

Guilherme Martins Lunhani

*Live coding: um algoritmo gerador de sonoridade tonal no A Study in Keith (2009),
de Andrew Sorensen*

9 de março de 2016

Guilherme Martins Lunhani

***Live coding: um algoritmo gerador de sonoridade tonal
no A Study in Keith (2009), de Andrew Sorensen***

Dissertação apresentada para pré-banca de qualificação no Programa Mestrado em Artes, Cultura e Linguagens do Instituto de Artes e Design da Universidade Federal de Juiz de Fora (UFJF), linha de Artes Visuais, Música e Tecnologia.

Universidade Federal de Juiz De Fora – UFJF

Instituto de Artes e Design – IAD

Programa de Pós-Graduação em Artes Visuais, Música e Tecnologia

Orientador: Prof. Dr. Luiz Eduardo Castelões

9 de março de 2016

*O Tao produziu o Um.
O Um produziu o Dois.
O Dois produziu o Três.
O Três produziu os dez mil seres.*

¹

¹ Lao Tsé. *Tao Te King*. Versão integral e comentários. 3^a edição. Editorial Attar.<<http://tao-te-king.org/42.htm>>.

Agradecimentos

Ao que é impossível pronunciar o verdadeiro Nome, mas cuja potência é o próprio sentido da palavra criatividade.

Aos sem nome, anônimos da Rua de Juiz de Fora, que deram sentido à fraqueza da pergunta deste trabalho.

Para uma família em cada canto do Universo, Jair, Olímpia e Júlia.

Aos Professores Dr. Luiz Eduardo Castelões, Dr. Alexandre Fenerich e Dr. Flávio Luiz Schiavonni, fundamentais no apoio institucional; na sugestão de leituras; na cobrança de prazos; nas críticas; nas conversas sobre Música. À FAPEMIG por suprir esta lacuna, em um momento delicado nas finanças da Universidade Brasileira.

Aos freakcoders do *labMacambira*, especialmente ao Caleb Luporini, Daniel Penalva, Renato Fabbri e Vilson Vieira pelo estímulo nestes anos.

Aos amigxs que estão (ou moraram em Juiz de Fora): Glerm Soares, Tiago Rubini, Anna Flávia.

Aos amigxs de Campinas e São Paulo, que estiveram presentes ou na memória: Celso, Dani, Dhiego e Luisa, Evandro, Fábio, Felício, Frederico, Gustavo, Larissa ,Rebechi, Simone, Tati, ao pessoal da república Lado C, João, Heron, Igor. Ao velho amigo Picchi!

Ao Professor Hans Joachim Koellreutter pelo centenário.

Resumo

Este documento é uma nota pedagógica para interessados em improvisação de códigos. Indicamos eventos históricos ligados à Música, e situamos a análise de um trecho exemplar (e pedagógico) desta história. Através do Quadro Conceitual de Sistema Criativo de [McLean \(2006\)](#), [Wiggins \(2006\)](#), [McLean \(2011\)](#) analisamos o primeiro algoritmo gerador de uma sonoridade tonal em *A Study in Keith* de Andrew [Sorensen e Swift \(2009\)](#). Por outro lado, consideramos este exemplo representativo por vestir o programador com a figura de intérprete musical, e ao mesmo tempo, manter o espírito do pianista concertista em um ambiente rico de concepções musicais.

Palavras-chaves: *Livecoding. Study in Keith. Sistemas criativos.*

Sumário

Errata	ix
Introdução	xi
1 DEFINIÇÕES DE BASE DA IMPROVISAÇÃO DE CÓDIGOS	1
1.1 Tecelagem	2
1.1.1 Charles Babbage e Joseph-Marie Jacquard	2
1.1.2 Weavecoding	3
1.1.2.1 Slub	7
1.2 Dança	8
1.2.1 Algorave	8
1.2.2 Coreografia	12
1.2.2.1 Hacking Coreography	13
1.2.2.2 Hacking the Body	16
1.3 Música Computacional na improvisação de códigos	17
1.3.1 FooBarBaz	17
1.3.2 Música de concerto e imagens em movimento	20
1.3.3 Telepresença e espaços virtuais	24
1.4 Discussão	25
2 DEFINIÇÕES HISTÓRICAS DA IMPROVISAÇÃO DE CÓDIGOS	27
2.1 Pietro Grossi	27
2.2 Baía de São Francisco	32
2.2.1 The League of Automatic Composers	32
2.2.2 The Hub	35
2.2.3 Ron Kuivila	37
2.3 LAPTOP	37
2.4 TOPLAP	40
2.4.1 <i>Show us your screens</i>	41
2.4.1.1 Obscurantismo é perigoso, mostre-nos suas telas	42
2.4.2 Algorithms are Thoughts, Chainsaws are Tools	44
2.5 Discussão	45
3 METODOLOGIA DE ANÁLISE DE UMA IMPROVISAÇÃO MUSICAL DE CÓDIGOS	47

3.1	Criatividade	47
3.1.1	Criatividade	47
3.1.2	Criatividade, códigos e imagens mentais	51
3.1.3	Comportamento Criativo por Bricolagem	54
3.2	Quadro Conceitual de sistemas criativos	58
3.2.1	O modelo de improvisação	59
3.3	Diagramação dos espaços conceituais	62
3.4	Formalização	69
4	ESTUDO DE CASO	71
4.1	Objetivo	72
4.2	Justificativa	72
4.3	Referentes Opcionais	72
4.3.1	Concertos Sun Bear	72
4.3.2	NI-Akoustik Piano	75
4.3.3	Ambiente e Linguagem: Impromptu	76
4.3.4	Extempore	77
4.3.5	Scheme	78
4.4	Blocos de Eventos	84
4.4.1	Definição do instrumento e do tempo	86
4.4.1.1	Definição de uma sequência de blocos	87
4.4.1.2	Definição de blocos	88
4.4.2	Primeira sonoridade tonal	89
4.4.2.1	Regras de qualidade	91
4.4.2.2	Primeira sequência de blocos de eventos	91
	Conclusão	97
	Referências	101
	APÊNDICE A – CÓDIGO FONTE DE UM UNIVERSO CONCEITUAL COMO NUVEM DE PALAVRAS SOBRE O IMPROVISO DE CÓDIGOS	107
A.1	Classes qualitativas de um Universo de conceitos do <i>live coding</i>	108
	APÊNDICE B – SUGESTÕES PROTO-HISTÓRICAS	113
B.1	GROOVE	113
B.2	SuperCollider	116
B.2.1	Just In Time Library (JITLib)	119

Lista de ilustrações

Figura 1 – Tecido resultante da prática <i>Weaving code</i> . Fonte: Griffths (2015a).	5
Figura 2 – Performance no Foam Kernow. Fonte: Griffths (2015b).	6
Figura 3 – Alex McLean manipulando uma Matriz de botões para tecelagem, conectado a um Raspberry Pi. Fonte: Griffths (2015b).	6
Figura 4 – Performance do duo Canute (Karlesruhe, 2015) Fonte: Canute... (2015-27-01).	10
Figura 5 – Performance do duo Mico Rex (Londres, 2013) Fonte: Griffths (2013).	11
Figura 6 – Performance do duo Mico Rex (Londres, 2013) Fonte: Griffths (2013).	12
Figura 7 – Dançarina (anônima) controlada por Kate Sicchio (2015) através de uma codificação improvisada. Fonte: < https://www.youtube.com/watch?v=uAq4BAbvRS4 >.	17
Figura 8 – Festival Contato, 2011. Fonte: Fabbri e Vieira (2011).	18
Figura 9 – Performance de <i>screenBashing</i> . Fonte: < https://vimeo.com/148626379 >.	22
Figura 10 – Sistema de música computacional de John Bischof <i>circa</i> 1980. Foto: Eva Shoshanny ² . Fonte: Brown e Bischof (2002).	33
Figura 11 – Circuito do computador caseiro dedicado à síntese sonora de Tim Perkis, usado no começo dos anos 1980. Foto: Eva Shoshany ³ . Fonte: Brown e Bischof (2002).	35
Figura 12 – Definição do significado de TOPLAP. Fonte: Ramsay (2010).	40
Figura 13 – Classificação da criatividade : 1) criatividade-psicológica/criatividade-histórica; 2) criatividade exploradora/criatividade transformacional. Fonte: autor com base em Wiggins (2006).	49
Figura 14 – Exemplo de uma característica de alta viscosidade, proporcional às diversas possibilidade de notação secundária no PureData. Fonte: autor.	54
Figura 15 – Modelo de bricolagem para o processo criativo realizado por um artista-programador. Fonte: McLean (2011, p. 122).	55
Figura 16 – Representação da justaposição entre dois espaços conceituais. A região em marrom representa um grupo de conceitos transitórios, bem como os limites desta transição. Fonte: autor.	61
Figura 17 – Verticalização de uma rede de conceitos. Fonte: (Thornton, 2007)	63
Figura 18 – Conceitos categóricos e Conceitos compostos. Fonte: (Thornton, 2007)	64
Figura 19 – Complexificação da Verticalização de uma rede de conceitos. Fonte: (Thornton, 2007)	65
Figura 20 – Extensões conceituais em uma malha de Complexificação. Fonte: (Thornton, 2007)	66
Figura 21 – Zonas esponjosas Fonte: (Thornton, 2007)	67

Figura 22 – Transcrição do motivo gerador do disco Kyoto, parte 1. Fonte: autor.	74
Figura 23 – Piano Disklavier de armário, com a parte interna exposta para exibir a placa-mãe. Fonte: wikimedia.org	76
Figura 24 – Distribuição, aproximada, de probabilidades de acontecimento com um conjunto de possíveis cadências tonais organizados como uma cadeia de Markov. Fonte: Swift (2012).	85
Figura 25 – Transcrição literal e perceptiva do primeiro evento em <i>A Study in Keith</i> . Fonte: autor.	90
Figura 26 – Primeiros eventos musicais gerados a partir das primeiras estruturas válidas de código. Fonte: autor.	92
Figura 27 – Segundo bloco de eventos musicais. Fonte: autor.	92
Figura 28 – Terceiro bloco de eventos musicais. Fonte: autor.	94
Figura 29 – Primeiros perturbações sistêmicas. Fonte: autor.	96
Figura 30 – Definição de <i>live coding</i> : “Insira a definição aqui”. Fonte: Collins (2014).	98
Figura 31 – Resultado da conversão do arquivo <i>pdf</i> para <i>txt</i> resulta em problemas de codificação que necessitaram ser corrigidos por comparação com o arquivo original. Fonte: autor.	107
Figura 32 – Esquema de concepção do projeto GROOVE. Fonte: (MATHEWS; MOORE, 1970).	114
Figura 33 – Laurie Spiegel configurando a saída analógica do GROOVE, durante a produção de <i>The Expanding Universe</i> . Fonte: (SPIEGEL, 1975).	115

Errata

Existe uma seção com problemas de compilação L^AT_EX (ver [subseção 3.1.3](#), p. [54](#)). Isto é, caracteres que deveriam aparecer como \\$ aparecem como \\\$. A segunda seção de análise está sendo finalizada (ver ??, p. ??) e necessita de edições e mais transcrições. A terceira seção de análise não foi concluída ainda (ver ??, p. ??).

Introdução

Em seu artigo “A filosofia à venda, a dourada ignorância e a aposta de pascal”, Santos (2008, p. 11–12) descreve a imagem mental de uma feira do conhecimento, onde teorias são antropomorfizadas, escravizadas e vendidas. As idéias perderam a utilidade para os ex-adeptos, que não estão mais interessados em comprá-las. E vendem aos que supõem algum valor. Para efetuar a venda, é necessário estabelecer uma relação de custo-benefício, negociadas através de respostas às perguntas: “qual a utilidade que esta ou aquela teoria poderá ter para mim? Qual o preço?”. A valorização ocorre quando esta teoria se torna mais apelativa que aquela. Com a concorrência, a livre-associação dos vendedores regulamentará compras e vendas de conhecimentos conforme seu interesse mais fundamental: se todas teorias forem vendidas não existirá teoria para se vender amanhã.

Podemos pensar que Santos realiza uma metáfora de um Mercado contemporâneo do conhecimento; nos termos deste documento, espaços conceituais da Filosofia e da Economia se conectam por imagens mentais e metáforas. No entanto Santos argumenta que tal imagem mental não é uma metáfora contemporânea, mas é anterior à formação do espírito científico moderno: no texto satírico *A venda de filosofias* (165), Luciano de Samósata (125 – 181?), escreve sobre um mercado estimulado por Zeus e gerenciado por Hermes:

Hermes atrai os potenciais compradores, todos comerciantes, gritando alto e bom som “À venda! Uma variedade sortida de filosofias vivas! Posições de todo o tipo! Pagamento à vista ou mediante garantia!” (1905: 190). A “mercadoria” vai sendo exposta, os comerciantes vão chegando e têm o direito de interrogar cada uma das filosofias à venda, começando invariavelmente com a pergunta pela utilidade para o comprador e a sua família ou grupo. O preço é estabelecido por Zeus que, por vezes, se limita a aceitar ofertas feitas pelos comerciantes compradores. A venda tem pleno êxito e Hermes termina, ordenando às teorias que deixem de oferecer resistência e sigam com os seus compradores, ao mesmo tempo que avisa o público: “Senhores, esperamosvê-los amanhã. Estaremos oferecendo novos lotes úteis para homens comuns, artistas e comerciantes”

Recolhendo esta imagem mental do Mercado de conhecimentos escravizados, espelhamos a metáfora para o assunto específico deste documento. As filosofias vendidas estão em uma feira chamada *live coding*, que traduzimos por improvisação de códigos. A tradução literal *codificação ao vivo* é o termo mais adequado; mas não cria a imagem mental necessária para apresentar um modelo lógico de improvisação musical (ver Capítulo 3, p. 47). Segundo McLean (2011, p. 129), “O termo *live coding* emergiu em torno de 2003, para descrever a atividade do grupo de praticantes e pesquisadores que começaram a

desenvolver novas abordagens na criação de música computacional e animação de vídeo.”⁴. Ali vendem o *jazz*, a música algorítmica, o minimalismo, música ambiental, música *rave*, e a música-ruído. Além disso são vendidas teorias da tecelagem, do audiovisual, da dança, e do lado científico, as ciências antropológicas e cognitivas. Compramos uma amostra, cujo exemplares foram divididos em três grupos (ver Objetivos, p. xiii).

Espaço Conceitual do Pensamento Ortopédico no *live coding*

Neste documento, atentamos para uma redução do escopo de investigação mais detalhada. No entanto esta própria redução, orientada pela prática científica moderna é comentada e classificada por Santos. São dois os tipos de limitações descritas: o Pensamento Ortopédico e Ecologia de Saberes. Não é nossa intenção discutir a Ecologia de Saberes⁵, mas atentar para o Pensamento Ortopédico que permeia esta prática de improvisação (ou pelo menos foi assim que percebemos).

O conceito de Pensamento Ortopédico pode ele mesmo ser tema de investigação sobre criatividade, limitados em alguns comentários sobre o termo, para que seja coerente com a metáfora inicial deste trabalho: a descrição simbólica do objeto de pesquisa, como uma feira de filosofias musicais acompanhadas de pesquisas científicas.

O termo PENSAMENTO ORTOPÉDICO , isto é, “o processo especialista do médico responsável em corrigir deformidades do corpo”⁶ , pode ser exemplificado, na improvisação de códigos, em dois momentos. No final do primeiro capítulo, Ward et al. (2004) recorrem aos termos *generativo* e *processo* (como agenciamento do tempo musical). Tais apropriações são revistas por McLean (2011), onde generativo é substituído por descriptivo, e o processo é substituído pela bricolagem.

A segunda metáfora, A RAZÃO INDOLENTE , isto é, “A carência a respeito da finitude transforma-se num problema técnico-científico, enquanto a carência a respeito da diversidade infinita é ignorada como um não-problema” (SANTOS, 2008, p. 15). A (in)finitude sobre um conceito , a criatividade, transformada em um problema técnico-científico será demonstrada como um método de análise, baseado no *Quadro Conceitual de Sistemas Criativos* (McLean, 2006; FORTH; WIGGINS; MCLEAN, 2010), onde metáforas e imagens mentais de uma improvisação de códigos são reduzidos como um objeto com diversas propriedades.

A terceira metáfora é O PENSAMENTO ABISSAL. Santos (2007, p. 1–4) discute este conceito da seguinte forma:

⁴ Tradução nossa de “The term live coding emerged around 2003, to describe the activity of group of practitioners and researchers who had begun developing new approaches to making computer music and video animation”

⁵ Cf. SANTOS, 2007.

⁶ Disponível em <<http://www.priberam.pt/dlpo/ortopedia>>

Consiste num sistema de distinções visíveis e invisíveis, sendo que as invisíveis fundamentam as visíveis. As distinções invisíveis são estabelecidas através de linhas radicais que dividem a realidade social em dois universos distintos: o universo 'deste lado da linha' e o universo 'do outro lado da linha'. A divisão é tal que 'o outro lado da linha' desaparece enquanto realidade, torna-se inexistente, e é mesmo produzido como inexistente. (...) **O pensamento abissal moderno salienta-se pela sua capacidade de produzir e radicalizar distinções.** Contudo, por mais radicais que sejam estas distinções e por mais dramáticas que possam ser as consequências de estar de um ou do outro dos lados destas distinções, elas têm em comum o facto de pertencerem a este lado da linha e de se combinarem para tornar invisível a linha abissal na qual estão fundadas.⁷

Na improvisação de códigos, as radicalizações das distinções são discutidas a partir de modificações em uma pergunta, e o tratamento dado a elas pela comunidade de improvisadores-programadores europeus. "Como as pessoas improvisam?"⁸ (Pressing, 1987). No caso específico deste trabalho, a partir de Pressing (1984): "como pode o psicólogo não-especialista acessar os resultados de tais improvisações quando os especialistas podem discordar entre eles?"⁹.

No caso específico deste documento, que investiga a improvisação de códigos, como o musicólogo não-especialista pode acessar os resultados de improvisações de códigos musicais, quando os programadores e musicólogos especialistas podem discordar entre os conceitos e idéias estabelecidas?

Deste problema radicalizado, outras ordens de conhecimento são acessadas. Nos termos deste trabalho, novos espaços conceituais são combinados, explorados e transformados, o que coloca em xeque a própria especialização da investigação. Neste sentido, buscamos limitar a pesquisa em *i*) Descrever alguns comportamentos criativos psicológicos. *ii*) Descrever comportamentos criativos históricos; *iii*) Descrever um pensamento ortopédico do comportamento criativo; *iv*) Analisar um comportamento criativo que gera um algoritmo de uma sonoridade tonal.

Objetivos

- Investigar um Universo de Conceitos sobre a *improvisação de códigos (live coding)*;
- Investigar um Espaço de Conceitos, historicamente restrito, sobre a improvisação de códigos;
- Investigar um método de análise/criação para uma improvisação de códigos;

⁷ Grifo nosso.

⁸ Tradução nossa de "How do people improvise?"

⁹ Tradução nossa de "how will the non-specialist psychologist assess the results of such improvisations when the specialists may disagree among themselves?"

- Investigar um Espaço Conceitual de uma improvisação de códigos, *A Study in Keith* (2009) de Andrew Sorensen, e seu algoritmo musical em um ciclo de transformação.

Estrutura dos Capítulos

Inicialmente, os [Capítulo 1](#) e [Capítulo 2](#) eram um só capítulo. Com a densidade de informações, foi necessário dividir em dois capítulos. De certa forma, esta divisão é um reflexo do universo psicológico do autor deste documento, e de uma aproximação do universo histórico da improvisação de códigos. No [Capítulo 1](#) selecionamos três abordagens, escolhidas por manterem alguma conexão com a improvisação de códigos, no contexto musical. No [Capítulo 2](#) buscamos contextualizar características históricas e musicais da improvisação de códigos. No [Capítulo 3](#) apresentamos um modelo de formalização da criatividade, do ponto de vista do Modelo de Improvisação discutido por [McLean \(2006\)](#), [Forth, Wiggins e McLean \(2010\)](#), [McLean \(2011\)](#) (segundo objetivo). No [Capítulo 4](#), organizamos conceitos para analisar o contexto de um algoritmo gerador de uma sonoridade tonal em *A Study in Keith* (2009) de Andrew Sorensen. O [Apêndice A](#) foi adicionado para expor o material que estimulou o interesse pelo tema discutido, onde conectamos a idéia de *nuvem de palavras* ao *Universo de Conceitos*. O [Apêndice B](#) sugere a inclusão de um trabalho de [Mathews e Moore \(1970\)](#) no âmbito proto-histórico da improvisação de códigos, bem como a consideração de sujeitos sócio-técnicos como a tecnologia JIT.

1 Definições de base da Improvisação de códigos

Giovanni Mori (2015a, p. 117) define a improvisação de códigos em relação à Música, Imagens em Movimento, Dança ou Tecelagem. É importante esclarecer que essa definição denota a aplicação em qualquer outra área, não apenas como metáfora, mas como estratégia de gerenciamento de fluxos criativos, assistidos por computador:

“*Live coding* é uma técnica artística de improvisação. Pode ser empregada em muitos contextos diferentes de performance: dança, música, imagens em movimento e mesmo tecelagem. Eu concentrei minha atenção no lado musical, que parece ser o mais proeminente.”¹

O problema desta definição é que ela não contempla uma classe puramente técnica da improvisação de códigos, como por exemplo os registros audiovisuais de um tutorial de como escrever um aplicativo *web*². No entanto, a premissa da pesquisa é, assim como Mori, situar a improvisação de códigos do ponto de vista musical. Descartamos uma discussão específica sobre imagens em movimento para evitar cair em uma digressão infinita; mas uma menção será feita em relação ao contexto musical. A discussão sobre tecelagem (ver [seção 1.1](#), p. 2) contextualiza a história da computação, com suas bases conceituais do que é computação, com a prática artística investigada. Além disso, improvisadores-programadores influentes, Alex McLean, Dave Griffths (e Adrian Ward), serão apresentados. A discussão sobre Dança apresenta a Música Eletrônica para Dançar³ (ver [subseção 1.2.1](#), p. 8), e a coreografia que nega o som como resultado da improvisação de códigos (ver [subseção 1.2.2](#), p. 12). Introduzindo a discussão musical (ver [seção 1.3](#), p. 17), apresentamos alguns improvisadores brasileiros, porém nosso foco último será discutir um conjunto de regras práticas que emergiram durante uma disputa acadêmica. Isto é, a partir de uma problematização feita por Schloss (2003), sobre o papel cênico (e visual) do músico que carrega um computador no palco de performance, uma resposta conjunta de McLean, Griffths, Amy Alexander, Adrian Ward, Fredrik Olofsson, Julian Rohrhuber e Nick Collins, (WARD et al., 2004), possibilitou a proliferação de idéias e a estruturação de um programa de investigação científica próprio para a improvisação de códigos (ver [seção 2.3](#), p. 37).

¹ Tradução nossa de “*Live coding is an improvisatory artistic technique. It can be employed in many different performative contexts: dance, music, moving images and even weaving. I have concentrated my attention on the music side, which seems to be the most prominent.*”

² Disponível em <<https://www.youtube.com/watch?v=dHtyDron5ik>>

³ Cf. RIETVELD, 2013.

1.1 Tecelagem

Contextualizar a atividade têxtil é uma forma de criar uma imagem mental, de como funciona o processo de computação. Seria possível usar a imagem de um ábaco. Mas essa última imagem não considera potenciais leitores de um programa de pesquisa que inclui investigadores na área de Moda (como é o caso do Programa de Pós Graduação em Artes, Cultura e Linguagens da Universidade Federal de Juiz de Fora). Não aprofundaremos o assunto de Moda, mas sim buscamos ilustrar um código de computador. Nas palavras de Griffiths (2015a),

“Um dos potenciais da tecelagem que eu fiquei mais interessado é a capacidade de demonstrar fundamentos de *softwares* por fios – parcialmente tornar a natureza física da computação auto-evidente, mas também como uma maneira de modelar novas formas de aprender e a entender o que são os computadores.”⁴

Buscamos demonstrar na próxima seção esta natureza física da computação, para auxiliar a inclusão da tecelagem na definição de Mori.

1.1.1 Charles Babbage e Joseph-Marie Jacquard

Os computadores atuais são máquinas desenvolvidas com base no modelo teórico elaborado por Alan Turing (1912-1954). Uma representação simplista considera uma fita abstrata de tamanho variável (o quanto for necessário), dividida em células, cada uma com um alfabeto finito. Cada alfabeto possui uma quantidade de símbolos de representação finita. Um cabeçote leitor desta fita lê as instruções escritas em cada célula, e depois passa para a próxima célula. Um registrador de estados desta fita, memoriza qual foi a última operação realizada na última célula executada. Uma tabela de ações indicará novas instruções, que serão escritas nesta fita.

Um modelo anterior ao de Turing foi elaborado por Charles Babbage (1791 – 1871), a *máquina analítica*, entre 1834 e 1836, revisado em 1837. Sua construção ocorreu após um colapso na construção de sua *máquina diferencial*. O projeto não vingou, mas a partir de 1838, Babbage se envolveu com a exploração intelectual dos conceitos elaborados, para otimizar o projeto e reduzir seu custo de construção. Uma sequência de seminários em Turin (1840) resultou em uma publicação sobre a máquina analítica, em francês, escrita por um cientista italiano (L.F. Menebrea). A Condessa de Lovelace (Ada Augusta Byron King), traduziu, sob supervisão de Babbage, esta publicação para o inglês. Historicamente, os primeiros programas de computador (para serem executados na máquina analítica) foram escritos ambos por Ada e Babbage. O primeiro programa escrito era dedicado ao

⁴ Tradução nossa de “One of the potentials of weaving I’m most interested in is being able to demonstrate fundamentals of software in threads – partly to make the physical nature of computation self evident, but also as a way of designing new ways of learning and understanding what computers are.”

cálculo de uma sequência numérica conhecida como *Números de Bernoulli*⁵. Apenas uma parte da máquina foi construída antes da morte de Babbage⁶.

Segundo McLean (2011, p.14–21), o mecanismo do projeto de Babbage é inspirado na máquina de tear de Joseph-Marie Jacquard (1752 – 1834). A principal contribuição da invenção, para a computação, foi um sistema que consiste em um cabeçote leitor de cartões perfurados. Na máquina de tear de Jacquard, a organização dos furos indicam, até hoje, uma rotina têxtil. Já no computador mecânico de Babbage, o cartão perfurado indicava estados binários que conduzem ao cálculo numérico:

“A indústria têxtil vislumbrou a primeira máquina programável de ampla utilização: a cabeça de tear Jacquard, uma tecnologia ainda usada. Longas tiras de cartão são alimentados na cabeça de tear Jacquard, que lê padrões perfurados no cartão para guiar a intrincada padronização de tecidos. O cabeçote Jacquard não computa, mas foi admirado por Charles Babbage, inspirando o trabalho na sua máquina analítica mecânica, a primeira concepção de um computador universal programável. Embora Babbage não tenha obtido sucesso em construir a máquina analítica, seu projeto inclui um mecanismo de entrada de cartão similar ao cabeçote Jacquard, mas com padrões perfurados descrevendo cálculos abstratos ao invés de fios têxteis.”⁷

1.1.2 Weavecoding

Se até hoje o mesmo sistema de Jacquard é utilizado para materializar imagens mentais de formas geométricas, como improvisadores de códigos elaboram uma *estratégia transversal* (ver subseção 3.1.3, p. 54), de transformar a imagem mental destas formas geométricas em códigos, e de códigos para o resultado desejado? De outra forma, como é realizada uma improvisação de códigos audio visuais, musicais e têxteis? Exemplificamos o caso com um grupo que criou o conceito *weavecoding*. Sua definição será dada conforme apresentamos o grupo *Weaving codes*⁸, em um encontro no espaço Foam Kernow⁹.

O grupo *Weaving codes* foi formado para a “(...) investigação de padrões a partir das perspectivas de tecelagem e música, e através do desenvolvimento de uma linguagem

⁵ Allan G. Broomley, *Charles Babbage's Analytical Engine*, 1838. Disponível em <<http://athena.union.edu/~hemmend/Courses/cs80/an-engine.pdf>>

⁶ Disponível em <http://www.sciencemuseum.org.uk/objects/computing_and_data_processing/1878-3.aspx>

⁷ Tradução nossa de “ *The textile industry saw the first programmable machine to reach wide use: the head of the Jacquard loom, a technology still used today. Long strips of card are fed into the Jacquard head, which reads patterns punched into the card to guide intricate patterning of weaves. The Jacquard head does not itself compute, but was much admired by Charles Babbage, inspiring work on his mechanical analytical engine (Essinger, 2004), the first conception of a programmable universal computer. Although Babbage did not succeed in building the analytical engine, his design includes a similar card input mechanism to the Jacquard head, but with punched patterns describing abstract calculations rather than textile weaves.* ”

⁸ Disponível em <<http://kairotic.org/about/>>

⁹ Disponível em <<http://fo.am/kernow/>>

de computador e código para descrever a construção de tecidos”¹⁰. É formado por membros da Universidades de Leeds, Nottingham Trent, Cambridge, Aberdeen, Copenhague; um museu (*Albert Museum*), uma rede de laboratórios transdisciplinares (FoAM Kernow); o Centro Dinamarquês para Pesquisa Têxtil, e Escola Robert Schumann de Música e Mídia de Düsseldorf.

Uma pequena digressão: dois membros deste grupo, Alex McLean e Dave Griffths, são praticantes e organizadores de improvisações de código como artistas-programadores. A principal contribuição dos autores foi uma heurística da improvisação de códigos, *Lubeck04*, mais conhecido como “Mostre-nos suas telas”¹¹, dentro de um manifesto publicado como “Programação de Algoritmo Ao vivo e Organização Temporária para sua Promoção”¹² (WARD et al., 2004). Este tema será discutido adiante (ver [subseção 2.4.1](#), p. 41).

Do manifesto à codificação têxtil, Griffths (2015a) apresenta um interessante exemplo. A partir de quatro tarefas fundamentais (rotinas), descritas no Exemplo 1.1, é possível elaborar um padrão como o apresentado na [Figura 1](#). A primeira rotina é *repeat*, uma repetição de ações por contagem, ou laço iterativo (*loop*); a segunda é *twist*, ou dar a volta em determinados pontos; a terceira, *weave-forward*, tecer à frente do ponto; e a quarta, *weave-back*, tecer atrás do ponto . Do lado direito da imagem (ver p. 5), é simbolizado o “código criptografado de tecido”, ou as operações fundamentais para um determinado padrão têxtil. Do lado esquerdo, seu resultado-padrão, uma textura de losangos e zigue-zagues.

Exemplo 1.1 (Um código-fonte que gera um tecido semelhante à [Figura 1](#).)

```
(twist 3 4 5 14 15 16)
(weave-forward 3)
(twist 4 15)
(weave-forward 1)
(twist 4 8 11 15)
```

```
(repeat 2
  (weave-back 4)
  (twist 8 11)
  (weave-forward 2)
  (twist 9 10)
  (weave-forward 2)
  (twist 9 10)
  (weave-back 2)
  (twist 9 10))
```

¹⁰ Tradução nossa de “We pursue these questions in the Weaving Codes- Coding Weaves project, by investigating patterns from the perspectives of weaving and music, and by developing a computer language and code for describing the construction of weaves”

¹¹ Tradução nossa de “Show Us Your Screens”

¹² Tradução nossa de “Live Algorithm Programming and Temporary Organization for its Promotion”

```
(weave-back 2)
(twist 8 11)
(weave-forward 4))
```

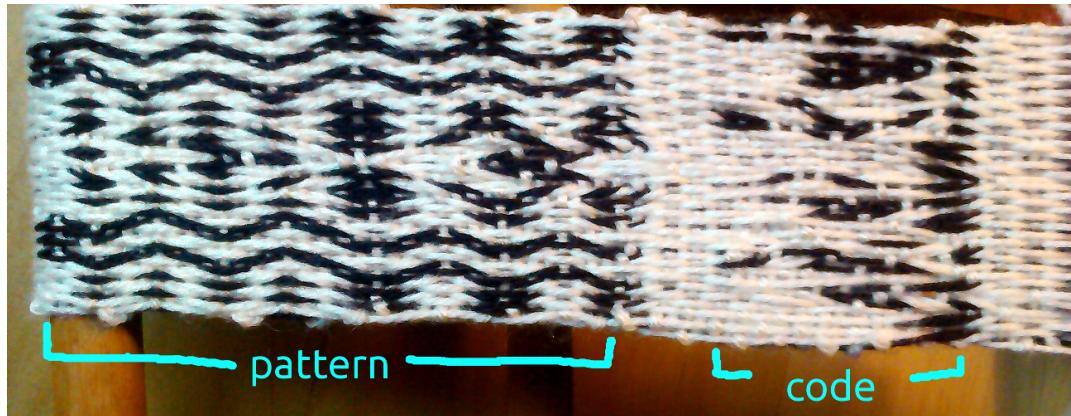


Figura 1 – Tecido resultante da prática *Weaving code*. **Fonte:** Griffths (2015a).

Esta experiência de *weavecoding* pode ser aplicada em uma performance. Griffths ilustra uma performance arquetípica da improvisação de códigos: programadores escrevendo enquanto os resultados são projetados em superfícies planas (ver Figura 2, p. 6). A tecelagem é programada por meio de um dispositivo tangível (ver Figura 3, p. 6), uma matriz de botões acopláveis, desenvolvida por Ellen Harlizius-Klück (investigadora da história da matemática, filosofia e tecelagem da Grécia Antiga na Universidade de Copenhague¹³) e Alex McLean (ver Figura 3, p. 6). Imagens em movimento foram projetadas como capturas das atividades têxteis e processadas por Griffths.

“ Uma das idéias originais era combinar tecelagem e codificação em um cenário de atuação, ambos para fornecer uma forma de tornar a codificação ao vivo mais inclusiva com a tecelagem, e ao mesmo tempo esclarecer os processos de pensamentos digitais envolvidos na tecelagem (...) Nossa audiência consistiu de pesquisadores de artesanato, biólogos antropológicos, arquitetos, designers de jogos e tecnólogos – foi mais do que antecipamos! Alex e eu disponibilizamos alguns códigos de música do *slub* para tecer, e minha parte favorita foi projetar a tecelagem ao vivo. ”¹⁴

¹³ Disponível em <<http://www.saumweberei.de/>>

¹⁴ Tradução nossa de “ One of the original ideas we had was to combine weaving and coding in a performance setting, to both provide a way to make livecoding more inclusive with weaving, and at the same time to highlight the digital thought processes involved in weaving. (...) Our audience consisted of craft researchers, anthropological biologists, architects, game designers and technologists – so it all went on quite a lot longer than we anticipated! Alex and I provided some slub livecoded music to weave by, and my favourite part was the live weaving projection. ”



Figura 2 – Performance no Foam Kernow. **Fonte:** Griffths (2015b).

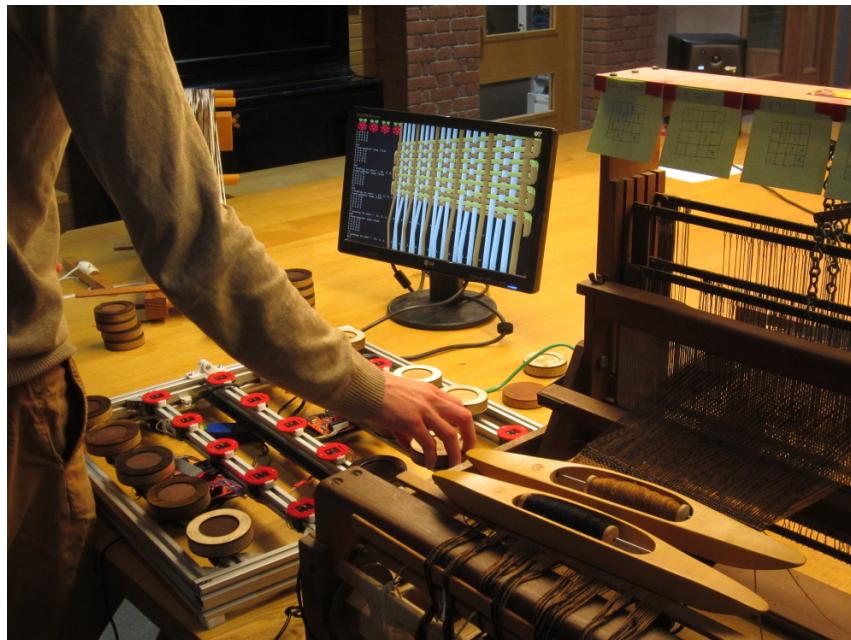


Figura 3 – Alex McLean manipulando uma Matriz de botões para tecelagem, conectado a um Raspberry Pi. **Fonte:** Griffths (2015b).

Esta descrição de Griffths possui uma documentação audiovisual ¹⁵. É interessante notar que a menção ao grupo *Slub* auxilia a discussão musical, e ao mesmo tempo articula esta seção com a próxima.

¹⁵ Disponível em <<https://www.youtube.com/watch?v=XrnIVUp9QgM>>

1.1.2.1 Slub

A banda *Slub* começou em 2000, como uma colaboração entre Adrian Ward e Alex McLean. A premissa do duo era utilizar a atividade de programação para realização de uma Música Eletrônica de Dança¹⁶. Sua primeira reunião foi em 2001, no *Paradiso club* em Amsterdã, durante o festival *Sonic Arts*. Em 2005 Griffiths se juntou ao duo durante o festival *Sonar*, o que abriu espaço para o desenvolvimento de uma estética de videogames (MCLEAN, 2011, p. 138–140).

É interessante notar que a improvisação de códigos já era mencionada em documentações de *softwares*, antes do *Slub* ou do manifesto de Ward et al. (2004). No entanto, as metodologias de improvisação de códigos são diversas. A utilização de ambientes como *SuperCollider*, *iXiLang*, PureData ou Max/MSP, estão restritas ao contexto de linguagens de domínios específicos¹⁷. O então duo *Slub* seguiu o seguinte caminho: a utilização de linguagens de propósito geral¹⁸, como Perl, REALBasic e Scheme (ver subseção 4.3.3, p. 76). Além disso, uma descrição destes sistemas GPLs oferecem uma característica interessante. Por exemplo, a justificativa de aceitação de uma música para dançar (ver subseção 1.2.1, p. 8) é feita através de uma postura bastante trivial, projetar o que está a fazer no computador; no entanto esta trivialidade não deve ser subestimada, afinal hoje ela é considerada como uma regra heurística. Segundo McLean (2011, p. 139):

“Um sistema Slub antigo é descrito em detalhes por Collins (2003). De maneira breve, ele apresentava um sintetizador, um antigo sistema de codificação ao vivo escrito por Ward, e uma série de programas geradores de batidas e linhas de baixo escritos por McLean. Embora o seu principal objetivo fosse musical, os membros do Slub gostavam de serem confrontados com o desafio de aceitação como programadores que fazem música. Para este fim, começou projetando suas telas de audiências com uma sobreposição conceitual, entre seu *softwares* artesanais, e a música que produziam com seu uso.”¹⁹

Em 2004, com a formação do TOPLAP (ver seção 2.4, p. 40), Ward e McLean focaram seus esforços no desenvolvimento de ambientes de improvisação de códigos. É interessante notar que os sistemas elaborados, são uma mistura de linguagens textuais, *patching* e interfaces gráficas de usuário²⁰: “O *Slub* controlava sua música usando interfaces criadas por e para eles mesmos. Eles variam desde as [interfaces] aparentemente convencio-

¹⁶ Cf. RIETVELD, 2013

¹⁷ *Domain Specific Language* ou DSL.

¹⁸ *General-Purpose Language* ou GPL.

¹⁹ Tradução nossa de “An early Slub system is described in detail by Collins (2003). In brief it featured a synthesiser and early live coding system written by Ward, and a Number of beat and bass-line generating programs by McLean. Although their primary aim was musical, Slub enjoyed being faced with the challenge of being accepted as programmers who make music. To this end they began projecting their screens audiences with the conceptual overlap between their and-crafted software and the music they produced using it.”

²⁰ *Graphical User Interfaces* ou GUIs.

nais para as abstratas, e das gráficas para as inteiramente textuais.”²¹ (COLLINS, 2003, p. 323). A descrição abaixo detalha algumas das funções destes programas, bem como um processo de composição por redes que será discutido em outra oportunidade (ver seção 2.2, p. 32):

“Por detrás das interfaces *slub* residem os processos ‘composicionais’ ou ‘musicais’ – muitos pedaços de códigos separados, escritos como exploração de idéias musicais. Cada pedaço de código descreve um experimento em áreas como matemática combinatorial, progressões de acordes, modelos sonificados para as pessoas dançarem, métricas que sofrem transformações, batidas sincopadas algorítmicas, e outros. (...) Estes processos compostionais enviam mensagens de um para o outro através de uma rede TCP/IP usando um protocolo de linha de comando. As mensagens viajam através de um servidor central, que administra a sincronização temporal entre os processos *Slub*. (...) O protocolo de rede resolve um problema que poderia, de outra forma, ser insolúvel: Adrian e Alex muitas vezes tomam abordagens muito diferentes para fazer música. Contudo eles não tem que argumentar sobre como a música é feita. Porque eles concordaram sobre, e implementaram um protocolo de rede, eles são livres para fazer música do jeito que gostarem, sabendo que seus programas irão sincronizar um com o outro.”²²

1.2 Dança

A Dança é ilustrada de duas maneiras. A primeira é dança como fim de uma improvisação musical. O código recodificado, e projetado de maneira semelhante ao *Slub*, corre o risco de ser a atração principal. O segundo caso foge do escopo sonoro; uma coreógrafa codifica apenas a orientação espacial de uma bailarina, resultando em uma sensação de quietude sonora. É uma posição que diverge da maioria dos trabalhos apresentados mas é pouco discutida no âmbito musical.

1.2.1 Algorave

O *algorave* é, em um sentido, essencial para o entendimento de categorizações musicais²³ de um grupo social. Este grupo social, que com seus diferentes/semelhantes sujeitos, cria artefatos artísticos frutos das relações entre as pessoas, o que por outro lado, possibilita

²¹ Tradução nossa de “*Slub control their music using user interfaces created by and for themselves. These vary from the apparently conventional to the abstract, and from graphical to entirely textual.*”

²² Tradução nossa de “*Behind the slub interfaces lie the ‘compositional’ or ‘musical’ processes – many separate pieces of code written as explorations of musical ideas. Each piece of code describes an experiment in such areas as combinatorial mathematics, chordal progressions, sonified models of dancing people, morphing metres, algorithmic breakbeats, and so on. (...) These compositional processes send messages to one another other across a TCP/IP network using a line-based protocol. The messages travel via a central server, which also manages time sync between all the slub processes. (...) The network protocol solves a problem which might otherwise be unsolvable: Adrian and Alex often take very different approaches to making music. However, they don’t have to argue about how the music is made. Because they agreed upon and implemented a network protocol between their programs, they are free to make music however they like, knowing that their programs will synchronise with each other.*”

²³ Cf. JR.; Sá, 2003, 2006, 2009.

criar institucionalizações dos artefatos. Discutiremos uma dessas institucionalizações, bem como sujeitos sociais influentes neste contexto, de forma que sua liminaridade²⁴ transita entre a Academia, salas de concertos e casas noturnas (ver [subseção 2.4.1](#), p. 41). Tal institucionalização é chamada de TOPLAP (ver [seção 2.3](#), p. 37), e para descrevê-la, como uma construção social que possui ideologias, apresentamos algumas figuras influentes, não todas (pois não caberia no escopo de nosso trabalho), no cenário europeu.

Sobre o termo *algorave*, é interessante notar um comportamento criativo por combinação (ver [seção 3.1](#), p. 47). A compositora colombiana Alexandra Cárdenas, em entrevista com [Cheshire \(2013\)](#), aponta que Nick Collins e Alex McLean combinaram dois termos *algorithm*, e *rave*, durante uma *gig* (um termo utilizado no início do *jazz* para caracterizar um trabalho temporário). Após sintonizarem em uma estação de rádio, decidiram combinar a música transmitida com a programação de uma música semelhante:

“ Algorave ‘comecou como uma piada’, de acordo com Alex McLean, um pesquisador de música computacional e um dos três de uma banda chamada *Slub*, que têm improvisado códigos por 13 anos. Ele veio com um termo enquanto conduzia uma *gig* em Nottingham com seu amigo Nick Collins (que tocava “*datapop*” sob o nome Sick Lincoln) no final de 2011. ‘Nós sintonizamos em uma estação pirata tocando *happy hardcore*, e nós pensamos que seria bom programar alguma música *rave*.’ Deste então, McLean organizou oito *algoraves* informais no mundo. ”²⁵

Em seu artigo “Algorave: Live Performance of Algorithmic Electronic Dance Music”, [Collins e McLean \(2014, p. 356\)](#) sustentam que as estruturas das práticas do *algorave* são anteriores à improvisação de códigos, utilizadas por praticantes da Música Eletrônica para Dançar. O que relaciona o *algorave* e a improvisação de códigos é a prática de projeção do código (ver [seção 2.3](#), p. 37):

“ *Algorave* não é sustentado exclusivamente por *live coders*, mas estes têm mantido uma forte presença em todos os eventos até agora. É assim talvez porque a tradição do *live coding* de projetar telas motiva todo o esforço; onde algoritmos não estão visíveis por períodos de tempo durante uma *algorave*, se corre o risco das coisas parecerem muito como um evento de música eletrônica padrão. ”²⁶

²⁴ Cf. [PROSPERO, 2015](#)

²⁵ Tradução nossa de “ *Algorave "started as a joke", according to Alex McLean, a computer-music researcher and one-third of a band called Slub that's been live coding for 13 years. He came up with the term while driving to a gig in Nottingham with his friend Nick Collins (who plays "datapop" under the name Sick Lincoln) in late 2011. "We tuned into a pirate station playing happy hardcore, and we thought it would be good to program some rave music." Since then, McLean has organised eight informal algoraves around the world.* ”

²⁶ Tradução nossa de “ *Algorave is not exclusively a preserve of live coders, but they have maintained a strong presence at every event thus far. This is perhaps because the live coding tradition of projecting screens help motivates the whole endeavour; where algorithms are not made visible for periods during an algorave, we run the risk of things feeling much like a standard electronic music event.* ”

Focando em um recorte histórico da Música Eletrônica para Dançar, Collins e McLean descrevem uma sequência de eventos (desenvolvimentos de *softwares* e apresentações). Em 1992, Charles Ames disponibiliza o *Cybernetic Composer*, “um *software* com um sistema baseado em Inteligência Artificial que compõe música em uma variedade de estilos populares.”²⁷ Em 1994, o duo *Koan*, formado pelos DJs Daniel Roeth e William Grey, realizam adaptações para entretenimento com base no *ambient music* de Brian Eno (1978). *Aphex Twin* (Richard David James) cria em 1997 o termo *live club algorithm*. Em 1999, o protocolo para edição audiovisual ao vivo *bbcut* (COLLINS; OLOFSSON, 2003) é incluído nos *opcodes* do *CSound*²⁸, e do *Supercollider*²⁹. Em 2000 o então duo *Slub*, realizam performances, autodenominadas *generative techno*, com abordagem *gabba*. Em 2001 é identificada a utilização de redes neurais para composição de padrões semelhantes ao *drum'n'bass*. Em 2004 é fundado o TOPLAP em uma casa noturna de Hamburgo.

Ilustramos três casos recentes, onde a improvisação de códigos é uma técnica utilizada. Junto com a improvisação de códigos, são utilizados um instrumento eletrônico, voz, e um instrumento elétrico. O inglês Canute, o mexicano Mico Rex e a colombiana residente na Alemanha, Alexandra Cárdenas.

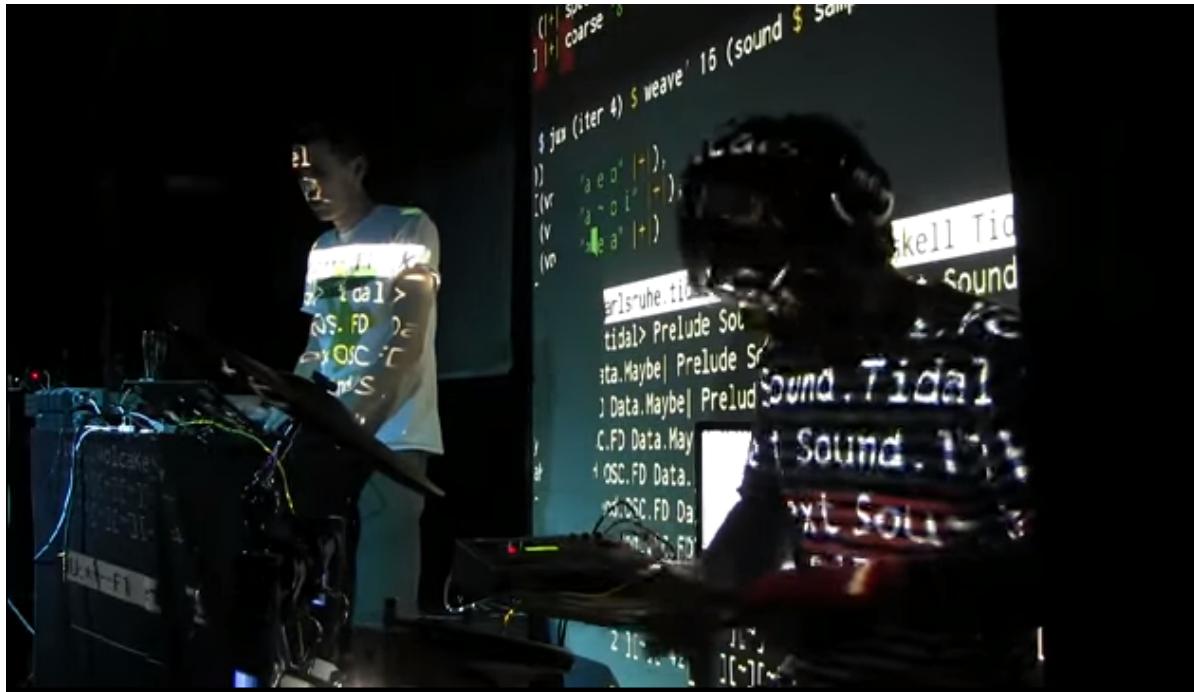


Figura 4 – Performance do duo Canute (Karlsruhe, 2015) **Fonte:** [Canute... \(2015-27-01\)](#).

O registro audiovisual do duo Canute, Matthew Yee-King (bateria eletrônica) e Alex McLean (*laptop*), reforça o arquétipo comentado anteriormente (p. 5). A recomendação

²⁷ Tradução nossa de “an AI based software system that composes music in a variety of popular styles. Disponível em <<http://www.kurzweilai.net/charles-ames>>”

²⁸ Disponível em <<https://csound.github.io/>>.

²⁹ Disponível em <<http://supercollider.sourceforge.net/audiocode-examples/>>

“Obscurantismo é perigoso, mostre-nos suas telas”(ver [subseção 2.4.1](#), p. 41) é seguida à risca. Categorizações musicais como *club* e *chordpunch* são mencionados na descrição do vídeo. É curioso notar que, em alguns momentos do vídeo, certas modificações nos códigos causam uma perturbação brusca em sistema de ritmos, percebido através do fluxo musical. Em alguns momentos Yee-King mantém o fluxo, mas em outros o instante musical codificado leva um curto período de tempo para ser sincronizado, o que leva Yee-King a se confundir, e por um breve instante, escutar o código e aí retornar à execução. Essa quebra no fluxo musical pode atrapalhar o fluxo de movimentos do corpo. No final deste capítulo, discutimos que este pode não ser *a priori* um erro do instrumentista, mas sim um problema entre o não-esforço cênico de McLean e o esforço de Yee-King. Esta questão cênica será mencionada na [subseção 2.4.1](#).

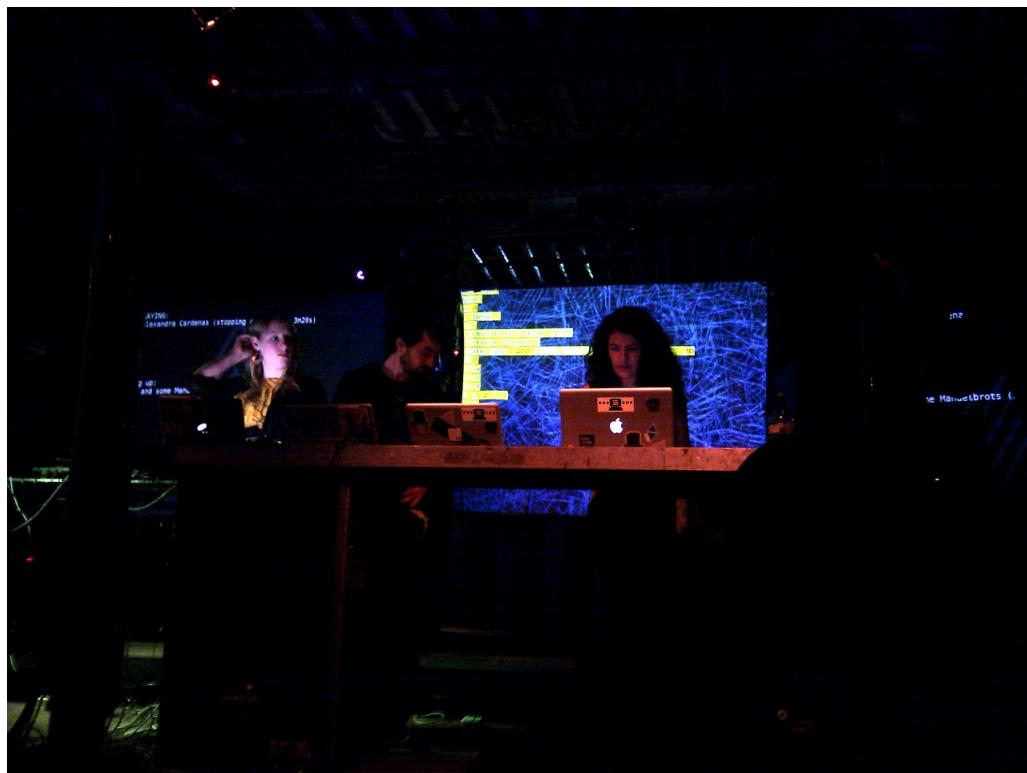


Figura 5 – Performance do duo Mico Rex (Londres, 2013) **Fonte:** [Griffths \(2013\)](#).

Griffiths registra uma *rave* na embarcação MS Stubnitz, em Canary Wharf, Londres, em 2013. Cárdenas e Ernesto Romero/Jorge Ramírez – Mico Rex, (ver [Figura 6](#), p. 12) – tocam neste evento. Encontramos um registro audiovisual de curta duração da apresentação do duo Mico Rex³⁰, mas não de Cárdenas. Exemplos sonoros específicos estão disponíveis nas redes sociais *SoundCloud* e *Vimeo*³¹. Notas descritivas nestes perfis especificam instrumentos e linguagens de programação, mas não o processo criativo. MicoRex utiliza voz e *programação* ao vivo com o ambiente de programação *SuperCollider*(ver [seção B.2](#),

³⁰ Disponível em <<https://vimeo.com/65309754>>

³¹ <<https://soundcloud.com/tiemposdelruido>> e <<https://soundcloud.com/micorex/>>.

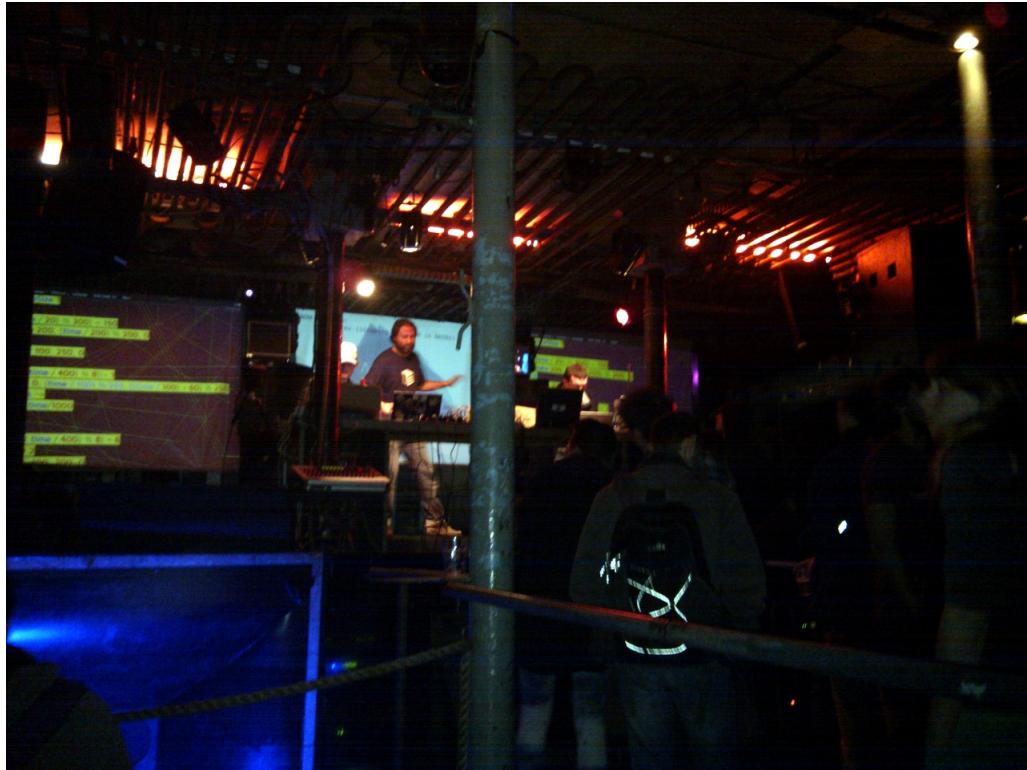


Figura 6 – Performance do duo Mico Rex (Londres, 2013) **Fonte:** Griffths (2013).

p. 116). Entre as categorizações musicais mencionadas, *electro-pop*, *8bits/glitch*, *electro*, *punk*, *bolero* e *breakz*. Cárdenas realiza suas performances com guitarra e programação ao vivo com o *SuperCollider*. É interessante que Cárdenas menciona a utilização de *techno*, *dubstep* e *noise* – além de distorções, utiliza um misto de instrumento expandido (objetos diversos jogados, friccionados, apoiados na guitarra) e retroalimentação de sinais com níveis saturados.

1.2.2 Coreografia

O segundo exemplo de um contexto de dança é projeto de Kate Sichio, *Hacking the Body/Hacking Choreography*. Dois trabalhos (*Hacking Choreography v.01* e *v.02*), mais um, *Hacking The Body 2.0*, serão colocados em discussão para ilustrar: *i*) uma estratégia metafórica do código em uma performance de dança; *ii*) uma estratégia transversal (ver subseção 3.1.3, p. 54) de codificação de movimentos ; *iii*) problematização do som ou imagem como resultado da improvisação.

Para Sichio, a relação entre a atividade de escrever programas, e a dança, é parte de um trabalho contínuo entre notação de coreografias e a improvisação de movimentos. É importante mencionar que, no escopo da pesquisa de Sichio, são raras as referências aos aspectos sonoros. Diferente das indicações estritas de um pentagrama musical, a partitura de dança oferece mais um guia para os movimentos do corpo. Isso significa sugerir que, do

ponto de vista computacional, a partitura coreográfica é mais próxima do código escrito em uma linguagem de computador do que a partitura musical tradicional. É curioso notar que ocorre como uma assimilação do pensamento algorítmico na Dança, chamado por Sicchio (2014, p. 31), através de Downie (2005, cap. 1, p. 3), de *Sensibilidades Computacionais*. Ou dispositivos metafóricos elaborados por coreógrafos como Merce Cunningham, Trisha Brown, Bill T. Jones, e William Forsythe – “mecanismos de generalização e abstração, representação da coreografia e dança como computação”³² (DOWNIE, 2005, cap. 1, p. 2–4):

“Esta sensibilidade computacional é presente em dois níveis em um trabalho destes coreógrafos. Primeiramente, em seus processos coreográficos – os sistemas, métodos, e notação através dos quais os coreógrafos criam a dança. Segundo, no trabalho ele mesmo, finalizado, que aparece no palco e é interpretado pelo observador. As primeiras invenções e proclamações de Cunningham – a democracia do espaço do palco, e a redescoberta do que está atrás do dançarino como ponto de origem do movimento – pode ser interpretado como generalizações do tipo; qualquer ponto do palco é a “frente”, e conectado por um conjunto de articulações pode ser pensado como um membro. O que eram constantes, uma vez especificados em uma descrição rígida, se tornam variáveis em uma estrutura gerativa.”³³

O projeto *Hacking the Body/Hacking Choreography* será descrito a partir de *Hacking Choreography beta v.01*, que vai até *Hacking Choreography beta v.04*, apresentados no *Lincoln Performing Arts Centre* em janeiro de 2012 e o último, em maio de 2013 no *Gnarl Festival* (Inglaterra).

1.2.2.1 Hacking Coreography

A experiência parte de uma primeira versão (*v.01*), um *hackeamento* de uma Partitura de Eventos do artista Alison Knowles (mais especificamente a peça de performance #8, de 1965) . Por exemplo, a partitura é projetada em um espaço de performance, e dançarinos lêem a partitura (ver exemplo 1.2), sem ensaios prévios:

Exemplo 1.2 (Partitura original de Alison Knowles (1965))

Divida uma variedade de objetos em dois grupos. Cada grupo é rotulado com "tudo". Estes grupos podem incluir diversas pessoas. Existe uma terceira divisão do palco, objetos vazios, rotulados com "nada". Cada um dos objetos é "alguma coisa". Um executante combina e ativa os objetos das seguintes maneiras para qualquer duração desejada de tempo :

³² Tradução nossa de “mechanisms of generalization and abstraction, choreography as representation, dance as computation”

³³ Tradução nossa de “This computational sensibility is present at two levels in the work of these choreographers. Firstly, in their choreographic processes — the systems, methods, and notations through which the choreographers create the dance. Secondly, in the finished work itself, as it appears on stage and as it is interpreted by the viewer. (...) Cunningham’s earliest inventions and proclamations — the democracy of the stage space, and the rediscovery of the dancer’s back as a point of origin of motion — can be interpreted as generalizations of a kind; any point of a stage can be a “front”, and any connected set of joints can be thought of as a limb. What were once specified constants in a rigid description become variables in in a generative framework.”

- "alguma coisa" com "tudo"
- "alguma coisa" com "nada"
- "alguma coisa" com "alguma coisa"
- "tudo" com "tudo"
- "tudo" com "nada"
- "nada" com "nada"

E seguem com pedaços de papéis, segundo a orientação dada por Knowles, até que executem a última instrução. No momento em que a última rotina de movimentos é executada, é esperado entrar em um laço iterativo (*loop*), e repetir as instruções desde o começo. Não é o que fazem os dançarinos, sendo que assimilam o personagem do *hacker*. Em outras palavras, ao completarem a partitura, são estimulados a desconstruir os rótulos uma vez organizados, e derivar novos rótulos para novas recombinações:

“Depois que a partitura foi completada, contudo, ela foi *hackeada*. Isso significa que o executante tenta de alguma forma contornar as instruções originais. Isto foi feito sem preparações prévias e a audiência assistiu isso se desdobrar enquanto era realizada. Nesta primeira performance, o papel e os rótulos foram rasgados para criar novas palavras e categorias (...) Então ao invés de “nada”[Nothing], foram formados dois grupos, “não”[No] e “coisa”[Thing].”³⁴

A segunda experiência, *Hacking Coreography v.02*, é inspirada na proposta de definir termos e associar uma ordem ao termo (como no exemplo anterior). Mas dessa vez, as orientações são escritas como um híbrido de texto discursivo, legível por um executante, e de código de computador em linguagem Java. Isto é, ele não é executável por um computador para resultar em sons, mas por um humano para resultar em movimentos.

Exemplo 1.3 (Exemplo de um hackeamento de partitura de movimentos)

```
/Dance/
set up()
{
dance a centre, right
dance b centre, left
}

movement()
{
move1 (dance a = rotate) (dance b = jump)
```

³⁴ Tradução nossa de “After the score was completed, however, it was then hacked. This meant that the performer had to try to somehow circumvent the original instructions. This was done with no previous preparation and the audience watched this unfold as the piece was performed. In this first performance, the paper and the labels were torn up to create new words and categories (...). So instead of “Nothing” there were two new groups, “No” and “Thing.””

```

move2 (dance a = brush) (dance b = lie down)
move3 (dance a = push) (dance b = run)
move4 (dance a = step) (dance b = kneel)
}

coreography()
{
if (dancer a = rotate right 180)
then both jump = 2 feet to 1
if (dancer b = travels)
then brush = right foot
}

run(){
move1
move4
move4
move1
move2
move3
move1
move2
move3
move4
}

/hack/
{
if (dancer a = kneel)
dancer a = kneel
if (dancer a = rotate)
dancer b = rotate opposite direction
}

```

Algumas seções são apresentadas como *funções* (*set up*, *movement*, *coreography* e *run*). A função *set up* define as posições iniciais de cada ator; *movement* define os tipos de movimentos que serão executados por intérpretes; *coreography* define uma estrutura de fluxo destes movimentos; e por último, uma ordem de execuções é estruturada em *run*. É interessante notar que Sichio aponta para um outro *hackeamento* da partitura. A utilização de números, como por exemplo na função *coreography*, dificultou a leitura dos intérpretes durante ensaios. Uma alteração na função *coreography*, notificada abaixo da linha */hack/*, foi feita pelos próprios intérpretes para alterar a notação numérica por uma descrição textual da ação. Isso tornou o código mais legível para humanos durante a execução de movimentos.

1.2.2.2 Hacking the Body

O *Hacking The Body 2.0*, ou *HTB2.0* (2015)³⁵ é uma performance mais formal. Se caracteriza por uma apresentação em um evento internacional, com um o palco-arena que reforça a hierarquia ator-público (o palco fica aproximadamente na linha da cabeça do espectador). A coreógrafa está sentada ao lado direito do palco, visível, mas debaixo de uma penumbra. Já a dançarina, cuja vestimenta branca equilibra com uma iluminação frontal e estática, expressa uma face de seriedade(ver Figura 7, p. 17). A coreografia segue uma partitura de movimentos corporais. Mas esta partitura não é visual (o registro não é feito em papel), o resultado da apresentação não são movimentos corporais pré-definidos, e tão pouco existe um acompanhamento musical. Esta partitura é um correlato acústico do tato, codificada por uma coreógrafa. Afastada de uma bailarina, controla a direção dos movimentos corpóreos da performance:

“Esta peça é uma exploração de eletrônica codificada ao vivo e movimentos improvisados. Uma dançarina veste uma peça de atuadores hapticos. Estes atuadores são programados em tempo-real via OSC³⁶ para ‘zunir’ sobre os lados direito e esquerdo da dançarina para indicar qual lado do corpo a dançarina deve mover. A partitura é codificada ao vivo pela coreógrafa enquanto a dançarina responde por uma retroalimentação haptica. Esta peça explora o *live coding* de corpos, e movimento como saída, ao invés de saídas sonoras ou visuais como encontrado em muitas execuções de *live coding* ”³⁷

É interessante notar uma sensação de quietude sonora ³⁸na performance, mas pouco discutida do ponto de vista musical. Como aponta a própria coreógrafa, a maioria das performances de improviso de códigos segue o seguinte procedimento: o código é criado, e um som, uma nota, uma imagem ou um vídeo são gerados, combinados, transformados de maneira contínua. Mas o padrão é a realização audiovisual. Mesmo em algumas performances de dança pesquisadas (e que não foram mencionadas neste documento), a dança e a projeção audiovisual se suportam. A criatividade deste trabalho toca um conceito técnico fundamental da computação: qual é o *dispositivo de entrada e de saída* praticado nas improvisações de códigos? Sicchio responde que o corpo já é um dispositivo de entrada e saída de interações sociais e pode ser controlado por outro humano através

³⁵ Disponível em <<https://www.youtube.com/watch?v=iOAffWTBVE0>>

³⁶ N.A.: “Open Sound Control é um protocolo de comunicação entre computadores, sintetizadores sonoros e outros dispositivos multimídia que são otimizados para as modernas tecnologias de rede”. Disponível em <<http://opensoundcontrol.org/introduction-osc>>

³⁷ Tradução nossa de “ *This dance piece is an exploration of live coded electronics and improvisational movement. A dancer wears a custom garment of haptic actuators. These actuators are programmed real-time via OSC to ‘buzz’ on the right and left sides of the dancer to indicate which side of the body the dancer will move. The score is being live coded by choreographer while the dancer is responding to the haptic feedback. This piece explores live coding of bodies and movement as output rather than a sonic or visual output as found in many live coding performances.* Disponível em <<http://iclc.livcodenetwork.org/performances.html>> . ”

³⁸ Cf. KOELLREUTTER; HAOULI; MANNIS, ,



Figura 7 – Dançarina (anônima) controlada por Kate Sicchio (2015) através de uma codificação improvisada. **Fonte:** <<https://www.youtube.com/watch?v=uAq4BAbvRS4>>.

de comandos de rede. A sensação de quietude é presente não por questões musicais, mas é definida por uma prática de dança.

1.3 Música Computacional na improvisação de códigos

Apresentamos três exemplos de pesquisadores brasileiros. O primeiro é uma nota sobre aquilo que consideramos como as primeiras manifestações de improvisação de códigos, em 2011, coerentes com algumas regras de performance e categorizações musicais descritas por Ward et al. (2004, ver [subseção 1.3.1](#), p. 17). O segundo é um exemplo que apresenta uma “(...) performance de *livecoding* arquetípica [que] envolve programadores escrevendo códigos no palco, com suas telas projetadas para a audiência.”³⁹(McLean; WIGGINS, 2010b, p. 1, ver [subseção 1.3.2](#), p. 20). O terceiro apresenta uma abordagem virtual, isto é, uma improvisação de códigos realizada em locais diferentes, com computadores diferentes, através de uma conexão de *internet* (ver [subseção 1.3.3](#), p. 24).

1.3.1 FooBarBaz

FooBarBaz é um grupo de improvisação de códigos formado por Gilson Beck, Renato Fabbri, Ricardo Fabbri e Vilson Vieira. Sua primeira apresentação foi durante

³⁹ Tradução nossa de “The archetypal live coding performance involves programmers writing code on stage, with their screens projected for an audience.”

o Festival Contato 2011. Os membros são ativos em um laboratório virtual conhecido como *labMacambira*⁴⁰. Além de cientistas, participam *hackativistas*, ex-programadores do Google, músicos e artistas plásticos interessados em processos criativos com assistência computacional. É interessante notar que as atividades do FooBarBaz estão sincronizadas com algumas das ideologias da improvisação de códigos, com base em regras práticas (ver [subseção 2.4.1](#), p. 41). Outro ponto interessante deste grupo foi a elaboração de um manifesto próprio, chamado de *Manifesto Freakcoding*, que inclue, como parte executável do manifesto, um ambiente de programação audiovisual chamado *Vivace* ([VIEIRA et al., 2015](#)).



Figura 8 – Festival Contato, 2011. [Fonte: Fabbri e Vieira \(2011\)](#).

Enumeramos três características que permearam as atividades do grupo, como FooBarBaz, e que ainda permeiam o contexto do *labMacambira*: *i*) abordagem ecossistêmica, ou diversidade de *softwares*; *ii*) atividade de escrita o mais simples possível; *iii*) codificação como processo psicosocial – humor como integrador de pessoas, como a inclusão de figuras afetivas, como um programa, *cowsay* (“uma vaca falante configurável”⁴¹), e de fotos de vacas se lambendo, e REM⁴² com o movimento repetitivo de uma imagem (ver [Figura 8](#), p. 18).

O membros do grupo FooBarBaz descrevem 3 dias de improvisações no Festival Contato, com a utilização de dois ambientes/linguagens de programação musical (Chuck

⁴⁰ Disponível em <<http://labmacambira.sourceforge.net/>>

⁴¹ Disponível em <<http://www.nog.net/~tony/warez/>>

⁴² *Rapid Eye Movement*, ou uma fase do sono

e PD), uma linguagem de propósito geral (Python), uma ferramenta de linha de comando (C) e dois diferentes editores de texto (Emacs e Vim). Focando na improvisação de códigos com o ChucK (ver, exemplo 1.4), é possível notar uma prática de improviso enraizada no sequenciamento de propriedades de uma amostra sonora, através da manipulação de listas. O escopo de cada lista, definido pelos colchetes [e], podem conter informações como o nome arquivo de áudio utilizado e, numericamente, sequências das posição de um *sample* da amostra sonora total, dos parâmetros de diferentes taxas de amostragem, de diferentes durações da amostra, e de diferentes níveis de ganho.

Exemplo 1.4 (Arquivo *foo.ck.*)

```
[ "samples/fx/s20.wav" ] @=> Foo.name;
[0.] @=> Foo.prop;
[.25, .15] @=> Foo.rate;
[2., 1., 1., 4.] @=> Foo.du;
[.8] @=> Foo.gain;
```

O funcionamento aparentemente simplista deste código é possível através do carregamento de outros dois códigos, sendo que o primeiro depende do segundo, um código elaborado por Graham Coleman⁴³, que estrutura uma *grade temporal* (TimeGrid).

Exemplo 1.5 (Arquivo *tg.ck.*)

```
//basic timing operations abbreviated

public class TimeGrid {

    dur beat;
    dur meas;
    dur sect;

    int nbeat;
    int nmeas;

    //phase and magnitude of offset
    float measPhase;
    dur measOffset;
    ...

    //sync to beat
    fun void sync() {
        beat - (now % beat) => now;
    }

    fun void sync(dur T) {
        T - (now % T) => now;
```

⁴³ Disponível em <<http://www.dtic.upf.edu/~gcoleman/>>

```

}

//how long to sync to this duration
fun dur syncDur(dur T) {
    return (T - (now % T));
}

//minimum time
fun dur tmin(dur a, dur b) {
    return (a < b) ? a : b;
}
...
}

```

Esta grade temporal é uma instância, ou seja, é possível criar diferentes grades temporais. Cada uma das memoriza o fluxo temporal através de propriedades, tais a posição de uma unidade de tempo (*beat*), compasso (*meas*) e seção (*sect*), e deslocamentos temporais (*measPhase*, *measOffset*). Existe também um mecanismo de sincronização (*sync*, *tmin*), o que caracteriza o conceito de níveis métricos da amostra sonora processada.

1.3.2 Música de concerto e imagens em movimento

A performance *screenBashing* de Magno Caliman (ver [Figura 9](#)) foi realizada durante o XIII ENCUN⁴⁴. A performance consiste no seguinte: Caliman senta-se ao computador, lateralmente à tela de projeção, com uma iluminação de penumbra. O projetor expõe o estado atual de seu *laptop*, que apresenta um editor de texto. O executante começa a programar em linguagem C (ver exemplo abaixo). Um pequeno laço iterativo (*for loop*) repete caracteres diversos, improvisados. Assim que termina de escrever, abre um console (ou terminal nos sistemas operacionais Unix) e compila o programa – a compilação é, de maneira superficial, um processo no qual a representação textual humana, definida pela ANSI⁴⁵, é convertida em códigos executáveis (binários, alguns representados como arquivos com extensões, como *.out*, *.bin* ou *.exe*). Este processo não demora, e o programa é executado. O resultado é uma sequência de caracteres de texto como texturas visuais. É importante lembrar que Caliman utiliza um recurso técnico do sistema operacional, que torna o terminal semi-transparente, o que permite sobreposições de texturas. A transformação da textura visual é levada a cabo através da modificação do argumento de entrada da função *printf* (ver exemplo 1.6, p. 22). Esse processo ocorre com uma sensação de quietude sonora (ainda assim, este aspecto não é explorado musicalmente, mas essa não é a intenção da música). O improvisador abre códigos preparados no ambiente de programação *SuperCollider*⁴⁶, que realiza a síntese de texturas sonoras ruidosas. Segundo o

⁴⁴ Encontro Nacional de Compositores Universitários em Campinas-SP no ano de 2015.

⁴⁵ Disponível em <<http://www.ansi.org/>>

⁴⁶ Disponível em <<https://supercollider.github.io>>

compositor, foram *sctweets*, pequenas mensagens como códigos do programa *SuperCollider*, na rede social *Twitter*⁴⁷. Um novo som ruidoso é adicionado para cada nova textura visual. Este processo, de alternar a codificação da textura visual, e depois executar um programa no *SuperCollider*, é repetido até o fim da peça. A acumulação de sons e caracteres em movimento cria um contínuo de texturas visuais e sonoras que se tornam uma mesma massa. Porém esta massa audiovisual sacrifica a memória RAM do computador. No auge de saturação audiovisual, a improvisação finaliza com um silêncio brusco, decorrente do congelamento do sistema operacional.

Nesta performance, a tradução “improviso de códigos” como correlato de *livecoding* (ver Capítulo 1, p. 1) se torna problemática. Segundo o próprio compositor, em comunicação pessoal:

Não acho totalmente adequado considerar como puramente um improviso. Existe, desde a concepção da peça, uma intenção de composição ali. Composição no sentido de agenciamento de materiais *a priori*, entende? Claro, existe um nível de abertura ali com relação a, por exemplo, os caracteres que eu uso no loop, ou nos valores dos parâmetros do *Supercollider* (...). Existem sessões formais ali que, se eu tocar a peça novamente, vão se manter. Daí vc já percebe uma determinação no âmbito macro da coisa também. Em uma situação limite, eu consigo imaginar até uma partitura (de execução) para essa peça, que permitiria que outra pessoa tocasse, sem ver o vídeo da minha performance, e chegasse em resultados bastante similares. Então eu vejo uma série de características aí que diferem de um improviso, onde se espera que uma parte grande do processo de tomada de decisão seja feita no momento da performance. Claro que essa distinção entre composição/peça/determinação e performance/improviso/indeterminação é complexa e a discussão foge totalmente do escopo do seu trabalho. Mas só para atentar para esse detalhe. Acredito que se imaginarmos um *continuum*, uma linha onde de um lado vc tem uma improvisação totalmente livre (impossível de se alcançar, claro) e do outro uma composição 100% determinada (tão impossível quanto), acredito que *screenBashing* está posicionada mais à direita...

O código sonoro foi pouco exposto, o que dificulta uma análise pormenorizada. Com uma valorização no aspecto imagético, para o público, é possível realizar análise do algoritmo visual, não para indicar a técnica, mas apontar a presença de uma ideologia. Limitamo-nos a comentar que o valor adicionado entre imagens em movimento, e sons, é contraposta à declaração do compositor. Por exemplo, a repetição dos algarismos seis e nove como uma menção direta (e espelhada) de signos do *black metal* abstrato. Ou seja, como ouvinte e espectador, ocorreu um processo sincrético entre a imagem, o código (que de certa forma, faz parte da imagem) e o som ruidoso. No entanto o compositor declara que esta foi uma decisão tomada na hora, como uma piada, embora tenha alguma relação. Mas isso não quer dizer que, para o compositor, a sonoridade se configura como *black metal*. É possível discutir se este tema ultrapassa questões de gosto ou prática sonora;

⁴⁷ Disponível em <<https://twitter.com/sc140tweets>>



Figura 9 – Performance de *screenBashing*. Fonte: <<https://vimeo.com/148626379>>.

ou seja, uma análise mais ampla de categorizações do gosto musical seria necessária⁴⁸. Por outro lado, uma segunda notação (ver subseção 3.1.2, p. 51) mais enxuta e sem erros, permite apontar a questão:

Exemplo 1.6 (Algoritmo inicial de Magno calimann)

```
# include <stdio.h>

int main()
{
    double i;

    for (int i=0; i<; i < 6666; i+=0.999)
    {
        printf("\\\\\" // \\\\"");
        fflush(stdout)
        system("sleep 0.0006")
    }
}
```

Execução:

```
# gcc - compilador
# a.c - arquivo em linguagem c
# -o - escreve o resultado em um outro arquivo
# a - arquivo binario alvo
gcc a.c -o a
```

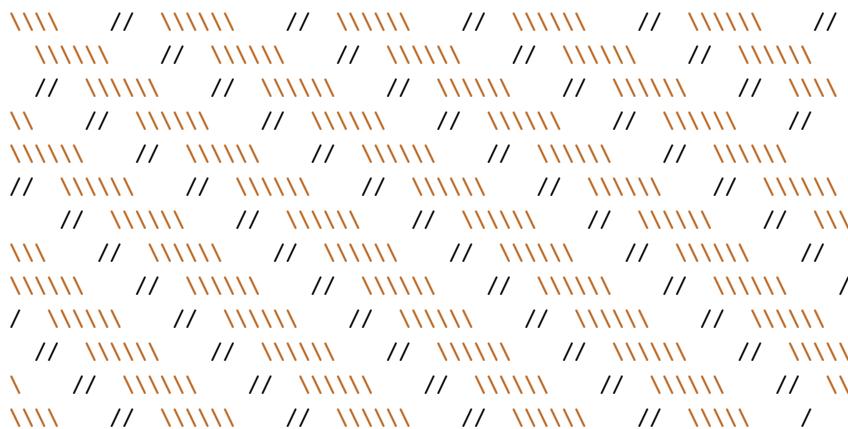
⁴⁸ Cf. JR.; Sá, 2003, 2009

Este código gera algumas mensagens classificadas como *warnings*, ou erros de lógica, que não atrapalhou na execução deste programa. Um conflito entre números de ponto flutuante (*float*) e números inteiros (*int*) é parcialmente ignorado pelo compilador do computador de Caliman:

Exemplo 1.7 (Resultado da compilação)

```
magno.c: In function 'main':
magno.c:7:12: error: conflicting types for 'i'
```

Resultado:



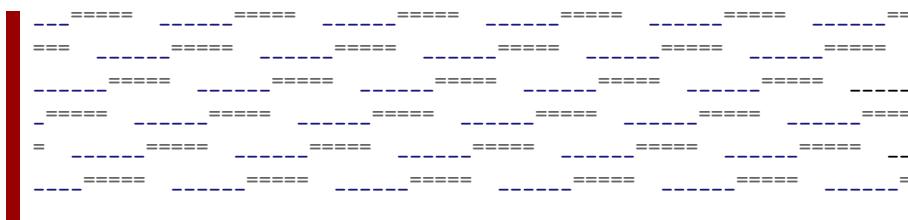
Correção para o algoritmo de Magno. Os novos caracteres foram inseridos por Caliman em 2'18":

Exemplo 1.8 (Correção)

```
# include <stdio.h>
int main(){
    int i=0;
    for (;;) i++{
        printf("===== -----");
        fflush(stdout);
        // 30 frames por segundo
        // ou um limite aproximado da
        // percepção humana (1/30)
        system("sleep 0.033");
    }
}
```

Resultado:





1.3.3 Telepresença e espaços virtuais

Uma performance virtual de *live coding* é aquela em que dois ou mais executantes, em endereços diferentes de uma rede de computadores. Isso situa três casos, do qual especificaremos um: i) uma rede local, com computadores diferentes, mas com os improvisadores fisicamente próximos; ii) uma rede remota, privada, que comunica um conjunto de pessoas fisicamente distantes; iii) a rede mundial de computadores, onde o navegador se torna o ambiente virtual de criação musical (ROBERTS; WAKEFIELD; WRIGHT, 2013). Nos três casos, a premissa é compartilhar o mesmo código entre improvisadores-programadores. Podem também compartilhar o mesmo som, mas isso depende de implementações técnicas.

O primeiro caso é bastante documentado nas *live coding sessions*, sessões de improvisações que ocorrem em encontros, simpósios e *workshops*, ricamente documentadas⁴⁹. Um exemplo do segundo caso é descrito por Junior, Lee e Essl (2015, p. 152–153). Uma performance telepresencial em 2014, por Ben Swift, Henry Gardner e Andrew Sorensen, realizada “entre dois intérpretes-programadores localizados na Alemanha e Estados Unidos usando um servidor SSH localizado na Austrália”⁵⁰. Junior, Lee e Essl ainda descrevem o desenvolvimento de um ambiente chamado *Supercopair*⁵¹, um ambiente cooperativo de improvisação de códigos, possível através de serviços de computação em nuvem.

Execuções remotas assíncronas (isto é, entre duas ou mais pessoas, em tempos diferentes) estão sendo exploradas a partir do uso musical de um navegador de *internet*. São realizadas em *apps* como *Gibber* (ROBERTS; KUCHERA-MORIN, 2012)⁵², *Wavepot*⁵³ e *Vivace* (VIEIRA et al., 2015), através da biblioteca *WebAudio API*⁵⁴. Um *software* na mesma direção foi desenvolvido, uma colaboração entre o autor deste documento e um membro da banca⁵⁵.

⁴⁹ Disponível em <<https://supercollider.github.io/archive>>

⁵⁰ Tradução nossa de “between two live coders located in Germany and United States using an SSH server located in Australia.”

⁵¹ Disponível em <<https://github.com/deusanyjunior/atom-supercopair>>

⁵² Disponível em <<http://gibber.mat.ucsb.edu/>>

⁵³ Disponível em <<http://www.wavepot.com>>

⁵⁴ Disponível em <<https://dvcs.w3.org/hg/audio/raw-file/tip/webaudio/specification.html>>

⁵⁵ Cf. LUNHANI; SCHIAVONI, 2015

1.4 Discussão

Neste capítulo delimitamos a discussão musical afim de ilustrar a improvisação de códigos do ponto de vista de uma bricolagem conceitual. Exemplificamos a prática de improvisar códigos em situações onde é possível seguir dois caminhos para criar artefatos artísticos: o primeiro segue a trilha do formalismo, com concertos, espetáculos em teatros. O segundo ocorre informalmente, e geralmente está ligado à diversão, ou um encontro como uma oficina. Por outro lado, esta divisão binária pode expandir para áreas como tecelagem, dança e música. Não discutimos a situação audiovisual em seus detalhes, pois é um campo tão particular quanto a música. No Próximo capítulo, discutiremos alguns fundamentos daquilo que foi chamado pela comunidade de improvisadores-programadores (*live-coders* audiovisuais de um Norte Global) por uma proto-história da improvisação de códigos (codificação ao vivo, ou *live coding*).

2 Definições Históricas da Improvisação de códigos

Este capítulo será dedicado à construção de um espaço conceitual histórico, do ponto de vista musical. Isto é, aqueles exemplos citados como “proto-históricos” que possuem alguma similaridade com o conjunto de regras práticas publicadas por Ward et al. (ver [seção 2.3](#), p. 37). Mori (2015b) descreve um caso prematuro de *live coding* na Itália, com o compositor Pietro Grossi (ver [seção 2.1](#), p. 27). Grossi sacrificou a questão timbrística para trabalhar na questão performática. As atividades de grupos californianos como *The League of Automatic Composers*, e *The Hub*, serão apresentadas no contexto cultural estadounidense, estimuladas pelo mercado emergente de microcontroladores (ver [seção 2.2](#), p. 32). O compositor Ron Kuivila será apresentado a partir de uma performance prototípica (sem projeção) de improvisação de códigos (ver [subseção 2.2.3](#), p. 37). Com um certo buraco cronológico, pulamos para meados da década de oitenta para o começo dos anos dois mil, período em que programadores ingleses, comprometidos com as artes visuais e a Música Eletrônica para Dançar, rebatem uma crítica sobre o papel cênico do músico durante uma apresentação com computadores (ver [seção 2.3](#), p. 37).

2.1 Pietro Grossi

Pouco conhecido no contexto geral da música européia, o compositor veneziano Pietro Grossi foi um dos pioneiros da *Computer Music Italiana*. De seu interesse pelos computadores como instrumento musical, o pensamento que rege seus programas de computador é bastante prático, e sacrifica questões timbrísticas para concentrar na performance. Isto é, o problema da diversidade timbrística é reduzido para uma única forma de onda, e o problema de performance é colocado em primeiro plano. Além disso, é possível notar a prática de transcrição de peças conhecidas. Nas palavras de Mori (2015b, p. 126):

“Grossi começou a se interessar por música computacional durante a primeira metade dos anos 60, quando ele organizou um programa de rádio centrado em torno da “música inovadora” em geral ([GIOMI; LIGABUE, 1999](#)). Contudo a primeira experiência de Grossi com um computador foi em Milão, no Centro de Pesquisa Elétrica da Olivetti-General. Aqui, auxiliado por alguns técnicos internos e engenheiros, ele conseguiu compor e gravar alguns de seus primeiros trabalhos em música computacional. Eles foram, em sua maior parte, transcrições de música clássica ocidental. Contudo, houve algumas exceções, por exemplo, uma faixa chamada Mixed Paganini”¹

¹ Tradução nossa de “*Grossi began to be interested in computer music during the first half of the 1960s, when he hosted a radio program centred around “innovative music” in general (Giomi1999). However,*

Existe um exemplo na *internet*². Um disco realizado no Studio di Fonologia musicale di Firenze, intitulado “GE-115 - Computer Concerto”, lançado pela Olivetti em 1967: “Do lado A existem algumas transcrições de música clássica, e do lado B existem três canções originais. (...) Este 7”[pode ser] distribuído como presente de natal e de ano novo pela companhia Olivetti.”³. No entanto, é necessária uma correção sobre o lado A, e um detalhe do lado B⁴. As transcrições realizadas foram da *Oferenda Musical BWV 1079* de J.S.Bach e um dos 24 Caprichos de Nicolò Paganini : *i) Canon a 2 Super Thema Regium; ii) Canon Perpetuum a 2 Quaerando Invenietis; iii) Canon a 3 Super Thema Regium e; iv) Capriccio nº 5.* As peças originais de Grossi foram três, sendo que uma dela é uma recomposição dos caprichos: *i) Mixed Paganini; ii) Permutations Of Five Sounds e; iii) Continuous.*

Mori explica que a peça *Mixed Paganini* derivou da transcrição do quinto *capriccio*: “Praticamente, Grossi modificou, auxiliado por alguns programas rudimentares, o material sonoro original. (...) Uma coleção posterior dos Capricci de Paganini, gravado em Pisa, foi revista por Barry Truax na Computer Music Journal ([TRUAX, 1984](#))”⁵ O tipo de material sonoro utilizado nestas peças utiliza um método tradicional, se comparada com o trabalho de Max Mathews. Por exemplo, as pesquisas desenvolvidas⁶ estão debruçadas na resolução do problema timbrístico, contraposto à capacidade de processamento dos *mainframes*. O caminho tomado nos EUA (e depois na Europa) seguiu a elaboração de algoritmos do timbre (Unidades Geradoras) para depois trabalhar sequência da programação-partitura. Para Grossi, com o problema da capacidade de processamento, os compositores deveriam esperar por melhores implementações técnicas dos engenheiros, e naquele momento, o computador foi capaz de maximizar o pensamento musical de um Período Comum (séc. XVII-XX, *circa*). Isto é, operações como inversão, retrogradação, retrogradação da inversão, aceleração, diminuição, eram executadas rapidamente com comandados.

Grossi não fica satisfeito com o trabalho, e a Olivetti não se interessa mais por suas pesquisas. Ao procurar emprego e novos espaços criativos, é contratado pelo “Centro de pesquisa IBM, dentro do Instituto CNR (*Centro Nazionale per la Ricerca*: Comitê

the first Grossi's experience with calculator took place in Milan, in the Olivetti-General Electric Research centre. Here, aided by some internal technicians and engineers, he managed to compose and record some of his first computer music works. They were, for the most part, transcriptions of Western classical music. However, there were some exceptions, for example a track called Mixed Paganini.”

² Disponível em <https://www.youtube.com/watch?v=ZQSP_wF7wSY>

³ Tradução nossa de “On side A there's transcribed classical music, on side B there are three original songs. (...). This 7” was distributed as a christmas and new year gift by the Olivetti company.”

⁴ Disponível em <<https://www.discogs.com/Studio-Di-Fonologia-Musicale-Di-Firenze-GE-115-Computer-Concerto/release/575632>>

⁵ Tradução nossa de “Practically, Grossi modified, aided by some rudimentary music programs, the original sound material. (...) A later collection of Paganini's Capricci, recorded in Pisa, was reviewed by Barry Truax on Computer Music Journal (Truax1984).”

⁶ Cf. MATHEWS; MATHEWS et al.; ROADS; MATHEWS; PARK; MATHEWS; NUNZIO, 1963, 1969, 1980, 2009, 2010

Nacional para a Pesquisa)”⁷ (*idem, ibdem*). Ali desenvolveu, em linguagem FORTRAN, o DCMP (*Digital Computer Music Program*), um programa integrado com um terminal de vídeo e um teclado alfanumérico, e segundo Mori, ao usar este terminal de áudio, o compositor escolheu deliberadamente abandonar o problema do timbre. Esta abordagem parte de uma abordagem “preguiçosa” (*prigo*). Grossi dizia sobre si mesmo, como “uma pessoa que está consciente de que o seu tempo é limitado e não quer perder tempo em fazer coisas inúteis ou na espera de alguma coisa quando não é necessária.”⁸ (*idem, ibdem*). Neste sentido, defendia que o desenvolvimento de novos timbres gerados por computador deveria esperar por melhores implementações de *hardware*:

“(...) o intérprete era capaz de produzir e reproduzir música em tempo real, digitando alguns comandos específicos e os parâmetros composticionais desejados. O som resultante vinha imediatamente depois da operação de decisão, sem qualquer atraso causado por cálculos. Havia muitas escolhas de reprodução no programa: era possível salvar na memória do computador peças de músicas pré-existentes, para elaborar qualquer material sonoro no disco rígido, para administrar arquivos musicais e iniciar um processo de composição automático, baseado em algoritmos que trabalham com procedimentos “pseudo-casuais”. Existia também uma abundância de escolhas para mudanças na estrutura da peça. Um dos mais importantes aspectos do trabalho de Grossi foi que todas intervenções eram instantâneas: o operador não tinha que esperar pelo computador terminar todas operações requisitadas, e depois ouvir os resultados. Cálculos de dados e reprodução sonoras eram simultâneos. **Esta simultaneidade não era comum no campo da Computer Music daquele tempo, e Grossi deliberadamente escolheu trabalhar desta forma, perdendo muito no lado da qualidade sonora. Seu desejo era poder escutar os sons resultantes imediatamente** (*idem, ibdem*).”⁹

Substituímos o termo “preguiçoso” por *reflexividade*, ou a “habilidade de um programa manipular como dados algo que representa o estado do programa durante sua própria execução, o mecanismo para codificação de estados de execução é chamado *reificação*.([MALENFANT; JACQUES; DEMERS, 1996](#), p. 1).”¹⁰ Parece existir apenas um

⁷ Tradução nossa de “IBM Research Centre in Pisa, inside the CNR Institute (*Centro Nazionale per la Ricerca: National Research Committee*)”

⁸ Tradução nossa de *a person who is aware that his or her time is limited and do not want to waste time in doing useless things or in waiting for something when it is not necessary*.

⁹ Tradução nossa de “(...) the performer was able to produce and reproduce music in real time by typing some specific commands and the desired composition’s parameters. The sound result came out immediately after the operator’s decision, without any delay caused by calculations. There were many reproduction choices inscribed in this software: it was possible to save on the computer memory pieces of pre-existing music, to elaborate any sound material in the hard disk, to manage the music archive and to start an automated music composition process based on algorithms that worked with “pseudo-casual” procedures. There were also plenty of choices for piece structure modifications. One of the most important aspects of Grossi’s work was that all the interventions were instantaneous: the operator had not to wait for the computer to finish all the requested operations and then hear the results. Data calculation and sound reproduction were simultaneous. This simultaneity was not common in the computer music field of that time and Grossi deliberately chose to work in this way, losing much on the sound quality’s side. His will was to listen to the sound result immediately.”

¹⁰ Tradução nossa de “the ability of a program to manipulate as data something representing the state of the program during its own execution, the mechanism for encoding execution states as data being

anseio em recuperar a reflexividade entre o dedo que toca a tecla e o som resultante. No entanto Mori (2015b, p. 127) coloca a figura do compositor como consciente dos problemas técnicos, e de um descarte pelo pensamento timbrístico corrente na Europa:

“O DCMP foi compilado na fase inicial do desenvolvimento de tecnologias computacionais. Naquele tempo, os recursos de cálculo eram escassos e, para obter a reprodução em tempo-real citada, era necessário pedir por pouca quantidade de dados. Contudo, o músico veneziano foi capaz escrever um programa muito leve, capaz de modificar somente os parâmetros necessários para um cálculo de recursos reduzidos: altura e duração. A síntese de timbres necessita de uma quantidade imensa de dados, e então a escolha foi descartá-la temporariamente, e todos os sons eram reproduzidos com o timbre de uma onda quadrada. Esta forma de onda era gerada por extração do estado binário do *pin* de saída da placa mãe que controla o programa. Essa saída tinha um único *bit*, e então a onda sonora gerada era o resultado desta mudança do estado binário. Desta forma, o computador não emprega quaisquer recursos para calcular a síntese sonora, economizando-os para o processo de produção musical. Grossi não estava interessado na qualidade da saída sonora em sua primeira fase em Pisa. O que importava particularmente era a capacidade em trabalhar em tempo real, ou, em outras palavras, para ter a escolha de escutar imediatamente ao que ele escreveu no teclado do terminal de vídeo (GIOMI; LIGABUE, 1999 apud MORI, 2015b).”¹¹

É importante situar que a escolha deliberada para DCMP é justificada nos anos 70. Até a metade da década, Grossi foi capaz de implementar melhorias de timbre, “digitalmente controladas, mas com uma tecnologia de síntese analógica. Foi lançado em 1975 e foi chamado de TAU2 (*Terminale Audio 2ª versione – Terminal de Áudio 2ª versão*) (*idem, ibdem*).”¹². Esta tecnologia tinha um programa, o TAUMUS, uma modificação do DCMP, que podia tocar:

“(. . .) até doze vozes simultâneas. Estas doze vozes eram divididas em três grupos, compostos de quatro canais cada. O operador poderia atribuir um timbre diferente para cada grupo, que era modulado usando síntese aditiva

called reification.”

¹¹ Tradução nossa de “The DCMP was compiled in the early phase of computer technology development. At that time, the calculation resources were low and, to obtain the just cited real time reproduction, it had to ask for very low quantity of data. Therefore, the Venetian musician chose to write very light software, able to modify only parameters that required a few calculation resources: pitch and duration. Timbre synthesis needed a big amount of data, so that choice was temporarily discarded and all the sounds were reproduced with square wave timbre. This waveform was generated by extracting the binary status of a motherboard’s exit pin controlled by the software. This exit had only one bit, so the sound wave generated was the result of this bit status changing. In this way, the computer did not employ any resources for calculating the sound synthesis, saving them for music production process. Grossi was not very interested in the quality of sound output in this first phase in Pisa. What he cared particularly was to be able to work in real time, or, in other words, to have the choice to listen immediately to what he typed on the video terminal’s keyboard.”

¹² Tradução nossa de “digitally controlled but with analog sound synthesis technology. It was launched in 1975 and called TAU2 (*Terminale Audio 2a versione – Audio Terminal 2nd version*)”

com sete sobretons. Cada sobreton era controlado individualmente pelo programa.”¹³

Segundo Mori (2015b, p. 128), Uma outra novidade do TAU2-TAUMUS, em relação às concepções do DCMP, era o conceito de modulação de modelos (*modelli modulanti*), ou “uma espécie de remendos que agem em um parâmetro musical”¹⁴. É importante notar que, ao aplicar um remendo (*patch*), através de comandos escritos com o teclado alfanumérico, o programa não interrompia o fluxo sonoro. “Esta era uma inovação crítica do ponto de vista performativo, porque então Grossi era capaz de tocar, e interagir, em tempo real com o programa, ao escrever instruções no teclado sem parar o fluxo sonoro.”¹⁵

Por outro lado, Grossi foi além deste problema reflexivo. É importante lembrar que, no final da década de 1970, Grossi contribuiu para o desenvolvimento de tecnologias telemáticas (ver [subseção 1.3.3](#), p. 24). O TAU2-TAUMUS sofreu uma considerável modificação, sendo que era possível controlar o sistema digital-analógico remotamente. O novo programa foi batizado de TELETAU Mori (2015b, p. 128–129). Uma descrição de 1986 aponta a possibilidade de acesso a um computador da CNR, em Pisa, com uma conexão da rede BITNET, que permitia de maneira virtual o acesso para qualquer programador. No entanto o TELETAU não vingou por diversos motivos: falhas e bugs que aumentavam de maneira dramática a manutenção e custos; o alto custo de transmissão e, por último mas não menos, a baixa qualidade da saída sonora devido à lentidão da conexão de dados.

“[Pietro] Grossi fez sua primeira experiência do tipo durante uma conferência de tecnologia em Rimini em 1970, onde o músico reproduzia algumas de suas composições, bem como sons randômicos, empregando um terminal de vídeo conectado pelo telefone para o computador da CNR em Pisa. A RAI, empresa de radiodifusão italiana, emprestou suas pontes de rádio [Comunicação entre duas antenas] para enviar sinais sonoros entre Pisa e Rimini. É como se fosse o primeiro experimento de telemática musical no mundo.(MORI, 2015b, p. 129)”¹⁶

¹³ Tradução nossa de “(. . .) twelve different voices simultaneously. These twelve voices were divided in three groups, composed of four channels each. The operator could choose to assign a different timbre to every single group, which was modulated using additive synthesis with seven overtones. Every overtone could be controlled individually by software.”

¹⁴ Tradução nossa de “they were a sort of patches that acted on some musical parameter.”

¹⁵ Tradução nossa de “ This was a critical innovation under the performative point of view, because then Grossi was able to play and to interact in real time with the software, by typing instructions on the keyboard without stopping the sound flux.”

¹⁶ Tradução nossa de “Grossi made his first experience of this kind during a conference on technology in Rimini in 1970, where the musician reproduced many of his compositions and random sounds as well, by employing a video terminal connected via telephone to the CNR’s computer in Pisa. RAI, the Italian public broadcasting company, lent its powerful FM radio bridges to send back sound signals from Pisa to Rimini. It is likely to be the first official experiment of musical telematics in the world.”

2.2 Baía de São Francisco

A prática musical com o computador, realizada na Costa Oeste dos EUA durante os anos 1970 e 1980, é bastante diversa daquela realizada em grandes centros europeus (como por exemplo Ircam ou o Conservatório de Haia). Brown e Bischof comentam que esta prática decorre de alguns fatores sociais. O primeiro fator seria uma diversidade musical existente na Costa Oeste dos EUA. O segundo fator é a transmissão de conhecimentos musicais propostos por Terry Riley, Pauline Oliveros e LaMonte Young, David Tudor e Gordon Mumma. Em especial, estes dois compositores propunham a utilização de circuitos eletrônicos eles mesmos como atores musicais (BROWN; BISCHOF, 2002, 3º parágrafo). Neste sentido, uma música computacional, colaborativa e livre de restrições de regras emerge em torno do *Mills College* em Oakland.

Com o florescimento da indústria de computadores pessoais na Baía de São Francisco, o acesso às novas tecnologias e pessoas que desenvolveram elas era talvez o melhor no mundo. Mas se para todos os jovens com fortunas como panos para suas mentes (e seus futuros) que perseguiam um excitamento aditivo na construção de máquinas eletrônicas, também existiam políticos utópicos que sonhavam com uma nova sociedade construída no livre e aberto acesso à informação, e na abrangente tecnologia baseada em sistemas inteligentes. Esta também é a cultura que deu ao mundo a música “New Age”, uma versão aguada e comercializada das músicas com base em modos e drones que Terry Riley, Pauline Oliveros, e LaMonte Young inventaram durante os anos cinquenta e sessenta. Mas a música feita na Costa Oeste também incluiam improvisações barulhentas e livre de restrições, que sobraram das revoluções contra-culturais dos anos 60(BROWN; BISCHOF, 2002, 1º parágrafo)¹⁷.

2.2.1 The League of Automatic Composers

Na segunda metade da década de setenta, Jim Horton começou a adquirir microcontroladores KIM-1¹⁸ com interesses musicais. Segundo Brown e Bischof, não demorou para que outros compositores interessados comprassem. Discussões informais posteriores, que incluiam, além de Horton, David Behrman e John Bischoff, Rich Gold, Cathy Morton, Paul Robinson, e Paul Kalbach. Em 1977 e 1978 Horton colaborou com duas peças, apresentadas no *Mills College*, que interligavam sistemas musicais elaborados com os microcontroladores (ver Figura 10, p. 33). A primeira peça era construída sobre algoritmos

¹⁷ Tradução de *With the flowering personal computer industry in the Bay Area, access to the new digital technologies and to the people who developed them was perhaps the best in the world. But for all the young men with fortunes in the back of their minds (and in their futures) who pursued the addictive excitement of building electronic machines, there were also the political utopians whose dream was of a new society built on the free and open access to information, and on a comprehensively designed technology based on embedded intelligence. This was also the culture that gave the world "New Age"music, a watered-down and commercialized version of the musics based on modes and drones that Terry Riley, Pauline Oliveros, and LaMonte Young invented here during the late fifties and early sixties. But West Coast music-making also included a free-wheeling, noisy, improvisational edge left over from the counter-cultural revolutions of the sixties.*

¹⁸ Keyboard Input Monitor. Disponível em <<http://www.6502.org/trainers/buildkim/kim.htm>>.

inspirados nas teorias matemáticas de Leonard Euler (séc. XVIII). A segunda peça também explorava a comunicação entre os microcontroladores, de forma que “notas ocasionais da minha [Bischof] máquina faziam a máquina de Jim transpor atividades melódicas de acordo com minha nota base(BROWN; BISCHOF, 2002, 5º parágrafo)”¹⁹. Em 1978, Bischof, Gold e Horton formaram uma banda nas proximidades de Berkley. Posteriormente Behrman se junta ao trio. No dia 26 de Novembro gravam um *Extended Play* (EP)²⁰ de quatro faixas no *Blind Lemmon*, um ponto de encontro musical fundado em 1958²¹. O disco foi lançado pela Lovely Music (NY) em 1980 como *The Hub: Computer Network Music*. Durante este tempo, foi formado o grupo “*The League of Automatic Music Composers*”²², que além de Bischof, Behrman, contava com Tim Perkis, Scot Gresham-Lancaster, Mark Trayle e Phil Stone. Mas nosso foco será a formação no trio formado por Horton, Bischof e Perkis.

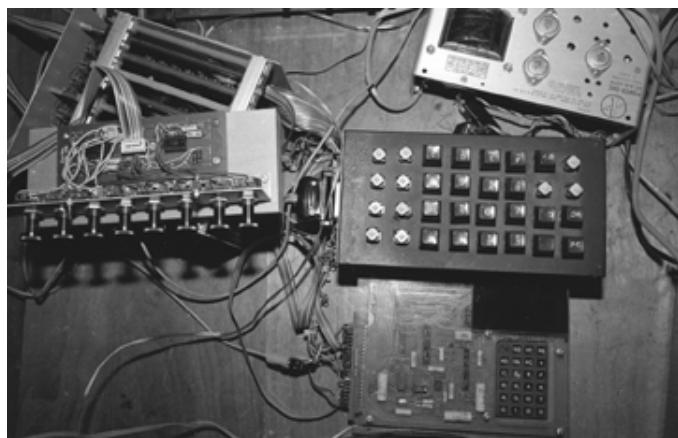


Figura 10 – Sistema de música computacional de John Bischof *circa* 1980. Foto: Eva Shoshanny²³. **Fonte:** Brown e Bischof (2002).

É interessante uma descrição de uma performance durante 1979. Propomos aqui realizar um paralelo com *happenings* (acontecimentos), manifestações artísticas já consolidadas para a época:

Na primavera de 1979, montamos uma série quinzenal regular de apresentações informais sob os auspícios da *Bay Center for the Performing Arts*. Todos outros domingos à tarde passávamos algumas horas configurando nossa rede de KIMs na sala *Finnish Hall*, na Berkeley, e deixávamos a rede tocando, com retoques aqui e ali, por uma ou duas horas. Os membros da audiência poderiam ir e vir como quisessem, fazer perguntas, ou simplesmente sentar e ouvir. Este foi um evento comunitário de tipos como outros compositores aparecendo, tocando ou compartilhando circuitos eletrônicos que tinham projetado e construído. Um interesse na

¹⁹ Tradução nossa de “the occasional tones of my [Bischof] machine caused Jim’s machine to transpose its melodic activity according to my “key” note.”

²⁰ Gravação muito longa para um *demo* e insuficiente para um disco de vinil da época.

²¹ Disponível em <<http://www.chickenonaunicycle.com/Berkeley%20Art.htm>>

²² Segundo Brown e Bischof (2002, 6ºparágrafo), o nome é uma referência ao grupo “The League of Composers” formado por Aaron Copland nos anos 20.

²³ Tradução de John Bischoff’s KIM-1 computer music system *circa* 1980 photo: Eva Shoshany

construção de instrumentos eletrônicos de todos os tipos parecia estar "no ar". Os eventos da sala *Finn Hall* foram feitos para uma cena com paisagens sonoras geradas por computador misturado com os sons de grupos de dança folclórica ensaiando no andar de cima e as reuniões ocasionais do Partido Comunista na sala de trás do edifício velho venerável. A série durou cerca de 5 meses que eu me lembre.(BROWN; BISCHOF, 2002, online)²⁴

Em 1980, Gold e Behrman abandonam o grupo, sendo que Tim Perkis se junta. Este foi período em que o grupo solidifica suas atividades na região da Baía de São Francisco. É interessante notar que uma metodologia modular começa a ser formalizada para permitir maior flexibilidade entre os sistemas de Horton, Bischof e Perkis. Isto é, ao invés de soldar componentes eletrônicos aos controladores, os membros conectavam os microcontroladores através de portas – o que para a época era arriscado ao ponto de queimar componentes. Com as conexões feitas, tocavam até três horas, tempo em que ouviam e ajustavam (*tuning*) os sistemas em interação²⁵(BROWN; BISCHOF, 2002, 7º parágrafo). Outro evento de importância é a associação do grupo com a banda *Rotary Club*, formada por alunos recém-formados da *Mills College*: Sam Ashley, Kenneth Atchley, Ben Azarm, Barbara Golden, Jay Cloidt e Brian Reinbolt. O grupo “baseava seu estilo de performance em torno de uma caixa de comutação projetada por Brian Reinbolt”²⁶(BROWN; BISCHOF, 2002, 8º parágrafo). Em 1983 o grupo reduziu suas atividades, época em que Horton contraiu artrite degenerativa.

Seria possível discutir a elaboração de uma “rede de composições”. No entanto, Brown e Bischof (2002, 11º parágrafo) comentam que estas não eram composições específicas, mas sim concertos inteiros: “ocasiões públicas para escuta compartilhada”²⁸. Este conceito pode ser melhor compreendido através de uma descrição do processo criativo da banda, que lidavam com um sistema limitado, de “baixa velocidade 1 Mhz e poucos dados (8 bits)”²⁹ com uma ênfase do grupo em explorar uma luteria composicional ³⁰ acompanhada de performance ao vivo; ou “A ênfase estava na exploração da tecnologia em

²⁴ Tradução nossa de: *In the spring of 1979, we set up a regular biweekly series of informal presentations under the auspices of the East Bay Center for the Performing Arts. Every other Sunday afternoon we spent a few hours setting up our network of KIMs at the Finnish Hall in Berkeley and let the network play, with tinkering here and there, for an hour or two. Audience members could come and go as they wished, ask questions, or just sit and listen. This was a community event of sorts as other composers would show up and play or share electronic circuits they had designed and built. An interest in electronic instrument building of all kinds seemed to be "in the air." The Finn Hall events made for quite a scene as computer-generated sonic landscapes mixed with the sounds of folk dancing troupes rehearsing upstairs and the occasional Communist Party meeting in the back room of the venerable old building. The series lasted about 5 months as I remember.*

²⁵ Disponível em <<https://www.youtube.com/watch?v=HW0qax8M68A>>

²⁶ Tradução nossa de “based their performance style around an automatic switching box designed by member Brian Reinbolt.”

²⁷ Tradução de *Tim Perkis' homebuilt computer-driven sound synthesis circuitry used in early 1980s. photo: Eva Shoshany.*

²⁸ Tradução nossa de “public occasions for shared listening.”

²⁹ Tradução nossa de “slow speed (1 Mhz) and data width (8 bits)”

³⁰ Cf. IAZZETTA; SOARES, 2009, 2015-03-13.

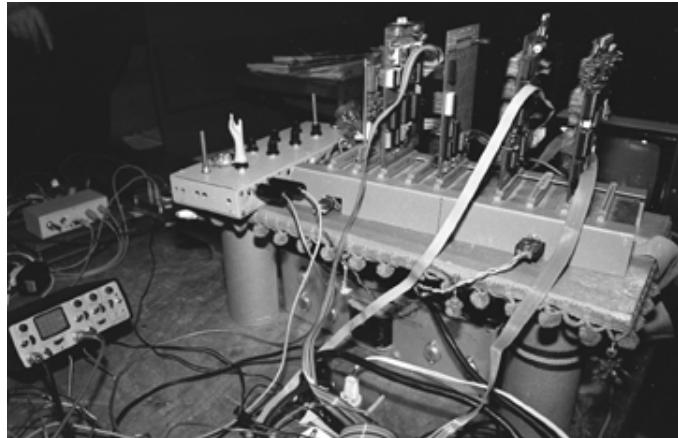


Figura 11 – Circuito do computador caseiro dedicado à síntese sonora de Tim Perkis, usado no começo dos anos 1980. Foto: Eva Shoshany²⁷. Fonte: Brown e Bischof (2002)

mãos que poderia ser adquirida pessoalmente ou construída a partir do zero, em vez do desejo incessante de melhores ferramentas.”³¹(BROWN; BISCHOF, 2002, 22º parágrafo):

“Os membros da liga geralmente adaptavam composições solo para usar dentro da banda. Estes solos eram desenvolvidos independentemente por cada compositor, e eram tipicamente baseados em esquemas de algoritmos de um tipo ou outro. Existiam características de improvisação diferentes para muitas delas, como bem as músicas eram diferentes em detalhes. Teorias matemáticas, sistemas de afinação experimentais, algoritmos de inteligência artificial, projetos de instrumentos de improvisação, e performance interativa eram algumas das áreas exploradas nestes trabalhos (...) Os solos tocavam simultaneamente no cenário de grupo, se tornando “sub”-composições que interagiam, cada uma enviando e recebendo dados pertinentes para o funcionamento musical. (BROWN; BISCHOF, 2002, 12º parágrafo).”³²

2.2.2 The Hub

O grupo *The Hub* era constituído, inicialmente pelo duo Bischoff e Perkis, após a saída de Horton. É interessante notar que um processo colaborativo entre os membros, e entre o duo formado por Chris Brown e Mark Trayle, em 1986, foi estimulado através de uma série de concertos, em galerias e espaços musicais comunitários, resultando em um festival , *THE NETWORK MUSE - Automatic Music Band Festival*. Neste festival

³¹ Tradução nossa de “The emphasis was on exploration of the technology at hand—technology that could be personally acquired or built from scratch—rather than the endless wish for better tools.”

³² Tradução nossa de “League members generally adapted solo compositions for use within the band. These solos were developed independently by each composer and were typically based on algorithmic schemes of one kind or another. There was a distinctly improvisational character to many of these as the music was always different in its detail. Mathematical theories of melody, experimental tuning systems, artificial intelligence algorithms, improvisational instrument design, and interactive performance were a few of the areas explored in these solo works. (...) The solos, played simultaneously in the group setting, became interacting “sub-compositions, each sending and receiving data pertinent to its musical functioning.”

outros grupos também realizaram suas apresentações, sendo um duo formado por Scott Gresham-Lancaster/Richard Zvonar e um trio formado por Phil Burk/Larry Polansky/Phil Stone.

Bischoff pontua que o nome da banda era uma maneira simbólica de caracterizar o sistema musical centralizado, “(…) um pequeno microcontrolador como caixa de correio, para postar dados usados no controle de seus sistemas individuais, que eram então acessados por outro intérprete, para usar de qualquer maneira e em qualquer tempo que escolher.”³³. O computador centralizado original, *Hub*, era um dos microcontroladores KIM-1 utilizados na época do *The League*. O sistema, de certa forma, é uma sensibilidade computacional, no âmbito musical, do modelo da máquina de Turing, permitindo a performance de até quatro executantes simultâneos:

“*The Hub* originalmente surgiu como uma maneira de limpar uma bagunça. (...) Toda vez que nós ensaiamos, um conjunto complicado de conexões *ad-hoc* entre computadores tinham de ser feitas. Isso criou um sistema com um comportamento rico e variado, mas sujeito a falhas, e trazer outros jogadores ficava difícil. Mais tarde, procuramos uma maneira de abrir o processo, para torná-lo mais fácil para os outros músicos tocarem no contexto de rede. O objetivo era criar uma nova maneira para pessoas fazerem música juntas. A solução bateu no ponto da facilidade de uso, e fornecimento de uma interface de usuário padrão, de modo que os jogadores poderiam conectar praticamente qualquer tipo de computador. *The Hub* é um pequeno computador dedicado a passar mensagens entre os jogadores. Ele serve como uma memória comum, mantendo informações sobre a atividade de cada jogador que seja acessível para os computadores de outros jogadores (BROWN; BISCHOF, 2002, seção 2.1). ”³⁴

Em 1987, Nick Collins e Phil Niblock realizam uma curadoria para realizar uma performance telemática entre a *Experimental Media* e *The Clocktower* em Nova York. Para isso, chamam os membros do *The Hub*, que estimulam uma performance de um grupo único, dividido em dois trios, formado por John Bischoff, Tim Perkis, Mark Trayle, Chris Brown, Scot Gresham-Lancaster, e Phil Stone. Estes dois trios se comunicam, entre os dois espaços, através de dois novos *Hub* intercomunicáveis. Cada *Hub* era um sistema centralizado para cada trio. As peças tocadas, “Simple Degradation”, “Borrowing and Stealing” e “Vague Notions” permitiram a concepção de uma performance de um “sexteto

³³ Tradução nossa de “(…) a small microcomputer as a mailbox to post data used in controlling their individual music systems, which was then accessible to the other player to use in whatever way and at whatever time he chose.”

³⁴ Tradução nossa de “*The Hub* originally came about as a way to clean up a mess. John Bischoff, (...) Every time we rehearsed, a complicated set of ad-hoc connections between computers had to be made. This made for a system with rich and varied behavior, but it was prone to failure, and bringing in other players was difficult. Later we sought a way to open the process up, to make it easier for other musicians to play in the network situation. The goal was to create a new way for people to make music together. The solution hit upon had to be easy to use and provide a standard user interface, so that players could connect almost any type of computer. The *Hub* is a small computer dedicated to passing messages between players. It serves as a common memory, keeping information about each player’s activity that is accessible to other players’ computers.”

acusticamente divorciado mas informacionalmente ligado.”³⁵ (BROWN; BISCHOF, 2002, seção 2.2).

2.2.3 Ron Kuivila

McLean e Wiggins (2009) comentam a performance *Water Surfaces*, realizada na edição de 1985 da STEIM³⁶, em Amsterdã, como significativa para a concepção de uma improvisação de códigos (excluindo a tecnologia de projeção visual). A performance chamou a atenção, e foi incluída na primeira faixa do disco “*TOPLAP001 - A prehistory of live coding*”, como uma reconstrução da peça, 2007³⁷; uma nota sobre a performance descreve o seguinte: “Esta obra usou programação FORTH ao vivo; Curtis Roads (1986) testemunhou e relatou a performance de Ron Kuivila feita na STEIM em Amsterdã, em 1985; a performance original termina com a quebra do sistema...”³⁸

Ronald Kuivila programou um computador Apple II no palco para cirar sons densos, rodopiantes e métricos, disposto em camadas e dobravam sobre si. Considerando o equipamento usado, os sons eram surpreendentemente grandes em escala. Kuivila teve problemas em controlar a peça devido q problemas sistêmicos. Ele finalmente entrou em dificuldades técnicas e finalizou a performance(ROADS, 1986, p. 47)³⁹.

Ge Wang (2005), em uma comunicação pessoal com Curtis Roads, cita a seguinte declaração: “Eu vi o software FORTH de Ron Kuivila quebrar e queimar no palco em Amsterdã em 1985, mas antes disso, não fez uma música muito interessante. A performance consistiu de digitação”⁴⁰

Nenhuma fonte sonora foi encontrada disponível online.

2.3 LAPTOP

Por último, vamos discutir um recorte do documento-manifesto “*Live Algorithm Programming and Temporary Organization for its Promotion*”, de Ward et al.; McLean e Wiggins. Nossa discussão visa apontar espaços conceituais mais diretos da improvisação de códigos. Isto é, uma identidade cultural da organização TOPLAP (ver seção 2.4, p. 40).

³⁵ Tradução nossa de “*acoustically divorced, but informationally joined sextet.*”

³⁶ *SStudio for Electro-Instrumental Music*, disponível em <<http://steim.org/about/>>.

³⁷ Disponível em <http://toplap.org/wiki/TOPLAP_CDs>.

³⁸ Tradução nossa de “*This work used live FORTH programming; Curtis Roads witnessed and reported a performance by Ron Kuivila at STEIM in 1985; the original performance apparently closed with a system crash...*”

³⁹ Tradução de *Ronald Kuivila programmed an Apple II computeronstage to create dense, whirling, metric sounds that layered in and folded over each other. Considering the equipment used, the sounds were often surprisingly gigantic in scale. Kuivila had trouble controlling the piece due to system problems. He finally gave in to technical difficulties and ended the performance*

⁴⁰ Tradução nossa de “*I saw Ron Kuivila’s Forth software crash and burn onstage in Amsterdam in 1985, but not before making some quite interesting music. The performance consisted of typing.*”

Dentre este manifesto, selecionamos dois pontos: *i*) um comentário sobre a ideologia de projeção de telas (ver [subseção 2.4.1.1](#), p. 42) e; *ii*) “Show us your screens”, como uma revisão de regras práticas do *live coding* (ver [subseção 2.4.1](#), p. 41).

“*Live Algorithm Programming and Temporary Organization for its Promotion*” ([WARD et al., 2004](#); [BLACKWELL; COLLINS, 2005](#)) é um primeiro documento-manifesto sobre o *live coding* como modalidade artística, e de suas regras práticas. O seu acrônimo LAPTOP representa o principal equipamento técnico utilizado. Este manifesto expõe o ambiente de performance característico do *algorave* e um suporte ideológico para o *Code DJing*. Ritos técnicos do improvisador, como por exemplo, a projeção do código, são justificados através do discurso de transparência e provável colaboração entre intérprete e público:

O *Livecoding* permite a exploração de espaços algorítmicos abstratos como uma improvisação intelectual. Como uma atividade intelectual, pode ser colaborativa. Codificação e teorização podem ser atos sociais. Se existe um público, revelar, provocar e desafiar eles com uma matemática complexa se faz com a esperança de que sigam, ou até mesmo participem da expedição. Estas questões são, de certa forma, independentes do computador, quando a valorização e exploração do algoritmo é o que importa. Outro experimento mental pode ser encarado com um DJ ao vivo codificando e escrevendo uma lista de instruções para o seu *set* (feito com o iTunes, mas aparelhos reais funcionam igualmente bem). Eles passam ao HDJ [*Headphone Disk Jockey*] de acordo com este conjunto de instruções, mas no meio do caminho modificam a lista. A lista está em um retroprojetor para que o público possa acompanhar a tomada de decisão e tentar obter um melhor acesso ao processo de pensamento do compositor. ([WARD et al., 2004](#), p. 245)⁴¹

Adiante podemos ver outros dois conceitos aglutinados: a Música de Processos, e a Música Generativa:

Contudo, alguns músicos exploram suas idéias como processos de *software*, muitas vezes ao ponto que o *software* se torna a essência da música. Neste ponto, os músicos podem ser pensados como programadores explorando seu código manifestado como som. Isso não reduz seu papel principal como um músico, mas complementa, com a perspectiva única na composição de sua música. **Termos como “música generativa” e “música de processos” tem sido inventados e apropriados para descrever esta nova perspectiva de composição.** Muita coisa é feita

⁴¹ Tradução nossa de: *Live coding allows the exploration of abstract algorithm spaces as an intellectual improvisation. As an intellectual activity it may be collaborative. Coding and theorising may be a social act. If there is an audience, revealing, provoking and challenging them with the bare bone mathematics can hopefully make them follow along or even take part in the expedition. These issues are in some ways independent of the computer, when it is the appreciation and exploration of algorithm that matters. Another thought experiment can be envisaged in which a live coding DJ writes down an instruction list for their set (performed with iTunes, but real decks would do equally well). They proceed to HDJ according to this instruction set, but halfway through they modify the list. The list is on an overhead projector so the audience can follow the decision making and try to get better access to the composer’s thought process.*

das supostas propriedades da chamada “música gerativa” que separa o compositor do resultado do seu trabalho. Brian Eno compara o fazer da música gerativa com o semear de sementes que são deixadas para crescer, e sugere abrir mão do controle dos nossos processos, deixando eles “brincarem ao vento”.⁴²

Se por um lado, a Música como um Processo Gradual⁴³ e a Música Generativa são referenciais possíveis na improvisação de códigos, essa não é a questão inicial. A ligação conceitual do *live coding* com a Música de Processos e a Música Generativa é relativa ao uso de algoritmos, mas não ao resultado sonoro como processo de escuta. Por exemplo, uma abordagem sobre a Música de Processos é apresentada por Mailman (2013, p. 128), e descreve a Música Minimalista de Processos como uma Música de Algoritmos Simples, um processo determinístico que age sobre focos de quadros temporais. Já a “Música Generativa é sensitiva às circunstâncias, isso quer dizer que irá reagir diferentemente dependendo das suas condições iniciais, onde ocorre e assim por diante.”⁴⁴(ENO, 1996). McLean (2011, p. 130) problematiza o processo na improvisação de códigos da seguinte forma:

“Na codificação ao vivo a performance é o processo de desenvolvimento de *software*, em vez de seu resultado. O trabalho não é gerado por um programa acabado, mas através de sua jornada de desenvolvimento do nada para um algoritmo complexo, gerando mudanças contínuas da forma musical ou visual ao longo do caminho. Isto contrasta com a arte generativa popularizada pela música geradora de Brian Eno (1996). (...) O resultado segue mais ou menos o mesmo estilo, com apenas algumas permutações, dando uma idéia das qualidades da peça. Isto é bem ilustrado pelo nosso estudo de caso de um artista-programador, que executa seu programa poucas vezes não para produzir novas obras, mas para obter diferentes perspectivas sobre o mesmo trabalho.”⁴⁵

⁴² Cf. WARD et al., op. cit., p. 245-246. Tradução nossa de *Indeed, some musicians explore their ideas as software processes, often to the point that a software becomes the essence of the music. At this point, the musicians may also be thought of as programmers exploring their code manifested as sound. This does not reduce their primary role as a musician, but complements it, with unique perspective on the composition of their music. Terms such as “generative music” and “processor music” have been invented and appropriated to describe this new perspective on composition. Much is made of the alleged properties of so called “generative music” that separate the composer from the resulting work. Brian Eno likens making generative music to sowing seeds that are left to grow, and suggests we give up control to our processes, leaving them to “play in the wind”.*

⁴³ Cf. REICH, 1968

⁴⁴ Tradução nossa de “*Generative music is sensitive to circumstances, that is to say it will react differently depending on its initial condition, on where it’s happening and so on.*”

⁴⁵ Tradução nossa de “*In live coding the performance is the process of software development, rather than its outcome. The work is not generated by a finished program, but through its journey of development from nothing to a complex algorithm, generating continuously changing musical or visual form along the way. This is by contrast to generative art popularised by the generative music of Brian Eno (1996) (...) Output more or less follows the same style, with only a few permutations giving an idea of the qualities of the piece. This is well illustrated by our case study of an artist-programmer, who ran their program a few times not to produce new works, but to get different perspectives on the same work.*”

2.4 TOPLAP

Uma permutação na ordem das letras do acrônimo LAPTOP dá origem ao acrônimo TOPLAP. Ward et al. (2004, p. 246) e Ramsay (2010) apontam que este acrônimo dinâmico; isto quer dizer que as primeira, terceira e quinta letras possuem diversos significados (ver Figura 12):

“A organização TOPLAP (www.toplap.org), cuja sigla possui diversas interpretações, uma sendo *Organização Temporária para a Proliferação da Programação de Algoritmos Ao Vivo*, foi criada para promover e explorar o *live coding*. TOPLAP nasceu em um bar enfumaçada em Hamburgo à uma da manhã em 15 de Fevereiro de 2004.”⁴⁶

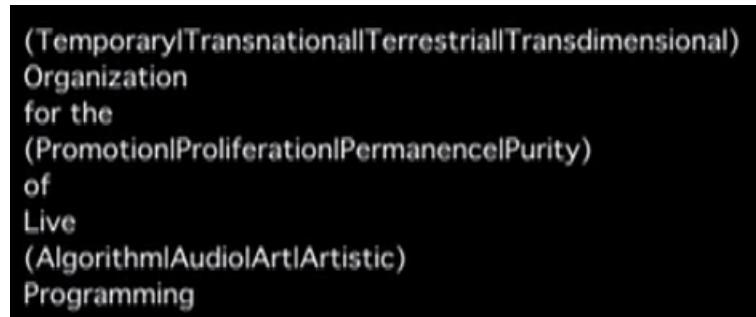


Figura 12 – Definição do significado de TOPLAP. **Fonte:** [Ramsay \(2010\)](#).

O símbolo “|” é uma representação gráfica do operador lógico *OR* (OU), bastante utilizado em algoritmos condicionais. Isto é, *Temporary | Transnational | Terrestrial | Transdimensional* significa que as letras ímpares “T”, e “P” e “A”, podem significar um ou outro termo indicado pelo algoritmo.

Este comportamento, de permitir a ordem das letras é praticado por Nick Collins (1975-); a permutação de suas letras é utilizada pelo pesquisador para gerar pseudônimos como Click Nilson, ou Sick Lincoln. Isso transparece uma técnica de uso frequente na improvisação de códigos, provavelmente pela facilidade de sua implementação computacional em amb. Por exemplo, o *SuperCollider* oferece um método chamado *scramble*, que embaralha a ordem de um conjunto (de caracteres). Mais especificamente, a permutação de letras transparece uma reorganização gramatical, mas que também reflete em técnicas de reorganização algorítmica da gramática musical.

⁴⁶ Tradução nossa de “The organisation TOPLAP (www.toplap.org), whose acronym has a number of interpretations, one being the Temporary Organisation for the Proliferation for Live Algorithm Programming, has been set up to promote and explore live coding. TOPLAP was born in a smoky Hamburg bar at 1am on Sunday 15th February 2004”

2.4.1 Show us your screens

Além das performances inaugurais nos festivais Europeus, o manifesto Lubeck04, “iniciado em um ônibus de trânsito Ryanair⁴⁷, em Hamburgo, para o aeroporto Lübeck(WARD et al., 2004, p. 247)”⁴⁸, mais conhecido como “*Show us your screens*”, prescreve algumas regras práticas do *live coding*.

Exigimos:

- Acesso à mente do intérprete, para todo o instrumento humano.
- Obscurantismo é perigoso. Mostre-nos suas telas.
- Programas são instrumentos que podem modificar eles mesmos.
- O programa será transcendido - Língua Artificial é o caminho.
- O código deve ser visto assim como ouvido, códigos subjacentes visualizados bem como seu resultado visual.
- Codificação ao vivo não é sobre ferramentas. Algoritmos são pensamentos. Motosserras são ferramentas. É por isso que às vezes algoritmos são mais difíceis de perceber do que motosserras.

Reconhecemos contínuos de interação e profundidade, mas preferimos:

- Introspecção dos algoritmos.
- A externalização hábil de algoritmo como exibição expressiva/impresiva de destreza mental.
- Sem *backup* (minidisc, DVD, safety net computer).

Nós reconhecemos que:

- Não é necessário para uma audiência leiga compreender o código para apreciar, tal como não é necessário saber como tocar guitarra para apreciar uma performance de guitarra.
- Codificação ao vivo pode ser acompanhada por uma impressionante exibição de destreza manual e a glorificação da interface de digitação.
- Performance envolve contínuos de interação, cobrindo talvez o âmbito dos controles, no que diz respeito ao parâmetro espaço da obra de arte, ou conteúdo gestual, particularmente direcionado para o detalhe expressivo. Enquanto desvios na tradicional taxa de reflexos táteis da expressividade, na música instrumental, não são aproximadas no código, por que repetir o passado? Sem dúvida, a escrita de código e expressão do pensamento irá desenvolver suas próprias nuances e costumes.⁴⁹

⁴⁷ Disponível em <<https://www.ryanair.com/pt/pt/>>

⁴⁸ Tradução nossa de “*begun on a Ryanair transit bus from Hamburg to Lubeck airport*”

⁴⁹ WARD et al., 2004, loc. cit.. Tradução nossa de: *We demand:* • Give us access to the performer's mind, to the whole human instrument. • Obscurantism is dangerous. Show us your screens. • Programs are instruments that can change themselves. • The program is to be transcended - Artificial language is the way. • Code should be seen as well as heard, underlying algorithms viewed as well as their visual outcome. • Live coding is not about tools. Algorithms are thoughts. Chainsaws are tools. That's why algorithms are sometimes harder to notice than chainsaws. . We recognise continuums of interaction and profundity, but prefer: • Insight into algorithms • The skillful extemporisation of algorithm as an expressive/impressive display of mental dexterity • No backup (minidisc, DVD, safety net computer). We acknowledge that: • It is not necessary for a lay audience to understand the code to appreciate it, much as it is not necessary to know how to play guitar in order to appreciate watching a guitar performance. • Live coding may be accompanied by an impressive display of manual dexterity and the glorification of the typing interface. • Performance involves continuums of interaction, covering perhaps the scope of controls with respect to the parameter space of the artwork, or gestural content,

Escolhemos dois pontos de interesse ; as frase “Obscurantismo é perigoso. Mostre-nos suas telas” e “Algoritmos são pensamentos, motosserras são ferramentas”, foi muito discutida no processo de qualificação desta tese. O primeiro questiona: é realmente necessário a projeção dos códigos para a questão da performance, do ponto de vista musical/cênico?

2.4.1.1 Obscurantismo é perigoso, mostre-nos suas telas

O manifesto acima surgiu, entre outros motivos, como uma resposta ao artigo “*Using Contemporary Technology in Live Performance; the Dilemma of the Performer*” (SCHLOSS, 2003). A crítica principal de Ward et al. refere-se ao sétimo dos questionamentos sugeridos para uma performance de improvisação ao vivo com computadores. Isto é, em um contexto de embate acadêmico, o desafio colocado por Schloss (2003, p. 241) foi um estímulo considerável para emancipação da improvisação de códigos. É curioso notar que o problema e a intenção de Schloss eram opostas ao que foi proposto por Ward et al.:

“Para reiterar, agora que nós temos computadores rápidos o suficiente para execução ao vivo, nós temos novas possibilidades, e um novo problema. Do começo da evidência arqueológica da música até agora, música era tocada acusticamente, e sempre foi fisicamente evidente como o som era produzido; ali existia uma relação de proximidade entre gesto e resultado. Agora nós não temos mais que seguir as leis da física (ultimamente temos, mas não nos termos do que o observador vê), uma vez que nós temos completo poder do computador como intérprete e intermediário entre nosso corpo físico e o som produzido. **Por esta causa, a ligação entre gesto e resultado foi completamente perdido, se é que existe ligação. Isto significa que nós podemos ir além da relação de causa-e-efeito entre executante e instrumento que faz a mágica.** Mágica é bom; muita mágica é fatal.”⁵⁰

A crítica de Schloss (2003, p. 239): “considerar a visão do observador sobre os modos de performance das interações físicas e mapeamentos de gestos em som, para fazer uma performance convincente e efetiva”⁵¹ era especificamente direcionada aos compositores que improvisam música computacional no palco com foco apenas no aspecto sonoro ou tecnológico. Sua questão tange a ausência de gestos referenciais, esforço físico, no caso

particularly directness of expressive detail. Whilst the traditional haptic rate timing deviations of expressivity in instrumental music are not approximated in code, why repeat the past? No doubt the writing of code and expression of thought will develop its own nuances and customs.

⁵⁰ Tradução nossa de “To reiterate, now that we have fast enough computers to perform live, we have new possibilities, and a new problem. From the beginning of the archeological evidence of music until now, music was played acoustically, and thus it was always physically evident how the sound was produced; there was a nearly one-to-one relationship between gesture and result. Now we don't have to follow the laws of physics anymore (ultimately we do, but not in terms of what the observer observes), because we have the full power of computers as interpreter and intermediary between our physical body and the sound production. Because of this, the link between gesture and result can be completely lost, if indeed there is a link at all. This means that we can go so far beyond the usual cause-and-effect relationship between performer and instrument that it seems like magic. Magic is great; too much magic is fatal”

⁵¹ Tradução nossa de “It's now necessary, (...) ;to consider the observer's view of the performer's modes of physical interactions and mappings from gesture to sound, in order to make the performance convincing and effective.”

de performances com dispositivos extendidos, o problema do movimento exagerado, e a expectativa cênica na performance musical:

- “1. Causa-e-efeito é importante, pelo menos para o observador/audiência em uma sala de concerto.
- 2. Corolário: Mágica na performance é bom. Muita mágica é fatal! (chato).
- 3. Um componente visual é essencial para a audiência, tal como existe um aparato visual de entrada para parâmetros e gestos.
- 4. Sutileza é importante. Grandes gestos são facilmente visíveis de longe, o que é bom, mas eles são movimentos de desenho animado se comparados à execução de um instrumento musical.
- 5. Esforço é importante. Neste sentido, nós estamos em desvantagem de desempenho na performance musical com o computador.
- 6. Improvisação no palco é bom, mas “mimar” o aparato no palco não é improvisação, é edição. É provavelmente mais apropriado fazer isso no estúdio antes do concerto, ou se durante o concerto, com o console no meio ou atrás da sala de concerto.
- 7. Pessoas que representam devem representar. Um concerto de música de computador não é uma excusa/oportunidade para um programador(a) se sentar no palco. Sua presença melhora ou impede o desempenho da apresentação?”⁵²

Duas opiniões divergentes resolvem seus problemas de maneiras divergentes sem considerarem como uma pode auxiliar a outra. No item 3, é apontado uma questão: para a audiência, e não para o improvisador, o componente visual é essencial (substantificação provável da prática). Ward et al. vão no caminho oposto ao de Schloss, e exageram este item, ao projetar códigos. Mas para Schloss, realizar isso é mimar o aparato (e o público), e tornar a apresentação pedante. É curioso notar que Schloss faz um apontamento importante, no item 5, sobre a ausência de esforço. Não que ela seja premissa para o resultado sonoro, mas para o público, e Schloss trata exatamente deste ponto, ela é importante na performance musical (item 1). Mas a crítica mais ácida é o item 7, cujo pensamento não difere de uma lógica produtivista: as atividades de artista e de programador devem ser bem definidas, e separadas. Para Schloss, são duas atividades que não se complementam. Para McLean, são interdisciplinares.

⁵² Tradução nossa de “1. *Cause-and-effect is important, at least for the observer/audience in a live concert venue.* 2. *Corollary: Magic in a performance is good. Too much magic is fatal! (Boring).* 3. *A visual component is essential to the audience, such that there is a visual display of input parameters/gestures. The gestural aspect of the sound becomes easier to experience.* 4. *Subtlety is important. Huge gestures are easily visible from far away, which is nice, but they are cartoon- movements compared to playing a musical instrument.* 5. *Effort is important. In this regard, we are handicapped in computer music performance.* 6. *Improvisation on stage is good, but “baby-sitting” the apparatus on stage is not improvisation, it is editing. It is probably more appropriate to do this either in the studio before the concert, or if at the concert, then at the console in the middle or back of the concert hall.* 7. *People who perform should be performers. A computer music concert is not an excuse/opportunity for a computer programmer to finally be on stage. Does his/her presence enhance the performance or hinder it?*”

2.4.2 Algorithms are Thoughts, Chainsaws are Tools

“Algorithms are Thoughts, Chainsaws are Tools” é o nome dado ao vídeo de Stephen Ramsay (2010), publicado no Vimeo, em 27 de fevereiro de 2010, como um *Coffee-Table Movie*. É uma análise pessoal da performance de *Strange Places* de Andrew Sorensen. O nome do vídeo é derivado de uma das regras práticas apresentadas na subseção 2.4.1, p. 41; mais especificamente, o sexto item.

O algoritmo como pensamento é um espaço conceitual abstrato (ver Capítulo 3, p. 47); pode conter qualquer fundamento teórico pertinente para uma improvisação específica. O dispositivo usado (motoserra, máquina de tecelagem ou o computador) é um meio pelo qual uma estratégia transversal (ver subseção 3.1.2, p. 51) toma sua forma sonora. É interessante aqui notar que este vídeo contém uma descrição e comentários que podem elucidar a frase-alvo sob o prisma da partitura musical. Abaixo realizei uma compilação de fragmentos de alguns dos comentários que considerei pertinentes. Ramsey apresenta a seguinte descrição do vídeo:

Um curta sobre *livecoding* apresentado como parte do Grupo de Estudos de Crítica de Códigos, em 2010, por Stephen Ramsay. Apresenta uma leitura ao vivo [*live reading*] de uma performance do compositor Andrew Sorensen. Também fala sobre J.D. Salinger, the Rockets, tocando instrumentos, Lisp, do clima em Brisbane e tímpanos ⁵³.

Sem entrar em méritos críticos do registro, limitamo-nos a descrever como um VLog, uma variante no formato audiovisual de *weblogs*⁵⁴. Se caracteriza por ser um vídeo de curta duração, com opiniões pessoais de quem fez, geralmente no quarto da pessoa, com *headsets* (microfone+headphones). A prática de inserir comentários em um é bastante útil para levantar outras opiniões. Realizamos a tradução de alguns comentários. Não são nossas opiniões, mas podem oferecer ao leitor uma abrangência sobre o que pensa um público entusiasta.

Amanda French nega a utilização do termo *partitura* para explicitar diferenças no uso da programação-partitura, em uma performance de improvisação com o computador, para uma performance não-improvisada com partitura.

A noção de partitura não se aplica aqui, é como não fosse possível aplicá-lo ao músico de *jazz* ou tocador de *bluegrass*. (...). Levanta a questão, para mim, se, em uma sessão de *livecoding* *feita*, constite simplesmente no ato de digitar em um programa existente, seria tão convincente – eu acho que isso pode definitivamente ter pontos de interesse. Ou qual

⁵³ RAMSAY, 2010, loc. cit. Tradução de *A short film on livecoding presented as part of the Critical Code Studies Working Group, March 2010, by Stephen Ramsay. Presents a "live reading" of a performance by composer Andrew Sorensen. It also talks about J. D. Salinger, the Rockettes, playing musical instruments, Lisp, the weather in Brisbane, and kettle drums..*

⁵⁴ Cf. BAKER, .

seria o análogo do *livecoding* para uma performance não-improvisada de música?⁵⁵

Um segundo comentário de Matt King, coloca a pergunta de Amanda em outra perspectiva:

O que torna o *livecoding* diferente, e pode a performance de música tradicional imitar isso? Para responder esta questão, parece importante notar que as formas nas quais a música improvisada muitas vezes apela para alguma noção de autenticidade ou gênio. Enquanto o *livecoding* ele mesmo à noção de virtuosismo de código, “autenticidade” parece fora de lugar aqui. Se música improvisada sugere expressão, o *livecoding* sugere um conjunto de restrições na expressão, descrevendo os parâmetros através dos quais a máquina [midi] ganha expressão⁵⁶

Michel Pasin defende que o ato de improvisação musical requer conhecimentos técnicos prévios, mas não necessariamente correlacionados ao conhecimento do que é uma partitura: “Em geral, é somente dominando um instrumento que você pode esquecer sobre a técnica e concentrar em ‘dizer’ coisas com o instrumento.”⁵⁷. Este caso é bastante específico de performances com linguagens de baixo e alto-nível. Porém seria possível objetar que a prática de construção de linguagens artificiais, no topo de outras linguagens artificiais, possibilita um praticante não-familiarizado com a programação elaborar rotinas computacionais. Porém aí caímos em um problema: estaria o praticante realizando uma improvisação de códigos? ou melhor, isso importa, se o objetivo é a criação musical? Para delinear estas questões buscamos definir no próximo capítulo o que consideramos por objetivo de um agenciamento sonoro improvisado.

2.5 Discussão

Oferecemos um cenário proto-histórico do ponto de vista, na Itália com o compositor Pietro Grossi, nos EUA com Jim Horton, John Bischoff, Tim Perkis, e na Holanda com Ron Kuivila (residente nos EUA), que deram suporte ao pensamento promovido na Inglaterra

⁵⁵ RAMSAY, 2010, loc. cit. Tradução parcial de *The notion of "sheet music" doesn't apply here, as it wouldn't apply to a jazz musician or a bluegrass picker. Even the name of his environment, Impromptu, makes that point. Raises the question for me precisely of whether a livecoding session that *did* consist of simply typing in an existing program would be as compelling – I think it would definitely have its points of interest, actually. Or what would the livecoding analog be to a non-improvisational live performance of music?*

⁵⁶ Tradução nossa de (...) *What makes livecoding different, and can a traditional music performance mimic it? To answer this question, it seems important to note the ways in which improvised music often appeals to some notion of authenticity or genius. While livecoding might lend itself to some notion of coding virtuosity, "authenticity" seems out of place here. If improvised music is expression, livecoding suggests a setting of constraints on expression, describing the parameters through which the machine (midi) gets expressed.*

⁵⁷ Tradução nossa de “In general, it is only by mastering an instrument that you can forget about the technique and concentrate on ‘saying’ things with the instrument.”

e Alemanha. Sugestões para essa proto-história foram colocados no Apêndice B. Com um hiato na década de noventa, não que sejam ausentes informações, mas nossa pesquisa focou mais em citar eventos de improvisações de códigos de um nicho específico, aqueles músicos interessados em uma luteria composicional⁵⁸, onde a atividade de *hackear* sistemas (conectar cabos, soldar componentes, programar códigos, etc.) se torna parte de uma sensibilidade computacional da atividade musical, próximo daquilo que Mumma e Tudor diziam como utilizar circuitos eletrônicos eles mesmos como atores musicais.

⁵⁸ Cf. IAZZETTA; SOARES, 2009, 2015-03-13

3 Metodologia de análise de uma Improvisação musical de códigos

O contexto deste capítulo discute uma taxonomia de tipos de criatividade (ver [seção 3.1](#), p. 47) para extender o assunto ao “Quadro Conceitual de Sistemas Criativos”¹ (ver [seção 3.2](#), p. 58). Este quadro conceitual realiza uma discretização de processos criativos contínuos, o que possibilita discutir um modelo de estruturação, e análise, de improvisações de códigos musicais (ver [subseção 3.2.1](#), p. 59). Através de diagramas (ver [seção 3.3](#), p. 62), buscamos selecionar uma parte deste quadro conceitual (ver [seção 3.4](#), p. 69), de forma que seja útil para a análise de um algoritmo musical no próximo capítulo.

3.1 Criatividade

Para definir *criatividadade* nos termos de uma improvisação de códigos, discutiremos a proposição do livro “The Creative Mind: myths and mechanisms” de Margaret Boden (2004) (ver [subseção 3.1.1](#), p. 47). Em seguida, apresentamos o conceito de uma *imagem mental*, idealmente formada antes do processo criativo, mas constantemente modificada durante a atividade que materializa o imaginário (ver [subseção 3.1.2](#), p. 51). Uma experiência mental será descrita como uma estratégia de ação, idealizada, de um improvisador-programador fictício (ver [subseção 3.1.3](#), p. 54).

3.1.1 Criatividade

É possível discutir quais valores a palavra *criatividadade* carrega. Boden (2004, p. 22) aponta: “Porque a criatividade, por definição, envolve não somente novidade, mas valor, e porque os valores são altamente variáveis, segue que muitos argumentos sobre criatividade estão enraizados em discordâncias sobre valores.”². Uma simples busca em um dicionário³ permite discutir alguns valores do termo. Um primeiro conceito, e o mais geral de todos, “Capacidade de criar, ou inventar”, ilumina um mecanismo *gerativo*, através de uma definição cíclica. Sobre esta ciclicidade, Chris Thornton (2007, p. 2) discute que Boden, ao investigar a criatividade do ponto de vista mecanicista, explicita um paradoxo desta definição geral:

¹ Tradução nossa de “Creative System Frameworks, ou CSF”

² Tradução nossa de “Because creativity by definition involves not only novelty but value, and because values are highly variable, it follows that many arguments about creativity are rooted in disagreements about value.”

³ Disponível em <<http://www.priberam.pt/dlpo/criatividade>>

“O ponto inicial de Boden para o desenvolvimento de sua explicação, é a observação de que o conceito de criatividade contém um paradoxo. Por definição, criatividade cria, i.e., produz alguma coisa nova. Mas se estamos comprometidos com uma abordagem mecanicista do mundo – nenhum milagre é permitido – iremos acreditar que tudo o que ocorre é, em princípio, previsível. Iremos acreditar também que qualquer coisa nova deve ser construída de componentes existentes. Isso implica que nada pode ser intrinsecamente novo.”⁴

A definição linguística do dicionário, “Capacidade que o falante de uma língua tem de criar novos enunciados sem que os tenha ouvido ou dito anteriormente.”, também possui este paradoxo. É possível, por exemplo, identificar um mecanismo de avaliação do que é impossível, e por consequência, criativo: “Para justificar a classificação de uma idéia criativa... alguém deve identificar os princípios gerativos com respeito ao que é impossível”⁵ (BODEN, 2004 apud Thornton, 2007, p. 51; p. 3). Mas isso não quer dizer que, do ponto de vista mecanicista, a impossibilidade surge do nada. Para Boden, o impossível deriva de um processo de especificação (*idem, ibidem*):

“A distinção é entre o sentido de criar um conceito que nunca foi criado antes [criatividade-P], e um conceito que nunca foi criado antes por um criador específico [criatividade-H]. Esta distinção será tangencialmente relevante para meu argumento aqui, mas antes de prosseguir, eu noto que esta não é uma simples escolha binária, mas ao invés, uma contextualização multidimensional: pode ser possível, por exemplo, para um comportamento criativo ser unicamente P-criativo em uma sociedade, mas H-criativo em outra; deste ponto de vista da segunda sociedade, apenas importam comportamentos H-criativos. (...) no trabalho de Boden, existe uma distinção entre criatividade exploradora e transformacional, que será relevante aqui, e então merece alguma explicação. **Boden concebe o processo de criatividade como uma identificação e/ou localização de novos objetos conceituais em um espaço conceitual.**”⁶

Para Thornton, na primeira edição de seu livro, Boden não explica como são os métodos de acesso aos espaços conceituais. No caso, os espaços conceituais investigados fazem

⁴ Tradução nossa de “*Boden’s starting point for the development of her account is the observation that the concept of creativity contains a paradox. By definition, creativity creates, i.e., it produces something new. But if we are committed to a mechanistic account of the world — no miracles allowed — we believe that everything that occurs is predictable in principle. We also believe that any new thing must be constructed from existing components. This implies that nothing can ever be intrinsically new.*”

⁵ Tradução nossa de “*To justify calling an idea creative... one must identify the generative principles with respect to which it is impossible.*”

⁶ Tradução nossa de “*The distinction is between the sense of creating a concept which has never been created before at all, and a concept which has never been created before by a specific creator. This distinction will be only tangentially relevant to my argument here, but before proceeding, I note that this is not a simple binary choice, but rather multi-dimensional, context-based one: it would be possible, for example, for a creative behaviour to be only P-creative in one society, but H-creative in another; from the point of view of the second society, only the H-creativity matters. (...) in Boden’s work, there is the distinction between exploratory and transformational creativity, which is directly relevant here, and so needs a little more explanation. Boden conceives the process of creativity as the identification and/or location of new conceptual objects in a conceptual space.*”

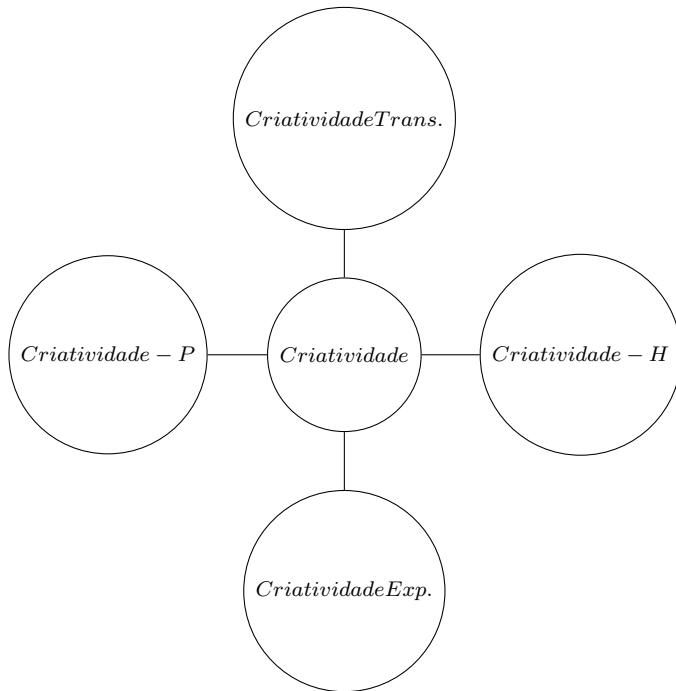


Figura 13 – Classificação da criatividade : 1) criatividade-psicológica/criatividade-histórica; 2) criatividade exploradora/criatividade transformacional. **Fonte:** autor com base em [Wiggins \(2006\)](#).

parte do imaginário do pensamento científico, e do comportamento criativo de crianças ([BODEN, 2004](#), p. 69–87). Com base nisso, uma taxonomia de tipos de criatividade é derivada (ver [Figura 13](#), p. 49). A primeira classificação divide a criatividade em criatividade-histórica (*H-creativity*), e criatividade-psicológica (*P-creativity*). A segunda classificação separa criatividade em criatividade-transformacional e criatividade-exploradora.

“De fato, na primeira edição [1990], ela não oferece uma explicação do número de diferentes tipos de criatividade que ela identificou. Parece que sua intenção era distinguir os dois tipos notados, espaços conceituais devem ter uma característica gerativa. E isso certamente é a interpretação comum. Ainda no sumário ‘em uma casca de noz’ de sua teoria, foi adicionado um prólogo à segunda edição ([BODEN, 2004](#)), e em [Boden \(1998\)](#), ela coloca que sua explicação distingue três principais formas de criatividade, sendo exploração, transformação e *combinação*. É somada à incerteza a observação que somente a definição forte [generalizadora] da definição possui o poder de resolver o paradoxo da criatividade”⁷

⁷ Tradução nossa de “*In fact, in the first edition, she offers no final count of the number of different types of creativity she has identified. It seems to be her intention to distinguish the two types noted, conceptual space must be generative in character, and this is certainly a common interpretation. Yet in the ‘nutshell’ summary of her theory, added as a prologue to the second edition (Boden, 2003), and in (Boden, 1998), she states that her account distinguishes three main forms of creativity, these being exploration, transformation and combination. Adding to the uncertainty is the observation that only the strong definition has the power to resolve the creativity paradox, arguably forcing us to recognise not two forms of creativity, or three, but one: transformation.*”

Quanto às três formas de criatividade, Boden chama de *comportamento combinatorial*, *comportamento explorador* e *comportamento transformacional*. Sobre o primeiro, “Exemplos incluem muito das imagens poéticas, e também analogias – onde duas idéias novas associadas compartilham heranças de estruturas conceituais.”⁸ (Boden, 1998, p. 348). Além de analogias, piadas se enquadraram neste tipo de comportamento criativo. Boden caracteriza uma semelhança entre o segundo e terceiro tipo. O comportamento explorador é definido ciclicamente, e “(...) envolve a geração de novas idéias pela exploração de espaços conceituais estruturados. Isso muitas vezes resulta em estruturas (“idéias”) que não são novidades, mas inesperadas.”⁹. O terceiro é considerado por Boden como a “criatividade genuína”. O comportamento transformador altera, remove, ou adiciona um ou mais conceitos relativos ao espaço conceitual trabalhado:

“Quanto mais fundamental a transformação, e/ou mais fundamental a dimensão que é transformada, mais diferenciadas serão as novas possibilidades de estruturas. O choque de espanto que atende tais (e previamente impossíveis) idéias, é muito maior do que a surpresa ocasionada por meras improbabilidades, por mais inesperadas que sejam. Se as transformações forem muito extremas, a relação entre os velhos e novos espaços não serão imediatamente aparentes, em tais casos, as novas estruturas serão ininteligíveis, e muito provavelmente rejeitadas. Contudo, pode levar um tempo para relação entre dois espaços ser reconhecida e aceitada de forma geral (*idem, ibdem*).”¹⁰

Wiggins dá maior importância para a classificação exploradora/transformacional ao enunciar quatro axiomas dos Sistemas Criativos (ver [seção 3.2](#), p. 58). Thornton (2007, p. 3–4) pontua que em algumas situações, uma criatividade-exploradora não é menos criativa que uma criatividade-transformadora. O comportamento explorador, como uma *exploração guiada*, é muito útil em atividades onde se requer “(...) a utilização de heurísticas e mapas para identificar conceitos valiosos dentro de um espaço conceitual existente”¹¹ (ver [subseção 2.4.1](#), p. 41). Neste sentido, comportamentos criativos *apenas transformadores* desenvolvem novos espaços conceituais que serão úteis para o comportamento explorador guiado.

⁸ Tradução nossa de “*Examples include much poetic imagery, and also analogy-wherein the two newly associated ideas share some inherent conceptual structure.*”

⁹ Tradução nossa de “*(...) involves the generation of novel ideas by the exploration of structured conceptual spaces. This often results in structures (“ideas”) that are not only novel, but unexpected.*”

¹⁰ Tradução nossa de “*The more fundamental the transformation, and/or the more fundamental the dimension that is transformed, the more different the newly-possible structures will be. The shock of amazement that attends such (previously impossible) ideas is much greater than the surprise occasioned by mere improbabilities, however unexpected they may be. If the transformations are too extreme, the relation between the old and new spaces will not be immediately apparent. In such cases, the new structures will be unintelligible, and very likely rejected. Indeed, it may take some time for the relation between the two spaces to be recognized and generally accepted.*”

¹¹ Tradução nossa de “*the use of heuristics and maps to identify valuable concepts within an existing conceptual space*”

Neste ponto Wiggins (2006, p. 451) apresenta uma definição cílica, e generalizadora de criatividade (forte), auxiliada por outras quatro definições (fracas). O que se observa é uma estrutura de pensamento piramidal, sendo que a definição geral , “A performance de tarefas que, quando executados por um humano, são consideradas criativas”¹², é suportada por quatro definições específicas (ver Tabela 1, p. 51).

Tabela 1 – Definições formais de criatividade por Wiggins (2006, p. 451)

Criatividade	“O estudo e suporte, através de meios e métodos computacionais, do comportamento exibido por sistemas naturais e artificiais, que podem ser considerados criativos se exibidos em humanos.” ¹³
Computação criativa	“O estudo e suporte, através de meios e métodos computacionais, do comportamento exibido por sistemas naturais e artificiais, que são considerados criativos”. ¹⁴
Sistemas criativos	“Uma coleção de processos, naturais ou automáticos, que são capazes de alcançarem ou simular em comportamentos que em humanos seriam considerados criativos”
Comportamento Criativo	“Um ou mais dos comportamentos exibidos por um sistema criativo” ¹⁵

3.1.2 Criatividade, códigos e imagens mentais

Para McLean (2011, p. 24–25), um comportamento criativo pode ser descrito como o processo de estruturação uma *imagem mental*. Mais especificamente, a imagem mental é um símbolo, e seu significado é governado por regras gramaticais e sociais (códigos). Por outro lado, McLean estabelece o conceito de *imagem mental* como uma hierarquia de códigos simbólicos, visuais e gramaticais, como descrita pela teoria da Codificação Dual (PAIVIO, 1990 apud MCLEAN, 2011, p. 25–29):

“Seu [Paivio] argumento não é que existem dois códigos, mas sim que existe uma hierarquia de códigos, que se ramificam no topo em códigos lingüísticos discretos e códigos de percepção contínua, que Paivio nomeia como *logogens* e *imagens* respectivamente. Esta divisão é apresentada em seus processos simultâneos; os seres humanos são capazes de compreender a linguagem ao mesmo tempo que podem ver imagens. (...), é fácil para seres humanos ler um texto enquanto simultaneamente ouve um discurso. A explicação oferecida pela teoria da Codificação Dual é que existem sistemas de símbolos distintos, mas integrados, para linguagens e figuras.”¹⁶

¹² Tradução nossa de “The performance of tasks which, if performed by a human, would be deemed creative”

¹³ Tradução de ‘The study and support, through computational means and methods, of behaviour exhibited by natural and artificial systems, which would be deemed creative if exhibited by humans’.

¹⁴ Tradução de *The study and support, through computational means and methods, of behaviour exhibited by natural and artificial systems, which would be deemed creative if exhibited by humans..*

¹⁵ Tradução de *One or more of the behaviours exhibited by a creative system.*

¹⁶ Tradução nossa de “His contention is not that there are two codes, but rather that there is a hierarchy of code, which branch at the top into discrete linguistic codes and continuous perceptual codes, which Paivio names logogens and images respectively. This split is shown in their concurrent processing; humans are able to comprehend language while simultaneously attending to imagery. (...), humans find it easy to simultaneously read text and listen to speech. The explanation offered by Dual Coding Theory is that there are distinct, yet integrated symbol systems for imagery and language.”

Para McLean, a teoria da Codificação Dual é suportada pela Neuropsicologia, onde “De maneira muito ampla, o hemisfério esquerdo é especializado para linguagem, e o direito para tarefas visuais-espaciais.”¹⁷ (MARTIN, 2006 apud MCLEAN, 2011, p. 128–129; *ibidem*), o que por outro lado, é admitido como uma perigosa generalização. Para exemplificar a formação de imagens (conceitos, lembranças, sentidos) durante a leitura e escrita de texto, apresentaremos um pequeno exemplo de código que formaliza, em uma linguagem de programação, um simples algoritmo musical (ver subseção 3.1.3, p. 54).

Antes de oferecer o exemplo, é importante esclarecer que o improvisador-programador será obrigado a adequar a imagem mental às Dimensões cognitivas da linguagem utilizada (CHURCH; GREEN, 2008 apud MCLEAN, 2011, p. 95). Se por um lado tais Dimensões foram elaboradas como parâmetros críticos para a estruturação de uma linguagem, elas não precisam ser utilizadas apenas como “(...)uma lista de verificação de um *design* bom ou ruim”¹⁸. Elas podem descrever um filtro de contextos desejáveis. Por exemplo, padrões e estilos de comentários, que explicam textualmente a imagem e seu resultado; a disposição espacial do código, ou nomes de variáveis e funções que sugerem imagens mentais específicas. No caso de uma linguagem de programação de propósito geral, são as regras definidas pela comunidade desenvolvedora da linguagem. No caso de uma linguagem de domínio específico (DSL), o improvisador-programador deve ter em mente que tipo de música quer realizar com seu alfabeto pessoal. Neste sentido, as Dimensões cognitivas da linguagem podem auxiliar na elaboração de performances de codificação ao vivo (ver Tabela 2, p. 52).

Tabela 2 – Dimensões cognitivas da Notação para linguagens de programação. **Fonte:** (CHURCH; GREEN, 2008 apud MCLEAN, 2011).

Dimensão	Significado
Abstração	“Disponibilidade de mecanismos de abstração” ¹⁹
Dependências escondidas	“Invisibilidade de ligações importantes entre entidades.” ²⁰
Compromisso prematuro	“Restrição na ordem de execução das coisas.” ²¹
Notação secundária	“Notação diversa da sintaxe formal.” ²²
Viscosidade	“Resistência à mudança.” ²³
Proximidade de mapeamento	“Proximidade de representação para o domínio-alvo.” ²⁴
Consistência	“Semânticas similares são expressadas em formas sintáticas similares.” ²⁵
Dispersividade	“Prolíxidade da linguagem.” ²⁶
Tendência ao erro	“Probabilidade de erros.” ²⁷
Operações mentais difíceis	“Demanda de recursos cognitivos.” ²⁸
Provisoriedade	“Grau de compromisso com ações e marcos.” ²⁹
Função de expressividade	“medida em que o efeito de um componente pode ser inferida.” ³⁰

¹⁷ Tradução nossa de “Very broadly speaking, the left hemisphere is specialised for language, and the right for visuospatial tasks”

¹⁸ Tradução nossa de “(...)a checklist of a good or bad design”

No caso específico da notação secundária, McLean (2011, p. 28) argumenta que esta dimensão cognitiva possui a capacidade de aumentar a representatividade da imagem mental. Porém este aumento não significa necessariamente uma vantagem. No caso de linguagens *Patcher* (como o PureData³¹), McLean sugere que o aumento das possibilidades de notação do código aumenta a propriedade aglutinativa de símbolos da linguagem, o que causaria uma certa impopularidade entre programadores profissionais no contexto de sua tese (mas não entre uma parcela considerável de músicos programadores e programadores interessados em música). A relação entre símbolos discretos e imagens mentais seriam próprios para uma tarefa específica:

“A falta de sucesso de uso de [linguagens] de Programação Visual sugere que as relações entre os símbolos discretos e imagens mentais são, por natureza, particulares à tarefa em mãos. Devemos olhar, em seguida, não para maneiras de agregar artificialmente a metáfora visual para a programação (...), mas por formas de apoiar o próprio sistema de imagens mentais de um programador, integrado com o texto do código fonte.”³²

Como lembra McLean (2011, p. 102–103), linguagens de programação visual apresentam uma vantagem, ou suas “dimensões visivelmente não-restritas, de forma que você pode colocar objetos em qualquer lugar com pouco ou nenhum significado sintático.”³³ Por outro lado, sua notação secundária irrestrita pode dificultar operações que em linguagens textuais, de propósito geral, seriam triviais, por exemplo um laço iterativo. No entanto, como isso não deve ser tomado como uma dicotomia entre vantagem e desvantagem:

“O [discurso] acima não é um ataque sobre as linguagens *Patcher* - sintaxe não é tudo. Bem, para o [programa] interpretador sintaxe é tudo, mas para o programador, que é o nosso foco, é apenas metade da história. Porque linguagens *Patcher* têm notações secundárias notavelmente livres, elas nos permitem estabelecer programas como desejarmos, nós podemos abraçar essa liberdade visual em um relance, para fazer belos *patches* que,

¹⁹ Tradução nossa de *Avaliability of abstraction mechanisms*.

²⁰ Tradução nossa de *Invisibility of important links between entities..*

²¹ Tradução nossa de *Constraints on the order of doing things..*

²² Tradução nossa de *Notation other than formal syntax..*

²³ Tradução nossa de *Resistance to change..*

²⁴ Tradução nossa de *Closeness of representation to target domain..*

²⁵ Tradução nossa de *Similar semantics are expressed in similar syntactic forms.*

²⁶ Tradução nossa de *Verbosity of language..*

²⁷ Tradução nossa de *Likelihood of mistakes..*

²⁸ Tradução nossa de *Demand on cognitive resources..*

²⁹ Tradução nossa de *Degree of commitment to actions or marks..*

³⁰ Tradução nossa de *Extent to which the purpose of a component may be inferred..*

³¹ Disponível em <<http://www.puredata.org>>

³² Tradução nossa de “The lack of success of general purpose Visual Programming suggests that relationships between discrete symbols and mental imagery are by nature particular to the task at hand. We should look then not for ways of adding ‘artificial’ visual metaphor to programming (...), but for ways of supporting a programmer’s own system of imagery, integrated with the text of source code.”

³³ Tradução nossa de “.”

através de conteúdos e formas, se relacionam estruturalmente ao humano, de maneiras que aquele [código] linguístico não pode fazer sozinho. Isto é o porque nós chamamos estas languages visuais; enquanto a sintaxe da linguagem não é visual, a notação é bem assim.”³⁴

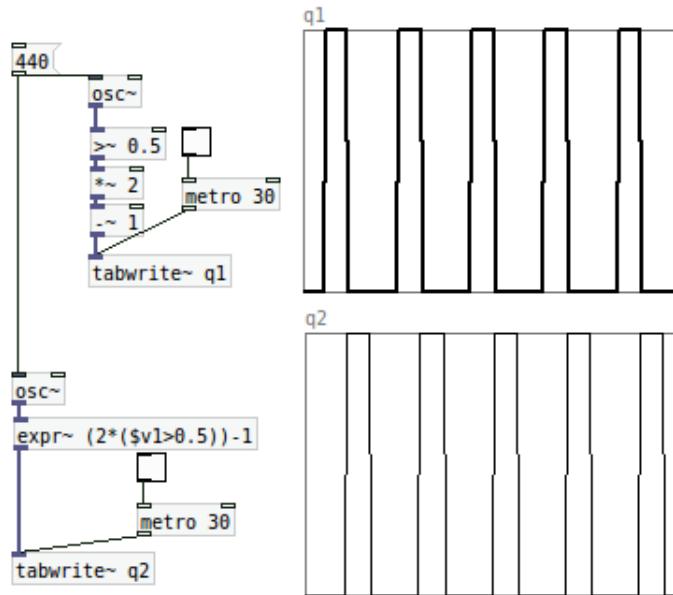


Figura 14 – Exemplo de uma característica de alta viscosidade, proporcional às diversas possibilidade de notação secundária no PureData. **Fonte:** autor.

A especificidade da linguagem significa a estratégia de manipulação de um material artístico. No caso de McLean (2011, p. 97), as dimensões cognitivas são usadas para separar os materiais (o que é representado como material sonoro), da maneira como serão manipulados (o como é representado o material sonoro), em um fluxo temporal.

3.1.3 Comportamento Criativo por Bricolagem

Vamos ilustrar um pequeno processo da *estratégia transversal* com o *software* Tidal. A estratégia transversal é uma etapa de um processo criativo, chamado McLean, de programação por bricolagem. Dentro deste processo, o improvisador elabora imagens mentais, codifica, observa o resultado do código, e recodifica:

“A Figura 6.2 [Figura 15] caracteriza a programação por bricolagem como um laço retroalimentado envolvendo o algoritmo escrito, sua interpretação, e a percepção do programador e sua reação do resultado

³⁴ Tradução nossa de “The above is not an attack on Patcher languages – syntax is not everything. Well, to the interpreter syntax is everything, but to the programmer, who is our focus, it is only half the story. Because Patcher languages have such remarkably free **secondary notation**, they allow us to lay programs out however we like, and we may embrace this visual freedom in making beautiful patches that through shape and form relate structure to human at a glance, in ways that linguistic alone cannot do. That is why we call these visual languages; while the language syntax is not visual, the notation is very much so”

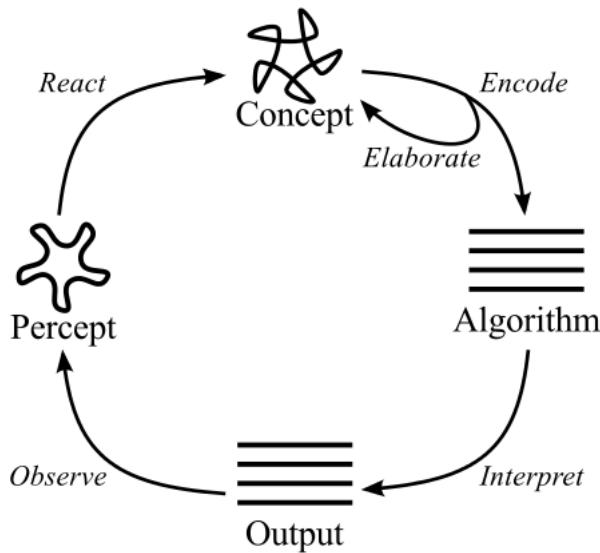


Figure 6.2: The process of action and reaction in bricolage programming

Figura 15 – Modelo de bricolagem para o processo criativo realizado por um artista-programador. **Fonte:** McLean (2011, p. 122).

ou comportamento [do algoritmo]. (...). No começo o programador tem um conceito meio-formado que só atinge consistência interna através do processo de ser expresso como um algoritmo. O laço interno é onde o programador elabora o objetivo de suas imaginações, e o laço externo é onde essa trajetória está fundamentada na pragmática do que elas realmente têm que fazer. Através deste processo ambos algoritmos e conceitos são desenvolvidos até que o programador sinta que um se aplica com o outro, ou de outra forma julga o processo criativo finalizado.”³⁵

Segundo McLean e Wiggins (2010b, p. 2), *Tidal* é uma linguagem de composição generativa, onde “padrões podem ser compostos de numerosos subpadrões em uma variedade de maneiras e para uma profundidade arbitrária, para produzir [partes] inteiras complexas de partes simples”³⁶. Amostras sonoras representam imagens mentais de suas fontes (por exemplo “sn” para *snare*, caixa-clara), com ritmos organizados com o auxílio de símbolos delimitadores de tempo (como espaço, “ “, e colchetes, “[”, “]”, { e }). Ritmos podem ser revertidos (*rev*), diminuídos e aumentados (*slow*, *density*), iterados (*every*) para recombinação permutação, padrões mais complexos (*can*), como o algoritmo de björklund que simula ritmos tradicionais³⁷. Efeitos de panoramização, atraso (*delay*), filtros e

³⁵ Tradução nossa de “*Figure 6.2 characterises bricolage programming as a creative feedback loop encompassing the written algorithm, its interpretation, and the programmer’s perception and reaction to its output or behaviour. (...). At the beginning, the programmer may have a half-formed concept, which only reaches internal consistency through the process of being expressed as an algorithm. The inner loop is where the programmer elaborates upon their imagination of what might be, and the outer where this trajectory is grounded in the pragmatics of what they have actually made. Through this process both algorithm and concept are developed, until the programmer feels they accord with one another, or otherwise judges the creative process to be finished.*”

³⁶ Tradução nossa de “*patterns may be composed of numerous subpatterns in a variety of ways and to arbitrary depth, to produce complex wholes from simple parts*”

³⁷ Cf. TOUSSAINT, 2005

comunicação de rede. No Exemplo 3.1. a imagem mental é a demanda da linguagem, que é produzir Música Eletrônica para Dançar.

“Tidal é uma linguagem de padrões imbebida em uma linguagem de programação Haskell, consistindo de representação de padrão, uma biblioteca de padrões geradores e combinadores, um [mecanismo] de agendamento de eventos e uma interface para programar ao vivo. Esta é uma extensiva re-escrita de um trabalho anterior introduzido sobre o título *Petrol* [McLean e Wiggins (2010a)]. Extensões incluem melhoramentos de representação de padrão e uma integração totalmente configurável do protocolo Open Sound Control [opensoundcontrol.org (2002)] (McLean; WIGGINS, 2010b) ”³⁸

Exemplo 3.1 (Exemplo de Estratégia Transversal)

Imagen mental: um *loop* sincopado, mas bastante regular, descrito em um compasso. Em uma “partitura-mental”, estruturamos o primeiro tempo com um baixo, que volta a tocar na segunda semicolcheia do terceiro tempo. No Segundo tempo, silêncio. No quarto tempo uma caixa aberta:

The image shows two staves of musical notation. The top staff is for the 'Snare drum' and the bottom staff is for 'bass3'. Both staves have a common time signature. The 'Snare drum' staff has a C-clef and a 120 BPM tempo marking. It contains four measures. The first measure has a eighth note followed by a rest. The second measure has a eighth note followed by a rest. The third measure has a eighth note followed by a rest. The fourth measure has a eighth note followed by a rest. The 'bass3' staff has a bass clef and a 120 BPM tempo marking. It contains four measures. The first measure has a quarter note followed by a rest. The second measure has a quarter note followed by a rest. The third measure has a quarter note followed by a rest. The fourth measure has a quarter note followed by a rest.

O padrão acima pode ser elaborado em uma voz (d1), que redireciona (\$) a função que toca amostras sonoras (sound). Esta função lê uma corrente de caracteres (**string**) separados por um espaço em branco. Espaços em branco são delimitadores temporais. Cada subdivisão temporal é representada por delimitadores como [e].

```
-- Electronic Dance Music, BPM = 120
-- tempo 1 - baixo          (bass)
-- tempo 2 - silencio       (silence)
-- tempo 3 - silencio + baixo
-- tempo 4 - caixa          (sn e sn:4)
d1 \$ (sound "bass3 silence [silence bass3] sn:4")
```

Sonoramente, é útil para começar. Mas uma Música Eletrônica para Dançar requer mais elementos. Seguiremos com mais dois passos. Podemos complementar os ritmos com uma caixa e um baixos mais secos no segundo e terceiro tempo.

³⁸ Tradução nossa de “*Tidal is a pattern language embedded in the Haskell programminglanguage, consisting of pattern representation, a library of pattern generators and combinators, an event scheduler and programmer’s live coding interface. This is an extensive re-write of earlier work introduced under the working title of Petrol [15]. Extensions include improved pattern representation and fully configurable integration with the Open Sound Control (OSC) protocol [16]* ”

House bass

Snare drum

bass3

```
-- Electronic Dance Music, BPM = 120
-- tempo 1 - baixo          (bass)
-- tempo 2 - silencio       (silence)
-- tempo 3 - silencio + house
-- tempo 4 - caixa          (sn e sn:4)
d1 \$ (sound "bass3 sn [silence house] sn:4")
```

É possível também fazer com que este padrão reduza seu tempo pela metade a cada quatro tempos :

House bass

Snare drum

bass3

```
-- Electronic Dance Music, BPM = 120
-- com uma caixa seca no segundo tempo
-- e uma caixa aberta no quarto tempo
-- A cada 4 tempos, o ritmo diminui pela metade
-- e depois volta ao normal.
d1 \$ every 4 (density 0.5) (sound "bass3 sn [silence house] sn:4")
```

Para McLean (2011, p. 130), esta estratégia criativa, de programar “no momento”, a partir de um arquivo de texto em branco, com uma imagem mental do resultado sonoro (ou visual), é caracterizada pela bricolagem. No início do exemplo acima, o programador elabora um meio-conceito do que quer fazer, cuja expressão apenas ganha existência através da codificação (ver Figura 15, p. 55). As fases de observação, e reação levam o improvisador programador à reconceitualização, e um novo código é escrito. No entanto,

ao invés de finalizar, o improvisador segue desenvolvendo.

3.2 Quadro Conceitual de sistemas criativos

Uma maneira adequada de descrever um sistema criativo (ou parte dele) considera um *Universo de Conceitos*:

O universo, \mathcal{U} , é um espaço multidimensional, no qual dimensões são capazes de representar qualquer coisa, e todos os possíveis conceitos distintos correspondentes àqueles pontos em \mathcal{U} (...) Para tornar a proposta um espaço-tipo possível, permitirei que \mathcal{U} contenha todos os conceitos abstratos, bem como os concretos, e que é possível representar os artefatos tanto completos e incompletos (WIGGINS, 2006, p. 451).³⁹

Wiggins esclarece que Boden não reconhece de forma explícita \mathcal{U} , “ela borra a distinção entre as regras que determinam a adesão do espaço (...) e outras disposições que possam permitir a construção e/ou detecção de um conceito representado por um ponto no espaço” (*Idem, ibdem*). Espaços conceituais \mathcal{C} , finitos ou infinitos são definidos como restrições de um universo \mathcal{U} , caracterizando um conjunto não-determinístico de conhecimentos, representações, e conceitos:

“A noção-chave na teoria de Boden é aquele do espaço conceitual. Enquanto nenhuma definição formal é provida, é comum interpretar esta frase literalmente, tomando o espaço conceitual sendo um espaço de conceptualizações, ou representações de conceitos (Thornton, 2007, p .7).”⁴⁰

Wiggins (2006, p. 452) considera “(...) um universo, \mathcal{U} , um espaço multidimensional, cujas dimensões são capazes de representar qualquer coisa, e todos possíveis conceitos distintos correspondentes com distintos pontos em \mathcal{U} .⁴¹ Uma incomensurabilidade é evitada através de uma restrição por meio de quatro axiomas. O primeiro axioma (Universalidade) estabelece que o universo \mathcal{U} pode conter tanto conceitos bem definidos (completos), parcialmente definidos (incompletos), e o mais incompleto dos conceitos. Os dois primeiros são representados pela letra c , enquanto o último é representado por T . O segundo axioma (Não-identidade dos conceitos), estabelece que dois conceitos em \mathcal{U} são mutuamente diferentes entre si ($c_1 \neq c_2$), e não são um Universo. Esta correção restringe a recursividade de conceitos, o que daria ênfase ao comportamento explorador e anularia a importância do comportamento

³⁹ Tradução de *The universe, U, is a multidimensional space, whose dimensions are capable of representing anything, and all possible distinct concepts correspond with distinct points in U. (...) To make the proposal as state-spacelike as possible, I allow that U contains all abstract concepts as well as all concrete ones, and that it is therefore possible to represent both complete and incomplete artefacts*

⁴⁰ Tradução nossa de “The key notion in Boden’s theory is that of the conceptual space. While no formal definition has been provided, it is common to interpret the phrase literally, taking the conceptual space to be a space of conceptualisations or concept representations.”

⁴¹ Tradução nossa de “The universe, U, is a multidimensional space,whose dimensions are capable of representing anything, and all possible distinct concepts correspond with distinct points in U”

transformacional. O terceiro axioma (Inclusão Universal 1) define que *espaços conceituais* \mathcal{C} , que contêm instâncias de conceitos c , são subconjuntos não-estritos do conjunto \mathcal{U} . O quarto axioma (Inclusão Universal 2) estabelece que existe um conceito vazio, T , em qualquer espaços conceituais \mathcal{C} .

Wiggins define o Universo de Conceitos (\mathcal{U}_x) como um conjunto não estrito dos *Espaços Conceituais* (\mathcal{C}_x) de Margaret Boden (2004). Isto é, um Universo de Conceitos a respeito de alguma coisa, no nosso caso da improvisação de códigos (ver Equação 3.2, p. 59).

$$\mathcal{U}_{livecoding} = [\mathcal{C}_{Tecelagem}, \mathcal{C}_{Audiovisual}, \mathcal{C}_{Dança}, \mathcal{C}_{Música}, \dots, ?] \quad (3.1)$$

McLean (2006) ainda descreve regras que validam concepções diferentes entre espaços conceituais \mathcal{C} diversos em um Universo de Conceitos \mathcal{U} (ver Tabela 3). McLean realiza uma comparação entre o *Universo de possibilidades* de Wiggins com o *Modelo de Improvisação* de Pressing (ver subseção 3.2.1, p. 59). No entanto, McLean argumenta que:

Pressing discute comportamento criativo no contexto do Modelo de Improvisação, e de fato é parte do Quadro conceitual de Sistemas Criativos. (...) Durante a transferência de notação do Modelo de Improvisação para a Ferramenta de Sistemas Criativos, nós consideramos improvisação musical de uma maneira clara e temos uma linguagem comum na qual comparar com outros modelos ⁵².

3.2.1 O modelo de improvisação

Segundo Pressing, o Modelo de Improvisação é “um esboço para uma teoria geral da improvisação integrada com preceitos da Psicologia Cognitiva (...) teoria do comportamento

⁴² Tradução de *A universe of all possible concepts*.

⁴³ Tradução de (...) a partial function from \mathcal{L} to functions yielding real numbers in $[0, 1]$. (...) 0.5 to mean Boolean true and less than 0.5 to mean Boolean false; the need for the real values will become clear below.

⁴⁴ Tradução de *A function interpreting \mathcal{R} , resulting in a function indicating adherence of a concept to \mathcal{R}*

⁴⁵ Tradução de *All conceptual spaces are non-strict subset*.

⁴⁶ Tradução de *A function interpreting \mathcal{R} , resulting in a function indicating adherence of a concept to \mathcal{R}* .

⁴⁷ Tradução de *Rules defined within \mathcal{L} to define a traversal strategy to locate concepts within \mathcal{U}*

⁴⁸ Tradução de (...) set of rules which allows us to evaluate any concept we find in \mathcal{C} and determine its quality, according to whatever criteria we may consider appropriate.

⁴⁹ Tradução de *Rules defined within \mathcal{L} which evaluate the quality or desirability of a concept c* .

⁵⁰ Tradução de *We need a means not just of defining the conceptual space, irrespective of order, but also, at least notionally, of enumerating it, in a particular order, under the control of \mathcal{T} – this is crucial to the simulation of a particular creative behaviour by a particular \mathcal{T}* .

⁵¹ Tradução de *A function interpreting the traversal strategy \mathcal{T} , informed by \mathcal{R} and \mathcal{E} . It operates upon an ordered subset of \mathcal{U} (of which it has random access) and results in another ordered subset of \mathcal{U}* .

⁵² Tradução de *However Pressing does discuss creative behaviour in the context of the IM, and indeed the CSF is in part. (...) In transferring the IM to the notation of the CSF we may consider music improvisation in a clearer manner and have a common language in which to compare it with other models*.

Tabela 3 – Definições formais do Universo de possibilidades de Wiggins (2006), ou Universo de Conceitos por McLean (2006).

Representação	Nome	Significado
c	Conceito	Uma instância de um conceito, abstrato ou concreto (WIGGINS, 2006).
\mathcal{U}	Universo de Conceitos	Superconjunto não restrito de conceitos. (WIGGINS, 2006). “Um universo de todos conceitos possíveis” (McLean, 2006) ⁴²
\mathcal{L}	Linguagem	Linguagem utilizada para expressar regras.
\mathcal{A}	Alfabeto	Alfabeto da linguagen que contêm caracteres apropriados para expressão das regras
\mathcal{R}	Regras de validação	Validam os conceitos em um universo, se apropriados ou não para o espaço trabalhado.
$[[.]]$	Função de interpretação	“Uma função parcial de \mathcal{L} para funções que resultam em números reais entre [0, 1] (...) 0.5 [ou maior] significa uma verdade booleana e menos que 0.5 significa uma falsidade booleana; a necessidade disso para valores reais se tornará clara abaixo” (WIGGINS, 2006, p. 452) ⁴³
$[[\mathcal{R}]]$	Regras de validação	“Uma função que interpreta \mathcal{R} , resultando em uma função indicando aderência ao conceito em \mathcal{R} ” ⁴⁴
$\mathcal{C} = [[\mathcal{R}]](\mathcal{U})$	Espaço Conceitual	“Todos espaços conceituais são um subconjunto não-estrito de \mathcal{U} ” ⁴⁵ Um subconjunto contido em \mathcal{U} (WIGGINS, 2006). Uma função que interpreta \mathcal{R} , resultando em uma função que indica aderência ao conceito em \mathcal{R} ⁴⁶
\mathcal{T}	Regras de detecção	“Regras definidas dentro de \mathcal{L} para definir estratégias transversais para localizar conceitos dentro de \mathcal{U} ” (McLean, 2006) ⁴⁷
\mathcal{E}	Regras de qualidade	“(...) conjunto de regras que permitem-nos avaliar qualquer conceito que nós encontramos em \mathcal{C} e determinar sua qualidade, de acordo com critérios que nós considerarmos apropriados” (WIGGINS, 2006, p.453) ⁴⁸ “Regras definidas dentro de \mathcal{L} para avaliar a qualidade ou a desejabilidade do conceito c ” (McLean, 2006) ⁴⁹
$<<< \mathcal{R}, \mathcal{T}, \mathcal{E} >>>$	Função de interpretação	Uma regra necessária para definir o espaço conceitual, “independentemente da ordem, mas também, ficionalmente, enumerá-los em uma ordem particular, sob o controle de \mathcal{T} – isto é crucial para a simulação de um comportamento criativo de um \mathcal{T} particular (WIGGINS, 2006) ⁵⁰ . “Uma função que interpreta a estratégia transversal \mathcal{T} , informada por \mathcal{R} e \mathcal{E} . Opera sobre um subconjunto ordenado de $mathcal{U}$ (do qual tem acesso randômico) e resulta em outro subconjunto ordenado de \mathcal{U} .” ⁵¹

de improvisação na música” (Pressing, 1987, p. 2). Este modelo será utilizado para especificar elementos de uma performance exemplar, como o caso investigado neste trabalho. Por exemplo, uma improvisação particionada em diferentes sequências pode ser parcialmente mapeada em categorias, como blocos sonoros, referentes conceituais e normas estilísticas, conjuntos de objetivos

e processos. Este nos pareceu um modelo mais transparente para o compositor, músico e intérprete. O que não quer dizer que é possível readequar ambos para nosso interesse. Um sumário sobre o modelo de improvisação é apresentado na [Tabela 4](#). Por seu caráter lógico, parece ser uma possibilidade interessante, e assumiremos como tal.

Tabela 4 – Definições formais do Modelo de improvisação de Jeff Pressing (1987), segundo McLean (2006, p. 2).

Representação	Significado
E'	Um bloco de eventos sonoros ⁵³
K'	Uma seqüência de blocos de eventos E, onde um bloco de eventos não se sobrepõe com o seguinte ⁵⁴
I'	Uma improvisação, particionada por interrupções em um número de K sequências ⁵⁵
R'	Um referente opcional, tal como uma partitura ou uma norma estilística ⁵⁶
G'	Um conjunto de objetivos ⁵⁷
M'	Uma memória de longo prazo ⁵⁸
O'	Um conjunto de objetos ⁵⁹
F'	Um conjunto de características dos objetos ⁶⁰
P'	Um conjunto de processos ⁶¹

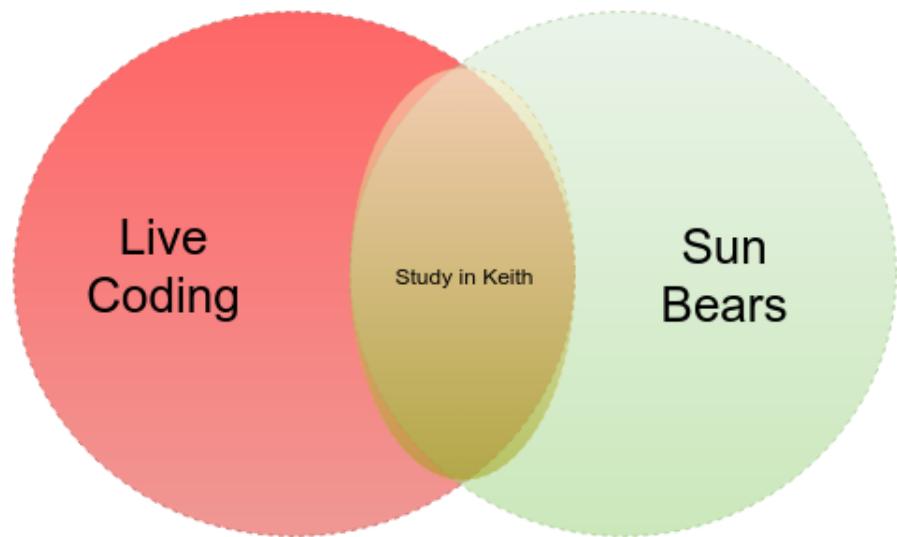


Figura 16 – Representação da justaposição entre dois espaços conceituais. A região em marrom representa um grupo de conceitos transitórios, bem como os limites desta transição. **Fonte:** autor.

⁵³ *A cluster of sound events.*

⁵⁴ A sequence of E event clusters, where event cluster onsets do not overlap with those of a following one

⁵⁵ An improvisation, partitioned by interrupts into a number of K sequences

⁵⁶ An optional referent, such as a score or stylistic norm

⁵⁷ A set of current goals.

⁵⁸ Long term memory.

⁵⁹ An array of objects.

⁶⁰ An array of objects Features.

⁶¹ An array of Process

3.3 Diagramação dos espaços conceituais

Formalmente, Thornton (2007, p. 8–9) estabelece que espaços conceituais podem ser definidos pela gramática gerativa (ver exemplo ??),

Exemplo 3.2 (Definição gerativa de um conceito)

Podemos definir frases que conceitualizam o mesmo princípio, como “aquele pessoa riu”, “a mulher andou” ou “um homem correu”, a partir de uma regras gerativas:

```

S -> NP V
NP -> Det N
Det -> o(a) | um | aquele | aquela
N -> pessoa | homem | mulher
V -> correu | andou | cantou | riu
  
```

Como redes hierárquicas, onde conceitos são verticalizados, e a relação entre os nós é representado por linhas (ver ??, p. ??):

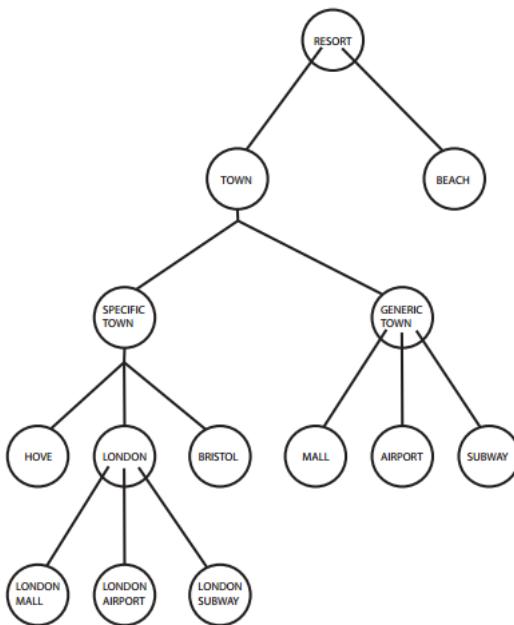


Figura 17 – Verticalização de uma rede de conceitos. **Fonte:** ([Thornton, 2007](#))

Esta rede pode ser subdividida em conceitos primitivos, categóricos, e conceitos compostos. Um conceito primitivo exemplar é *live coding*. Como demonstramos ao longo do trabalho, um conceito primitivo pode gerar por exploração, conceitos categóricos, como *algorave*, *sensibilidade computacional*, *agenciamento sonoro*. Por composição, Thornton (2007, p. 11) coloca da seguinte forma:

“Na linguagem da lógica, categorização é simplesmente o arranjo de contínuos de uma *disjunção*. Composição, por outro lado, é o arranjo

de constituintes de uma relação (i.e., uma função) tal como cada um que segue uma regra de uma parte ou componente. No caso mais simples, isso pode significar que a formação de um conjunto simples (i.e., introdução da relação AND [E]). Mas o número de possibilidades depende de quantas relações podem ser aplicadas. Com mais de uma relação disponível, existem outras maneiras de formar conceitos compostos.”⁶²

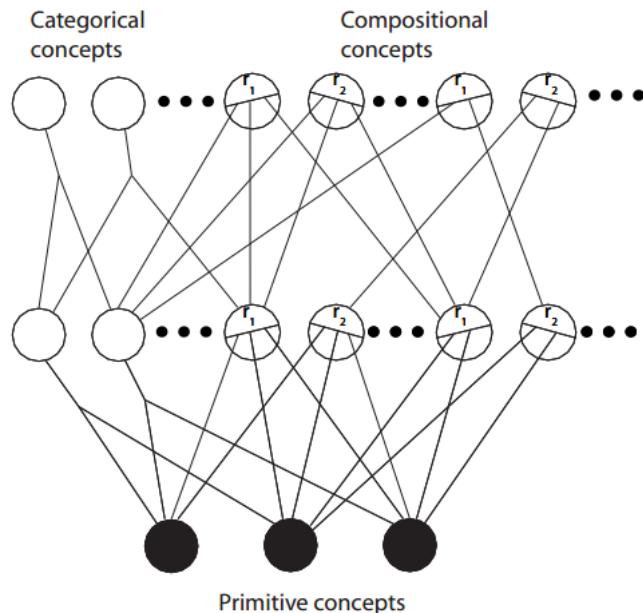


Figura 18 – Conceitos categóricos e Conceitos compostos. **Fonte:** (Thornton, 2007)

É importante lembrar que esta hierarquia verticalizada aumenta em complexidade, conforme novas relações são encontradas no espaço conceitual, definido por Thornton (2007, p. 15-16) como *complexidade conceitual bruta*, “(...) qualquer conceito [cujo] o número de conceitos que deve ser desenvolvido a partir da fundação dada para garantir a sua construção”⁶³:

Conceitos também podem ter extensões em um espaço conceitual, como um subgrupo que surge de uma nova verticalização (oposta) do campo definido pela complexidade conceitual bruta:

“Um universo conceitual é então uma paisagem em que nós esperamos encontrar áreas pronunciadas de baixa e alta complexidade – ‘zonas esponjosas’ como nós chamaremos – localizadas em partes da hierarquia em que dominam imediatamente um conjunto de conceitos formados

⁶² Tradução nossa de “*n the language of logic, categorisation is simply the arrangement of the constituents a disjunction. Composition, on the other hand, is the arrangement of the constituents in a relationship (i.e., a function) such that each one plays the role of a part or component. In a simple case this might mean the formation of a straightforward conjunction (i.e., introduction of the AND relationship). But the number of compositional possibilities depends on how many relationships can be applied. With more relationships available, there are more ways to form compositional concepts.*”

⁶³ Tradução nossa de “*The gross conceptual complexity of any concept is the number of concepts which must be developed from the given foundation to guarantee its construction*”

continuar
até as
zonas
esponjo-
sas para
concluir
com um
pedaço
das
zonas,
que é a
intersec-
ção dos

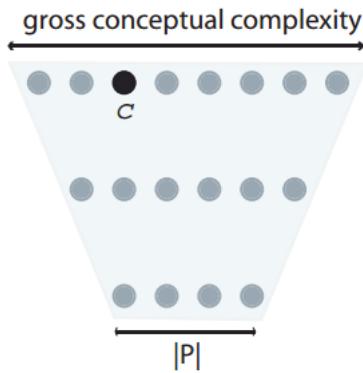


Figura 19 – Complexificação da Verticalização de uma rede de conceitos. **Fonte:** (Thornton, 2007)

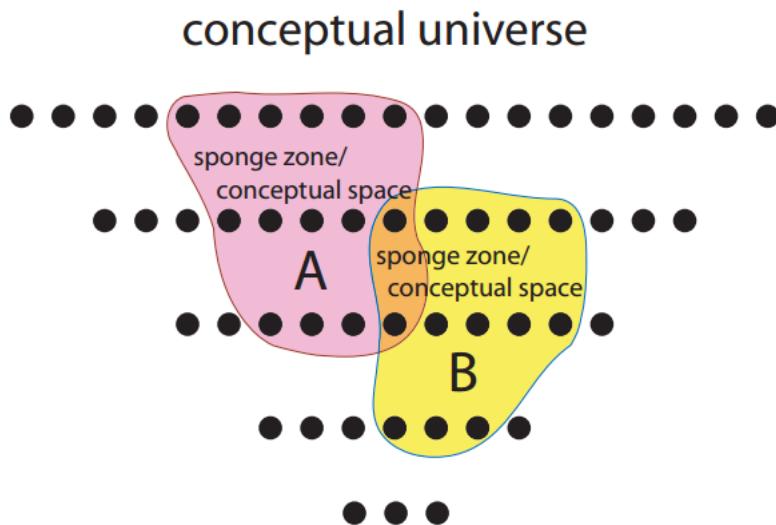


Figura 20 – Extensões conceituais em uma malha de Complexificação. **Fonte:** (Thornton, 2007)

pelos constituintes de uma explicação útil de um quadro conceitual. Ao acessar conceitos nesta zona, o agente move-se em lucro representacional, colhendo uma recompensa de representação que excede o custo de desenvolvimento.”⁶⁴

Para simplificação do nosso estudo (ver Capítulo 4, p. 71), simplificaremos o processo como dois grupos conceituais em intersecção (ver exemplo 16), com uma representação visual que indica a aderência de alguns, não todos, conceitos de dois diferentes espaços conceituais.

⁶⁴ Tradução nossa de “A conceptual universe is thus a landscape in which we expect to find pronounced areas of low or negative complexity — ‘sponge zones’ as they will be termed — located in parts of the hierarchy which immediately dominate any set of concepts forming the constituents of an explanatorily useful conceptual framework. By accessing concepts in these zones, the agent moves into representational profit, reaping a representational reward that exceeds the cost of development.”

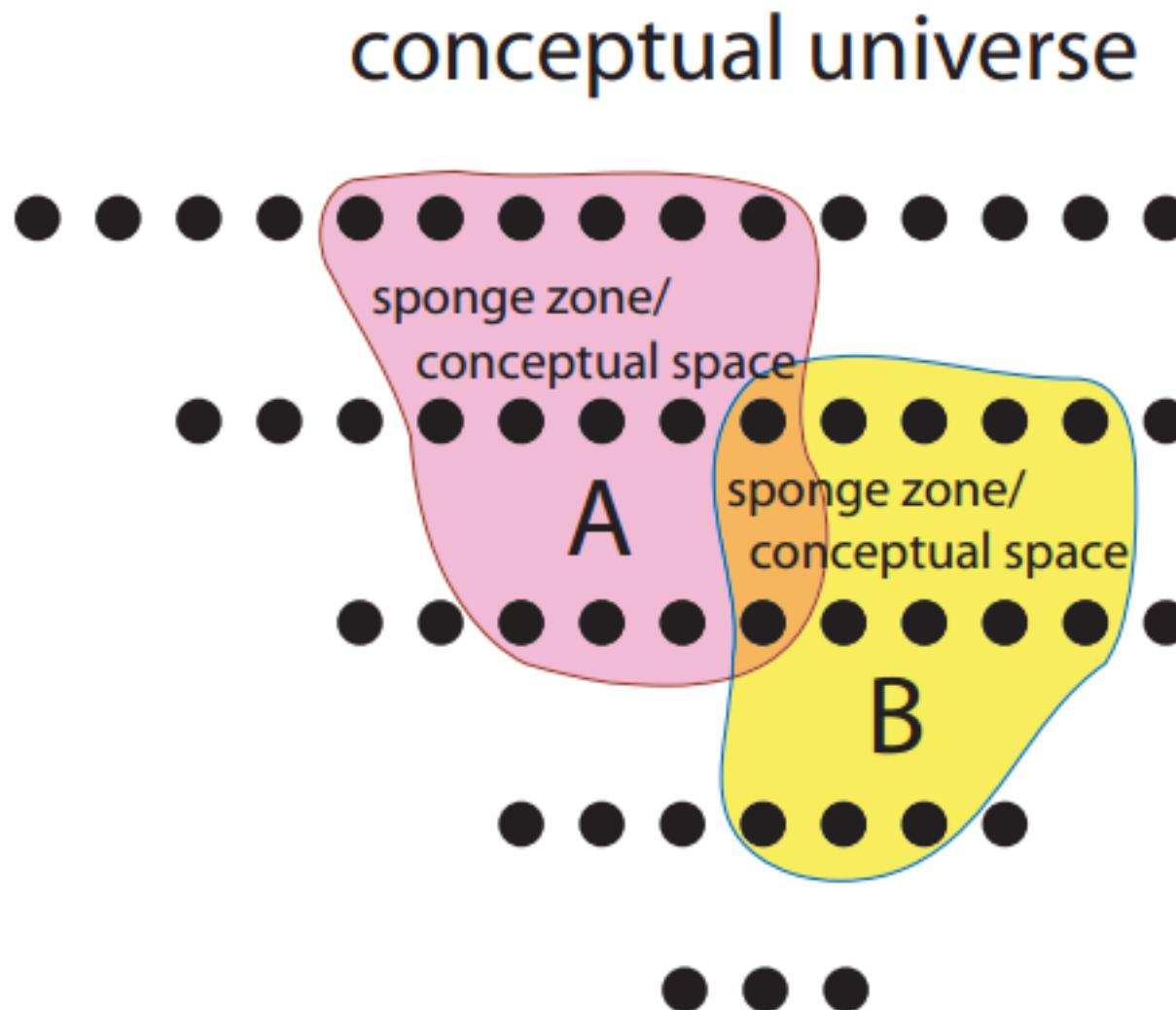


Figura 21 – Zonas esponjosas **Fonte:** ([Thornton, 2007](#))

Exemplo 3.3 (Representação formal da Figura 16)

$$\mathcal{C}_{Study \ in \ Keith} = \mathcal{C}_{live \ coding} \cap \mathcal{C}_{Sun \ Bears} \quad (3.2)$$

Este grupo também pode ser descrito como uma lista de propriedades como na [Equação 3.3](#):

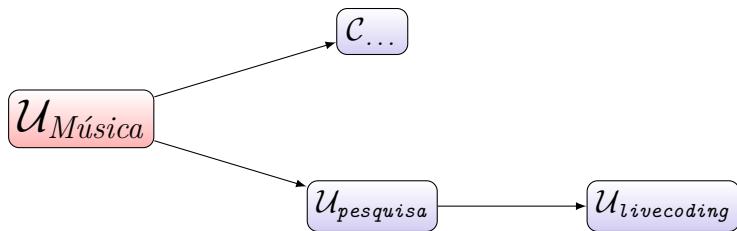
Exemplo 3.4 (Representação formal das propriedades da Figura 16)

$$\mathcal{C}_{SK} = [\mathcal{E}'_{SK}, \mathcal{K}'_{SK}, \mathcal{I}'_{SK}, \mathcal{R}'_{SK}, \mathcal{G}'_{SK}, \mathcal{M}'_{SK}, \mathcal{O}'_{SK}, \mathcal{F}'_{SK}, \mathcal{P}'_{SK}] \quad (3.3)$$

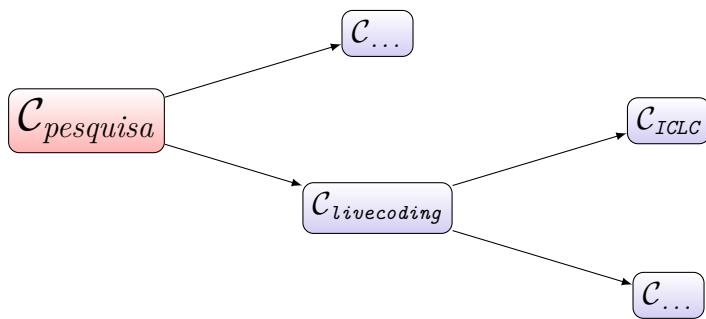
Nos diagramas abaixo, $C_{...}$ representa qualquer espaço conceitual abstrato (que pode incluir outro previamente apresentado). Entre os elementos iniciais (raízes, vermelho) e transitórios (nós, azul), ocorrem as ramificações (ramos, linhas pretas), isto é, a exploração de conceitos dentro de outros conceitos. De um lado, a aplicação de regras de validação sobre o universo conceitual da pesquisa (tudo aquilo que foi produzido em dois anos de mestrado) gerou o espaço conceitual desta tese. Estas regras de validação foram, em sua maior parte, os processos de orientação e qualificação. Em outras palavras, $\mathcal{C}_{pesquisa} = [[\mathcal{R}_{pesquisa}]](\mathcal{U}_{pesquisa})$.

Exemplo 3.5 (Representação do universo conceitual da *pesquisa*)

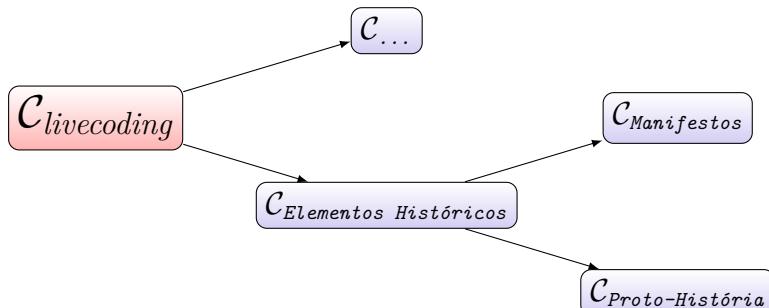
O Universo de Conceitos da pesquisa, $\mathcal{U}_{pesquisa}$, é um recorte do universo conceitual da música, $\mathcal{U}_{música}$:



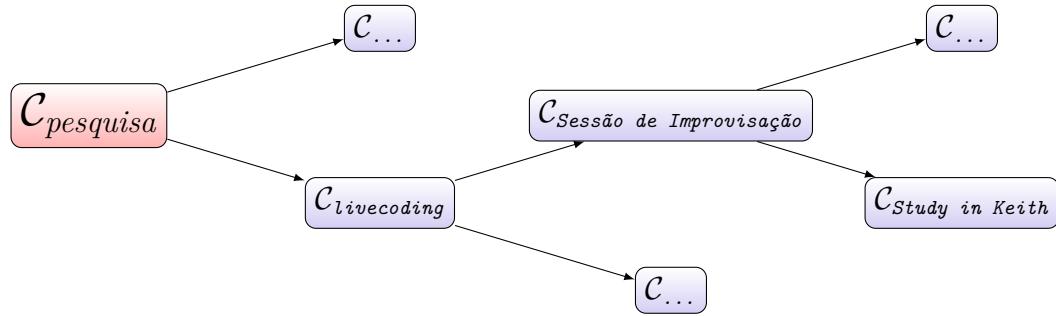
No primeiro capítulo, incluímos um subconjunto neste Espaço Conceitual da Pesquisa (ver [Apêndice A](#), p. 107)).



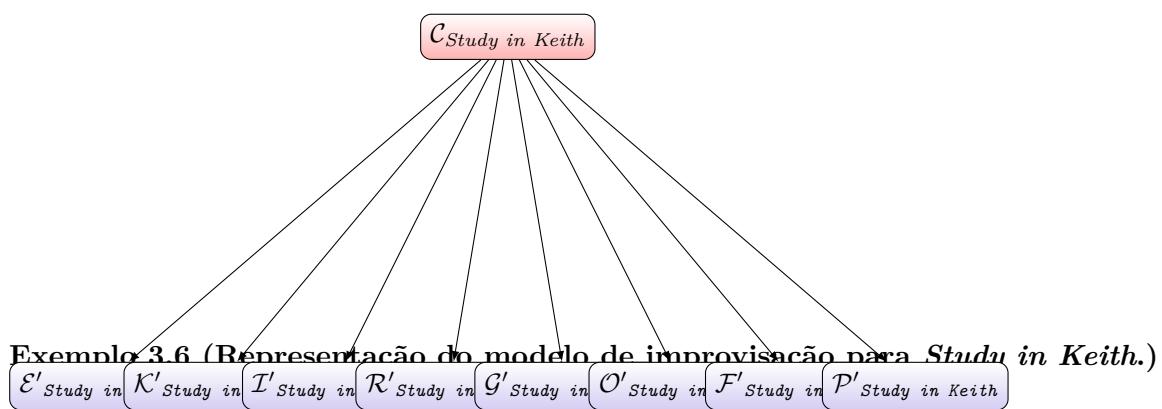
Podemos incluir elementos históricos, o período transitório entre 1970 e 2000 (*circa*), quando emanciparam as práticas e as regras heurísticas.



Por último, $\mathcal{C}_{pesquisa}$ investiga o *live coding* a partir de um caso específico:



Por outro lado $C_{Study \ in \ Keith}$ pode ser definido pelo modelo de improvisação de Pressing (Tabela 4, 61).



3.4 Formalização

O espaço conceitual do *livecoding* é definido como uma função de interpretação das regras de validação (o que pode ser ou não considerado como próprio de uma categorização musical), de gosto (questões de estilo) e de localização transversal de conceitos (conceitos internos que permitem o cruzamento com outros conceitos). As regras de validação foram estudadas neste trabalho como as regras heurísticas do *live coding*. Isto é, que conjunto de métodos são utilizados para caracterizar uma performance de *live coding* como tal? Elementos históricos, e ideológicos (divulgados em manifestos), são levantados para responder esta pergunta. Por outro lado, este estudo abandonou a investigação das regras de gosto, tema que pode ser melhor explorado em trabalhos posteriores, a partir de Janotti Jr. (2003), Sá (2006), Sá (2009). A tarefa de localização transversal de conceitos é trabalhada no último capítulo. O espaço conceitual de *Study in Keith* está contido no espaço conceitual do *live coding* através da intersecção entre os conceitos deste último, com os espaços conceituais dos concertos *Sun Bears*, de Keith Jarret, misturados. No entanto o espaço conceitual não será investigado em sua totalidade, e sim apenas uma sonoridade.

4 Estudo de caso

Dentro de um espaço conceitual da improvisação de códigos, selecionamos um exemplo específico, *A Study in Keith* (SORENSEN; SWIFT, 2009; SORENSEN, 2015), para ilustrar o intérprete concertista de jazz, readequado para os propósitos do programador:

“Para fundamentar a discussão em música, considere uma peça de jazz, onde jazz é um conceito e uma composição particular é uma instância de um conceito. O musicista, explorando os limites do jazz, encontra então uma peça para além das regras usuais do jazz. Através deste processo, os limites do gênero musical podem ser redefinidos em algum grau, ou se a peça está em um novo terreno particularmente fértil, um novo sub-gênero de jazz emerge. Contudo uma peça de música que não quebra limites, de alguma forma pode ser considerada não-criativa (MCLEAN, 2011, p. 117).”¹

São dois os registros audiovisuais principais, mas com duas descrições de um espaço conceitual \mathcal{E}_{ask} , cuja semelhança é um referente opcional \mathcal{R}_{ask}^0 , ou os Concertos *Sun Bear* (ver seção 4.3, p. 72):

“*A Study in Keith* é uma performance de programação ao vivo por Andrew Sorensen, inspirado nos concertos *Sun Bear* de Keith Jarret. Toda a música que você ouve é gerada a partir do código do programa que é escrito e manipulado em *tempo-real* durante a performance. O trabalho foi executado usando o ambiente de desenvolvimento [em linguagem] Scheme [chamado] Impromptu (<<http://impromptu.moso.com.au>>). Não é Keith, mas inspirado por Keith (SORENSEN, 2015). ”²

Sorensen (2015) indica outros referentes, \mathcal{R}_{ask}^1 como o ambiente de programação *Impromptu* e \mathcal{R}_{ask}^2 a linguagem de programação *Scheme* (ver subseção 4.3.3, p. 76):

“*A Study In Keith* é um trabalho para piano solo (NI’s Akoustik Piano), inspirado nos concertos *Sun Bear* de Keith Jarrett. Note que não existe som para os dois primeiros 2 minutos da performance, enquanto estruturas iniciais são construídas. Não é bem Keith, mas inspirado por Keith (SORENSEN; SWIFT, 2009)”³

¹ Tradução nossa de “ *To ground the discussion in music, consider a piece of jazz, where jazz is the concept and the particular composition is an instance of that concept. The musician, in exploring the boundaries of jazz, then finds a piece beyond the usual rules of jazz. Through this process, the boundaries of a music genre may be redefined to some degree, or if the piece is in particularly fertile new ground, a new sub-genre of jazz may emerge. Indeed a piece of music which does not break boundaries in some way could be considered uncreative.*”

² Tradução nossa de “ *“A Study In Keith” is a live programming performance by Andrew Sorensen inspired by Keith Jarrett’s Sun Bear concerts. All of the music you hear is generated from the program code that is written and mani[p]ulated in real-time during the performance. The work was performed using the Impromptu Scheme software development environment (<<http://impromptu.moso.com.au>>). Not Keith, but inspired by Keith.*”

³ Tradução nossa de “ *“A Study In Keith” is a work for solo piano (NI’s Akoustik Piano) by Andrew Sorensen inspired by Keith Jarrett’s Sun Bear concerts. Note that there is no sound for the first 2 minutes of the performance while initial structures are built. Not quite Keith, but inspired by Keith.”*

4.1 Objetivo

Observação e análise de um comportamento criativo musical, de um improvisador-programador, que escreve uma programação-partitura, e realiza a manutenção de um pensamento musical tradicional. Mais especificamente, analisamos o contexto musical de uma simples sequência de blocos sonoros \mathcal{K}_{ask}^0 , gerada por uma função de interpretação $<<\mathcal{R}_{ask}, \mathcal{T}_{ask}, \mathcal{E}_{ask}>>$, cujo referente opcional direto, \mathcal{R}_{ask}^0 , são os Concertos *Sun Bear* de Keith Jarret.

4.2 Justificativa

Parafraseamos McLean (2011, p. 121) ao afirmarmos que “Nosso estudo de caso é de alguma forma simplista e não é intenção ilustrar uma grande arte ou um grande código. Contudo delineia um processo criativo de classes, como efetuado pelo presente autor.”⁴.

4.3 Referentes Opcionais

A estratégia possui um referente opcional principal, \mathcal{R}_{ask}^0 , ou os Concertos *Sun Bear* de Keith Jarret (ver subseção 4.3.1, p. 72), cuja análise possibilitou compreender simples estratégias harmônicas como regras de qualidade aplicadas em *A Study in Keith*. Em seguida tratamos do timbre de piano utilizado \mathcal{R}_{ask}^1 , um plugin VSTi (ver subseção 4.3.2, p. 75), manipulado com um terceiro referente opcional, \mathcal{R}_{ask}^2 , ou o ambiente de programação musical chamado *Impromptu* (ver subseção 4.3.3, p. 76).

4.3.1 Concertos Sun Bear

Os concertos *Sun Bear* são originalmente dez LPs de improvisações de Keith Jarret no Japão, produzidos pela *ECM Records*⁵ entre 1976 e 1978. É o terceiro dos concertos de improvisação que incluem o *Solo Concerts: Bremen/Lausanne* (1973) e *The Köln Concert* (1975).

Foram realizados e gravados como sessões de improvisação contínua, variando entre 31 a 43 minutos cada. Para cada dia, duas sessões de improvisação, em cidades diferentes. Kyoto, 5 de novembro⁶; Osaka, 8 de novembro⁷; Nagoya, 12 de novembro⁸. Tokyo, 14 de novembro⁹; Sapporo, 18 de Novembro¹⁰.

Um documento crítico impresso é mencionado na *internet* como um antigo documento contendo notas discográficas (SWENSON, 1985). Seu acesso foi restrito durante a pesquisa, e

⁴ Tradução nossa de “Our case study is somewhat simplistic, and is not intended to illustrate either great art or great code. However it does trace a creative process of sorts, as carried out by the present author.”

⁵ <http://www.ecmrecords.com/>

⁶ Disponível em <<https://www.youtube.com/watch?v=T2TfIQNxhjc>>.

⁷ Disponível em <<https://www.youtube.com/watch?v=FC4iZ1wMoU8>>

⁸ <<https://www.youtube.com/watch?v=3a7ezm3D1jA>>.

⁹ Disponível em <<https://www.youtube.com/watch?v=ZH8VIjjhPQ4>>

¹⁰ Disponível em <https://www.youtube.com/watch?v=BqYBT_HoG4M>

não foi possível incluir alguma citação. Da mesma forma, documentos analíticos sobre a peça não foram encontrados em alguma base de dados. Existem algumas notas discográficas compiladas por uma comunidade de fãs e críticos musicais estadounidenses. Duas notas sugerem uma descrição da forma musical aplicada por Keith Jarret: “O tema de *Kyoto Parte 1* é repetido por Keith Jarret no fim de *Kyoto Parte 2*. Então podemos considerar o todo deste concerto como uma grande Súite.”¹¹ (GARBOLINO, 2014, p. 129).

Revisto por Richard S. Ginnel¹²: [–] Este pacote gigantesco – um conjunto de dez LPs agora comprimidos em uma caixa robusta de seis [embalagens de] CDs – foi ridicularizado uma vez como uma última viagem de ego, provavelmente por muitos que não tomaram um tempo para ouvir tudo. (...) Ainda assim, o milagre é como esta caixa é consistentemente muito boa. Na abertura de *Kyoto*, a meditação direcionada para o *gospel* está em plena atuação, ao nível de suas melhores performances solo em Bremen e Koln, e os concertos *Osaka* e *Nagoya* possuem citações de primeira linha, geralmente do tipo *folk*, mesmo profundas, idéias líricas (GARBOLINO, 2014, p. 130)¹³.

O *gospel* e o *folk* são citados como gêneros musicais inclusivos nesta súite. Porém esta Súite não possui pausas entre as partes (o improviso é contínuo, mas seccionado por transições). Uma transcrição do motivo gerador deste *gospel* no concerto de Kyoto buscou encontrar referentes opcionais adicionais para \mathcal{R}'_{ask} (isto é, informações de harmonia e ritmo), mesmo com a afirmação anterior que nenhuma relação pode ser encontrada (ver Figura 22, p. 74).

Uma sequência do blocos de eventos, \mathcal{K}_{KJ}^0 apresenta três blocos de eventos $[\mathcal{E}_{KJ}^0 \dots \mathcal{E}_{KJ}^2]$. Uma figura sincopada, que alterna uma nona menor, terça menor, segunda maior, e oitava, pode ser serparada como dois objetos \mathcal{O}_{KJ}^0 e \mathcal{O}_{KJ}^1 : um ostinato na mão esquerda que preenche todo \mathcal{K}_{ask}^0 , e uma sequência repetida seis vezes que alterna a terça menor e segunda maior. Nos compassos 3 a 5, ou o bloco de eventos \mathcal{E}_{KJ}^1 , aparecem mais dois objetos que formam uma relação harmônica de tritono com o baixo, um intervalo de quarta justa \mathcal{O}_{KJ}^2 , e um ostinato de terças maiores \mathcal{O}_{KJ}^3 . O acorde de Sol bemol Maior (transcrito assim para facilitar a leitura), é expandido nos compassos 6 a 10, ou \mathcal{E}_{KJ}^2 , gerando uma figura cromática proto-melódica, cuja transcrição apresenta a seguinte cadência prototípica: Sol Bemol Maior (com décima primeira aumentada adicionada, em terceira inversão), Fá Maior com sexta adicionada (ou Ré menor com sétima, em terceira inversão) e Dó Maior com sétima menor (posição fundamental). Limitamo-nos a considerar a progressão do ponto de vista do *blues*, se aproximando do *gospel* através de uma exploração da subdominante e da cadência plagal.

¹¹ Tradução nossa de “The theme of *Kyoto Part 1* is repeated By Kj at the end of *Kyoto Part 2*. So we can consider the whole of this concert as one big Suite”

¹² Disponível em <<http://www.mcana.org/formembersatlarge.html>>.

¹³ Tradução de Review by Richard S. Ginell: [–] This gargantuan package – a ten-LP set now compressed into a chunky six-CD box – once was derided as the ultimate ego trip, probably by many who didn’t take the time to hear it all. You have to go back to Art Tatum’s solo records for Norman Granz in the ’50s to find another large single outpouring of solo jazz piano like this, all of it improvised on the wing before five Japanese audiences in Kyoto, Osaka, Nagoya, Tokyo, and Sapporo. Yet the miracle is how consistently good much of this giant box is. In the opening Kyoto concert, Jarrett’s gospel-driven muse is in full play, up to the level of his peak solo performances in Bremen and Koln, and the Osaka and Nagoya concerts have pockets of first-rate, often folk-like, even profound, lyrical ideas.

Figura 22 – Transcrição do motivo gerador do disco Kyoto, parte 1. **Fonte:** autor.

Tomando um *blues* tradicional de 12 compassos, seguimos uma fórmula prática $C : I^7 \Rightarrow IV^7 \Rightarrow I^7 \Rightarrow I^7 \Rightarrow IV^7 \Rightarrow IV^7 \Rightarrow I^7 \Rightarrow I^7 \Rightarrow V^7 \Rightarrow IV^7 \Rightarrow I^7$. É possível explorar “seção plagal” do padrão, ao separarmos os acordes 4 a 7 ($C : I^7 \Rightarrow IV^7 \Rightarrow IV^7 \Rightarrow I^7$), e transformarmos, por substituição de tritono, a primeira subdominante da sequência, ou $C : subV/IV$ ($subV/IV$ em um *blues* em Dó Maior). Isto é, a substituição-padrão, $C : bV/V^7 \Rightarrow I^7$, ou $C : bII^7 \Rightarrow I^7$, que seria Ré bemol Maior com sétima, seguido de Dó maior com sétima, passa a ser operacionalizada como um contra-polo à tônica ¹⁴. O que pode ser notado como $C : (bV^7/V)/IV \Rightarrow IV^7$, ou $C : bII^7/IV \Rightarrow IV^7$ (sequência do segundo grau bemol da subdominante para a subdominante do tom), pode ser simplificado como $C : bV^7 \Rightarrow IV^7$, ou uma sequência do quinto grau bemol para o quarto grau. No entanto, a transcrição de Jarret (considerando sua futura correção), suprime e transforma as sétimas, o que caracteriza uma sonoridade de tensão progressiva (e ambígua) com um baixo pedal. Uma tríade do quinto grau bemol, uma tétrade do quarto grau com sexta e quinta (com sua sétima no baixo), e primeiro grau com sétima: $C : bV \Rightarrow IV_5^6 \Rightarrow I^7$. O baixo pedal pode sugerir um 11º aumentado no primeiro acorde da sequência ($C : 11+bV$ ou $C : bV^{b2}$) e, no segundo acorde da sequência, um segundo grau menor invertido (ii^2). Uma notação sintética desta sequência sugere um padrão muito parecido, permutado e transformado da sequência-padrão ii-V-I: $C : bV^{b2} \Rightarrow ii^2 \Rightarrow I^7$.

Uma pequeno esquema schenkeriano pode ilustrar uma situação curiosa (ver exemplo 4.1). Subdividimos dois grupos melódicos, $\hat{b}2-\hat{1}-\hat{7}$, inicialmente em uma forma permutada, $\hat{7}-\hat{b}2-\hat{1}$,

¹⁴ Cf. SOARES, 2015-03-13.

como uma figura alternada com um ostinato no baixo, cuja primeira sonoridade, não possui uma função harmônica *a priori*. Em seguida, apresentamos grupo $\hat{b}5-\hat{4}-\hat{3}$.

Exemplo 4.1 (Redução para a primeira sonoridade dos concertos *Sun Bear*)

Entre estes dois grupos, existe um pequeno ostinato de terças maiores e menores, e que a segunda instância do grupo $\hat{b}2-\hat{1}-\hat{7}$ possui uma pequena alteração $\hat{b}2-\hat{1}-\hat{2}-\hat{3}$.

Os esquemas acima apresentam uma questão: Jarret improvisou este início, ou preparou esta pequena célula, como estímulo para o restante do disco? Na página anterior, uma nota crítica lembra que este tema é recuperado no final da segunda parte do disco, o que reforça o argumento de uma estrutura pré-definida. Em *Study in Keith* a mesma pergunta permanece: o código elaborado por Sorensen e Swift (2009), e sua sonoridade resultante, é de fato uma improvisação de códigos, ou existe um agenciamento onde o improvisador delineia um objetivo inicial?

4.3.2 NI-Akoustik Piano

O NI, é uma abreviação para *Native Instruments*, uma empresa de tecnologias para áudio¹⁵. O *Akoustic Piano* é uma extensão (*plugin*) VST, que emula diferentes pianos acústicos. Os instrumentos são gravados nota a nota por um complexo sistema de tomada de som, amostrados digitalmente, para então ser possível utilizar os registros como eventos MIDI¹⁶

Existe um objetivo artístico, de Sorensen e Swift, em capacitar a improvisação de códigos com o piano. Isso não quer dizer que é a máquina que improvisa, mas sim como um humano que improvisa com um mecanismo que gera detalhes dinâmicos, mas cuja estrutura é pré-definida. E *A Study in Keith* pode ser observado como uma simulação para um trabalho posterior com o *Disklavier* da Yamaha (ver Figura 23, p. 76). Este último caso não é citado por Sorensen (2015) e Sorensen e Swift (2009), mas em *Disklavier Sessions* (SORENSEN, 2013), este mesmo um outro referencial \mathcal{R}'_{ask}^1 , é semelhante à atividade descrita em *A Study in Keith*:

¹⁵ Disponível em <<http://www.native-instruments.com/en/company/>>.

¹⁶ Disponível em <<http://www.native-instruments.com/en/products/komplete/keys/definitive-piano-collection/>>



Figura 23 – Piano Disklavier de armário, com a parte interna exposta para exibir a placa-mãe. **Fonte:** wikimedia.org

“Em *Disklavier Sessions* os programas escritos em tempo-real por Ben e Andrew geram um fluxo de dados de notas que é enviado para ser executado em um piano disklavier mecanizado. Assim como as alturas das notas, toda a performance do piano deve ser codificada na informação gerada pelo programa e enviada para o piano disklavier.”¹⁷

Adiante daremos alguns detalhes de como Sorensen e Swift geram este fluxo de dados, que serão convertidos em ações do martelo do piano (ver [seção 4.4](#), p. 84). No momento, podemos dizer que são programados, em tempo-real, em um *software/Ambiente de programação* nomeado como *Impromptu*, cuja base de desenvolvimento é o *Exttempore*.

4.3.3 Ambiente e Linguagem: Impromptu

“Impromptu é uma linguagem e um ambiente de programação OSX¹⁸ para compositores, artistas sonoros, VJ’s e artistas gráficos com um

¹⁷ Tradução nossa de “In the Disklavier Sessions the programs beign written in real-time by Ben and Andrew are generating a live stream of note data which is sent to a mechanized disklavier piano to be performed. As well the individual note pitches all of the piano performance must be encoded into the information being generated by the program and sent to disklavier piano”

¹⁸ Sistema Operacional Mac OSX.

interesse em programação ao vivo ou interativa. Impromptu é um ambiente de linguagem Scheme, um membro da família das languages Lisp. Impromptu é usado por artistas-programadores em performances de *livecoding* em torno do mundo.¹⁹

Segundo Sorensen e Gardner (2010, p. 823), o Impromptu é um ambiente de programação ciberfísico, análogo à *partitura* tradicional. O ambiente suporta a compilação de pequenos trechos de códigos executáveis em linguagem *Scheme* (\mathcal{L}_{ask}). Nos termos de Magnusson (2011), os algoritmos codificados nesta linguagem são instrumento. Nos termos de Fenerich, Obici e Schiavoni (2014, p. 5), o código é uma programação-partitura:

“Considere a analogia da partitura musical tradicional. A partitura provê uma especificação estática da intenção – um programa de domínio estático. Musicistas, representam o domínio do processo, executam ações requeridas para realizar ou reificar a partitura. Finalmente, as ações no domínio do processo resultam em ondas sonoras que são percebidas por uma audiência humana como música. Este estágio final é o nosso domínio real de trabalho. Agora considere um domínio de programação dinâmica no qual o compositor concebe e descreve uma partitura em *tempo-real*. Nós geralmente chamamos este tipo de composição de improvisação. Na improvisação o(a) musicista é envolvido em um circuito-fechado retroalimentado que envolve premeditação, movendo para ação casual e finalmente para reação, refinamento e reflexão.”²⁰

Existe uma restrição quanto ao nicho de usuários do *software*, com suporte para usuários de computadores Apple. Para lidar com outros sistemas (como por exemplo, sistemas operacionais Linux) e arquiteturas de processamento (32bit e 64 bit), o projeto foi liberado como código-aberto, com o nome *Extempore*.

4.3.4 Extempore

O *Extempore* possui um sistema humano-máquina reflexivo (ver seção 2.1, p. 27). Um nome específico, para a atividade de programar, escutar o resultado, e recodificar, é simbolicamente chamado de *programação ciberfísica*:

“*Extempore* é projetado para suportar um estilo de programação apelidado de “[”]programação ciberfísica”. Programação ciberfísica suporta

¹⁹ Tradução nossa de “ *Impromptu is an OSX programming language and environment for composers, sound artists, VJ's and graphic artists with an interest in live or interactive programming. Impromptu is a Scheme language environment, a member of the Lisp family of languages. Impromptu is used by artist-programmers in livecoding performances around the globe.* Disponível em <<http://impromptu.moso.com.au/>>”

²⁰ Tradução nossa de “ *Consider the analogy of a traditional musical score. The score provides a static specification of intention – a static program domain. Musicians, representing the process domain, perform the actions required to realise or reify the score. Finally, the actions in the process domain result in sound waves which are perceived by a human audience as music. This final stage is our real-world task domain. Now consider a dynamic program domain in which a composer conceives of and describes a musical score in real-time. We commonly call this type of composition improvisation. In it, the improvising musician is involved in a feedback loop involving forethought, moving to causal action and finally to reaction, refinement and reflection.*”

a noção de um programador humano operando como um agente ativo em uma rede de *tempo-real* distribuída de sistemas ambientalmente conscientes”²¹

Entre suas características de interesse musical, incluem²³:

- Processamento de Sinais Digitais (DSP)²⁴ em tempo-real;
- Sequenciamento de áudio de alto-nível, baseado em notas, como o disparo de sons baseado em parâmetros como altura, intensidade e duração.²⁵ .;

A segunda característica será explorada neste capítulo como base técnica para o processo criativo em *Study in Keith*

4.3.5 Scheme

Scheme é citado em diferentes fontes na *internet* como uma definição de linguagem, ou dialeto, da linguagem Lisp (criado por John McCarthy em 1958), criado por Guy L. Steele e Gerald Jay Sussman em 1975. Uma das características da linguagem LISP é o tipo de representação de um código, ou seu padrão de notação, baseado em uma gramática generativa:

“Um sistema chamado LISP (para Processador de LISta) foi desenvolvido para um computador IBM 704 pelo grupo de Inteligência Artificial no M.I.T. O sistema foi projetado para facilitar experimentos com um sistema proposto chamado “Recebedor de conselhos” [Advice Taker], onde uma máquina pode ser instruída para lidar com sentenças declarativas, bem como imperativas, e poderia exibir um “senso comum” no desempenho de suas instruções. A proposta original para o *Advice Taker* foi feita em novembro de 1958. O principal requerimento foi um sistema de programação para manipular expressões que representam sentenças formais, declarativas e imperativas, de modo que o sistema *Advice Taker* pode fazer deduções. No curso do desenvolvimento, o sistema LISP passou por diversas simplificações e, eventualmente, se baseou em um esquema para representar funções recursivas parciais de certas classes de expressões simbólicas. Esta representação é independente do computador IBM 704, ou qualquer outro computador eletrônico, e agora parece útil expor o sistema, começando com a classe de expressões chamadas expressões-S e as chamadas funções-S (McCarthy, 1960, seção 1).”²⁶

²¹ Tradução nossa de “*Extempore is designed to support a style of programming dubbed ‘cyberphysical’ programming. Cyberphysical programming supports the notion of a human programmer operating as an active agent in a real-time distributed network of environmentally aware systems.*”²² .”

²³ Disponível em <<http://benswift.me/2012/08/07/extempore-philosophy/>>

²⁴ Sobre DSP, Cf. SMITH, 2012-06.

²⁵ Disponível em <<http://benswift.me/2012/10/15/playing-an-instrument-part-i/>>

²⁶ Tradução nossa de “A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit “common sense” in carrying out its instructions. The original proposal [1] for the Advice Taker was made in November 1958. The main requirement was a programming system for manipulating expressions representing formalized declarative and imperative sentences so that the Advice Taker system could make deductions. In the course of its development the LISP system went through several stages of simplification and eventually came to be based on a scheme for representing the partial recursive functions of a certain class of

A definição de funções-S foge do escopo de nossa pesquisa, mas ela pode ser compreendida de maneira intuitiva, a partir das expressões-S. McCarthy (1960, seção 3) define expressões-S como “átomos” e listas de átomos, onde um átomo também pode ser uma lista de átomos. Existe uma classe de expressões simbólicas definida por parênteses. Dentro desta expressão simbólica são inseridos átomos (ver exemplo 4.2, 4.3 e 4.4).

Exemplo 4.2 (Expressão simbólica vazia)

```
( )
```

Átomos, extraídos de sentenças abstratas, como listas de átomos:

Exemplo 4.3 (Expressão simbólica com átomos)

```
; ;A      ->
( A )

; ;AB     ->
( A B )

; ;ABA    ->
( A B A )

; ;ABAC   ->
( A B A C )
( A B A C A )
```

Átomos também podem ser eles mesmos outras expressões:

Exemplo 4.4 (Expressão simbólica com átomos)

```
; ;A = A
; ;B = AB
; ;C = BAB

;; ABA ->
( A ( A B ) A )
;; ABAC ->
( A ( A B ) A ( B A B ) )
( A ( A B ) A (( A B ) A ( A B )))

; ;A = 1
; ;B = +
; ;C = 2
```

symbolic expressions. This representation is independent of the IBM 704 computer, or of any other electronic computer, and it now seems expedient to expound the system by starting with the class of expressions called S-expressions and the functions called S-functions.”

```
;; ABA ->
( 1 + 1 ) ;; = 2

;; ABAC ->
( 1 + 1 2 ) ;; = 4
```

A linguagem LISP implementa um tipo de notação chamada *notação prefixada*, inventada por Jan Łukasiewicz em 1924, para simplificar a lógica proposicional ($P \rightarrow Q$). De fato, neste ponto, explicitamos uma característica da linguagem \mathcal{L}_{ask} . Esta simplificação da notação pode, por exemplo, expressar uma proposição como “some uma unidade e uma unidade, logo teremos duas unidades”, ou $1 + 1 = 2$, para uma operação indefinida, como “some elementos, que neste caso, são dois unitários, logo duas unidades”, ou $+11$:

Exemplo 4.5 (Notação prefixada)

```
;;A = 1
;;B = +
;;C = 2
;; ABA -> BAA
( + 1 1 ) ;; = 2

;;ABAC -> BAAC ->
( + 1 1 2 ) ;; = 4
```

Entre as diferenças do Lisp e Scheme, existe um vocabulário prédefinido para criação de sentenças, de forma que o significado do código possa ser legível, sem a necessidade de consideração de contextos. Para os propósitos deste trabalho definimos as variáveis e funções:

Exemplo 4.6 (Notação Scheme)

```
;; define A = 1
(define A 1)

;; define B = 2
(define B 2)

;; divisao na forma (lambda argumentos operacao)
(define divide ;; define nome da funcao
  (lambda (a b) ;; argumentos da funcao (calcula lambda)
    (/ a b)) ;; o que faz a funcao
)

;; execucao descriptiva
(divide A B)
```

Para os propósitos deste trabalho, será útil exemplificar musicalmente. Sorensen e Gardner (2010, p. 823-824) apresenta um pseudo-código musical, mais especificamente, direcionado para uma sonoridade de *jazz tonal*.

Exemplo 4.7 (Exemplo musical para o Scheme)

Este exemplo é semelhante com o primeiro algoritmo gerador de sonoridades tonais em *A Study in Keith* (ver seção 4.4, p. 84).

“ Dois performers se apresentam no palco. Um violinista, em pé e parado, com seu arco preparado. Outro senta-se atrás do brilho da tela do *laptop*. Uma projeção da tela do *laptop* é projetada acima do palco, e mostra uma página em branco, com um simples cursor piscando. O musicista-programador começa a digitar ... ”²⁷

```
( play-sound ( now ) synth c3 soft minute)
```

“ ... a expressão é avaliada, e lampeja no retroprojector para exibir a ação do executante. Um som etéreo sintetizado entra imediatamente no espaço e o violinista começa a improvisar em simpatia com a novidade da textura. O músico-programador, ouve o material temático fornecido pelo violinista e começa a delinear um processo generativo Markoviano para acompanhar o violino: ”²⁸

```
( define chords
  ( lambda ( beat chord duration )
    ( for-each ( lambda ( pitch )
      ( play synthj pitch soft duration ))
      chord )
    ( schedule (* metro * ( + beat duration )) chords
      (+ beat duration )
      ( random ( assoc chord (( Cmin7 Dmin7 )
        ( Dmin7 Cmin7 )))))
      duration )))

( chords (* metro * get-beat 4) Cmin7 4)
```

“... A função *chords* é chamada no primeiro tempo de um nova barra de tempo, e uma simples progressão recursiva de acordes come a suportar a performance melódica do violino. A função *chords* cria um laço temporal, gerando uma sequência interminável de acordes de quatro tempos. Depois de poucos momentos de reflexão, o musicista-programador começa a modificar a função *chords* para suportar uma progressão de acordes mais variada, com uma razão aleatória [em função] da recursão temporal... ”²⁹

```
( define chords
  ( lambda ( beat chord duration )
```

```
( for-each ( lambda ( pitch )
    ( play dls (+ 60 pitch) soft duration ))
    chord )
( schedule (* metro * ( + beat duration )) chords
  (+ beat duration )
  ( random ( assoc chord (( Cmin7 Dmin7 Bbmaj )
    ( Bbmaj Cmin7 )
    ( Dmin7 Cmin7 ))))
  ( random (3 6))))))
( chords (* metro * get-beat 4) Cmin7 4)
```

Este pseudo-código será discutido, na próxima seção, como a estratégia transversal, \mathcal{T}_{ask} . Propomos um particionamento do código para melhor compreensão:

Exemplo 4.8 (Nome da estratégia transversal)

`chords` é o nome da estratégia.

```
;; Definicao de acordes
( define chords
...
)
```

`chords` executado, como um impulso musical, com um único acorde com os seguintes parâmetros: momento de execução, grau e qualidade do acorde, duração do acorde:

Exemplo 4.9 (Estímulo inicial para a estratégia)

```
; Execucao da funcao
( chords (* metro * get-beat 4) Cmin7 4)
```

Adiante são definidas propriedades com termos do vocabulário da música tonal, ou, coloquialmente, batida (no sentido da posição de uma unidade de tempo em um pulso, *tactus*), acorde (tríades, tétrade, formadas por relações de intervalos de terças maiores e menores), e duração (o quanto, em relação à unidade de tempo, este acorde irá durar):

Exemplo 4.10 (O que operacionaliza a estratégia)

```
( ...
  ( lambda ( beat chord duration )
  ...
)
```

Existem duas estratégias internas na estratégia principal, cuja execução é realizada através de outras palavras-chaves. A palavra-chave **for-each** realiza um laço iterativo para cada altura do acorde:

Exemplo 4.11 (Laço iterativo para cada altura do acorde)

```
;; Primeira estrategia interna
;; Para cada acorde operacionalize cada altura
(for-each ( lambda ( pitch )
    ( play dls (+ 60 pitch) soft duration ))
    chord )
```

Para cada acorde **chord**, é tocada uma nota (**pitch**), com um centro em Dó 3 (MIDI 60), em piano (**soft**) e uma duração padrão (**duration**):

Exemplo 4.12 (Execução da nota)

```
( play dls (+ 60 pitch) soft duration ))
```

A palavra-chave **schedule** executa, recursivamente, um fluxo de acordes associados (**random (assoc chord)**), em resposta ao estímulo (**((chords (* metro * get-beat 4) Cmin7 4))**).

Exemplo 4.13 (Fluxo de novos acordes)

```
( schedule (* metro * ( + beat duration )) chords
    (+ beat duration )
    ( random ( assoc chord (( Cmin7 Dmin7 Bbmaj )
        ( Bbmaj Cmin7 )
        ( Dmin7 Cmin7 ))))
    ( random (3 6))))
```

O momento de execução deste acorde depende da execução do acorde anterior

Exemplo 4.14 (Quando novos acordes serão computados)

```
( schedule (* metro * ( + beat duration )) chords
    ...
)
```

Sendo que o acorde será executado logo em seguida que anterior terminar, com uma cadência harmônica escolhida dentre uma lista de cadências, com uma duração randômica entre três e seis unidades de tempo:

Exemplo 4.15 (Propriedades de novos acordes)

```
( schedule ... chords
  (+ beat duration )
  ( random ( ... ))
  ( random (3 6)))
```

A escolha de acordes é feita de maneira randômica, segundo uma lista de cadências predeterminados. Neste ponto, podemos indicar de maneira mais explícita uma regra de qualidade:

Exemplo 4.16 (Propriedades de novos acordes)

```
( random ( assoc chord (( Cmin7 Dmin7 Bbmaj )
  ( Bbmaj Cmin7 )
  ( Dmin7 Cmin7 ))))
```

Como definido pela função `chords`, o acorde será tocado em um momento que depende do cronograma, cuja duração pode variar de 3 a 6 unidades de tempo. No caso, é prototipado um fluxo recursivo de acordes. Por exemplo, podemos restringir uma sequência de cadências-modelo com estruturas que se afastam da tônica, e as que voltam para a tônica. A maneira por qual podem ser acessadas, pode variar, e é interessante que o computador escolha randomicamente as progressões desejadas pelo improvisador programador (ver Figura 24, p. 85).

No caso do bloco de código da explicação Sorensen e Gardner (2010, p. 823-824), são utilizadas os seguintes movimentos harmônicos: $I^7+ \Rightarrow ii^7 \Rightarrow IV^7/IV$, e $IV^7/IV \Rightarrow I^7$ e $ii^7 \Rightarrow I^7$.

Discussão

A partir de uma exploração destes referentes opcionais – $[\mathcal{R}_{ask0}, \mathcal{R}_{ask1}, \mathcal{R}_{ask2}]$ – foi possível classificar, além de uma linguagem \mathcal{L}'_ask^0 , o algoritmo gerador da uma sonoridade tonal em *Study in Keith*, ou \mathcal{T}_{ask} (ver subseção 4.3.5, p. 78). Uma análise do algoritmo permite verificar algumas regras de qualidade \mathcal{E}_{ask} . A interpretação $<<< \mathcal{R}_{ask}, \mathcal{T}_{ask}, \mathcal{E}_{ask} >>>$ produziu uma sequência de blocos de eventos \mathcal{K}'_{ask}^0 (ver seção 4.4, p. 84). Outros blocos também são produzidos, porém nossa análise busca investigar o espaço conceitual que possibilitou os primeiros resultados em um ciclo de bricolagem.

4.4 Blocos de Eventos

No capítulo anterior, definimos o espaço conceitual, nos termos dos Quadros Conceituais de Sistemas Criativos, como uma aplicação do espaço conceitual da improvisação de códigos, e provavelmente, com um estilo de *jazz*. Apresentamos algumas regras de validação em momentos

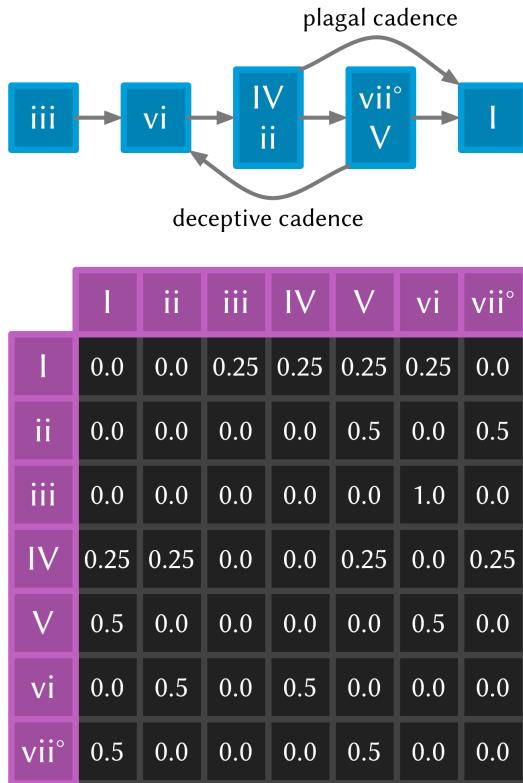


Figura 24 – Distribuição, aproximada, de probabilidades de acontecimento com um conjunto de possíveis cadências tonais organizados como uma cadeia de Markov.
Fonte: Swift (2012).

anteriores, da improvisação de códigos (ver subseção 2.4.1, p. 41), e de um referencial opcional (ver seção 4.3, p. 72).

Seguiremos com configuração da estratégia transversal de Sorensen, como regra de detecção \mathcal{T}_{ask} , que possui uma regra de qualidade \mathcal{E}_{ask} (ver subseção 4.4.1, p. 86). Este espaço conceitual gera uma sequência de blocos de eventos \mathcal{K}_{ask}^0 , muito semelhantes à neumas musicais básicos³⁰, em um contraponto de primeira espécie que sofre uma primeira transformação (ver subseção 4.4.1.2, p. 88).

Uma nota sobre esta improvisação é feita pelo próprio Sorensen: nos primeiros dois minutos do vídeo (aproximadamente 1'53''). Existe um silêncio característico do momento em que os primeiros códigos são escritos. Este comportamento, do tempo de codificação, ao tempo de ação musical, é similar em outros dois vídeos, de Sorensen: An evening of livecoding at 53 Rusden Street³¹, Just for Fun³², A Study in Part³³, Stained³⁴, Transmissions in Sound³⁵, An-

³⁰ Cf. GASPERINI, 1905

³¹ Disponível em <<https://vimeo.com/2433303>>

³² Disponível em <<https://vimeo.com/2433971>>

³³ Disponível em <<https://vimeo.com/2434054>>

³⁴ Disponível em <<https://vimeo.com/2502546>>

³⁵ Disponível em <[TransmissionsinSound](#)>

tiphony³⁶, Strange Places³⁷, Orchestral³⁸, UMDT³⁹, Day of Triffords⁴⁰, Face to Face⁴¹, BM&E⁴², A Christmas Carol⁴³ Dancing Phalanges⁴⁴, Livecoding Audio DSP⁴⁵, Jazz Ensemble Study⁴⁶, Variations on a Christmas Theme⁴⁷. Esta característica também foi observada em uma outra performance (ver subseção 1.3.2, p. 20). Isso não quer dizer que o silêncio é um ator musical, com distância em relação a uma proposta apresentada anteriormente (ver subseção 1.2.2, p. 12).

4.4.1 Definição do instrumento e do tempo

Seu início é um pequeno comentário que contém o nome do executante e seu email para contato (primeiros sete segundos), bem como a escrita de um código que inicializa o NI-Akoustik (até 0'43", ver Propriedades de novos acordes 4.17).

Exemplo 4.17 (Definição de instrumento)

Primeiros eventos musicais gerados a partir das primeiras estruturas válidas de código. **Fonte:** (SORENSEN, 2015).

```
;;;;;;;;;;;;;;;;;;
;; Andrew Sorensen andrew@moso.com.au
(define piano (au:make-node "aumu" "NaDd" "-NI-"))
(au:connect-node piano 0 *au:output-node* 0)
(au:update-graph)

(au:load-preset piano "/tmp/convert_grand.aupreset")
```

Em 0'52" Sorensen define um tempo base. Em seguida, Sorensen apaga o código para então iniciar definições de notas (0'54").

Exemplo 4.18 (Definição de tempo)

Definição do tempo base. **Fonte:** (SORENSEN, 2015).

```
(define *metro* (make-metro 110))
```

³⁶ Disponível em <<https://vimeo.com/2503188>>

³⁷ Disponível em <<https://vimeo.com/2503257>>

³⁸ Disponível em <<https://vimeo.com/2579694>>

³⁹ Disponível em <<https://vimeo.com/2579880>>

⁴⁰ Disponível em <<https://vimeo.com/2735394>>

⁴¹ Disponível em <<https://vimeo.com/5690854>>

⁴² Disponível em <<https://vimeo.com/7339135>>

⁴³ Disponível em <<https://vimeo.com/8364077>>

⁴⁴ Disponível em <<https://vimeo.com/8732631>>

⁴⁵ Disponível em <<https://vimeo.com/15585520>>

⁴⁶ Disponível em <<https://vimeo.com/15679078>>

⁴⁷ Disponível em <<https://vimeo.com/18008372>>

4.4.1.1 Definição de uma sequência de blocos

Até 1'07'', uma rotina auxiliar é definida como um laço iterativo. Porém não encontramos sua especificação no código-fonte do *Extempore*.

Exemplo 4.19 (Definição de uma função auxiliar)

```
(pc:cb-for-each-p chords piano
  (pc:make-chord 50 70 2 (pc:diatonic 0 '- degree))
  dur)
```

Internamente, existe uma rotina que será o cerne de execução de uma nota, acompanhada de uma lista de 4 parâmetros (50, 70, 2, dur):

Exemplo 4.20 (Definição de uma nota)

```
(pc:make-chord 50 70 2 (pc:diatonic 0 '- degree))
```

A abreviação pc significa *pitch class*, e a função pc:make-chord significa que a função cria um acorde segundo parâmetros definidos no código-fonte do *Extempore*⁴⁸:

“Cria uma lista do “número” [com] alturas entre limites “menor” e “maior” do pc dado. Uma divisão dos limites, pelo número de elementos requisitados, decompõem a seleção em extensões diferentes, do qual cada altura é selecionada. make-chord tenta selecionar alturas para todos os graus do pc. É possível, para os elementos de um acorde resultarem em -1, se não existir nenhum pc para a extensão dada. [É] não-determinístico (i.e., resultados variam com o tempo). Argumento 1: limite menor (inclusivo). Argumento 2: Limite maior (exclusivo). Argumento 3: Número de alturas no acorde. Argumento 4: pitch class (SWIFT, 2012).”⁴⁹

Este bloco de códigos cria uma diáde, no âmbito de um Ré 2 (MIDI 50) e Si bemol 3 (MIDI 70), dentro de um campo harmônico diatônico (pc:diatonic). Por sua vez, este último cria “um acorde seguindo regras básicas de harmonia diatônica: baseado em uma raiz (0 para C, etc.), maior/menor ('- ou '^) e graus (i-vii)”⁵⁰. O resultado não é previsível, e depende de regras específicas de qualidade, que apresentaremos adiante, para classificar os *pitch class* dentro de um grau de um campo harmônico.

⁴⁸ Disponível em <https://github.com/digego/extempore/blob/master/libs/core/pc_ivl.xtm>

⁴⁹ Tradução nossa de “Creates a list of “number” pitches between “lower” and “upper” bounds from the given “pc”. A division of the bounds by the number of elements requested breaks down the selection into equal ranges from which each pitch is selected. make-chord attempts to select pitches of all degrees of the pc. It is possible for elements of the returned chord to be -1 if no possible pc is available for the given range. Non-deterministic (i.e. result can vary each time). arg1: lower bound (inclusive). arg2: upper bound (exclusive). arg3: number of pitches in chord. arg4: pitch class”

⁵⁰ Tradução nossa de: (...) a chord following basic diatonic harmony rules: based on root (0 for C etc.) maj/min ('- or '^) and degree (i-vii).

4.4.1.2 Definição de blocos

Em 1'08'', a função *chords* surge no fluxo audiovisual, sem nenhum processo de escrita. Este comportamento caracteriza a utilização de, ou uma cópia/cola de texto, ou de uma execução de um macro do editor de texto usado. Macros são pequenos programas no editor que auxiliam o processo de produção do código. De qualquer forma é importante salientar que o código é preparado (SORENSEN, 2015).

Exemplo 4.21 (Algoritmo que define os acordes)

O algoritmo apresenta apenas uma propriedade, tempo (*time*).

```
(define chords
  (lambda (time)
    (for-each (lambda (p)
      (play-note (*metro* time) piano p 80 (*metro* 'dur dur)))
      (pc:make-chord 50 70 2 (pc:diatonic 0 (quote -) degree)))
    (callback (*metro* (+ time (* .5 dur))) chords (+ time dur)))))

(chords (*metro* 'get-beat 4.0) 'i 3.0)
```

Primeiro é definida a estratégia transversal, \mathcal{T}_{ask} , com um parâmetro, *time*

Exemplo 4.22 (Estratégia transversal)

```
(define chords
  (lambda (time) ...))
```

Seguido de um “impulso”, ou um estímulo sonoro:

Exemplo 4.23 (Impulso, ou acorde inicial)

```
(chords (*metro* 'get-beat 4.0) 'i 3.0)
```

Dentro de \mathcal{T}_{ask} , é executado um laço iterativo, *for-each*, para cada nota de uma diáde.

Exemplo 4.24 (Laço iterativo)

```
(for-each (lambda (p)
  (play-note (*metro* time) piano p 80 (*metro* 'dur dur)))
  (pc:make-chord 50 70 2 (pc:diatonic 0 (quote -) degree)))
```

Cada nota é executada com uma altura p , para cada diáde definida em `pc:make-chord`, em um momento definido por `time` em relação ao pulso rítmico, com uma duração ainda a ser definida.

Exemplo 4.25 (Execução da nota)

```
(play-note (*metro* time) piano p 80 (*metro* 'dur dur))
```

`play-note` é definido com os seguintes argumentos, momento de execução ($time \Rightarrow (*metro* time)$), o instrumento tocado, ($instr \Rightarrow piano$), a altura ($pitch \Rightarrow p$), o volume ($vol \Rightarrow 80$) e a duração do acorde ($dur \Rightarrow (*metro*' durdur)$)⁵¹.

4.4.2 Primeira sonoridade tonal

Este código inicial é então modificado, e finalizado em 1'57'', momento em que aparece uma figura, duas diádes, um intervalo de quarta justa entre Sol 2 (MIDI 55) e Dó 3 (MIDI 60). entre Mi bemol 2 (MIDI 51) e Dó 3 (MIDI 60).

Exemplo 4.26 (Estratégia transversal)

```
(define chords
  (lambda (time degree dur)
    (if (member degree '(i)) (set! dur 3.0))
        (for-each (lambda (p)
                     (play-note (*metro* time) piano p
                               (+ 50 (* 20 (cos (* pi time)))))
                               (*metro* 'dur dur)))
                  (pc:make-chord 50 70 2 (pc:diatonic 0 (quote -) degree)))
        (callback (*metro*) (+ time (* .5 dur))) chords (+ time dur)
        (random (assoc degree '((i vii)
                                  (vii i))))
        dur))

(chords (*metro* 'get-beat 4.0) 'i 3.0)
```

Duas transcrições desta primeira figura seguem uma estrutura literal do código, e uma perceptiva. Os primeiros eventos sonoros que ocorrem após o momento de silêncio foram transcritos antes da análise do código. Enquanto Sorensen define um tempo regular de 110 BPM (ver Estratégia transversal 4.18, p. 86), transcrevemos este trecho com um andamento entre 35–40 BPM (ver Figura 25, p. 90). É interessante notar que tais figuras simbolizam neumas (ver

⁵¹ Disponível em <<https://github.com/digego/extempore/blob/5aec8b35c6b3058d1c66de7abf752dc667ab61e4/libs/core/instruments-scm.xtm>>

??, p. ??). No caso específico desta primeira figura, na mão direita, um *bipunctum*, e na mão esquerda, uma *clavis* (ver Estratégia transversal 4.29, p. 92).

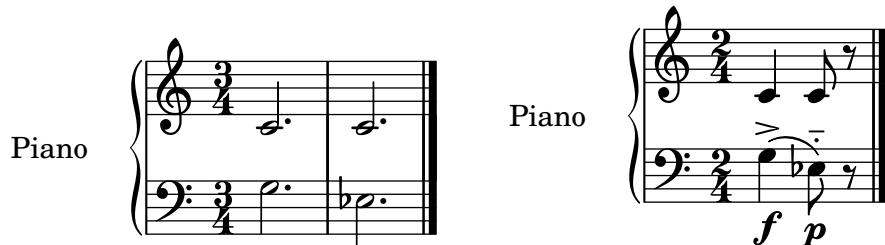


Figura 25 – Transcrição literal e perceptiva do primeiro evento em *A Study in Keith*.
Fonte: autor.

É importante notar que algumas alterações são feitas. A primeira delas é definir outros argumentos para **chords**, como um acorde localizado em um grau de um campo harmônico abstrato, e a duração do acorde executado:

Exemplo 4.27 (Modificação do código original)

```
(define chords
  (lambda (time degree dur) ...))
```

A segunda alteração é a indicação de uma situação condicional na primeira transformação da estratégia transversal T_{ask} . Se o grau a ser executado for uma tônica, no caso, menor (ver Modificação do código original 4.24, p. 88), a duração deste acorde será configurada para uma duração de três unidades de tempo – no caso da nossa transcrição, uma unidade de pulso (ver Figura 25, p. 90).

```
(define chords
  (lambda (time degree dur)
    (if (member degree '(i)) (set! dur 3.0) ...)))
```

A terceira alteração modifica a intensidade das notas:

```
(play-note (*metro* time) piano p
          (+ 50 (* 20 (cos (* pi time))))
          (*metro* 'dur dur))
```

Onde a dinâmica específica ocorre como um comportamento periódico de volumes máximos (fortes), e mínimos (pianos), em, proporcional ao cosseno do tempo instantâneo ($\cos (* \pi time)$), escalonado para valores MIDI:

```
(+ 50 (* 20 (cos (* pi time))))
```

4.4.2.1 Regras de qualidade

A estrutura interna da estratégia `chords` explicita algumas regras de qualidade, bem como permite apresentar uma primieira sequênciа de blocos de eventos \mathcal{K}_{ask}^0 , um conjunto de características \mathcal{F}_{ask}^0 e um pequeno grupo de objetos \mathcal{O}_{ask}^0 . Um conjunto de características é definido pelo momento de execução do evento, \mathcal{F}_{ask}^0 , o grau, \mathcal{F}_{ask}^1 , e a duração deste evento, \mathcal{F}_{ask}^2 . É importante destacar que o momento de execução é relativo ao tempo base, definido dentro do padrão `* metro *` (que será explicado a seguir) de um campo harmônico, onde i representa uma tônica menor, e vii, um acorde de sétimo grau, e a duração deste acorde.

Exemplo 4.28 (Regra de qualidade \mathcal{R}_{ask} .)

```
( ... ( ... (callback (*metro*) (+ time (* .5 dur))) chords (+ time dur)
                    (random (assoc degree '((i vii)
                                              (vii i))))
                    dur))
```

Cujas características irão gerar blocos de eventos, e sequências de blocos de eventos:

```
( ...
  (lambda (time degree dur) ... ))
```

O que permite executar como:

```
(chords (*metro* 'get-beat 4.0) 'i 3.0)
```

4.4.2.2 Primeira sequênciа de blocos de eventos

A [Figura 27](#) indica uma primeira sequênciа de n umas, gerados pelo algoritmo acima, em um padr o que  e repetido por dois ciclos (blocos de eventos \mathcal{E}_{ask}^0 e \mathcal{E}_{ask}^1). Durante este tempo, Sorensen realiza uma mudan a (1º ciclo de bricolagem). Esta mudan a transita entre o segundo bloco \mathcal{E}_{ask}^1 e terceiro bloco \mathcal{E}_{ask}^2 , e sua execuci o resulta em uma transforma o da acentua o, o que termina por colocar, no  ltimo compasso deste ciclo, o s timo grau no tempo forte e o primeiro grau no tempo fraco.

Primeiro Bloco

O primeiro bloco de eventos \mathcal{E}_{ask}^0  e um contraponto de primeira esp cie, articulado em tempos fortes e fracos (ver [Figura 27](#), p. 92).

A aparente repeti o de um mesma classe de eventos sonoros, este mesmo um objeto \mathcal{O}_{ask}^0 , pode ser diferenciada atrav s de figuras neum ticas na m o direita e na m o esquerda (ver exemplo [4.29](#)):

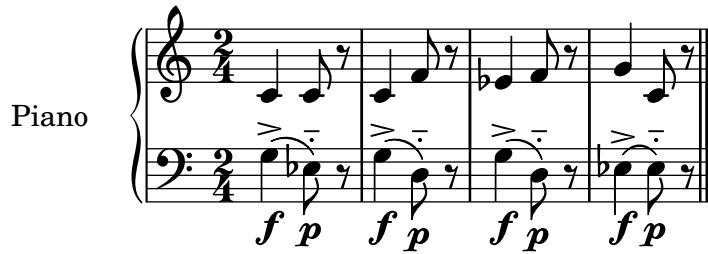
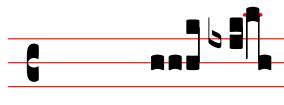


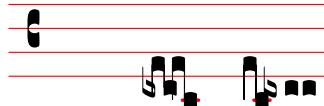
Figura 26 – Primeiros eventos musicais gerados a partir das primeiras estruturas válidas de código. **Fonte:** autor.

Exemplo 4.29 (Transcrição de neumas do primeiro bloco)

Notação neumática para a um *bipunctum*, dois *podatus* e uma *clivis* na mão direita.



E na mão esquerda, três *clivis* e um *bipunctum*.



Segundo Bloco

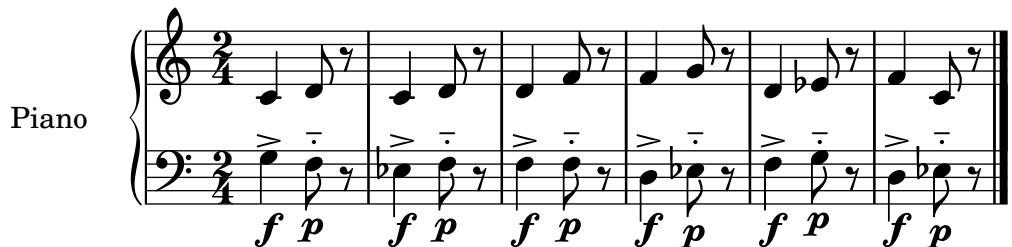
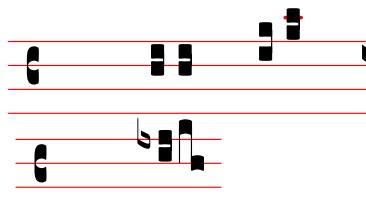


Figura 27 – Segundo bloco de eventos musicais. **Fonte:** autor.

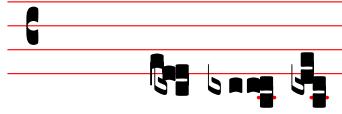
Que pode ser reescrito como neumas na mão direita:

Exemplo 4.30 (Transcrição de neumas do segundo bloco)

Notação neumática para cinco *podatus* e uma *clivis* na mão direita.



E na mão esquerda uma *clavis*, um *podatus*, um *bipunctus*, três *podatus*.



Terceiro Bloco

Enquanto nos blocos \mathcal{E}'^0_{ask} e \mathcal{E}'^1_{ask} existem eventos significativos do ponto de vista figurativo, o aspecto rítmico é único (um tempo forte no *i* grau, um tempo fraco na *vii* grau). É importante destacar que, entre estes blocos, Sorensen realiza uma transformação na estratégia transversal

Exemplo 4.31 (Primeira transformação da estratégia transversal)

```
(define chords
  (lambda (time degree dur)
    (if (member degree '(i)) (set! dur 3.0))
        (for-each (lambda (p)
                     (let* (dur1 (* dur (random '(0.5 1))))
                         (dur2 (- dur dur1)))
                     (play-note (*metro* time) piano p
                               (+ 50 (* 20 (cos (* pi time))))
                               (*metro* 'dur dur1)))
                     (if (> dur2 0)
                         (play-note (*metro* (+ time dur1)) piano
                                   (pc:relative p (random '(-2 -1 1 2))
                                                 (pc:scale 0 'aeolian))
                                   (+ 50 (* 20 (cos (* pi (+ time dur1)))))
                                   (*metro* 'dur dur2)))
                         (pc:make-chord 50 70 2 (pc:diatonic 0 (quote -) degree)))
                     (callback (*metro*) (+ time (* .5 dur)) chords (+ time dur)
                               (random (assoc degree '((i vii)
                                             (v i)))))
                               (random (list 1 2 3)))))

        (chords (*metro* 'get-beat 4.0) 'i 3.0)
```

O que, durante esta transição, gera uma transformação na acentuação (ver Figura 28, p. 94).

Esta estratégia modifica o laço iterativo interno de cada altura da diáde:

Exemplo 4.32 (Laço iterativo modificado)

```
(for-each (lambda (p)
                 (let* (dur1 (* dur (random '(0.5 1)))))
```

Transcrição do terceiro bloco

Figura 28 – Terceiro bloco de eventos musicais. **Fonte:** autor.

```
(dur2 (- dur dur1))
(play-note (*metro* time) piano p
           (+ 50 (* 20 (cos (* pi time))))
           (*metro* 'dur dur1))
(if (> dur2 0)
    (play-note (*metro* (+ time dur1)) piano
               (pc:relative p (random '(-2 -1 1 2)))
               (pc:scale 0 'aeolian))
               (+ 50 (* 20 (cos (* pi (+ time dur1)))))
               (*metro* 'dur dur2)))
    (pc:make-chord 50 70 2 (pc:diatonic 0 (quote -) degree)))
```

A primeira grande mudaça é a definição de duas variáveis internas, através do comando `let` (seja), chamadas `dur1` edur`2`:

Exemplo 4.33 (Laço iterativo modificado)

```
(let* (dur1 (* dur (random '(0.5 1))))
      (dur2 (- dur dur1)))
```

Estas variáveis irão tornar os ritmos de ambas as mãos independentes. O ritmo da mão direita pode ser mantido ou diminuido (`(* dur (random '(0.5 1)))`), enquanto o ritmo da mão esquerda é uma diferença entre uma duração geral, e o ritmo da mão direita. No caso desta nova duração da mão esquerda, é aplicado uma verificação condicional:

Exemplo 4.34 (Laço iterativo modificado)

```
(if (> dur2 0)
    (play-note (*metro* (+ time dur1)) piano
              (pc:relative p (random '(-2 -1 1 2))
                            (pc:scale 0 'aeolian))
              (+ 50 (* 20 (cos (* pi (+ time dur1))))))
              (*metro* 'dur dur2)))
```

Se a diferença entre a duração total e a nova duração for inválida (igual a 0), o ritmo da mão esquerda será igual ao da mão direita, mas sua altura será relativa à segundas menores ou maiores ascendentes ao modo eólico da escala (que no caso transforma a sonoridade tonal em sonoridade modal).

Exemplo 4.35 (Laço iterativo modificado)

```
(pc:relative p (random '(-2 -1 1 2))
              (pc:scale 0 'aeolian))
              (+ 50 (* 20 (cos (* pi (+ time dur1))))))
```

O seu ritmo depende da duração da mão direita:

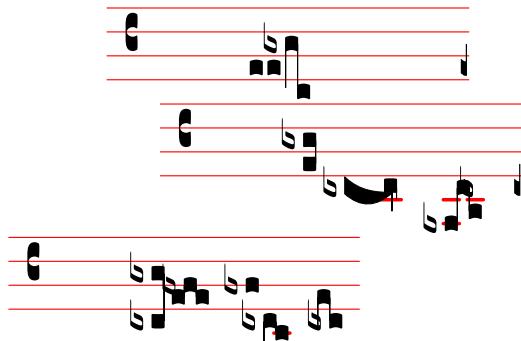
Exemplo 4.36 (Laço iterativo modificado)

```
(+ 50 (* 20 (cos (* pi (+ time dur1))))))
```

Exemplo 4.37 (Transcrição de neumas do terceiro bloco)

Notação neumática para um *bipunctus*, uma *clivis*, um *porrectus subpunctis*, dois *torculus*, uma *clivis* cinco *podatus* e uma *clivis subpunctis*, na mão direita.

E na mão esquerda,



E na mão esquerda:

Rascunho:

The musical score consists of two staves. The top staff is for the right hand (R.H.) and the bottom staff is for the left hand (L.H.). Both staves are in common time (indicated by '2'). The R.H. staff starts with a forte dynamic (f) and a piano dynamic (p). The L.H. staff begins with a piano dynamic (p). Measures 1-16 show eighth-note patterns. Measure 17 begins with a piano dynamic (p), followed by sixteenth-note patterns. Measure 25 concludes with a fermata over the R.H. staff.

Figura 29 – Primeiros perturbações sistêmicas. **Fonte:** autor.

Conclusão

Estimulada por artistas-programadores(as), a interiorização de um processo criativo (bricolagem) pode ter conduzido à formação de programas de pesquisas interdisciplinares no Norte global. Isto é, teorias das Artes Músicais, Audiovisuais, Corporais, Têxteis, Filosofia, e se a criatividade permitir, da Engenharia Crítica, são coisificadas para elaboração de novos conceitos. Uma classe particular de artistas-programadores(as) (MCLEAN, 2011, p. 16), o(a) improvisador(a) de códigos (*live coder*), admite conhecimentos da engenharia de computação, revisada sob o prisma de uma ou mais teorias como extensão para improvisações ou análise de registros.

O *live coder* é aquele que pratica o *live coding* como divulgado em uma página da Internet⁵² e uma publicação (WARD et al., 2004); também se mantém informado das publicações que investigam o *live coding*, como em revistas acadêmicas como *Computer Music Journal*, *Leonardo*, listas de email de *softwares*, páginas do Google, e improvisa programas com fins pedagógicos, musicais, audiovisuais, têxteis e filosóficos. O improvisador de códigos pratica o *live coding*:

Programação imediata (ou: programação de conversa, programação no fluxo, programação interativa) é um paradigma que inclue a atividade de programação ela mesma como uma operação do programa. Isto significa um programa que não é tomado como ferramenta que cria primeiro, e depois é produtivo, mas um processo de construção dinâmica de descrição e conversação - escrever o código e então se tornar parte da prática musical ou experimental. (SuperCollider.ORG, 2014, Verbete JITLib)⁵³

Ward et al. (2004) definem a improvisação de códigos como “atividade da escrita integral (ou partes) de um programa enquanto ele é executado”⁵⁴. Blackwell e Collins (2005) enfatizam a definição do ponto de vista da linguagem de programação como instrumento musical. McLean (2006-07-30) relata o *live coding* como ferramenta para um *Disk Jockey codificado*. Sorensen e Swift (2009) pontuam a improvisação de códigos como “uma prática de performance para o qual linguagens de computador definem o meio primário de expressão artística.”⁵⁵. Para Sorensen e Gardner (2010), *live coding* (ou *livecoding*) envolve a premissa de uma programação-partitura audiovisual reativa:

Livecoding é uma prática de arte computacional que envolve criação em tempo-real de programas de audiovisual generativo para performances

⁵² Disponível em <<http://www.toplap.org>>

⁵³ Tradução de *Just in time programming (or: conversational programming, live coding , on-the fly-programming, interactive programming) is a paradigm that includes the programming activity itself in the program's operation. This means a program is not taken as a tool that is made first, then to be productive, but a dynamic construction process of description and conversation - writing code thus becoming a closer part of musical or experimental practice.*

⁵⁴ Tradução nossa de “*Live coding is the activity of writing (parts of) a program while it runs*”

⁵⁵ Tradução nossa de “*Live coding is a performance practice for which computer languages define the primary means of expression.*”

multimídias interativas. Comumente as ações dos programadores são expostas para uma audiência por projeção do ambiente de edição. Performances de livecoding geralmente envolvem mais de um participante, e são geralmente iniciadas a partir de uma folha conceitual em branco (SORENSEN; GARDNER, 2010, p. 823)⁵⁶.

Magnusson (2011), Collins (2014) sintetizam o *live coding* como improvisação audiovisual. Sorensen (2014) define como “programar sistemas de tempo-real em tempo real”⁵⁷. Em um discussão entitulada “Wtf is livecoding”⁵⁸ diz que o “*Live coding* celebra a efemeridade da própria definição”⁵⁹ (ver Figura 30, p. 98).

Embora semelhantes, as definições mudam de detalhes de acordo com o contexto. Por exemplo, Ward et al. (2004) enfatizam que o código pode ser (re)composto de partes menores. McLean (2006-07-30) enfatiza algum onde um código é (re)programado. Sorensen (2014) enfatiza que modificar algo é próprio da técnica, em seus significados técnicos. O compositor Nick Collins situa que a definição nunca deve ser uma constante, e sim caracterizada em função do contexto.



Figura 30 – Definição de *live coding*: “Insira a definição aqui”. **Fonte:** Collins (2014).

Se por um lado a definição agrega definições, o que dificulta a tarefa inicial de descrever os fundamentos objeto de pesquisa, por outro ilustra a improvisação de códigos como um *Universo de conceitos*. Neste trabalho consideramos que definições ou performances de improvisação de códigos estão contidas em diferentes *Espaços Conceituais* (WIGGINS, 2006; McLean, 2006). Artistas-programadores (*live coders*) transitam entre os Espaços Conceituais para criação de Sistemas Criativos (códigos, programas). Estes Sistemas Criativos são representados em diferentes Linguagens de Programação. Regras práticas conduzem o processo de escrita e exposição desta linguagem; mas não restringem o resultado (no caso da pesquisa, musical). Mas algumas

⁵⁶ Tradução de *Livecoding [10, 50] is a computational arts practice that involves the real-time creation of generative audiovisual software for interactive multimedia performance. Commonly the programmers' actions are exposed to the audience by projection of the editing environment. Livecoding performances often involve more than one participant, and are often commenced from a conceptual blank slate*

⁵⁷ Tradução nossa de “*programming real-time systems in real-time*”

⁵⁸ Disponível em <<http://lurk.org/groups/livecode/messages/topic/ofAxZpxsKFpDRLnoA48Bh>>

⁵⁹ Tradução nossa de “*Live coding celebrates the ephemerality of definition itself*”

categorizações musicais se destacam. Neste sentido, selecionamos um exemplo simbólico, *A Study in Keith* de Andrew Sorensen e Swift (2009)⁶⁰. Representa um caso particular que foge dos exemplos citados anteriormente, mas envolve a manutenção de uma tradição musical tonal através de um interessante esforço de *replicação do estilo*.

No Capítulo 1 selecionamos algumas abordagens de uma grande quantidade de exemplos possíveis. Foram escolhidos por manterem alguma conexão com a improvisação de códigos no contexto sonoro. De certa forma, induzimos conceitos ligados à Música. No Capítulo 3 apresentamos um modelo de formalização dEspaços conceituais observados pelo prisma do Modelo de Improvisação discutido por Alex McLean (2006). No Capítulo 4, organizamos conceitos de uma sonoridade-algoritmo inicial de *A Study in Keith* segundo este modelo. Por último uma conclusão, mais uma reflexão sobre o processo de pesquisa do que obtenção de um resultado final. Dois apêndices foram adicionados para exposição do material que estimulou o interesse pelo tema discutido.

⁶⁰ Disponível em <<https://vimeo.com/2433947>>.

Referências

- AYCOCK, j. A brief history of just-in-time. p. 97–113, 2003. Disponível em: <<http://www.cs.tufts.edu/comp/150IPL/papers/aycock03jit.pdf>>. Citado 3 vezes nas páginas 111, 113 e 119.
- BAKER, J. E-mail, *Origins of "Blog" and "Blogger"*. Disponível em: <<http://listserv.linguistlist.org/cgi-bin/wa?A2=ind0804C&L=ADS-L&P=R16795&I=-3>>. Citado na página 44.
- BLACKWELL, A.; COLLINS, N. The programming language as a musical instrument. p. 120–130, 2005. Disponível em: <http://www.researchgate.net/publication/250419052_The_Programming_Language_as_a_Musical_Instrument>. Citado 2 vezes nas páginas 38 e 97.
- Boden, M. Creativity and artificial intelligence. online, p. 10, 1998. Disponível em: <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.667.3710&rep=rep1&type=pdf>>. Citado 2 vezes nas páginas 49 e 50.
- BODEN, M. *The Creative Mind: myths and mechanisms*. 2. ed. Routledge, Taylor & Francis Group, 2004. ISBN 0-203-34008-6. Disponível em: <http://www.pauladaunt.com/books/The%20Creative%20Mind_%20Myths%20and%20Mechanisms.pdf>. Citado 4 vezes nas páginas 47, 48, 49 e 59.
- BROWN, C.; BISCHOF, J. *INDIGENOUS TO THE NET: Early Network Music Bands in the San Francisco Bay Area*. 2002. Disponível em: <<http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html>>. Citado 7 vezes nas páginas vii, 32, 33, 34, 35, 36 e 37.
- CANUTE live in Jubez Karlsruhe Algorave. 2015–27–01. Disponível em: <<https://www.youtube.com/watch?v=uAq4BAbvRS4>>. Citado 2 vezes nas páginas vii e 10.
- CHESIRE, T. *Hacking meets clubbing with the 'algorave'*. Wired Magazine, 2013. Disponível em: <<http://www.wired.co.uk/magazine/archive/2013/09/play/algorave>>. Citado na página 9.
- CHURCH, L.; GREEN, T. Cognitive dimensions - a short tutorial. In: *Proceedings of 20th Psychology of Programming Interest Group*. [S.l.: s.n.], 2008. Citado na página 52.
- COLLINS, N. Generative music and laptop performance. v. 22, n. 4, p. 67–79, 2003. Disponível em: <<http://portal.ku.edu.tr/~megunal/articles/Generative%20Music%20and%20Laptop%20Performance.pdf>>. Citado 2 vezes nas páginas 7 e 8.
- COLLINS, N. *Origins of live coding*. Durham University, 2014. Disponível em: <<http://www.livecodenetwork.org/files/2014/05/originsoflicecoding.pdf>>. Citado 2 vezes nas páginas viii e 98.
- COLLINS, N.; McLean, A. Algorave: Live performance of algorithmic electronic dance music. In: *Proceedings of the International Conference on New Interfaces for Musical Expression*. [s.n.], 2014. p. 355–358. Disponível em: <http://nime2014.org/proceedings/papers/426_paper.pdf>. Citado 2 vezes nas páginas 9 e 10.
- COLLINS, N.; OLOFSSON, F. A protocol for audiovisual cutting. p. 4, 2003. Disponível em: <<http://quod.lib.umich.edu/cache//b/b/p/bbp2372.2003.011/bbp2372.2003.011.pdf#page=2;zoom=75>>. Citado na página 10.

- DOWNIE, M. *Choreographing the Extended Agent: performance graphics for dance theater*. phdthesis — MIT, 2005. Disponível em: <<http://openendedgroup.com/writings/downieThesis.html>>. Citado na página 13.
- ENO, B. Base de dados, *Music for Airports liner notes*. 1978. Disponível em: <http://music.hyperreal.org/artists/brian_eno/MFA-txt.html>. Citado na página 10.
- ENO, B. *Generative Music: "Evolving metaphors, in my opinion, is what artists do. A talk delivered in San Francisco*. 1996. Disponível em: <<http://www.inmotionmagazine.com/eno1.html>>. Citado na página 39.
- FABBRI, R.; VIEIRA, V. *Livecoding*. Wiki - Pontão Nós Digitais, 2011. Disponível em: <<http://wiki.nosdigitais.teia.org.br/Livecoding>>. Citado 2 vezes nas páginas vii e 18.
- FENERICH, A.; OBICI, G.; SCHIAVONI, F. Marulho TransOceânico: performance musical entre dois continentes. p. 12, 2014. Disponível em: <<https://www.academia.edu/t/M8Fvh/8178331>>. Citado na página 77.
- FORTH, J.; WIGGINS, G. A.; MCLEAN, A. Unifying conceptual spaces: Concept formation in musical creative systems. *Minds & Machines*, v. 20, p. 503—532, 2010. Disponível em: <http://www.retiary.org/ls/expanding_universe/>. Citado 2 vezes nas páginas xii e xiv.
- GARBOLINO, M. *Keith jarrett Disco Version 19*. [s.n.], 2014. Disponível em: <http://www.keithjarrett.org/wp-content/uploads/Discographie_Jarrett-november-2014.pdf>. Citado na página 73.
- GASPERINI, G. *Storia della semiografia musicale*. 1st. ed. [S.l.]: Manualli Ulrico Hoepli – Libraio dela Casa Real, 1905. Citado na página 85.
- GIOMI, F.; LIGABUE, M. *Conversazioni e riflessioni con Pietro Grossi*. [S.l.]: Sismel Edizioni del Galluzzo L'istante zero, 1999. Citado 2 vezes nas páginas 27 e 30.
- GRIFFTHS, D. *Tag Archives: algorave*. Dave's blog of art and programming, 2013. Disponível em: <<http://www.pawfal.org/dave/blog/tag/algorave/>>. Citado 3 vezes nas páginas vii, 11 e 12.
- GRIFFTHS, D. *A cryptoweaving experiment*. 2015. Disponível em: <<http://kairotic.org/a-cryptoweaving-experiment>>. Citado 4 vezes nas páginas vii, 2, 4 e 5.
- GRIFFTHS, D. *Weavecoding performance experiments in Cornwall*. 2015. Disponível em: <<http://www.pawfal.org/dave/blog/tag/weavecoding/>>. Citado 2 vezes nas páginas vii e 6.
- HAOULI, J. E.; MANNIS, J. A. Abertura para o silêncio. In: . [S.l.: s.n.]. p. 8. Citado na página 16.
- IAZZETTA, F. *Música e mediação tecnológica*. [S.l.]: Ed. Perspectiva-Fapesp, 2009. ISBN 9.788527308724E12. Citado 3 vezes nas páginas 34, 46 e 116.
- JR., J. S. J. À procura da batida perfeita: a importância do gênero musical para a análise da música popular massiva. v. 6, n. 2, p. 31–46, 2003. Citado 3 vezes nas páginas 8, 22 e 69.
- Junior, A. D. d. C.; Lee, S. W.; Essl, G. Supercopair: Collaborative live coding on supercollider through the cloud. In: *ICLC2015 Proceedings*. [S.l.: s.n.], 2015. p. 152–158. ISBN 978 0 85316 340 4. Citado 2 vezes nas páginas 24 e 120.
- KOELLREUTTER, H.-J. Wu-li: Um ensaio de música experimental. p. 203–208. Citado na página 16.

- LUNHANI, G.; SCHIAVONI, F. Termpot: criação, edição de funções no navegador em tempo de execução. p. 154–159, 2015. Disponível em: <<http://compmus.ime.usp.br/sbcm2015/files/proceedings-print.pdf>>. Citado 2 vezes nas páginas 24 e 120.
- MAGNUSSON, T. Algorithms as scores: Coding live music. v. 21, p. 19–23, 2011. Citado 2 vezes nas páginas 77 e 98.
- MAILMAN, J. B. Agency, determinism, focal time frames, and processive minimalist music. In: *Music and Narrative Since 1900*. [s.n.], 2013. p. 125–144. Disponível em: <https://www.academia.edu/749803/Agency_Determinism_Focal_Time_Frames_and_Narrative_in_Processive_Minimalist_Music>. Citado na página 39.
- MALENFANT, J.; JACQUES, M.; DEMERS, F.-N. A tutorial on behavioral reflection and its implementation. v. 38, n. 1, p. 65–76, 1996. Disponível em: <<http://www2.parc.com/csl/groups/sda/projects/reflection96/docs/malenfant/malenfant.pdf>>. Citado na página 29.
- MARTIN, G. *Human Neuropsychology*. 2nd. ed. [S.l.]: Prentice Hall, 2006. Citado na página 52.
- MATHEWS, M. V. The digital computer as a musical instrument. v. 142, n. 3592, p. 553–557, 1963. Disponível em: <<http://www.jstor.org/stable/1712380>>. Citado 2 vezes nas páginas 28 e 117.
- MATHEWS, M. V. et al. *The technology of computer music*. 2a, 1974. ed. [S.l.]: MIT press, 1969. ISBN 0 26213050 5. Citado na página 28.
- MATHEWS, M. V.; MOORE, F. GROOVE a program to compose, store, and edit functions of time. p. 7, 1970. Citado 4 vezes nas páginas viii, xiv, 113 e 114.
- McCarthy, J. Recursive functions of symbolic expressions and their computation by machine, part i. p. 34, 1960. Disponível em: <<http://www-formal.stanford.edu/jmc/recursive.pdf>>. Citado 2 vezes nas páginas 78 e 79.
- MCCARTNEY, J. Superollider: a new real time synthesis language. p. 257–258, 1996. Disponível em: <https://www.academia.edu/7249277/Patterns_of_movement_in_live_languages>. Citado 2 vezes nas páginas 116 e 117.
- McLean, A. Music improvisation and creative systems. online, p. 6, 2006. Disponível em: <https://www.academia.edu/467101/Music_improvisation_and_creative_systems>. Citado 8 vezes nas páginas iv, xii, xiv, 59, 60, 61, 98 e 99.
- McLean, A. *hacking perl music*. Youtube, 2006–07–30. Disponível em: <<https://www.youtube.com/watch?v=fbefIdbSmD4>>. Citado 2 vezes nas páginas 97 e 98.
- MCLEAN, A. *Artist-Programmers and Programming Languages for the Arts*. Tese (Doutorado) — Department of Computing, Goldsmiths, University of London, October 2011. Disponível em: <<http://slab.org/writing/thesis.pdf>>. Citado 17 vezes nas páginas iv, vii, xi, xii, xiv, 3, 7, 39, 51, 52, 53, 54, 55, 57, 71, 72 e 97.
- McLean, A.; WIGGINS, G. *Patterns of movement in live languages*. 2009. Disponível em: <https://www.academia.edu/7249277/Patterns_of_movement_in_live_languages>. Citado na página 37.
- McLean, A.; WIGGINS, G. *Petrol: Reactive pattern language for improvised music*. Proceedings of the International Computer Music Conference, 2010. Disponível em: <https://www.academia.edu/467094/Petrol_Reactive_pattern_language_for_improvised_music>. Citado na página 56.

McLean, A.; WIGGINS, G. *TIDAL – PATTERN LANGUAGE FOR LIVE CODING OF MUSIC*. Centre for Cognition, Computation and Culture; Department of Computing Goldsmiths, University of London, 2010. Disponível em: <<http://smcnetwork.org/files/proceedings/2010/39.pdf>>. Citado 3 vezes nas páginas 17, 55 e 56.

MORI, G. Analysing live coding with ethnographical approach. In: *ICLC2015 Proceedings*. [S.l.: s.n.], 2015. p. 117–124. ISBN 978 0 85316 340 4. Citado na página 1.

MORI, G. Pietro grossi's live coding. an early case of computer music performance. In: *ICLC2015 Proceedings*. [S.l.: s.n.], 2015. p. 125–132. ISBN 978 0 85316 340 4. Citado 4 vezes nas páginas 27, 30, 31 e 109.

NUNZIO, A. D. *Genesi, sviluppo e diffusione del software "MUSIC N"nella storia della composizione informatica*. phdthesis — Facoltà di Lettere e Filosofia - Università degli Studi di Bologna, 2010. Citado 2 vezes nas páginas 28 e 113.

OPENSSOUNDCONTROL.ORG. *OSC: Open Sound Control*. opensoundcontrol.org, 2002. Disponível em: <<http://opensoundcontrol.org/about>>. Citado na página 56.

PAIVIO, A. *Mental representations: A Dual Coding Approach*. Oxford Psychology Series. [S.l.]: Oxford University Press, 1990. Citado na página 51.

PARK, T. H.; MATHEWS, M. An interview with max mathews. v. 33, n. 3, p. 9–22, 2009. Disponível em: <<http://www.jstor.org/stable/40301041>>. Citado na página 28.

Pressing, J. Cognitive process in improvisation. In: *Cognitive Processes in the Perception of Art*. Elsevier Science Publishers B.V. (North-Holland), 1984. p. 345–363. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0166411508623584>>. Citado na página xiii.

Pressing, J. Improvisation: Methods and models. In: *Generative processes in music*. (ed. J. Sloboda) Oxford University Press, 1987. p. 50. Disponível em: <<http://musicweb.ucsd.edu/~sdubnov/Mu206/improv-methods.pdf>>. Citado 3 vezes nas páginas xiii, 60 e 61.

PROSPERO, C. D. Social participation. In: *ICLC2015 Proceedings*. [S.l.: s.n.], 2015. p. 68–73. ISBN 978 0 85316 340 4. Citado na página 9.

RAMSAY, S. *Algorithms are Thoughts, Chainsaws are Tools*. Vimeo, 2010. Disponível em: <<https://vimeo.com/9790850>>. Citado 4 vezes nas páginas vii, 40, 44 e 45.

REICH, S. Music as a gradual process. In: *Writings about Music, 1965–2000*. Oxford University Press, 1968. ISBN 978-0-19-511171-2. Disponível em: <<http://ccnmtl.columbia.edu/draft/ben/feld/mod1/readings/reich.html>>. Citado 2 vezes nas páginas 39 e 116.

RIETVELD, H. C. Bloomsbury Publishing Inc., 2013. 1–14 p. Disponível em: <<http://file.ebook777.com/005/DjCulInTheMixPowTecAndSocChaInEleDanMus.pdf>>. Citado 2 vezes nas páginas 1 e 7.

ROADS, C. The second steim symposium on interactive composition in live electronic music. p. 45–50, 1986. Disponível em: <<http://www.jstor.org/stable/3679484>>. Citado na página 37.

ROADS, C.; MATHEWS, M. Interview with max mathews. v. 4, n. 4, p. 15–22, 1980. Disponível em: <<http://www.jstor.org/stable/3679463>>. Citado na página 28.

ROBERTS, C.; KUCHERA-MORIN, J. *Gibber: live-coding audio in the browser*. [S.l.]: University of California at Santa Barbara: Media Arts & Technology Program, 2012. Citado 3 vezes nas páginas 24, 111 e 120.

- ROBERTS, C.; WAKEFIELD, G.; WRIGHT, M. The web browser as synthesizer and interface. p. 6, 2013. Citado 2 vezes nas páginas 24 e 120.
- SANTOS, B. d. S. Para além do pensamento abissal: Das linhas globais a uma ecologia de saberes. n. 78, p. 3–46, 2007. Disponível em: <http://www.ces.uc.pt/myces/UserFiles/livros/147_Para%20alem%20do%20pensamento%20abissal_RCCS78.pdf>. Citado na página xii.
- SANTOS, B. d. S. A filosofia à venda, a dourada ignorância e a aposta de pascal. n. 80, p. 11–43, 2008. Disponível em: <http://www.ces.uc.pt/myces/UserFiles/livros/47_Douta%20Ignorancia.pdf>. Citado 2 vezes nas páginas xi e xii.
- SCHLOSS, A. Using contemporary technology in live performance; the dilemma of the performer. v. 32, p. 239–242, 2003. Disponível em: <https://people.finearts.uvic.ca/~aschloss/publications/JNMR02_Dilemma_of_the_Performer.pdf>. Citado 2 vezes nas páginas 1 e 42.
- SICCHIO, K. Hacking choreography: Dance and live coding. p. 31–39, 2014. Disponível em: <http://muse.jhu.edu/journals/computer_music_journal/v038/38.1.sicchio.pdf>. Citado na página 13.
- SMITH, S. W. *DSP Guide - The Scientist and Engineer's Guide to Digital Signal Processing*. 2012–06. Disponível em: <<http://dspguide.com/>>. Citado na página 78.
- SOARES, G. R. *Luteria Composicional de algoritmos pós-tonais v1.1FINAL*. Prévia da dissertação para a banca de qualificação para o Mestrado em Arte, Cultura e Linguagens do IAD-UFJF. — UFJF, 2015–03–13. Disponível em: <https://github.com/glerm/luteria/raw/master/LUTERIA_2015janeiro.pdf>. Citado 4 vezes nas páginas 34, 46, 74 e 116.
- SORENSEN, A. *Disklavier sessions*. Vimeo, 2013. Disponível em: <<https://vimeo.com/50061269>>. Citado na página 75.
- SORENSEN, A. *Programming in Time - Live Coding for Creative Performances*. 2014. Disponível em: <<https://www.youtube.com/watch?v=Sg2BjFQnr9s>>. Citado na página 98.
- SORENSEN, A. *The Disklavier Sessions*. Youtube, 2015. Disponível em: <<https://www.youtube.com/watch?v=cFEadvBeBqw>>. Citado 4 vezes nas páginas 71, 75, 86 e 88.
- SORENSEN, A.; GARDNER, H. Programming with time: cyber-physical programming with impromptu. p. 822–834, 2010. Disponível em: <<http://diypl.us/~bryan/papers2/paperbot/67845a4fb5b009259c389f90ab02c1c0.pdf>>. Citado 5 vezes nas páginas 77, 81, 84, 97 e 98.
- SORENSEN, A.; SWIFT, B. *A Study in Keith*. Vimeo, 2009. Disponível em: <<https://vimeo.com/2433947>>. Citado 5 vezes nas páginas iv, 71, 75, 97 e 99.
- SPIEGEL, L. The expanding universe: 1970s computer music from bell labs by laurie spiegel. disponível em <http://www.retiary.org/ls/expanding_universe/>. *Retiary*, 1975. Disponível em: <http://www.retiary.org/ls/expanding_universe/>. Citado 4 vezes nas páginas viii, 113, 114 e 115.
- SuperCollider.ORG. *SuperCollider Overviews: JITLib - An overview of the Just In Time library*. 2014. Citado na página 97.
- SWENSON, J. *The Rolling stone jazz record guide*. Rolling Stone Press, 1985. 219 p. ISBN 039472643-X. Disponível em: <https://openlibrary.org/books/OL2867249M/The_Rolling_stone_jazz_record_guide>. Citado na página 72.
- SWIFT, B. *Playing an instrument (part II)*. 2012. Disponível em: <<http://benswift.me/2012/10/15/playing-an-instrument-part-ii/>>. Citado 3 vezes nas páginas viii, 85 e 87.

Sá, S. P. d. A música na era de suas tecnologias de reprodução. v. 12, n. 19, p. 19, 2006. Disponível em: <<http://www.compos.org.br/seer/index.php/e-compos/article/viewFile/92/92>>. Citado 2 vezes nas páginas 8 e 69.

Sá, S. P. d. Se você gosta de madonna também vai gostar de britney! ou não? gêneros, gostos e disputa simbólica nos sistemas de recomendação musical. v. 12, n. 2, p. 1808–2599, 2009. Citado 3 vezes nas páginas 8, 22 e 69.

Thornton, C. A quantitative reconstruction of boden's creativity theory. p. 29, 2007. Disponível em: <<http://users.sussex.ac.uk/~christ/papers/boden-reconstruction.pdf>>. Citado 11 vezes nas páginas vii, 47, 48, 50, 58, 62, 63, 64, 65, 66 e 67.

TOUSSAINT, G. T. The euclidean algorithm generates traditional musical rhythms. p. 47–56, 2005. Disponível em: <<http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf>>. Citado na página 55.

TRUAX, B. Review of pietro grossi: 24 capricci by niccolò paganini. v. 8, p. 59–60, 1984. Citado na página 28.

VIEIRA, V. et al. Vivace: A collaborative live coding language. arXiv, n. 1502, p. 16, 2015. Disponível em: <<http://arxiv.org/abs/1502.01312>>. Citado 2 vezes nas páginas 18 e 24.

WANG, G. *Read me paper - Revision as of 01:11, 1 August 2005 - A Historical Perspective*. 2005. Disponível em: <http://toplap.org/wiki/index.php?title=Read_me_paper&oldid=60#A_Historical_Perspective>. Citado na página 37.

WARD, A. et al. *Live algorithm programming and temporary organization for its promotion*. TOPLAP.ORG, 2004. Disponível em: <<http://art.runme.org/1107861145-2780-0/livecoding.pdf>>. Citado 16 vezes nas páginas xii, 1, 4, 7, 17, 27, 37, 38, 39, 40, 41, 42, 97, 98, 109 e 118.

WIGGINS, G. A. A preliminary framework for description, analysis and comparison of creative systems. v. 19, n. 3592, p. 449–458, 2006. Disponível em: <<http://axon.cs.byu.edu/Dan/673/papers/wiggins.pdf>>. Citado 7 vezes nas páginas iv, vii, 49, 51, 58, 60 e 98.

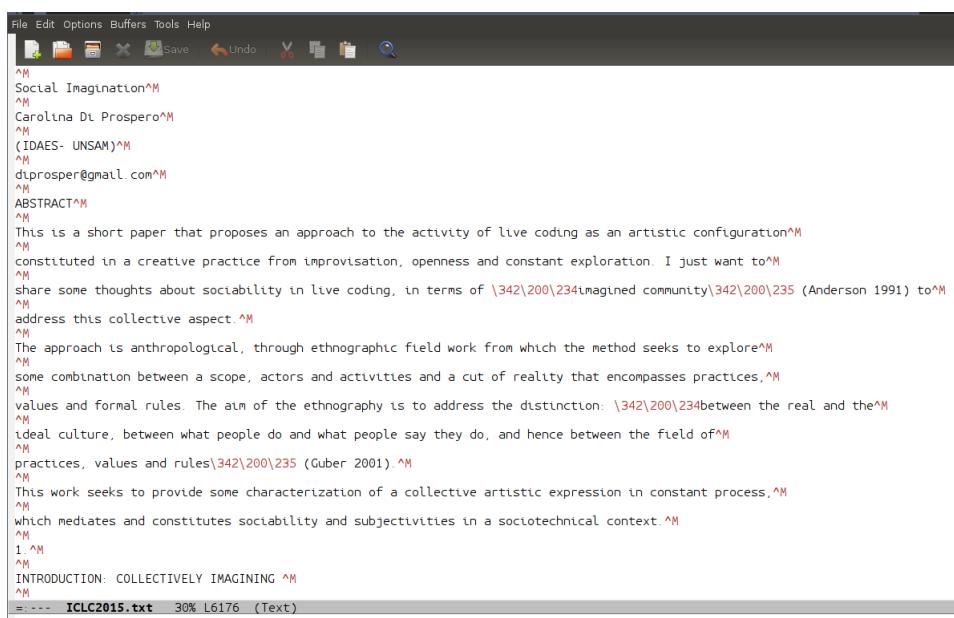
WYSE, L.; SUBRAMANIAN, S. The viability of the web browser as a computer music platform. v. 37, n. 4, p. 10–23, 2014. Citado na página 111.

APÊNDICE A – Código fonte de um Universo Conceitual como nuvem de palavras sobre o improviso de códigos

Apresentamos no exemplo A.1 o código-fonte, em linguagem python, utilizado para extração parcial do espaço conceitual da pesquisa (??, p. ??). A biblioteca utilizada, *Wordcloud*, pode ser encontrada em <https://github.com/amueller/word_cloud>.

O Código abaixo considera a seguinte situação: é possível converter um arquivo de texto em formato *.pdf* para formato *.txt*. Feita a conversão, é possível realizar um levantamento estatístico das palavras mais usadas (o que pode, parcialmente, indicar idéias e conceitos).

Existem programas online, como o encontrado no link <<http://convertonlinefree.com/PDFToTXTEN.aspx>>, que realizam a conversão. É necessária a correção de alguns erros de caracteres (ver Figura 31). Além disso, informações de cabeçalho, códigos-fontes, e outros elementos de editoração, foram descartados por considerarmos que não eram parte do corpo textual. Em outras palavras, descartamos palavras que não faziam parte de um discurso de texto, ou atrapalhavam o processo de criação da imagem. O que por si não resolve todos os problemas, mas auxilia na elaboração da imagem.



```

File Edit Options Buffers Tools Help
^M Social Imagination^M
^M Carolina Di Prospero^M
^M (IDAES- UNSAM)^M
^M diprosp@gmail.com^M
^M ABSTRACT^M
^M This is a short paper that proposes an approach to the activity of live coding as an artistic configuration^M
^M constituted in a creative practice from improvisation, openness and constant exploration. I just want to^M
^M share some thoughts about sociability in live coding, in terms of \342\200\234imagined community\342\200\235 (Anderson 1991) to^M
^M address this collective aspect.^M
^M The approach is anthropological, through ethnographic field work from which the method seeks to explore^M
^M some combination between a scope, actors and activities and a cut of reality that encompasses practices.^M
^M values and formal rules. The aim of the ethnography is to address the distinction: \342\200\234between the real and the^M
^M ideal culture, between what people do and what people say they do, and hence between the field of^M
^M practices, values and rules\342\200\235 (Guber 2001). ^M
^M This work seeks to provide some characterization of a collective artistic expression in constant process.^M
^M which mediates and constitutes sociability and subjectivities in a sociotechnical context.^M
^M
1^M
^M INTRODUCTION: COLLECTIVELY IMAGINING ^M
^M
= --- ICLC2015.txt 30% L6176 {Text}

```

Figura 31 – Resultado da conversão do arquivo *pdf* para *txt* resulta em problemas de codificação que necessitaram ser corrigidos por comparação com o arquivo original. **Fonte:** autor.

Exemplo A.1 (Código-fonte que utiliza a biblioteca wordcloud)

```
# Bibliotecas utilizadas
from urllib2 import urlopen
from bs4 import *
import re
import datetime
from iso import *
import matplotlib.pyplot as plt

from wordcloud import WordCloud

# Abra o arquivo em modo de leitura
# ICLC2015.txt:
#   - um arquivo em modo de texto
#   convertido de um conjunto de artigos
#   - Universo conceitual publicado
#   em 2015 na Inglaterra
t = open("ICLC2015.txt", "r").read()

# Gere uma nuvem de palavras
wc = WordCloud().generate(t)

# Mostre a imagem gerada com o matplotlib,
# biblioteca para plotar imagens por dados numericos,
# levando em conta as frequencias das palavras
plt.figure()
plt.imshow(wc)
plt.axis("off")
plt.show()
```

A.1 Classes qualitativas de um Universo de conceitos do *live coding*

Com auxílio da biblioteca NLTK¹, categorizamos a nuvem de conceitos de acordo com sua função textual (ver exemplo B.1).

¹ Disponível em <<http://nltk.org/>>.

Exemplo A.2 (Categorização dos dados)

```
# Importe bibliotecas
from os import path
from wordcloud import WordCloud
import math
import nltk

# Abra o arquivo em modo de leitura
t = open("assets/ICLC2015.txt", "r").read()
wc = WordCloud().generate(t)

# Organize as palavras por frequencia
# de presenca no texto
groups = [[] for i in range(10)]

for i, t in enumerate(wc.words_):
    freq = t[1]
    word = t[0]
    index = int(math.floor(freq*10))
    if freq >= index/10.0 and freq < (index+1)/10.0:
        print word
        print index
        groups[index-1].append(word)

# CLASSIFIQUE AS PALAVRAS
groups = [nltk.pos_tag(e) for e in groups]
print groups
```

O código acima gerou uma saída textual que foi reorganizada na tabela [Tabela 5](#).

Uma breve análise da nuvem de palavras (ver ??, p. ??), pode elucidar parte das questões-satélites do *live coding*. Na [Tabela 5](#) filtrei parte dos resultados por conjuntos de funções textuais – sujeitos-humanos, sujeitos-ferramentas, verbos, adjetivos e substantivos – e quantas vezes foram utilizados, em categorias qualitativas (0, menos usado e 9 o mais usado, sendo que 6 e 7 não apresentaram resultados).

No caso dos sujeitos-humanos, podemos ver nomes de Nick Collins e Alex McLean, praticantes responsáveis pela criação de um manifesto, em parceria com [Ward et al. \(2004\)](#). Pietro Grossi, é um personagem recentemente estudado por [Mori \(2015b\)](#) como um caso prematuro de *live coding*, a partir do final da década de sessenta.

No caso dos sujeitos-ferramentas, destacamos o papel do *SuperCollider*, já citado anterior-

Tabela 5 – Tabela de classes qualitativas de termos utilizados nos anais do ICLC2015, agrupados por funções textuais.

Número Qualitativo/Função	0	1	2	3	4	5	8	9
Pessoas	-	Collins, Blackwell, McLean, Grossi	-	-	-	-	-	-
Aplicativos	-	SuperCollider, Gibber, SonicPi	-	-	-	-	-	
Verbos	take, see, shared, networked, explore, made	make, provide, writing, solving, making	used	using, coding	performer	-	-	-
Adjetivo ou num- meral, ordinal	less, open, potential, similar, important, cognitive, virtual	first, real, electronic, visual, ensemble, possible, free, livecoding, aspect	musical, many	new, one	-	-	live	-
Substantivo	Browser, point, approach, order, node, collaborative, number, source, present, community, server, framework, orchestra, digital, level, kind, type, memory, analysis, line, body, concept, technology, working, org, current, show, mean, end, processes, people, internati- onal	University, conference, proceedings, network, interface, environment, text, form, context, musician, space, paper, program, audience, function, change, control, human, laptop, interaction, structure, part, session, tool, result, create, object, case, algorithm, value, development, material, set, technique, parameter, idea, screen, video, application, support, composition, piece, knowledge, feature, cell, activity, art, action, information, method, web, rule, group, need, particular, project, allow, collaboration, programmer, member, play, output	use, coder, process, state, example, way, software, research, problem, experience, design, improvisation, different, machine, pattern, audio	work, instrument	system, computer, user, language, time, practice, sound	programming	performance, code	“live coding”, music

mente, e do *Gibber*(ROBERTS; KUCHERA-MORIN, 2012; WYSE; SUBRAMANIAN, 2014)². Ambos são ambientes de programação para de síntese sonora e composição algorítmica. Uma característica em comum destes ambientes, o procedimento de compilação de códigos, é conhecido como *Just In Time* (AYCOCK, 2003), dispositivo técnico que permitiu a execução de códigos durante o tempo de execução.

Verbos fornecem informação sobre o comportamento dos improvisadores de códigos. Além da atividades como *performatizar* e *codificar*, é notável atividades sociais ligadas à visão, à escrita, à técnica, à lógica. Embora a Música seja a atividade proeminente do *live coding*, não obtivemos resultados que retornassem, por exemplo, a palavra *hearing*.

Adjetivos destacam características da prática. *Live* é a palavra-chave, e sugere uma prática de performance. *Visual* sugere uma característica fundamental, tanto quanto a Música, para uma performance. *Ensemble* destaca a natureza de grupos, isto é, poucas performances *solo* são realizadas se comparadas às performances de *duos*, *trios*.

Palavras como *university*, *research* e *technology*, e *laptop* acusam não apenas uma prática artística, mas um Programa de Investigação Científica. A esfera de pesquisa acadêmica permitiu ramos de desenvolvimento com linguagens de programação, cognição, inteligência artificial, semiologia, performance musical (improvisação), e mais recentemente, antropologia, conferindo à produção de *live coding* espécie de autenticidade acadêmica.

² Disponível em <<http://gibber.mat.ucsb.edu>>

APÊNDICE B – Sugestões Proto-históricas

Descrevemos neste capítulo um trabalho de Mathews e Moore (1970), GROOVE, ainda pouco observado por improvisadores de códigos e a atuação de uma tecnologia, a compilação JIT (AYCOCK, 2003) como um sujeito sócio-técnico fundamental para que o *live coding* fosse possível.

B.1 GROOVE

GROOVE, ou *Generated Real-time Operations On Voltage-controlled Equipment* foi um computador desenvolvido na Bell Labs por (MATHEWS; MOORE, 1970). Alex Nunzio (2010) discute como um precedente direto da família de softwares MUSIC N¹. É o primeiro de trabalho de Mathews com reflexões nos aspectos performáticos. Não foi usado para ambientes de performance, mas a peça *The expanding universe* da compositora Laurie Spiegel (1975) foi considerada como exemplar por sua execução instrumental e disponibilidade *online* (ver a seguir). Seu desenvolvimento iniciou em 1968 na *Bell Labs*. Segundo o próprio Mathews, o funcionamento do sistema oferece algumas possibilidades a partir de três conceitos: *criação, retroalimentação e ciberificação*. O primeiro conceito foi implementado com um sistema de arquivos, onde as funções criadas no processo criativo são memorizadas, e podem ser editadas. O segundo conceito se relaciona com o terceiro:

O GROOVE provê oportunidades para uma retroalimentação imediata de observações dos efeitos das funções temporais para as entradas do computador, que compõem a função. No modo de composição do sistema GROOVE, um ser humano está em um ciclo de retroalimentação, como mostrado na figura 1 [Figura 32]. Assim ele é capaz de modificar as funções instantâneamente como um resultado de suas observações daqueles efeitos (MATHEWS; MOORE, 1970, p. 715)².

O terceiro conceito observa a existência de uma relação entre um humano e uma máquina. Mathews descreve-o como uma *engenharia humana*. Esta engenharia consistiu na observação de um tempo diferencial entre o que o(a) musicista cria e o que edita:

O conceito final é mais nebuloso. Desde que o GROOVE é um sistema homem-máquina, a engenharia humana do sistema foi a mais importante. Por exemplo, nós descobrimos que o controle do programa de tempo

¹ Desenvolvidos a partir de 1957. As versões softwares MUSIC I, II, III, IV, IV-B, IV-BF, V (que passou por modificações no IRCAM), MUSIC 360, MUSIC 11 acarretaram no desenvolvimento do software CSound, disponível em <<https://csound.github.io/>>.

² Tradução de *GROOVE provides opportunity for immediate feedback from observations of the effects of time functions to computer inputs which compose the function. In the compose mode of the GROOVE system, a human being is in the feedback loop (...) Thus he is able to modify the functions instantaneously as a result of his observations of their effects.*

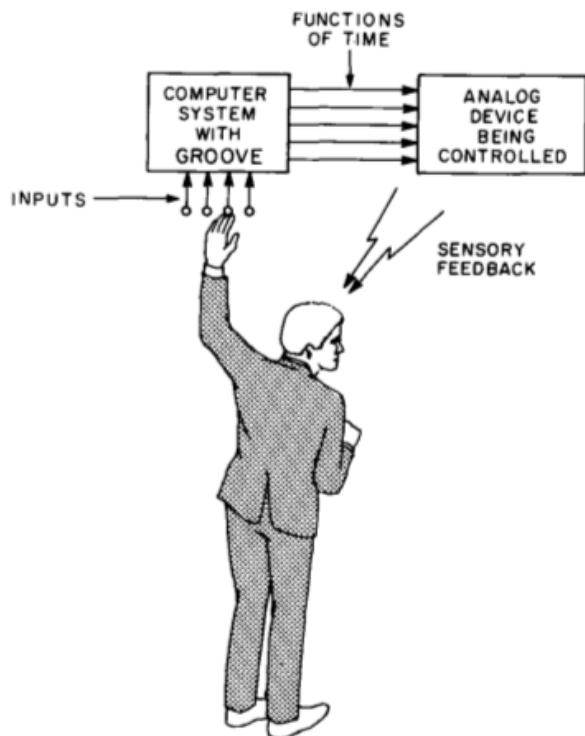


FIG. 1. Feedback loop for composing functions of time

Figura 32 – Esquema de concepção do projeto GROOVE. **Fonte:** (MATHEWS; MOORE, 1970).

necessita ser bastante diferente para a composição do que para a edição, e o programa foi modificado de acordo. (...) O intérprete de computador não deve tentar definir todo o som em tempo real. Ao invés, o computador deve ter uma partitura e o intérprete deve influenciar a forma como a partitura é tocada. Seus modos de influência podem ser mais variados do que aqueles que um regente convencional, que pode principalmente controlar o tempo, intensidade, e estilo (MATHEWS; MOORE, 1970, p. 715-716)³.

Como exemplo, selecionamos uma descrição da compositora Laurie Spiegel (1975) (ver Figura 33) para sumarizar as características do GROOVE, durante a produção de *The Expanding Universe*⁴, entre as salas 2D-506 da Bell Labs (contendo o computador DDP-224) e a sala analógica 2D-562 (laboratório de Mathews). A “performance” da obra era realizada, com a programação de funções temporais e a manipulação de parâmetros dessas funções através de dispositivos físicos:

³ Tradução de *The final concept is more nebulous. Since GROOVE is a man-computer system, the human engineering of the system is most important. For example, we discovered that the control of the program time needs to be quite different for composing than for editing, and the program was modified accordingly. (...) The computer performer should not attempt to define the entire sound in real-time. Instead, the computer should have a score and the performer should influence the way in which the score is played. His modes of influence can be much more varied than that a conventional conductor who primarily controls tempo, loudness, and style.*

⁴ Disponível em <<https://www.youtube.com/watch?v=dYUZmsfm4Ww>>.

Todas as músicas no GROOVE eram representadas na memória digital como funções abstratas do tempo, séries paralelas de dois pontos, cada ponto sendo um instante no tempo e um valor instantâneo. A taxa de amostragem para essas funções, usada principalmente como controle de voltagem, era cronometrada por um grande e antiquado oscilador analógico que era normalmente fixado em 100 Hertz, cada ciclo do oscilador pulsando à frente do código, o computador lia, em cada uma das funções, naquele ponto do tempo, todos dispositivos de entrada e executava todas amostras. (...) Tínhamos uma pequena caixa com 4 potenciômetros e quatro chaves (alternadores fixados onde você os colocava) e dois botões de disparo.⁵



Figura 33 – Laurie Spiegel configurando a saída analógica do GROOVE, durante a produção de *The Expanding Universe*. **Fonte:** ([SPIEGEL, 1975](#)).

Embora não declare ser uma peça minimalista, a descrição de *The Expanding Universe* considera, de maneira asséptica, os fenômenos psicoacústicos como elementos compostionais. Por exemplo, a utilização da continuidade progressiva de sons (ou *drones* transitórios) como elemento criativo permite, segundo a compositora, à sensibilização do ouvido, o que não seria possível na música minimalista instrumental:

⁵ Tradução de *All music in GROOVE was represented in digital memory as abstract functions of time, parallel series of point pairs, each point being an instant in time and an instantaneous value. The sampling rate for these functions, which would be used mostly as control voltages, was clocked by a big old-fashioned analog oscillator that was usually set to 100 Hertz, each cycle of the oscillator pulsing one run through the code, the computer reading all of the real time input devices and playing of all of the samples at that time point in each of the time functions. (...) We had a small box with 4 knobs, 4 set switches (toggles that stay where you put them) and 2 momentary-contact push buttons on it.*

A violência da perturbação sonora, disjunção, descontinuidade e mudanças súbitas desanitizam o ouvinte e nos afastam, de forma que não estamos mais abertos aos sons mais sutis. Mas com continuidade e gentileza, o ouvido se torna re-sensibilizado para mais e mais fenômenos auditórios sutis dentro do som que estamos imersos. Em vez de sermos arrastados, como nas cascatas de muitas notas executadas em blocos de tempo que mudam repentinamente, tal como tantas vezes consiste a música "minimalista", abrimos nossos ouvidos mais e mais para os fenômenos que nos envolvem. Isto também não é música ambiente, um termo que veio a ser usado alguns anos depois. Esta é música para atenção concentrada, uma experiência musical do através, pensando que, lógico, existe também um pano de fundo.⁶

Nesta citação podemos sumarizar um conceito para o *live coding*: Música como um Processo Gradual Cf. REICH, 1968. Porém, o significado de processo pode ser desenvolvido infinitamente. Isso não será realizado. O Processo para Spiegel é diverso daquele considerado no *live coding*, e uma digressão desta pode afastar demais o foco do trabalho principal. Para compreensão deste termo, será necessário explorar outros aspectos correlacionados no decorrer deste trabalho.

Para finalizar esta sessão, a figura Figura 33 sugere um conceito rotineiro para o *live coder*. Esta rotina é uma atividade constante de improvisação códigos para aquisição de destreza para uma performance. Iazzetta (2009), Soares (2015-03-13) lembram que esta atividade, de codificar como se construísse um instrumento musical, se caracteriza por sua conexão com a esfera composicional, nomeado *luteria composicional*.

B.2 SuperCollider

O SuperCollider⁷ é um ambiente de programação desenvolvido por James McCartney, lançado em 1996. Segundo McCartney (1996),

"SuperCollider começou como dois programas separados que eu escrevi. O primeiro foi um programa chamado *Synth-O-Matic* que era um sintetizador de tempo diferido, escrito de forma semelhante à linguagem C, para Macintosh em 1990 e foi abandonado. O segundo era um objeto *MAX* chamado *Pyrite* que continha um interpretador para a linguagem que se estendeu e foi usado no SuperCollider. Escrever o SuperCollider envolveu integrar uma linguagem interpretada, um coletor de lixo [administração automática da memória], e uma biblioteca de funções do Pyrite com uma máquina de síntese e funções do Synth-O-Matic. Eu gostaria de

⁶ Tradução de *The violence of sonic disruption, disjunction, discontinuity and sudden change desensitizes the listener and pushes us away so we are no longer open to the subtlest sounds. But with continuity and gentleness, the ear becomes increasingly re-sensitized to more and more subtle auditory phenomena within the sound that immerses us. Instead of being swept along, as with cascades of many running notes in suddenly-changing blocks of time, such as “minimalist” music so often consists of, we open up our ears more and more to the more minute phenomena that envelope us. This is also not “ambient music”, a term that came into use some years later. This is music for concentrated attention, a through-composed musical experience, though of course it also can be background.*

⁷ Disponível em <<https://superollider.github.io/>>

agradecer ao Curtis Roads por encorajar-me a reviver o programa do Synth-O-Matic, que levou ao SuperCollider.”⁸

Uma outra perspectiva é oferecida como uma possibilidade lógica e enxuta de outros paradigmas de programação musical. McCartney (1996, p. 1), descreve alguns problemas com o paradigma de programação musical desenvolvido a partir do Music N (MATHEWS, 1963), como por exemplo, a uma estrutura estática inerente à concepção de objetos conectados por cabos:

“As abstrações fornecidas pelas linguagens MUSIC N, incluindo o CSound, são abstrações de unidades geradoras, o laço de computação para a amostra de áudio, a representação de conexões entre unidades geradoras, e instanciação e desalocação de instrumentos. Estas abstrações tornaram a escrita de algoritmos de processamento de sinais mais fácil, porqueles abstraem um número de detalhes incômodos. Contudo, a família Music N provê poucas estruturas de controle, nenhuma estrutura de dados reais, e nenhuma função de usuário. SAOL melhora o paradigma do Music N provendo tipos de abstrações encontradas na linguagem C, tais como estruturas de controle, funções e algumas estruturas de dados. Max, que é um tipo diferente de linguagem de programação, provê um conjunto interessante de abstrações que permite muitas pessoas usá-lo sem perceberem que estão programando acima de tudo. (...) A linguagem Max também é limitada em sua habilidade de tratar seus objetos como dados, o que torna uma estrutura de objetos estáticos. Evoluções posteriores do Max, como o jMax, e o Pd, fazem várias coisas para expandir as limitações de estrutura de dados do Max, mas ainda assim possuem uma estrutura estática de objeto.”⁹

McCartney discute adiante um modelo alternativo de notação musical, adaptado aos padrões de uma linguagem que expresse comportamentos musicais, ao invés da determinação de pontos fixos de parâmetros musicais:

“Uma linguagem musical de computador deve prover um conjunto de abstrações que expressam idéias composicionais e de processamento

⁸ Tradução nossa de “*SuperCollider began as two separate programs that I wrote. The first was a program called Synth-O-Matic which was a non-real-time C-like synthesis programming language for the Macintosh written in 1990 and abandoned. The second was a MAX object called Pyrite which contained the interpreter for the language which was extended and used in SuperCollider. Writing SuperCollider involved integrating the language interpreter, garbage collector and function library of Pyrite with the synthesis engine and functions of Synth-O-Matic. I'd like to thank Curtis Roads for encouraging me to revive the Synth-O-Matic program which ultimately led to SuperCollider.*”

⁹ Tradução nossa de “*The abstractions provided by the Music N languages, including Csound (www.csounds.com), are the abstraction of a unit generator, the audio sample computation loop, the representation of the connections between unit generators, and instrument instantiation and deallocation. These abstractions make writing signal-processing algorithms easier, because they abstract a number of cumbersome details. However, the Music N family provides few control structures, no real data structures, and no user functions. SAOL (www.saol.net) improves the Music N paradigm by providing the kinds of abstractions found in the C language, such as control structures, functions, and some data structures. Max (www.cycling74.com/products/maxmsp.html), which is quite a different kind of programming language, provides an interesting set of abstractions that enable many people to use it without realizing they are programming at all. (...). The Max language is also limited in its ability to treat its own objects as data, which makes for a static object structure. Later evolutions of Max, such as jMax (www.ircam.fr/produits/logiciels/log-forum/jmax-e.html) and Pd (www.pure-data.org), do various things to expand the data structure limitations of Max but still have a generally static object structure.*”

de sinais da maneira mais fácil e direta possível. Os tipos de idéias que alguém pode expressar, contudo, podem ser diferentes e levar para diferentes ferramentas. Se alguém interessado em realizar uma partitura que represente uma peça musical como um artefato fixado, então o modelo de partitura/orquestra será suficiente. Motivações que levaram a projetar o SuperCollider estavam na habilidade de perceber processos sonoros que são diferentes, a cada vez que eles são tocados, para escrever peças que de alguma forma descrevem um campo de possibilidades ao invés de uma entidade fixa, e o que facilita a improvisação ao vivo por um compositor/executante.”¹⁰

Um exemplo de uso pode ilustrar o discurso do McCartney. O exemplo abaixo (p. 119) é um código de Fredrik Olofson, outro personagem importante para a improvisação de códigos (WARD et al., 2004). Este exemplo é peculiar, uma vez que expõe não somente estruturas dinâmicas e deterministas, mas possibilita criar uma conexão com dois assuntos discutidos anteriormente, algoraves (ver subseção 1.2.1, p. 8) e microchips (ver ??, p. ??). Um sintetizador recria o timbre do videogame *Atari2600* (laçado no EUA em 1977); mais especificamente é um simulador do *chip TIA* (*Television Interface Adapter*), responsável pela geração de gráficos e imagens no videogame Atari¹¹. O exemplo é relativamente simples. Um sintetizador (*SynthDef*) e um sequenciador (*Pbind*) são definidos. O sequenciador controla parâmetros como os tons, frequências de modulação e panoramização do sintetizador. É interessante notar que padrões fixos (*Pseq* e *Pn*) se misturam com padrões variáveis (*Pbrown*) e segue padrões de movimentos brownianos diferentes que variam entre 28 e 31 Hz, alternados com 23 e 26 Hz. Enquanto isso a frequência moduladora segue um padrão repetitivo que alterna 10 e 16 Hz com 11 e 16 Hz.

Exemplo B.1 (Notação do SuperCollider)

Fonte: <<http://supercollider.sourceforge.net/audiocode-examples/>>

```
// Simple synth definition using the Atari2600 UGen:
(
SynthDef(\atari2600, { |out= 0, gate= 1, tone0= 5,
tone1= 8, freq0= 10, freq1= 20, amp= 1, pan= 0|
  var e, z;
  e= EnvGen.kr(Env.asr(0.01, amp, 0.05), gate, doneAction:2)
  ;
  z= Atari2600.ar(tone0, tone1, freq0, freq1, 15, 15);
  Out.ar(out, Pan2.ar(z*e, pan));
}
```

¹⁰ Tradução nossa de “A computer music language should provide a set of abstractions that makes expressing compositional and signal processing ideas as easy and direct as possible. The kinds of ideas one wishes to express, however, can be quite different and lead to very different tools. If one is interested in realizing a score that represents a piece of music as a fixed artifact, then a traditional orchestra/score model will suffice. Motivations for the design of SuperCollider were the ability to realize sound processes that were different every time they are played, to write pieces in a way that describes a range of possibilities rather than a fixed entity, and to facilitate live improvisation by a composer/performer.”

¹¹ Disponível em <https://www.atariage.com/2600/archives/schematics_tia/index.html>

```

}).store
)

// And a pattern to play it:
(
Pbind(
  \instrument, \atari2600,
  \dur, Pseq([0.25, 0.25, 0.25, 0.45], inf),
  \amp, 0.8,
  \tone0, Pseq([Pseq([2, 5], 32), Pseq([3, 5], 32)], inf),
  \tone1, 14,
  \freq0, Pseq([Pbrown(28, 31, 1, 32), Pbrown(23, 26, 3, 32)
    ], inf),
  \freq1, Pseq([Pn(10, 16), Pn(11, 16)], inf)
).play
)

```

B.2.1 Just In Time Library (JITLib)

A reflexividade (ver [seção 2.1](#), p. 27) é uma característica de diversos ambientes de *live coding*. Segundo [Aycock \(2003\)](#), os primeiros programas JIT foram Genesis (com base no LISP, 1960), LC² (*Language for Conversational Computing*, 1968) e APL (1970). Este último deu origem ao conceito *lazy evaluation* (avaliação preguiçosa).

O *SuperCollider* foi o primeiro dos ambientes de programação musical a implementar esta característica. Com a divulgação da biblioteca JITLib¹², os primeiros Espaços Conceituais do *live coding* se estruturaram de maneira bastante formal na comunidade de músicos-programadores. Isto é, durante o ato de codificação podemos codificar a execução de um som antes mesmo de definí-lo (ver [Notação do SuperCollider B.2](#)).

Exemplo B.2 (Exemplo de avaliação preguiçosa no *SuperCollider*.)

Um tipo de variável específica começa com o caractere ~ para indicar um ambiente propício para a avaliação preguiçosa. Mesmo antes de sua definição, podemos tocar um sintetizador:

```

// play some output to the hardware busses,
// this could be any audio rate key.
~out.play;
~out = { SinOsc.ar([400, 408] * 0.8, 0, 0.2) };

```

¹² Disponível em <<http://doc.sccode.org/Overviews/JITLib.html>>.

Depois que o código acima é escrito e executado, podemos escrever outros códigos para substituir o sintetizador durante sua execução (*runtime*):

```
// replacing the node.
// the crossfade envelope is created internally.
~out = { SinOsc.ar([443, 600 - Rand(0,200)], 0, 0.2) };
~out = { Resonz.ar(Saw.ar(40 + [0,0.2], 1), [1200, 1600], 0.1)
    + SinOsc.ar(60 * [1,1.1],0,0.2) };
~out = { Pan2.ar(PinkNoise.ar(0.1), LFClipNoise.kr(2)) };
```

Fonte: <http://doc.sccode.org/Tutorials/JITLib/proxyspace_examples.html>

Outros *softwares* e ambientes também merecem menção: *ixiLang*¹³, *ChucK*¹⁴, *Extempore*¹⁵, *Impromptu*¹⁶, *SonicPi*¹⁷

Esta técnica têm sido largamente implementada em navegadores de internet (ROBERTS; WAKEFIELD; WRIGHT, 2013), ou remotamente (Junior; Lee; Essl, 2015). Isto é, entre duas pessoas distantes uma da outra, mas concetadas através da *internet* ou de redes privadas.

Trabalhos neste caminho incluem o *Gibber*¹⁸ e *Wavepot*¹⁹. Com base neste último, implementamos um ambiente chamado *Termpot*²⁰.

¹³ Disponível em <<http://www.ixi-audio.net/ixilang/>>

¹⁴ Disponível em <<http://chuck.cs.princeton.edu/>>.

¹⁵ Disponível em <<http://benswift.me/extempore-docs/>>.

¹⁶ Disponível em <<http://impromptu.moso.com.au/>>

¹⁷ Disponível em <<http://sonic-pi.net/>>

¹⁸ Disponível em <<http://gibber.mat.ucsb.edu/>>. Cf. ROBERTS; KUCHERA-MORIN, 2012

¹⁹ Disponível em <<https://www.wavepot.com>>.

²⁰ Disponível em <<https://jahpd.github.io/termpot>>. Cf. LUNHANI; SCHIAVONI, 2015.