

Guilherme Martins Lunhani

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

Guilherme Martins Lunhani

Live Coding: um algoritmo gerador de uma sonoridade tonal no A Study in Keith (2009) de Andrew Sorensen

Dissertação apresentada ao Programa de Pós-Graduação em Artes, Cultura e Linguagens, Área de Concentração: Teorias e Processos Poéticos Interdisciplinares, Linha de pesquisa: Artes Visuais, Música e Tecnologia, da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do grau de Mestre

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

Ficha catalográfica elaborada através do programa de geração automática da Biblioteca Universitária da UFJF, com os dados fornecidos pelo(a) autor(a)

Lunhani, Guilherme Martins.

Live Coding : um algoritmo gerador de uma sonoridade tonal em A Study in Keith (2009) de Andrew Sorensen / Guilherme Martins Lunhani. -- 2016.

84 f.

Orientador: Luiz Eduardo Castelões

Coorientadores: Alexandre Sperandéo Fenerich, Flávio Luiz Schiavonni

Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Artes e Design. Programa de Pós Graduação em Artes, Cultura e Linguagens, 2016.

1. Improvisação de códigos. 2. Música Computacional. I. Castelões, Luiz Eduardo, orient. II. Fenerich, Alexandre Sperandéo, coorient. III. Schiavonni, Flávio Luiz, coorient. IV. Título.

*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. 3ª edição. Editorial Attar. Ver também <<http://tao-te-king.org/42.htm>>.

Agradecimentos

Maiores agradecimentos a Deus.

Para uma família, pelo apoio, Jair, Olímpia e Júlia.

Aos moradores de rua de Juiz de Fora que deram sentido à fraqueza da pergunta deste trabalho.

Aos Professores Dr. Luiz Eduardo Castelões, Dr. Alexandre Fenerich e Dr. Flávio Luiz Schiavonni, fundamentais no apoio institucional, na sugestão de leituras acadêmicas e extra-acadêmicas, na cobrança de prazos, nas críticas e nas conversas sobre Música e Computadores.

À FAPEMIG por suprir a lacuna financeira em um momento delicado nas economias das Universidades Federais.

Simone pelo Amor, e pela filosofia de Espinosa.

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

Aos *freakcoders* Daniel Penalva, Renato Fabbri e Vilson Vieira.

Alguém poderia imaginar uma interface musical na qual um músico especifica o som resultante desejado, em uma linguagem descritiva, na qual poderia ser então traduzida em parâmetros de partículas e renderizados em som. Uma alternativa poderia especificar um exemplo: "Faça um som assim (arquivo de som), mas com pouco vibrato"

Curtis [Roads](#) (2001)

Resumo

Este documento discute uma versão sintetizada de uma técnica polivalente cujo nome é *live coding*, suas construções históricas na Música, e uma simulação de improvisação tonal guiada por improvisação com linguagens de programação.

Na Introdução (ver p. [xv](#)) apresentamos uma definição de *live coding*. A definição destaca o fazer musical, mas não exclui outras potências artísticas.

No [Capítulo 1](#) (ver p. [1](#)) destacamos um mecanismo criativo desta técnica em dois contextos não musicais.

No [Capítulo 2](#) (ver p. [13](#)) listamos períodos de atividades musicais que prototiparam e formalizaram o mecanismo criativo do primeiro capítulo.

No [Capítulo 3](#) (ver p. [31](#)) analisamos uma proposição musical, um vídeo intitulado *A Study in Keith* de [Sorensen e Swift \(2009\)](#), de acordo com o mecanismo mental do primeiro capítulo.

A contribuição deste trabalho para a musicologia brasileira é a organização historiográfica de uma técnica ainda pouco elaborada em português.

Palavras-chaves: Improvisação de códigos; Música Computacional; *A Study in Keith*.

Abstract

This document presents a synthesized version of a versatile technique whose name is live coding, its historical buildings in music, and a simulation of a tonal improvisation, guided by improvisation with programming languages.

In the Introduction (see p. [xv](#)), we present a definition of live coding. The definition highlights a focus on music, but does not exclude other artistic powers.

In Chapter 1(see p. [1](#)), we highlight a creative mechanism of this technique in two unmusical contexts.

In Chapter 2 (see p. [13](#)), we listed periods of musical activities that prototyped and formalized the creative engine of the first chapter.

In Chapter 3 (see p. [31](#)), we analyzed a musical proposition, a Sorensen and Swift's video entitled *A Study in Keith* (2009), according to the first mental mechanism chapter.

The contribution of this work to the Brazilian musicology is a historiographical organization of a technique still little developed in portuguese.

Key-words: Live Coding; Computer Music; *A Study in Keith*.

Sumário

	Introdução	xv
1	DEFINIÇÕES DE BASE PARA UMA IMPROVISAÇÃO DE CÓDIGOS	1
1.1	Tecendo códigos	2
1.1.1	Slub	4
1.2	Dança	7
1.2.1	Hacking Coreography	8
1.2.2	Hacking the Body	10
1.3	Discussão	11
2	DEFINIÇÕES HISTÓRICAS DA IMPROVISAÇÃO DE CÓDIGOS	13
2.1	Pietro Grossi	13
2.1.1	Reflexividade	14
2.1.2	Telemática	16
2.2	Baía de São Francisco	17
2.2.1	The League of Automatic Composers	18
2.2.2	The Hub	20
2.3	Ron Kuivila	22
2.4	Live coding	22
2.4.1	LAPTOP	23
2.4.2	TOPLAP	24
2.4.3	<i>Show us your screens</i>	25
2.5	Discussão	28
3	ESTUDO DE CASO	31
3.1	Metodologia de Análise	31
3.1.1	O modelo de improvisação	33
3.2	<i>A Study in Keith: Proposição</i>	34
3.3	Referentes Opcionais	34
3.3.1	Concertos Sun Bear	35
3.3.2	NI-Akoustik Piano	37
3.3.3	Ambiente e Linguagem: Impromptu	38
3.3.4	Extempore	39
3.3.5	Scheme	39
3.4	Blocos de Eventos	45

3.4.1	Definição do instrumento e do tempo	46
3.4.2	Definição de uma sequência de blocos	47
3.4.3	Definição de blocos	48
3.4.4	Primeira sonoridade tonal	49
3.4.5	Regras de qualidade	51
3.5	Primeira sequência de blocos de eventos	52
3.5.1	Primeiro Bloco	52
3.5.2	Segundo Bloco	53
3.5.3	Terceiro Bloco	54
3.6	Discussão	57
	Conclusão	59
	Referências	61

Lista de ilustrações

Figura 1 – O Processo de ação e reação no desenvolvimento de programas <i>guiados por</i> bricolagem. Substituímos, ao longo deste trabalho, a palavra <i>concept</i> (conceito) por <i>proposição</i> para não confundirmos questões ontológicas. Fonte: (MCLEAN, 2011, p. 122), grifo nosso.	1
Figura 2 – Tecido resultante da prática <i>Weaving code</i> . Fonte: Griffiths (2015a).	3
Figura 3 – Performance no Foam Kernow. Fonte: Griffiths (2015b).	3
Figura 4 – Alex McLean manipulando uma Matriz de botões para tecelagem, conectado a um Raspberry Pi. Fonte: Griffiths (2015b).	4
Figura 5 – Dançarina (anônima) controlada por Kate Sicchio (2015) através de uma codificação improvisada. Fonte: < https://www.youtube.com/watch?v=uAq4BAbvRS4 >.	11
Figura 6 – Sistema de música computacional de John Bischof <i>circa</i> 1980. Foto: Eva Shoshanny ² . Fonte: Brown e Bischof (2002).	18
Figura 7 – Circuito do computador caseiro dedicado à síntese sonora de Tim Perkis, usado no começo dos anos 1980. Foto: Eva Shoshanny ³ . Fonte: Brown e Bischof (2002)	20
Figura 8 – Definição do significado de TOPLAP. Fonte: Ramsay (2010).	25
Figura 9 – Transcrição do motivo gerador do disco Kyoto, parte 1. Fonte: Uwe Karcher (2009).	36
Figura 10 – Distribuição, aproximada, de probabilidades de acontecimento com um conjunto de possíveis cadências tonais organizados como uma cadeia de Markov. Fonte: Swift (2012).	45
Figura 11 – Transcrição literal e perceptiva do primeiro evento em <i>A Study in Keith</i> . Fonte: autor.	50
Figura 12 – Primeiros eventos musicais gerados a partir das primeiras estruturas válidas de código. Fonte: autor.	52
Figura 13 – Segundo bloco de eventos musicais. Fonte: autor.	53
Figura 14 – Terceiro bloco de eventos musicais. Fonte: autor.	55
Figura 15 – Definição de <i>live coding</i> : “Insira a definição aqui”. Fonte: Collins (2014).	60

Introdução

No contexto das Músicas produzidas com computadores no final do século vinte e no começo do século vinte e um, discutimos uma técnica que recorre às linguagens textuais de programação para performances artísticas (MCLEAN, 2011). Artistas-programadores (e até mesmo técnicos profissionais) chamam-na de *live coding*, de forma que o antropólogo Giovanni Mori (2015a, p. 117) define-a como uma técnica polivalente:

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 notável.⁴

Mori enfatiza o *live coding* – *codificação ao vivo* ou *improvisação de códigos* – como foco de proposições musicais. Durante a pesquisa, fomos desafiados a encarar uma técnica que permite produzir, algumas vezes ao mesmo tempo, Música Eletrônica para Dançar, Música-Erro, Música-Ruído, Jazz, e como será possível observar no primeiro capítulo, performances corporais, visuais e têxteis. Dividir os capítulos deste documento exigiu recortar a definição do pesquisador italiano, com exemplos de Tecelagem e Dança como pertinentes para as definições de base, e Música como discussão histórica e de estudo de caso. Exemplos audiovisuais podem ser tão numerosos quanto os exemplos musicais, de forma que não serão incluídos integralmente, no máximo, como elemento adjacente.

Capítulo 1

Tecelagem

Contextualizamos a improvisação de códigos a partir da tecelagem. Nas palavras do improvisador de códigos Dave Griffiths (2015a), a tecelagem possibilita a compreensão dos mecanismos de programação, e dos significados dos códigos de computador:

Um dos potenciais da tecelagem que eu fiquei mais interessado é a capacidade de demonstrar fundamentos de *softwares* por tecidos – 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 computadores⁵.

⁴ Tradução 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 prominente.*

⁵ Tradução 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.*

Por outro lado, descrever uma técnica de improvisação direcionada para a prática de tecelagem é uma forma de costurar Dança e Música. Griffiths, ao lado de Alex McLean e Adrian Ward, realizam, no começo dos anos dois mil, apresentações diversas do *live coding* em terreno inglês com um formato *geek* de Música Eletrônica para Dançar: programação, *VJing/DJing* e ambientes *rave* com sonoridade *happy hardcore*.

Dança

Contrapondo este cenário, apresentamos um trabalho da coreógrafa Kate [Sichio \(2015\)](#) que merece algum destaque, ao negar o som como resultado da improvisação. Através de improvisações de códigos de uma coreógrafa, os movimentos corporais de outra mulher são controlados nos níveis conceitual e corpóreo.

É importante mencionar que esta coreografia nasce de uma bricolagem de uma partitura de performance, no escopo da pesquisa de Sichio, como uma assimilação de *Sensibilidades Computacionais*, ou dispositivos metafóricos elaborados por coreógrafos como Merce Cunningham, Trisha Brown, Bill T. Jones, e William Forsythe.

Capítulo 2

Proto-História

Em fóruns de internet, improvisadores de códigos discutem a origem do *live coding* como técnica polivalente. Entre elas, obras audiovisuais de Tom de Fanti, em 1976 ⁶.

Um consenso da origem na Música destaca a performance *Water Surfaces* do compositor estadunidense Ron Kuivila ([WARD et al., 2004](#)). Uma desconstrução desta idéia, feita por Giovanni Mori, sugere que o compositor italiano Pietro Grossi elaborou, no começo dos anos 1970, as primeiras experiências formais com um paradigma menor da *Computer Music*, em contraste com aquele paradigma maior formado pela divulgação da família MUSIC N de [Mathews \(1963\)](#), [Mathews et al. \(1969\)](#). Outros paradigmas menores também são formados através da *Live Computer Music* da Baía de São Francisco, durante o final da década de 1970, e meados da década de 1980, com o grupo *The League of Automatic Composers*, embrião de outro, *The Hub*. Este paradigma menor considerava a realização de *shows* (algumas vezes *happenings*) em pequenos estabelecimentos públicos, cuja premissa era a programação de microcontroladores que trocassem informações composicionais. Em outras palavras, a composição colaborativa de compositores automáticos.

⁶ Disponível em <http://lurk.org/groups/livecode/messages/topic/5abPazJSxfegYfVFOzN4T6/>>

Atualidade

O manifesto *Lubeck04* ou *Show us your screens* é um pequeno texto que estabelece algumas regras de conduta para o artista-programador em situação de performance. Rascunhado em um ônibus por Alex McLean, foi formalizado em um documento maior intitulado *Live algorithm programming and temporary organization for its promotion*, ou simplesmente LAPTOP (WARD et al., 2004), como uma resposta conjunta de sete artistas-programadores ingleses ao artigo *Using contemporary technology in live performance; the dilemma of the performer*, publicado por Schloss (2003). Estas regras são até hoje utilizadas como condutor técnico para uma improvisação de códigos, seja ela musical, audiovisual, corporal ou têxtil.

Capítulo 3

A Study in Keith (2009)

Selecionamos *A Study in Keith* (SORENSEN; SWIFT, 2009; SORENSEN, 2015) como um caso que possibilitou investigar com maior profundidade o momento de elaboração de sua proposição, e sua primeira codificação em texto como um algoritmo executável.

O principal registro audiovisual desta improvisação de códigos aponta uma proposição que será o foco do capítulo: após a escuta dos Concertos *Sun Bear* do pianista e compositor Keith Jarret, Andrew Sorensen improvisa um código com o ímpeto de automatizar uma improvisação pianística, semelhante a um Jazz, com eventos MIDI. Segundo Sorensen, “**Não é Keith, mas inspirado por Keith**”.

1 Definições de base para uma improvisação de códigos

Uma improvisação de códigos supõe ao menos um(a) artista-programador(a) capaz de reagir a um ou mais estímulos, e de codificá-lo(s) em caracteres textuais. Segundo Giovanni Mori (ver [Capítulo](#) , p. xv), estes caracteres indicam como sons, imagens, cores, movimentos e tecidos são projetados, movimentados ou tramados para quem vê o(a) artista-programador(a). Este momento que transita entre a elaboração da imaginação, e a codificação textual do que acontecerá, será o norteador deste documento. Ele é ilustrado no pequeno ciclo da [Figura 1](#):

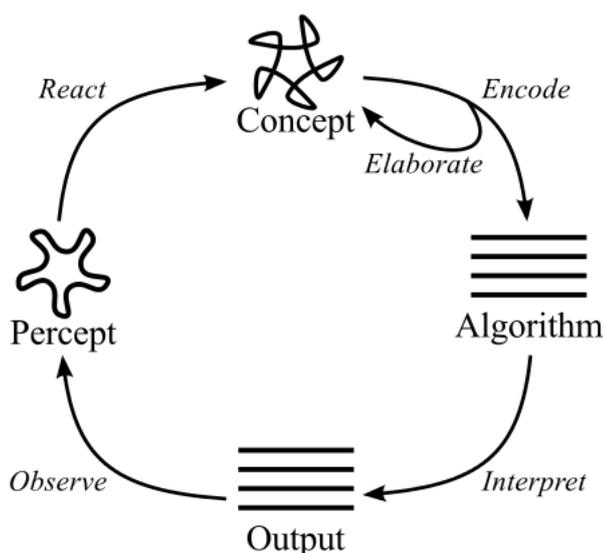


Figure 6.2: The process of action and reaction in bricolage programming

Figura 1 – O Processo de ação e reação no desenvolvimento de programas *guiados por* bricolagem. Substituímos, ao longo deste trabalho, a palavra *concept* (conceito) por *proposição* para não confundirmos questões ontológicas. **Fonte:** (MCLEAN, 2011, p. 122), grifo nosso.

Começamos a investigação deste ciclo com um tipo específico de improvisação de códigos, o *weavecoding* (ver [seção 1.1](#), p. 2). Através de uma atividade não-musical, buscamos ilustrar uma proposição e o código-fonte resultante. Costuramos este fio condutor com a Música Eletrônica para Dançar (ver [subseção 1.1.1](#), p. 4), e fechamos com uma proposição coreográfica (ver [seção 1.2](#), p. 7).

1.1 Tecendo códigos

O *weavingcode* será definido conforme apresentamos o grupo *Weaving codes*¹, formado para a “(...) investigação de padrões a partir das perspectivas de tecelagem e música, e através do desenvolvimento de uma linguagem de computador e código para descrever a construção de tecidos”². É formado por membros das Universidades de Leeds, Nottingham Trent, Cambridge, Aberdeen, Copenhagen; um museu (*Albert Museum*), uma rede de laboratórios transdisciplinares (FoAM Kernow); o Centro Dinamarquês para Pesquisa Têxtil, e Escola Robert Schumman de Música e Mídia de Düsseldorf.

Um de seus membros e um dos principais articuladores do *live coding* inglês, Dave Griffiths (2015a) descreve a seguinte proposição têxtil: com fios de tecido, são repetidas ações de dar a volta em pontos, pela frente, ou por trás do fio, afim de produzir uma figura geométrica. Estas ações são descritas em linguagem *Scheme* a partir de quatro palavras-chave: *repeat*, uma repetição de ações por contagem, ou laço iterativo (*loop*); *twist*, ou dar a volta em determinados pontos; *weave-forward*, tecer o ponto alto; e *weave-back*, tecer o ponto baixo. Do lado direito da imagem (ver p. 3), é simbolizado o código compactado do tecido, ou as operações fundamentais para um determinado padrão têxtil. Do lado esquerdo, seu resultado, uma textura de losangos e zigue-zagues.

Exemplo 1.1 (Um código-fonte que gera um tecido semelhante à Figura 2.)

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

¹ Disponível em <<http://kairotic.org/about/>>

² 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*”

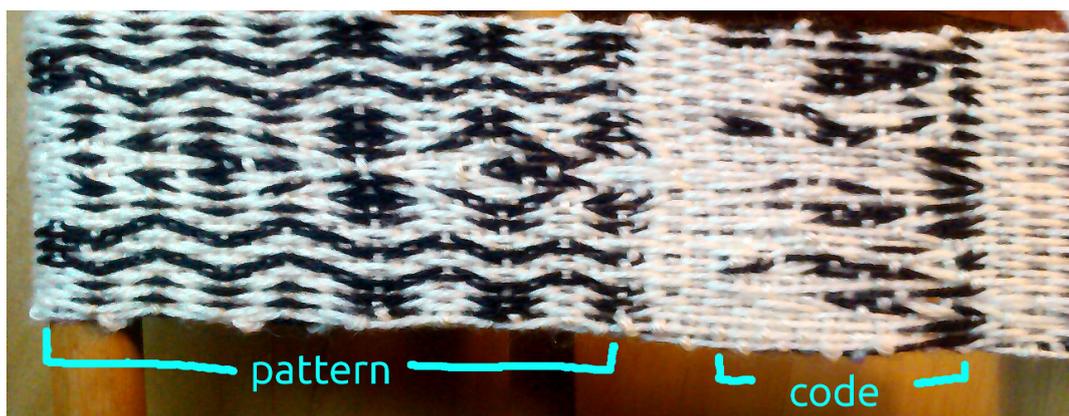


Figura 2 – Tecido resultante da prática *Weaving code*. Fonte: Griffiths (2015a).

Esta experiência de *participação social*³ foi co-orientada por Griffiths como uma *sessão de improvisação*: programadores escrevem, enquanto observam e ouvem os resultados projetados por superfícies planas, caixas de som e máquinas de tear (ver Figura 3, p. 3). A tecelagem é programada por meio de um dispositivo tangível (ver Figura 4, p. 4) conectado a um computador portátil⁴, uma matriz de botões acopláveis, desenvolvida por Ellen Harlizius-Klück (investigadora da história da matemática, filosofia e tecelagem da Grécia Antiga na Universidade de Copenhague⁵) e Alex McLean (artista-programador de destaque). Imagens em movimento foram projetadas como capturas das atividades têxteis e processadas por Griffiths.



Figura 3 – Performance no Foam Kernow. Fonte: Griffiths (2015b).

³ Cf. PROSPERO, 2015

⁴ Disponível em <<https://www.raspberrypi.org/>>

⁵ Disponível em <<http://www.saumweberei.de/>>



Figura 4 – Alex McLean manipulando uma Matriz de botões para tecelagem, conectado a um Raspberry Pi. **Fonte:** Griffiths (2015b).

Uma das ideias originais que tivemos foi combinar tecelagem e codificação em um cenário de performance, ambos como 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.⁶

Esta descrição de Griffiths possui uma documentação audiovisual⁷.

1.1.1 Slub

A parceria entre Griffiths e McLean é essencial para entender o *live coding* inglês. Ambos são membros da banda *Slub*, que começou como uma colaboração entre Adrian Ward e Alex McLean. O grupo foca na atividade de programação para realização de uma Música Eletrônica para Dançar⁸.

⁶ 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.* ”

⁷ Disponível em <<https://www.youtube.com/watch?v=XrnIVUp9QgM>>

⁸ Cf. RIETVELD, 2013

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* em Barcelona, momento em que iniciaram um processo de *gameficação* das performances, (MCLEAN, 2011, p. 138–140), e mais tarde, na inclusão de atividades têxteis. McLean (2011, p. 139) descreve um sistema *Slub*, uma suíte de programas desenvolvidos para o fim específico de improvisar códigos:

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, o[s membros do] *Slub* gostavam de serem confrontados com o desafio de aceitação como programadores que fazem música. Para este fim, começaram a projetar suas telas de audiência com uma sobreposição conceitual, entre seu *softwares* artesanais, e a música que produziam com seu uso.⁹

Em 2004, Ward e McLean focaram seus esforços no desenvolvimento de ambientes de improvisação de códigos, estruturados como editores de linguagens textuais ou visuais, e interfaces gráficas de usuário¹⁰: “O *Slub* controlava sua música usando interfaces criadas por e para eles mesmos. Eles variam desde as [interfaces] aparentemente convencionais para as abstratas, e das gráficas para as inteiramente textuais”.¹¹ (COLLINS, 2003, p. 323):

Por detrás das interfaces *slub* residem os processos ‘composicionais’ ou ‘musicais’ – muitos pedaços de códigos separados, escritos como exploração de ideias 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 composicionais 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.¹²

⁹ 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.*”

¹⁰ *Graphical User Interfaces* ou GUIs.

¹¹ 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.*”

¹² 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*

A história do *Slub* se confunde com o desenvolvimento de um tipo específico de Música Eletrônica para Dançar, ou *Algorave*. Segundo Chesire (2013), o termo surgiu durante uma *gig*, onde o compositor Nick Collins e o artista-programador Alex McLean combinaram dois termos, *algorithm* e *rave* para caracterizar uma performance que sintonizava uma estação de rádio transmitindo uma programação festiva:

Algorave 'começou 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.¹³

Em seu artigo "Algorave: Live Performance of Algorithmic Electronic Dance Music", Collins e McLean (2014, p. 356) sustentam que o *algorave* é anterior à improvisação de códigos. O que relaciona ambos é a prática de projeção da tela do computador (ver seção 2.4, p. 22):

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.¹⁴

Afim de esclarecer este paralelo entre o *Slub*, a improvisação de códigos e o *Algorave*, descrevemos um resumo histórico do *Algorave* feito por Collins e McLean (2014). 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".¹⁵ Em

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."

¹³ 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. "

¹⁴ 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."

¹⁵ 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>>."

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*¹⁶, e do *Supercollider*¹⁷. 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 (ver seção 2.4, p. 22) em uma casa noturna de Hamburgo.

1.2 Dança

Nesta seção contrapomos a visão da seção anterior através uma suíte coreográfica de Kate Sichio, *Hacking the Body*. Discutimos a elaboração/codificação como parte de uma *estratégia transversal*¹⁸, de uma partitura de performances para um pseudo-código de computador, e finalmente em uma coreografia codificada ao vivo.

Para Sichio, a relação entre a atividade de escrever programas, e a Dança como composição de movimentos corporais, é parte de um trabalho contínuo entre notação de coreografias e a improvisação de movimentos. Este trabalho parte daquilo que Sicchio (2014, p. 31), através de Downie (2005, cap. 1, p. 3), cita como *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 computação¹⁹ (DOWNIE, 2005, cap. 1, p. 2–4):

Esta sensibilidade computacional é presente em dois níveis nos trabalhos 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 [, como] a democracia do espaço do palco, e a redescoberta do que está atrás do dançarino como ponto de origem do movimento – podem ser interpretadas 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 generativa.²⁰

¹⁶ Disponível em <<https://csound.github.io/>>.

¹⁷ Disponível em <<http://supercollider.sourceforge.net/audiocode-examples/>>

¹⁸ Cf. FORTH; WIGGINS; MCLEAN; MCLEAN, 2010, 2011

¹⁹ Tradução nossa de “*mechanisms of generalization and abstraction, choreography as representation, dance as computation*”

²⁰ 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*

1.2.1 Hacking Coreography

A improvisação de códigos parte de uma proposição (*v.01*): *hackear* uma Partitura de Eventos do artista Alison Knowles (mais especificamente a peça de performance #8, de 1965), e projetá-la em um espaço de performance, onde 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 :

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

A orientação (*hack*) de Knowles aos dançarinos é que executem a partitura original integralmente uma única vez. No momento em que a última rotina de movimentos (combinar "nada" com "nada"), deve ocorrer uma desconstrução dos rótulos originais através de separações de suas sílabas , de forma que são derivados 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].²¹

A segunda experiência, *Hacking Coreography beta v.02*, é inspirada na mesma partitura de Alison Knowles, agora com o objetivo de definir algoritmos associados a um termo técnico de movimento corporal, híbridos de texto discursivo e código de computador

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. Grifo nosso.”

²¹ 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.””

em linguagem Java. Isto é, o código é executável por um computador para resultar em sons ou imagens, 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)  
  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 Hacking the Body

O *Hacking The Body 2.0*, ou *HTB2.0* (2015)²² é uma performance posterior de Sichio, que seguiu desenvolvimentos posteriores aos citados na seção anterior.

A coreógrafa está sentada em uma penumbra. Já uma dançarina recebe aos poucos uma iluminação contrastante, com uma vestimenta branca e uma iluminação frontal (ver Figura 5, p. 11). A coreógrafa improvisa um código de movimentos corporais que são executados por outra mulher:

Esta peça é uma exploração de eletrônica codificada ao vivo e movimentos improvisados. Uma dançarina veste uma peça de atuadores hápticos. Estes atuadores são programados em tempo-real via OSC²³ para 'buzz' 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 háptica. 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*²⁴

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 na seguinte pergunta: qual é o *dispositivo de entrada e de saída* praticado nas improvisações de códigos? Sicchio responde que o corpo

²² Disponível em <<https://www.youtube.com/watch?v=iOAffWTBVE0>>

²³ 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>>

²⁴ 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.livecodenetwork.org/performances.html>>. ”



Figura 5 – Dançarina (anônima) controlada por Kate Sicchio (2015) através de uma codificação improvisada. **Fonte:** <<https://www.youtube.com/watch?v=uAq4BAbvRS4>>.

já é um dispositivo de entrada e saída de interações sociais e pode ser controlado por outro humano através de comandos de rede. A sensação de quietude existe não por questões musicais, mas por perguntar por onde passam os fluxos de informações digitais.

1.3 Discussão

Neste capítulo, delimitamo-nos a exemplificar ciclos de elaboração e codificação de uma proposição artística. De acordo com o antropólogo Giovanni Mori, esta proposição artística pode ser musical, visual, coreográfica ou têxtil. Afim de ilustrar a pluralidade da técnica de improvisação, escolhemos descrever duas formas, a têxtil e a coreográfica. O Audiovisual foi descrito como uma abordagem satélite, o que certamente encoraja futuras pesquisas na área específica. A Música não ganhou foco neste capítulo, de forma que será destacada nos próximos dois capítulos. No [Capítulo 2](#) (ver p. 13), situamos propostas prototípicas de uma improvisação musical realizada através de computadores, nos moldes do *live coding*. No [Capítulo 3](#) (ver 31), analisaremos uma proposta, seu primeiro ciclo de elaboração e codificação em código de linguagem LISP, e um simples resultado sonoro.

2 Definições Históricas da Improvisação de códigos

Este capítulo desenvolve um contexto cronológico da improvisação de códigos, do ponto de vista musical. Mori (2015b) descreve um caso prematuro de *live coding* na Itália, com o compositor Pietro Grossi (ver seção 2.1, p. 13). As atividades dos grupos californianos *The League of Automatic Composers/The Hub* contextualizam o ambiente cultural estadunidense (ver seção 2.2, p. 17). Deste ambiente, o compositor Ron Kuivila propõe em Amsterdã uma improvisação de códigos prototípica, sem projeção (ver seção 2.3, p. 22). No começo dos anos 2000, sete programadores ingleses respondem à crítica de Schloss (2003), ou o papel cênico do músico durante uma apresentação com computadores (ver seção 2.4, p. 22).

2.1 Pietro Grossi

O compositor veneziano Pietro Grossi foi um dos pioneiros da *Computer Music Italiana*. Sacrificou questões timbrísticas e focou na *responsividade* (ver subseção 2.1.1, p. 14) e na *telemática* (ver subseção 2.1.2, p. 16).

Segundo Mori (2015b, p. 126):

Grossi começou a se interessar por música computacional durante a primeira metade do anos 60, quando ele organizou um programa de rádio centrado em torno de uma "música inovadora" (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

Uma cópia do disco *GE-115 - Computer Concerto*, gravado no *Studio di Fonologia musicale di Firenze* e lançado pela Olivetti em 1967, possui a seguinte descrição das composições de Grossi: “Do lado A existem algumas transcrições de música clássica, e do lado B existem três canções originais. (...) Este 7”[polegadas] foi distribuído como

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, 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*.”

presente de natal e de ano novo pela companhia Olivetti”^{2,3}. 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 o quinto dos 24 Caprichos de Nicolò Paganini. As peças originais de Grossi foram três: *i) Mixed Paganini* (derivado do 5º capricho) ; *ii) Permutations Of Five Sounds* e; *iii) Continuous*:

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). (MORI, 2015b, p. 126).⁵

2.1.1 Reflexividade

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 Comitê Nacional para a Pesquisa”⁶ (MORI, 2015b, p. 126). 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.”⁷ (*idem, ibidem*). Propomos substituir *prigo* por *reflexivo*, 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*”⁸. (MALENFANT; JACQUES; DEMERS, 1996, p. 1).⁸

Este sentido "preguiçoso" ou reflexivo levou Grossi a advogar que novos timbres gerados por computador deveriam 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 composicionais

² 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.*”

³ Disponível em <https://www.youtube.com/watch?v=ZQSP_wF7wSY>

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

⁷ 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.*

⁸ 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 called reification.*”

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, ibidem*).⁹

Mori (2015b, p. 127) destaca o compositor 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).¹⁰

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 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.*”

2.1.2 Telemática

É importante situar que a escolha deliberada para o 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, ibidem*).¹¹ 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 com sete sobretons. Cada sobreton era controlado individualmente pelo programa.¹²

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”¹³. 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”.¹⁴

Grossi foi além deste problema reflexivo no final da década de 1970. O TAU2-TAUMUS sofreu uma considerável modificação para controlar um sistema digital-analógico remotamente. O novo programa foi batizado em 1986 de TELETAU e, segundo Mori (2015b, p. 128–129), possibilitava o acesso a um computador da CNR, em Pisa, com uma conexão da rede BITNET. No entanto o TELETAU não vingou por diversos motivos: falhas e bugs que aumentavam de maneira dramática a manutenção e custos; o alto custo

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.”

¹¹ 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)*”

¹² 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.*”

¹³ Tradução nossa de “*they were a sort of patches that acted on some musical parameter.*”

¹⁴ 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.*”

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)¹⁵

2.2 Baía de São Francisco

A prática musical com o computador, realizada na Costa Oeste dos EUA durante os anos 1970 e 1980, segundo [Brown e Bischof](#), decorre de um contexto construído em torno do *Mills College* em Oakland ([BROWN; BISCHOF, 2002](#), 3º parágrafo):

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 perseguiram 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 incluíam improvisações barulhentas e despreocupadas, que sobraram das revoluções contra-culturais dos anos 60 ([BROWN; BISCHOF, 2002](#), 1º parágrafo)¹⁶.

¹⁵ 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.*”

¹⁶ 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.*

2.2.1 The League of Automatic Composers

Na segunda metade da década de setenta, Jim Horton começou a adquirir microcontroladores KIM-1¹⁷ com interesses musicais. Segundo [Brown e Bischof \(2002\)](#), não demorou para que outros compositores interessados comprassem. Discussões informais posteriores incluíram, além de Horton, David Behrman, 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 6](#), p. 18). A primeira peça foi realizada com algoritmos inspirados nas teorias matemáticas de Leonard Euler (séc. XVIII). A segunda peça 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)¹⁸. 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)¹⁹ de quatro faixas no *Blind Lemmon*, um ponto de encontro musical fundado em 1958²⁰. 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*”²¹, que, além de Bischof e Behrman, participavam Tim Perkis, Scot Gresham-Lancaster, Mark Trayle e Phil Stone.

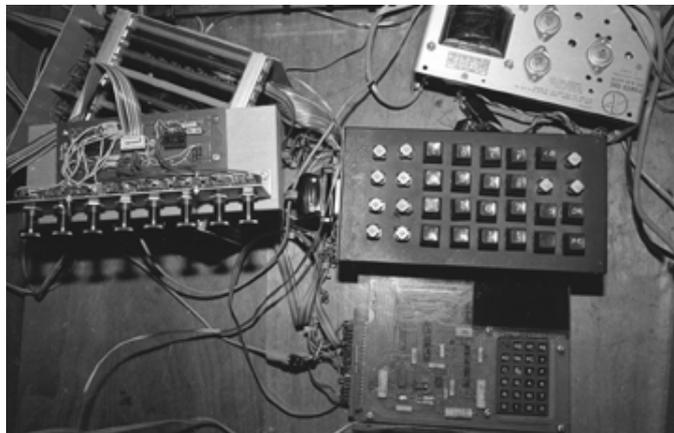


Figura 6 – Sistema de música computacional de John Bischof *circa* 1980. Foto: Eva Shoshanny²². Fonte: [Brown e Bischof \(2002\)](#).

¹⁷ *Keyboard Input Monitor*. Disponível em <<http://www.6502.org/trainers/buildkim/kim.htm>>.

¹⁸ Tradução nossa de “*the occasional tones of my [Bischof] machine caused Jim’s machine to transpose its melodic activity according to my “key”note.*”

¹⁹ Gravação muito longa para um *demo* e insuficiente para um disco de vinil da época.

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

²¹ Segundo [Brown e Bischof \(2002, 6º parágrafo\)](#), o nome é uma referência ao grupo “The League of Composers” formado por Aaron Copland nos anos 20.

²² Tradução de *John Bischoff’s KIM-1 computer music system circa 1980 photo: Eva Shoshany*

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 do tipo em que outros compositores aparecem, tocando ou compartilhando circuitos eletrônicos que tinham projetado e construído. Um interesse na 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 'venerável e velho edifício'. A série durou cerca de 5 meses que eu me lembro. (BROWN; BISCHOF, 2002, online)²³

Em 1980, Gold e Behrman abandonam o grupo, e Tim Perkis se junta. Este foi período em que o grupo solidifica suas atividades na região da Baía de São Francisco. O trio (Horton, Bischof e Perkis) formaliza a comunicação entre os microcontroladores de cada membro – o que para a época era arriscado ao ponto de queimar componentes. Realizadas as conexões, tocavam ao menos três horas, tempo em que ouviam e ajustavam os sistemas²⁴(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 Cloldt e Brian Reinbolt. O grupo “baseava seu estilo de performance em torno de uma caixa de comutação projetada por Brian Reinbolt²⁵(BROWN; BISCHOF, 2002, 8º parágrafo). Em 1983 o grupo reduziu suas atividades, época em que Horton contraiu artrite degenerativa.

Brown e Bischof (2002, 11º parágrafo) suas redes de composições, ou “ocasiões públicas para escuta compartilhada²⁷. O som era produzido por um sistema limitado, de “baixa velocidade (1 MHz) e poucos dados (8 bits)”²⁸ com ênfase em uma artesanaria instrumental híbrida de performance. Em outras palavras, “A ênfase estava na exploração

²³ 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.*

²⁴ Disponível em <<https://www.youtube.com/watch?v=HW0qax8M68A>>

²⁵ Tradução nossa de “based their performance style around an automatic switching box designed by member Brian Reinbolt”.

²⁶ Tradução de *Tim Perkis' homebuilt computer-driven sound synthesis circuitry used in early 1980s. photo: Eva Shoshany.*

²⁷ Tradução nossa de “public occasions for shared listening”.

²⁸ Tradução nossa de “slow speed (1 MHz) and data width (8 bits)”

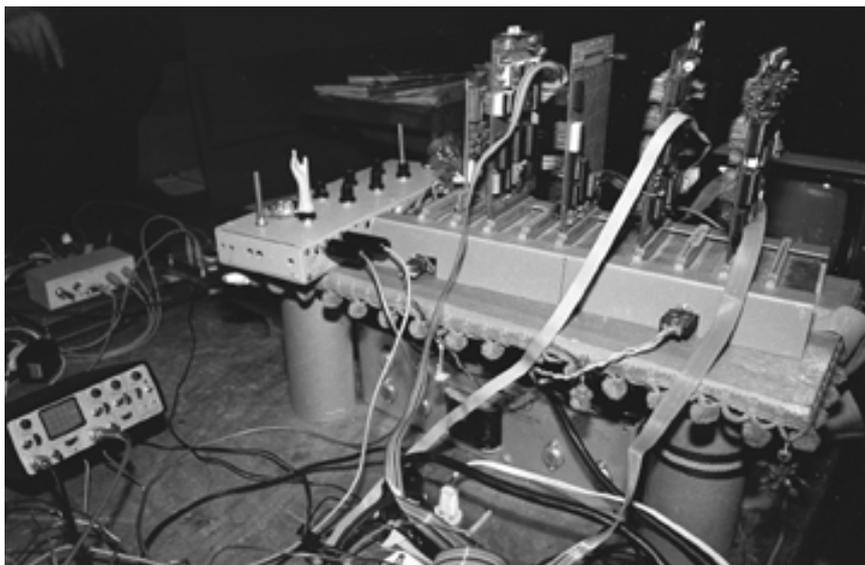


Figura 7 – Circuito do computador caseiro dedicado à síntese sonora de Tim Perkis, usado no começo dos anos 1980. Foto: Eva Shoshany²⁶. Fonte: Brown e Bischof (2002)

da tecnologia em mãos – que poderia ser adquirida pessoalmente ou construída a partir do zero, em vez do desejo incessante de melhores ferramentas”.²⁹(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, bem como 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 interagem, cada uma enviando e recebendo dados pertinentes para o funcionamento musical. (BROWN; BISCHOF, 2002, 12º parágrafo).³⁰

2.2.2 The Hub

O primeiro *The Hub* foi surgido com o duo Bischoff e Perkis que desenvolveram trabalhos com outros grupos em 1986, como o duo formado por Chris Brown e Mark Trayle,

²⁹ 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.*”

³⁰ 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.*”

o duo Scott Greham-Lancaster/Richard Zvonar, e o trio Phil Burk/Larry Polansky/Phil Stone. Bischoff pontua que o nome da banda era uma maneira simbólica de caracterizar um 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”.³¹ O computador centralizado original, *Hub*, era um dos microcontroladores KIM-1 utilizados na época do *The League*:

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 juntos. 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).³²

Em 1987, Nick Collins e Phil Niblock realizaram uma curadoria de performances telemáticas entre a *Experimental Media* e *The Clocktower* em Nova York. Participam os membros do *The Hub*, divididos em dois trios, formados por John Bischoff/Tim Perkis/Mark Trayle e Chris Brown/Scot Gresham-Lancaster/Phil Stone. Cada trio possuiu um *Hub* intercomunicável. As performances “*Simple Degradation*”, “*Borrowing and Stealing*” e “*Vague Notions*” ocorrem através da intercomunicação entre os *Hubs*, de forma que o “sexteto [está] acusticamente divorciado mas informacionalmente ligado”.³³ (BROWN; BISCHOF, 2002, seção 2.2).

³¹ 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.”

³² 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.”

³³ Tradução nossa de “acoustically divorced, but informationally joined sextet.”

2.3 Ron Kuivila

McLean e Wiggins (2009) comentam a performance *Water Surfaces*, realizada na edição de 1985 da STEIM³⁴, em Amsterdã, como significativa para a concepção de uma improvisação de códigos (excluindo a tecnologia de projeção visual). A performance chamou a atenção de, e foi incluída na primeira faixa do disco “*TOPLAP001 - A prehistory of live coding*” (2007),³⁵; 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...”³⁶

Ronald Kuivila programou um computador Apple II no palco para criar 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 a problemas sistêmicos. Ele finalmente entrou em dificuldades técnicas e finalizou a performance (ROADS, 1986, p. 47)³⁷.

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”.³⁸

Nenhuma fonte sonora foi encontrada disponível online.

2.4 Live coding

O documento-manifesto “*Live Algorithm Programming and Temporary Organization for its Promotion*”, de Ward et al. (2004), McLean e Wiggins (2009) formaliza regras para uma improvisação de códigos (ver subseção 2.4.1, p. 23) e posteriormente possibilita a construção da organização TOPLAP (ver subseção 2.4.2, p. 24). Deste manifesto, selecionamos um ponto: o lema “Show us your screens” como uma síntese das regras práticas do *live coding* (ver subseção 2.4.3, p. 25).

³⁴ *STudio for Electro-Instrumental Music*, disponível em <<http://steim.org/about/>>.

³⁵ Disponível em <http://toplap.org/wiki/TOPLAP_CDs>.

³⁶ 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...*”

³⁷ Tradução de *Ronald Kuivila programmed an Apple II computer on stage 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*

³⁸ 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.*”

2.4.1 LAPTOP

“*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 *Livcoding* 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 retroprojektor 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)³⁹

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

³⁹ 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.*

feita das supostas propriedades da chamada “música generativa” que separa o compositor do resultado do seu trabalho.”⁴⁰

A Música como um Processo Gradual⁴¹ e a Música Generativa são referenciais possíveis na improvisação de códigos, mas não estão ligadas necessariamente como processo de escuta. A primeira é descrita por Mailman (2013, p. 128) como processos determinísticos que agem sobre focos de quadros temporais. A segunda “(...) é sensitiva às circunstâncias, isso quer dizer que irá reagir diferentemente dependendo das suas condições iniciais, onde ocorre e assim por diante”⁴² (ENO, 1996). Ambas diferem do processo de codificação descrito por McLean (2011, p. 130):

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.⁴³

2.4.2 TOPLAP

Ward et al. (2004, p. 246) e Ramsay (2010) significam acrônimo LAPTOP e descrevem o surgimento da organização TOPLAP (ver Figura 8):

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 esfumaçado em Hamburgo a uma da manhã em 15 de Fevereiro de 2004.⁴⁴

⁴⁰ 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.*

⁴¹ Cf. REICH, 1968

⁴² 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.*”

⁴³ 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*

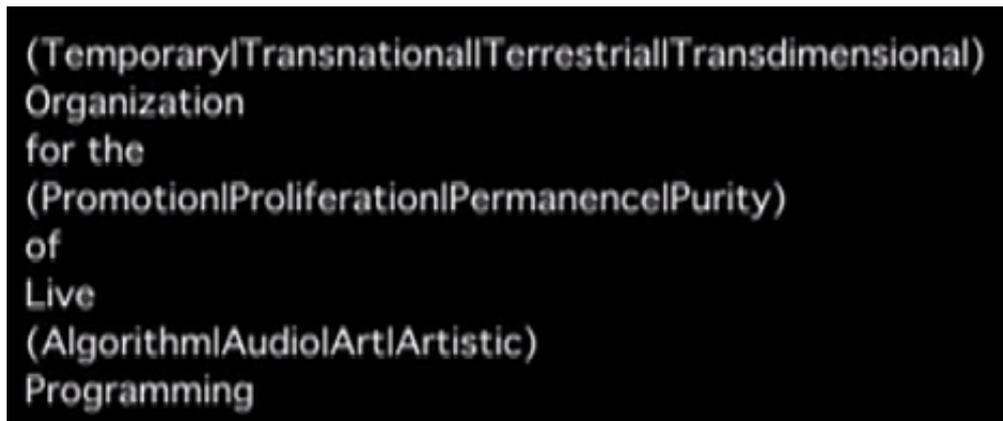


Figura 8 – Definição do significado de TOPLAP. **Fonte:** Ramsay (2010).

O símbolo “|” é uma representação gráfica do operador lógico *OR* (OU), bastante utilizado em estruturas condicionais. Isto é, *Temporary | Trasnational | Terrestrial | Transdimensional* significa que as letras ímpares “T”, e “P” e “A”, podem significar um ou outro termo indicado pelo algoritmo. Este comportamento é praticado por Nick Collins (1975-) para gerar pseudônimos como Click Nilson, ou Sick Lincoln.

2.4.3 *Show us your screens*

Além das performances inaugurais nos festivais Europeus, o manifesto Lubeck04, “iniciado em um ônibus *Ryanair*, em Hamburgo para o aeroporto Lübeck”(WARD et al., 2004, p. 247)⁴⁵, 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.

program a few time not to produce new works, but to get different perspectives on the same work. ”

⁴⁴ 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*”

⁴⁵ Tradução nossa de “*begun on a Ryanair transit bus from Hamburg to Lubeck airport*”

- A externalização hábil de algoritmo como exibição expressiva/impres-siva 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. ⁴⁶

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**

⁴⁶ 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/impresive 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, 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.

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.⁴⁷

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”⁴⁸ 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 de performances com dispositivos estendidos, 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 desculpa/oportunidade para um programador(a) se sentar no palco. Sua presença melhora ou impede o desempenho da representação?⁴⁹

⁴⁷ 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* ”

⁴⁸ Tradução nossa de “ *Its 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.*”

⁴⁹ 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?*”

No item 3, é apontado uma justificativa para projeções visuais 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, ao projetarem códigos-fonte. Para Schloss, isso é mimar o aparato (e o público) e tornar a apresentação pedante. No item 5, a representação cênica do esforço é importante, e neste sentido, é essencial para o ouvinte receber a música como performance. O item 7 restringe a atividade do programador como artista, e é, ao mesmo tempo, o gatilho para Ward et al. começarem a divulgar o *live coding*.

2.5 Discussão

Oferecemos um cenário proto-histórico do ponto de vista, na Itália com o compositor Pietro Grossi (ver seção 2.1, p. 13), nos EUA com Jim Horton, John Bischoff, Tim Perkis (ver seção 2.2, p. 17), e na Holanda com Ron Kuivila (ver seção 2.3, p. 22) que deram suporte ao pensamento promovido por Ward et al. (2004).

O tipo de material sonoro utilizado nas peças de Grossi, se comparada com os trabalhos de Max Mathews⁵⁰, estão debruçadas na resolução do problema de performance, contraposto à capacidade de processamento dos *mainframes* da época. Para Grossi, com o problema da capacidade de processamento, os compositores deveriam esperar por melhores implementações técnicas dos engenheiros, e naquele contexto, o computador foi uma ferramenta de bricolagem para reprodução de uma música do séc. XVII-XX (*circa*). Isto é, operações composicionais tradicionais como inversão, retrogradação, retrogradação da inversão, aceleração, diminuição, são reproduzidas como comandos de computador.

Nossas descrições sobre o contexto estadunidense centraram-se no trio formado por Jim Horton, John Bischoff, Tim Perkis, e posteriormente como um sexteto, formador por John Bischoff/Tim Perkis/Mark Trayle/Chris Brown/Scot Gresham-Lancaster/Phil Stone. Membros do grupo referem-se ao *The Hub* como um *sexteto acusticamente divorciado mas informacionalmente ligado*.. Os materiais composicionais são informações interoperadas por um sistema comum, e então distribuídas entre os integrantes, que agem como *jogadores*.

Seguindo os desenvolvimentos tecnológicos da época, Ron Kuivila representa um uso peculiar do computador para a atividade de composição musical, ainda que dado ao aparente fracasso. No entanto, não descartamos a possibilidade de uma queima programada⁵¹.

Por último, descrevemos a formalização de uma improvisação de códigos. Esta formalização se deu a partir de um embate acadêmico levado a cabo por sete artistas-programadores ingleses (WARD et al., 2004). Um documento-manifesto foi produzido e

⁵⁰ Cf. MATHEWS; MATHEWS et al.; ROADS; MATHEWS; PARK; MATHEWS; NUNZIO, 1963, 1969, 1980, 2009, 2010

⁵¹ Disponível em <http://www.tycomsystems.com/beos/beobook/The%20Kernel%20Kit/index.html>

estimulou a divulgação de uma forma de estética *hacker* em Músicas Eletrônicas para Dançar.

3 Estudo de caso

A pesquisa desenvolvida nos capítulos anteriores suscitaram a seguinte pergunta: como investigar um caso musical de improvisação de códigos, um vídeo de *A Study in Keith* de Andrew Sorensen (2012)?

Os trabalhos de Forth, Wiggins e McLean (2010), McLean (2011) possibilitaram investigar o vídeo dentro de um abordagem cognitivista, de forma que contextualizamos um método de análise sugerida por McLean (2011, p. 117):

Aqui nós tomamos a perspectiva que uma propriedade conceitual é representada por um melhor *exemplo simples* possível, ou *protótipo*. (...) 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. Um musicista, que explora os limites do jazz, encontrou 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.¹

3.1 Metodologia de Análise

Para analisar *A Study in Keith* utilizamos um recorte do *Quadro Conceitual de Sistemas Criativos*² (ver Tabela 1, p. 32), e do modelo-baseem que este foi desenvolvido, o modelo de improvisação de Jeff Pressing (1987) (ver subseção 3.1.1, p. 33). É importante esclarecer que, para não confundirmos questões ontológicas, substituímos *conceito* por *proposição da improvisação* ou simplesmente *proposição* (ver seção 3.2, p. 34). Renomeamos o termo *Referente opcional* de Pressing por *Referentes de A Study in Keith Zero, Um, Dois e Três*. A Linguagem é *A Linguagem de Programação utilizada em A Study in Keith*.

¹ Tradução nossa de “Here we take the view that a conceptual property is represented by a single best possible example, or prototype. In accordance with the theories reviewed in chapter 2, these prototypes arise through perceptual states, within the geometry of quality dimensions. 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 ”

² Creative System Frameworks, ou CSF, Cf. McLean; FORTH; WIGGINS; MCLEAN; MCLEAN, 2006, 2010, 2011.

Tabela 1 – Definições formais do Universo de possibilidades de Wiggins (2006), ou Universo de Conceitos por McLean (2006), Forth, Wiggins e McLean (2010). Neste trabalho, como quadro de proposições.

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) ³
\mathcal{L}	Linguagem	Linguagem utilizada para expressar regras.
\mathcal{A}	Alfabeto	Alfabeto da linguagem que contém 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.
$[[\cdot]]$	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) ⁴
$[[\mathcal{R}]]$	Regras de validação	“Uma função que interpreta \mathcal{R} , resultando em uma função indicando aderência ao conceito em \mathcal{R} ” ⁵
$\mathcal{C} = [[\mathcal{R}]](\mathcal{U})$	Espaço Conceitual	“Todos espaços conceituais são um subconjunto não-estrito de \mathcal{U} ” ⁶ . 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} ” ⁷
\mathcal{T}	Regras de detecção	“Regras definidas dentro de \mathcal{L} para definir estratégias transversais para localizar conceitos dentro de \mathcal{U} ” (McLean, 2006) ⁸
\mathcal{E}	Regras de qualidade	“(…) conjunto de regras que permitenos 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) ⁹ “Regras definidas dentro de \mathcal{L} para avaliar a qualidade ou a desejabilidade do conceito c ” (McLean, 2006) ¹⁰
$\langle\langle\langle \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle\rangle\rangle$	Função de interpretação	Uma regra necessária para definir o espaço conceitual, “independentemente da ordem, mas também, ficcionalmente, 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) ¹¹ . “Uma função que interpreta a estratégia transversal \mathcal{T} , informada por \mathcal{R} e \mathcal{E} . Opera sobre um subconjunto ordenado de $\mathit{mathcal{U}}$ (do qual tem acesso randômico) e resulta em outro subconjunto ordenado de $\mathit{mathcal{U}}$ ” ¹²

3.1.1 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” (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 2. Por seu caráter lógico, parece ser uma possibilidade interessante, e assumiremos como tal.

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

Representação	Significado
E'	Um bloco de eventos sonoros ¹³
K'	Uma seqüência de blocos de eventos E, onde um bloco de eventos não se sobrepõe com o seguinte ¹⁴
I'	Uma improvisação, particionada por interrupções em um número de K seqüências ¹⁵
R'	Um referente opcional, tal como uma partitura ou uma norma estilística ¹⁶
G'	Um conjunto de objetivos ¹⁷
M'	Uma memória de longo prazo ¹⁸
O'	Um conjunto de objetos ¹⁹
F'	Um conjunto de características dos objetos ²⁰
P'	Um conjunto de processos ²¹

³ Tradução de *A universe of all possible concepts.*

⁴ 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.*

⁵ Tradução de *A function interpreting \mathcal{R} , resulting in a function indicating adherence of a concept to \mathcal{R}*

⁶ Tradução de *All conceptual spaces are non-strict subset.*

⁷ Tradução de *A function interpreting \mathcal{R} , resulting in a function indicating adherence of a concept to \mathcal{R} .*

⁸ Tradução de *Rules defined within \mathcal{L} to define a traversal strategy to locate concepts within \mathcal{U}*

⁹ 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.*

¹⁰ Tradução de *Rules defined within \mathcal{L} which evaluate the quality or desirability of a concept c .*

¹¹ 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} .*

¹² 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} .*

3.2 A Study in Keith: Proposição

Sorensen faz duas as descrições de uma mesma proposição, ou o *Espaço conceitual de A Study in Keith*. Os Concertos *Sun Bear* de Keith Jarret (ver seção 3.3, p. 34) são citados como inspiradores da improvisação de códigos. Desta forma, existe um *Referencial Zero de A Study in Keith*, ou \mathcal{R}_{ask}^0 .

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)²²

Sorensen (2015) indica outros referenciais, que chamamos de *Referencial Um, Dois e Três de A Study in Keith*, $\mathcal{R}_{ask}^1, \mathcal{R}_{ask}^2, \mathcal{R}_{ask}^3$, ou o piano virtual *Akoustik Piano NI*, o ambiente de programação *Impromptu* e a linguagem de programação *Scheme*. Este último é a *Linguagem de Programação de A Study in Keith*, ou \mathcal{L}_{ask} :

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).²³

3.3 Referentes Opcionais

Aqui foi possível elaborar uma solução possível ao problema enunciado como “*Study in Keith* não é *Sun Bears*”. Em seguida tratamos do referencial um, \mathcal{R}_{ask}^1 , ou o timbre de piano utilizado (ver subseção 3.3.2, p. 37), e de um ambiente de programação musical chamado *Impromptu* (ver subseção 3.3.3, p. 38) como referencial dois \mathcal{R}_{ask}^2 .

¹³ A cluster of sound events.

¹⁴ A sequence of E event clusters, where event cluster onsets do not overlap with those of a following one

¹⁵ An improvisation, partitioned by interrupts into a number of K sequences

¹⁶ An optional referent, such as a score or stylistic norm

¹⁷ A set of current goals.

¹⁸ Long term memory.

¹⁹ An array of objects.

²⁰ An array of objects Features.

²¹ An array of Process

²² 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.*”

²³ 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.*”

3.3.1 Concertos Sun Bear

Os concertos *Sun Bear* são originalmente dez LPs de improvisações de Keith Jarret no Japão, produzidos pela *ECM Records*²⁴ entre 1976 e 1978. 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 novembro²⁵; Osaka, 8 de novembro²⁶; Nagoya, 12 de novembro²⁷. Tokyo, 14 de novembro²⁸; Sapporo, 18 de Novembro²⁹.

Existem algumas notas discográficas compiladas por uma comunidade de fãs e críticos musicais estadunidenses. 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 Suíte.”³⁰([GAR BOLINO, 2014](#), p. 129).

Revisto por Richard S. Ginell³¹: [-] 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 ([GAR BOLINO, 2014](#), p. 130) ³².

O *gospel* e o *folk* são categorizados como gêneros musicais nesta suíte que não possui pausas entre as partes (o improvisado é contínuo, mas seccionado por transições). Seu motivo gerador é, segundo Uwe [Karcher \(2009\)](#), “Na verdade, a abertura não é realmente improvisada - ela é baseada em uma canção chamada *Song Of The Heart*”^{33,34}.

²⁴ <http://www.ecmrecords.com/>

²⁵ Disponível em <<https://www.youtube.com/watch?v=T2TfIQNxhjc>>.

²⁶ Disponível em <<https://www.youtube.com/watch?v=FC4iZ1wMoU8>>

²⁷ <<https://www.youtube.com/watch?v=3a7ezm3D1jA>>.

²⁸ Disponível em <<https://www.youtube.com/watch?v=ZH8VIjjhPQ4>>

²⁹ Disponível em <https://www.youtube.com/watch?v=BqYBT_HoG4M>

³⁰ 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*”

³¹ Disponível em <<http://www.mcana.org/formembersatlarge.html>>.

³² 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.*

³³ Tradução nossa de “*Actually, the opening was not really improvised - it is based on a tune named Song Of The Heart.*”

³⁴ Disponível em <<https://www.youtube.com/watch?v=JgyRoQPDwM8>>

The image displays a musical score for piano, labeled 'Piano' on the left. It consists of three systems of music, each with a treble and bass clef staff. The first system (measures 1-3) begins with a piano (*pp*) dynamic. The second system (measures 4-7) includes a repeat sign and dynamic markings of *pp* and *f*. The third system (measures 8-10) continues the melodic and bass lines with dynamic markings of *f* and *pp*. The score includes various musical notations such as notes, rests, slurs, and dynamic markings.

Figura 9 – Transcrição do motivo gerador do disco Kyoto, parte 1. textbfFonte: Uwe Karcher (2009).

A maneira como Jarret improvisou este concerto possibilita questionar como o código é improvisado por Sorensen e Swift (2009): o resultado é de fato uma improvisação de códigos, ou existe um agenciamento onde o improvisador prepara um código?

Exemplo 3.1 (Redução da primeira sonoridade dos concertos *Sun Bear*)

Song of the heart apresenta três blocos de eventos iniciais: uma figura que alterna, a partir de um baixo, os intervalos nona menor, terça menor, segunda maior, e oitava, formando um ostinato. Nos compassos 3 a 5 aparecem uma nota que forma uma relação de trítono com o baixo. O acorde formado, um Sol bemol Maior (transcrito assim para facilitar a leitura), é expandido nos compassos 6 a 10, gerando uma figura cromática cuja transcrição apresenta a seguinte cadência: Sol Bemol Maior (com décima primeira aumentada adicionada, em terceira inversão), Fá Maior (que alterna o $\hat{5}$ com o $\hat{1}3^{35}$) e Dó Maior com sétima menor (posição fundamental). Limitamo-nos a considerar a progressão do ponto de vista da exploração de um contra-polo seguido de uma subdominante da dominante e dominante.

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 trítono, a primeira subdominante da sequência, ou $C : subV/IV$ (substituição por trítono da subdominante em Dó Maior). Isto é, a substituição-padrão, $C : bV/V^7$ (acorde de quinto grau bemol da dominante com sétima), ou $C : bII^7$ (acorde de segundo grau bemol com sétima), passa a ser operacionalizada como um contra-polo à tônica ³⁶. 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 última transcrição de Jarret (considerando sua futura correção), suprime e transforma as sétimas no quinto grau bemol, o que caracteriza uma sonoridade de tensão progressiva (e ambígua) com um baixo pedal. Uma tríade do quinto grau bemol, uma téttrade do quarto grau com sexta e quinta (com sua sétima no baixo), e primeiro grau com sétima: $C : bV \Rightarrow IV_4^6 \Rightarrow I^7$. O baixo pedal pode sugerir um 11 grau aumentado no primeiro acorde da sequência ($C : {}_{11\#add}bV$). Uma notação sintética desta sequência sugere o padrão $C : {}_{11\#add}bV \Rightarrow {}_5IV^{4-5} \Rightarrow I^{7-8}$.

The image shows a musical score for a blues progression in C major. It consists of two staves: a treble clef staff and a bass clef staff. The bass line starts on a low C (pedal point) and moves through notes corresponding to the chords: b5, 4, 6, 5. The treble line has notes corresponding to the chords: b2, 1, b7, 1. Above the notes are red annotations: (b2) b5, 1, 4, 6, 5 in the treble; 1, b2, 1, 1, b2, 1, b7, 1 in the bass. Below the staff are chord symbols: C, bV/(add11+), IV64, I7.

3.3.2 NI-Akoustik Piano

A Study in Keith pode ser observado como uma simulação de *Disklavier Sessions Sorensen* (2013). Este último caso não é citado por Sorensen e Swift (2009) e Sorensen (2015) mas a improvisação de códigos é semelhante àquela de *A Study in Keith*.

A Study in Keith utiliza o piano virtual *Akoustic Piano NI* (*Native Instruments*)³⁷. Como um *plugin* VST, os sons do instrumento virtual são gravações nota-a-nota de um

³⁷ Disponível em <<http://www.native-instruments.com/en/company/>>.

piano acústico; a tomada de som é realizada em diversos pontos do tampo harmônico, de forma que detalha as ressonâncias do instrumento. São então amostrados digitalmente³⁸.

Disklavier Sessions utiliza um piano *Disklavier* da Yamaha, um modelo que internalizou um computador registrador de eventos, que podem ser codificados e então são convertidos como movimentos do martelo do Piano:

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*.³⁹

3.3.3 Ambiente e Linguagem: Impromptu

Impromptu é uma linguagem e um ambiente de programação OSX⁴⁰ para compositores, artistas sonoros, VJ's e artistas gráficos com um interesse em programação ao vivo ou [programação] interativa. Impromptu é um ambiente de linguagem Scheme, um membro da família das linguagens Lisp. Impromptu é usado por artistas-programadores em performances de *livecoding* em torno do mundo.⁴¹

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 *o instrumento*:

Considere a analogia da partitura musical tradicional. A partitura prevê 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**

³⁸ Disponível em <<http://www.native-instruments.com/en/products/komplete/keys/definitive-piano-collection/>>

³⁹ 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*”

⁴⁰ Sistema Operacional Mac OSX.

⁴¹ 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/>>”

retroalimentado que envolve premeditação, movendo para ação casual e finalmente para reação, refinamento e reflexão.⁴²

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 outras arquiteturas de processamento (32bit e 64 bit), o projeto foi liberado como código-aberto, com o nome *Extempore*.

3.3.4 Extempore

O *Extempore* possui um sistema de *programação ciberfísica* reflexiva (ver seção 2.1, p. 13):

Extempore é projetado para suportar um estilo de programação apelidado de ["]programação ciberfísica". Programação ciberfísica suporta a noção de um programador humano operando como um agente ativo em uma rede distribuída em tempo-real de sistemas ambientalmente conscientes.⁴³

Entre suas características incluímos⁴⁵:

- Codificação de alto-nível em linguagem Scheme;
- Processamento de Sinais Digitais (DSP)⁴⁶ em tempo-real;
- Sequenciamento de áudio, baseado em notas, como o disparo de sons parametrizados em altura, intensidade e duração⁴⁷;

As primeira e terceira características serão explorada neste capítulo como base do processo de codificação na improvisação *Study in Keith*.

3.3.5 Scheme

Esta subseção demonstra a *Linguagem de A Study in Keith* (ver Tabela 1, p. 32), ou o dialeto *Scheme* da linguagem de programação *Lisp*. Abaixo descrevemos uma característica

⁴² 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.*"

⁴³ 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.* ⁴⁴. "

⁴⁵ Disponível em <<http://benswift.me/2012/08/07/extempore-philosophy/>>

⁴⁶ Sobre DSP, Cf. SMITH, 2012-06.

⁴⁷ Disponível em <<http://benswift.me/2012/10/15/playing-an-instrument-part-i/>>

pertinente à análise de *Study in Keith*, a saber, sua expressão textual através de uma gramática generativa:

Exemplo 3.2 (Expressão simbólica vazia)

()

Segundo McCarthy (1960, seção 3), listas são sentenças abstratas de átomos, que são os elementos constituintes de uma lista:

Exemplo 3.3 (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 outras listas:

Exemplo 3.4 (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 )))
```

Uma característica da *Linguagem de A Study in Keith* (\mathcal{L}_{ask}) é sua notação para expressar uma proposição como “some as unidades de uma lista”:

Exemplo 3.5 (Notação prefixada)

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

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

;;ABACA -> BAACA ->
(+ 1 1 2 1) ;; = 5
```

Existe um vocabulário pré-definido para criação de sentenças, de forma que o significado do código possa ser legível para os propósitos desejados

Exemplo 3.6 (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 (calculo lambda)
    (/ a b))      ;; o que faz a funcao
)

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

Para os propósitos deste trabalho, será útil apresentar um código musical fictício, como um protótipo de *jazz* tonal. O exemplo abaixo alterna citações e códigos para contextualizarmos um pseudo-código descrito por [Sorensen e Gardner \(2010, p. 823-824\)](#):

Exemplo 3.7 (Elaboração/Codificação Musical em Scheme)

Este exemplo é semelhante ao primeiro algoritmo de *A Study in Keith* (ver [seção 3.4, p. 45](#)):

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 ...
48

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

... a expressão é avaliada, e lampeja no retroprojetor, 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: 49

```
( 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 e uma nova barra de tempo, e uma simples progressão recursiva de acordes começa 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... 50

```
( 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 código é a *estratégia transversal* de *A Study in Keith*, ou \mathcal{T}_{ask} :

Exemplo 3.8 (Nome da estratégia transversal)

`chords` é o nome da estratégia.

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

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

Exemplo 3.9 (Estímulo inicial para a estratégia)

```
; Execuciao 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, tétrades, 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 3.10 (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 3.11 (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 3.12 (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 3.13 (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 3.14 (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 3.15 (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 predeterminadas. Neste ponto, podemos indicar de maneira mais explícita uma regra de qualidade (ver [Tabela 1](#), p. 32):

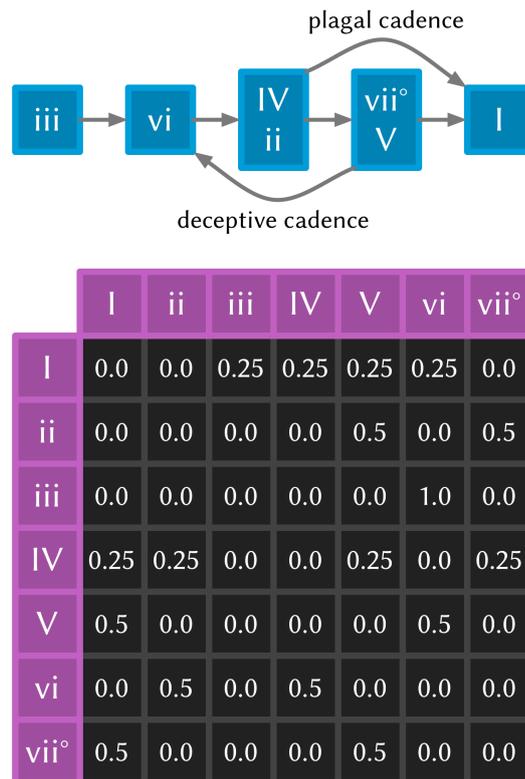


Figura 10 – Distribuição, aproximada, de probabilidades de acontecimento com um conjunto de possíveis cadências tonais organizados como uma cadeia de Markov. **Fonte:** Swift (2012).

Exemplo 3.16 (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.

No caso do bloco de código de 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 $ii^7 \Rightarrow I^7$.

3.4 Blocos de Eventos

Na seção anterior definimos qual é a proposição de *A Study in Keith* e seus referenciais.

Seguiremos com a fase de codificação da estratégia transversal (ver [Tabela 1](#), p. 32) de Sorensen, como regra de detecção \mathcal{T}_{ask} , que possui uma regra de qualidade \mathcal{E}_{ask} (ver [subseção 3.4.1](#), p. 46). Este espaço conceitual gera uma sequência de blocos de eventos \mathcal{K}_{ask}^{t0} (ver [Tabela 2](#), p. 33), como um contraponto de primeira espécie (ver [subseção 3.4.1](#), p. 46), sem relação alguma com o plano harmônico de *Sun Bears* (ver [seção 3.2](#), p. 34).

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 vídeos de Sorensen: *An evening of livecoding at 53 Rusden Street*⁵¹, *Just for Fun*⁵², *A Study in Part*⁵³, *Stained*⁵⁴, *Transmissions in Sound*⁵⁵, *Antiphony*⁵⁶, *Strange Places*⁵⁷, *Orchestral*⁵⁸, *UMDT*⁵⁹, *Day of Triffords*⁶⁰, *Face to Face*⁶¹, *BM&E*⁶², *A Christmas Carol*⁶³, *Dancing Phalanges*⁶⁴, *Livecoding Audio DSP*⁶⁵, *Jazz Ensemble Study*⁶⁶, *Variations on a Christmas Theme*⁶⁷.

3.4.1 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''):

Exemplo 3.17 (Definição de instrumento)

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 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")

```

51 Disponível em <<https://vimeo.com/2433303>>
52 Disponível em <<https://vimeo.com/2433971>>
53 Disponível em <<https://vimeo.com/2434054>>
54 Disponível em <<https://vimeo.com/2502546>>
55 Disponível em <[TransmissionsinSound](#)>
56 Disponível em <<https://vimeo.com/2503188>>
57 Disponível em <<https://vimeo.com/2503257>>
58 Disponível em <<https://vimeo.com/2579694>>
59 Disponível em <<https://vimeo.com/2579880>>
60 Disponível em <<https://vimeo.com/2735394>>
61 Disponível em <<https://vimeo.com/5690854>>
62 Disponível em <<https://vimeo.com/7339135>>
63 Disponível em <<https://vimeo.com/8364077>>
64 Disponível em <<https://vimeo.com/8732631>>
65 Disponível em <<https://vimeo.com/15585520>>
66 Disponível em <<https://vimeo.com/15679078>>
67 Disponível em <<https://vimeo.com/18008372>>

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

Exemplo 3.18 (Definição de tempo)

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

3.4.2 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 3.19 (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):

Exemplo 3.20 (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*⁶⁸:

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).⁶⁹

⁶⁸ Disponível em <https://github.com/digego/extempore/blob/master/libs/core/pc_ivl.xtm>

⁶⁹ 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*”

Este bloco de códigos cria uma díade, 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)”⁷⁰. 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.

3.4.3 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 por [Sorensen \(2015\)](#).

Exemplo 3.21 (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 3.22 (Estratégia transversal)

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

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

⁷⁰ Tradução nossa de: (...) *a chord following basic diatonic harmony rules: based on root (0 for C etc.) maj/min ('- or '^) and degree (i-vii).*

Exemplo 3.23 (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 3.24 (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 3.25 (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 ($dur \Rightarrow (*metro* 'dur dur)$)⁷¹.

3.4.4 Primeira sonoridade tonal

Este código inicial é então modificado, e finalizado em 1'57", momento em que é possível ouvir uma figura musical (uma classe de objeto \mathcal{O}'_{ask}), 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 3.26 (Estratégia transversal)

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

⁷¹ Disponível em <<https://github.com/digego/extempore/blob/5aec8b35c6b3058d1c66de7abf752dc667ab61e4/libs/core/instruments-scm.xtm>>

```

(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, transcrevemos este trecho com um andamento entre 35–40 BPM (ver Figura 11, p. 50).

The image shows two musical staves for piano. The left staff is a literal transcription of the first event, showing a treble clef with a dotted quarter note and an eighth note, and a bass clef with a dotted quarter note and an eighth note. The right staff is a perceptual transcription of the same event, showing a treble clef with a quarter note and an eighth note, and a bass clef with a quarter note and an eighth note, with dynamic markings 'f' and 'p'.

Figura 11 – 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 3.27 (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 \mathcal{T}_{ask} . Se o grau a ser executado for uma tônica, no caso, menor, a duração deste acorde será configurada para uma duração de três unidades de tempo – no caso da nossa transcrição, uma unidade de pulso.

```
(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 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))))
```

3.4.5 Regras de qualidade

A estrutura interna da estratégia `chords` explicita algumas regras de qualidade, bem como permite apresentar uma primeira sequência 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 3.28 (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)
```

3.5 Primeira sequência de blocos de eventos

A [Figura 12](#) indica uma primeira sequência de notas geradas pelo algoritmo, um padrão que é repetido por dois ciclos (blocos de eventos \mathcal{E}_{ask}^0 e \mathcal{E}_{ask}^1). Durante este tempo, Sorensen realiza uma mudança (ver [Figura 1](#), p. 1). Esta mudança transita entre o segundo bloco \mathcal{E}_{ask}^1 e terceiro bloco \mathcal{E}_{ask}^2 , e sua execução resulta em uma transformação da acentuação, o que termina por colocar, no último compasso deste ciclo, o sétimo grau no tempo forte e o primeiro grau no tempo fraco.

3.5.1 Primeiro Bloco

O primeiro bloco de eventos \mathcal{E}_{ask}^0 apresenta um contraponto de primeira espécie, aticulado em tempos fortes e fracos, de acordo com um movimento cadencial $i \Rightarrow vii$ (ver [Figura 12](#), p. 52).

Piano

The musical score for Piano is in 2/4 time and consists of four measures. The right hand (treble clef) plays a sequence of notes: C4, D4, E4, F4, G4, A4, B4, C5. The left hand (bass clef) plays a sequence of notes: C3, B2, A2, G2, F2, E2, D2, C2. The notes are grouped into pairs, with the first note of each pair being accented. The dynamics are marked as *f p* for each pair of notes.

Figura 12 – Primeiros eventos musicais gerados a partir das primeiras estruturas válidas de código. **Fonte:** autor.

Esta cadência descaracteriza *A Study in Keith* ser uma peça, do ponto de vista harmônico, semelhante aos Concertos *Sun Bear* de Keith Jarret. Em outras palavras, esta é uma razão pela qual *A Study in Keith* é apenas inspirado por Keith Jarret, mas não é Keith Jarret, no sentido harmônico que a expressão carrega (ver [seção 3.2](#), p. 34).

Uma pequena comparação com o plano harmônico do concerto de Kyoto (ver [subseção 3.3.1](#), p. 35) com primeiro bloco de *A Study in Keith* pode elucidar a questão. Enquanto em Jarret temos uma estrutura cromática oblíqua a um baixo pedal, em Sorensen temos uma estrutura diatônica e com movimentos contrários:

Exemplo 3.29 (Redução do primeiro bloco)

Movimentos da voz superior ($\hat{1}-\hat{4}-\hat{1}$) e inferior ($\hat{5}-\hat{2}-\hat{3}$).

Musical notation for the first block of events. The score is in 2/4 time. The treble clef part has notes G4, A4, Bb4, and A4. The bass clef part has notes G3, F3, and E3. Fingerings are indicated by red numbers with hats: $\hat{1}$ for the treble clef and $\hat{5}$, $\hat{2}$, $\hat{3}$ for the bass clef. Chord symbols *i*, *vii*, and *i* are shown below the bass clef part.

3.5.2 Segundo Bloco

O Segundo bloco dá continuidade ao contraponto inicial, de forma que o âmbito melódico da voz superior é mantida e da voz inferior é restringida:

Musical notation for the second block of events, labeled "Piano". The score is in 2/4 time. The treble clef part has notes G4, A4, Bb4, A4, G4, F4, E4. The bass clef part has notes G3, F3, E3, D3, C3, B2, A2. Dynamics *f* and *p* are indicated below the bass clef part. Fingerings are indicated by red numbers with hats: $\hat{1}$ for the treble clef and $\hat{5}$, $\hat{2}$, $\hat{3}$ for the bass clef.

Figura 13 – Segundo bloco de eventos musicais. **Fonte:** autor.

Exemplo 3.30 (Redução do segundo bloco)

Movimento da voz superior ($\hat{1}$ - $\hat{5}$ - $\hat{1}$) e inferior ($\hat{5}$ - $\hat{2}$ - $\hat{3}$).

Musical notation for the reduced second block of events. The score is in 2/4 time. The treble clef part has notes G4, A4, Bb4, and A4. The bass clef part has notes G3, F3, and E3. Fingerings are indicated by red numbers with hats: $\hat{1}$ for the treble clef and $\hat{5}$, $\hat{2}$, $\hat{3}$ for the bass clef. Chord symbols *C:i64*, *vii43*, and *i6* are shown below the bass clef part.

3.5.3 Terceiro Bloco

Enquanto nos blocos \mathcal{E}_{ask}^0 e \mathcal{E}_{ask}^1 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 3.31 (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 14](#), p. 55).

Exemplo 3.32 (Redução do terceiro bloco)

Movimento da voz superior ($\hat{1}-\hat{2}-\hat{1}$) e da voz inferior ($\hat{5}-\hat{7}b-\hat{3}$).

Transcrição do terceiro bloco

Piano

f p f p f p f p

f p f p f > p f p > f p

Figura 14 – Terceiro bloco de eventos musicais. Fonte: autor.

1 2 1

5 (5) 3

C:i64 **'v/iv** **i6**

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

Exemplo 3.33 (Laço iterativo modificado)

```
(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)))
```

A primeira grande mudança é a definição de duas variáveis internas, através do comando `let` (seja), chamadas `dur1` e `dur2`:

Exemplo 3.34 (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 diminuído (`(* 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 aplicada uma verificação condicional:

Exemplo 3.35 (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), a nota tocada dependerá do resultado de `pc:relative`. A função `pc:relative` é definida⁷² como seleção de uma altura, de uma classe de alturas relativa à uma dada altura⁷³. Sua altura serão dadas em passos de segundas menores ou maiores ascendentes/descendentes, relativas ao modo eólico da escala (que no caso transforma a sonoridade tonal em sonoridade modal).

Exemplo 3.36 (Laço iterativo modificado)

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

O ritmo da mão esquerda será semelhante ao da mão direita.

⁷² Disponível em https://github.com/digego/extempore/blob/master/libs/core/pc_ivl.xtm

⁷³ Tradução nossa de “*select pitch from pitch class relative to a given pitch*”

Exemplo 3.37 (Laço iterativo modificado)

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

No entanto esta característica é um dos fios condutores de uma seção $\mathcal{K}'_{ask,1}$, o que excede um objetivo deste documento. Nosso interesse nesta análise foi investigar, através do estudo de contextos, e de diferentes notações musicais (código, partitura, neuma e esquema analítico), de uma mesma música, a sonoridade que irá gerar outras sonoridades, no caso desta pesquisa, uma sonoridade com raízes em esquemas tonais.

3.6 Discussão

Este capítulo buscou analisar uma improvisação publicada em 2009 por Andrew Sorensen. Levantamos um conjunto de informações que estimularam o improvisador-programador a realizar *A Study in Keith*. A partir de uma outra improvisação, definida como *referencial zero*, ou os *Concertos Sun Bear*, Sorensen buscou simular um estilo de jazz do pianista e compositor Keith Jarrett. No entanto, esta relação é pouco significativa do ponto de vista harmônico. Na nossa transcrição de uma cadência da abertura do disco *Kyoto I*, encontramos uma exploração da cadência plagal do blues por substituição de trítone. Já em Sorensen o procedimento é uma simples cadência *i – vii*. A partir desta regra, foi possível apontar figuras musicais separadas em três blocos, o que delineou um método de análise para outras improvisações de código. Por outro lado, nossa análise não contemplou sequências seguintes, o que impediu observar detalhes sobre o processo geral da improvisação. Nosso interesse residiu em investigar o ciclo de elaboração/codificação apresentado na [Figura 1](#) (ver p. 1) de um ponto de vista cognitivista.

Conclusão

Este documento contextualizou o que é uma improvisação de códigos, e como podemos entender suas manifestações quando se codificam artefatos sonoros, visuais, corporais e têxteis. Diferente de outras definições, buscamos situar o *live coding* como uma técnica de programação que possibilita elaborar proposições artísticas de qualquer tipo. Ward et al. (2004) definem a improvisação de códigos como “atividade da escrita integral (ou partes) de um programa enquanto ele é executado”⁷⁴. 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) definem “uma prática de performance para o qual linguagens de computador definem o meio primário de expressão artística”.⁷⁵ Para Sorensen e Gardner (2010), *live coding* 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 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)⁷⁶.

Magnusson (2011), Collins (2014) sintetizam o *live coding* como improvisação audiovisual. Sorensen (2014) define como “programar sistemas de tempo-real em tempo real”⁷⁷. Uma discussão intitulada “*Wtf is livecoding*”⁷⁸ dificultou o próprio processo de definição, onde o compositor Nick Collins diz que o “*Live coding* celebra a efemeridade da própria definição”⁷⁹. Aqui fica explícito uma *bricolagem* da técnica. Sorensen (2014) destaca que modificar alguma coisa é próprio da técnica, de forma que é possível estender essa bricolagem para proposições. Nick Collins (2014) situa essa questão da seguinte forma:

Neste ponto encontramos um desafio à metodologia de pesquisa acadêmica: se o termo que contextualiza um estudo de caso é variável por definição, como analisar este caso?

⁷⁴ Tradução nossa de “*Live coding is the activity of writing (parts of) a program while it runs*”

⁷⁵ Tradução nossa de “*Live coding is a performance practice for which computer languages define the primary means of expression.*”

⁷⁶ 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*

⁷⁷ Tradução nossa de “*programming real-time systems in real-time*”

⁷⁸ Disponível em <<http://lurk.org/groups/livecode/messages/topic/ofAxZpxsKFpDRLnoA48Bh>>

⁷⁹ Tradução nossa de “*Live coding celebrates the ephemerality of definition itself*”

Live coding

<INSERT DEFINITION HERE>

Figura 15 – Definição de *live coding*: “Insira a definição aqui”. Fonte: Collins (2014).

Foi necessária uma revisão histórica para entender como a atividade de programar, no sentido de *tocar um instrumento*, se construiu historicamente. Ali noções como *reflexividade* e instrumentos acusticamente divorciados mas informacionalmente ligados, possibilitaram o artistas-programadores se auto-organizarem dentro de suas agremiações. Uma dessas agremiações merece destaque, e surgiu no seio acadêmico inglês. De certa forma esta agremiação prescreveu regras de conduta para improvisar códigos, bem aceitas entre diversos artistas-programadores. Desta agremiação que se tornou uma organização, selecionamos um caso exemplar que segue as regras estipuladas. Investigamos uma improvisação de códigos como uma simulação de uma improvisação instrumental. Esta improvisação instrumental, para facilitação do que pode ser harmonicamente simples, possibilitou destacar a diferença entre o que foi proposto e aquilo que foi plenamente executado.

Por outro lado, julgar *A Study in Keith* apenas com base na sua simplicidade harmônica, se comparada ao pensamento harmônico inicial dos Concertos *Sun Bear*, é obscurecer a possibilidade de uma metodologia composicional, neste caso, uma espécie de desenvolvimento orientado a testes⁸⁰ musicais. Ao aprofundarmo-nos em uma escavação etnográfica (MORI, 2015a), foi possível notar que o mesmo mecanismo de códigos é utilizado em uma performance com pianos acústicos, ou *Disklavier Sessions* de 2011. Neste sentido, o recorte desta pesquisa foi determinar como um algoritmo musical é elaborado e codificado por um artista-programador, aplicado ao desenvolvimento de *softwares*.

⁸⁰ Tradução nossa de “*Test-driven development*”

Referências

- 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 23 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 6 vezes nas páginas xiii, 17, 18, 19, 20 e 21.
- 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 6.
- 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 na página 5.
- COLLINS, N. *Origins of live coding*. Durham University, 2014. Disponível em: <<http://www.livecodenetwork.org/files/2014/05/originsoflivecoding.pdf>>. Citado 3 vezes nas páginas xiii, 59 e 60.
- 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 na página 6.
- 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 7.
- 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 7.
- 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 7.
- 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 24.
- 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: <https://www.researchgate.net/publication/220636741_Unifying_Conceptual_Spaces_Concept_Formation_in_Musical_Creative_Systems>. Citado 3 vezes nas páginas 7, 31 e 32.

- 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 35.
- 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 13 e 15.
- GRIFFTHS, D. *A cryptoweaving experiment*. 2015. Disponível em: <<http://kairotic.org/a-cryptoweaving-experiment>>. Citado 4 vezes nas páginas xiii, xv, 2 e 3.
- GRIFFTHS, D. *Weavecoding performance experiments in Cornwall*. 2015. Disponível em: <<http://www.pawfal.org/dave/blog/tag/weavecoding/>>. Citado 3 vezes nas páginas xiii, 3 e 4.
- KARCHER, U. *Musique algorithmique*. K. Jarrett - Kyoto 1976, Part I / II (Medley) transcribed and performed by Uwe Karcher, 2009. Disponível em: <<https://www.youtube.com/watch?v=BMh68J3HcGo>>. Citado 3 vezes nas páginas xiii, 35 e 36.
- MAGNUSSON, T. Algorithms as scores: Coding live music. v. 21, p. 19–23, 2011. Citado 2 vezes nas páginas 38 e 59.
- 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 24.
- 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 14.
- 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 xvi e 28.
- MATHEWS, M. V. et al. *The technology of computer music*. 2a, 1974. ed. [S.l.]: MIT press, 1969. ISBN 0 26213050 5. Citado 2 vezes nas páginas xvi e 28.
- 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 na página 40.
- McLean, A. Music improvisation and creative systems. online, p. 6, 2006. Disponível em: <https://www.academia.edu/467101/Music_improvisation_and_creative_systems>. Citado 3 vezes nas páginas 31, 32 e 33.
- McLean, A. *hacking perl music*. Youtube, 2006–07–30. Disponível em: <<https://www.youtube.com/watch?v=fbefIdbSmD4>>. Citado na página 59.
- 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 7 vezes nas páginas xiii, xv, 1, 5, 7, 24 e 31.

- 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 22.
- 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 2 vezes nas páginas xv e 60.
- 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 5 vezes nas páginas 13, 14, 15, 16 e 17.
- 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 na página 28.
- 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. 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 2 vezes nas páginas 31 e 33.
- 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 3.
- RAMSAY, S. *Algorithms are Thoughts, Chainsaws are Tools*. Vimeo, 2010. Disponível em: <<https://vimeo.com/9790850>>. Citado 3 vezes nas páginas xiii, 24 e 25.
- 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 na página 24.
- RIETVELD, H. C. Bloomsbury Publishing Inc., 2013. 1–14 p. Disponível em: <<http://file.ebook777.com/005/DjCulInTheMixPowTecAndSocChaInEleDanMus.pdf>>. Citado na página 4.
- 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 22.
- ROADS, C. *Microsound*. [S.l.]: MIT press, 2001. Citado na página vi.
- 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.
- 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/~schloss/publications/JNMR02_Dilemma_of_the_Performer.pdf>. Citado 4 vezes nas páginas xvii, 13, 26 e 27.
- 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 7.

SICHIO, K. *Kate Sicchio, Hacking the Body 2.0 at Stage@Leeds, University of Leeds / HTB2.0 - Kate Sicchio Concert D, Tuesday*. Youtube, 2015. Disponível em: <<https://www.youtube.com/watch?v=hKs3i1hEo7E>,<https://www.youtube.com/watch?v=iOAffWTBVE0>>. Citado na página xvi.

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 39.

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>. Nenhuma citação no texto.

SORENSEN, A. *Disklavier sessions*. Vimeo, 2013. Disponível em: <<https://vimeo.com/50061269>>. Citado na página 37.

SORENSEN, A. *Programming in Time - Live Coding for Creative Performances*. 2014. Disponível em: <<https://www.youtube.com/watch?v=Sg2BjFQnr9s>>. Citado na página 59.

SORENSEN, A. *The Disklavier Sessions*. Youtube, 2015. Disponível em: <<https://www.youtube.com/watch?v=cFEadvBeBqw>>. Citado 4 vezes nas páginas xvii, 34, 37 e 48.

SORENSEN, A.; GARDNER, H. Programming with time: cyber-physical programming with impromptu. p. 822–834, 2010. Disponível em: <<http://diyhpl.us/~bryan/papers2/paperbot/67845a4fb5b009259c389f90ab02c1c0.pdf>>. Citado 4 vezes nas páginas 38, 41, 45 e 59.

SORENSEN, A.; SWIFT, B. *A Study in Keith*. Vimeo, 2009. Disponível em: <<https://vimeo.com/2433947>>. Citado 6 vezes nas páginas vii, xvii, 34, 36, 37 e 59.

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 xiii, 45 e 47.

TEREFENKO, D. *Keith Jarrett's Transformation of Standard Tunes*. — UFJF, 2004. Disponível em: <<https://urresearch.rochester.edu/institutionalPublicationPublicView.action?institutionalItemId=27134>>. Nenhuma citação no texto.

TRUAX, B. Review of pietro grossi: 24 capricci by niccolò paganini. v. 8, p. 59–60, 1984. Citado na página 14.

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 22.

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 9 vezes nas páginas xvi, xvii, 22, 23, 24, 25, 26, 28 e 59.

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 na página 32.