4
19   puts "let's go!" if clock < 1
20   puts "bar : #{bar}, bpm: #{mybpm}"; puts (ring "1 !","2 ! !","3 ! ! !","4 ! ! ! !")[clock]
21 end #metro
22 ##########################################################################################
23 with_fx :compressor, slope_below: 1.66, slope_above: 0.95 do
24 ##########################################################################################
25
26 live_loop :gamelan do
27  with_fx :pitch_shift, pitch:([-16,-12,-8,-6,-4].choose+16),window_size:rrand(0.0001,0.001) do
28   sample sampleGamelan, rate: (ring 1,1,1,-1,0.5,0.5,0.9,-0.85)[tick],
29   amp: (ring 19,5,7,5,7,5,7,5)[tick], pan: rrand(-0.33,0.33), pan_slide: rrand(0.02,0.8)
30 end
31
32 with_fx :pitch_shift, pitch: [-18,-14,-10,-8,-6].choose, window_size: rrand(0.0001,0.001) do
33  sample sampleGamelan, rate: (ring 1,1,1,-1,0.5,0.5,0.9,-0.85)[tick] * 0.75,
34  pan: rrand(-0.6,0.6), pan_slide: rrand(0.02,0.8), amp: (ring 18,5,7,5,7,5,7,5)[tick]
35 end
36
37 with_fx:pitch_shift,pitch:[-20,-16,-12,-10,-8].choose,window_size:rrand(0.0001,0.001) do
38   sample sampleGamelan, rate: (ring 1,1,1,-1,0.5,0.5,0.9,-0.85)[tick] * 0.666,
39   pan: rrand(-0.2,0.2), pan_slide: rrand(0.4,0.8), amp: (ring 20,5,7,5,7,5,7,5)[tick]
40 end
41
42   sleep (ring 12,12,8,24)[tick]
```

```
end
##########################################################################################
sleep 3
##########################################################################################
live_loop :notes do

  if rand(100) > [12,24,36,48].choose

    x = rand_i(6)
    rand_note1 = (ring :a2,:b2,:c2,:d2,:e2,:f2,:g2)[tick+x]
    rand_note2 = (ring :a2,:b2,:c2,:d2,:e2,:f2,:g2)[tick+x+3]
    rand_note3 = (ring :a2,:b2,:c2,:d2,:e2,:f2,:g2)[tick+x+5]

    with_fx :slicer, phase: (ring 32,16,8,16,4,16,2)[tick/2],
    smooth_up: (ring 0.1,1,0.05,0.08,0,0,0.02)[tick/2],
    smooth_down: (ring 0.2,0.2,0.2,0.2,0,0,0.05)[tick/2] do

with_fx :echo, phase: [4,8,8].choose do

with_fx:pitch_shift,pitch:[-8,-8,8].choose,pitch_dis:0.2,mix:0.66,window_size:0.001 do
with_fx:pitch_shift,pitch:[-8,8,8].choose,pitch_dis:0.5,mix: 0.66,window_size:0.001 do
with_fx :pitch_shift, pitch: [-6,-12].choose,window_size: rrand(0.0001,0.001) do

use_synth :dsaw
play rand_note1, detune: rrand(0,1), pitch_shift: [-4,-6,-8,-12].choose,
window_size: rrand(0.0001,0.001), detune_slide: 4, attack: 3, sustain: 2, release: 2,
pan: rrand(-0.53,0.53), note_slide: 2, amp: 0.55

with_fx:pitch_shift, pitch:[-12,-16,-18,-24,-36].choose, window_size:rrand(0.0001,0.001) do
  use_synth :prophet
  play rand_note2, detune: rrand(0,1), detune_slide: 4, attack: 2, sustain: 3, release: 3,
  pan: rrand(-0.54,0.54), note_slide: 2, amp: rrand(0.5,0.7) * 0.55
end #:pitch_shift

with_fx :krush, cutoff: rrand(0,131) do
with_fx :pitch_shift, pitch:[-18,-24,-36].choose, window_size:rrand(0.0001,0.0005) do
with_fx :krush, cutoff: rrand(1,64), mix: 0.5 do
use_synth :mod_pulse
play rand_note3, detune: rrand(0,1), detune_slide: 4, attack: 4, sustain: 1, release: 4,
pan: rrand(-0.5,0.5), note_slide: 2, amp: rrand(0.5,0.8) * 0.55
                    end #:krush
                  end #:pitch_shift
                end #:krush
              end #:pitch_shift
            end #:pitch_shift
          end #:pitch_shift
        end #:echo
      end #:slicer

    end #if
    sleep 8
  end #:live_loop
##########################################################################################
live_loop :synth1 do
  with_fx :compressor, slope_below: 1.5, slope_above: 1.5, mix: 0.5 do
  with_fx :echo, delay: [3,6].choose, phase: rrand(0.6,0.8),pre_amp:1.33,mix:rrand(0.7,1.0) do
  with_fx :distortion, distort: rrand(0.4,0.6), mix: rrand(0.8,1.0) do
    use_synth :supersaw
    play [:c1,:c2,:c2,:c2].choose, amp: 0.333 *1.1
    sleep [2,2,2,2,2,2,2,2,2,3].choose
      end
```

```
    end
  end
end
##########################################################################################
live_loop :drums1 do
  sample :bd_haus, amp: rrand(0.69,0.76) * 1.5
  sleep 0.5
end
##########################################################################################
live_loop :drums2 do
  with_fx :echo, phase: 0.25, mix: 0.8 do
    with_fx :hpf do
      with_fx :bitcrusher, bits: (ring 7,9,7,9, 7,5,7,5)[tick/2] do
        sample :drum_snare_soft, amp: rrand(5,7) * 0.74
      end
    end
  end
  sleep 0.5
end
##########################################################################################
live_loop :drums3 do
  with_fx :echo, phase: 0.3333, mix: [0.6,0.8].choose do
    with_fx :hpf, cutoff: 30 do
      sample :drum_tom_mid_soft, amp: rrand(0.9,0.95) * 1.83
    end
  end
  sleep [0.333,0.333,0.333,0.333,0.5,0.5,0.5,0.5,1,4].choose
end
##########################################################################################
live_loop :drums4 do
  with_fx :echo, phase: [1,2,3,4].choose, mix: 0.5 do
    with_fx :wobble, res: rrand(0.1,0.9) do
      sample :drum_bass_hard , rate: [0.6,0.7].choose, amp: rrand(0.7,0.8) * 1.55
      sleep 2
    end
  end
end
##########################################################################################
end #:compressor
```

07. "still intact and comfortably outside the event horizon"

Linhas **5** e **11**: A variável *arrangement* controla se a composição será executada seguindo uma ordem cronológica de eventos, de acordo com a variável numérica **crescente *bar***.

Linhas **37**: Mostra na janela de informações (*log)* o número do compasso.

	www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-polaris-untamed-ritual.mp3
	www.freesound.org/people/modularsamples/sounds/291042
	www.youtube.com/watch?v=fzb5sf2Dcyo

```
01 # Alexandre rANGEL 'still intact and comfortably outside the event horizon:
02 # polaris untamed ritual' v18
03 # 21-Feb-2016 / Sonic Pi 2.12
04
05 arrangement = true
06 mybpmbase = 30
07 mybpm = mybpmbase # * 2 #6, 8, 12, 16, 4, 2
08 use_bpm mybpm
09 startClock = 0 # 0 to start song at beginning
10 clock = startClock # global var
11 bar = 0 # global var
12 set_sched_ahead_time! 2
13 set_volume! 0.75
14
15 t = Time.new
16 x = ((t.year - t.month - t.day - t.hour - t.min - t.sec) * 1).to_int
17 puts "x = #{x}"; use_random_seed = x #1970 1962 1964 1918
18
19 samplePolaris = 'c:/samples/polaris.wav'; load_sample samplePolaris
20
21 live_loop :metro do
22
23   #ramdom bpms:
24   mybpm = mybpmbase * [2,2,2,2,2,2,2,2,2,2,2,2,0.5,1,4,4,4].choose
25   mybpm = 15.0 / 2.0 if bar == 1
26   mybpm = 15 if bar == 2
27   mybpm = 30 if bar == 3
28   mybpm = 60 if bar == 4
29   mybpm = 60 if bar == 5
30
31   with_bpm mybpm do
32     cue :metro
33     sleep 0.5
34     clock = startClock + tick
35     bar = (clock / 4) + 1
36     puts "let's go!" if clock < 1
37     puts "bar : #{bar}, bpm: #{mybpm}"
38     puts ring( "1 !","2 ! !","3 ! ! !","4 ! ! ! !")[clock]
39   end #mybpm
40 end #metro
41 ##########################################################################################
42 with_fx :compressor, slope_below: 2.0, slope_above: 1.15, relax_time: 0.1, mix: 0.5 do
43
44   live_loop :polaris1 do
45     with_bpm mybpm do
46       sample samplePolaris, beat_stretch: (ring 128,64,32)[tick], amp: 2.3
47       sleep 2
```

```
48
49        with_fx :flanger, delay: ring( 2,4).tick do
50          sample samplePolaris, beat_stretch: 128, pitch: 16, amp: rrand(1.55,1.75),
51            pan: -1, rate: (ring 1,-1,1,-1,1,-1)[tick]
52        end #:flanger
53        sleep 64 / 2
54
55        with_fx :slicer, phase: 0.25 do
56          sample samplePolaris, beat_stretch: [32,64].choose, window_size: rrand(0.001,0.006),
57            amp: rrand(1.6,1.9), pan: -1, rate: (ring 1,-1,1,-1,1,-1)[tick+1]
58        end #:slicer
59
60        with_fx :slicer, phase: 0.5, smooth_up: 0.1, smooth_down: 0.1 do
61          sample samplePolaris, beat_stretch: [64,32,64,32].choose,
62            window_size: rrand(0.001,0.01), rate: -1, pitch: -16, amp: 1.75, pan: -1
63        end #:slicer
64        sleep 64 / 2
65
66      end #bpm
67    end #:polaris1
68
69    with_fx :compressor, slope_below: 2.73, slope_above: 1.8, relax_time: 0.1, mix: 0.5 do
70      with_fx :pitch_shift, pitch: -8, window_size: rrand(0.001,0.004) do
71        with_fx :ixi_techno,phase:[2,4,8,16,24].choose,res:rrand(0.5,0.9),mix:rrand(0.5,0.75) do
72
73          live_loop :polaris2 do
74            with_bpm mybpm do
75
76              sample samplePolaris, beat_stretch: [1,2,4,8].choose, pitch: [-3,-5,-7].choose,
77                window_size: 0.001, amp: rrand(0.3,0.55) * 2.25,pan: 1
78              sleep 2
79
80              sample samplePolaris, beat_stretch: [1,2,4,8].choose, pitch: [-3,-5,-7].choose-8,
81                window_size: rrand(0.001,0.005), amp: rrand(0.3,0.4) * 2.34, pan: 1
82              sleep [2,4,8,16,24].choose / 2
83
84            end #mybpm
85          end #:polaris2
86
87        end
88      end
89    end
90
91  end #:compressor1
92  ##########################################################################################
93  sleep 16 if arrangement == true # delay start
94  ##########################################################################################
95  live_loop :beat1 do
96
97  with_fx :krush, cutoff: 80, mix: rrand(0.3,0.6) do
98  with_fx :pitch_shift, pitch: [-72,-64,-64,-48,-36].choose, window_size: rrand([0.004,0.007].choose,0.008) do
99  with_fx :distortion, distort: rrand(0.8,1.0), mix: rrand(0.8,0.9) do
100 with_fx :lpf, cutoff: rrand(70,90) do
101   sample samplePolaris, start: 0.35, finish: 0.4, beat_stretch: 4, amp: rrand(1.0,1.4) * 1.65,
102   rate: rrand(0.9,1.0), pan: rrand(0.5,1.0), pan_slide: [0.25,0.5,1].choose
103         end
104       end
105
106       with_fx :distortion, distort: rrand(0.75,1.0), mix: rrand(0.8,0.9) do
107         with_fx :lpf, cutoff: rrand(80,100) do
108           sample samplePolaris, start: 0.4, finish: 0.42, beat_stretch: 2, amp: rrand(1.0,1.4) * 1.85,
109             rate: -1, pan: rrand(0.5,1.0), pan_slide: [0.25,0.5,1].choose
110         end
111
112       end
```

```
    end
    sleep 0.25 * [0.5,1,1,1,1,2,4].choose
    sleep 0.25 if one_in(4)
    sleep 0.50 if one_in(16)
  end
end
##########################################################################################
sleep 4 if arrangement == true # delay start
##########################################################################################
live_loop :beat2 do

with_bpm (mybpm * 1.0) do
with_fx :echo, phase: [0.25].choose, decay: 1, mix: (ring 0.0,0.333,0.4,0.5,0.6,0.8)[tick/4], reps: 2 do
with_fx :reverb, room: 0.7, mix: 0.3 do
with_fx :bitcrusher, sample_rate: rrand(17000,28000), window_size: [0.001,0.002,0.003,0.004].choose,
        mix: rrand(0.2,0.4), bits: [11,12,13,14,15,16].choose, pitch: [-24,-16,-8].choose do

sample :bd_haus, rate: [0.9,0.94,0.96,0.98,1.0].choose, pan: rrand(-0.3,0.3),
rate: [0.5,0.75,1,1,1,1,1,1,1,1].choose, amp: rrand(1.3,1.5) * 1.1 * ring( 0.33,0.5,0.33,0.7)[tick] * 1.4

        end #:bitcrusher

sample [:bd_klub,:bd_pure,:bd_klub].choose, rate: [0.9,1.0].choose,
pan: rrand(-0.12,0.12), rate: [-1,1].choose, amp: rrand(1.4,1.7) * ring( 0.7,0.2,0.5,0.1)[tick] * 3.48

sleep 1.0 * 0.5

      end #mybpm
    end #:reverb
  end #echo
end #:beat2
##########################################################################################
sleep 0.25 / 2.0  # offset
##########################################################################################
live_loop :tss1 do
  with_bpm (mybpm/4.0) do
    with_fx :reverb, room: rrand(0.06,0.8), mix: rrand(0.3,0.6) do
      sample :drum_snare_hard, amp: rrand(0.6,0.9) * 0.6
      sample :drum_snare_soft, amp: rrand(1.3,1.5) * 0.8
    end #:reverb
    sleep 0.25
  end #mybpm
end #:tss1
##########################################################################################
sleep 0.2 # offset
##########################################################################################
live_loop :bass do
with_fx :slicer, phase: ring( 0.25,1,2,4,8,0.5)[tick/4],
        slope_up: (ring( 0.25,1,2,4,8,0.5)[tick/4])/2, slope_down: ring(0.25,1,2,4,8,0.5)[tick/4] do
with_fx :slicer, phase: ring( 0.5,0.75,1,0.25)[tick/[4,6].choose] do
  sample :bass_drop_c, rate: -[0.9,1.0].choose, pitch: [-36,-24,-16,-8].choose,
         pan: rrand(-0.52,0.52), pan_slide: [0.2,0.25].choose,
         beat_stretch: [4,6,8].choose, window_size: rrand(0.0001,0.01),
         start: rrand(0,0.3), finish: rrand(0.4,0.8), amp: rrand(1.2,1.35) * 0.75
  end #:slicer
  end #:slicer
  sleep ring( 4,1,1,1, 2,1,2,1, 2,1,2,1, 2,1,2,1)[tick/8] * 1 #[0.5,1,1,1,1,2].choose
end
##########################################################################################
sleep 127 if arrangement == true  # delay start
sleep 0.6 # offset
##########################################################################################
with_fx :compressor, slope_below: 2.6, slope_above: 1.66 do
```

```
  live_loop :tss2 do #crazy hatter
    use_synth [:pnoise].choose
    with_bpm (mybpm/2.0) do

with_fx :pitch_shift, pitch: 16, window_size: [rrand(0.0001,0.001),rrand(0.0001,0.1)].choose do
  play :C3, amp:ring(0.3,0.2,0.3,0.33)[tick]*rrand(0.44,1.2)*0.5, release:[0.2,0.4,2].choose, rate: 0.5
end

with_fx :pitch_shift, window_size: [rrand(0.0001,0.001),rrand(0.0001,0.1)].choose do
  play [:A3,:C3,:G3].choose,amp:ring(0.3,0.2,0.3,0.33)[tick]*0.6, release:[0.1,0.2,1].choose, rate: 0.5
end

      sleep 1.5 / [2,4].choose
      sleep 1.5 * 2 if one_in(24)
      sleep 1.5 * 4 if one_in(128)
    end

  end #mybpm
end #:compressor
```

08. "jupiter coming with the wolves"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-jupiter-coming-with-the-wolves.mp3

www.freesound.org/people/clarinet_pablo_proj/sounds/248870
www.freesound.org/people/bcnlab/sounds/270736

```
# Alexandre rANGEL 'jupiter coming with the wolves' (v13)
# 28-Feb-2016 / Sonic Pi 2.12

arrangement = true
mybpmbase = 120
mybpm = mybpmbase # * 2 #6, 8, 12, 16, 4, 2
use_bpm mybpm
startClock = 0 # 0 to start song at intro
clock = startClockç bar = 0
set_sched_ahead_time! 1
set_volume! 1.0 #1.1 #0.75
t = Time.new
x = ((t.year - t.month - t.day - t.hour - t.min - t.sec) * 1).to_int
puts "x = #{x}"
use_random_seed = x #1946 1966 1858

sampleRhodes120bpm =  'C:/samples/rhodes120bpm.wav' # your path here
sampleClarinetf6 = 'C:/samples/clarinetf6.wav' # your path here
load_samples [sampleRhodes120bpm, sampleClarinetf6, :bd_haus, :bd_pure, :bd_klub, :bd_fat]

live_loop :metro do

  mybpm = mybpmbase * [0.5,1,1,1,2].choose # random bpms
  mybpm = mybpmbase / 2.0 if bar < 3

  with_bpm mybpm do
    cue :metro
    sleep 0.5
    clock = startClock + tick; bar = (clock / 4) + 1
    puts "let's go!" if clock < 1
    puts "bar : #{bar}, bpm: #{mybpm}"
    puts ring( "1 !","2 ! !","3 ! ! !","4 ! ! ! !")[clock]
  end #mybpm

end #metro

with_fx :compressor, slope_below: 2.0, slope_above: 1.15, mix: 0.5 do
  live_loop :song1 do

    with_fx :slicer, phase: 4 do
      with_bpm mybpm do

        # the break
        if bar < (8+16) or bar > (24+16) then
          myAmp = 1.0
        else
          myAmp = 0.0
        end

        sample sampleRhodes120bpm, rate: 1.0 / 4.0, amp: 12 * myAmp,
          window_size: (ring 0.003333,0.002,0.001)[tick + 1]

      end

    end #slicer
```

```
    sleep ( (sample_duration sampleRhodes120bpm) * 4)
  end #:song1
##########################################################################################
  sleep 2 # offset
##########################################################################################
  live_loop :song2 do
    if bar > (24+16) then # the break

      with_fx :slicer, phase: 4, slope_down: 1.0/8.0 do
        with_bpm mybpm do
          sample sampleRhodes120bpm, rate: 1.0 / 4.0, amp: 7.777 * 1.3,
            window_size: (ring 0.003333,0.002,0.001)[tick], pitch: [12,14,16].choose
        end #bpm
      end #slicer

    end #if

    sleep ( (sample_duration sampleRhodes120bpm) * 4)

  end #:song2
##########################################################################################
  sleep 4 # offset
##########################################################################################
  with_fx :echo do
    with_fx :ixi_techno do

      live_loop :song3, phase: 0.5 do

        # the break
        if bar < (8+16) or bar > (24+16) then
          myAmp = 1.0
        else
          myAmp = 0.0
        end

        with_fx :slicer, phase: 8, slope_down: 1.0/8.0 do

          with_bpm mybpm do

            with_fx :flanger, phase: [0.250,0.5].choose, delay: [1,2,4].choose do
              with_fx :slicer, phase: 0.25 *0.5, slope_up: (ring 0,0.1,0,0.25)[tick/1] do

                sample sampleRhodes120bpm, rate: 1.0 / 4.0, amp: 5.555 * myAmp,
                  window_size: (ring 0.001,0.003333,0.002,)[tick], pitch: -24

              end #:slicer
            end #:flanger

          end #:bpm
      end #slicer
          sleep ((sample_duration sampleRhodes120bpm) * 4.0)

      end #:song3

    end #:ixi
  end #:echo
##########################################################################################
  #sleep 24
##########################################################################################
  live_loop :scream1 do

    with_bpm mybpm do

      sample sampleClarinetf6, pitch: rrand(-36,-12),
        window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick],
```

```
        beat_stretch: (ring 16,1,2,1,4,1,2,1,8)[tick] * 2,
        pan: rrand(-0.2,0.2), pan_slide: (ring 0.5,1,2,4,8)[tick], rate: [-1,1].choose, amp: rrand(1.0,1.25)
      sleep (sample_duration sampleClarinetf6) * 0.25

      sample sampleClarinetf6, pitch: rrand(-36,-20),
        window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick],
        beat_stretch: (ring 1,2,1,4,1,2,1,8)[tick] * 2, rate: [-1,1,-1].choose, amp: 0.25 if one_in(8)
      sleep (sample_duration sampleClarinetf6) * 0.5

      sample sampleClarinetf6, pitch: rrand(-24,-12),
        window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick],
        beat_stretch: (ring 1,2,1,4,1,2,1,8)[tick] * 2, rate: [-1,1,1,-1].choose, amp: 0.25 if one_in(6)
      sleep (sample_duration sampleClarinetf6) * 0.25

      with_fx :slicer, phase: 0.25 do
        sample sampleClarinetf6, pitch: rrand(-36,-16), beat_stretch: (ring 4,8,16,32,64)[tick] * 2,
          window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick-1],
          pan: rrand(-0.5,0.5), pan_slide: (ring 4,8,16,32,64)[tick],
          rate: [1,-1,1].choose, amp: 0.25 if one_in(16)

      end #:slicer
      sleep (sample_duration sampleClarinetf6) * [1,2,4].choose
    end

  end #if
##########################################################################################
  live_loop :scream2 do

    if bar > (12+16) and bar < (36+16) then # the break

      with_bpm mybpm do

        sample sampleClarinetf6, pitch: rrand(-36,-12), rate: [-1,1].choose, amp: rrand(1.0,1.25) * 2,
          window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick],
          beat_stretch: 16, pan: rrand(-0.2,0.2), pan_slide: (ring 0.5,1,2,4,8)[tick]
        sleep (sample_duration sampleClarinetf6) * 0.25

        sample sampleClarinetf6, pitch: rrand(-36,-20), beat_stretch: 8, rate: [-1,1,-1].choose,
          window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick], amp: 0.5
        sleep (sample_duration sampleClarinetf6) * 0.5

        sample sampleClarinetf6, pitch: rrand(-24,-12), beat_stretch: 4, rate: [-1,1,1,-1].choose,
          window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick], amp: 0.25
        sleep (sample_duration sampleClarinetf6) * 0.25

        with_fx :slicer, phase: 0.25 do
          sample sampleClarinetf6, pitch: rrand(-36,-16), beat_stretch: (ring 4,8,16,32,64)[tick] * 2,
            window_size: (ring 0.003333,0.002,0.001,rrand(0.0001,0.01))[tick-1],
            pan: rrand(-0.5,0.5), pan_slide: (ring 4,8,16,32,64)[tick],
            rate: [1,-1,1].choose, amp: 0.25 if one_in(16)
        end #:slicer
      end #bpm

    end #if

    sleep (sample_duration sampleClarinetf6) * [1,2,4].choose
  end # scream2
end # compressor

##########################################################################################
live_loop :hat do
  x = [0.05,0.1].choose # [0.25/2.0,0.25/2.0].choose #offset

  if bar < (10+16) or bar > (23+16) then

```

```
    with_fx :echo, phase: [0.25,0.5].choose, mix: rrand(0.5,0.9) do
      with_fx :reverb, mix: 0.2 do
      use_synth :cnoise
      play rrand(90,120), attack: 0.05, release: rrand(0.10,0.12), sustain: 0.05, amp: rrand(0.10,0.16) *0.6
      sleep x #offset
      use_synth :pnoise
      play rrand(90,120), attack: 0.05, release: rrand(0.05,0.07), sustain: 0.05, amp: rrand(0.10,0.14) *0.6
      end
    end
  end #if
  sleep ((ring 1,0.5,1,0.25)[tick] * 1) - x
  sleep 4 if one_in(64)

end #hat
##########################################################################################
live_loop :kick do

  if bar < (8+16) or bar > (22+16) then
    if bar > 4 then
      if bar < 22 or bar % 13 == 0 then
        with_fx :slicer, phase: 0.25 do
          with_fx :ixi_techno, mix: 0.4, res: rrand(0.1,0.3),
          phase: (ring 0.25,48,0.5,24,16,8,4)[tick/4] do
            with_fx :echo, phase: 1.0 / 4.0, reps: 2, mix: 0.666 do
              sample :bd_fat, amp: rrand(0.6,0.8) * (ring 0.4,0.5,0,0.4,0.2)[tick/4] * 4,
                pan: rrand(-0.333,0.333), pan_slide: 1.0 / 8.0
              sleep 1.0
            end #fx
          end #fx
        end #fx
      else
        sample :bd_zome, amp: rrand(0.8,1.0) * (ring 0.2,0.5,0,0.333,0)[tick/8] * rrand(1.2,1.5),
          rate: -(1.0/[1.0,2.0,3.0,4.0,8.0].choose), window_size: rrand(0.001,0.002),
          beat_stretch: (ring 0.5,1,2,4,8)[tick/4]
        with_fx :reverb, room: 0.5 do
          sample [:drum_bass_hard,:drum_bass_soft].choose, amp: (ring 0,1,1.5,2.0,0)[tick/4]
        end #fx
        sleep 1
      end #if
      sample :bd_klub, amp: rrand(0.8,1.0) *6.4
    end #if
  end #if

  sleep 1
  sleep 4 if bar > (24+16) and one_in(24)
end #kick
```

09. "void running"

Linhas **39**, **48**, **58**, **67**, **78** e **89**: início de *loops* com variações do mesmo *sample*, cada um com parâmetros de execução diferentes.

	www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-void-running.mp3
	www.freesound.org/people/modularsamples/sounds/284669
	www.youtube.com/watch?v=nSoJeEpawvE

```
01 # Alexandre rANGEL "void running" v25
02 # 06-Mar-2016 / Sonic Pi 2.12
03
04 arrangement = true
05 mybpmbase = 116; mybpm = mybpmbase * 2
06 use_bpm mybpm
07 startClock = 0 # 0 to start song at intro
08 clock = startClock; bar = 0
09 #set_sched_ahead_time! 2
10 set_volume! 0.9 #1.1 #0.75
11 t = Time.new; x = ((t.year - t.month - t.day - t.hour - t.min - t.sec) / 30).to_int
12 puts "x = #{x}"; use_random_seed = x
13
14 sampleG5fm = 'C:/samples/g5fm.wav' # your path
15 load_samples [sampleG5fm,:bd_haus,:bd_klub,:bd_zum]
16 ##########################################################################################
17 live_loop :metro do
18   mybpm = mybpmbase * (ring 2,2,2,2,2,2,4,2)[tick/128]
19   mybpm = mybpmbase * 4 if 1 == (ring 0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0)[tick/16]
20   with_bpm mybpm do
21     cue :metro
22     sleep 0.5
23     clock = startClock + tick; bar = (clock / 4) + 1
24     puts "let's go!" if clock < 1
25     puts "bar : #{bar}, bpm: #{mybpm}; puts ring( "1 !","2 ! !","3 ! ! !","4 ! ! ! !")[clock]
26   end #mybpm
27 end #metro
28
29 live_loop :metro2 do
30   with_bpm mybpm do
31     cue :metro2
32     sleep 0.5
33   end
34 end #metro2
35 ##########################################################################################
36 with_fx :echo, phase: 4, reps: 2, mix: 1.0/3.0 do
37   with_fx :reverb do
38 ##########################################################################################
39 live_loop :note1 do
40   with_bpm mybpm * [1,0.5].choose do
41     sample sampleG5fm, pitch: rrand(-16,0), windows_size: rrand(0.003,0.005),
42       beat_stretch: 6*rrand(1,2), attack: rrand(0,6)/2.0, release: rrand(0,6),
43       pan: rrand(0,1), pan_slide: 6*rrand(1,2), amp: rrand(0,3) * 4
44     sleep 8
45   end
46 end
```

```
47  ##########################################################################################
48  live_loop :note2 do
49    with_bpm mybpm * [1,0.5].choose do
50      sample sampleG5fm, amp: rrand(0,3) *4,
51        pitch: rrand(-24,0), windows_size: rrand(0.002,0.004), beat_stretch: 4 * rrand(1,3),
52        attack: rrand(0,4)/2.0, release: rrand(0,5), pan: rrand(-1,0), pan_slide: rand(4)
53      sleep 10
54      end
55    end
56  sleep 2  #delay start
57  ##########################################################################################
58  live_loop :note3 do
59    with_bpm mybpm * [1,2].choose do
60      sample sampleG5fm, pitch: rrand(-36,0), windows_size: rrand(0.001,0.003),
61        beat_stretch: 8*rrand(1,2), attack: rrand(0,5)/2.0, release: rrand(0,5),
62        pan: rrand(-1,1), pan_slide: 8*rrand(1,2), amp: rrand(0,3) * 5
63      sleep 12
64    end
65  end
66  ##########################################################################################
67  live_loop :note4 do
68    with_bpm mybpm * [2,1].choose do
69        sample sampleG5fm, pitch: rrand(-48,0), attack: rrand(0,4), release: rrand(2,6),
70          windows_size: rrand(0.03,0.05), beat_stretch: 6*rrand(1,4),
71          pan: rrand(-0.8,0.8), pan_slide: 6*rrand(1,4), amp: rrand(0,3) * 5
72        sleep 8*2
73      end
74  end
75  ##########################################################################################
76  sleep 2
77  ##########################################################################################
78  live_loop :note5 do
79    with_bpm mybpm * [1,0.5].choose do
80        sample sampleG5fm, pitch: rrand(-64,0), attack: rrand(0,2), release: rrand(2,5),
81          windows_size: rrand(0.02,0.04), beat_stretch: 2*rrand(4,8),
82          pan: rrand(-0.5,0.5), pan_slide: 2*rrand(4,8), amp: rrand(0,3) *6
83        sleep 12 * 2
84    end
85  end
86  ##########################################################################################
87  sleep 2
88  ##########################################################################################
89  live_loop :note6 do
90  with_bpm (mybpm * [1,0.5].choose) do
91  with_fx :reverb, room: 1, mix: 0.97 do
92    sample sampleG5fm, pitch: rrand(-72,0), windows_size: rrand(0.01,0.03),
93    beat_stretch: 8*rrand(1,6), attack: rrand(0,2), release: rrand(2,5), amp: rrand(0,3) * 6
94        sleep 12*2
95      end
96    end
97  end
98  ##########################################################################################
99  end #reverb
100 end #echoo
101 ##########################################################################################
102 sleep 8
103 ##########################################################################################
104 live_loop :bass1 do
105 with_bpm mybpm * 0.5 do
106
107 16.times do
```

```
  use_synth ring(:blade,:prophet)[tick/16]
          play [36,32,30].choose, attack: 0.1, release: 3,
            amp: ring( 4.0,4.2)[tick/2] * rrand(0.25,0.3) * ring(0.7,0.8)[tick/16]
          if one_in(8)
            sleep ring( 2,2,2,8)[tick]
          else
            sleep ring( 2,1,2,1)[tick]
          end #if
        end

        sleep 4

        with_fx :slicer, phase: 0.25 * 0.5 do
          18.times do
            use_synth :fm
            with_fx :reverb, mix: 0.75 do
              play_chord chord(:e2, :m13), attack: 0.1, release: [1,3,6].choose,
                amp: ring( 3.5,3.8)[tick/2] * rrand(0.25,0.3) * ring(0.9,1.1)[tick/16]
              if one_in(8)
                sleep ring( 2,2,2,8)[tick]
              else
                sleep ring( 2,1,2,1)[tick]
              end #if
            end
          end
        end

        sleep 4

        48.times do
          use_synth :dsaw
          with_fx :slicer, phase: (ring 0.25,0.5,0.5,1,1)[tick/6] do
            with_fx :reverb, mix: rrand(0.3,0.99) do
              play_chord chord(:g3, :maj11), attack: 0.1, release: [1,3,6].choose,
                amp: (ring 3.3,3.5)[tick/2] * rrand(0.25,0.3) * ring(0.9,1.2)[tick/16]
              if one_in(8)
                sleep (ring 2,2,2,8)[tick]
              else
                sleep (ring 2,1,2,1)[tick]
              end #if
            end
          end
        end

        sleep 2
      end #bpm
    end #bass
##########################################################################################
    sleep 24 #delay start
##########################################################################################
    live_loop :kick do
      with_bpm mybpm do
        sample :bd_klub, amp: rrand(1.7,1.8) * (ring 0.2,0.9,1.1,0.9)[tick] * 1.4
        sample :bd_haus, amp: rrand(0.5,0.6) * (ring 0.7,1.1,0.9,0.2)[tick]
        sleep 1
        sample :bd_zum, amp: rrand(0.37,0.45), rate: rrand(-1.0,-0.8), start: 0.2
        sample :bd_zum, amp: rrand(0.37,0.44), rate: rrand(0.8,1.0)
        sleep 1
      end #bpm
    end #kick
##########################################################################################
```

```
    live_loop :ride do
      with_bpm mybpm do
        use_synth :cnoise
        with_fx :echo, phase: 1.0/(ring 2,6)[tick/6], mix: rrand(0.5,0.66), reps: 2 do
          play 60, release: [1,2].choose, amp: rrand(1.5,1.58) * (ring 0.0444,0.024,0.04,0.022)[look]
        end #echo
        sleep (ring 3,4,3,4,3)[tick/3]
      end
    end #ride
##########################################################################################
sleep 8 #delay start
##########################################################################################
live_loop :synth1 do
with_bpm mybpm do
  if bar % [3,4].choose == 0 then
with_fx :slicer, phase: (ring 0.1,0.1,0.2,0.3,0.5,0.15,0.75,1)[tick/2],
  slope_up: (ring 0.05,0.11,0.2,1)[tick], slope_down: (ring 0.5,1)[tick] do
with_fx :flanger, phase: [0.5,1,2].choose do
with_fx :nrbpf, centre: rrand(30,70), centre_slide: [0.5,1,2].choose,
  res: rrand(0.1,0.333), mix: rrand(0.2,0.5) do
with_fx :echo, phase: [0.25,0.5,0.5,1,2,3,4,3,8,10,12,16].choose, reps: 3, mix: 0.4444 do
use_synth :pulse
play_chord chord(:g5, :sus2), pan: rrand(-1,1), pan_slide: (ring 0.5,1)[tick/2],
  amp: (ring 0.5,0.1,0.5,0.1)[tick/4] * rrand(0.0,0.002) if one_in(3)
play (ring 70,76,74,92)[tick/2] - rrand(8,16), pan: rrand(-1,1), pan_slide: (ring 0.5,1)[tick/2],
  amp: (ring 0.5,0.1,0.5,0.1)[tick/4] * rrand(0.0,0.001) if one_in(3)
            end #echo
          end #flanger
        end #nrbpf
      end #slicer
    end #bar
    sleep (ring 1,1,1,2,1,1,2,36,  1,1,1,1,1,1,1,48)[clock/2] * [2,4].choose
  end #bpm
end #synth1
```

10. "the book of changes"

Linhas **134** a **140**: Condição de arranjo. O trecho será executado somente após o compasso 15.

	www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-the-book-of-changes.mp3
	www.freesound.org/people/Jovica/sounds/86578 www.freesound.org/people/patchen/sounds/24670
	www.youtube.com/watch?v=Fl-bDokHL5g

```
01 # Alexandre rANGEL "the book of changes" v16
02 # 13-Mar-2016 / Sonic Pi 2.9
03
04 arrangement = true
05 mybpmbase = 240
06 mybpm = mybpmbase
07 use_bpm mybpm
08 startClock = 0 # 0 to start song at intro
09 clock = startClockç bar = 0
10 set_sched_ahead_time! 2
11 set_volume! 0.80
12 t = Time.new; x = ((t.year - t.month - t.day - t.hour - t.min - t.sec) / 1.0).to_int
13 puts "x = #{x}"; use_random_seed = x
14 use_sample_pack_as 'C:/samples/', :mySamples
15 load_samples [:mySamples__dirty80bpm,:mySamples__madrid,:bd_fat,:bd_haus,:bd_pure]
16 ##########################################################################################
17 live_loop :metro do
18   mybpm = mybpmbase #* (ring 2,2,2,2,2,2,4,2)[tick/128]
19   mybpm = mybpmbase * 2 if 1 == (ring 0,0,0,0,0,1,1,0)[tick/192]
20
21   mybpm = mybpmbase / 32 if bar == 0
22   mybpm = mybpmbase / 16 if bar == 1
23   mybpm = mybpmbase /  8 if bar == 2
24   mybpm = mybpmbase /  2 if bar == 3 or bar == 4
25   mybpm = mybpmbase /  2 if bar % 63 == 0 or bar % 64 == 0
26
27   with_bpm mybpm do
28     cue :metro
29     sleep 0.5
30     clock = startClock + tick; bar = (clock / 4) + 1
31     puts "let's go!" if clock < 1
32     puts "bar : #{bar}, bpm: #{mybpm}"; puts ring( "1 !","2 ! !","3 ! ! !","4 ! ! ! !")[clock]
33   end #mybpm
34 end #metro
35 ##########################################################################################
36 live_loop :metro2 do
37   with_bpm mybpm do
38     cue :metro2
39     sleep 0.5
40   end
41 end #metro2
42 #dirty intro#################################################################
43 with_fx :echo, phase: 8, mix: 0.25, reps: 3 do
44   with_fx :echo, phase: 4, mix: 0.66 do
45     with_fx :lpf, cutoff: 75 do
46       sample :mySamples__dirty80bpm, finish: (1.0 / 8.0), amp: 0.80
47     end #lpf
```

```
  end #echo
end #echo
sleep ((sample_duration :mySamples__dirty80bpm) / 8.0)
##########################################################################################
live_loop :dirty do
  with_bpm mybpm do
  grid = 1.0 / [16.0,32.0,32.0,64.0,1024.0,2048.0].choose
  x = grid*(rrand_i(0.0,grid).to_int)
  if one_in(3) or one_in(5) or one_in(7) or one_in(9)
    with_fx :lpf, cutoff: rrand([[0,32,64].choose,72,96].choose,[102,110,120].choose),
      cutoff_slide: 1.0/[8.0,4.0,2.0,1.0,0.5].choose do
    with_fx :compressor, slope_below: 4.0, slope_above: 0.9, mix: 0.8 do
    with_bpm 40 do
      sample :mySamples__dirty80bpm, start: x, finish: x+grid,
        rate: (ring 1,1,1,-1,1,1,1,-2)[tick/6]/(ring 4.0,2.0,2.0,1.0,1.0,1.0,1.0,0.5)[tick/6],
        window_size: rrand(0.0001,0.0003), amp: rrand(2.0,3.6) * 1.2
      sleep (sample_duration :mySamples__dirty80bpm)*grid *(ring 4,2,2,1,3,1,3,0.5)[look/6]
      end
    end #compressor
  end #lpf
  else
    sleep (sample_duration :mySamples__dirty80bpm)*grid *(ring 4,2,2,1,1,1,1,0.5)[tick/6]
  end #if
  end #bpm
end #dirty
##########################################################################################
live_loop :madrid do
  with_bpm mybpm do
    with_fx :compressor, slope_below: 1.3, mix: 0.333 do
      with_fx :slicer, phase: (ring 1,0.25)[tick/32], mix: (ring 0,0.8,0,0.8)[tick/16] do
        with_bpm 40 do
          if one_in(4) or one_in(5) or one_in(6)
            with_fx :echo, phase: (ring 0.5,1,2,0.25)[tick/2], reps: 2 do
              grid = 1.0 / [16.0,32.0,32.0,64.0,128.0].choose
              x = grid*(rrand_i(0.0,grid).to_int)

with_fx :pitch_shift, pitch: -16-[3,5,7].choose, pitch_slide:0.5, window_size:rrand(0.0001,0.0005) do
 with_fx :pitch_shift, pitch: -16-[3,5,7].choose, pitch_slide: 0.5, window_size:rrand(0.001,0.002) do
  sample :mySamples__madrid, start: x, finish: x+grid, window_size: rrand(0.0001,0.0003),
    rate: (ring 1,1,1,0.5,1,1,1,0.5)[tick/2]*(ring 4,2,2,1,1,1,1,0.5)[tick/2],
    pitch: [-48,-36,-24,-16].choose - [2,3,5,7].choose, attack: 1, release: 2,
    pan: rrand(-0.6,0.7), pan_slide: [0.25,0.5,1,2].choose, amp: rrand(0.6,0.8)*rrand(1.0,1.11)
  end #pitch_shift
 end #pitch_shift
end #if

if one_in(2) or one_in(4) or one_in(6)
grid = 1.0 / [16.0,32.0,32.0,64.0,128.0].choose
x = grid*(rrand_i(0.0,grid).to_int)
with_fx :pitch_shift, pitch: -16-[3,5,7].choose, pitch_slide:0.5, window_size: rrand(0.0001,0.001) do
with_fx :flanger, phase: (ring 0.5,1,2,0.25,4,8)[tick/5] do
  sample :mySamples__madrid, start: x, finish: x+grid, window_size: rrand(0.0001,0.0003),
  rate: (ring 1,1,1,0.5,1,1,1,0.5)[tick/2]*(ring 4,2,2,1,1,1,1,0.5)[tick/2],
  attack: 2, release: 1, pitch: [-48,-36,-24,-16].choose - [2,3,5,7].choose - 16,
  pan: rrand(-0.65,0.55), pan_slide: 1.0/[1,2,4,8].choose, amp: rrand(0.6,0.8) * rrand(1.0,1.1)
end #flanger
end #pitch_shift
end #if
end

grid = 1.0 / [16.0,32.0,32.0,64.0,128.0].choose
```

```
x = grid*(rrand_i(0.0,grid).to_int)
sleep (sample_duration :mySamples__madrid)*grid *(ring 2,2,4,4,1,1,2,2)[tick/2] /4.0

        end #echo
      end #slicer
    end #compressor
  end #bpm
end #madrid
##########################################################################################
sleep 0.5
##########################################################################################
live_loop :kick do
  if bar > 23
    with_fx :lpf, cutoff: rrand(110,130), mix: 0.84 do
      with_fx :distortion, distort: rrand(0.4,0.7), mix: 0.5 do
        sample :bd_haus, amp: rrand(2,2.15) * 0.666
      end
      with_fx :distortion, distort: rrand(0.6,0.8), mix: 0.5 do
        sample :bd_pure, amp: rrand(1.6,2.0) * 0.666
      end
    end
  end #lpf

  sleep 2

if bar > 15
  with_fx :echo, phase: 1.0/(ring 6,2,4,2)[tick/2], reps: 2, mix: 0.66 do
    with_fx :distortion, distort: rrand(0.6,0.8), mix: 0.7 do
      sample :bd_fat, amp: rrand(0.8,0.9)
    end #distortion
  end #echo
end

  sleep 2
end #kick
##########################################################################################
sleep 0.5
##########################################################################################
with_fx :flanger, phase: (ring 32,16,2,4)[tick/64], wave:(ring 1,2,3,4)[tick/256], mix: 0.4 do
live_loop :hat do
  if bar > 47
    if rand(100) > 12
      8.times do
        with_fx :lpf, cutoff: rrand([100,110].choose,117), mix: 0.80 do
          use_synth (ring :cnoise,:pnoise,:pnoise,:pnoise)[tick]
          play 60, attack: 0.05, release: 0.1, amp: rrand(0.5,0.6)
        end #lpf
        sleep 0.5
      end #32times
    end #if rand

    if rand(100) > 75
      with_fx :lpf, cutoff: rrand(90,110), mix: 0.80 do
        16.times do
          use_synth (ring :cnoise,:pnoise,:pnoise,:pnoise)[tick]
          play 60, attack: 0.05, release: 0.1, amp: rrand(0.44,0.48)
          sleep 0.25
        end #32times
      end #lpf
      sleep 1
    end #if rand
```

```

    if rand(100) > 80
      with_fx :lpf, cutoff: rrand(90,100), cutoff_slide: [0.1,0.2].choose, mix: 0.80 do
        32.times do
          use_synth (ring :cnoise,:pnoise,:pnoise,:pnoise)[tick]
          play 60, attack: 0.03, release: 0.075, amp: rrand(0.52,0.57)
          sleep 1.0/4.0 #0.25
        end #32times
      end #lpf
    end #if rand
  end #if bar
################################################################################
  sleep 0.5
################################################################################
  if one_in(5)
    with_fx :pitch_shift, pitch: rrand(1,16), window_size: 0.01 do
      with_fx :distortion, distort: 0.99, mix: 0.8 do
        sample :drum_cymbal_closed,rate: rrand(0.98,1.02),attack: 0.25,release: 0.25,amp: 0.3
      end
    end
    with_fx :pitch_shift, pitch: rrand(0,16), window_size: 0.01 do
      with_fx :distortion, distort: 0.99, mix: 0.8 do
sample :drum_cymbal_closed, rate: rrand(0.98,1.02), attack:0.25, release:0.25, amp: rrand(0.15,0.25)
      end
    end
    sleep 1
  end
  if one_in(7)
    with_fx :pitch_shift, pitch: rrand(-16,16), window_size: 0.02 do
      with_fx :distortion, distort: 0.66, mix: 0.5 do
      sample :drum_cymbal_open, start: rrand(0.2,0.4), rate: rrand(0.6,1.0),
      attack: 0.33, release: 0.77, amp: rrand(0.15,0.25)
      end
    end
  end
  sleep 4.5
end #hat
end #flanger
```

11. "tilt your head in a way so that you can hear the wind"

Linhas **15** a **25**: Inicialização de dois metrônomos, com velocidades diferentes.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-tilt-your-head.mp3

www.freesound.org/people/modularsamples/sounds/278021
www.freesound.org/people/modularsamples/sounds/282331
www.freesound.org/people/modularsamples/sounds/300019

```
01 # Alexandre rANGEL "tilt your head in a way so that you can hear the wind" v13
02 # 20-Mar-2016 / Sonic Pi 2.9
03
04 arrange = true
05 set_volume! 0.73
06 use_bpm 144
07 clock = 0
08 t = Time.new; tx = (t.year / t.month / t.day + t.hour + t.min + t.sec)
09 use_random_seed tx; puts tx
10
11 use_sample_pack_as 'C:/samples/', :mySamples
12 load_samples [:mySamples__moog55,:mySamples__tetra75,:mySamples__xpander71]
13
14 ##########################################################################################
15 live_loop :metro do
16   clock = tick(:metro)
17   bar = clock / 4
18   puts "bar : #{bar}"
19   puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
20   sleep 1
21 end #metro
22
23 live_loop :metroFast do
24   sleep 0.5
25 end #metroFast
26 ##########################################################################################
27 live_loop :xpander do
28   with_fx :compressor, slope_above: 0.75, slope_below: 1.333 do
29     with_fx :slicer, phase: (ring 1,4,2,1,0.5,0.5,0.5,0.25)[tick(:xpander)/4] * 2,
30             mix: 0.7, smooth_up: 0.2, smooth_down: 0.1 do
31       with_fx :pitch_shift, pitch: rrand(-8,8), window_size: 0.001 do
32         sample [:mySamples__xpander71].choose, pitch: rrand(-24,-12),
33           window_size: rrand(0.0001,0.005), rate: rrand(-0.5,0.5),
34           attack: [1,2,4].choose, release: [1,2,4].choose,
35           pan_slide: [1,2].choose, pan: rrand(-0.75,0.75),
36           amp_slide: 2, amp: rrand(1.5,2.5) * 3.6 #if one_in(3)
37       end
38     end
39   end
40   sleep 16
41 end
42 ##########################################################################################
43 sleep 36 if arrange == true
44 ##########################################################################################
45 live_loop :bee do
46 with_fx :gverb, mix: rand(1) do
47 with_fx :flanger, phase: [0.25,0.5,1,2].choose do
48 with_fx :echo, phase: [0.25,0.5,1,2,4].choose do
```

```
    with_fx :pitch_shift, pitch: rrand(12,24),
            window_size: [0.0001,0.0002,0.0002,0.0003,0.0003].choose do
      use_synth :prophet
      play 60+12, amp: rrand(0.4,0.6), amp_slide: 0.1
      play [60,64].choose+t.sec, sustain: 4, amp: 0.08 if one_in(2)

      use_synth :pulse
      play [30,33,35].choose+t.sec, amp: 0.7
      play 64+12, sustain: 4,  amp_slide: 0.1, amp: rrand(0.2,0.3) * 0.1 if one_in(3)
    end #pitch
end #flanger
end #gverb
end #echo
    sleep (ring 1,4,1,2)[tick(:bee)] * 4
    sleep 9 if one_in(rrand_i(7,17))
end #bee
##########################################################################################
sleep 8 if arrange == true
##########################################################################################
live_loop :moog do
    with_fx :slicer, phase: (ring 1,4,2,1,0.5,0.5,0.5,0.25)[tick(:moog)/4], mix: 0.95 do
      with_fx :pitch_shift, pitch: rrand(-8,8), window_size: 0.001 do
        sample [:mySamples__moog55,:mySamples__tetra75].choose, pitch: rrand(0,12),
        window_size: rrand(0.0001,0.01), rate: rrand(-1,1), attack: 1, release: 1,
          pan_slide: [1,2,4].choose, pan: rrand(-1.0,1.0), amp_slide: 2, amp: rrand(2.6,3.8) * 1.1
      end #slicer
    end #pitch
    sleep 16
  end
##########################################################################################
  sleep 24 if arrange == true
##########################################################################################
  live_loop :tum do
    cue :metro
    use_synth [:mod_dsaw,:mod_fm,:mod_pulse].choose
    with_fx :slicer, phase: [0.25,0.5,1].choose do
      play 48-(ring 12,12,18,24)[tick(:tum1)], sustain: 2, pan_slide: [0.5,1].choose,
        pan: rrand(-0.5,0.5), rate:[-1,1].choose, amp_slide: [0.5,1].choose,
        amp: rrand(0.8,(ring 1.6,1.8)[tick(:tum2)])*0.8 if one_in(12)
    end
    sleep 4
  end #tum
##########################################################################################
  sleep 2
  sleep 24 if arrange == true
##########################################################################################
live_loop :tss do
  cue :metroFast
  use_synth (ring :cnoise,:pnoise,:pnoise,:pnoise)[tick(:tss)]
  with_fx :pitch_shift, pitch: rrand(2,4) do
    with_fx :echo, reps: 1, phase: [0.5].choose, mix: 0.75 do
play 90, attack: 0.05, release: 0.16, pan: rrand(-0.3,0.3), pan_slide: 0.2, amp: rrand(1.4,1.6) * 0.5
    end #pitch
  end #echo
  sleep 1
end #tss
##########################################################################################
sleep 64 if arrange == true
##########################################################################################
live_loop :kick do

```

```
with_fx :flanger, mix:(ring 0.5,0.9)[tick(:kick1)/2], phase:(ring 2,1,0.5,1)[tick(:kick2)/2] do
sample (ring :bd_haus,:bd_fat)[tick(:kick3)], pitch: (ring -8,[4,6,8,10,12].choose)[tick(:kick4)],
  window_size: (ring 0.0001,0.0003,0.0002,0.01,0.0004,0.001)[tick(:kick5)/32],
  amp: (ring 2.6,2,2.6,2)[tick(:kick6)] * rrand(0.5,0.7) * 0.8
sample :bass_hard_c, pitch: (ring 0,[6,8,10,12].choose)[tick(:kick4b)],
  window_size: (ring 0.0001,0.0003,0.0002,0.01,0.0004,0.001)[tick(:kick5b)/32],
  rate: rrand(0.5,1.05) * [-1,1].choose, release: 2,
  amp: 3 * rrand(0.5,0.7) * 0.5555 if one_in([36,21,18,9].choose)
end #flanger

sleep (ring 2,1,1,1,1,1,1,1)[tick(:kick7b)]*(ring 0.5,0.5,0.5,0.5,0.5,0.5,0.5,2)[tick(:kick6)/24] * 4
    sleep 4 if one_in(24)
end #kick
##########################################################################################
sleep 0.5
##########################################################################################
live_loop :kick2 do
  with_fx :flanger, mix: (ring 0.3,0.3,0.3,0.3, 0.3,1,0.3,1)[tick(:kick2d)/8], phase: 0.25 do
      sample (ring :bd_haus,:bd_klub)[tick(:kick2a)], amp: (ring 2,3,2,3)[tick(:kick2b)] * 1.22
  end #flanger
  sleep (ring 1,1,1,0.5)[tick(:kick2c)] /1.0
  sleep 4 if one_in(18)
end #kick
```

12. "ascension hill"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-ascension-hill.mp3

www.youtube.com/watch?v=VHxGCGPwHJA

```
# Alexandre rANGEL "ascension hill" v14
# 26-Mar-2016 / Sonic Pi 2.9

myBPM = 144
bpmDivider = 2
use_bpm myBPM
set_volume! 1.0 # the age of
i1 = 0
i2 = 0

##########################################################################################
live_loop :synth1 do
with_fx :normaliser, level: ring( 0,0.8,1,0.8)[tick(:synth1)] do
with_fx :reverb do
with_fx :slicer, phase: [2,0.25,0.5,1].choose do
with_bpm myBPM / [2,3,3,3,4].choose do
with_fx :normaliser, level: [0.222,0.333,0.444].choose do
with_fx :echo, phase: [2,4,8,16].choose do
with_fx :slicer, phase: [0.25,0.5,1,2].choose do
with_fx :gverb, room: rrand(30,77), spread: rrand(0.5,0.99) do
with_fx [:flanger,:ixi_techno,:krush].choose, phase: [4,2,0.5,1].choose do
with_fx [:panslicer,:flanger,:ixi_techno].choose, phase: [0.5,1,2].choose do
with_fx :echo do
with_fx :distortion, distort: 0.9, mix: 0.66 do
  i1 = i1 + 0.1
  r = (120.0 / ring( 2.0,1.0,0.5)[i1.to_int])
  with_bpm r do
    ([0,2,4,8].choose).times do
    use_synth [:blade,:prophet,:pulse].choose
    play scale(:c2, :minor, num_octaves: [3,2,1].choose).choose,
      release: [4,8,16].choose/2, amp: rrand(1.03,1.11) * 0.02, pan: 0.5
    sleep 1.0/3
    play scale(:g2, :minor, num_octaves: [1,2,3].choose).choose,
```

```
      release: [4,8,16].choose, amp: rrand(1.03,1.10) * 0.02, pan: -0.5 if one_in([2,3,5].choose)
    sleep 4 / [1,2].choose
    end # times
    ([0,2,4,8].choose).times do
      use_synth [:prophet,:blade,:fm,:beep].choose
      play scale(:g1, [:mixolydian,:major_pentatonic].choose, num_octaves: [7,1,2,3].choose).choose,
        release: [4,8,16].choose/2, amp: rrand(0.03,0.10)*0.02, pan: -0.5
      sleep 1.0/3
      play scale(:c2, :major, num_octaves: [3,2,1].choose).choose,
        release: [4,8,16].choose, amp: rrand(0.03,1.17)*0.02, pan: 0.5 if one_in([3,2,1,7,9].choose)
                                sleep 4 / [1,2].choose
                              end #times
                            end #bpm
                          end #distortion
                        end #flanger
                      end #slicer
                    end #echo
                  end #echo
                end #gverb
              end #echo
            end #normaliser
            sleep 4
          end #bpm
      end #slicer
    end #reverb
  end #level
end #synth
##########################################################################################
live_loop :synth2 do # ambience
  with_bpm myBPM / 4 do
    use_synth [:pnoise, :hollow].choose
    if one_in(2)
      with_fx :pitch_shift, pitch: [0,-2,-4,-6].choose,
        window_size: ring( 0.001,0.002,0.003,0.004)[tick(:synth2a)] do
      play (:c3), amp: 0.45, attack: [2,4,8,16].choose, release: [4,8,16].choose
      end
    else
with_fx :pitch_shift,pitch:[0,-8,-16].choose,window_size:ring(0.01,0.02,0.03,0.04)[tick(:synth2b)] do
        play (:g2), amp: 0.45, attack: [2,4,8,16].choose, release: [4,8,16].choose
      end
    end
    sleep [1,2,2,2,4,4,8,8,16,32,64].choose
  end #bpm
end
##########################################################################################
live_loop :drums1 do
  with_bpm myBPM / bpmDivider do
    #with_fx :slicer, phase:  do
    i2 = i2 + (1.0/13.0)
    i2int = quantise(i2,1)
    with_fx :slicer, phase: ring( 0.1,0.2,0.25,0.5,1) [i2int] do
      with_fx :flanger, phase: [0.25,1.0/3,0.5,1,2,3,4].choose do
sample 'C:/samples/clap909.wav', rate:rrand(0.7,1.2), pan:rrand(-0.05,0.05), amp:rrand(0.10,0.13)*2.5
sleep ring( 0.25,0.5,1.0)[rand_i(100)] #im all over the place
      end
    end
  end #bpm
end
##########################################################################################
live_loop :drums2 do
  with_bpm myBPM / bpmDivider do
```

```
    20.times do
      sample 'C:/samples/clap909.wav', pan: rrand(-0.07,0.13), amp: rrand(0.22,0.26)*1.15, rate: 1.5
      sleep 1.0/4 * 2
    end
    8.times do
      sample 'C:/samples/clap909.wav', pan: rrand(-0.07,0.13), amp: rrand(0.24,0.28)*1.2, rate: 1.75
      sleep 1.0/4 * 2
    end
    8.times do
      sample 'C:/samples/clap909.wav', pan: rrand(-0.07,0.13), amp: rrand(0.23,0.29)*1.3, rate: 2.0
      sleep 1.0/4 * 2
    end
  end
end
##########################################################################################
sleep 16
##########################################################################################
live_loop :drums3 do
  with_fx :compressor, slope_below: 1.333, slope_above: 1.0, mix: rrand(0.8,1.0) do
    with_fx :gverb, mix: rrand(0.17,0.27) do
      sample :bd_ada, amp: rrand(1.213,1.53) * 2.1 *0.6; sleep 1
      sample :bd_ada, amp: rrand(1.213,1.53) * 1.7 *1.1 *0.6; sleep 2
      sample :bd_fat, amp: rrand(1.213,1.53) * 2.1 *0.6; sleep 1
      sample :bd_ada, amp: rrand(1.213,1.53) * 1.8 *1.1 *0.6; sleep [0.5,1].choose
    end #fx
    with_fx :gverb, mix: rrand(0.33,0.4) do
      sample :bd_fat, amp: rrand(1.213,1.53) * 2.1 *0.6; sleep 1
      sample :bd_ada, amp: rrand(1.213,1.53) * 1.7 *1.1 *0.6; sleep [0.5,1].choose
      sample :bd_fat, amp: rrand(1.213,1.53) * 2.1 *0.6; sleep 1
      sample :bd_ada, amp: rrand(1.213,1.53) * 1.8 *1.1 *0.6 sleep [0.5,1].choose
    end #fx
  end #fx
  sleep 4 if one_in(16)
end
```

13. knowledge representation and reasoning

Linhas **20** a **30**: A cadeia de efeitos, por estar fora de todos os blocos de loops, aplica os onze efeitos especificados nos dois loops seguintes.

Linhas **52** a **62**: Cadeias de efeitos encontram seus comandos de término.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-reasoning.mp3

www.freesound.org/people/felipejordani/sounds/324395
www.freesound.org/people/modularsamples/sounds/307533
www.freesound.org/people/MissCellany/sounds/240640

www.youtube.com/watch?v=wSfpYoqTZas

```
01 # Alexandre rANGEL "knowledge representation and reasoning" v20
02 # 3-Apr-2016 / Sonic Pi 2.9
03
04 use_bpm 67
05 c1 = c2 = q1 = q2 = k1 = k2 = 0
06 set_volume! 0.84
07 t = Time.new; x = (t.year / t.month / t.day * t.hour * t.min * t.sec); use_random_seed x
08
09 use_sample_pack_as 'C:/samples/', :mySamples
10 load_samples[:mySamples__drumloopalien,:mySamples__roland95B6,:mySamples__hatMetal]
11 load_samples [:bd_ada,:bd_klub,:bd_zome]
12
13 live_loop :metro do
14   clock = tick(:metro)
15   bar = clock / 4; puts "bar : #{bar}"
16   puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
17   sleep 1
18 end #metro
19 ##############################################################
20 with_fx :compressor, slope_above: 0.9, slope_below: 1.13, mix: 0.3 do
21  with_fx :level, amp: 0.4 do
22    with_fx :flanger, phase: 4, depth: 5 do
23      with_fx :flanger, phase: 16, depth: 7 do
24        with_fx :flanger, phase: 128, depth: 5 do
25          with_fx :normaliser, mix: 0.5 do
26            with_fx :flanger, phase: [0.25,0.5,1,2,3].choose do
27              with_fx :pitch_shift,window_size:rrand(0.01,0.05),pitch:rrand(-36,0) do
28                with_fx :echo, phase: [0.25,0.5,1.0/3].choose do
29                  with_fx :ixi_techno, res: rrand(0.05,0.8) do
30                    with_fx :distortion, bits: [6,8,10,12,14].choose do
31 ##############################################################
32 live_loop :drone1 do
33   use_synth :fm
34   with_fx :slicer, phase: [0.25,1.0/3,0.5].choose do
35     with_fx :echo, phase: 1.0/3 do
36 play_pattern [:c4,:c4,:e4], release: rrand(0, 0.5), rate: 0.5, pan: rrand(-0.66,-0.75), amp: 0.5
37       end
38     end
39     sleep [2,4,8,16].choose / 2
40   end # bass
41
42 ##############################################################
```

```
43  live_loop :drone2 do
44    use_synth :pretty_bell
45    with_fx :slicer, phase: [0.25,1.0/3].choose do
46      with_fx :flanger, phase: 1.0/4 do
47        play_pattern [:e3,:e3,:g3], release: rrand(2, 3), rate: 0.5, pan: rrand(0.66,0.75), amp: 2
48      end
49    end
50    sleep [32,16,16,8,8,4].choose / 2
51
52                        end
53                      end
54                    end
55                  end
56                end
57              end
58            end
59          end
60        end
61      end
62    end
63
64  ######################################################################
65  sleep 6 + 8 + 8
66  ######################################################################
67
68  live_loop :quiquito do
69    with_fx :band_eq, freq: 80, db: 2, mix: 0.5 do
70      with_fx :echo, phase: 0.25, mix: rrand(0.333,0.383) do
71        8.times do
72          sample :mySamples__drumloopalien, rate: 1, attack: rrand(0.01,0.02),
73            amp: (ring 0.4,0.4,0.4,0.5)[q1], start: 0.066, finish: 0.077
74          q1 = q1 + 1
75          #sample :bd_klub, amp: 3
76          sleep 0.5
77        end
78      end
79
80      with_fx :echo, phase: 0.5, mix: rrand(0.5,0.55) do
81        4.times do
82          sample :mySamples__drumloopalien, rate: 1, attack: rrand(0.005,0.01),
83                 amp: (ring 0.4,0.5)[q2], start: 0.064, finish: 0.075,
84          q2 = q2 + 1
85          #sample :bd_klub, amp: 3
86          sleep 0.5
87        end
88      end
89    end
90  end
91  ######################################################################
92  sleep 6
93  ######################################################################
94  live_loop :hat1 do
95    with_fx :slicer, phase: 0.25, smooth_up: 0.1, smooth_down: 0.1 do
96      with_fx :flanger, mix: rrand(0.7,0.9) do
97        with_fx :pitch_shift, pitch: [-2,0,2].choose, window_size: 0.001 do
98          sample :mySamples__hatMetal, pan: rrand(-0.15,0.10), pan_slide: 0.02,
99                 rate: (ring 0.75,0.75,0.75,1,0.75,1.01,1.11,1.22)[c1] * 0.25 * [0.5,0.25].choose,
100                amp: rrand(1.4,1.5) * 5 if one_in([1,3,6,9,12].choose)
101         c1 = c1 + 1
102         sleep [1.0,3.0].choose/[3,3,6,6,6].choose
103         sleep [1.0,3.0].choose/[3,3,6,6,6].choose * 2 if one_in(12)
```

```
      end
    end
  end
end

#######################################################################
sleep 8
#######################################################################
live_loop :hat2 do
  with_fx :gverb, mix: rrand(0.2,0.5) do
    sample :mySamples__hatMetal, pitch: [0,2,4].choose,
           pan: rrand(-0.15,0.10), pan_slide: 0.05, amp: rrand(1.4,1.5) *3,
           rate: (ring 0.75,0.75,0.75,1,0.75,1.01,1.11,1.22)[c2]*[0.25,0.5].choose
    #sample :mySamples__hatMetal, rate: 0.25 if one_in(5)
    c2 = c2 + 1
    sleep [2,4].choose
    sleep [2,4].choose * 4 if one_in(13)
  end
end

#######################################################################
live_loop :roland do

  if one_in(2)
    with_fx :flanger, phase: [0.25,0.5,1,2,3,4,5].choose, mix: rrand(0.3,0.7) do
      with_fx :band_eq, freq: rrand(60,100), freq_slide: [0.5,1].choose, db: -2 do
        with_fx :band_eq, freq: rrand(80,130), freq_slide: [0.5,1,2,4].choose,
                res: rrand(0.4,0.8), res_slide: [0.25,0.5,1,2,4].choose, db: 2 do
          with_fx :slicer, phase: 0.5, mix: rrand(0.6,1) do
            with_fx :pitch_shift, pitch: rrand(-32,-8), window_size: rrand(0.0001,0.001) do
sample :mySamples__roland95B6, rate: rrand(0.1,1.0), pan: 0.3, amp: 2.1 if one_in(2)
            end
          end

          with_fx :slicer, phase: 0.25, mix: rrand(0.6,1) do
            with_fx :pitch_shift, pitch: rrand(-32,-8), window_size: rrand(0.001,0.01) do
sample :mySamples__roland95B6, rate: rrand(0.2,0.6), attack: 2, pan: 0.3, amp: 2.1 if one_in(2)
            end
          end
          sleep [2,4,8,16,32].choose
        end
      end
    end

   else

    with_fx :echo, phase: [0.5,1,2,4,8].choose do
      with_fx :flanger, phase: [0.25,0.5,1,2,3,4,5].choose, mix: rrand(0.3,0.7) do
        with_fx :band_eq, freq:rrand(60,100), freq_slide: [0.5,1].choose, db:-2 do
          with_fx :band_eq, freq: rrand(80,130), freq_slide: [0.5,1,2,4].choose,
                  res: rrand(0.4,0.8), res_slide: [0.25,0.5,1,2,4].choose, db: 2 do
            with_fx :slicer, phase: 0.5, mix: rrand(0.6,1) do
            with_fx :pitch_shift, pitch: rrand(-32,-8),window_size: rrand(0.0001,0.001) do
sample :mySamples__roland95B6, rate: rrand(0.1,1.0), pan: 0.3, amp: rrand(2.2,3.1) if one_in(2)
              end
            end

            #with_fx :slicer, phase: 0.25, mix: rrand(0.6,1) do
            with_fx :pitch_shift, pitch: rrand(-32,-8), window_size: rrand(0.001,0.01) do
sample :mySamples__roland95B6,rate:rrand(0.2,0.6),attack:2,pan:0.3,amp: rrand(2.2,2.8) if one_in(2)
            end
```

```
            #end
            sleep [2,4,8,16,32].choose
          end
        end
      end
    end

  end #if
end
##########################################################################################
end #compressor
##########################################################################################
sleep 14
sleep 1.0/4 #kick offset
##########################################################################################
live_loop :kick2 do
  with_fx :band_eq, freq: 90, db: [1,1,1,1,1,1.3][k1] do
    k1 = k1 + 1
    with_fx :gverb, mix: (ring 0.06,0.14)[k2.to_int] do
      k2 = k2 + (1.0 / 3.0)
      with_fx :wobble, phase: [0.25,0.5,1,2,2,2,4,4].choose, mix: 0.25 do
        with_fx [:ixi_techno,:ring_mod].choose, phase: [2,4].choose, mix: 0.5 do
          sample :bd_ada, amp: 2.25
          sample :bd_klub, amp: 2.6 * rrand(0.66,0.88) if one_in(2)
          sleep 0.5
          sample :bd_zome, amp: 2.25, rate: rrand(0.88,0.92)
          sample :bd_klub, amp: 2.6 * rrand(0.66,0.88) if one_in(2)
        end
      end
    end
  end
  sleep 0.5
  sleep 7 if one_in(32)
end
```

14. "the computer is singing and playing based on rules since the beginning of the night"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-the-computer.mp3

www.freesound.org/people/digifishmusic/sounds/44928
www.freesound.org/people/toiletrolltube/sounds/179881

```
# Alexandre rANGEL "the computer is singing and playing based on rules since the beginning of the night" v18
# 20-Apr-2016 / Sonic Pi 2.9

myBPM = 72
use_bpm myBPM
c1 = c2 = q1 = q2 = k1 = k2 = 0
set_volume! 0.75
use_random_seed(Time.new.year / Time.new.month / Time.new.day * Time.new.hour * Time.new.min * Time.new.sec)

use_sample_pack_as '/Users/rangel/pisamples', :mySamples
load_samples [:mySamples__vocal_la_c4,:mySamples__cello,:bd_gas,:bd_klub,:bd_fat,:bd_808]

live_loop :metro do ###############
  clock = tick(:metro); bar = clock / 4
  puts "bar : #{bar}"; puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
  tick_clock = tick
  sleep 1
end #metro

with_fx :compressor, slope_below: 1.2, slope_above: 0.8, mix: 0.9 do
  live_loop :vocal do ##############
    with_fx :compressor, slope_below: 0.9, slope_above: 0.5, mix: 0.8 do
      with_fx :flanger, phase: [1,4,8,16,21].choose do
        with_fx :slicer, phase: [2,1,0.5,0.25].choose, phase_slide: 0.25/2 do
          with_fx :echo, phase: [2,4,8].choose do
            with_fx :gverb, mix: rrand(0.5,0.9) do
              with_bpm [myBPM/4,myBPM/2,myBPM].choose do
with_fx :pitch_shift, pitch: rrand(-52,8),pitch_slide:1.5,window_size: rrand(0.001,0.01) do
  sample :mySamples__vocal_la_c4, rate: [-1,1,1,0.5,2].choose, amp: rrand(2,4.5)/2
sleep [3,1,2,2,4,4,8].choose
end
sleep [1,2,4].choose
with_fx :pitch_shift,pitch:rrand(-32,8),pitch_slide:0.25,window_size:rrand(0.001,0.003) do
  sample :mySamples__vocal_la_c4, rate: [-1,1,1,0.5,2].choose, amp: rrand(2,4)/2
end
sleep [8,2,2,4,4,1].choose
sleep [1,2,4].choose
with_fx :pitch_shift,pitch: rrand(-16,8),pitch_slide:0.5,window_size:rrand(0.0001,0.0005) do
  sample :mySamples__vocal_la_c4, rate: [-1,1,1,0.5,2].choose, amp: rrand(1,2)
end
sleep [1,2,2,4,4,8].choose
              end
            end
          end
        end
      end
    end
    sleep 8 if one_in(12)
  end

  live_loop :cello do ###############
    x = (1.0/8)*rrand(1,7)
    with_fx :slicer, phase: [0.25,0.5,0.5,0.5,0.75,1,1,1,2].choose, slope_down: [0,0,0.25,0.5].choose do
```

```
      with_fx :pitch_shift, pitch: rrand(-8,8), window_size: rrand(0.0001,0.001) do
        sample :mySamples__cello, amp: rrand(5,7.7), start: x, finish: x + (1.0/8), rate: [-0.5,0.5].choose,
attack: 0.1, release: 0.2
      end
      x = (1.0/16)*rrand(1,15)
      with_fx :pitch_shift, pitch: rrand(-24,8), window_size: rrand(0.001,0.01) do
        sample :mySamples__cello, amp: rrand(5,7.7), start: x, finish: x + (1.0/16), rate: [-1,1,1].choose,
attack: 0.1, release: 0.2
      end
      x = (1.0/32)*rrand(1,31)
      with_fx :pitch_shift, pitch: rrand(-16,8), window_size: rrand(0.001,0.01) do
        sample :mySamples__cello, amp: rrand(5,7.7), start: x, finish: x + (1.0/32), rate: [-1,1,1].choose,
attack: 0.05, release: 0.1
      end
      sleep [1,2,4].choose
    end
  end
  sleep 8 #delay start
  live_loop :texture1 do ###############
    with_fx :pitch_shift, windows_size: rrand(0.01,0.1), pitch: [-32,-24,16,-12,-8,-4].choose do
      with_fx :pitch_shift, windows_size: rrand(0.01,0.1), pitch: [-32,-24,-16,-12,-8,-4].choose do
        with_fx :krush, res: (ring 0.01,0.2,0.01,0.3)[tick/16] do
          with_fx :wobble, phase: [0.5,1,2,4,4,8,8,16,16,32,64,128].choose do
            use_synth :tb303
            play (([60,64,68].choose) - ([8,16,24,32].choose)), attack: 0, sustain: 2, release:
[6,14].choose, amp: 1, pan: 0.75
          end
        end
      end
    end
    sleep 4
    sleep 8 if one_in(12)
  end
  sleep 8 #delay start
  live_loop :texture2 do #############
    with_fx :pitch_shift, windows_size: rrand(0.01,0.1), pitch: [-32,-24,16,-12,0,4,8,12].choose do
      with_fx :pitch_shift, windows_size: rrand(0.01,0.1), pitch: [-32,-24,-16,-12,0,4,8,12].choose do
        with_fx :krush, res: (ring 0.01,0.18,0.01,0.24)[tick/16] do
          with_fx :wobble, phase: [16,16,32,64,128,256].choose do
            use_synth :prophet
            play (([60,64,68].choose) - ([8,16,24,32].choose)), attack: 4, sustain: 2, release: 28, amp:
rrand(3,8), pan: -1
            use_synth :pulse
            play (([60,64,68,72].choose) + ([8,16,24,32].choose)), attack: 4, sustain: 2, release: 28, amp:
rrand(3,8), pan: -1
            if one_in(2)
              use_synth :beep
              8.times do
                play ( ([60,64,68,72].choose) + ([8,16,24,32].choose) ), attack: 0.1, sustain: 0.2, release:
0.1, amp: 3, pan: -1
                sleep 0.25
              end #for
            end #if
          end
        end
      end
    end
    sleep 2
    with_fx :pitch_shift, windows_size: rrand(0.01,0.1),
    pitch: [-32,-24,16,-12,0,4,8,12].choose do
      with_fx :pitch_shift, windows_size: rrand(0.01,0.1), pitch: [-32,-24,-16,-12,0,4,8,12].choose do
        with_fx :krush, res: (ring 0.01,0.18,0.01,0.24)[tick/16] do
          with_fx :wobble, phase: [4,8,8,16,16,32,64,128,256].choose do
            use_synth :prophet
            play ( ([60,64,68].choose) - ([8,16,24,32].choose) ), attack: 4, sustain: 2, release: 28, amp:
```

```
rrand(2,5), pan: +1
            use_synth :pulse
            play ( ([60,64,68,72].choose) + ([8,16,24,32].choose) ), attack: 4, sustain: 2, release: 28,
amp: rrand(2,5), pan: +1
            if one_in(7)
              use_synth :beep
              8.times do
                play ( ([60,64,68,72].choose) + ([8,16,24,32].choose) ), attack: 0.1, sustain: 0.2, release:
0.1, amp: 10, pan: 0.9
                sleep 0.25
              end #for
            end #if
          end #wobble
        end #krush
      end #pitch
    end #pitch
    sleep 16
    sleep 8 if one_in(12)
  end
  sleep 4 #delay start
  live_loop :clap do ##############
    with_fx :echo, phase: 0.1, mix: rrand(0.5,0.7) do
      with_fx :krush, mix: [0.444,0.5].choose do
        sample :drum_snare_hard, rate: rrand(2,2.1), amp: 0.84
        sample :drum_snare_soft, rate: rrand(1,1.1), amp: 1.2
      end
    end
    sleep (ring 8,4)[tick_clock/4]
  end
  sleep 18 #delay start

  live_loop :beep do ##############
    with_fx :echo, phase: (ring 2,0.5,0.25,0.5,3)[tick_clock/4] do
      with_fx :pitch_shift, window_size: [0.01,0.01,0.01,0.02,0.04,0.001].choose, pitch:
[-16,-8,0,8,12,24].choose do
        sample :bd_pure, rate: [4,8].choose, amp: 3
        sample :bd_pure, rate: rrand(5,10), amp: 1.333
        use_synth :noise
        play 60, attack: 0.1, release: rrand(0.7,0.9), pan: [-0.75,0.75].choose, pan_slide:
[0.1,0.25,0.5,1].choose, amp: rrand(0.55,0.60) *0.3
      end
    end
    sleep (ring 8,2,2,16,8,8,4,2,2,2,1,0.5,0.25,32)[tick_clock/64]
    use_synth :beep
    8.times do
play [:g1,:c2,:c3,:g4].choose,amp:[0,0.4,0.6,0.7,0.8,0.9,0.7].choose+0.54,attack:0.1,release:[0.1,2].choose
play chord(:c2, :minor).choose,amp:[0,0.4,0.6,0.7,0.8,0.9,0.7].choose+0.57,attack:0.1,release:[0.1,2].choose
      sleep 0.5
    end
    8.times do
      play [:g1,:g2,:g3,:g4].choose, release: 2, amp: [0,0.4,0.6,0.7,0.8,0.9,1.0].choose + 0.333, attack:
0.1, release: [0.1,2].choose
      sleep 0.5
    end
    sleep (ring 8,2,2,16,8,8,4,2,2,2,1,0.5,0.25,32)[tick_clock/64]
    sleep 8 if one_in(12)
  end #beep
  sleep 8

  live_loop :hat do ###############
    with_fx :lpf, cutoff: [120,125,130].choose, cutoff_slide: [0.5,1,2].choose, mix: rrand(0.3,0.6) do
      use_synth :cnoise
      with_fx :krush, mix: rrand(0.4,0.6) do
        play 60, attack: 0.01, sustain: 0.5, release: (ring 2,1.8,2,0.2,2,1.8,2,0.8,0.8,0.8)[tick_clock],
amp: (ring 0.07,0.09,0.11)[tick_clock/2] + rrand(0.2,0.444)
```

```
      end #lpf
    end
    sleep (ring 16,8,4,4,4,4,4,4,4,4,4,4)[tick_clock/4]
    sleep 8 if one_in(12)
  end #hat
end #compressor
sleep 8.5 #offbeat from hat #############

live_loop :kick do
  with_bpm 72/2 do
  y = (ring
0.5,0.25,0.5,0.25,1.0/8,0.5,0.25,0.25,0.25,1.0/16,1.0/16,1.0/8,1.0/4,1.0/8,1.0/[8,4].choose)[k1/2]
    with_fx :krush, mix: rrand(0.5,0.7) do
      with_fx :echo, phase: (ring 0.25,0.5,2,16,4,8)[k1], decay: 1, mix: 0.6 do
        sample [:bd_808,:bd_fat].choose, rate: rrand(0.93,0.98), amp: 2.2
        sample :bd_haus, rate: rrand(0.9,0.94), amp: rrand(1.4,1.6) * 1.3
        sleep y/2
        sample :bd_fat, rate: rrand(0.93,0.98), amp: 2.0
        sample :bd_gas, rate: rrand(0.9,0.94), amp: rrand(0.8,1.0) if one_in(4)
        sleep y/2
        sleep 1 if one_in([6,9].choose)
        k1 = k1 + 1
      end
    end
  end
  sleep 8 if one_in(12)
end #kick
```

15. "my mothers were angels with fire veils and feather wings"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-mother.mp3

www.freesound.org/people/firnwald/sounds/84162
www.freesound.org/people/qubodup/sounds/183994

```
# Alexandre rANGEL "my mothers were angels with fire veils and feather wings" v14
# 16-Apr-2016 / Sonic Pi 2.10

use_bpm 140
set_volume! 0.777
t = Time.new; x = (t.year / t.month / t.day * t.hour * t.min * t.sec)
use_random_seed x; puts x

sample_kick = 'C:/samples//kick_mpc.wav'
sample_switch = 'C:/samples/switch.wav'
what_sound = 1

live_loop :metro do
  clock = tick(:metro)
  bar = clock / 4; puts "bar : #{bar}"
  puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
  what_sound = rrand_i(1,5) if one_in(rrand_i(1,2))
  sleep 1
end #metro

lfo_start = 60
c = 0
lfo = 0
live_loop :bass do
  lfo = lfo + (ring +0.5,-0.5)[c/120] #lfo from 60 to 120
  c = c + 1
  with_fx :slicer, smooth: 0.5, phase: 4 do
    with_fx :hpf, cutoff: (lfo_start+lfo) do
sample :bass_trance_c, attack: 0.1,
rate: rrand(0.85,1.05)/[1,2].choose, amp: rrand(2.5,2.7) *[1.6,1.8].choose
  end
end
  sleep 2
end
##########################################################################################
s1 = 1
live_loop :switch do
if what_sound = 1 or what_sound = 4
if one_in(2)
  with_fx :compressor, slope_above: 0.48 do
    #with_fx :echo, phase: (ring 0.5,0.5,1,2,2,2,4,4)[s1/2], mix: [0.66,0.75].choose do
        with_fx :gverb, room: [1,1,1,1,1,2,2,4,8].choose, mix: rrand(0.22,0.5) do
          s1=s1+2
          with_fx :slicer, phase: [0.25,0.25,0.25,0.5,0.5,0.5,0.5,0.5,1].choose do
            with_fx :pitch_shift, pitch: rrand(-2,2), window_size: rrand(0.0001,0.01) do
              sample sample_switch, start: 0.966, finish: 0.968,
                amp: rrand(2.5,3.333) *0.666, pan: rrand(-1,1) #if one_in(3)
            end
          end
          #end
        end
```

```
52        end
53      end
54    end
55    sleep 2 #[0.5,0.5,0.5,1].choose
56  end
57  ##########################################################################################
58  sleep 24
59  with_fx :compressor, slope_below: 1.3, slope_above: 0.7, mix: 0.7 do
60  ##########################################################################################
61    k2 = 0
62    live_loop :kick2 do
63      with_fx :ixi_techno, res: rrand(0.1,0.9), phase: rrand(0.02,0.24),
64      cutoff_min: rrand(1,99), mix: 0.5 do
65        with_fx :distortion, distort: rrand(0.2,0.7), mix: 0.4 do
66          sample :drum_tom_lo_hard, start: rrand(0,0.99), rate: rrand(0.96,0.97),
67            amp: (ring 0.9,1.0)[k2] *2.5
68          sample :drum_heavy_kick, rate: 0.9, amp: (ring 0,0.4,0,0.6)[k2/3] *[1.3,1.5].choose
69        end
70      end
71      sleep 1.0 / 2
72      sleep 4 if one_in(24)
73      sleep 8 if one_in(36)
74      k2=k2+1
75    end # kick2
76  ##########################################################################################
77    sleep 15
78    k1 = 0
79    live_loop :kick do
80      with_fx :ixi_techno, res: rrand(0.2,0.5), phase: 0.1, cutoff_min: 30, mix: 0.5 do
81        with_fx :distortion, distort: rrand(0.2,0.7), mix: 0.7 do
82          sample sample_kick, start: 0.97, rate: rrand(0.94,0.96),
83            amp: (ring 0.9,0.7)[k1] * 0.8
84          sample :bd_haus, rate: 0.9, amp: (ring 0,0.4,0,0.6)[k1/3] * 0.8
85        end
86      end
87      sleep (ring 2,2,2,1)[k1] / 2.0
88      sleep 4 if one_in(32)
89      sleep 8 if one_in(48)
90      k1=k1+1
91    end # kick
92  ##########################################################################################
93    sleep 8
94    b1 = 0
95    live_loop :beep1 do
96      if what_sound = 2 or what_sound = 4 or  what_sound = 5
97        if one_in(2)
98          with_fx :slicer do
99            with_fx :level, amp: (ring 0.8,0,0.8,0.4)[b1/4] do
100             b1=b1+1
101             with_fx :flanger, phase: (ring 1,0.1,2,0.2,0.3,0.4,0.5,1,2)[b1/24] do
102               4.times do
103                 sample :elec_blip, amp: rrand(1.4,1.6)
104                 sleep (ring 0.2,0.2,0.4,0.2,0.4,0.4,0.2,0.2) [x] * 2
105                 x = x + 1
106               end
107               2.times do
108                 sample [:elec_blip,:elec_blip2].choose, amp: rrand(1.3,1.6)
109                 sleep (ring 0.2,0.2,0.4,0.2,0.4,0.4,0.2,0.2) [x] * 2
110                 x = x + 1
111               end
112             end
```

```
113          end
114        end
115      end
116    end
117    sleep 2
118  end
119 ##########################################################################################
120  live_loop :beep2 do
121    if what_sound = 3 or what_sound = 5
122      if one_in(2)
123        puts 'go'
124        with_fx :octaver, mix: [0,1].choose do
125          with_fx :wobble, phase: [0.5,1,2].choose do
126            with_fx :vowel, vowel_sound: rrand_i(1,5), voice: 2 do
127              my_scale = scale([:f,:c].choose, :melodic_minor_asc)
128              permutations = my_scale.permutation(5)
129              use_synth [:chiplead,:chipbass].choose
130              permutations.each do |notes|
131                play_pattern_timed(notes,1.0/2)
132                play_pattern_timed(notes,1.0/1)
133              end
134            end
135          end
136        end
137      end
138    end
139    sleep 2
140  end
141
142 end #compressor
143 ##########################################################################################
144 sleep 8
145 live_loop :hat do
146   sample :drum_cymbal_pedal, finish: rrand(0.2,0.6), amp: 0.7
147   sleep 0.5
148   sleep 4 if one_in(36)
149 end
```

16. "enoch"

Linha **99**: As opções do comando de espera, quando colocadas em uma estrutura de anel (***ring***), são escolhidas sequencialmente e, quando esgotadas, são repetidas.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-enoch.mp3

www.freesound.org/people/j1987/sounds/178836

```
01 # Alexandre rANGEL "enoch" v12
02 # 22 April 2016 / Sonic Pi 2.10
03
04 use_random_seed 3.14
05 x = ; y = 0 ; d3= 0
06
07 use_bpm 48 #(96)
08 set_volume! 0.79
09
10 sample_metal = 'C:/samples/metal_box.wav'
11
12 live_loop :metro do
13   clock = tick(:metro); bar = clock / 4
14   puts "bar : #{bar}"; puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
15   sleep 1
16 end #metro
17 ##########################################################################################
18 with_fx :compressor, slope_above: 0.8, slope_below: 1.333, mix: 0.7 do
19 ##########################################################################################
20   live_loop :synth1 do
21     use_synth :beep
22     with_fx :whammy, mix: rrand(0.3,0.7) do
23       with_fx :slicer, phase: [1.0/3,0.25,0.1].choose, mix: 0.75 do
24         with_fx :echo, phase: 0.25, mix: 0.66 do
25           with_fx :bitcrusher, bits: [4,6,8,10,12].choose do
26             play_pattern [:f2,:c1,:f3], release: rrand(2,5),
27               amp: rrand(0.2,0.6), pan: 1, rate: 0.5, mod_range: [12,24,36,48,64,128].choose
28             sleep [0.25,0.5,1,2].choose
29           end
30         end
31       end
32     end
33     sleep 2
34     #sleep 100
35   end
36 ##########################################################################################
37   live_loop :synth2 do
38     with_synth :dsaw do
39       with_fx :slicer, phase: [1.0/3,0.25,0.1,0.5,0.75,0.9].choose,
40       mix: rrand(0.75,1.0) do
41         with_fx :echo, phase: 0.25, mix: 0.66 do
42           with_fx :bitcrusher, bits: [4,6,8,10,12].choose do
43           play_pattern [:c2,:c1,:e2], release: rrand(2,5), mod_range: [12,24,36,34].choose,
44               amp: rrand(0.10,0.16), pan: 1, rate: [0.2,0.5,0.8].choose
45             sleep [0.25,0.5,1,2].choose
46           end
47         end
48       end
```

```
    end
    sleep 2
  end
##########################################################################################
live_loop :synth3 do
  sleep 3
  with_synth :pulse do
    with_fx :slicer, phase: [1.0/3,0.25,0.2,0.5,0.75,0.9].choose, mix: rrand(0.75,1.0) do
      with_fx :distortion, mix: 0.66 do
        with_fx :bitcrusher, bits: [4,6,8,10,12].choose do
        play_pattern [60+rrand_i(0,12),60+rrand_i(0,30),60+rrand_i(0,12)],release: rrand_i(1.0,7.0),
          rate: 0.5,mod_range: [3,4,6,12].choose,amp: rrand(0.01,0.12),pan: 1
        sleep [1,2,3,5,6,6,8].choose
        end
      end
    end
  end
  sleep 2
end
##########################################################################################
m = 0
live_loop :metal do
  m = m + (1.0/32)
  with_fx :gverb, mix: rrand(0.1,0.5) do
    with_fx :whammy, transpose: rrand(-16,16), grainsize: rrand(0.01,0.1),
    mix: rrand(0.01,0.477) do
      with_fx :slicer, phase: [0.25/2,0.25,0.5,1,2].choose do
        sample sample_metal, amp: rrand(3,6), pan: -1, start: rrand(0,0.99),
          finish: rrand(0,0.99), rate: rrand((ring 0.5,0.25,1)[m.to_int],1.5)*[-1,1].choose
        sleep 1.0/2
        sleep [0,0,0,0,2,2,2,4,4,8].choose
      end
    end
  end
end
##########################################################################################
live_loop :drums1 do
  with_fx :gverb, mix: rrand(0.16,0.18) do
    sample :drum_bass_hard, amp: 1.3, pan: -1, rate: rrand(0.97,0.99)
    sample :bd_fat, amp: rrand(0,0.5), pan: -1, rate: rrand(0.97,0.99)
  end
  sleep (ring 0.25,0.25,0.25,0.25,0.25,0.25,0.50) [x] * [2,2,4].choose
  x = x + 1
  sleep [2,4,8,0,0].choose
end
##########################################################################################
live_loop :drums2 do # hat
  with_fx :echo, mix: rrand(0.3,0.65), phase: 1.0/[1,2,3,4,5].choose do
    sample :drum_cowbell, amp: rrand(0.5,1.6), pan: -1, rate: rrand(6.47,9.53)
  end
  sleep (ring 0.25,0.25,0.25,0.25,0.25,0.25,0.50) [y]
  y = y + 1
  y = y + 1 if one_in(7)
  sleep [4,2,0,0].choose
end
##########################################################################################
  live_loop :drums3 do
    d3 = d3 + (1.0/32)
    sample :bass_trance_c, amp: [0.4,0.5,0.6,0.7,0.8].choose, pan: -1,
      rate: rrand((ring 1.0,0.5)[d3.to_int],1.53)
    sleep 1.0/2
```

```
    sleep [0,0,2,4].choose
  end
##########################################################################################
end #compressor
##########################################################################################
live_loop :drums4 do
  with_fx :echo, delay: 0.125, mix: 0.2 do
    with_fx :ixi_techno, phase: rrand(0.10,0.2), res: rrand(0.1,0.7), mix: rrand(0.4,0.8) do
      sample :drum_cymbal_pedal, amp: 0.15, rate: rrand(0.35,0.70), pan: -1
    end
    sleep 1.0/4
  end
  sleep 4
end
```

17. "bast"

Linhas **72** a **74**, **85** e **99**: Estrutura de bifurcação de acordo com o valor aleatório da variável **x**.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-Bast.mp3

www.freesound.org/people/SpankMyFilth/sounds/215491
www.freesound.org/people/LG/sounds/101401

```
01 # Alexandre rANGEL "Bast" v06
02 # 1 May 2016 / Sonic Pi 2.10
03
04 set_sched_ahead_time! 2
05 use_random_seed 3.14
06 y = 0; d3= 0
07 use_bpm 144
08 set_volume! 0.8
09
10 sample_pim = 'C:/samples/pim.wav'; sample_bottle = 'C:/samples/bottle.wav'
11
12 bar = 0
13 live_loop :metro do
14   clock = tick(:metro); bar = clock / 4
15   puts "bar : #{bar}"; puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
16   sleep 1
17 end #metro
18 ##########################################################################################
19 p1 = 0
20 live_loop :perc1 do
21   cue :metro
22   if bar > 16
23     use_synth :pulse
24     play 18, attack: (ring 2,3,4)[p1], sustain: 1, release: (ring 4,3,2)[p1],
25     pan: rrand(-0.18,0.18), pan_slide: 1, amp: [0.25,0.5,0.75,1,1.25,1.5,1.75].choose * 0.333
26   end
27   sleep 4 / 1.5
28   p1 = p1 + 1
29 end
30 ##########################################################################################
31 live_loop :pulso do
32   cue :metro
33   use_synth :dsaw
34   with_fx :echo, phase: 2 do
35     play 24, attack: (ring 2,3,4)[tick], sustain: 1, release: (ring 4,3,2)[tick],
36       pan: rrand(-0.18,0.18), pan_slide: [0.5,1,2,4,8].choose,
37       amp: rrand([0.222,0.333].choose,[0.555,0.666].choose)
38   end #fx
39   sleep 4
40 end
41 ##########################################################################################
42 e1 = 0
43 live_loop :eim do
44   use_synth :growl
45   play 18+(ring 36,48,60)[(e1/3.0).to_int],
46     pan: rrand(-0.25,0.25), pan_slide: 2, amp: rrand(0.9,1.222),
47     attack: (ring 2,3,4)[e1], sustain: 1, release: (ring 4,3,2)[e1]
48   sleep 4
49   e1 = e1 + 1
```

```
50  end
51  ##########################################################################################
52  live_loop :kick do
53    if bar > 22
54      sample :bd_haus, amp: 2.0
55    end
56    sleep 1
57  end
58  ##########################################################################################
59  live_loop :kick2 do
60    if bar > 28
61      with_fx :slicer, phase: [0.25,0.5,1,2].choose, mix: [0,0.5,1].choose do
62        with_fx :echo, phase: 1.5 do
63          sample :sn_dub, rate: 1, amp: rrand(1.3,1.5) *0.6, pitch: [-1,0,0,0].choose
64          sample :sn_dub, rate: rrand(0.38,0.4), amp: 0.9 *rrand(0.666,0.7)
65        end
66      end
67    end
68    sleep 4
69  end
70  ##########################################################################################
71  live_loop :bottle do
72    x = (rrand_i(1,3))
73    case x
74    when 1
75      with_fx :slicer, phase: [0.25,0.5,1].choose do
76        with_fx :echo, phase: [2,4,8].choose, reps: 2 do
77          with_fx :pitch_shift, pitch: rrand(-6,4), window_size: rrand(0.001,0.003) do
78            with_fx :gverb do
79              sample sample_pim, amp: rrand(1.0,1.5), rate: [1.0/8,0.25,0.5,1].choose
80            end
81          end
82        end
83      end
84      sleep [1,2,4,8,16].choose
85    when 2 then
86      with_fx :wobble, phase: [1.0/8,0.25,0.5,1].choose, wave:[0,1,2,3].choose, smooth: 0.1 do
87        with_fx :vowel, vowel_sound: [1, 2, 3, 4, 5].choose do
88          with_fx :slicer, phase: [0.25,0.5].choose do
89            with_fx :pitch_shift,pitch: rrand(4,12), window_size:rrand(0.001,0.003) do
90              with_fx :gverb do
91                sample sample_bottle, amp: rrand(1.0,1.1) * 1.5,
92                  rate: [-1,-0.5,0.5,1].choose, start: rrand(0.03,0.05)
93              end
94            end
95          end
96        end
97      end
98      sleep [2,2,2,2,4,4,4,8,8,16].choose
99    when 3
100     sleep 2
101   end #end case
102 end
```

18. "mother gave me wings"

Linha **7**: O comando ***load_samples*** carrega na memória do computador os arquivos de *samples* especificados, para acesso mais rápido do que a partir do disco do computador.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-mother.mp3

www.freesound.org/people/dwareing/sounds/187775

```
01 # Alexandre rANGEL "mother gave me wings" v05
02 # 8 May 2016 / Sonic Pi 2.10
03
04 use_bpm 60
05 set_volume! 0.8
06 sample_birds = 'C:/samples/birds.flac'
07 load_samples [sample_birds,:drum_cowbell,:bd_fat,:bd_klub,:drum_cowbell,:drum_tom_lo_soft]
08
09 r1 = 0  # ring iterator
10 r2 = 0  # ring iterator
11 x = 0
12 y = 0
13 ##########################################################################################
14 p = 0
15 live_loop :metro do
16   clock = tick(:metro)
17   bar = clock / 4
18   puts "bar : #{bar}"
19   puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
20   p = rrand(-0.7,0.7)
21   sleep 1
22 end #metro
23 ##########################################################################################
24 b = 0
25 d = sample_duration sample_birds
26
27 live_loop :birds do
28   sample sample_birds, rate: (ring 0.25/2, 0.25, 0.5, 1)[b],
29     attack: (ring 4,2,1,1)[b], release: 1, amp: (ring 28, 18, rrand(13,15), 9)[b]
30   sleep (ring 1, d*4, d*2, d*2)[b]
31   b = b + 1
32 end
33 ##########################################################################################
34 sleep 6
35 ##########################################################################################
36 live_loop :myNotes1 do
37   use_synth :tb303
38   if one_in(2)
39     play_pattern_timed [:c2,:d2,:e2], [1.0], cutoff: rrand(70,110),
40       cutoff_slide: 0.3, amp: 0.75, sustain: 2.0, release: 2.0, attack: [0,1.5,0.5,1,2].choose
41   else
42     play_pattern_timed [:c2,:e2,:d2], [1.0], cutoff: rrand(70,110),
43     cutoff_slide: 0.3, amp: 0.75, sustain: 2.0, release: 2.0, attack: [0.5,1,0.25].choose
44   end
45   sleep rrand_i(0,1)
46 end
47 ##########################################################################################
48 sleep 2
```

```
49  ##########################################################################################
50  live_loop :myNotes2 do
51    use_synth :square
52    with_fx :ring_mod, mix: rrand(0.5,1.0), freq: rrand(10,50), freq_slide: 0.3 do
53      if one_in(4)
54        with_fx :echo do
55          play_pattern_timed [:f4,:g4], [1.0], attack: 0.01, release: 0.2, amp: rrand(0.7,0.8)
56        end
57      else
58        play_pattern_timed [:f4,:g4], [1.0], attack: 0.01, release: 0.2, amp: rrand(0.7,0.9)
59      end
60    end
61    sleep rrand_i(0,8)
62  end
63  ##########################################################################################
64  sleep 2
65  ##########################################################################################
66  live_loop :cowbellA do
67    with_fx :echo, phase: 0.5, reps: 1, mix: 0.7 do
68      with_fx :gverb, mix: 0.7 do
69        18.times do
70          ampVal = (ring 0.0,0.01,0.0,0.02,0.04,0.06,0.07,0.10) [r1.to_int]
71          if ampVal > 0.0
72            ampVal = ampVal + rrand(0.0,0.05)
73          end
74          sample :drum_cowbell, amp: ampVal, pan: p, pan_slide: 0.05
75          sleep 1.0/3
76        end
77      end
78    end
79    r1 = r1 + 0.75
80    sleep 6 if one_in(5)
81  end
82  ##########################################################################################
83  sleep 8
84  ##########################################################################################
85  live_loop :cowbellB do
86    12.times do
87      ampVal = (ring 0.0,0.0,0.0,0.02,0.04,0.06,0.08,0.10) [r2] *0.75
88      ampVal = ampVal + rrand(0.10,0.24) if ampVal > 0.0
89      end
90      sample :drum_cowbell, amp: ampVal, pan: -p
91      sleep 1.0/3
92    end
93    r2 = r2 + 1
94    sleep 6 if one_in(5)
95  end
96  ##########################################################################################
97  sleep 8
98  ##########################################################################################
99  live_loop :mySynth1 do
100   with_fx :bitcrusher, bits: [6,7,8,9,10,11,12,13,14,15,16].choose do
101     with_fx :flanger, phase: [0.25,0.5,1.0].choose do
102       use_synth :beep
103       2.times do
104         play_pattern_timed [:c2,:c2,:c1,:c2,:c2,:c1],
105           0.25, amp: 0.87, release: [0.5,1.0].choose, pan: -1 if one_in(2)
106         play_pattern_timed [:c2,:c2,:c1,:c2,:c2,:c1],
107           0.25, amp: 0.87, release: [0.5,1.0].choose, pan: -1 if one_in(2)
108         play_pattern_timed [:e2,:e2,:f1,:e2,:e2,:f2],
109           0.25, amp: 0.87, release: [0.5,1.0].choose, pan: -1 if one_in(6)
```

```
110         sleep (ring 0.25,0.25,0.25,0.5) [x]
111         x = x + 1
112       end
113     end
114   end
115   with_fx :slicer, phase:[0.1,0.2,0.25,0.25,1.0/3,0.5].choose,phase_slide:[0.1,0.2,0.5,0.5].choose do
116     2.times do
117       use_synth :beep
118       play_pattern_timed [:e1,:e1,:e1,:e2,:e2,:e1],
119         0.25, amp: 0.87, release: [0.3,0.7].choose, pan: -1 if one_in(2)
120       play_pattern_timed [:e1,:e1,:e1,:e2,:e2,:e1],
121         0.25, amp: 0.87, release: [0.3,0.7].choose, pan: -1 if one_in(2)
122       play_pattern_timed [:g1,:g1,:g1,:g2,:g2,:f2],
123         0.25, amp: 0.87, release: [0.3,0.7].choose, pan: -1 if one_in(6)
124       sleep (ring 0.25,0.25,0.25,0.5) [y]
125       y = y - 1
126     end
127   end
128 end
129 ##########################################################################################
130 sleep 4
131 ##########################################################################################
132 live_loop :drums1 do
133   sample :bd_haus, amp: rrand(1.38,1.78) * 1.1
134   sample :bd_fat, amp: rrand(1.38,1.78) * 1.1
135   sleep 0.5
136 end
137 ##########################################################################################
138 sleep 4
139 ##########################################################################################
140 live_loop :drums2 do
141   sleep 1
142   sample :bd_klub, amp: rrand(2.65,2.95) * 1.1
143   sample :drum_tom_lo_soft, amp: rrand(2.65,3.05) * 1.1
144 end
145 ##########################################################################################
146 sleep 4
147 ##########################################################################################
148 live_loop :myDrums3 do
149   with_fx :compressor do
150     with_fx :echo, phase: 1.0/8 do
151       with_fx :bitcrusher, bits: [8,9,10,11,12,13,14,15].choose do
152         sample :elec_cymbal, amp: rrand(0.0,0.04)
153       end
154     end
155   end
156   sleep 1.0/1
157 end
```

19. "a new bark in town"

Linhas **122** e **123**: Dois *samples* de bateria (bumbos) tocam sempre ao mesmo tempo, porém resultando em sons diferentes, graças aos parâmetros aleatórios de volume de cada som.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-new-bark.mp3

www.freesound.org/people/jesabat/sounds/119725

```
01 # Alexandre rANGEL "a new bark in town" v01
02 # 15 May 2016 / Sonic Pi 2.10
03
04 use_bpm 48
05 sample_drawer = 'C:/samples/drawer.wav'
06 ##########################################################################################
07 puts "let's go!"
08 live_loop :metro do
09   clock = tick(:metro)
10   bar = clock / 4
11   puts "bar : #{bar}"
12   puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
13   sleep 1
14 end #metro
15 ##########################################################################################
16 live_loop :kick1 do
17   with_fx :echo do
18     sample :bd_klub, rate: 0.75, amp: 2.15
19   end
20   sleep 4
21 end
22 ##########################################################################################
23 sleep 2
24 ##########################################################################################
25 live_loop :kick2 do
26   with_fx :echo, mix: rrand(0.111,0.222) do
27     with_fx [:reverb].choose, mix: [0.4,0.5].choose do
28       sample :bd_haus, rate: 0.5, amp: rrand(0.8,0.9) * 1.1,
29         pan: rrand(-0.55555,0.55555), pan_slide: [0,0.25,0.5,1].choose
30     end
31   end
32   sleep 4
33 end
34 ##########################################################################################
35 sleep 1
36 ##########################################################################################
37 b = 0
38 live_loop :beep do
39   use_synth :beep
40   with_fx :slicer, phase: (ring 1, 0.25, 0.25, 0.5, 2)[b.to_int] do
41     with_fx :compressor, slope_below: 2.5, slope_above: 0.6, mix: 0.8 do
42       with_fx :pitch_shift, pitch: rrand(-4,4),
43         window_size: rrand(0.001,0.01), window_size_slide: [0,0.25,0.5].choose, mix: 0.75 do
44         with_fx :echo, phase: [0.25,0.5].choose, decay: [0.1,0.5,0.9].choose do
45           with_fx :pitch_shift, pitch: rrand(-6,8), window_size: rrand(0.0001,0.001),
46             window_size_slide: [0.1,0.5,1,2,4].choose, mix: 0.75 do
47             with_fx :flanger, phase: [0.25,0.333,0.5,1,2,3,4].choose, mix: rrand(0.5,1) do
48             with_fx :reverb, mix: 0.5 do
```

```
play [:F4,:G3,:F3,:G2,:F2].choose, attack: 1, release: 2, amp: rrand(0.37,0.444),
pan: rrand(-0.55555,0.55555), pan_slide: [0,0.25,0.5,1,2,3,4,8].choose
              end
            end
          end
        end
      end
    end
  end
  sleep 3
  b = b + (1.0/3)
end
##########################################################################################
sleep 1
##########################################################################################
c = 0
live_loop :prophet do
  use_synth :prophet
  with_fx :slicer, phase: (ring 0.25, 0.25/2)[c.to_int] do
    with_fx :compressor, slope_below: 2.5, slope_above: 0.6, mix: 0.8 do
      with_fx :pitch_shift, pitch: rrand(-8,8), window_size: rrand(0.0001,0.003),
        window_size_slide: [0,0.25,0.5].choose, mix: 0.75 do
        with_fx :echo, phase: [0.25,0.5].choose, decay: [0.3,0.5,0.7].choose do
          with_fx :pitch_shift, pitch: rrand(-6,8), window_size: rrand(0.0001,0.001),
            window_size_slide: [0.1,0.5,1,2,4].choose, mix: 0.75 do
            with_fx :flanger,phase: [0.25,0.333,0.5,1].choose, mix: rrand(0.5,1) do
              with_fx :reverb, mix: 0.5 do
play [:F4,:G3,:F3].choose, attack: 1, release: 2, amp: rrand(0.37,0.444), pan: rrand(-0.7,0.7),
pan_slide: [0,0.25,0.5,1,2,3,4,8].choose
              end
            end
          end
        end
      end
    end
  end
  sleep 3
  c = c + (1.0/3)
end
##########################################################################################
r1 = 0
live_loop :cowbell do
  with_fx :echo, phase: 0.5, decay: 0.4, mix: 0.7 do
    with_fx [:reverb,:gverb].choose, mix: rrand(0.5,0.67) do
      18.times do
        i = r1.to_int + [-1,-1,0,0,0,0,1,2,3].choose
        ampVal = (ring 0.3,0.01,0.3,0.02,0.03,0.05,0.06,0.08) [i]
        if ampVal > 0.0
          ampVal = ampVal * rrand(0.4444,0.55555) * 0.333
        end
        sample :drum_cowbell, amp: ampVal, pan: rrand(-0.7,0.7), pan_slide: 0.25
        sleep 1.0/3 * [2,1,3].choose
      end
    end
  end
  r1 = r1 + 0.75
  sleep 2 if one_in(4)
end
##########################################################################################
sleep 3
##########################################################################################
```

```
live_loop :kick3 do
  with_fx :level, amp: 0.75 do
    with_fx :echo, decay: [0.2,0.333,0.5,0.8].choose do
sample sample_drawer, amp: rrand(1.4,2.0)*0.5, start: rrand(0.04,0.045), finish: rrand(0.1,0.2),
  pan: rrand(-0.7,0.7), pan_slide: 0.2
    end
  end
  sleep 3.0
  sleep 6 if one_in(4)
end
##########################################################################################
live_loop :kick4 do
  sample :bd_haus, amp: rrand(1.7,3.4) * 0.1
  sample :bd_klub, amp: rrand(1.0,1.2) * 0.3
  sleep 0.75
end
```

20. "my scales"

Linha **16**: Mostra na janela de histórico do Sonic Pi (*log*) uma representação visual – um ponto – cada vez que o *loop* é executado.

Linhas **22**: A variável ***ns*** guarda todas as notas da escala chinesa, com a raiz na nota C2 e abrangendo de uma a três oitavas.

Linhas **23**: Escolhe e toca uma nota do grupo de notas ***ns***.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-myscales.mp3

www.youtube.com/watch?v=xLjPC4wLBig

```
01 # Alexandre rANGEL "my-scales" v08
02 # 21 May 2016 / Sonic Pi 2.10
03
04 t = Time.new
05 use_random_seed Math::PI
06 puts Math::PI
07 set_volume! 1.0
08 set_sched_ahead_time! 4
09
10 use_bpm 1.5
11
12 with_fx :tanh, mix: 0.777 do
13   with_fx :wobble, phase: 16, mix: 0.7 do
14 ##########################################################################################
15   live_loop :leadSlow do
16     puts "."
17     with_fx :pitch_shift, pitch: [-24,-18,-12,-8].choose,
18       pitch_slide: [2,4,6,8].choose, window_size: rrand(0.002,0.006) do
19       with_fx :bitcrusher, sample_rate: rrand(3000,16000), mix: 0.7 do
20         use_synth :dsaw
21         with_fx :reverb, room: rrand(0.01,0.99), mix: rrand(0.2,0.7) do
22 ns = (scale :c2, :chinese, num_octaves: rrand_i(1,3))
23 play ns.choose, detune: rrand(-72,72), attack: rrand(0.1,0.2), release: rrand(0.01,2),
24 cutoff: rrand(80, 110), cutoff_slide: 0.25, pan: rrand(-0.9,0.5), pan_slide: [0.25,0.5,1].choose,
25 amp: rrand(0.55,0.90), amp_slide: [0.5,1,2,4].choose
26           sleep 0.5
27         end
28       end
29     end
30   end
31 ##########################################################################################
32   live_loop :leadFast do
33     with_bpm 1.5/4.0 do
34       with_fx :pitch_shift, pitch: rrand(-12,2), window_size: rrand(0.001,0.007) do
35         with_fx :bitcrusher,sample_rate: rrand(7000,16000), mix: rrand(0.6,0.8) do
36           use_synth :zawa
37           with_fx :slicer,phase: 0.25/[0.25,0.5,1,2,4,8].choose, phase_slide: 1 do
38             with_fx [:whammy,:wobble].choose, phase: [1,2,4,8,16].choose do
39               ns = (scale :c1, [:whole,:whole_tone].choose, num_octaves: rrand_i(1,2))
40 play ns.choose, detune: rrand(-72,72), attack: rrand(1.05,1.75), release: rrand(0.01,2),
41 cutoff: rrand(100, 110), cutoff_slide: 0.25, pan: rrand(-0.5,0.9), pan_slide: [0.5,1,2].choose,
42 amp: rrand(0.55,0.90), amp_slide: [0.5,1,2,4].choose
43               sleep 0.25/[1,2,4,8].choose
```

```
              end
            end
          end
        end
      end
    end
  end
end
```

21. "i'm at home"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-at-home.mp3

www.freesound.org/people/junggle/sounds/29940
www.freesound.org/people/lex1975/sounds/114468
www.freesound.org/people/barcelonetasonora/sounds/186337

```
# Alexandre rANGEL "i'm at home" v18
# 29 May 2016 / Sonic Pi 2.10

use_bpm 96; use_random_seed Math::PI; set_sched_ahead_time! 4
set_volume! 0.66
sample_scratch = 'C:/samples/scratch.wav'
sample_train = 'C:/samples/train.wav'
sample_barceloneta = 'C:/samples/barceloneta.wav'
load_samples [sample_scratch,sample_train,sample_barceloneta,:bd_808,:bd_haus]
################################################################################
puts "let's go!"
live_loop :metro do
  clock = tick(:metro); bar = clock / 4; puts "bar : #{bar}"
  puts ring("1 |","2 | |","3 | | |","4 | | | |")[clock]; sleep 1
end #metro
################################################################################
sample sample_train, attack: 2, pan: -0.2, amp: 1.5; sleep 6
sample sample_barceloneta, start: 0.4448, attack: 6, release: 4, pan: 0.2, amp: 20
sleep 29
sample :bd_haus, amp: rrand(1,1.2) * [2.0,2.2].choose
with_fx :tanh, mix: 0.5 do
  live_loop :leadSlow1 do
    with_bpm 1.5 do
      with_fx :wobble, phase: [2,4,8].choose, mix: 0.7 do
        with_fx :pitch_shift, pitch: [-14,-12.5,-12,-11.5,-10].choose, window_size: rrand(0.002,0.006) do
          with_fx :bitcrusher, sample_rate: rrand(8800,15000), mix: 0.7 do
            with_fx :reverb, room: rrand(0.01,0.99), mix: rrand(0.2,0.7) do
              use_synth :dsaw
              ns =  scale(:c2, :chinese, num_octaves: rrand_i(1,3))
play ns.choose,detune: rrand(-72,72),attack: rrand(0.1,0.2),
release: rrand(0.01,2),cutoff: rrand(80, 110),cutoff_slide: [0.25,1].choose,
pan:rrand(-0.9,0.5),pan_slide:[0.25,0.5,1].choose,amp:rrand(0.55,0.90),amp_slide: [0.5,1,2,4].choose
              sleep 0.5
            end #reverb
          end #bitcrush
        end #pitch
      end #wobble
    end #bpm
  end #:leadSlow1
  sample :bd_haus, amp: rrand(1,1.2) * [2.0,2.2].choose
  sleep 4
  sample :bd_haus, amp: rrand(1,1.2) * [2.0,2.2].choose
################################################################################
  live_loop :leadFast1 do
    with_bpm 1.5/4.0 do
      with_fx :wobble, phase: 4, mix: rrand(0.333,0.9) do
      with_fx :pitch_shift, pitch: rrand(-12.6,1.6), window_size: rrand(0.001,0.007) do
      with_fx :bitcrusher,sample_rate: rrand(8800,16000),mix: rrand(0.3,0.9) do
      with_fx :slicer, phase: 1.0/[0.25,0.5,1,2,4,8].choose, phase_slide: [0.5,1,2].choose do
      with_fx :wobble, phase: [0.5,1,2,4,8,16].choose do
              use_synth [:chipbass,:zawa].choose
              ns = scale(:c1,[:whole,:whole_tone].choose,num_octaves: rrand_i(1,2))
play ns.choose,detune: rrand(-72,58),attack: rrand(1.05,1.75),release: rrand(0.01,2),cutoff:rrand(100,
110),cutoff_slide:[0.25,2].choose,pan: rrand(-0.5,0.9),
```

```
pan_slide:[0.5,1,2].choose,amp:rrand(0.55,0.80),amp_slide: [0.5,1,2,4].choose
              sleep 1.0/[1,2,4,8].choose
              end #:whammy
            end #:slicer
          end #:bit
        end #:pitch
      end #:wobble
    end #bpm
  end #:leadFast1
  sample :bd_haus, amp: rrand(1,1.2) * [2.0,2.2].choose
  sleep 2
  sample :bd_haus, amp: rrand(1,1.2) * [2.0,2.2].choose
##########################################################################################
with_fx :flanger, phase: [4,8,16].choose, mix: 0.333 do
with_fx :pitch_shift,pitch:[-4,-8,-12,-16,-20].choose,window_size:[0.001,0.01].choose,mix:0.25 do

live_loop :scratch do
with_fx :slicer, phase: [0.25,0.25,0.25,1].choose, mix: [1,0.5,1,1,1].choose do
with_fx :panslicer, phase: [0.25,0.5,1,1,1,2,2,2].choose, pan_min: [-0.77,-0.5,-0.3].choose, pan_max:
[0.3,0.5,0.77].choose do
with_fx :bitcrusher, bits: rrand(8,12), mix: 0.666666 do
with_fx :echo, phase: [0.25,0.5,0.5,0.5,0.75,1,2].choose, decay: [1.5/2,1.5].choose, mix: rrand(0.5,1.0),
reps: 2 do
with_fx :octaver, mix: rrand(0.4,0.88) do
sample sample_scratch, start: 0.01, finish: 0.10, rpitch: rrand(-8,2), rate: rrand(0.74,1.0) if one_in(2)
                  sleep 0.5
sample sample_scratch, start: 0.48, finish: 0.5, rpitch: rrand(-12,2), rate: rrand(-0.24,-0.26),
window_size: rrand(0.0001,0.001), amp: rrand(0.5,0.8) if one_in(5)
                  sleep 0.5
sample sample_scratch, start: 0.25, finish: 0.4, rpitch: rrand(-12,2), rate: rrand(0.9,1.1), amp:
rrand(0.5,0.8) if one_in(4)
                  sleep 0.5
sample sample_scratch, start: 0.48, finish: 0.5, rpitch: rrand(-8,2), rate: 0.25, window_size:
rrand(0.0003,0.0006), amp: rrand(0.5,0.8) if one_in(2)
                  #another sound:
                  x = rrand(0.001,0.9)
with_fx :vowel, voice: [1,2,3,4].choose, vowel_sound: [1,2,3,4,5].choose do
                    with_fx :wobble, phase: [0.5,1,2,4].choose do
sample sample_scratch,start: x,finish: x+0.1,rpitch: rrand(-16,2),rate: [0.25,0.25/2].choose,
window_size: rrand(0.0003,0.0006),amp: rrand(0.0,1.222) if one_in(2)
                    end #wobble
                  end #vowel
                  sleep 1
                end #octaver
              end #echo
            end #bit
          end #panslicer
        end #slicer
      end #:scratch
    end #pitch
  end #flanger
##########################################################################################
  with_fx :slicer, phase: 0.5, phase_offset: 0.5, smooth_down: [0.25,0.25,0.5,1].choose do
    live_loop :leadSlow2 do
      with_bpm 3 do
        x = [-16,-12,-8,-4,-2,0,2].choose
with_fx :pitch_shift,pitch: x,pitch_slide: [0.25,0.5,1,2].choose,window_size: rrand(0.0001,0.001) do
          with_fx :bitcrusher, sample_rate: rrand(8800,14000), mix: 0.7 do
            with_fx :gverb, room: rrand(7,27), mix: rrand(0.2,0.5) do
              use_synth :dsaw
              ns =  scale(:c2, :chinese, num_octaves: rrand_i(1,3))
play ns.choose, detune: rrand(-100,100),attack: rrand(0.05,0.1),release: 0.05,cutoff: rrand(80,
120),cutoff_slide: 0.25,pan: rrand(-1,0.2),pan_slide: [0.25,0.5,1].choose,amp: rrand(0.3,0.75),amp_slide:
[0.5,1,2,4].choose if one_in(3)
              sleep 2
```

```
            end #gverb
          end #bit
        end #pitch
      end #bpm
    end #:leadSlow2
##########################################################################################
live_loop :leadFast2 do
with_bpm 1.5 do
x = [-16,-12,-8,-4,-2,0,2].choose
with_fx :pitch_shift,pitch: x,pitch_slide:[0.25,0.5,1,2].choose,window_size:rrand(0.0001,0.001) do
with_fx :bitcrusher, sample_rate: rrand(6000,14000), mix: 0.7 do
with_fx :slicer,phase: 0.25/[0.25,0.5,1,2,4,8,16].choose,phase_slide: 1 do
with_fx [:whammy,:wobble].choose, phase: [1,2,4,8,16].choose do
use_synth [:zawa,:fm].choose
ns =  scale(:c1, [:whole,:whole_tone].choose, num_octaves: rrand_i(1,2))
play ns.choose,detune: rrand(-100,100),attack: rrand(1.05,1.1),release: 0.05,cutoff: rrand(100,
120),cutoff_slide: 0.25,pan: rrand(-0.2,1),pan_slide: [0.5,1,2].choose,amp: rrand(0.3,0.75),
amp_slide: [0.5,1,2,4].choose if one_in(3)
                sleep 0.25/[1,2,4,8].choose
              end #whammy
            end #slicer
          end #bit
        end #pitch
      end #bpm
    end #:leadFast2
  end #slicer
  sleep 38
  sample sample_barceloneta, attack: 2, release: 4, amp: 12, pan: 0.7
##########################################################################################
  k = 0 #initialize k
  live_loop :kick do
    with_fx :echo, reps: 2, phase:  ring(0.25,0.25,0.5,0.25,1)[k], mix: [0.2,0.3,0.4,0.5].choose do
      sample :bd_808, rate: ring(1, -1)[(k/2.0).to_int], amp:ring(0.222,1.6)[k]*2.4
      sample :bd_haus, amp: ring(rrand(1,1.2), rrand(1.4,1.6))[k] * [1.0,1.2].choose
    end
    k = k + 1
    sleep  ring(0.25,1,1,1,1,1,1,1,1,0.25,0.25,1,1,1,1,1,1,1,1,1,1,1,0.25,0.25,0.25)[k] *2
sample sample_barceloneta, amp: rrand(27,35)*1.333, start: 0.689, finish: 0.686 + 0.019, attack: 0.1,
release: 0.07, rate: rrand(0.222,1.3) if one_in(16)
sample sample_barceloneta, attack: 2, release: [2,4,6].choose, amp: rrand(8,15), pan: [-1.0,1.0].choose if
one_in(102)
  end #:kick
  sleep 120; sample sample_barceloneta, attack: 2, release: 4, amp: 12, pan: -0.7
end #tahn
```

22. "glitter"

Linha **8**: O comando ***set_sched_ahead_time*** configura o Sonic Pi para calcular a síntese sonora alguns segundos antes de sua execução, evitando que o programa engasgue por falta de poder de processamento do computador.

Linhas **13** e **14**: Este *sample*, fora de todos os *loops*, é executado em ordem linear (neste caso, no início da composição). O programa aguarda três segundos após o início do *sample* e então segue com a execução das instruções.

	www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-glitter.mp3
	www.freesound.org/people/alex@vsi.tv/sounds/170522
	www.youtube.com/watch?v=p-mLV22dABo Video 360°: www.youtube.com/watch?v=p-mLV22dABo

```
01 # Alexandre rANGEL "glitter" v19
02 # 5 June 2016 / Sonic Pi 2.10
03
04 use_bpm 72
05 t = Time.new; myt = (t.year * t.month * t.day * t.hour * t.min * t.sec); puts myt
06 use_random_seed myt
07 set_volume! 0.7
08 set_sched_ahead_time! 2
09
10 sample_woosh = 'c:/samples/woosh.wav'
11 load_samples [sample_woosh,:bd_klub,:bd_haus]
12
13 sample sample_woosh, amp: 2
14 sleep 3.0
15
16 with_fx :slicer, phase: 0.8, phase_offset: 0.0,smooth_up: 0.05,smooth_down: 0.05,mix:0.6 do
17   with_fx :normaliser, mix: 0.4444 do
18  with_fx :compressor, mix: 0.9, slope_below: 1.777, slope_above: 0.4444,
19     clamp_time: 0.05, threshold: 0.7 do
20       with_fx :echo, phase: 1.0/3, mix: 0.333 do
21 ##########################################################################################
22         live_loop :glitter1 do
23           with_fx :pitch_shift, pitch: [-1,-2].choose, window_size: rrand(0.0001,0.1) do
24             with_fx [:flanger,:ixi_techno,:compressor].choose,
25             phase: [2,4,8].choose, mix: 0.333 do
26               #slow
27               slowfactor = rrand(1,5)
28               with_fx :whammy, transpose: -4, mix: 0.5 do
29                 with_fx [:tanh,:wobble].choose, mix: rrand(0.01,0.333) do
30                 with_fx :pitch_shift, pitch: rrand(-8,4), window_size: rrand(0.0001,0.1) do
31 sample sample_woosh, rate: 1.0 / slowfactor, attack: rrand(4.2,12.4), release: rrand(4.2,8.4),
32 pan: rrand(-0.66,0.66), pan_slide: [0.25,0.5,1,2,4,6].choose, amp: 2.0 * (slowfactor/2.8)
33                   end
34                 end
35               end
36               sleep (((sample_duration sample_woosh) * (slowfactor)) * (1.0/3)).to_f
37               #fast
38               if one_in(3)
```

```
                slowfactor = rrand(5,20)
                with_fx :whammy, transpose: 4, mix: 0.5 do
                with_fx :panslicer, phase: [0.5,1,2,4].choose,smooth:[1,2].choose,mix:0.5 do
                  with_fx :octaver, mix: 0.5 do
                  with_fx [:krush,:tanh].choose, mix: rrand(0.01,0.333) do
                  with_fx :pitch_shift, pitch: rrand(-4,16), window_size: rrand(0.0001,0.1) do
sample sample_woosh, rate: 1.0 / slowfactor * [-1,1].choose, amp: 2.0 * (slowfactor/2.8),
attack: rrand(4.2,8.4), release: rrand(4.2,8.4), pan: rrand(-0.77,0.77), pan_slide:
[0.25,0.5,1,2,4].choose
                        end
                      end
                    end
                  end
                end
              end
              sleep (((sample_duration sample_woosh) * (slowfactor)) * (1.0/3)).to_f
            end
          end
        end
      end
      sleep 2
##########################################################################################
with_fx :echo, phase: 1.0/2, mix: 0.333 do
live_loop :glitter2 do
  with_fx :rbpf, centre: rrand(80,110),centre_slide:4,res:rrand(0.3,0.6),mix:0.333 do
  #slow
  slowfactor = rrand(1,6)
  with_fx :pitch_shift, pitch: [1,2].choose, window_size: rrand(0.0001,0.1) do
  with_fx [:ixi_techno,:flanger,:compressor].choose, phase: [2,4,8,16].choose, mix: 0.333 do
  with_fx :whammy, transpose: -8, mix: 0.5 do
  with_fx :tanh, mix: rrand(0.01,0.4444) do
  with_fx :pitch_shift, pitch: rrand(-8,4), window_size: rrand(0.0001,0.1) do
    sample sample_woosh, rate: 1.0 / slowfactor, attack: rrand(4.2,8.4), release: rrand(4.2,8.4),
      pan: rrand(-0.7,0.7), pan_slide: [0.25,0.5,1,2,4,6].choose, amp: 1.8 * (slowfactor/2.4)
  end
  end
  end
  sleep (((sample_duration sample_woosh) * (slowfactor)) * (1.0/3)).to_f
  #fast
  if one_in(3)
  slowfactor = rrand(6,21)
  with_fx :whammy, transpose: 8, mix: 0.5 do
  with_fx :panslicer, phase:[0.5,1,2,4].choose,smooth:[1,2].choose,mix:0.5 do
  with_fx :octaver, mix: 0.5 do
  with_fx :tanh, mix: rrand(0.01,0.4444) do
  with_fx :pitch_shift, pitch: rrand(-4,16), window_size: rrand(0.0001,0.1) do
    sample sample_woosh, rate: 1.0 / slowfactor * [-1,1].choose,
      attack: rrand(4.2,8.4), release: rrand(4.2,8.4),
      pan: rrand(-0.7,0.7), pan_slide: [0.25,0.5,1,2,4].choose, amp: 1.8 * (slowfactor/2.4)
                        end
                      end
                    end
                  end
                end
                sleep (((sample_duration sample_woosh) * (slowfactor)) * (1.0/3)).to_f
              end
            end
          end
        end
      end
```

```
    end
  end
end
sleep 20
sleep 0.8
##########################################################################################
k=0
live_loop :kick1 do
  ampkick = 1.3
  sample :bd_klub, amp: rrand(3.3,3.5) * ampkick
  sleep 1
with_fx :pitch_shift, pitch: rrand([-4,-2].choose,[2,4].choose), pitch_slide: [0.5,1].choose,
window_size: rrand(0.07,0.09), mix: ring(0.1,0.1,0.1,0.88888888)[k] do
with_fx :echo, phase: ring(0.25,0.333,0.25,rrand(0.02,0.2))[k],
decay: ring(0.5,0.4,0.5,[1,2,3,4,8].choose)[k], mix: ring(0.25,0.333,0.25,0.333)[k] do
      sample :bd_haus, amp: 1.73 * ampkick
    end
  end
  sleep 1
  k=k+1
end
##########################################################################################
live_loop :melody do
  with_bpm 15 do
    with_fx :echo, phase: rrand(0.1,1) do
      with_fx :flanger, phase: [0.1,0.2,0.5,1].choose do
        use_synth :chipbass
        play rrand(50,70), attack: 2, release: 0
        sleep 1
        use_synth :prophet
        play rrand(52,72), attack: 2, release: 0
        sleep 8 if one_in(6)
      end
    end
  end
end
sleep 1
##########################################################################################
k2=0
live_loop :kick2 do
  with_fx :echo, phase: 0.5, mix: ring(0,0.5,0,0.5,0,1)[k2] do
    with_fx :distortion, mix: rrand(0.4,0.6) do
      sample :bd_fat, amp: rrand(5,7), rate: rrand(0.9,1.0)
    end
  end
  sleep 2
end
sleep 0.2
##########################################################################################
h=0
live_loop :hat do
  use_synth :cnoise
  with_fx :echo, phase: 0.25 do
   play ring(60,60,60,80)[h], attack: 0.1, release: rrand(0.001,0.08),amp: rrand(1.9,2.1) * 0.8
  end
  sleep (ring 1.5,1.5,1.5,1)[(h/2).to_i]
  h=h+1
end
```

23. "what's this flux of thoughts I call I?"

Linha **4**: O comando ***set_volume*** controla o volume geral da composição; é um recurso prático de mixagem.

Linhas **18** e **19**: O comando ***set_mixer_control*** permite a aplicação de filtros na sonoridade geral da composição. Tais comandos, por terem a condição ***if*** no final da linha, só são executados se o valor aleatório preencher as condições estipuladas (***uma chance em [...]***)

Linha **97**: *Loop* com várias opções de velocidade. A repetição de valores dentro do comando de escolha aumenta a possibilidade de escolha do valor repetido.

Linhas **167**, **174** e **179**: Estes comandos play são influenciados, respectivamente, pela hora, pelo minuto, e pelo segundo do momento da execução da composição.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-thoughts.mp3

www.freesound.org/people/niqoplitnouk/sounds/220063
www.freesound.org/people/nicolasdrweski/sounds/179625

```
1  # Alexandre rANGEL "what's this flux of thoughts I call I?" v16
2  # 12 June 2016 / Sonic Pi 2.10
3
4  set_volume! 0.8
5  use_bpm 600
6
7  clock = 0
8  sample_metalhit = 'C:/samples/metalhit.wav'; sample_triangle = 'C:/samples/triangle.wav'
9  set_sched_ahead_time! 2
10 ##########################################################################################
11 live_loop :metro do
12   clock = tick
13   bar = clock / 4
14   puts "bar : #{bar}"
15   puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
16   set_mixer_control! lpf: [rrand(100,108),rrand(108,116)].choose,
17 lpf_slide: [2,4,8,16,32].choose if one_in(64*4)
18   set_mixer_control! hpf: 0, lpf_slide: [4,8,16,32].choose if one_in(32*4)
19   set_mixer_control! hpf: 90, hpf_slide: [8,16,32].choose if one_in(256*4)
20   sleep 1
21 end #metro
22 ##########################################################################################
23 live_loop :texture1 do
24   with_bpm 12 do
25     with_fx :flanger, phase: [1,2,4,8,16].choose, mix: rrand(0.2,0.5) do
26
27       use_synth :growl
28       play scale(:g2, :egyptian).choose, res: rrand(0.3,0.8), res_slide: [0.5,1,2,4].choose,
29         pan: rrand(-0.8,0.8), pan_slide: [1,2,3,6].choose,
30         attack: [2,4,8.16].choose, sustain: 2, release: 8, amp: rrand(1,5) *0.7 *0.2
31       sleep 2
32       use_synth :fm
33       play chord(scale(:g2, :egyptian).choose,:major), amp: rrand(1,5.1) *0.7 *0.2,
34         res: rrand(0.2,0.77), res_slide: [0.5,1,2,4,8].choose,
35         divisor: [[2,4].choose,rrand(0,4)].choose, divisor_slide: 4,
```

```
36          attack: [2,4,8,].choose, sustain: 2, release: [8,8,8,16].choos
37        sleep 2
38        play scale(:g3, :egyptian).choose,
39          res: rrand(0.2,0.71), res_slide: [0.5,1,2,4,8].choose,
40          divisor: [[2,4,8,12,16].choose,rrand(0,4)].choose, divisor_slide: 4,
41          attack: [2,4].choose, sustain: 2, release: [6,8,12,16].choose,
42          pan: rrand(-0.6,0.6), pan_slide: [1,2,3,6,12].choose,
43          divisor_slide: 4, amp: rrand(1,4) *0.7 *0.2
44        sleep 8
45      end
46    end
47  end
48  ##########################################################################################
49  with_fx :slicer, phase: 0.5, smooth: 0.01, mix: 0.7 do
50  with_fx :compressor, threshold: 0.7, slope_above: 0.7, slope_below: 1.05, mix: 0.6 do
51
52  ##########################################################################################
53  live_loop :linger do
54  cue :metro
55  myAmp = rrand(0.75,0.9) *1.5
56
57  with_fx :compressor, mix: 0.7 do
58    with_fx :panslicer, phase: [0.5,1,2,4,8,16].choose, smooth: [1,2,4,8].choose do
59      with_fx :pitch_shift, pitch: rrand(-8,0), window_size: rrand(0.001,0.01) do
60        with_fx :pitch_shift, pitch: [-8,-12,-16].choose, window_size: rrand(0.01,0.1) do
61          with_fx :ixi_techno, res: rrand(0.01,0.8), mix: rrand(0.001,0.7) do
62
63  use_synth :blade
64  ata = 0.01
65  rel = rrand(1.7,2.1) * 2.2
66  dif = rrand(0.01,0.08)
67  p = rrand(-0.4,0.4) #pan
68  ps = [4,8].choose #pan slide
69
70  play 60, attack: ata, release: rel + dif, amp: myAmp
71  sleep 0.02
72  play rrand_i(62,63), attack: ata, release: rel, amp: myAmp
73  sleep 0.02
74  play 67, attack: ata, release: rel, amp: myAmp
75  play 60, attack: ata*2, release: (rel + dif)*1.8, detune: [6,12,18].choose,
76    pan: p, pan_slide: ps, amp: myAmp/3.5
77  sleep 0.02
78  play rrand_i(62,63), attack: ata*2, release: (rel + dif)*1.8,
79    detune: [6,12,18].choose, pan: p, pan_slide: ps, amp: myAmp/3.5
80  sleep 0.02
81  play 67, attack: ata*2, release: (rel + dif)*1.8,
82    detune: [6,12,18].choose, pan: p, pan_slide: ps, amp: myAmp/3.5
83
84  sleep ( 8 - dif - dif)
85
86  end #fx
87  end #fx
88  end #fx
89  end #fx
90  end #fx
91
92  end #linger
93  ##########################################################################################
94  sleep 60
95  ##########################################################################################
96  live_loop :tum do
```

```
with_bpm 600 / [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,4,8].choose do

      with_fx :echo, reps: 2, phase: [rrand(0.001,0.03),0.5].choose, mix: rrand(0.1,0.5) do
          with_fx :krush, gain: rrand(1.2,2.5), mix: rrand(0.01,0.2) do
            sample :drum_bass_hard, amp: rrand(0.5,0.7) *1.1
            sleep 2
            sample :drum_bass_soft, amp: rrand(0.7,1.1) *rrand(1.1,1.3)
            sleep 1
            sample :drum_bass_hard, amp: rrand(0.7,1.1) *1.1
            sleep 1
          end
        end
      end
    end #tum
##########################################################################################
    sleep 120
    sleep 0.5
##########################################################################################
    live_loop :tss do
      with_fx :panslicer, phase: [1,2,4,8].choose, smooth: [1,2,4,8].choose do
        with_fx :pitch_shift, pitch: rrand(-8,0), window_size: rrand(0.001,0.01) do
with_fx :echo, reps: 2, decay: 4, phase: [rrand(0.001,0.03),0.5,3].choose, mix: rrand(0.1,0.222) do

            use_synth :pluck
play 60, attack: 0.01, sustain: 0, release: [4,6].choose, amp: rrand(0.6,0.7) if one_in(20)
            sleep [2,4].choose
            sleep 8 if one_in(72)
            sample sample_metalhit, rate:[0.5,1,1,1,1].choose, amp: 0.333
            sleep [2,4].choose
          end
        end
      end
    end #tss
##########################################################################################
  end #compressor
##########################################################################################
live_loop :texture2 do
  with_bpm 6 do
    with_fx :flanger, phase: [1,2,4,8,16,32].choose, mix: rrand(0.2,0.5) do
      use_synth :fm
      with_fx :pitch_shift, pitch: -8, window_size: rrand(0.001,0.01) do
        play scale(:g2, :spanish).choose, divisor: [2,4,8].choose, divisor_slide: 4,
          pan: rrand(-0.8,0.8), pan_slide: [1,2,3,6].choose,
          attack: 2, sustain: 2, release: 8, amp: rrand(2,5.2) * 0.7 * 0.1
        sleep 2
        use_synth :fm
        play chord(scale(:g2, :spanish).choose,:minor),
          divisor: [[2,4].choose,rrand(0,4)].choose, divisor_slide: 4,
            attack: [2,4].choose, sustain: 2, release: [8,8,8,16].choose,  amp: rrand(2,6) *0.7 * 0.1
        sleep 2
        play scale(:g2, :super_locrian).choose, divisor: [4,8].choose,
          attack: 2, sustain: 2, release: [6,8,12,16].choose,
          pan: rrand(-0.6,0.6), pan_slide: [1,2,3,6,12].choose,
          divisor_slide: 4, amp: rrand(1,5.3) * 0.7 * 0.1
        sleep 4
      end
    end
  end
end
##########################################################################################
sleep [0,0,0,0,1,2,3,4].choose
```

```
158 ##########################################################################################
159 live_loop :texture3 do
160   with_bpm 24 do
161     t = Time.new
162     use_synth :fm
163     with_fx :pitch_shift, pitch: -2, window_size: rrand(0.01,0.1) do
164       with_fx :vowel, voice: [1,2,3,4].choose, vowel_sound: [1,2,3,4,5].choose do
165         with_fx :pitch_shift, pitch: [1,2,4].choose, window_size: rrand(0.001,0.01), mix: 0.77 do
166           ### influenced by the hours:
167           play ((scale(:g2, :spanish).choose) +12 - t.hour),
168             divisor: [2,4,8].choose, divisor_slide: 4,
169             pan: rrand(-0.8,0.8), pan_slide: [1,2,3,6].choose,
170             attack: [2,4,8].choose, sustain: 2, release: 8, amp: rrand(2,5.4) * 0.12
171           sleep 4
172           ### influenced by the minutes:
173           use_synth :fm
174           play (chord(scale(:g2, :spanish).choose,:minor) -30 + t.min),
175             divisor: [[2,4].choose,rrand(0,4)].choose, divisor_slide: 4,
176             attack: [2,4].choose, sustain: 2, release: [8,8,8,16].choose, amp: rrand(2,6) * 0.12
177           sleep 2
178          ### influenced by the seconds:
179          play ((scale(:g3, :minor_pentatonic).choose) -30 + t.sec), divisor: [4,8].choose,
180          attack: 2, sustain: 2, release: [6,8,12,16].choose,
181          pan: rrand(-0.6,0.6), pan_slide: [1,2,3,6,12].choose, amp: rrand(1,5.22) * 0.12
182          sleep 2
183         end
184       end
185     end
186   end
187 end
188
189 end
```

24. "let's play this for 24 hours"

Linha **68**: Graças ao vínculo com o horário, essa composição será diferente a cada hora do dia.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-lets-play-this-for-24h.mp3

```
01 #Alexandre rANGEL "let's play this for 24 hours"
02
03 t = Time.new
04 tx = t.year * t.month * t.day * t.hour * t.min * t.sec
05 use_random_seed tx
06 puts 'the magic number now is... '
07 puts tx #2016-06-11 17:25:58 -> 3279830400
08 set_volume! 0.5
09 ##########################################################################################
10 live_loop :texture1 do
11   with_bpm 12 do
12     with_fx :flanger, phase: [1,2,4,8,16].choose, mix: rrand(0.2,0.5) do
13       use_synth :growl
14       play scale(:g2, :egyptian).choose, res: rrand(0.3,0.8), res_slide: [0.5,1,2,4].choose,
15         pan: rrand(-0.8,0.8), pan_slide: [1,2,3,6].choose,
16         attack: [2,4,8.16].choose, sustain: 2, release: 8, amp: rrand(1,5) *0.7
17       sleep 2
18       use_synth :fm
19       play chord(scale(:g2, :egyptian).choose,:major),
20         res: rrand(0.2,0.77), res_slide: [0.5,1,2,4,8].choose,
21         divisor: [[2,4].choose,rrand(0,4)].choose,
22         divisor_slide: 4, attack: [2,4,8,].choose, sustain: 2,
23         release: [8,8,8,16].choose, amp: rrand(1,5.1) *0.7
24       sleep 2
25       play scale(:g3, :egyptian).choose, res: rrand(0.2,0.71), res_slide: [0.5,1,2,4,8].choose,
26         divisor: [[2,4,8,12,16].choose,rrand(0,4)].choose, divisor_slide: 4, amp: rrand(1,4)*0.7,
27         attack: [2,4].choose, sustain: 2, release: [6,8,12,16].choose,
28         pan: rrand(-0.6,0.6), pan_slide: [1,2,3,6,12].choose
29       sleep 8
30     end
31   end
32 end
33 ##########################################################################################
34 live_loop :texture2 do
35   with_bpm 6 do
36     with_fx :flanger, phase: [1,2,4,8,16,32].choose, mix: rrand(0.2,0.5) do
37       use_synth :fm
38       with_fx :pitch_shift, pitch: -8, window_size: rrand(0.001,0.01) do
39         play scale(:g2, :spanish).choose, divisor: [2,4,8].choose, divisor_slide: 4,
40           pan: rrand(-0.8,0.8), pan_slide: [1,2,3,6].choose,
41           attack: 2, sustain: 2, release: 8, amp: rrand(2,5.2) * 0.7
42         sleep 2
43         use_synth :fm
44         play chord(scale(:g2, :spanish).choose,:minor), divisor: [[2,4].choose,rrand(0,4)].choose,
45           divisor_slide: 4, attack: [2,4].choose, sustain: 2,
46           release: [8,8,8,16].choose, amp: rrand(2,6) * 0.7
47         sleep 2
48         play scale(:g2, :super_locrian).choose, divisor: [4,8].choose,
49           attack: 2, sustain: 2, release: [6,8,12,16].choose,
50           pan: rrand(-0.6,0.6), pan_slide: [1,2,3,6,12].choose,
51           divisor_slide: 4, amp: rrand(1,5.3) * 0.7
52         sleep 4
```

```
      end
    end
  end
end
##########################################################################################
sleep [0,0,0,0,1,2,3,4].choose
#######################################################################################
live_loop :texture3 do
 with_bpm 24 do
   t = Time.new
   use_synth :fm
   with_fx :pitch_shift, pitch: -2, window_size: rrand(0.01,0.1) do
   with_fx :vowel, voice: [1,2,3,4].choose, vowel_sound: [1,2,3,4,5].choose do
   with_fx :pitch_shift, pitch: [1,2,4].choose, window_size: rrand(0.001,0.01), mix: 0.77 do
          ### influenced by the hours
          play ((scale(:g2, :spanish).choose) +12 - t.hour),
            divisor: [2,4,8].choose, divisor_slide: 4,
            pan: rrand(-0.8,0.8), pan_slide: [1,2,3,6].choose,
            attack: [2,4,8].choose, sustain: 2, release: 8,amp: rrand(2,5.4) * 1.2
          sleep 4
          ### influenced by the minutes:
          use_synth :fm
          play (chord(scale(:g2, :spanish).choose,:minor) -30 + t.min),
            divisor: [[2,4].choose,rrand(0,4)].choose,
            divisor_slide: 4, attack: [2,4].choose, sustain: 2,
            release: [8,8,8,16].choose, amp: rrand(2,6) * 1.2
          sleep 2
          ### influenced by the seconds:
          play ((scale(:g3, :minor_pentatonic).choose) -30 + t.sec),
            divisor: [4,8].choose, amp: rrand(1,5.22) * 1.2,
            attack: 2, sustain: 2, release: [6,8,12,16].choose,
            pan: rrand(-0.6,0.6), pan_slide: [1,2,3,6,12].choose
          sleep 2
        end
      end
    end
  end
end
```

25. "group echo"

Linhas **15** a **27**: A variável ***part*** é modificada de acordo com o compasso da composição, acompanhado pela variável ***playhead***. Esse controle permite a execução de trechos diferentes, de acordo com o desdobramento da música.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-group-echo.mp3

```
01 # Alexandre rANGEL 我的回應你 "group echo" v06
02 # 26 jun 2016 / Sonic Pi 2.10
03
04 use_bpm 128/2.0
05 set_volume! 0.65
06 time = Time.new
07 use_random_seed (187.5*time.hour) # influence the composition: throw a time based dice
08 playhead = 0; part = 0
09 d1 = 0; d2 = 0; d3 = 0
10
11 #parts#=[    0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10]
12 cue    =[    0,  32,  72, 116, 164, 198, 256, 312, 354, 412, 460]
13 ##########################################################################################
14 live_loop :clock do
15   playhead = playhead + (1.0 / 4)
16   case playhead
17     when (cue[0]..cue[1])  then part = 0
18     when (cue[1]..cue[2])  then part = 1
19     when (cue[2]..cue[3])  then part = 2
20     when (cue[3]..cue[4])  then part = 3
21     when (cue[4]..cue[5])  then part = 4
22     when (cue[5]..cue[6])  then part = 5
23     when (cue[6]..cue[7])  then part = 6
24     when (cue[7]..cue[8])  then part = 7
25     when (cue[8]..cue[9])  then part = 8
26     when (cue[9]..cue[10]) then part = 9
27   end
28
29   if one_in(10)
30     24.times do puts "part: " + part.to_s end
31   end
32   sleep 0.25 # 4 measures per beat
33 end
34 ##########################################################################################
35 live_loop :synth1 do
36   use_synth :pulse
37   #with_fx :slicer, phase: (ring 1,2,0.5,0.4,0.25,0.1)[part] do
38   with_fx :ring_mod, freq: rrand(40,80), freq_slide: 1.0/2, mix: [0.1,0.2,0.3].choose do
39     with_fx :echo, phase: 0.25 do
40 play_chord [20-12,25,27-rand(12)], attack: 1.5, sustain: 0.1, release: 1.5,  amp: (rrand(1.3,1.8))
41     end
42   end
43   #end
44   sleep 2
45 end
46 ##########################################################################################
47 with_fx :bitcrusher, bits: 3 do
48   live_loop :drum1b do
49     sample :bd_haus, amp: 2.5
```

```
    #sleep 1.5/2
    sleep 1.0/2
  end
end
##########################################################################################
live_loop :drums2b do
  if part > 1
    with_fx :distortion, mix: 0.5 do
      with_fx :compressor do
        d1 = d1 + 1
        d1s = (ring 1,0,1,0) [d1]
        d1s = (ring 1,1,1,1) [d1] if one_in(24)
        sample :bd_klub, amp: rrand(2.9,4.0) if d1s==1
        sample :bd_haus, amp: rrand(0.4,0.7), rate: rrand(-1.0,-1.05) if d1s==1
      end
    end
  end
  sleep 1.0/2
  time = Time.new
  sleep 1.5 if one_in(time.sec) # olha o breque (break it down)
end
##########################################################################################
live_loop :drums3b do
  if part > 2
    d3 = d3 + 1.0
    d3s = (ring 0,1,0,1) [d3.to_int]
    with_fx :echo, phase: (ring 0.5,0.5,0.25) [d3.to_int] do
      sample 'C:/samples/clap909.wav', amp: rrand(1.9,2.9),
        cutoff: (ring 60,80,100,80,100) [d3.to_int], res: rrand(0.4,0.6) if d3s==1
    end
  end
  sleep 1.0/4
end
##########################################################################################
live_loop :synth2 do
  use_synth :square
  if (part/3).even?
    with_fx :flanger, mix: rand(0.75) do
      with_fx :slicer, phase: (ring 0.25,0.5,1)[part] do
        temp = rand_i(48)
        play_chord [rrand_i(60,65)-temp,rrand_i(60,65)-temp,rrand_i(60,65)-temp],
          sustain: 4, release: 1, amp: 1.3
      end
    end
  end
  sleep 2
end
##########################################################################################
live_loop :synth3 do
  if part > 2 and (part/3).odd?
    use_synth :mod_saw
    with_fx :slicer do
      with_fx :distortion, mix: rrand(0.4,0.9) do
play chord(:F3, :minor).choose, amp: rrand(0.5,0.6), attack: 2, sustain: [1,1.5].choose, release: 2,
res:(rrand(0.2,0.6)),res_slide:[0.25,0.5].choose,cutoff:rrand(70,110), cutoff_slide:[0.5,1,2].choose
      end
    end
  end
  sleep 2
end
##########################################################################################
```

```
live_loop :synth4 do
  if part > 1 and part.even?
    use_synth :pulse
    with_fx :distortion, mix: rrand(0.3,0.5) do
play chord(:A3,:minor).choose,amp: 0.55,attack:2,sustain:[1,1.5].choose,release: [2,3].choose,
res:(rrand(0.2,0.6)),res_slide:[0.25,0.5].choose,cutoff:rrand(70,110),cutoff_slide: [0.5,1,2].choose
    end
  end
  sleep 2
end
##########################################################################################
live_loop :textura do
  play scale([:c4,:c2].choose,:spanish).choose, attack: [2,4].choose,
    release: [2,4].choose if one_in(6)
  sleep 1
end
##########################################################################################
live_loop :notes do
  use_synth :beep
  play [:c5,:f5].choose, attack: 0.1, release: 0.1, amp: 1.1 if one_in(6)
  sleep 0.5
end
##########################################################################################
live_loop :ghost do
  with_fx :slicer, phase: 0.1, mix: 0.5 do
    with_fx :pitch_shift, pitch: rrand(-12,-4) do
      with_fx :flanger, phase: 0.25 do
      sample :drum_cymbal_hard, amp: rrand(1.8,2.4) * 3, rate: rrand(-0.3,0.3) if one_in(6)
      end #fx
    end #fx
  end #fx
  sleep 2
end #loop
```

26. "mystic breath"

Linhas **43** a **62** e linhas **66** a **85**: Vários *loops* utilizando o mesmo *sample*, com efeitos (***fx***) e possibilidades de execução *(**if**)* diferentes.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-mystic-breath.mp3

```
01 # Alexandre rANGEL "mystic breath" v06
02 # 3-July-2016 / Sonic Pi 2.10
03
04 mistic = 'C:/samples/synth_mistic.wav'; load_sample mistic
05 water = 'C:/samples/water_rural.wav'; load_sample water
06 set_volume! 1.87
07
08 myStart = 0
09 live_loop :water do
10   8.times do
11     sample water, attack: 1, release: 1, amp: rrand(0.5,1.5) * 1.1,
12       start: myStart, finish: myStart+0.13
13     sleep [1,2].choose #sample_duration water
14   end
15   myStart = rand(0.86)
16   sleep 8
17 end
18 ##########################################################################################
19 with_fx :compressor,threshold:0.4,pre_amp:1.57,slope_below:1.5,slope_above:0.8,mix: 0.8 do
20 ##########################################################################################
21   with_fx :flanger, phase: 256, depth: 64, wave: 1, mix: 0.5 do
22     live_loop :synth1 do
23       with_bpm 12 do
24         use_synth :blade
25         with_fx :echo, phase: [0.5,2,4,8].choose, mix: 0.7 do
26           with_fx :slicer, phase: (ring 1,2,0.5,0.4,0.25,0.1).tick do
27           with_fx :ring_mod,freq:rrand(40,80),freq_slide:1.0/2,mix: [0.1,0.2,0.3].choose do
28               with_fx :echo, phase: 0.25 do
29                 play_chord [20-12,25,27-rand(12)], attack: 1.5, sustain: 0.1, release: 1.5,
30                   amp: (rrand(1.0,1.3)) * 0.244 if one_in(rand(7))
31               end
32             end
33           end
34         end
35         sleep 2
36       end
37     end
38   end
39 ##########################################################################################
40 with_fx :flanger, phase: 128, depth: 9, wave: 3, mix: 0.4444 do
41 ##########################################################################################
42 myRate = 0.1
43 live_loop :mistic1 do
44
45   with_fx :octaver do
46     sample mistic, rate: myRate, amp: 4.9 if one_in(3)
47   end
48
49   with_fx :pitch_shift, pitch: -4, window_size: rrand(0.001,0.005) do
50     sample mistic, rate: myRate, amp: 4.9 if one_in(3)
51   end
```

```
52
53    with_fx :pitch_shift, pitch: -8, window_size: rrand(0.005,0.010) do
54      sample mistic, rate: myRate, amp: 4.5 if one_in(2)
55    end
56
57    with_fx :pitch_shift, pitch: -12, window_size: rrand(0.010,0.020) do
58      sample mistic, rate: myRate, amp: 4.5 if one_in(2)
59    end
60
61    sleep sample_duration mistic, rate: myRate
62  end
63
64  ##########################################################################################
65  myRate2 = -0.2
66  live_loop :mistic2 do
67
68    with_fx :octaver do
69      sample mistic, rate: myRate2, amp: rrand(1,3) if one_in(rand(7))
70    end
71
72    with_fx :pitch_shift, pitch: rrand(3.9,4.1), window_size: rrand(0.01,0.05) do
73      sample mistic, rate: myRate2, amp: rrand(1,3) if one_in(rand(8))
74    end
75
76    with_fx :pitch_shift, pitch: rrand(7.9,8.1), window_size: rrand(0.05,0.1) do
77      sample mistic, rate: myRate2, amp: rrand(1,3) if one_in(rand(9))
78    end
79
80    with_fx :pitch_shift, pitch: rrand(11.9,12.1), window_size: rrand(0.1,0.5) do
81      sample mistic, rate: myRate2, amp: rrand(1,3) if one_in(rand(10))
82    end
83
84    sleep sample_duration mistic, rate: myRate2
85  end
86
87  end
88  ##########################################################################################
89  end #compressor
```

27. "robot confirmative"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-robot-confirmative.mp3

www.freesound.org/people/sfstorm/sounds/275577

```
# Alexandre rANGEL "robot confirmative" v03
# 10-July-2016 / Sonic Pi 2.11

robot = 'C:/samples/robot_confirmative.flac'; load_sample robot
live_loop :robot1 do ################################
  sample robot, rate: rrand(0.94,1.02); sleep 4
end
live_loop :robot2 do ################################
  sleep 0.5
  sample robot, rate: rrand(0.93,1.02) / 2; sleep 3.5
end
live_loop :robot3 do ################################
  sample robot, rate: rrand(0.92,1.0) / 4; sleep 8
end
live_loop :robot4 do ################################
  sample robot, rate: rrand(0.92,1.04) / 8, amp: 3; sleep 24
end
live_loop :robot5 do ################################
  with_fx :pitch_shift, pitch: rrand(3.9,4.1),window_size: rrand(0.001,0.002), mix: 0.6 do
    sample robot, rate: rrand(0.98,1.02) / 8, pan: [-1,1].choose, pan_slide: 2, amp: 3
  end
  sleep 24
end
live_loop :robot6 do ################################
  sleep 2
  with_fx :pitch_shift, pitch: rrand(-3.9,-4.1), window_size: rrand(0.004,0.007), mix: 0.6 do
    sample robot, rate:rrand(0.98,1.02)/8,pan:[-1,1].choose,pan_slide:[1,2].choose, amp:3
  end
  sleep 22
end
live_loop :robot7 do ################################
  with_fx :slicer, phase: 0.75 do
    with_fx :pitch_shift, pitch: rrand(-7.9,-8.1) * (ring 0.5,0.5,0.5,1).tick,
      window_size: rrand(0.004,0.007), mix: [0.4,0.8].choose do
      with_fx :octaver, mix: rrand(0.6,0.9) do
        with_fx :echo, phase: 0.25, mix: 0.8 do
        sample robot, rate: rrand(0.94,1.02), start:0.2, finish:0.24, amp:rrand(1.8,2.2)*0.4
        end
      end
    end
  end
  sleep 1
end
```

28. "freed"

Linha **147**: O sintetizador ***sound_in*** recebe a entrada de áudio do computador, que pode ser o som de um instrumento físico cabeado ou o som de um microfone.

	www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-freed.mp3
	www.freesound.org/people/lazr2012/sounds/169606 www.freesound.org/people/Freed/sounds/11257
	www.youtube.com/watch?v=hfr_lhlDXwA

```
01 # Alexandre rANGEL "freed" v05
02 # 17-July-2016 / Sonic Pi 2.11
03
04 set_volume! 1.2
05 munch = 'C:/samples/munching.flac'; load_sample munch
06 freed = 'C:/samples/freed.wav'; load_sample freed
07 ##########################################################################################
08 myBPM = 10
09 live_loop :metro do
10   with_bpm myBPM do
11     4.times do
12       clock = tick(:metro)
13       bar = (clock / 4) + 1
14       puts "bpm: #{myBPM} bar: #{bar}"
15       puts (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
16       sleep 1
17     end # 4 times
18     myBPM = [1,2,2,6,6,10,12,16,10,12,16,20,24,30].choose
19   end #bpm
20 end #metro
21 ##########################################################################################
22 live_loop :munch do
23   with_bpm myBPM do
24     4.times do
25       x = (ring 2,4,8).tick
26       with_fx :compressor,
27         slope_below: 1.6, slope_above: 0.8, threshold: 0.5, mix: 0.8 do
28         with_fx :octaver, mix: rrand(0.02,0.7) do
29           sample munch, rate: 1.0 / x, pan: [-1,-0.75,0.75,1].choose, pan_slide: 4, amp: 1.4
30           sleep 2
31           sample munch, rate: 1.0 / x / 2.0, pan: [-1,-0.75,0.75,1].choose * -1,
32             pan_slide: [2,4].choose, amp: rrand(1,3)
33           sleep 2
34           sample munch, rate: 1.0 / x / 4.0,
35             pan: [-1,-0.75,0.75,1].choose * -1, pan_slide: 2, amp: rrand(2.8,3.3)
36         end
37       end
38       sleep 4 * x
39     end # 4 times
40   end #bpm
41 end
42 ##########################################################################################
43 live_loop :freed do
44   with_bpm myBPM do
45     4.times do
```

```
      with_fx :compressor, slope_below: 1.444, slope_above: 0.8, threshold: 0.5, mix: 0.6 do
        with_fx :octaver, mix: rrand(0.01,0.7) do
          with_fx :echo, phase: rrand(0.1,0.3), mix: rrand(0.01,0.8) do
            with_fx :pitch_shift, pitch: [-24,-20,-16,-12,-8,-4,-2].choose,
            window_size: rrand(0.01,0.1) do
              with_fx :tanh, krunch: rrand(0,4), mix: rrand(0.5,0.8) do
                with_fx :whammy do
                  with_fx :pitch_shift, pitch: ((ring 0,0,-2,-2,-4).tick)-4,
                  window_size: rrand(0.002,0.01) do
                    with_fx :echo, phase: 1.0 / 3, mix: rrand(0,0.9) do
                      sample freed, pan: [-1,-0.75,0.75,1].choose, pan_slide: [0.5,1,2,4].choose
                    end
                  end
                  sample freed, rate: 1.0 / 8, start: rrand(0,0.9),
                  attack: 0.1, pan: [-1,-0.75,0.75,1].choose, pan_slide: [1,2].choose,amp: 3
                end
              end
            end
          end
        end
      end
      sleep 8
    end # 4 times
  end #bpm
end
##########################################################################################
with_fx :compressor, slope_above: 0.5, slope_below: 1.3, threshold: 0.5, mix: 0.7 do
  with_fx :ixi_techno, phase: 64, mix: 0.222 do
    live_loop :birds do
      with_bpm myBPM do
        4.times do
          with_bpm [3,6,12,30,60].choose do
            with_fx :pitch_shift, pitch: (ring 8,8,8.4,8.8,8,8,10,12).tick,
              pitch_slide: [0.2,0.2,0.4,1].choose, window_size: rrand(0.004,0.006) do
              with_fx :slicer, phase: [0.199,0.2,0.201].choose do
              with_fx :reverb, room: rrand(0.7,0.9), mix: rrand(0.2,0.4444), mix_slide: 1 do
                  with_fx :pitch_shift, pitch: rrand(-8,0), pitch_slide: [1.2,1.4,1.6,1.8].choose,
                    window_size: rrand(0.0002,0.2), mix: rrand(0.2,0.5) do
                    with_fx :flanger, phase: [8,9,12,16].choose, mix: rrand(0,1) do
                      with_fx :octaver do
                        with_fx :echo, phase: [3,6,9].choose do
                          with_fx :pitch_shift, pitch: rrand(0,8), pitch_slide: 1,
                          window_size: [0.001,0.002,0.01,0.1].choose do
                            with_fx :echo, phase: [0.1,0.2,0.25,0.5,1,2,3,4,6,8].choose do
                              with_fx :flanger, phase: rrand(0.5,3) do
                                with_fx :octaver do
                                  synth :sound_in, pan: [-1,1].choose,
                                    #pan: [-1,-0.9,-0.8,-0.7,0.7,0.8,0.9,1].choose,
                                    pan_slide: [0.25,0.5,1,2,4].choose,
                                    amp: rrand(1,1.8) * [0.6,0.75,0.9].choose
                                end
                              end
                            end
                          end
                        end
                      end
                    end
                  end
                end
              end
            end
```

```
            sleep [1,2,4,8].choose
          end
        end # 4 times
      end #bpm
    end
  end
end
##########################################################################################
# feeedback!
with_fx :compressor, slope_above: 0.5, slope_below: 1.3, threshold: 0.5, mix: 0.7 do
live_loop :fingers do
with_bpm 60 do
with_fx :pitch_shift, pitch: (ring 8,8,8.4,8.8,8,8,10,12).tick,
        pitch_slide: [0.2,0.2,0.4,1].choose, window_size: rrand(0.004,0.006) do
 with_fx :reverb, room: rrand(0.7,0.9), mix: rrand(0.2,0.4444), mix_slide: 1 do
  with_fx :pitch_shift, pitch: rrand(-8,0), pitch_slide: [1.2,1.4,1.6,1.8].choose,
          window_size: rrand(0.0002,0.2), mix: rrand(0.2,0.5) do
    with_fx :flanger, phase: [8,9,12,16].choose, mix: rrand(0,1) do
     with_fx :octaver do
      with_fx :echo, phase: [1,2,3,6,9,12].choose do
       with_fx :pitch_shift, pitch: rrand(0,8), pitch_slide: [0.5,1,2].choose,
               window_size: [0.0001,0.001,0.01,0.1].choose do
        with_fx :echo, phase: [0.2,0.25,0.5,1,2,3,4].choose do
         with_fx :ixi_techno, mix: 0.5 do
          with_fx :flanger, phase: rrand(0.2,6) do
           with_fx :octaver do
            synth :sound_in, pan: [-1,1].choose, amp: rrand(1,1.4444) * rrand(0.4,0.8)
                          end
                        end
                      end
                    end
                  end
                end
              end
            end
          end
        end
      end
      sleep [1,2,4].choose
    end
  end
end
```

29. "this is not a siren"

Linhas **12** a **24**: Trecho de código para recebimento de parâmetros MIDI a partir da leitura de arquivos de texto. Técnica detalhada no Capítulo 3, Seção 3.2.3.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-not-a-siren.mp3

www.freesound.org/people/modularsamples/sounds/280911
www.freesound.org/people/modularsamples/sounds/280911

```
01 # Alexandre rANGEL "this is not not a siren" v14
02 # Sonic Pi 2.11 / 24-July-2016
03
04 set_volume! 0.5; use_bpm 30; set_sched_ahead_time! 2
05 use_random_seed Time.new.sec * Time.new.min * Time.new.hour
06 akai = 'C:/samples/akai_note58.aiff'; load_sample akai
07 siren = 'C:/samples/dubsiren.wav'; load_sample siren
08 ##########################################################################################
09 x = 0
10 y = 2
11
12 ##| live_loop :midi do
13 ##|   with_bpm 120 do
14 ##|     k1f = File.read("/Users/rangel/knob1f.txt").to_f
15 ##|     k2f = File.read("/Users/rangel/knob2f.txt").to_f
16 ##|     k3f = File.read("/Users/rangel/knob3f.txt").to_f
17 ##|     k4f = File.read("/Users/rangel/knob4f.txt").to_f
18 ##|     k5f = File.read("/Users/rangel/knob5f.txt").to_f
19 ##|     k6f = File.read("/Users/rangel/knob6f.txt").to_f
20 ##|     k7f = File.read("/Users/rangel/knob7f.txt").to_f
21 ##|     k8f = File.read("/Users/rangel/knob8f.txt").to_f
22 ##|     sleep 1
23 ##|   end
24 ##| end
25 ##########################################################################################
26 k1f=k2f=k3f=k4f=k5f=k6f=k7f=k8f=rrand(0.001,1)
27 live_loop :variables do
28   k1f=k2f=k3f=k4f=k5f=k6f=k7f=k8f=rrand(0.001,1)
29   sleep 0.5
30 end
31 ##########################################################################################
32 live_loop :siren1 do
33
34   with_fx :slicer, phase: 0.25, smooth_up: 0.01, smooth_down: [0,0.1,0.2].choose do
35
36     if one_in(3)
37
38       with_fx :vowel, voice: [1,2].choose, voice_slide: 1,
39       vowel: [4,5].choose, vowel_slide: 1, mix: rrand(0.2,1.0) do
40         with_bpm ([1,2].choose * 2) do
41           4.times do
42             use_random_seed Time.new.sec * Time.new.min * Time.new.hour
43             position = (ring 0.1,0.18,0.16,0.22,0.24,0.3,0.32,0.38)[x/8]
44             with_fx :octaver, mix: rrand(0,1) *k1f do
45               with_fx :echo, phase: rrand(0.001,0.1), mix: rrand(0.2,0.6)*k2f do
46                 with_fx :pitch_shift, pitch: (ring -5,0,-5,-12,-5,-8)[x/2],
47                 window_size: rrand(0.01,0.2), mix: 0.7 *k3f do
48                   with_fx :slicer, phase: [0.25,0.5,1,1,1,2,2].choose do
```

```
                    with_fx :flanger, phase: [0.5,1,2,4,8,16,32,64].choose, phase_offset: rrand(0,1),
                    phase_offset_slide: [0.5,1,2].choose, mix: 0.7 *k4f do
                      with_fx :echo,phase: [0.05,1,2,4,8,16].choose,mix: 0.7 *k5f do
                        with_fx :octaver, mix: rrand(0,0.6) * k6f do
                          sample siren, start: position, finish: position + 0.1,
                            amp: rrand(0.5,2.2), amp_slide: [1,2].choose,
                            pan: rrand(-0.7,0.7), pan_slide: 2, rate: rrand(0.3,0.5)
                          position = rrand(0,0.799)
                sample akai, start: position, finish: position + 0.2,
                amp: rrand(2,3.6), amp_slide: [0.2,0.5,1,2].choose,
                pan: rrand(-0.7,0.7), pan_slide: 0.5, attack: [0.25,0.5,1].choose, rate: rrand(-1,1)
                        end
                      end
                    end
                  end
                end
              end
            end
            sleep [1,2].choose
            x = x + 1
          end
        end
      end

    else

with_fx :vowel, voice: [1,2,1,2].choose, voice_slide: 1,
  vowel: [4,5].choose, vowel_slide: 1, mix: rrand(0.2,0.9) do
with_bpm ([2,4,5,10].choose * 2) do
4.times do
  use_random_seed Time.new.sec * Time.new.min * Time.new.hour
  position = (ring 0.1,0.18,0.16,0.22,0.24,0.3,0.32,0.38)[y/8]
  with_fx :octaver, mix: rrand(0,1) do
  with_fx :echo, phase: rrand(0.001,0.1), mix: rrand(0.2,0.6) do
  with_fx :pitch_shift, pitch: (ring 0,-5,-12,-5,-8)[y/2], window_size: rrand(0.01,0.1), mix: 0.7 do
  with_fx :slicer, phase: [0.25,0.5,1,1,1,2,2].choose do
  with_fx :flanger, phase: [0.5,1,2,4,8,16,32,64].choose,phase_offset: rrand(0,1),
    phase_offset_slide: [1,2].choose, mix: 0.7 do
  with_fx :echo, phase: [0.05,1,2,4,8,16].choose, mix: 0.7 do
sample siren, start: position, finish: position + 0.1, amp: rrand(0.7,1.3),
  amp_slide: [0.5,1,2].choose, pan: [-1,-0.7,0.7,1].choose, pan_slide: 2, rate: rrand(0.3,0.5)
  position = rrand(0,0.799)
sample akai, start: position, finish: position + 0.2, amp: rrand(1,2.2),amp_slide: [0.5,1,2].choose,
  pan: [-1,1].choose, pan_slide: 0.5, attack: [0.25,0.5,1].choose, rate: rrand(-1,1)
      end
     end
    end
   end
  end
 end
end
    sleep [4,2,2].choose
    y = y + 1
  end
end

end #if
end
end
##########################################################################################
sleep 0.25
```

```
110 ##########################################################################################
111 live_loop :drums1 do
112 myVolume = rrand(0.333,0.4)
113 with_bpm 90 / [4,4,3,8,8].choose do
114   with_fx :slicer, phase: 0.05 do
115     with_fx :panslicer, phase: [0.1,0.2,0.2,0.5,1].choose, mix: 0.333 do
116       with_fx :echo, phase: 0.1, mix: 0.8 do
117         with_fx :krush, res: rrand(0,0.8), mix: rrand(0.222,0.4444) do
118           with_fx :reverb, mix: rrand(0.2,0.4) do
119             with_fx :flanger, phase: 4.0, depth: [5.0,10.0].choose do
120               sample :bd_klub, pan: rrand(-0.2,0.2), amp: 3.3 * myVolume
121               sleep 0.25
122               sample :bd_haus, rate: 0.9, pan: rrand(-0.2,0.2), amp: 3.5
123               sleep 0.25 * myVolume
124               sample :bd_haus, pan: rrand(-0.2,0.2), amp: 3.4 * myVolume
125               sleep 0.25
126               2.times do
127                 sample :bd_haus, rate: 1.1, pan: rrand(-0.2,0.2), amp: 3.3 * myVolume
128                 sleep 1.0/4
129                 sample :bd_haus,rate: 1.1,pan:rrand(-0.6,0.6),amp:3.5 * myVolume if one_in(3)
130                 sleep 1.0/4 if one_in(12)
131               end
132             end
133           end
134         end
135       end
136     end
137   end
138 end
139 end
```

30. "skeleton with flesh and light"

Linha **88**: *Sample* executado somente uma vez na composição, pois encontra-se fora das indicações de *loop*.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-skeleton.mp3

www.freesound.org/people/Snapper4298/sounds/176991
www.freesound.org/people/Snapper4298/sounds/176991

```
01 # Alexandre rANGEL "skeleton with flesh and light" v07
02 # 31-July-2016 / Sonic Pi 2.11
03
04 phone = 'C:/samples/phone.wav'; load_sample phone
05 hatopen = 'C:/samples/hat_open.wav'; load_sample hatopen
06 use_bpm 61
07 set_volume! 0.7
08 t = Time.new
09 use_random_seed (t.sec * t.min * t.hour)
10 ##########################################################################################
11 with_fx :panslicer, phase: 1.0/3, mix: 0.8 do
12   sample phone, amp: 0.74, release: 2
13 end
14 ##########################################################################################
15 sleep 3
16 ##########################################################################################
17 sample phone, start: 0.85, amp: 1.2
18 ##########################################################################################
19 with_fx :compressor, threshold: 0.333, slope_above: 0.5, mix: 0.6 do
20   with_fx [:flanger,:octaver].choose, phase: [0.5,1,2,4,8,16].choose,  mix: rrand(0.3,0.7) do
21     with_fx :slicer, phase: 0.25 do
22 ##########################################################################################
23 live_loop :notasB1 do
24   with_bpm 122 / 16.0 do
25   t = Time.new
26   use_random_seed (t.sec * t.min * t.hour)
27   use_synth :fm
28   with_fx :tanh, mix: rrand(0.1,0.2) do
29     with_fx :vowel, voice: rrand_i(0,4), vowel_sound: rrand_i(1,5), mix: rrand(0.3,1) do
30       with_fx :echo, phase: (1.0/3)*2, mix: 0.3 do
31   play scale(:c2, :chinese).choose, amp: rrand(0.1,0.175), divisor: [2,4,8,16].choose,
32   attack: [1.5,2].choose, pan: [-0.5,0.5].choose, pan_slide: [0.25,1,2].choose
33       end
34     end
35     sleep [1,1,3,3,6].choose * 2
36     end
37   end
38 end
39 ##########################################################################################
40 sleep 10
41 ##########################################################################################
42       live_loop :notasB2 do
43         with_bpm 122 / 32.0 do
44           t = Time.new
45           use_random_seed (t.sec * t.min * t.hour)
46           use_synth :fm
47           with_fx :vowel, voice: rrand_i(0,4), vowel_sound: rrand_i(1,5),mix: rrand(0.5,1) do
48             with_fx :tanh, mix: rrand(0.0,0.222) do
```

```
              with_fx :echo, phase: (1.0/3)*2, mix: 0.3 do
                play scale(:c2, :chinese).choose, amp: rrand(0.1,0.175),
                  divisor: [2,4,8,16].choose, depth: [0.5,1,2,4,8,16].choose,
                  depth_slide: [0.5,1,2,4].choose, attack: [1.5,2].choose,
                  pan: [-0.4,0.4].choose, pan_slide: 0.25
              end
            end
            sleep [1,1,3,3,6].choose * 2
          end
        end
      end

    end #fx
  end #fx
end #fx
##########################################################################################
sleep 16
sleep 8
##########################################################################################
sample phone, start: 0.9, amp: 1.3
##########################################################################################
live_loop :hat do
  x = [2,4,8].choose
  with_fx :slicer, phase: 0.25/[1,2].choose do
    sample hatopen, rate: 0.25, amp: rrand(2.8,3.0)
    if one_in(9)
      sample phone, rate: [-1,rrand(0.1,1.0/x)].choose, amp: 1.2 * x
    end
    sleep [2,4].choose
  end
end
##########################################################################################
sleep 16
sleep 4.5
##########################################################################################
sample phone, start: 0.6, amp: 1.1
##########################################################################################
live_loop :kick do
  sample :bd_haus, rate: rrand(0.96,0.98), amp: rrand(1.4,1.8)
  sample :bd_fat, amp: rrand(1.4,1.6) * (ring 2.5,1.6).tick
  sleep 0.5
end
```

31. "i trying to remember my dreams"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-remember.mp3

http://freesound.org/people/modularsamples/sounds/300877
http://freesound.org/people/Timbre/sounds/71341
http://freesound.org/people/Barruchi/sounds/348411

```
# Alexandre rANGEL "i trying to remember my dreams" v09
# 7-August-2016 / Sonic Pi 2.11

use_bpm 126
set_volume! 1.0
set_sched_ahead_time! 1
#let it roll...
expander = 'C:/samples/expander_E4.aiff'; load_sample expander
sirenfunk = 'C:/samples/siren_funk.wav'; load_sample sirenfunk
talk = 'C:/samples/talk_war.wav'; load_sample talk
##########################################################################################
live_loop :synth do
  sample expander, beat_stretch: 4, amp: rrand(3,4.4), amp_slide: 1,
    pan: rrand(-0.5,0.5), pan_slide: [0.5,1,2,4].choose
  sleep sample_duration expander, beat_stretch: 4
end
##########################################################################################
sleep 16
##########################################################################################
live_loop :synth2 do
  with_fx :bitcrusher, bits: [8,10,12,14,16].choose, sample_rate: rrand(13000,40000),
          mix: [0.2,0.4,0.88].choose do
    with_fx :octaver, sub_amp: 3, subsub_amp: 3, mix: [0.222,0.5,0.9].choose do
      with_fx :tanh, mix: 0.6 do
        with_fx :pitch_shift, pitch: [-4,-4,-2,-6,-8].choose, pitch_slide: [0.5,1].choose do
          with_fx :panslicer, phase: 1.0/[2,1,0.5].choose, mix: 0.8 do
            with_fx :slicer, phase: 1.0/[0.5,2,1].choose do
              with_fx :flanger do
                sample expander, beat_stretch: 8, amp: ring(2,5).tick * 5
              end
            end
          end
        end
      end
    end
  end
  sleep (sample_duration expander, beat_stretch: [4,6,6,8].choose) * [1,2].choose
end
##########################################################################################
sleep 16
##########################################################################################
myStart = 0.45
live_loop :sirenfunk do
  x = [0.2,0.25,1,1,2].choose
  with_fx :compressor, slope_above: 0.6, mix: 0.9 do
    with_fx :slicer, phase: x, mix: [0,0.4,0.6,1].choose do
      sample sirenfunk, start: myStart, beat_stretch: 96,
        attack: 0.16, release: 0.3, amp: 1.1, slope_down: [0,0.05].choose
    end
  end
```

```
  sleep (sample_duration sirenfunk, beat_stretch: 96) * 0.57 + (1.0/48)
  myStart = [0.45,0.42].choose
end
##########################################################################################
live_loop :sirenfunk2 do
  x = [0.2,0.25,1,1,2].choose
  with_fx :slicer, phase: x, phase_offset: x*[1,0.5].choose,
  mix: [0.25,0.5,1].choose do
    sample sirenfunk, start: myStart, pitch_stretch: 96,
      attack: 0.222, release: 0.3, amp: 2.3, window_size: [0.0001,0.001,0.01,0.1,0.5].choose
  end
  sleep (sample_duration sirenfunk, beat_stretch: 96) * 0.57 + (1.0/48)
end
```

32. "orchestra of roses and petals on an orange sky"

Linhas **36** a **38**: Nas primeiras 16 batidas do bumbo, a espera é de 4 compassos.

Linhas **41** a **43**: *Loop* com o bumbo tocado mais rápido; só é executado quando a sequência lenta anterior, fora do *loop*, termina sua execução.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-orange.mp3

https://freesound.org/people/quartzgate/sounds/178112
https://freesound.org/people/pwausc1/sounds/118272

```
01 # Alexandre rANGEL "orchestra of roses and petals on an orange sky" v04
02 # 13-08-2016 / Sonic Pi 2.11
03
04 use_bpm 126
05 wire = 'C:/samples/wire.wav'; load_sample wire
06 machine = 'C:/samples/answermachine2.wav'; load_sample machine
07 ##########################################################################################
08 live_loop :wire1 do
09   with_fx :pitch_shift, pitch: rrand(0,0.2), window_size: rrand(0.0001,0.5), mix: 0.6 do
10     sample wire, amp: 2.4; sleep 4
11   end
12 end
13 sleep 0.5
14 ##########################################################################################
15 live_loop :wire2 do
16   sample wire, rate: 2, amp: 2; sleep 2
17 end
18 sleep 0.25
19 ##########################################################################################
20 live_loop :wire3 do
21   sample wire, rate: 3, amp: 2; sleep 1
22 end
23 ##########################################################################################
24 live_loop :wire4 do
25   with_fx :pitch_shift, pitch: rrand(-8,2), window_size: rrand(0.0001,0.5) do
26     sample wire, attack: 0.55, finish: 0.3, rate: rrand(0.03,0.06)/[1,2].choose,
27        amp: rrand(31,33), pan: rrand(-0.3,0.3), pan_slide: [0.25,0.5,1,2].choose
28   end
29   sleep (ring 16,8,8,8,2,1,0.5)[tick/4]
30   end
31 ##########################################################################################
32   sample machine
33   16.times do
34     sample :bd_haus, amp: 2; sleep 4
35   end
36 ##########################################################################################
37   sample machine
38   live_loop :kick do
39     sample :bd_haus, amp: 2; sleep 1
40   end
```

33. "laugh or cry"

Linhas **16** e **44**: Início e fim de um efeito de compressão dinâmica, englobando três *loops*.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-cry.mp3

www.freesound.org/people/Nanakisan/sounds/132812
www.freesound.org/people/Anillogic/sounds/122579

```
1  # Alexandre rANGEL "laugh or cry" v03
2  # 21-August-2016 / Sonic Pi 2.11
3
4  use_bpm 136
5  scream = 'C:/samples/scream.wav'; load_sample scream
6  subbass = 'C:/samples/subbass.wav'; load_sample subbass
7  ##########################################################################################
8  live_loop :scream1 do
9    sample scream, start: 0.1, rate: 0.5, amp: 5
10   sleep 12
11   sample scream, start: 0.1, rate: 0.25, amp: 6
12   sleep [48,64,72].choose
13 end
14 sleep 40
15 ##########################################################################################
16 with_fx :compressor, slope_above: 0.5, mix: 0.7 do
17   live_loop :scream2 do
18     with_fx :slicer, phase: (ring 1,0.5,0.25).tick do
19       sample scream, start: 0.1, rate: [-0.25,0.25,0.25].choose,
20         pan: rrand(-0.5,0.5), pan_slide: [0.25,0.5,1,2].choose, amp: 8
21     end
22     sleep 24
23   end
24 ##########################################################################################
25   live_loop :drums do
26     x = tick
27     sample :bd_klub,          amp: ring(1,1,1,1) [x]
28     sample :drum_snare_soft,  amp: ring(0,0,0,0, 2,0,1,0)[x]
29     sleep 0.25
30   end
31 ##########################################################################################
32   live_loop :bass do
33     with_fx :slicer, phase_offset: [0.5,1].choose, phase: 0.5 do
34       with_fx :bitcrusher, sample_rate: rrand(10000,32000) do
35         with_fx :tanh do
36           with_fx :pitch_shift, pitch: [2,4,8].choose do
37             sample subbass, amp: 20
38           end
39         end
40       end
41     end
42     sleep [4,8,16].choose
43   end
44 end #compressor
```

34. "once upon a morning"

Linhas **136** a **144**: Sintetização da sonoridade de prato de bateria com uso de sintetizador de ruído modulado com notas musicais.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-once.mp3

www.freesound.org/people/csengeri/sounds/188765
(convert mp3 to wav)

```
01 # Alexandre rANGEL "once upon a morning"
02 # Sonic Pi 2.11 dev
03
04 thunder = "C:/samples/thunderstorm.wav"; load_sample thunder
05
06 use_bpm 46
07 set_sched_ahead_time! 2
08 ##########################################################################################
09 live_loop :drums do
10   with_fx :tanh, mix: rrand(0,0.2) do
11     with_fx :slicer, phase: 0.5/[1,2,4,8,2,4,8].choose do
12       with_fx :lpf, cutoff: rrand(100,120) do
13         sample :bd_haus, amp: 1.4
14       end
15       if one_in(6)
16         sleep 1.0/3
17       else
18         sleep 1.0/6
19       end
20       sample :bd_fat, amp: 3
21       if one_in(4)
22         sleep 1.0/3
23       else
24         sleep 1.0/6
25       end
26     end
27   end
28 end
29 ##########################################################################################
30 with_fx :compressor, threshold: 0.6, relax_time: 0.2, slope_below: 3.2, mix: 0.7 do
31 with_fx :reverb, mix: 0.5 do
32 ##########################################################################################
33 live_loop :thunder do
34   sample thunder, attack: 3, release: 6, start: 0.004, finish: 0.1, amp: 8
35   sleep (sample_duration thunder) / 10.0
36   sample thunder, attack: 4, release: 4, start: (1.0/3.0)*2, amp: 2
37   sleep ((sample_duration thunder)/3) * 2
38 end
39 ##########################################################################################
40 sleep 3
41 ##########################################################################################
42 live_loop :thunderLoop1 do
43 if one_in(2)
44  with_fx :lpf, cutoff: rrand(1,130), cutoff_slide: [1,2,3,4].choose, mix: 0.8 do
45   with_fx :hpf, cutoff: rrand(1,126), cutoff_slide: [1,2,3,4,8].choose, mix: 0.8 do
46    with_fx :slicer, phase: [0.25,0.5].choose do
47     sample thunder, attack: 0.1, release: 0.2, start: 0.004, finish: 0.014, rate: rrand(0.7,1.0),
```

```
                    pan: rrand(-0.2,-0.1), pan_slide: rrand(0.1,0.3), amp: rrand(0.6,0.9)
  end #fx
 end #fx
end #fx
sleep 4
else
 with_fx :hpf, cutoff: rrand(1,126), cutoff_slide: [1,2,3,4,8].choose, mix: 0.8 do
  with_fx :lpf, cutoff: rrand(1,130), cutoff_slide: [1,2,3,4].choose, mix: 0.8 do
   with_fx :slicer, phase: [0.25,0.5].choose do
    sample thunder, attack: 0.1, release: 0.2, start: 0.004, finish: 0.014, rate: rrand(0.5,1.0),
    pan: rrand(0.1,0.2), pan_slide: rrand(0.1,0.3), amp: rrand(0.6,0.9)
   end #fx
  end #fx
 end #fx
sleep 4

end #if
end #loop
##########################################################################################
with_fx :reverb, mix: 0.3 do
  live_loop :rocks1 do
    if rand(100) > 30
      use_synth :pnoise
      with_fx :hpf, cutoff: rrand(1,130), cutoff_slide: [1,2,3,4,8].choose do
        y = rrand(50,70);
        puts note_info(y)
        play y, attack: rrand([0.333,1].choose,2), release: rrand(6,8) * rrand(1,2),
                pan: rrand(-0.9,-0.3), pan_slide: rrand(1,3), amp: rrand(0.3,1.222) * 1.333
      sleep 6.0 * rrand(1,4)
      end #fx
    end #if
  end #loop
end #if
##########################################################################################
with_fx :reverb, mix: 0.7 do
  live_loop :rocks2 do
    sleep rrand(3,6)
    if rand(100) > 31
      use_synth :pnoise
      with_fx :lpf, cutoff: rrand(1,130), cutoff_slide: [1,2,3,4,8].choose do
      z = rrand(50,70)-30; puts note_info(z)
play z, attack: rrand([0.5,1].choose,2), release: rrand(6,8)*2  * rrand(1,2),
pan: rrand(0.3,0.9), pan_slide: rrand(1,3), amp: rrand(0.2,0.9) * 1.333
      end #fx lpf
    end #if
    sleep 9.0 * rrand(1,4)
  end #loop
end #fx reverb
##########################################################################################
with_fx :compressor, threshold:0.7, relax_time:0.05, slope_below:1.222, slope_above:0.7, mix:0.8 do
with_fx :reverb, mix: 0.5 do

live_loop :fish do
use_synth [:chipbass,:chipbass,:chiplead,:beep,:beep,:fm].choose
12.times do
 with_bpm 46/[0.5,1,2,4,8].choose do
  with_fx :hpf, cutoff: rrand(1,128), cutoff_slide: [0.25,0.5,1,2].choose do
   with_fx :lpf, cutoff: rrand(1,127), cutoff_slide: [0.25,0.5,1,2,3].choose do
    with_fx :panslicer, phase: [0.1,0.2,0.25,0.5,1].choose,
            pulse_width: rrand(0.01,0.09), mix: rrand([0,0.5].choose,1) do
     with_fx :echo, phase: [2,3,4,6].choose, decay: [3,6].choose do
```

```
      with_fx :gverb, mix: rrand(0.01,0.25) do
       with_fx :echo, phase: [0.25,0.25,0.5,1].choose do
        x = scale([[:g3,:g4].choose,:a3].choose,[:zhi].choose).choose
        puts note_info(x)
        play x, attack: 0.01, release: rrand(0.001,[0.05,0.1].choose),
          amp: rrand(0.777,1.4444)*rrand(1.0,1.4), pan: rrand(-0.333,0.333),
          pan_slide: [0.1,0.2,0.5,1].choose, divisor: [1,2,1,2,4,8,rand(16)].choose,
          depth: [2,4,2,4,2,4,8,rand(16)].choose
      end #fx
     end #fx
    end #fx
   end #fx
  end #fx
 end #fx
end #bpm
sleep [0.1,0.25,0.25,0.5,1,1,2].choose
end #times
sleep [4,8,12,16].choose
end #loop
end #fx
end #fx

end #reverb
end #compressor
##########################################################################################
sleep 4
##########################################################################################
live_loop :hat do
  use_synth :pnoise
  myRes = rrand(0,0.9)
  mySleep = [3,6,12].choose
  64.times do
    play 60, attack: 0.01, release: 0.01, amp: 0.4, res: myRes
    sleep 1.0/mySleep
  end
end
```

35. "are shadows faster than the light?"

Linha **13**: O comando ***use_random_seed*** modifica a semente de geração de números aleatórios. Ao mudar esse valor, cada vez que a composição é executada, é garantido que os números aleatórios não seguirão a mesma ordem entre as execuções. Nesse caso, a semente é modificada de acordo com o horário no qual a composição é iniciada (que será sempre diferente). A semente aleatória é detalhada no Capítulo 2.

Linhas **18** a **24**: Técnica de programação de sequências sonoras (sequenciador de bateria) desenvolvida nas oficinas *CODE MUZIK*, relatadas no Capítulo 2, Seção 2.3.1.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-shadows.mp3

```
1  #are shadows faster than the light?
2  #Sonic Pi 2.11
3
4  #satelites 18
5  #sounds from other dimensions
6  #are shadows faster than the light?
7  #we are moving so fast through space
8  #are are we leaving behind?
9  #do you hear the footsteps?
10
11 dir = 'C:/samples/sat'
12 set_sched_ahead_time! 2
13 use_random_seed Time.new.usec
14 puts Time.new.usec
15 set_volume! 0.6
16 use_bpm 14
17 ##########################################################################################
18   live_loop :drums do
19     sleep 0.05
20     x = tick
21     sample :bd_klub,          amp: ring(7,0,5,0)[x]
22     sample :bd_fat,           amp: ring(0,0,0,0, 5,0,3,0)[x]
23     sample :drum_snare_soft,  amp: ring(0,0,0,0, 2,0,0,0, 0,0,0,0, 1,0,0,0)[x]
24   end
25 ##########################################################################################
26   with_fx :compressor, slope_below: 0.5, slope_above: 0.3, threshold: 0.6 do
27     with_fx :slicer, phase: 0.25/2, smooth_down: 0.25, mix: 0.7 do
28       with_fx :pitch_shift, pitch: -6, pitch_slide: 0.5, window_size: 0.001, mix: 0.555 do
29         with_fx :flanger, phase: 48, phase_offset: 0.5, mix: 0.6 do
30           with_fx :ixi_techno, phase: 56, mix: 0.5 do
31             with_fx :flanger, phase: 16, mix: 0.4 do
32               with_fx :ixi_techno, phase: 24, mix: 0.4 do
33                 with_fx :echo, phase: 0.2, mix: 0.4444 do
34 ##########################################################################################
35 live_loop :satelites1 do
36 with_bpm [7,14,35,70,140,280,350,700,1400].choose do
37 x1 = rrand_i(1,96); y1 = dir, x1
38 with_fx :pitch_shift, pitch: (ring 2,4,2,0,0,-2).tick do
39  with_fx :octaver, subsub_amp: [0,1.4].choose, subsub_amp_slide: [0.5,1,2].choose, mix: 0.333 do
40   with_fx :pitch_shift, pitch: [6,12,18].choose, pitch_slide: 2, window_size: rrand(0.0001,0.001) do
41   rate1 = rrand(0.2,0.3)
42   sample y1, rate: rate1, amp: rrand(1,3),pan: [-0.75,0.75].choose, pan_slide:2, attack:1, release:1
```

```
  end #fx
 end #fx
end #fx
sleep (sample_duration y1) * 0.25
sleep (sample_duration y1) * 0.5 if one_in(3)
 end
end
sleep 2
##########################################################################################
live_loop :satelites2 do
with_bpm [7,14,35,70,140,280,350,700,1400].choose do
x2 = rrand_i(0,97); y2 = dir, x2
with_fx :pitch_shift, pitch: (ring 0,-4,0,-2,-2,-5).tick do
  with_fx :octaver, subsub_amp: [0,1.4].choose, subsub_amp_slide: [0.5,1,2].choose, mix: 0.333 do
    with_fx :pitch_shift, pitch: [6,12,18].choose, pitch_slide: [1,2,4].choose,
            window_size: rrand(0.0001,0.001) do
    rate2 = rrand(0.2,0.3)
sample y2, rate: rate2, amp: rrand(1,3),pan: [-0.75,0.75].choose, pan_slide: 2, attack:2, release:2
    end #fx
  end #fx
end #fx
sleep sample_duration y2, rate: 0.25
sleep (sample_duration y2) * 0.5 if one_in(3)
  end #bpm
end #loop
sleep 2
##########################################################################################
live_loop :satelites3 do
with_bpm [7,14,35,70,140,280,350,700,1400].choose do
x3 = rrand_i(1,98); y3 = dir, x3
with_fx :octaver, super_amp: [0,1.4,2.8].choose, super_amp_slide: [0.5,1,2].choose, mix: 0.333 do
  with_fx :pitch_shift, pitch: [6,12,18].choose, pitch_slide: [1,2,4].choose,
          window_size: rrand(0.0001,0.001) do
  sample y3, rate: rrand(0.9,1.03), amp: rrand(0.5,2.4),
             attack: 1, release: 1, pan: (ring -0.88,0.88).tick, pan_slide: 2
  end
end
sleep sample_duration y3, rate: rrand(0.9,1)
sleep sample_duration y3, rate: rrand(0.9,1) if one_in(3)
  end
end
sleep 2
##########################################################################################
live_loop :satelites4 do
with_bpm [7,14,35,70,140,280,350,700,1400].choose do
x4 = rrand_i(2,98); y4 = dir, x4
with_fx :octaver, super_amp: [0,1.4,2.8].choose, super_amp_slide: [0.5,1,2].choose, mix: 0.333 do
  with_fx :pitch_shift, pitch: [6,12,18].choose, pitch_slide: [1,2,4].choose,
          window_size: rrand(0.0001,0.001) do
sample y4, rate: rrand(0.9,1.05), amp: rrand(0.5,2.4),attack: 1, release: 1,
           pan: (ring -0.88,0.88).tick, pan_slide: 4
  end
end
sleep sample_duration y4, rate: rrand(0.9,1)
sleep sample_duration y4, rate: rrand(0.9,1) if one_in(3)
  end
end

              end #fx
            end #fx
          end #fx
```

```
        end #fx
      end #fx
    end #fx
  end #fx
end #fx
```

36. "waking up, finally"

Linhas **78** e **81**: Sons simultâneos, no mesmo *loop*, porém um em cada da lado do campo estéreo (esquerda e direita, ***pan*: -1** e ***pan*: +1**).

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-wakingup.mp3

www.freesound.org/people/modularsamples/sounds/279656
www.freesound.org/people/caculo/sounds/347040
www.freesound.org/people/thecityrings/sounds/174162

```
01 # Alexandre rANGEL "waking up finally" v05
02 # Sonic Pi 2.11 dev
03
04 set_volume! 1.1
05 set_sched_ahead_time! 3
06 sine = 'C:/samples/sine-b6-95.aiff'
07 rooster = 'C:/samples/rooster.wav'
08 rings = 'C:/samples/ring.wav'
09 ##########################################################################################
10 y = 0
11 live_loop :b6 do
12   with_fx :band_eq, freq: rrand(20,50), freq_slide: 2, db: rrand(1,2.4) do
13     s = sample sine, amp: 2, rate: 0.25 / 2
14     16.times do
15       control s, hpf: (ring 40,120)[y], hpf_slide: (ring 0.5,0.25,0.5,1)[y/4]
16       sleep 0.5
17       y = y + 1
18     end
19   end
20   sleep 16
21 end
22 ##########################################################################################
23 with_fx :compressor, slope_below: 5, slope_below: 0.7, threshold: 0.8 do
24   live_loop :rooster do
25     with_fx :pitch_shift, pitch: rrand(-6,0), window_size: rrand(0.001,0.01), mix: 0.5 do
26       with_fx :pitch_shift, pitch: rrand(-8,4), window_size: rrand(0.0001,0.01) do
27         with_fx :octaver, mix: 0.33 do
28           sample rooster,start:0.04,amp:1.333,rate:rrand(0.9,1.03),pan:-0.8 if one_in([3,2].choose)
29         end
30       end
31       with_fx :pitch_shift, pitch: rrand(-8,4), window_size: rrand(0.01,0.1) do
32         with_fx :octaver, mix: 0.33 do
33           sample rooster, start: 0.04, amp: 1.333,
34             rate: rrand(0.91,1.03), pan: 0.8 if one_in([2,3].choose)
35         end
36       end
37       with_fx :pitch_shift, pitch: rrand(-8,4), window_size: rrand(0.001,0.01) do
38         with_fx :octaver, mix: 0.66 do
39           sample rooster, start: 0.04, amp: 1.4, rate: rrand(0.9,1.1) * 0.5 * [-1,1].choose,
40             pan: [-1,1].choose, pan_slide: [1,2,4].choose if one_in(2)
41         end
42       end
43     end
44     sleep 8
45   end
46 end
```

```
47  ##########################################################################################
48  x = 0
49  live_loop :drums do
50    with_fx :level, amp: 0.9 do
51      sample :tabla_ghe2, amp: ring(2,0,1,0) [x]
52      #sample :tabla_ghe5, amp: ring(2,0,0,0, 2,0,2,0, 2,0,0,0, 2,2,2,1)[x] #sr
53      sample :tabla_dhec, amp: ring(0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0)[x] #ch
54      sample :tabla_ke3,  amp: ring(0,1,0,0, 0,1,0,1, 0,1,0,0, 0,1,1,1)[x], pan: -1 #oh
55      #sample :tabla_na_o, amp: ring(0,0,0,0, 0,0,0,0, 1,0,1,1, 0,0,0,0)[x] #cp
56      sample :tabla_tas2, amp: ring(0,0,1,1, 0,0,0,0, 0,0,0,0, 1,1,1,1)[x] #cp
57      #sample :tabla_te_ne,amp: ring(0,0,0,0, 1,2,1,2, 0,0,0,0, 0,0,0,0,)[x] #cp
58    end
59    x = x + 1
60    sleep 0.25
61  end
62  ##########################################################################################
63  live_loop :drums2 do
64    sleep 4
65    8.times do
66      with_fx :slicer, phase: 1.0/8, pulse_width: rrand(0.85,0.95) do
67        with_fx :bitcrusher, bits: rrand(5,9) do
68          sample :drum_snare_soft, amp: rrand(0.6,0.8), rate: rrand(0.43,0.45)
69        end
70      end
71      sleep (ring 0.25,0.25,0.5).tick
72    end
73  end
74  ##########################################################################################
75  live_loop :rings do
76    z = tick
77    with_fx :pitch_shift, pitch: (ring -6,-12,6,-6)[z], mix: 0.7 do
78      sample rings, rate: (ring 0.2,0.1)[z], amp: 0.66, start: 0.01, pan: -1
79    end
80    with_fx :pitch_shift, pitch: (ring 6,12,6,-6)[z], mix: 0.7 do
81      sample rings, rate: (ring 0.2,0.1)[z], amp: 0.66, start: 0.01, pan: +1
82    end
83    sleep 16
84  end
```

37. "sky on the eye"

Linha **14**: Redução de taxa de amostragem de 44.000 para 43.000, o que gera um sutil efeito de distorção.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-bitcrusher.mp3

```
01 # Alexandre rANGEL "sky on the eye" v03
02 # 18-Sep-2016 / Sonic Pi 2.11 dev
03
04 use_bpm 144 #72
05 i1 = 0; i2 = 0
06 playhead = 128
07 ##########################################################################################
08 live_loop :count do
09   playhead = playhead + 1
10   puts '>>>>> #{playhead}'
11   sleep 1
12 end
13 ##########################################################################################
14 with_fx :bitcrusher, sample_rate: 43000 do
15   live_loop :drums1 do
16     i1 = i1 + 1
17     if i1 > 128/4
18       myRate = 1.0
19       if one_in(32)
20         i1 = i1 - 1
21         myRate = rrand(0.8,0.9)
22       end
23       with_fx :bitcrusher, bits: (ring 12,10,12)[i1] do
24         sample :drum_tom_lo_hard, rate: myRate, amp: 0.3
25         if i1 > 64 and i1.odd?
26           sample :bd_zum, rate: -myRate/3.5, amp: 0.75
27           sleep 0.25
28           sample :bd_pure, rate: rrand(-1.5,-8.5), amp: 1.0 if one_in(2)
29         end
30       end
31     end
32     sleep 1.0/2
33     #sleep 16
34   end
35 ##########################################################################################
36   live_loop :drums2 do
37     i1 = i1 + 1
38     if i1 > 96/4
39       i2 = i2 + (1.0/8)
40       i2 = 0 if i2 > 4
41       with_fx :hpf , cutoff: (ring 90,120,90,60)[i1] do
42         if rrand(0,100) > (ring 10,33,70,40,90)[i2.to_int]
43           sample [:elec_blip,:elec_blip2].choose, amp: 0.8
44         end
45       end
46     end
47     sleep 1.0/4
48     #sleep 64
49   end
50 ##########################################################################################
51   live_loop :drums3 do
```

```
52
53      i1 = i1 + 1
54      with_fx :bitcrusher, bits: [5,6].choose,
55      sample_rate: rrand(8000,24000) do
56        sample :drum_heavy_kick
57      end #fx
58
59      sleep (ring 4,2,2,1,1,1,1,1,1,1,1,1).tick
60      #sleep 64
61    end
62  ##########################################################################################
63    live_loop :synth1 do
64
65      with_fx :bitcrusher, sample_rate: 6000 do
66        with_fx :echo, phase: 2, decay: 4 do
67
68          with_fx :slicer, phase: [0.25,0.25,0.5,0.5,1].choose do
69            with_fx :ixi_techno, phase: rrand_i(1,32), res: rrand(0,0.5), mix: rrand(0,0.5) do
70
71              use_synth [:fm,:fm,:blade,:prophet].choose
72              play_chord scale(:a2, :minor), release: [16,32].choose,
73                divisor: rrand_i(2,16), depth: rrand_i(0,16), amp: 0.8
74
75              sleep 8
76
77              use_synth [:fm,:fm,:blade,:prophet].choose
78              play_chord scale(:a2, :minor), release: [4,8,16,24,32].choose,
79                divisor: rrand_i(2,8), depth: rrand_i(0,8), amp: 0.5
80
81            end #fx
82          end #fx
83        end #fx
84      end #fx
85
86      sleep 8
87      #sleep 64
88    end
89
90  end #bitcrusher
```

38. "the target calls the arrow"

www.alexandrerangel.art.br/mp3/

```
# Alexandre rANGEL "the target calls the arrow" v04
# 17-Out-2016 / Sonic Pi 2.11 dev

synth = 'C:/samples/synth-psycho.wav'
use_bpm 140

live_loop :synth1 do
  part = rrand_i(1,7)
  with_fx :compressor, slope_above: 0.75, slope_below: 3 do
    with_fx :flanger, phase: [4,8,16,32,64].choose do
      with_fx :slicer, phase: [1,0.25,0.5].choose do
        with_fx :echo, phase: rrand(0.1,0.333), decay: 2 do
          with_fx :pitch_shift, pitch: -8 do
          sample synth, rate: 0.25/2, amp: 5, start: (1.0/8)*part, finish: (1.0/8)*(part+1),
              attack: 1, release: 1, pan: (ring -0.8,0.8).tick, pan_slide: 1
          end
        end
      end
    end
  end
  sleep 8
end
##########################################################################################
sleep 8
##########################################################################################
live_loop :synth2 do
  with_fx :echo, phase: [1,2,2,2,3].choose, mix: [0,0.25].choose do
    with_fx :slicer, phase: [0.25,0.25,0.25,0.25,0.5].choose do
      with_fx :wobble, phase: 2, res: 0, wave: 2, smooth: 0.25, mix: 0.6 do
        with_fx :whammy, transpose: -12, mix: 0.5, grainsize: 0.25 do
          with_fx :slicer, phase: 0.5 do
            with_fx :nrlpf, mix: 0.7 do
              sample synth, rate: [0.5,1].choose, amp: 0.2, release: 2 if one_in(2)
            end
          end
        end
      end
    end
  end
  sleep 16
end
##########################################################################################
sleep 8
##########################################################################################
live_loop :drums do
  x = tick
  sample :bd_haus, amp: ring(1,0,0,0)[x]

  with_fx :slicer, phase: (ring 0.5,0.5,0.25,0.1)[(x/32)] do
    with_fx :echo, phase: 0.05, mix: rrand(0.1,0.2), decay: 3 do
      sample :drum_snare_soft,  amp: ring(0,0,0,0, 0.8,0,0,0)[x]
    end
  end

```

```
  with_fx :slicer, phase: 0.25 do
    with_fx :krush, mix: 0.5 do
      sample :guit_em9, amp: ring(1,0,0,0)[x] if one_in(24)
    end
  end

  sample :drum_snare_soft, amp: ring(0,0,0,0, 0,0,0,0)[x]
  sample :drum_bass_soft,  amp: ring(0,1,0,1, 0,1,0,1)[x]
  sleep 0.25

end
```

39. "capitalism will make humanity interplanetary"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-interplanetary.mp3

```
# Alexandre rANGEL "capitalist will make humanity interplanetary" v01
# 2-Oct-2016 / Sonic Pi 2.11 dev

use_bpm 133
set_volume! 1

with_fx :compressor, slope_below: 1.2, slope_above: 0.7, threshold: 0.7, mix: 0.9 do

live_loop :drums1 do
  sample :bd_klub, amp: (ring 2,3).tick
  sleep 1
end
##########################################################################################
sleep 4
##########################################################################################
live_loop :drums2 do
  x = tick
  with_fx :tanh, krunch: rrand(1,5) do
    sample :drum_cymbal_soft, amp: (ring 0.4,0.2)[x], finish: (ring 0.1,0.1,0.1,0.2)[x]
  end
  sleep 1.0/3
end
##########################################################################################
sleep 8
##########################################################################################
live_loop :notes do
  n = scale(:f3,:bartok).choose
  with_fx :slicer, phase: [0.25,0.25,0.25,0.5,0.5,0.5,1].choose do
    use_synth :fm
    play n, depth: 4, divisor: 8, attack: [2,2,2,4].choose, sustain: 7, amp: 0.5,
      pan: [-0.3,0.3].choose, pan_slide: [1,2,4,6].choose
    use_synth :tech_saws
    play n, sustain: 2, amp: 0.4, pan: [-0.5,0.5].choose, pan_slide: 1
    sleep 2
    play n, attack: 2, amp: 0.5, pan: [-0.5,0.5].choose, pan_slide: 1
    sleep 6
  end
end
##########################################################################################
end #compressor
```

40. "born out of concrete"

Linha **15**: Os parâmetros com o sufixo ***_slide*** *(deslizar)*, a cada mudança de valor, transformam-se suavemente (com duração determinada) entre o valor antigo e o valor novo. Sem o comando ***_slide***, o valor é modificado instantaneamente.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-concrete.mp3

```
01 # Alexandre rANGEL "born out of concrete" v03
02 # 9-Oct-2016 / Sonic Pi 2.11 dev
03
04 use_bpm 120
05 ##########################################################################################
06 live_loop :notes1 do
07   with_bpm 240 do
08     x = tick
09     use_synth :blade
10     with_fx :panslicer, phase: [0.5,1].choose do
11       with_fx :echo, phase: 1, decay: 3 do
12         with_fx :slicer, phase: 0.5 do
13           play [scale(:c5,:chinese).tick,scale(:c5,:chinese).choose].choose,
14              sustain: [2,4,4,8,8,16].choose, amp: (ring 0,0,1.5,1.2,1.5,9,0,0,0,1.2,1.5,0,0,0)[x],
15              amp_slide: 2
16         end
17       end
18     end
19     sleep [3,6].choose
20     sleep 3 if one_in(4)
21   end
22 end
23 ##########################################################################################
24 live_loop :notes2 do
25   with_bpm 240 do
26     x = tick
27     use_synth :fm
28 with_fx :pitch_shift,pitch: [-6,-12,-12,-18].choose, window_size: rrand(0.0001,[0.001,0.1].choose) do
29       4.times do
30         with_fx :slicer, phase: [0.25,0.5,0.5,0.5].choose do
31           with_fx :flanger, phase: [2,4,8,12].choose, depth: 8, feedback: 1, mix: rrand(0,1) do
32             with_fx :echo, phase: 2, decay: 4 do
33               with_fx :reverb, mix: rrand(0.1,0.4444) do
34                 with_fx :echo, phase: 0.2, decay: 2 do
35                   with_fx :slicer, phase: 0.5 do
36 play [scale([:c3,:e3].choose,:gong).tick,scale([:c3,:f3].choose,:gong).choose].choose,
37 sustain: [0.25,0.1].choose, divisor: [0.5,1,2].choose, depth: [0.5,1,2,4,8,16,32,64].choose,
38 amp: (ring 0,0,1.8,2.2,0,0,1.8,2.2,0,0,1.8,2.4)[x], amp_slide: 0.5
39                   end
40                 end
41               end
42             end
43           end
44         end
45         sleep [0.5,1].choose
46       end
47     end
48     sleep [6,8,12].choose
49   end
```

```
50  end
51  ##########################################################################################
52  live_loop :drums1 do
53    with_fx :echo, phase: 1, decay: 3 do
54      sample :tabla_ghe3, amp: ((ring 1.1,1.5).tick) * 1.1
55    end
56    sleep 4
57  end
58  ##########################################################################################
59  live_loop :drums2 do
60    x = tick
61    with_fx :echo, phase: 1.0/4, decay: (ring 1,2)[x] do
62      sample :tabla_re, amp: (ring 1.1,1.5)[x]
63    end
64    sleep (ring 2,1)[x/4]
65  end
66  ##########################################################################################
67  live_loop :kick do
68    x = tick
69    sample :bd_808, amp: (ring 2,2.4)[x]
70    sample :bd_haus, amp: (ring 1.8,2)[x] + rand(0.2)
71    sleep 1
    end
```

41. "shapeshifter"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-shapeshifter.mp3

```
# Alexandre rANGEL "shapeshifter" v13
# 16-Out-2016 / Sonic Pi 2.11 dev

#delay = 1 # does arrangement
delay = 0 # full on
myBPM = 36; use_bpm myBPM
clock = 0
t = Time.new
set_volume! 0.6
##########################################################################################
live_loop :metro1 do
  with_bpm myBPM do
    myBPM = 72  if (clock/4) < 23 #medium
    myBPM = 36  if (clock/4) < 11 #slow
    myBPM = 144 if (clock/4) > 23 #fast

    clock = tick
    puts "bpm: #{myBPM}, bar: #{clock/4}, beat: #{clock}"
    x = (ring "1 |","2 | |","3 | | |","4 | | | |")[clock]
    puts x
    #puts "sec : #{t.sec}"
    sleep 1
  end
end #metro1

live_loop :metro2 do
  with_bpm myBPM do
    sleep 0.5
  end
end #metro2

with_fx :compressor, threshold: 0.55, mix: 0.9,
slope_above: 0.66, slope_below: 1.18 do
##########################################################################################
live_loop :hoover do
  with_bpm myBPM do
    sync :metro1
    use_synth :hoover

    if one_in(2)
      play chord(:a5, :minor).choose,
        release: [6,8,12,16].choose, attack: 2, sustain: 2, amp: rrand(0.2,0.24),
        pan: rrand(-0.7,0.7), pan_slide: [2,4].choose
    end #if

    sleep 4

    if one_in(2)
      play chord(:c4, :m13).choose,
        release: [6,8,12,16].choose, attack: 2, sustain: 2, amp: rrand(0.2,0.22),
        pan: rrand(-0.7,0.7), pan_slide: [2,4].choose
    end #if

    sleep [4,8,16,32].choose
```

```
  end
end #hoover
##########################################################################################
sleep 4 #delay start
##########################################################################################
live_loop :beep2 do
  with_bpm myBPM do
    sync :metro1
    use_synth :growl

    with_fx :krush, phase: (ring 0.2,3,0.5,1)[clock/2], mix: rrand(0.6,0.8) do

      play (ring 51,51,39,63)[tick] - 32 + ((t.sec)),
        attack: (ring 2,0.1,1,0.1)[clock/4], release: [1,2,3,4,8,16].choose,
        amp: (ring 1.66, 1)[clock/3] * rrand(1.1,1.4) - ((t.sec/60 * 2))

    end #fx

    sleep (ring 1,2,0.5,1,0.5)[clock/2] * 4
  end
end #beep2
##########################################################################################
sleep 16 if delay = 1 #delay start?
##########################################################################################

live_loop :ride1 do
  with_bpm myBPM do
    sync :metro1

    with_fx :echo, phase: (ring 1.5, 2, 3, 2)[clock/4], mix: 0.5, reps: 2 do

      use_synth :noise
      play 60, amp: 0.10,attack: 0.2,release: rrand(1.0,1.3),pan:rrand(-1,1),pan_slide: 0.25

      sleep 3

      use_synth :gnoise
      play 60, amp: 0.20,attack: 0.1,release: rrand(1.1,1.3),pan:rrand(-1,1),pan_slide: 0.05

      sleep 9.0 / 2

    end #fx
  end
end #ride
##########################################################################################
sleep 16 if delay = 1 #delay start ?
##########################################################################################
with_fx :slicer, phase: 0.25, mix: 0.5, smooth_up: 0.1, smooth_down: 0.1 do
  with_fx :slicer, phase: 0.5 do
    with_fx :echo, phase: 0.14, mix: 0.8 do
##########################################################################################
live_loop :beep1 do
with_bpm myBPM do
sync :metro2

myScale = scale_names.choose
use_synth [:pulse,:growl,:beep,:growl,:prophet].choose

if one_in(2)

with_fx :echo, phase: (ring 0.2,3,0.5,1)[clock/3], reps: 2 do
```

```
  with_fx :pitch_shift, pitch: [-8,-4,0,0,4,8].choose,
          window_size: (rrand(0.001,0.05)+(t.sec/[1000,2000,600].choose)) do
  x = (ring 51,51,39,63)[clock/2] + (t.sec/(ring 2,3,10)[tick])
  play_pattern scale(x, myScale),release:0.1,amp:(ring 1.333, 1)[clock/3] * 2.4
  end #fx
end #fx

else

with_fx :echo, phase: (ring 0.2,3,0.5,1)[clock/3], reps: 2 do
  x = (ring 51,51,39,63)[clock/2] + (t.sec/(ring 2,3,10)[tick])
  play_pattern scale(x, myScale), release: 0.1, amp: (ring 1.6, 0.8)[clock/3] * 2
end #fx

end #if

sleep (ring 2,1,0.5)[clock/4] * 1

end #beep1
##########################################################################################
      end #fx
    end #fx
  end #fx
end #fx
##########################################################################################
sleep 16 if delay = 1 #delay start ?
##########################################################################################
live_loop :kick do
  with_bpm myBPM do
    x = tick
    sync :metro2
    sample :bd_haus, amp: (ring 0.2,0.8,0.2,1)[x] + 4
    sample :bd_klub, amp: (ring 1.1,0.7,1.5,0.6)[x] + 7
    sleep (ring 1,2,1,1, 1,1,1,1, 1,1,1,2, 1,1,1,1, 1,1,1,2, 1,1,1,1)[x/24]
  end #myBPM
end #kick
##########################################################################################
live_loop :bass do
  sync :metro1
  with_fx :slicer, phase: (ring 0.5,0.2,0.5,0.1)[tick/4] do
    with_fx :bitcrusher, sample_rate: rrand(6000,40000) do
      with_fx :normaliser, level: 4, mix: 0.7 do
        with_fx :flanger, phase: (ring 0.02,1,0.05,0.2,0.5,0.75)[tick/4] do
          sample [:bass_thick_c, :bass_hit_c].choose, finish: 0.2, rate: 1.0/[2,3].choose, amp: 2.8
        end #fx
      end #fx
    end #fx
  end #fx
  sleep 3
end
##########################################################################################
live_loop :ride2 do
  with_bpm myBPM do
    t = Time.new
    with_fx :slicer, phase: 0.5 do
      use_synth :prophet
      play ( (t.sec/3) + 36 - rand_i(36)), sustain: 4, release: 4, amp: rrand(1,6),
        pan: rrand(-0.75,0.75), pan_slide: 0.15
    end #fx
    sleep (ring 3,3,6, 3,3,12)[clock/2] * 4.0
  end #myBPM
```

```
end #bass
##########################################################################################
live_loop :ride3 do
  use_synth :cnoise
  with_bpm myBPM do
    play 60, release:0.05,amp:(ring 2,1.7,1.3,1.7)[tick]-1.2, pan:rrand(-0.75,0.75), pan_slide:0.15
      sleep (ring 3,3,6, 3,3,1.5)[clock/24] / 2.0
    end #myBPM
  end #ride 3
##########################################################################################
end #compressor
```

42. "on set"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-onset.mp3

```
# Alexandre rANGEL "on set" v03
# 23-Out-2016 / Sonic Pi 2.11 dev

use_bpm 140
x = 1
##########################################################################################
live_loop :onset do
  if one_in(8)
    with_fx :echo, phase: 1, decay: [1,2,3,4].choose,mix:[0,0,0,0,0.5,0.7].choose do
      sample [:loop_industrial,:loop_garzul].choose, onset: x+rand_i(10),
        rate: [1,1,1,1,1,1,-0.5,0.5,0.333].choose, amp: 2.4,
        pan: [-0.6,-0.3,0.3,0.6].choose, pan_slide: [0.1,0.2,0.5].choose
    end
  else
    myRate = rrand(0.93,1.0)
    sample :loop_amen, onset: x, rate: myRate, amp: 2.4
  end
  sleep 0.5
  x = x + [1,1,1,1,1,0,-1,-2,-4,2].choose
end
##########################################################################################
with_fx :slicer, phase: 0.5 do
  live_loop :notes do
    with_bpm 140/[1,2,3,4,8,9,16].choose do
      if one_in(3) then
          with_fx :whammy, transpose: -12, grainsize: rrand(0.001,0.1), mix: 0.6 do
            with_fx :wobble, phase: 2, mix: 0.6 do
              use_synth [:pulse,:pulse,:beep].choose
              with_fx :slicer, phase: (ring 0.25,0.25,0.5,0.25,0.5,1,0.1).tick do
      play scale([:c4,[:e3,:e4].choose].choose,:gong).tick, amp: 0.5, release: 3 if one_in(3)
              end
            end
          end
        end
      end
      sleep 1
    end
  end
```

43. "starting euclides"

Linha **22**: Acionamento de *sample* com distribuição de probabilidade euclidiana.

Linhas **34** e **35**: Seleção aleatória do sintetizador que tocará as próximas quatro notas.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-euclides.mp3

```
01 # Alexandre rANGEL "starting euclides" v02
02 # 30-Oct-2016 / Sonic Pi 2.11dev
03
04 use_bpm 120
05 set_sched_ahead_time! 0.1
06 set_volume! 1.6
07 ##########################################################################################
08 live_loop :beat1 do
09   sample :elec_bong, amp: 1.5 if (spread 3, 8).tick
10   sample :bd_haus, amp: 2 if (spread 1, 4).look
11   sleep 0.25
12 end
13 ##########################################################################################
14 x = 0
15 live_loop :beat2 do
16   p = (ring 0.25,0.1,0.25,0.5,0.25,0.2)[x]
17   x = x + 1
18   t = tick
19   22.times do
20     with_fx :echo, phase: p, phase_slide: 6, decay: 2 do
21       with_fx :reverb do
22       sample :elec_ping, amp:0.66 if (spread 7,(ring 11,10,12,11,14,21,28,11)[t])[t]
23       end
24     end
25     sleep 0.25
26   end
27   sleep 12 * 0.25
28 end
29 ##########################################################################################
30 live_loop :notes do
31   t = tick
32   with_fx :slicer, phase: [0.25,0.5].choose do #2,0.5,0.25,0.1,0.05
33     with_fx :compressor, slope_above: 0.5, slope_below: 1.2, mix: 0.8 do
34       use_synth (ring :fm,:chipbass,:chiplead,:prophet,:square,:tb303,:piano,:noise).choose
35       4.times do
36         play [scale(60,:chinese)[t],scale(60,:chinese).choose].choose,
37           divisor: 8, depth: 4, pulse_width: 0.75, attack: 3, sustain: 0, release: 8,
38           pan: [-1,1].choose, pan_slide: [2,4].choose, amp: 0.555
39         sleep 8
40       end
41     end
42   end
43 end
```

44. "oh, Euclides"

Linhas **4** a **9**: Ritmo percussivo programado com algoritmo de distribuição euclidiana.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-oheuclides.mp3

```
01 # Alexandre rANGEL "oh euclides" v1
02 # 5-Nov-2016 / Sonic Pi 2.11
03
04 live_loop :euclid_beat do
05   sample :tabla_tas3, amp: 1.5 if (spread 3, 8).tick
06   sample :elec_blip2, amp: 0.9 if (spread 7, (ring 11,12,22)[look/16]).look
07   sample :bd_fat, amp: 7 if (spread 1, 4).look
08   sleep 0.125
09 end
10 ##########################################################################################
11 sleep 4
12 ##########################################################################################
13 live_loop :notas do
14   use_synth :pluck
15   play scale([:e3,:c6].choose,:chinese).choose, attack: 0.1, sustain: 0.1, release: 0.1
16   sleep 0.125
17 end
18 ##########################################################################################
19 sleep 8
20 ##########################################################################################
21 live_loop :notas do
22   use_synth :blade
23   play scale(:e3,:spanish).choose, attack: 0.1, sustain: 0.1, release: 2
24   sleep 0.125 * 8
25 end
```

45. "discovering myself"

Linha **5**: Várias opções de velocidade de andamento.

Linhas **13** e **32**: Controlam a quantidade de vezes que o trecho deverá ser repetido. Essa composição não tem o comando ***live_loop,*** como têm as outras obras apresentadas.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-discovering.mp3

```
01  # Alexandre rANGEL "discovering myself" v1
02  # 13-Nov-2016 / Sonic Pi 2.11
03
04  set_recording_bit_depth! 32
05  use_bpm 45/1.5 #60, 45, 45/1.5, 45/2, 45/4, 45/8, 45/16
06  start = 0
07
08  with_fx :reverb, mix: 0.5 do
09    with_fx :compressor, pre_amp: 1.5, thereshold: 0.5, slope_above: 2, slope_above: 0.7 do
10
11      lfo = range(0.0, 1.0, 0.0045).mirror
12
13      323.times do
14        x = tick+start
15        y = (ring 12,8,8,9,8,8,7)[x/48]
16
17        with_fx :wobble, res: lfo[x], res_slide: 0.12, mix: ring(0.53,0,0.72,0)[x] * lfo[x]  do
18
19          print x
20          print lfo[x]
21
22          use_synth (ring :chiplead, :chipbass)[x/48]
23          play scale(:c,:spanish)[x], pan: -1 if (spread 3, y)[x]
24
25          use_synth :piano
26          play scale(:c,:spanish)[x], amp: rrand(1.8,2), pan: 1 if (spread 3, 8)[x] == false
27
28        end
29
30        sleep 0.125
31
32      end #loop
33
34    end
35  end
```

46. "good winds"

Linha **6**: Configura a gravação de áudio para o padrão de qualidade de 32 *bits* (o padrão é a qualidade de CD de áudio: 16 *bits*).

Linha **7**: Afinação especial do Lá em 432 *hertz*.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-good-winds.mp3

```
01 # Alexandre rANGEL "good winds" v42
02 # 20-Nov-2016 / Sonic Pi 2.11
03
04 #good winds carry me to a journey to the inner space
05
06 set_recording_bit_depth! 32
07 use_cent_tuning -31.76665363342928
08 set_volume! 0.14444
09 use_bpm 64
10 set_sched_ahead_time! 4
11
12 with_fx :compressor, pre_amp: 0.777, amp: 0.777, threshold: 0.6, clamp_time: 0.015, relax: 0.05,
13         slope_above: 0.4444, slope_below: 1.1, mix: 0.9 do
14
15 with_fx :slicer, phase: 0.008, mix: 0.84 do
16   with_fx :slicer, phase: 0.64, slope_up: 0.001, slope_down: 0.01, mix: 0.3 do
17
18     live_loop :universe do
19     with_bpm 16 do
20       t = tick
21       print status
22
23 with_fx :pitch_shift, pitch: [rand(1),rand(4),rand(8),8,12,rand(16)].choose,
24   pitch_slide: [0.125,0.5,1,2,4,8,16].choose, window_size: rrand(0.0001,[0.001,0.01,0.1,1].choose),
25   mix: [0,rrand(0.3,0.8)].choose, mix_slide:[0.5,1,2,4].choose do
26 with_fx :ring_mod, freq: rrand(20,70), freq_slide: [0.5,1,2,4,8].choose,
27   mix: rrand(0.05,0.66), mix_slide:[0.25,0.5,1,2,4].choose do
28 with_fx :flanger, phase: [0.125,0.25,0.5,1,4].choose, phase_slide: [0.1,0.5,1,2,4,8].choose,
29   mix: rrand(0.0,0.7), mix_slide:[0.5,1,2,3,4,5].choose do
30 with_fx :vowel, vowel: [1,2,3].choose, voice: [1,2,3].choose,
31   mix: rrand(0.0,0.9), mix_slide: [0.25,0.5,1,2,4,8,16,32].choose do
32 with_fx :bitcrusher, bits: rrand(14,16), bits_slide: 0.1, sample_rate: rrand(20000,44000),
33   sample_rate_slide: [0.125,0.25,0.5,1,2,4,8,16].choose, mix: rand(0.66),
34   mix_slide: [0.5,1,2,4].choose do
35 with_fx :octaver, mix: rrand(0.3,0.93), mix_slide: [0.5,2,4,12].choose do
36 with_fx :wobble,phase:0.5,mix:rrand(0.1,0.49),mix_slide:[0.5,2,4].choose do
37 with_fx :ring_mod, freq: rrand(30,70), freq_slide: [0.25,0.5,1,2].choose, mix: rrand(0.05,0.5) do
38 with_fx :ixi_techno, mix: rrand(0,rand(0.7)), mix_slide: [0.5,2,4,12].choose,
39   phase: rand(2), phase_slide: [0.125,0.25,0.5,1,2,4,8].choose, res: rand(0.7) do
40 with_fx :octaver, sub_amp: rrand(0.7,2.2), subsub_amp: rrand(0.7,2.2),
41   mix: rrand(0.0,rand(0.7)), mix_slide: [0.5,2,4,1,8].choose do
42 with_fx :flanger, feedback: rand(0.6), feedback_slide: rand(0.6),
43   depth: [rand(2),rand(8),rand(16),rand(24)].choose, depth_slide: [0.5,2,4,8].choose,
44   phase: (ring 1,0.5,0.25,2,4)[t], phase_slide: [0.5,1,2,2,4,8,16].choose,
45   mix: rrand(0.2,0.6), mix_slide: [0.5,2,4].choose do
46 with_fx :whammy, mix: rrand(0.0,rand(0.5)), mix_slide: [0.25,2,4,8].choose,
47   transpose: rrand(-32,[16,24].choose), transpose_slide: 3,
48   grainsize: [rand(0.05),rand(0.2)].choose, grainsize_slide: [0.5,2,4,8].choose do
49 with_fx :octaver, mix: rrand(0.0,rand(0.7)), mix_slide: [0.5,2,4,8,8].choose do
```

```
with_fx :vowel, vowel_sound: [1,4,4,4,5].choose, voice: [3,4].choose, mix: rrand(0.2,0.77),
  mix_slide: [0.2,0.5,1,2,4,12].choose do

x1 = scale(:e3,:egyptian).choose
use_synth [:hoover,:growl,:growl].choose

mynote1 = play x1, amp: rrand(0.1,0.25), note_slide: [1,2,3,4,8].choose,
  attack: [2,3,4,8].choose * 1.5, sustain: 1, release: 6 * 1.5, pan: -1
print 'x1'

sleep 0.3 * 1.5

x1 = scale([:e2,:e3,:c4].choose,:egyptian).choose
x2 = x1 + 3
use_synth [:hoover,:growl,:growl].choose
mynote2 = play x2, amp: rrand(0.1,0.25), note_slide: [1,2,3,4].choose,
  attack: [2,3,4].choose * 1.5, sustain: 1, release: 4 * 1.5, pan: +1
print 'x2'

sleep 0.4 * [1.5,2,3,4].choose

x1 = scale(:e2,:egyptian).choose
x3 = x1 + 5
use_synth [:hoover,:growl,:growl].choose
mynote3 = play x3, amp: rrand(0.1,0.25), note_slide: [1,2,3,4,8].choose,
  attack:[6,8,12].choose, sustain:0, release:12, pan:[-1,1].choose, pan_slide: [0.1,0.25,0.5].choose
print 'x3'

sleep 0.5 * [1.5,2,3,4].choose

4.times do
  x1 = scale([:e2,:e3].choose,:egyptian).choose
  control mynote1, note: x1, amp: rrand(0.22,0.46), amp_slide: 0.2,
  note_slide: [0.25,0.5,1,2,4].choose
  print 'x1'
  sleep [0.3,0.4,0.5,0.6].choose * [1.5,2,3,4].choose
  x2 = x1 + 2
  control mynote2, note: x2, amp: rrand(0.22,0.42), amp_slide: 0.2,
  note_slide: [0.3,0.5,1,2,4].choose
  print 'x2'
  sleep [0.2,0.4,0.6].choose * [1.5,2,3,4].choose
  x3 = x1 + 5
  control mynote3, note: x3, note_slide: [0.5,1,2,4,8].choose,
  amp: rrand(0.22,0.33), amp_slide: 0.2, pan: [-1,1].choose, pan_slide: [0.5,1,2,4].choose
  print 'x3'
  sleep [0.2,0.3,0.4,0.5].choose * [1.5,2,3,4].choose
end

sleep 0.15 * [0.5,1,1.5,2].choose

end
end
end
end
end
end
end
end
end
end
end
```

```
end
end
end
end

end
end
end

sleep 12

with_fx :reverb, mix: 0.777 do

lfo = range(0.0, 1.0, 0.00375).mirror

live_loop :arp do
with_bpm 32 do
x = tick
y = (ring 12,8,8,9,8,8,7)[x/48]

with_fx :pitch_shift, pitch: -8+rrand(-1,1), pitch_slide: [0.5,1,2].choose,
window_size: rrand(0.0001,[0.001,0.01,0.1].choose),mix: rrand(0.5,0.9) do

with_fx :flanger, phase: [0.5,1,2,3,4].choose, mix: [0,0,0,0,1].choose do

with_fx :wobble, res: lfo[x], res_slide: 0.12, mix: ring(0.3,0,0.55555,0)[x] * lfo[x]  do

print "lfo is #{lfo[x]}" if one_in(6)

use_synth (ring :piano, :pluck)[x/48]
play scale(:c3,:spanish, num_scales: [1,3,3,5].choose)[x],
amp: rrand(1.0,3.3) * 3.33 * 1.12, pan: -1 if (spread 3, y)[x]

use_synth :fm
play scale(:c3,:spanish, num_scales: [1,3,3,5].choose)[x], amp: rrand(1.0,3.3) * 3.7, pan: 1,
divisor: rand(2), depth: rand(2) if (spread 3, 8)[x] == false

end
end
end

sleep 0.125 * [0.25,0.5,0.5,0.5,1,1,1,1,2,2,2,4].choose * 2

end
end

end #loop

end #compressor
sleep 12

with_fx :reverb, mix: 0.4 do
  live_loop :tabla do
    t2 = tick
    x2 = (ring 5,4,7)[(t2/(48*2.4)).to_int]
    y2 = (ring 11,12,13)[(t2/(64*2.4)).to_int]
    sample :bd_fat, amp: 27 *rrand(1,1.1) if spread(1,4)[t2]
    sample :tabla_tas2, amp: 2.6 *rrand(1,1.1), rate: rrand(0.98,1.02) if spread(x2,y2)[t2]
    sleep 0.125
  end
end
```

47. "this is a trip"

Linhas **12** a **22**: Sistema de escolha de notas baseado nas notas escolhidas anteriormente.

Linha **26**: Esse efeito, se dentro do *loop*, distorce o som, variando, a cada passagem, a taxa de amostragem e a porcentagem de aplicação do efeito (***bitcrusher***).

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-trip.mp3

```
01 # Alexandre rANGEL 'this is a trip' v10
02 # 27-Nov-2016 / Sonic Pi 2.11
03
04 set_recording_bit_depth! 32
05 set_sched_ahead_time! 4
06 use_bpm 96
07 note1 = 60; note2 = 63
08 arrangement = true
09
10 live_loop :perc do
11
12   note1previous = note1
13   note1 = scale(-16,:blues_minor,num_octaves: 3).tick if one_in(2)
14   note1 = note1previous - 1 if one_in(6)
15   note1 = note1previous - 2 if one_in(11)
16   note1 = note1previous if one_in(8)
17
18   note2previous = note2
19   note2 = scale(-16,:blues_minor,num_octaves: 3).choose if one_in(2)
20   note2 = note2previous - 1 if one_in(4)
21   note2 = note2previous - 2 if one_in(8)
22   note2 = note2previous if one_in(6)
23
24   print "note1: #{note1}"; print "note2: #{note2}"
25
26   with_fx :bitcrusher, sample_rate: rrand(100,44000), mix: rrand(0.2,0.8) do
27
28     if one_in(3)
29       #note1 = scale(-16,:blues_minor,num_octaves: 3).tick if one_in(2)
30       sample [:elec_plip,:elec_ping].choose, rpitch: note1, amp: 2.0 + rand(0.5)
31     else
32       #note2 = scale(-16,:blues_minor,num_octaves: 3).choose if one_in(2)
33       sample [:elec_plip,:elec_ping].choose, rpitch: note2, amp: 2.0 + rand(0.5)
34     end
35
36   end #fx
37
38   sleep 0.25
39 end
40 ##########################################################################################
41 sleep 18 if arrangement == true
42 ##########################################################################################
43 live_loop :arp1 do
44   t3 = tick
45   use_synth [:piano,:beep,:chiplead,:chiplead,:hollow,:pluck].choose if one_in(4)
46   with_fx :flanger, phase: [1,2,4].choose, phase_slide: [1,2].choose, mix: 0.5 do
47     with_fx :ring_mod, freq: rrand(30,90), freq_slide: 1,
48     phase: [0.25,0.5].choose, mix: rrand(0.3,0.6) do
49 with_fx :ring_mod,freq:rand(100),freq_slide:1,phase:[0.25,0.5].choose,mix:rrand(0.3,0.6) do
```

```
50          with_fx :slicer, phase: [0.25,0.5].choose  do
51          if one_in(3)
52            play scale((ring :e2,:g2,:c3)[t3/24],:blues_major,num_octaves: 1)[t3],
53              release: 2.5 * [1,1,1,2].choose
54          else
55            play scale((ring(ring :e2,:f3)[t3/64],:g2)[t3/16], :blues_major,
56              num_octaves: 1).choose, release: 2.5 * [1,1,1,2].choose
57            end
58          end
59        end
60      end
61    end
62    sleep 1.5
63    sleep 1.5 if one_in(16)
64    sleep 3 if one_in(24)
65  end
66
67  sleep 12 if arrangement == true
68
69  ##| live_loop :kick do
70  ##|   sample :bd_fat, amp: 3+rand(1) if one_in([3,4].choose)
71  ##|   sleep 0.25
72  ##| end
73
74  sleep 12 if arrangement == true
75  sleep 0.25
76
77  live_loop :drums do
78    with_fx :compressor do
79      sample :bd_klub, amp: 8+rand(0.4)
80      sample :drum_bass_soft, amp: 2+rand(1.4)
81    end
82    sleep 1
83  end
```

48. "dive"

Linhas **28** a **34**: Uma chance em três (33%) de tocar a primeira nota e 77% de chance de tocar a segunda nota.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-dive.mp3

```
01 # Alexandre rANGEL 'this is a trip' v10
02 # 27-Nov-2016 / Sonic Pi 2.11
03
04 set_recording_bit_depth! 32
05 set_sched_ahead_time! 4
06 use_bpm 96
07 note1 = 60; note2 = 63
08 arrangement = true
09 ##########################################################################################
10 live_loop :perc do
11
12   note1previous = note1
13   note1 = scale(-16,:blues_minor,num_octaves: 3).tick if one_in(2)
14   note1 = note1previous - 1 if one_in(6)
15   note1 = note1previous - 2 if one_in(11)
16   note1 = note1previous if one_in(8)
17
18   note2previous = note2
19   note2 = scale(-16,:blues_minor,num_octaves: 3).choose if one_in(2)
20   note2 = note2previous - 1 if one_in(4)
21   note2 = note2previous - 2 if one_in(8)
22   note2 = note2previous if one_in(6)
23
24   print "note1: #{note1}"; print "note2: #{note2}"
25
26   with_fx :bitcrusher, sample_rate: rrand(100,44000), mix: rrand(0.2,0.8) do
27
28     if one_in(3)
29       #note1 = scale(-16,:blues_minor,num_octaves: 3).tick if one_in(2)
30       sample [:elec_plip,:elec_ping].choose, rpitch: note1, amp: 2.0 + rand(0.5)
31     else
32       #note2 = scale(-16,:blues_minor,num_octaves: 3).choose if one_in(2)
33       sample [:elec_plip,:elec_ping].choose, rpitch: note2, amp: 2.0 + rand(0.5)
34     end
35
36   end #fx
37
38   sleep 0.25
39 end
40 ##########################################################################################
41 sleep 18 if arrangement == true
42 ##########################################################################################
43 live_loop :arp1 do
44  t3 = tick
45  use_synth [:piano,:beep,:chiplead,:chiplead,:hollow,:pluck].choose if one_in(4)
46  with_fx :flanger, phase: [1,2,4].choose, phase_slide: [1,2].choose, mix: 0.5 do
47    with_fx :ring_mod, freq: rrand(30,90), freq_slide: 1,
48    phase: [0.25,0.5].choose, mix: rrand(0.3,0.6) do
49    with_fx :ring_mod,freq:rand(100),freq_slide:1,phase:[0.25,0.5].choose,mix:rrand(0.3,0.6) do
50         with_fx :slicer, phase: [0.25,0.5].choose  do
51         if one_in(3)
```

```
play scale((ring :e2,:g2,:c3)[t3/24],:blues_major,num_octaves: 1)[t3],release: 2.5 * [1,1,1,2].choose
        else
          play scale((ring(ring :e2,:f3)[t3/64],:g2)[t3/16], :blues_major,
            num_octaves: 1).choose, release: 2.5 * [1,1,1,2].choose
          end
        end
      end
    end
  end
  sleep 1.5
  sleep 1.5 if one_in(16)
  sleep 3 if one_in(24)
end
##########################################################################################
sleep 12 if arrangement == true
##########################################################################################
##| live_loop :kick do
##|   sample :bd_fat, amp: 3+rand(1) if one_in([3,4].choose)
##|   sleep 0.25
##| end
##########################################################################################
sleep 12 if arrangement == true
sleep 0.25
##########################################################################################
live_loop :drums do
  with_fx :compressor do
    sample :bd_klub, amp: 8+rand(0.4)
    sample :drum_bass_soft, amp: 2+rand(1.4)
  end
  sleep 1
end
```

49. "12 hours of into the night time"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-12hours.mp3

```
# Alexandre rANGEL "12 hours of into the night time" v17
# 11-Dec-2016 / Sonic Pi 2.11

use_bpm 12 #24
set_recording_bit_depth! 32
set_sched_ahead_time! 16
set_volume! 0.8

with_fx :mono, mix: 0.333 do
with_fx :compressor, mix: 0.5 do
with_fx :compressor do
##########################################################################################
live_loop :xyz do
with_bpm 12 do

with_fx :echo, phase: [0.01,1,2].choose, mix: 0.5, decay: 2 do
  with_fx :pitch_shift, pitch: -16, pitch_dis: rand(0.1),
    window_size: rrand(0.0001,0.11),pitch_slide: [0.25,0.5,1,2,4,8].choose,mix: 0.5 do

([1,2,3,4].choose).times do
  x = tick
  use_synth [:fm,:mod_fm,:dtri,:pluck].choose
    with_fx :flanger, mix: rand(1), phase: [2,4,8,16,32].choose do
    with_fx :vowel, voice: [0,0,0,1,2,3].choose, vowel: [1,2,3,4,5].choose,
      vowel_slide: [1,2,4].choose, mix: 0.7 do
    with_fx :bitcrusher, sample_rate: rrand(4000,44000), mix: rand(1) do
    with_fx :pitch_shift, pitch: rrand(-1,1), pitch_dis:rand(0.1), window_size: rrand(0.0001,0.11) do
    with_fx :flanger, mix: rand(1), phase: rrand(0.01,0.2), mix: rand(1) do

      play scale([:c0,:g0].choose,:minor,num_octaves: 2).choose,
        attack: [4,6,8,12,16].choose, release: [4,6,8,12,16].choose,
        pan: (ring -1,1)[x], amp: rrand(1.6,1.9), amp_slide: 2,
        divisor: [1,2,4,8,9,12,16].choose, depth: rand(8)

  end
  end
  end
  end
  end
sleep [1,2,4,4,4,6,6,6,6,8,8].choose
end
end

end
end
end
end
end
end
##########################################################################################
live_loop :cem do
  with_bpm 60 do
    with_fx :slicer do
      with_fx :bitcrusher, sample_rate: rrand(1000,44000), mix: 0.5 do
```

```
        use_synth [:mod_fm,:fm].choose
        play scale(:c2,:blues_minor).choose, attack: 2, release: 2,
          divisor: rand(4), depth: rand(4), amp: rrand(1.4,2.2)
      end
    end
    sleep 2
  end
end
##########################################################################################
live_loop :drums1 do
  with_bpm 60 do
    sample :bd_haus if spread(1,4).tick
    sample :bd_fat, amp: 4 if spread(1,2).look
    sleep 0.5
  end
end
```

50. "the encounter"

www.alexandrerangel.art.br/mp3/

```
# Alexandre rANGEL "the encounter" v10
# 11-Dec-2016 / Sonic Pi 2.11

set_recording_bit_depth! 32
set_sched_ahead_time! 4
use_bpm 111

##########################################################################################
live_loop :xx do
  with_bpm 222 do
    with_fx :pitch_shift, pitch: rrand(-8,-0.1),
      window_size: [rrand(0.001,0.01),rrand(0.01,0.1)].choose, mix: rrand(0.5,1) do
      with_fx :slicer, phase: 1.0/12 do
        with_fx :slicer, phase: 1.0/6 do
          with_fx :slicer, phase: 1.0/3 do
            use_synth :pluck
            play scale(:e3,[:diminished,:diminished2].choose,num_scales:[1,2,3].choose).choose,
              release: rrand(0.4,0.6) + 0.7, amp: rrand(1.0,1.3)
            sleep 0.015
            play scale([:e1,:e2].choose,[:diminished,:diminished2].choose,
num_scales:[1,2,3].choose).choose, release: rrand(0.4,0.6) + 0.7, amp: rrand(1.0,1.3)
            sleep 0.015
            play scale([:e1,:e2].choose,[:diminished,:diminished2].choose,
num_scales:[1,2,3].choose).choose, release: rrand(0.4,0.6) + 0.7, amp: rrand(1.0,1.3)
            sleep 0.97
            sleep 1 if one_in(6)
          end
        end
      end
    end
  end
end
##########################################################################################
sleep 8
##########################################################################################
#with_fx :echo, phase: 1, decay: 4, mix: 0.5 do
live_loop :zzz do
  with_fx :pitch_shift, pitch: rrand(-8,-0.1),
    window_size: [rrand(0.001,0.01),rrand(0.01,0.1)].choose, mix: rrand(0.5,1) do
    with_fx :panslicer, wave: rand_i(3), smooth: rand(0.5) do
      with_fx :slicer, phase: [0.5,0.25,0.125].choose, mix: 0.7 do
        with_fx :slicer, phase: [2,1,0.5].choose, mix: 0.7 do
          with_bpm [11,22,44].choose do
            use_synth :mod_pulse
play scale(:e3,[:diminished,:diminished2].choose, num_scales:[1,2,3].choose).choose,
release: rrand(0.4,0.6) + 0.7, divisor: rand(16), depth: rand(16)
            sleep 0.015
play scale(:e1,[:diminished,:diminished2].choose, num_scales:[1,2,3].choose).choose,
release: rrand(0.4,0.6) + 0.7
            sleep 0.015
            play scale(:e0,[:diminished,:diminished2].choose,
num_scales:[1,2,3].choose).choose, release: rrand(0.4,0.6) + 0.7
            sleep 0.97
          end
```

```
        end
      end
    end
  end
  sleep 1 if one_in(4)
end
##########################################################################################
sleep 16
##########################################################################################
with_fx :echo, phase: 0.25, mix: 0.3, decay: 2 do
  live_loop :beat1 do
    with_fx :bitcrusher, bits: rand(8), mix: 0.5 do
      with_fx :bitcrusher, bits: rand(16), mix: 0.5 do
        sample :bd_haus, amp: rrand(1.4,1.6)
        sample :bd_tek, amp: 1.6 if spread(1,4).tick
      end
    end
    sleep 1
  end
end
##########################################################################################
sleep 14
##########################################################################################
live_loop :beat2 do
  if spread(5,7).tick
    sample :elec_cymbal, finish: rrand(0.09,0.11), amp: rrand(0.6,0.8)
  end
  sleep 1.0/4
end
```

51. "slowly melting the city"

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-melting.mp3

www.freesound.org/people/Omar%20Alvarado/sounds/115203
www.freesound.org/people/johnnypanic/sounds/320817
www.freesound.org/people/ArcTis/sounds/347652

```
# Alexandre rANGEL "slowly melting the city" v13
# 25-Dec-2016 / Sonic Pi 2.11.1

use_bpm 62
set_sched_ahead_time! 8
set_volume! 0.8

ufo = 'C:/samples/ufo.aif'; bass = 'C:/samples/bass-slice.wav'; kick = 'C:/samples/kick-perc.wav'
##########################################################################################
live_loop :ufo do
  with_fx :whammy, transpose: [8,12,16].choose, mix: 0.5 do
    with_fx :slicer, phase: [0.25,0.5,1].choose, mix: rrand(0.3,0.7) do
      sample ufo, rate: 0.5, amp: 4.2, pan: -1, attack: 8
      if one_in(2)
        sample ufo, rate: 0.25, finish: 0.5, amp: 12, pan: 1, attack: 8
      else
        sample ufo, rate: 0.25, start: 0.5, amp: 12, pan: 1, attack: 8
      end
    end
  end
  sleep sample_duration ufo, rate: 0.5
end
##########################################################################################
sleep 32
##########################################################################################
with_fx :flanger, phase: 1 do
  with_fx :echo, phase: 0.5, reps: 2 do
    live_loop :bass1 do
      with_fx :slicer, phase: [0.25,0.5,1,2].choose do
        sample bass, attack: 0.15, release: 2, amp: rrand(7,9) + rand(3),
          rpitch: scale(:c4,:blues_minor,num_scales:3).choose,pan: [1,0.5,1].choose
        sleep 2
      end
      sleep 2 if one_in(2)
    end
  end
end
##########################################################################################
sleep 16
##########################################################################################
with_fx :reverb, mix: 0.7 do
  with_fx :flanger, phase: 4 do
    with_fx :flanger, phase: 1 do
      with_fx :echo, phase: 0.5, reps: 2 do
        live_loop :bass2 do
          with_fx :slicer, phase: [0.25,0.5,1,2].choose do
            with_fx :pitch_shift, pitch: rrand(-10,10), window_size: 0.01 do
              sample bass, attack: 0.15, release: 2, amp: rrand(6,10) + rand(3),
                rpitch: scale(:c4,:blues_minor,num_scales:3).choose, pan: [-1,-0.5,-1].choose
            end
```

```
          end
          sleep 2
          sleep 2 if one_in(2)
        end
      end
    end
  end
end
##########################################################################################
sleep 64
##########################################################################################
live_loop :kick do
  x = tick
  s = rrand(0,0.9)
  with_fx :lpf, cutoff: rrand(80,120), cutoff_slide: 2 do
    with_fx :slicer, phase: [0.25,0.5,1].choose do
      with_fx :pitch_shift,pitch: rrand(-17,-15), window_size: rrand(0.005,0.010) do
        sample kick, amp: 1.85+rand(0.1), rate: 0.4+rand(0.1) if spread(7,[11,22].choose)[x]
        sample kick, amp: 1.85+rand(0.1), rate: 0.8+rand(0.1) if spread(1,[4,8].choose)[x]
        with_fx :flanger, phase: [0.25,0.5,1,1.5].choose do
          with_fx :echo, reps: 2, phase: [0.25,0.333].choose do
            sample kick, amp: rrand(1.85,2.1), pan: [-1,1].choose,
              start: s, finish: s+0.1, rate: rrand(1.5,2.5) if spread(5,[7,9].choose)[x]
          end
        end
      end
    end
  end
  sleep 0.5
end
##########################################################################################
sleep 8
##########################################################################################
live_loop :clap do
  y = tick
with_fx :whammy, transpose: [-16,-12,-10,-8,-4,-2].choose, mix: rrand(0.2,0.4) do
sample :elec_blip, pan: rrand(-0.2,0.2), rate: [-1,-0.8,0.8,1].choose, finish: rrand(0.5,0.8),
  amp: rrand(1.2,2)+rand(3) if spread(2,[2,4,4,4,6,6].choose)[y]
end

  sample :elec_cymbal, pan: rrand(-0.2,0.2), finish: rrand(0.01,0.48),
    amp: rrand(0.18,0.333) if spread(8,13)[y]
  sleep 0.25
  sleep 0.25 if one_in(4)
end

16.times do
  sample :bd_haus, amp: rand(2)
  sample :bd_fat, amp: rand(1)
  sleep 1
end
##########################################################################################
live_loop :kick2 do
  with_fx :bitcrusher, sample_rate: rrand(8000,44000), mix: rrand(0.3,0.7) do
    sample :bd_haus, amp: 1.8 + rand(0.4)
  end
  sample :bd_fat, amp: rand(1)
  sleep 0.5
end
```

52. "i am the knight you are the pilgrim"

Linha **53**: A cada execução do ***loop***, uma velocidade de andamento diferente é escolhida (132, 66 ou 33 BPM - batidas por minuto). Essa velocidade influencia o comando de espera (*sleep*), bem como os parâmetros dos efeitos sonoros dentro do *loop*.

www.alexandrerangel.art.br/mp3/Alexandre_rANGEL-pilgrim.mp3

```
01 # Alexandre rANGEL "i am the knight you are the pilgrim" v05
02 # 31-Dec-2016 / Sonic Pi 2.11.1
03
04 use_bpm 132
05 set_volume! 0.8
06 set_sched_ahead_time! 4
07 ##########################################################################################
08 live_loop :clap do
09   with_fx :whammy, pitch: (ring 0,-8,0,-16,0,0).tick, mix: rrand(0.2,0.8) do
10     with_fx :slicer, phase: 0.125/2 do
11       with_fx :echo, phase: 0.25, decay: 16 do
12         with_fx :flanger, phase: [0.125,0.25,0.5,1,2].choose, mix: rrand(0.5,0.9) do
13         sample :drum_cymbal_soft, finish: 0.02, amp: rrand(0.2,0.6) +1, pan: rrand(-0.3,0.3)
14         end
15       end
16     end
17   end
18   sleep 1
19   sleep 4 if one_in(4)
20 end
21 ##########################################################################################
22 sleep 16
23 ##########################################################################################
24 live_loop :notes1 do
25   use_synth :beep
26   with_fx :slicer, phase: 0.5 do
27     play [:e3,:g2,:e2,:e4].choose, attack: 3, release: 3, amp: rrand(1,1.3) + 0.7,
28      amp_slide: [1,2].choose, pan: rrand(-0.2,0.2)
29   end
30   sleep 4
31 end
32 ##########################################################################################
33 sleep 20
34 ##########################################################################################
35 live_loop :kick1 do
36   x = tick
37   with_fx :flanger, phase: [0.1,0.2,0.25].choose, mix: rrand(0.38,0.66) do
38     if one_in(72)
39       16.times do
40         sample :bd_boom, amp: rrand(2.2,2.4)+2, rate: (ring 0.9,1)[x], finish: 0.4
41         sample :bd_ada, amp: rrand(0.3,0.4), rate: (ring 0.8,1)[x]
42         sleep 1
43       end
44     else
45       sample :bd_ada, amp: rrand(2.2,2.4)-0.4, rate: (ring 0.8,1)[x]
46       sleep 1
47     end
48   end
49 end
```

```
50 ##########################################################################################
51 sleep 8
52 ##########################################################################################
53 live_loop :notes2 do
54   with_bpm 132/[1,2,4].choose do
55     with_fx :whammy, pitch: [-24,-16,-12,-8].choose, mix: rrand(0.777,0.9) do
56       with_fx :flanger, phase: [16,8,4,2].choose, mix: rrand(0.5,0.9) do
57         with_fx :echo, phase: 8, reps: 2, decay: 16, mix: 0.333 do
58           with_fx :bitcrusher, bits: [6,8,12,14].choose, mix: rrand(0.2,0.444) do
59             with_fx :slicer, phase: 0.125, mix: 0.777 do
60               use_synth :mod_fm
61               play scale([:e2,:e3,:g2,:g3].choose,:blues_minor).choose,
62                 attack: 6, release: 6, depth: [2,3,4,6,8,12,16].choose,
63                 divisor: [2,3,4,6,8,12,16].choose, amp: rrand(0.02,0.03) + 0.4
64             end
65           end
66         end
67       end
68     end
69     sleep 8
70   end
71 end
```