Guilherme Martins Lunhani

Live Coding: um algoritmo de sonoridade tonal em *A Study in Keith* (2009) de Andrew Sorensen

Guilherme Martins Lunhani

Live Coding: um algoritmo de sonoridade tonal em *A Study in Keith* (2009) de Andrew Sorensen

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

Universidade Federal de Juiz De Fora – UFJF

Instituto de Artes e Design – IAD

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

Orientador: Prof. Dr. Luiz Eduardo Castelões

Guilherme Martins Lunhani

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

69 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Luiz Eduardo Castelões

Tese (Mestrado) – Universidade Federal de Juiz De Fora – UFJF
Instituto de Artes e Design – IAD
Programa de Pós-Graduação em Artes Visuais, Música e Tecnologia, 2016

1. Livecoding. 2. Study in Keith. 3. Sistemas criativos I. Orientador: Prof. Dr. Luiz Eduardo Castelões Pereira da Silva II. UFJF - Universidade Federal de Juiz de Fora. III. Instituto de Artes e Design IV. *Live Coding*: um algoritmo de sonoridade tonal em *A Study in Keith* (2009) de Andrew Sorensen

CDU XX:XX:XXX.X

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. Editoral Attar. Ver também <http://tao-te-king.org/42.htm>.

Resumo

Este documento apresenta 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. xi) apresentamos uma definição de *live coding*. A definição destaca o fazer musical, mas não exclúi 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 proposta de ação, um vídeo entitulado *A Study in Keith*, de Sorensen e Swift (2009), a partir de uma ferramenta de análise deste mecanismo.

Uma é a contribuição deste trabalho para a musicologia brasileira: um documento que organiza conceitos básicos de uma técnica ainda pouco elaborada em território nacional.

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

Sumário

Lista de ilustrações

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 Griffths (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 computadores[5].

[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. Griffths, ao lado de Alex McLean e Adrian Ward, realizam, no começo dos anos dois mil, aprensentações 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 estadounidense 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 Franscisco, 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 entitulado *Live algorithm programming and temporary organization for its promotion*, ou simplismente 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. xi), 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:

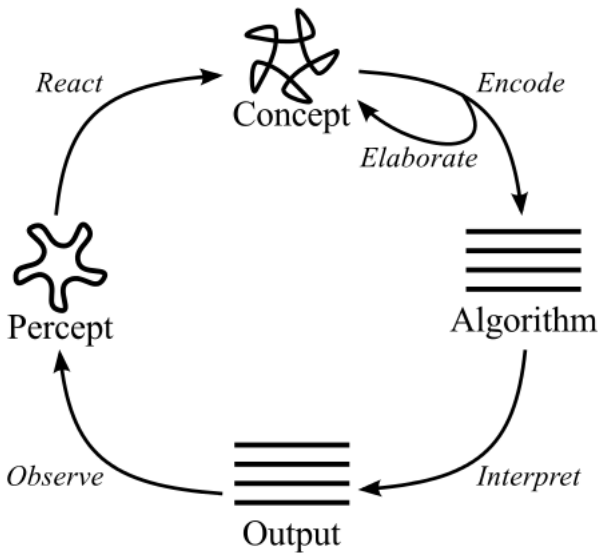


Figure 6.2: *The process of action and reaction in bricolage programming*

Figura 1 – O Processo de ação e reação no desenvolvimento de programas *guiados por* bricolagem. Substituimos, 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, Copenhague; 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 Griffths (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*"

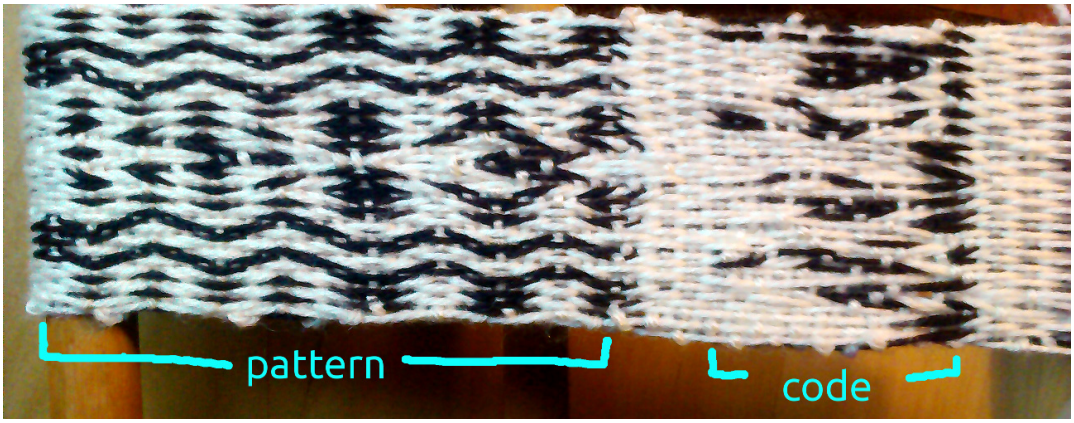


Figura 2 – Tecido resultante da prática *Weaving code*. **Fonte**: Griffths (2015a).

Esta experiência de *participação social* [3] foi co-orientada por Griffths 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 portátil[4], uma matriz de botões acopláveis, desenvolvida por Ellen Harlizius-Klück (investigadora da história da matemática, filosofia e tecelagem da Grécia Antiga na Universidade de Copenhague[5]) e Alex McLean (artista-programador de destaque). Imagens em movimento foram projetadas como capturas das atividades têxteis e processadas por Griffths.

Figura 3 – Performance no Foam Kernow. **Fonte**: Griffths (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**: Griffths (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 envolvolvidos 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 Griffths possui uma documentação audiovisual [7].

1.1.1 Slub

A parceria entre Griffths 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 Dançar [8].

[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 Griffths se juntou ao duo durante o festival *Sonar* em Barcelona, momento em que iniciaram um processo *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 eforços no desenvolvimento de ambientes de improvisação de códigos, estruturados como editores de linguagens textuais ou visuais, e interfaces gráficas de usuário[10]: "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 beign faced with the challenge of beign 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 'comecou como uma piada', de acordo com Alex McLean, um pesquisador de música computacional e um dos três de uma banda chamada *Slub*, que têm improvisado códigos por 13 anos. Ele veio com um termo enquanto conduzia uma *gig* em Nottingham com seu amigo Nick Collins (que tocava "datapop" sob o nome Sick Lincoln) no final de 2011. 'Nós sintonizamos em uma estação pirata tocando *happy hardcore*, e nós pensamos que seria bom programar alguma música *rave*.' Deste então, McLean organizou oito *algoraves* informais no mundo. "[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 musica 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.

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 computação"[19] (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 OSC[23] para 'zunir' sobre os lados direito e esquerdo da dançarina para indicar qual lado do corpo a dançarina deve mover. A partitura é codificada ao vivo pela coreógrafa enquanto a dançarina responde por uma retroalimentação 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.* Disponivel 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 estadounidense (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 B[4]. 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 5^o 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*, *ibdem*). 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*.(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 intéprete era capaz de produzir e reproduzir música em tempo real, digitando alguns comandos específicos e os parâmetros composicio-

[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.*"

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

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*, *ibdem*)."[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 sobretom 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 Franscisco

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 Franscisco, o acesso às novas tecnologias e pessoas que desenvolveram elas era talvez o melhor no mundo. Mas se para todos os jovens com fortunas como panos para suas mentes (e seus futuros), que perseguiam um excitamento aditivo na construção de máquinas eletrônicas, também existiam políticos utópicos que sonhavam com uma nova sociedade construída no livre e aberto acesso à informação, e na abrangente tecnologia baseada em sistemas inteligentes. Esta também é a cultura que deu ao mundo a música "New Age", uma versão aguada e comercializada das músicas com base em modos e drones que Terry Riley, Pauline Oliveros, e LaMonte Young inventaram durante os anos cinquenta e sessenta. Mas a música feita na Costa Oeste também incluiam improvisações barulhentas e 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 micro-controladores KIM-1[17] com interesses musicais. Segundo Brown e Bischof (2002), não demorou para que outros compositores interessados comprassem. Discussões informais posteriores incluiram, 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 1958[20]. 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.

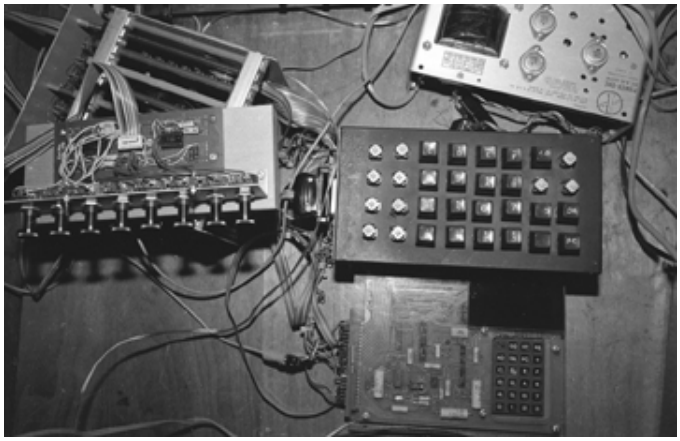


Figura 6 – Sistema de música computacional de John Bischof *circa* 1980. Foto: Eva Shoshanny[22]. **Fonte**: Brown e Bischof (2002).

[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.chickenonaunicycle.com/Berkeley%20Art.htm>

[21] Segundo Brown e Bischof (2002, 6ºparágrafo), 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 lembre. (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 Franscisco. 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 sistemas[24](BROWN; BISCHOF, 2002, 7^o parágrafo). Outro evento de importância é a associação do grupo com a banda *Rotary Club*, formada por alunos recém-formados da *Mills College*: Sam Ashley, Kenneth Atchley, Ben Azarm, Barbara Golden, Jay Cloidt e Brian Reinbolt. O grupo "baseava seu estilo de performace em torno de uma caixa de comutação projetada por Brian Reinbolt"[25](BROWN; BISCHOF, 2002, 8^o parágrafo). Em 1983 o grupo reduziu suas atividades, época em que Horton contraiu artrite degenerativa.

Brown e Bischof (2002, 11^o parágrafo) suas redes de composições, ou "ocasiões públicas para escuta compartilhada"[27]. O som era produzido por um sistema limitado, de "baixa velocidade (1 *MHz*) e poucos dados (8 *bits*)"[28] com ênfase em uma artesania 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*)*"

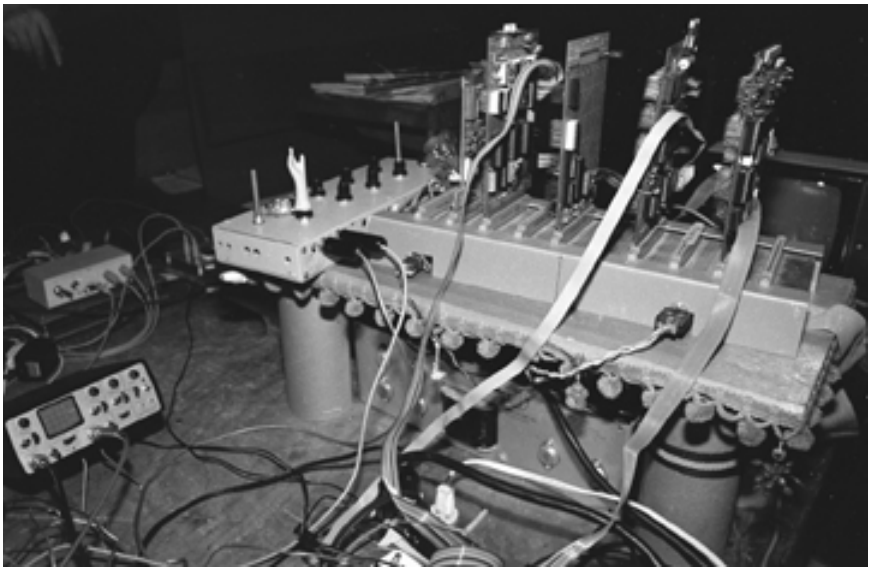


Figura 7 – Circuito do computador caseiro dedicado à síntese sonora de Tim Perkis, usado no começo dos anos 1980. Foto: Eva Shoshany[26]. **Fonte**: Brown e Bischof (2002)

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, 22^o 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, 12^o parágrafo)."[30]

2.2.2 The Hub

O primeiro *The Hub* foi surgiu 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 sitemas 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 STEIM [34], 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 cirar sons densos, rodopiantes e métricos, disposto em camadas e dobravam sobre si. Considerando o equipamento usado, os sons eram surpreendentemente grandes em escala. Kuivila teve problemas em controlar a peça devido q problemas sistêmicos. Ele finalmente entrou em dificuldades técnicas e finalizou a performance (ROADS, 1986, p. 47)[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 computeronstage to create dense, whirling, metric sounds that layered in and folded over each other. Considering the equipment used, the sounds were often surprisingly gigantic in scale. Kuivila had trouble controlling the piece due to system problems. He finally gave in to technical difficulties and ended the performance*

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

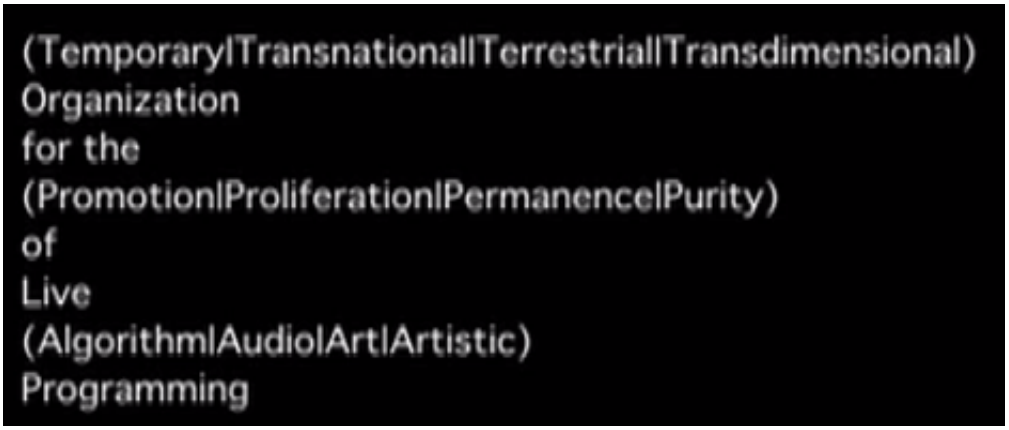


Figura 8 – Definição do siginificado de TOPLAP. **Fonte**: Ramsay (2010).

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/impressiva 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; alí 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**

[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/impressive display of mental dexterity • No backup (minidisc, DVD, safety net computer) . We acknowledge that: • It is not necessary for a lay audience to understand the code to appreciate it, much as it is not necessary to know how to play guitar in order to appreciate watching a guitar performance. • Live coding may be accompanied by an impressive display of manual dexterity and the glorification of the typing interface. • Performance involves continuums of interaction, covering perhaps the scope of controls with respect to the parameter space of the artwork, or gestural content, 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.*

> **existe ligação. Isto significa que nós podemos ir além da relação de causa-e-efeito entre executante e instrumento que faz a mágica.** Mágica é bom; muita mágica é fatal."[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 extendidos, o problema do movimento exagerado, e a expectativa cênica na performance musical:

> "1. Causa-e-efeito é importante, pelo menos para o observador/audiência em uma sala de concerto.
> 2.Corolário: Mágica na performance é bom. Muita mágica é fatal! (chato).
> 3. Um componente visual é essencial para a audiência, tal como existe um aparato visual de entrada para parâmetros e gestos.
> 4. Sutileza é importante. Grandes gestos são facilmente visíveis de longe, o que é bom, mas eles são movimentos de desenho animado se comparados à execução de um instrumento musical.
> 5. Esforço é importante. Neste sentido, nós estamos em desvantagem de desempenho na performance musical com o computador.
> 6. Improvisação no palco é bom, mas "mimar" o aparato no palco não é improvisação, é edição. É provavelmente mais apropriado fazer isso no estúdio antes do concerto, ou se durante o concerto, com o console no meio ou atrás da sala de concerto.
> 7. Pessoas que representam devem representar. Um concerto de música de computador não é uma 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 toperform live, we have new possibilities, and a new problem.From the beginning of the archeological evidence of musicuntil now, music was played acoustically, and thus it wasalways physically evident how the sound was produced; there was a nearly one-to-one relationship between gesture andresult. Now we don't have to follow the laws of physicsanymore (ultimately we do, but not in terms of what theobserver observes), because we have the full power of computers as interpreter and intermediary between our physicalbody 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 playi ng 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 Mathews [50], 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 estadounidense 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 infomações 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: em comunicação pessoal com o compositor James McCartney, é afirmado que "Se ele tivesse feito isso com um BeOS, Ron poderia ter usado *is computer on fire()*"[51,52]. Em outras palavras, é possível que se trate de uma programada pelo compositor.

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

[51] Tradução de *If he'd done it on BeOS, Ron could've used the is computer on fire() API.*

[52] Disponível em http://www.tycomsystems.com/beos/bebook/The%20Kernel%20Kit/index.html

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

3 Estudo de caso

A pesquisa desenvolvida nos capítulos anteriores sucitaram 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 de um ponto de vista cognitivista, de forma que será útil contextualizar o método de análise a partir dos termos *conceito* e *instância de um conceito* (ver seção 3.1, p. 31), sugeridas por McLean (2011, p. 117):

> "Aqui nós tomamos a perspectiva que uma propriedade conceitual é representada pelo 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

A análise de *A Study in Keith* será feita com base em um recorte do *Quadro Conceitual de Sistemas Criativos*[2] (ver Tabela 1, p. 32). Este quadro se baseia no modelo de improvisação de Jeff Pressing (1987) (ver subseção 3.1.1, p. 33). É importante esclarecer que, o que os autores ingleses chamam de *conceito*, entendemos ao longo deste trabalho como proposição para não confundirmos questões ontológicas. Desta forma analisamos uma única proposta de ação indicada por Sorensen (ver seção 3.2, p. 34)

[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 Quadro Conceitual de Sistemas Criativos McLean (2006), Forth, Wiggins e McLean (2010).

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 linguagen que contêm caracteres apropriadospara expressão das regras
$\mathcal{R}$	Regras de validação	Validam os conceitos em um universo, se apropriados ou não para o espaço trabalhado.
$[[.]]$	Função de interpretação	"Uma função parcial de $\mathcal{L}$ para funções que resultam em números reais entre [0, 1] (...) 0.5 [ou maior] significa uma verdade booleana e menos que 0.5 siginifica 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 permitem-nos avaliar qualquer conceito que nós encontramos em $\mathcal{C}$ e determinar sua qualidade, de acordo com critérios que nós considerarmos apropriados" (WIGGINS, 2006, p.453)[9]"Regras definidas dentro de $\mathcal{L}$ para avaliar a qualidade ou a desejabilidade do conceito c" (McLean, 2006)[10]
$<<< \mathcal{R}, \mathcal{T}, \mathcal{E} >>>$	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 $mathcalU$ (do qual tem acesso randômico) e resulta em outro subconjunto ordenado de $\mathcal{U}$."[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, *A Study in Keith* pode ser particionada em diferentes sequências e mapeada em categorias, como blocos sonoros, músicas de referência secundária, normas estilísticas, conjuntos de objetivos e processos Tabela 2. . Este nos pareceu um modelo mais transparente para a análise de um bloco sonoro de *A Study in Keith*.

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 sequências[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 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 anordered subset of mathcalU (of which it has random access) and results in another ordered subset of $\mathcal{U}$.*

3.2 *A Study in Keith*: Proposição

São duas as descrições de uma mesma proposição, ou *Espaço conceitual de A Study in Keith* ($\mathcal{E}_{ask}$). Andrew Sorensen cita também os Concertos *Sun Bear* de Keith Jarret (ver seção 3.3, p. 34) como inspiradores da improvisação de códigos. Desta forma, estes conceitos são um *Referencial Zero de A Study in Keith*, ou $\mathcal{R}'^{0}_{ask}$.

> "*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 e Dois de A Study in Keith* (ver subseção 3.3.3, p. 38), ou o ambiente de programação *Impromptu* ($\mathcal{R}'^{1}_{ask}$) e a linguagem de programação *Scheme* ($\mathcal{R}'^{2}_{ask}$):

> "*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://impŕomptu.moso.com.au>). Não é Keith, mas inspirado por Keith (SORENSEN, 2015). "[23]

3.3 Referentes Opcionais

A Study in Keith possui um referencial zero, $\mathcal{R}'^{0}_{ask}$, ou os Concertos *Sun Bear* de Keith Jarret (ver subseção 3.3.1, p. 35). Nesta seção, indicamos uma regra harmônica sugerida por *Sun Bears* não realizada por *A Study in Keith*. Em seguida tratamos do referencial um, $\mathcal{R}'^{1}_{ask}$, ou o timbre de piano utilizado (ver subseção 3.3.2, p. 37), e de um

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. "*

ambiente de programação musical chamado *Impromptu* (ver subseção 3.3.3, p. 38) como referencial dois $\mathcal{R}'^{2}_{ask}$.

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. É o terceiro dos concertos de improvisação que incluem o *Solo Concerts: Bremen/Lausanne* (1973) e *The Köln Concert* (1975).

Foram realizados e gravados como sessões de improvisação contínua, variando entre 31 a 43 minutos cada. Para cada dia, duas sessões de improvisação, em cidades diferentes. Kyoto, 5 de novembro[25]; Osaka, 8 de novembro[26]; Nagoya, 12 de novembro[27]. Tokyo, 14 de novembro[28]; Sapporo, 18 de Novembro[29].

Existem algumas notas discográficas compiladas por uma comunidade de fãs e críticos musicais estadounidenses. Duas notas sugerem uma descrição da forma musical aplicada por Keith Jarret: "O tema de *Kyoto Parte 1* é repetido por Keith Jarret no fim de *Kyoto Parte 2*. Então podemos considerar o todo deste concerto como uma grande Suíte."[30](GARBOLINO, 2014, p. 129).

> Revisto por Richard S. Ginnel[31]: [–] Este pacote gigantesco – um conjunto de dez LPs agora comprimidos em uma caixa robusta de seis [embalagens de] CDs – foi ridicularizado uma vez como uma última viagem de ego, provavelmente por muitos que não tomaram um tempo para ouvir tudo. (. . .) Ainda assim, o milagre é como esta caixa é consistentemente muito boa. **Na abertura de Kyoto, a meditação direcionada para o *gospel*** está em plena atuação, ao nível de suas melhores performances solo em Bremen e Koln,**e os concertos Osaka e Nagoya possuem citações de primeira linha, geralmente do tipo *folk***, mesmo profundas, idéias líricas (GARBOLINO, 2014, p. 130) [32].

[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.*

O *gospel* e o *folk* são categorizados como gêneros musicais nesta suíte que não possui pausas entre as partes (o improviso é contínuo, mas seccionado por transições). Seu motivo gerador é, segundo Uwe Karcher (2009), o tema de *A song of the heart*:

> "**Na verdade, a abertura não é realmente improvisada - ela é baseada em uma música chamada "Song Of The Heart"**[33]. Eu tenho interesse especial em transcrever os primeiros 10, 11 minutos (que são simplesmente fenomenais). Por fim, eu usei a Reprise que Keith jogou no final da Parte II."[34]

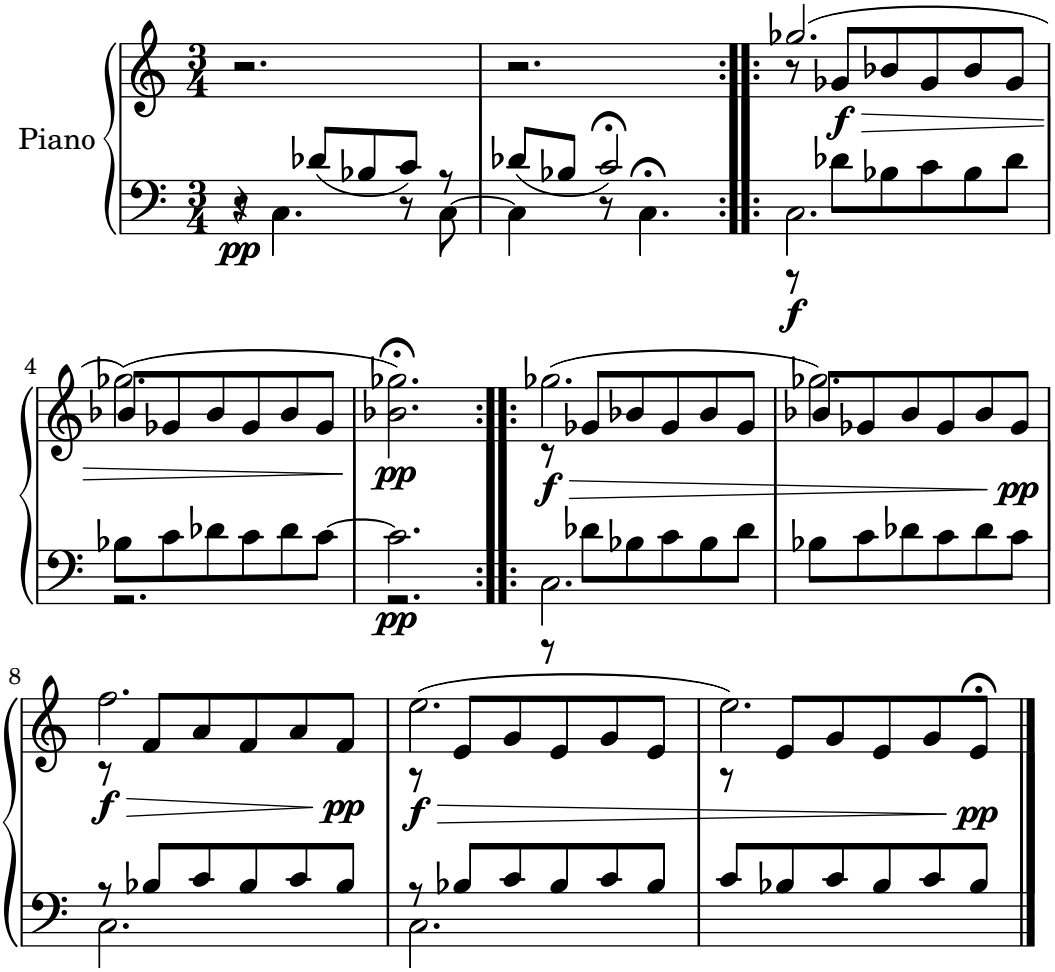


Figura 9 – Transcrição do motivo gerador do disco Kyoto, parte 1. textbfFonte: Uwe Karcher (2009).

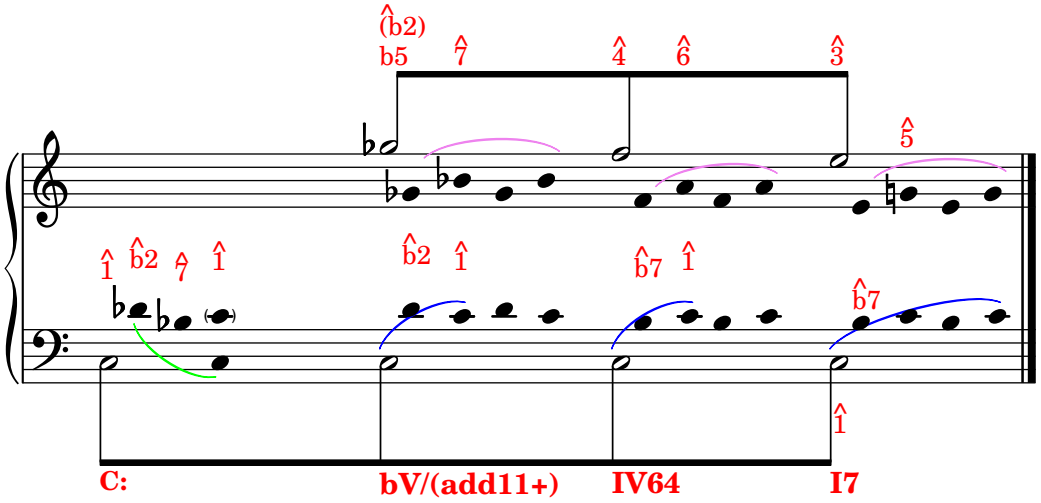
A questão se Jarret ou não improvisou este tema cria uma outra pergunta: o código elaborado por Sorensen e Swift (2009), e sua sonoridade resultante, é de fato uma

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

[34] Tradução nossa de "*My transcription of the famous opening of the concert played by Keith in Kyoto on November 5th, 1976. Released in a 6-CD-Box-Set called "Sun Bear Concerts"(ECM). A must have for all (Jazz)piano enthusiasts! Actually, the opening was not really improvised - it is based on a tune named "Song Of The Heart". I have been interested particularly in transcribing the first 10, 11 minutes (which are simply phenomenal). To come to an end, I used the Reprise which Keith played at the end of Part II. To memorize these 34 pages was pretty ambitious, but it worked :) Hope you enjoy!*"

improvisação de códigos, ou existe um agenciamento onde o improvisador prepara um código, e define um objetivo inicial?

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 $\hat{13}$[35]) e Dó Maior com sétima menor (posição fundamental). Limitamo-nos a considerar a progressão do ponto de vista da exploração da subdominante e da cadência plagal (ver exemplo 3.1, p. 37).



3.3.2 NI-Akoustik Piano

A Study in Keith pode ser observado como uma simulação para um trabalho posterior com o *Disklavier* da Yamaha (ver Figura 10, p. 38). Este último caso não é citado por Sorensen (2015) e Sorensen e Swift (2009), mas em *Disklavier Sessions* (SORENSEN, 2013), sua execução musical é semelhante à atividade descrita em *A Study in Keith*, exceto o fato de que no primeiro é utilizado um piano acústico e, no segundo, um piano virtual.

Este piano virtual, NI, é uma abreviação para *Native Instruments*, uma empresa de tecnologias para áudio [36]. O *Akoustic Piano* é uma extensão (*plugin*) VST, que emula diferentes pianos acústicos. Os instrumentos são gravados nota a nota por um complexo

[36] Disponível em <http://www.native-instruments.com/en/company/>.

sistema de tomada de som, amostrados digitalmente, para então ser possível utilizar os registros como eventos MIDI[37].

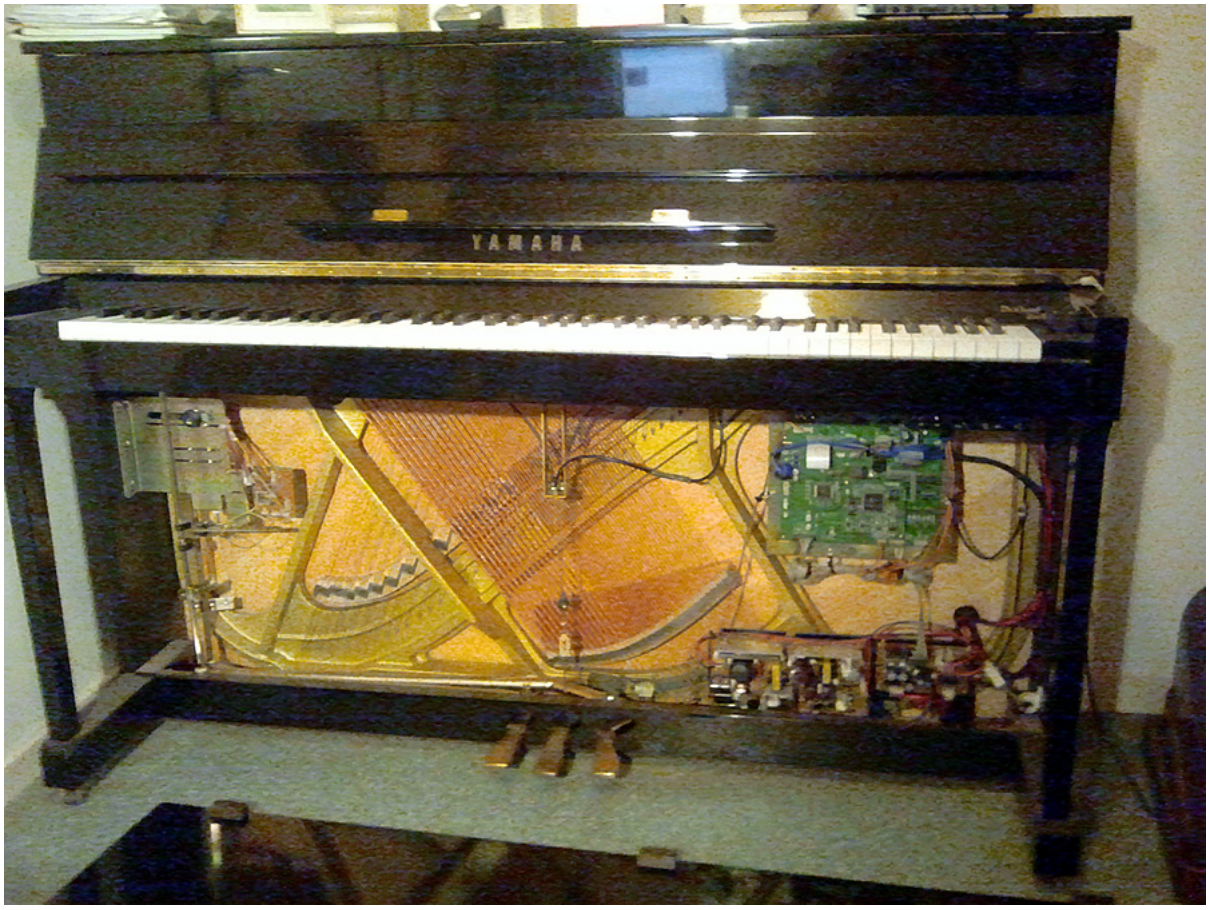


Figura 10 – Piano Disklavier de armário, com a parte interna exposta para exibir a placa-mãe. **Fonte**: wikimedia.org

> "Em *Disklavier Sessions* os programas escritos em tempo-real por Ben e Andrew geram um fluxo de dados de notas que é enviado para ser executado em um piano disklavier mecanizado. Assim como as alturas das notas, toda a performance do piano deve ser codificada na informação gerada pelo programa e enviada para o piano disklavier."[38]

3.3.3 Ambiente e Linguagem: Impromptu

> " Impromptu é uma linguagem e um ambiente de programação OSX[39] para compositores, artistas sonoros, VJ's e artistas gráficos com um

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

[38] 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*"

[39] Sistema Operacional Mac OSX.

> interesse em programação ao vivo ou interativa. Impromptu é um ambiente de linguagem Scheme, um membro da família das linguages Lisp. Impromptu é usado por artistas-programadores em performances de *livecoding* em torno do mundo. "[40]

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

> "Considere a analogia da partitura musical tradicional. A partitura provê uma especificação estática da intenção – um programa de domínio estático. Musicistas, representam o domínio do processo, executam ações requeridas para realizar ou reificar a partitura. Finalmente, as ações no domínio do processo resultam em ondas sonoroas que são percebidas por uma audiência humana como música. Este estágio final é o nosso domínio real de trabalho. Agora considere um domínio de programação dinâmica no qual o compositor concebe e descreve uma partitura em *tempo-real*. Nós geralmente chamamos este tipo de composição de improvisação. **Na improvisação o(a) musicista é envolvido em um circuito-fechado retroalimentado que envolve premeditação, movendo para ação casual e finalmente para reação, refinamento e reflexão.**"[41]

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

3.3.4 Extempore

O *Extempore* possui um sistema humano-máquina reflexivo (ver seção 2.1, p. 13). Um nome específico, para a atividade de programar, escutar o resultado, e recodificar, é simbolicamente chamado de *programação ciberfísica*:

---

[40] 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/>"

[41] 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 toreaction, refinement and reflection.*"

> "*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."[42]

Entre suas características de interesse musical, incluem[44]:

- Processamento de Sinais Digitais (DSP) [45] em tempo-real;

- Sequenciamento de áudio de alto-nível, baseado em notas, como o disparo de sons baseado em parâmetros como altura, intensidade e duração. [46].;

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

3.3.5 Scheme

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

> "Um sistema chamado LISP (para Processador de LISta) foi desenvolvido para um computador IBM 704 pelo grupo de Inteligência Artificial no M.I.T. O sistema foi projetado para facilitar experimentos com um sistema proposto chamado "Recebedor de conselhos" [Advice Taker], onde uma máquina pode ser instruída para lidar com sentenças declarativas, bem como imperativas, e poderia exibir um "senso comum" no desempenho de suas instruções. A proposta original para o *Advice Taker* foi feita em novembro de 1958. O principal requerimento foi um sistema de programação para manipular expressões que representam sentenças formais, declarativas e imperativas, de modo que o sistema *Advice Taker* pode fazer deduções. No curso do desenvolvimento, o sistema LISP passou por diversas simplificações e, eventualmente, se baseou em um esquema para representar funções recursivas parciais de certas classes de expressões simbólicas. Esta representação é independente do computador IBM 704, ou qualquer outro computador eletrônico, e agora parece útil expor o sistema, começando com a classe de expressões chamadas expressões-S e as chamadas funções-S (McCarthy, 1960, seção 1)."[47]

[42] Tradução nossa de "*Extempore is designed to support a style of programming dubbed 'cyberphysical' programming. Cyberphysical programminEntre as diferenças do Lisp e Schemeg supports the notion of a human programmer operating as an active agent in a real-time distributed network of environmentally aware systems.* [43]. "

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

[45] Sobre DSP, Cf. SMITH, 2012-06.

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

[47] Tradução nossa de "*A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed*

A definição de funções-S foge do escopo de nossa pesquisa, mas ela pode ser compreendida de maneira intuitiva, a partir das expressões-S. McCarthy (1960, seção 3) define expressões- S como "átomos" e listas de átomos, onde um átomo também pode ser uma lista de átomos. Existe uma classe de expressões simbólicas definida por parênteses. Dentro desta expressão simbólica são inseridos átomos (ver exemplo 3.2, 3.3 e 3.4).

Exemplo 3.2 (Expressão simbólica vazia)

```
( )
```

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

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 eles mesmos outras expressões:

Exemplo 3.4 (Expressão simbólica com átomos)

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

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

to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions. The original proposal [1] for the Advice Taker was made in November 1958. The main requirement was a programming system for manipulating expressions representing formalized declarative and imperative sentences so that the Advice Taker system could make deductions.In the course of its development the LISP system went through several stages of simplification and eventually came to be based on a scheme for representing the partial recursive functions of a certain class of symbolic expressions. This representation is independent of the IBM 704 computer, or of any other electronic computer, and it now seems expedient to expound the system by starting with the class of expressions called S-expressions and the functions called S-functions."

```
( A ( A B ) A ( B A B ) )
( A ( A B ) A (( A B ) A ( A B )))

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

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

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

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

Exemplo 3.5 (Notação prefixada)

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

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

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

Exemplo 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 exemplificar musicalmente. Sorensen e Gardner (2010, p. 823-824) apresenta um código musical fictício, mais especificamente, direcionado como uma experiência mental de uma sonoridade de *jazz* tonal.

Exemplo 3.7 (Exemplo musical para o Scheme)
Este exemplo é semelhante com o primeiro algoritmo gerador de sonoridades tonais em *A Study in Keith* (ver seção 3.4, p. 47).

> “ 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 expressao 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 um 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 será discutido, na próxima seção, como a estratégia transversal, $\mathcal{T}_{ask}$. Propomos um particionamento do código para melhor compreensão:

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)

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

Adiante são definidas propriedades com termos do vocabulário da música tonal, ou, coloquialmente, batida (no sentido da posição de uma unidade de tempo em um pulso,

tactus), acorde (tríades, 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 atavés 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)))
```

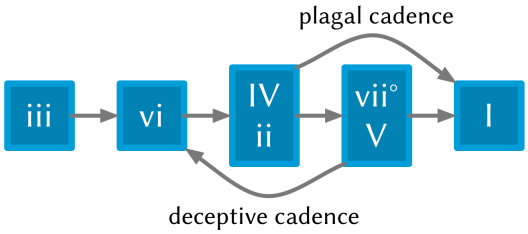
A escolha de acordes é feita de maneira randômica, segundo uma lista de cadências predeterminados. Neste ponto, podemos indicar de maneira mais explícita uma regra de qualidade:

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. Por exemplo, podemos restringir uma sequência de cadências-modelo com estruturas que se afastam da tônica, e as que voltam para a tônica. A maneira por qual podem ser acessadas, pode variar, e é interessante que o computador escolha randomicamente as progressões desejadas pelo improvisador programador (ver Figura 11, p. 47).

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



	I	ii	iii	IV	V	vi	vii°
I	0.0	0.0	0.25	0.25	0.25	0.25	0.0
ii	0.0	0.0	0.0	0.0	0.5	0.0	0.5
iii	0.0	0.0	0.0	0.0	0.0	1.0	0.0
IV	0.25	0.25	0.0	0.0	0.25	0.0	0.25
V	0.5	0.0	0.0	0.0	0.0	0.5	0.0
vi	0.0	0.5	0.0	0.5	0.0	0.0	0.0
vii°	0.5	0.0	0.0	0.0	0.5	0.0	0.0

Figura 11 – Distribuição, aproximada, de probabilidades de acontecimento com um conjunto de possíveis cadências tonais organizados como uma cadeia de Markov. **Fonte**: Swift (2012).

Discussão

3.4 Blocos de Eventos

No capítulo anterior, definimos o espaço conceitual, nos termos dos Quadros Conceituais de Sistemas Criativos, como uma aplicação do espaço conceitual da improvisação de códigos e, com um estilo de *jazz*. Apresentamos algumas regras de validação em momentos anteriores, da improvisação de códigos (ver subseção 2.4.3, p. 25), e de um referencial opcional (ver seção 3.3, p. 34).

Seguiremos com configuração da estratégia transversal de Sorensen, como regra de detecção $\mathcal{T}_{ask}$, que possui uma regra de qualidade $\mathcal{E}_{ask}$ (ver subseção 3.4.1, p. 48). Este espaço conceitual gera uma sequência de blocos de eventos $\mathcal{K}'^{0}_{ask}$, muito semelhantes ao neumas musicais básicos[51], em um contraponto de primeira espécie que sofre uma primeira transformação (ver subseção 3.4.1.2, p. 50).

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,

[51] Cf. GASPERINI, 1905, p. 136

ao tempo de ação musical, é similar em outros dois vídeos, de Sorensen: An evening of livecoding at 53 Rusden Street[52], Just for Fun[53], A Study in Part[54], Stained[55], Transmissions in Sound[56], Antiphony[57], Strange Places[58], Orchestral[59], UMDT[60], Day of Triffords[61], Face to Face[62], BM&E[63], A Christimas Carol[64] Dancing Phalanges[65], Livecoding Audio DSP[66], Jazz Ensenble Study[67], Variations on a Christmas Theme[68]. Esta característica também foi observada em uma outra performance (ver **??**, p. **??**). Isso não quer dizer que o silêncio é um ator musical, com alguma distância em relação a uma proposta apresentada anteriormente (ver seção 1.2, p. 7).

3.4.1 Definição do instrumento e do tempo

Seu início é um pequeno comentário que contem o nome do executante e seu email para contato (primeiros sete segundos), bem como a escrita de um código que inicializa o NI-Akoustik (até 0′43″, ver Propriedades de novos acordes 3.17).

Exemplo 3.17 (Definição de instrumento)
Primeiros eventos musicais gerados a partir das primeiras estruturas válidas de código.
Fonte: (SORENSEN, 2015).

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

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

52 Disponível em <https://vimeo.com/2433303>
53 Disponível em <https://vimeo.com/2433971>
54 Disponível em <https://vimeo.com/2434054>
55 Disponível em <https://vimeo.com/2502546>
56 Disponível em <TransmissionsinSound>
57 Disponível em <https://vimeo.com/2503188>
58 Disponível em <https://vimeo.com/2503257>
59 Disponível em <https://vimeo.com/2579694>
60 Disponível em <https://vimeo.com/2579880>
61 Disponível em <https://vimeo.com/2735394>
62 Disponível em <https://vimeo.com/5690854>
63 Disponível em <https://vimeo.com/7339135>
64 Disponível em <https://vimeo.com/8364077>
65 Disponível em <https://vimeo.com/8732631>
66 Disponível em <https://vimeo.com/15585520>
67 Disponível em <https://vimeo.com/15679078>
68 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)
Definição do tempo base. **Fonte**: (SORENSEN, 2015).

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

3.4.1.1 Definição de uma sequência de blocos

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

Exemplo 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*[69]:

> "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)."[70]

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

[70] 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*

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 diatônca: baseado em uma raiz (0 para C, etc.), maior/menor (`'-` ou `'^`) e graus (i-vii)"[71]. 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.

3.4.1.2 Definição de blocos

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

Exemplo 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 transveral)

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

Seguido de um "impulso", ou um estímulo sonoro:

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*"

[71] 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 inical)

```
(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 díade.

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 díade 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 ($dur \Rightarrow (*metro *' durdur)$)[72].

3.4.2 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 $\mathcal{O}'^{0}_{ask}$), duas díades, 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))
```

[72] 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 (ver Estratégia transversal 3.18, p. 49), transcrevemos este trecho com um andamento entre 35–40 BPM (ver Figura 12, p. 52). É interessante notar que tais figuras simbolizam neumas, no caso, um *bipunctum*, ou duas notas repetidas, na mão direita, e na mão esquerda um *clivis*, ou um "acento agudo com um grave"[73]. No caso específico desta primeira figura, na mão direita, um *bipunctum* , e na mão esquerda, uma *clivis*.

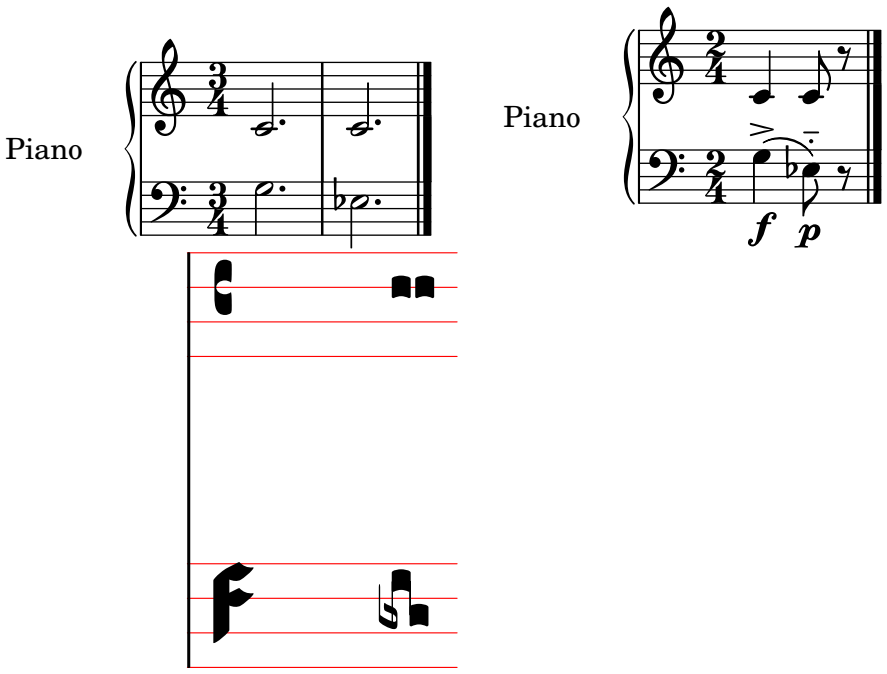


Figura 12 – Transcrição literal e perceptiva do primeiro evento em *A Study in Keith*. **Fonte**: autor.

[73] Tradução nossa de "*Cf. GASPERINI, 1905*, idem. *Unione dell'accento acuto col grave*"

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))))
```

3.4.2.1 Regras de qualidade

A estrutura interna da estratégia `chords` explicita algumas regras de qualidade, bem como permite apresentar uma primieira sequência de blocos de eventos $\mathcal{K}'^{0}_{ask}$, um conjunto de características $\mathcal{F}'^{0}_{ask}$ e um pequeno grupo de objetos $\mathcal{O}'^{0}_{ask}$. Um conjunto de características é definido pelo momento de execução do evento,$\mathcal{F}'^{0}_{ask}$, o grau, $\mathcal{F}'^{1}_{ask}$, e a duração deste evento, $\mathcal{F}'^{2}_{ask}$. 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)
```

3.4.2.2 Primeira sequência de blocos de eventos

A Figura 14 indica uma primeira sequência de nêumas, gerados pelo algoritmo acima, em um padrão que é repetido por dois ciclos (blocos de eventos $\mathcal{E}'^{0}_{ask}$ e $\mathcal{E}'^{1}_{ask}$). Durante este tempo, Sorensen realiza uma mudança (1^o ciclo de bricolagem). Esta mudança transita entre o segundo bloco $\mathcal{E}'^{1}_{ask}$ e terceiro bloco $\mathcal{E}'^{2}_{ask}$, e sua exeucção 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.

Primeiro Bloco

O primeiro bloco de eventos $\mathcal{E}'^{0}_{ask}$ aprensenta um contraponto de primeira espécie, aticulado em tempos fortes e fracos, de acordo com um movimento cadencial $i \Rightarrow vii$ (ver Figura 14, p. 56). Uma característica $\mathcal{F}'^{0}_{ask}$ do algoritmo é sua direcionalidade em um âmbito de quinta em um número de compassos pares (4 nesse caso).

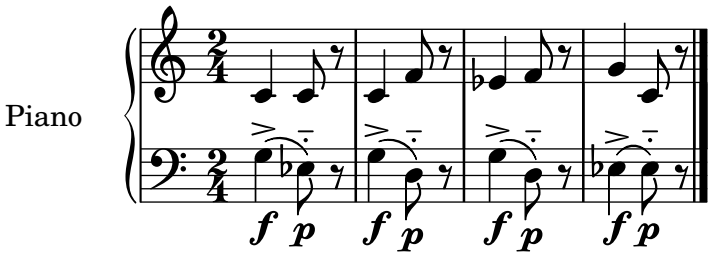
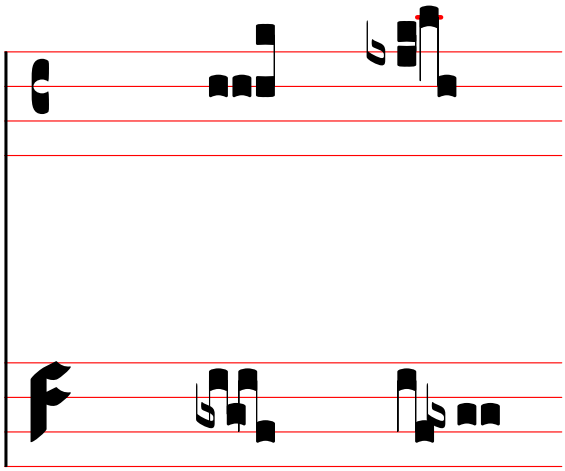


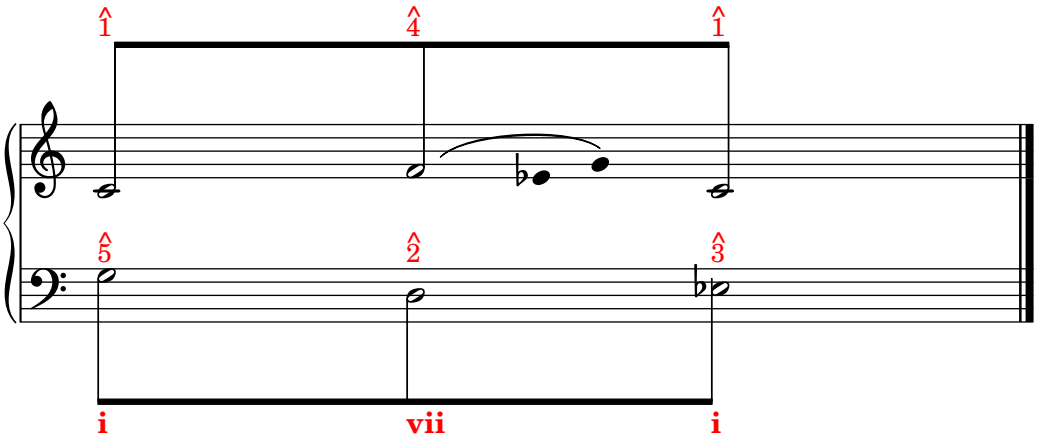
Figura 13 – Primeiros eventos musicais gerados a partir das primeiras estruturas válidas de código. **Fonte**: autor.

A aparente repetição de um mesma classe de eventos sonoros, este mesmo um objeto $\mathcal{O}'^{0}_{ask}$, pode ser diferenciada através de figuras neumáticas na mão direita e na mão esquerda (ver exemplo 3.29):

Exemplo 3.29 (Transcrição de neumas do primeiro bloco)
Notação neumática para a um *bipunctum*, dois *podatus* e uma *clivis* na mão direita. E na mão esquerda, três *clivis* e um bipunctum.



Notação schenkeriana que expõe um movimento plagal estrutural na mão direita (Î-4̂-Î), com um *porrectus* (um acento agudo, um grave e um agudo) estrutural na mão esquerda (5̂-2̂-3̂).



Segundo Bloco

Que pode ser reescrito como neumas na mão direita:

Exemplo 3.30 (Transcrição de neumas do segundo bloco)
Notação neumática para cinco *podatus* e uma *clivis* na mão direita. E na mão esquerda uma *clivis*, um *podatus*, um *bipunctus*, três *podatus*.

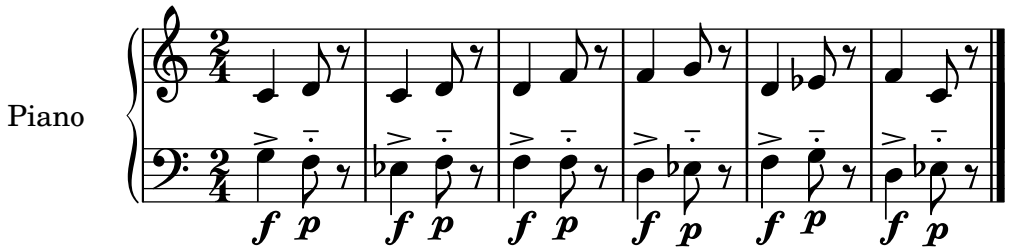
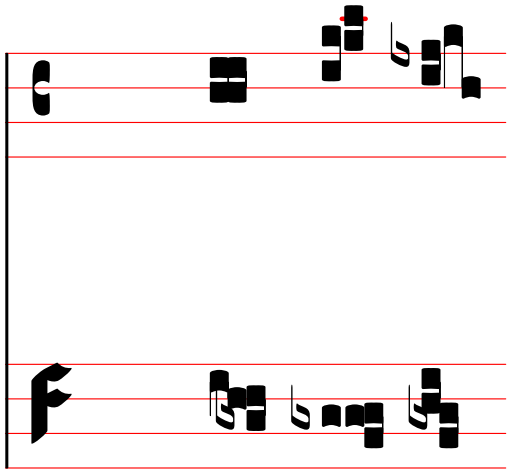
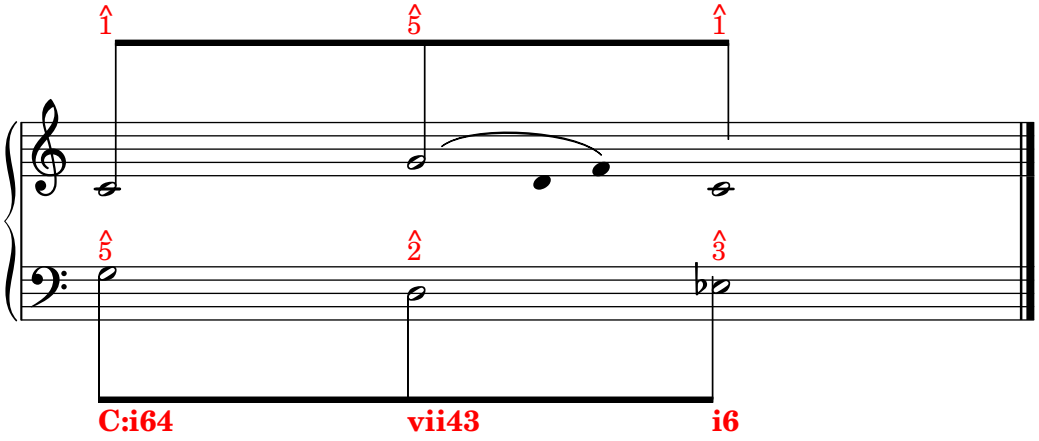


Figura 14 – Segundo bloco de eventos musicais. **Fonte**: autor.



Notação schenkeriana que expõe um movimento autêntica estrutural na mão direita (Î-$\hat{5}$-Î), com uma repetição do *porrectus* anterior ($\hat{5}$-$\hat{2}$-$\hat{3}$).



Terceiro Bloco

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

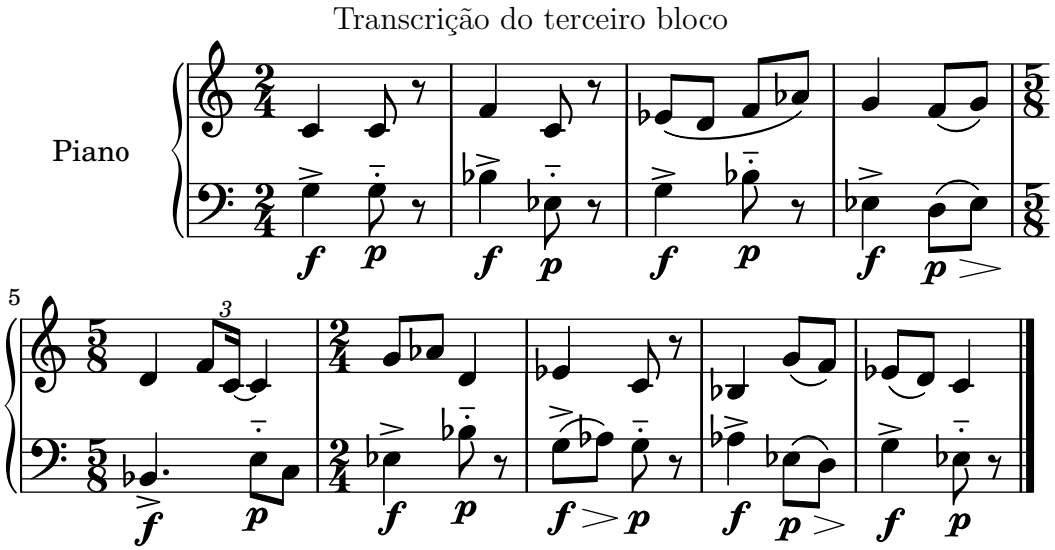


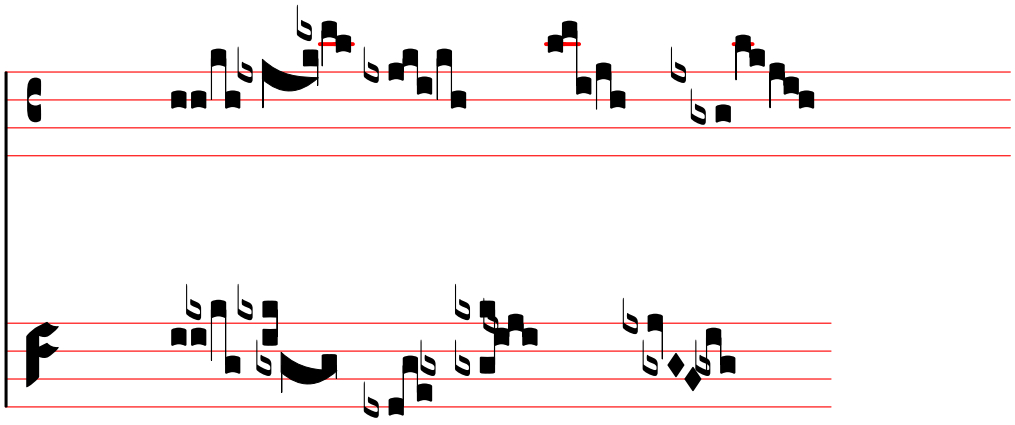
Figura 15 – Terceiro bloco de eventos musicais. **Fonte**: autor.

Exemplo 3.31 (Primeira transformação da estratégia transversal)

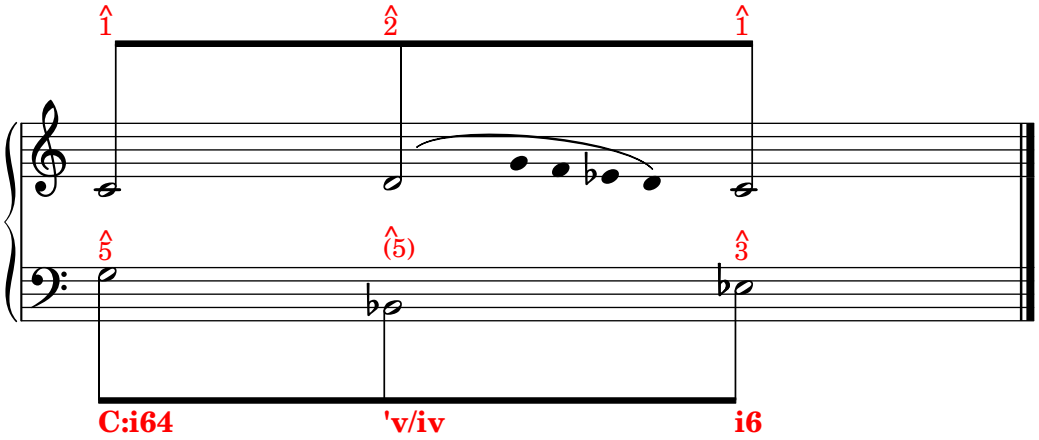
```
(define chords
   (lambda (time degree dur)
      (if (member degree '(i)) (set! dur 3.0))
      (for-each (lambda (p)
                  (let* (dur1 (* dur (random '(0.5 1))))
                        (dur2 (- dur dur1)))
                  (play-note (*metro* time) piano p
                             (+ 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 15, p. 57).

Exemplo 3.32 (Transcrição de neumas do terceiro bloco)
Notação neumática para: *bipunctus*, um *clivis*, um *porrectus*, um *clivis*, um *torculus*, um *clivis*, um *torculus*, dois *clivis* e um *clivis subpunctum* na mão direita. E na mão esquerda um *bipunctus*, um *clivis*, um *podatus*, um *porrectus*, um *torculus*, um *podatus*, um *torculus*, um *climatus*, e um *clivis*.



A notação schenkeriana sugere um *torculus* estrutural na mão direita ($\hat{1}$-$\hat{2}$-$\hat{1}$), onde existe um tetracorde decendente ($\hat{5}$-$\hat{4}$-$\hat{3}$-$\hat{2}$). Na mão esquerda, destacamos o ponto mais grave da tessitura aponta para um modalismo ($\hat{5}$-$\hat{7}b$-$\hat{3}$), bem como alterações nas estruturas rítmicas do trecho citado.



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

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 mudaça 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 diminuido (`(* dur (random '(0.5 1)))`), enquanto o ritmo da mão esquerda é uma diferença entre uma duração geral, e o ritmo da mão direita. No caso desta nova duração da mão esquerda, é aplicado uma verificação condicional:

Exemplo 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 definida[74] como "seleção de uma altura, de uma classe de alturas relativa à uma dada altura"[75]. 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).

[74] Disponível em <https://github.com/digego/extempore/blob/master/libs/core/pc_ivl.xtm>
[75] Tradução nossa de "*select pitch from pitch class relative to a given pitch*"

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.

Exemplo 3.37 (Laço iterativo modificado)

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

No entanto esta característica é um dos fios condutores de uma seção $\mathcal{K}'^{1}_{ask}$, o que excede um objetivo deste documento. Nosso interesse nesta análise foi investigar, através do estudo de contextos, e de diferente 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.

3.5 Discussão

Este capítulo buscou analisar uma zona de conceitos que permeiam a improvisação *A Study in Keith*, publicada em 2009 por Andrew Sorensen. Levantamos, através do quadro conceitual de sistemas criativos, um conjunto de informações sobre um contexto que estimulou o improvisador-programador para sua realização. A partir de uma outra improivisação, definida como *referente opcional*, ou os *Concertos Sun Bear*, Sorensen buscou simular um estilo de *jazz* do pianista e compositor Keith Jarret. No entanto, esta relação é metafórica, sendo que destacamos um discurso musical eclesiástico, ou baseado na no quarto grau de uma tonalidade. Na nossa transcrição de uma cadência da abertura do disco Kyoto I (que ela mesma, necessita de verificações), encontramos uma exploração da cadência plagal, através de uma substituição por trítono. Já em Sorensen o procedimento é bastante simplificado, quase pedagógico, de como programar figuras musicais – interpretadas como neumas –, dentro de uma simples cadência autêntica imperfeita (i – vii). A partir desta regra, foi possível apontar figuras musicias separadas em três blocos, que formam uma primeira sequência do improviso, separada por uma primeira interrupção de uma próxima sequência, ainda não analisada. Desta forma, foi possível delinear um objetivo da improvisação, que é transformar uma classe de objetos sonoros (cuja caracterísitca é ser um contraponto de primeira espécie, com articulação forte-fraco), dentro de um contorno melódico, em um contraponto de segunda espécie, cuja acentuação

é alterada. Por outro lado, nossa análise não contemplou sequências seguintes, o que impediu observar detalhes sobre o processo geral da improvisação. Por fim, descrevemos esta análise como uma experiência preliminar em análise de códigos e, por isso mesmo, o Quadro Conceitual de Sistemas Criativos de Alex McLean, e o Modelo de Improvisação de Jeff Pressing, se apresentam como uma interessante ferramente metodológica.

Conclusão

Em um espaço conceitual mais amplo, o objeto de investigação deste documento contextualiza o que é uma improvisação de códigos, e como podemos entender pluralidades quando se codificam artefatos sonoros, visuais, corporais e têxteis. Ward et al. (2004) definem a improvisação de códigos como "atividade da escrita integral (ou partes) de um programa enquanto ele é executado"[76]. Blackwell e Collins (2005) enfatizam a definição do ponto de vista da linguagem de programação como instrumento musical. McLean (2006-07-30) relata o *live coding* como ferramenta para um *Disk Jockey codificado*. Sorensen e Swift (2009) pontuam a improvisação de códigos como "uma prática de performance para o qual linguagens de computador definem o meio primário de expressão artística."[77]. Para Sorensen e Gardner (2010), *live coding* (ou *livecoding*) envolve a premissa de uma programação-partitura audiovisual reativa:

> Livecoding é uma prática de arte computacional que envolve criação em tempo-real de programas de audiovisual generativo para performances 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)[78].

Magnusson (2011), Collins (2014) sintetizam o *live coding* como improvisação audiovisual. Sorensen (2014) define como "programar sistemas de tempo-real em tempo real"[79]. Em um discussão entitulada "*Wtf is livecoding*"[80] diz que o "*Live coding* celebra a efemeridade da própria definição"[81] (ver Figura 16, p. 64). Embora semelhantes, as definições mudam de detalhes de acordo com o contexto. Por exemplo, Ward et al. (2004) enfatizam que o código pode ser (re)composto de partes menores. McLean (2006-07-30) pontua o ambiente de codificação onde o código é (re)programado, e que tipo de música é feita. Sorensen (2014) destaca que modificar alguma coisa (bricolagem) é próprio da técnica, de forma que é possível extender essa bricolagem para ideologias, enfraquecendo a própria definição de *live coding*. Nick Collins (2014) situa essa questão da seguinte formafundamentos objeto:

---

76 Tradução nossa de "*Live coding is the activity of writing (parts of) a program while it runs*"

77 Tradução nossa de "*Live coding is a performance pratice for which computer languages define the primary means of expression.*"

78 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*

79 Tradução nossa de "*programming real-time systems in real-time*"

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

81 Tradução nossa de "*Live coding celebrates the ephemerality of definition itself*"

Figura 16 – Definição de *live coding*: “Insira a definição aqui”. **Fonte**: Collins (2014).

Neste ponto encontramos um desafio à metodologia de pesquisa acadêmica: se a definição do termo que contextualiza um estudo de caso é mutante, como analisar este caso? Desta forma, investigamos uma improvisação de códigos como uma simulação de uma improvisação instrumental. Esta improvisação instrumental, para facilitação do que será analisado pode ser harmonicamente simples. Desta forma foi possível entender, em termos teóricos da harmonia tradicional, a razão pela qual um improvisador destaca a diferença entre o que foi testado e aquilo que o inspirou a fazer.

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 testes”[82] musicais. Ao aprofundarmo-nos em uma escavação netnográfica (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 conceitos satélites de uma proposição exemplar, e como o algoritmo improvisado se relaciona com conceitos de harmonia tradicional aplicados ao de desenvolvimento de *softwares*.

[82] 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 63.

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 ix, 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 ix, 63 e 64.

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.

FENERICH, A.; OBICI, G.; SCHIAVONI, F. Marulho TransOceânico: performance musical entre dois continentes. p. 12, 2014. Disponível em: <https://www.academia.edu/t/M8Fvh/8178331>. Citado na página 39.

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.

GASPERINI, G. *Storia della semiografia musicale*. 1st. ed. [S.l.]: Manualli Ulrico Hoepli – Libraio dela Casa Real, 1905. Citado 2 vezes nas páginas 47 e 52.

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 ix, xi, 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 ix, 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 2 vezes nas páginas ix e 36.

MAGNUSSON, T. Algorithms as scores: Coding live music. v. 21, p. 19–23, 2011. Citado 2 vezes nas páginas 39 e 63.

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 xii 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 xii 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 2 vezes nas páginas 40 e 41.

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 63.

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 ix, xi, 1, 5, 7, 24 e 31.

McLean, A.; WIGGINS, G. *Patterns of movment 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 etnographical approach. In: *ICLC2015 Proceedings*. [S.l.: s.n.], 2015. p. 117–124. ISBN 978 0 85316 340 4. Citado 2 vezes nas páginas xi e 64.

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 difusione 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 ix, 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: <ttp://www.jstor.org/stable/3679484>. Citado na página 22.

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/~aschloss/publications/JNMR02_Dilemma_of_the_Performer.pdf>. Citado 4 vezes nas páginas xiii, 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 xii.

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 40.

SOARES, G. R. *Luteria Composicional de algoritmos pós-tonais v1.1FINAL*. Prévia da dissertação para a banca de quali- ficação 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 63.

SORENSEN, A. *The Disklavier Sessions*. Youtube, 2015. Disponível em: <https://www.youtube.com/watch?v=cFEadvBeBqw>. Citado 6 vezes nas páginas xiii, 34, 37, 48, 49 e 50.

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 39, 43, 46 e 63.

SORENSEN, A.; SWIFT, B. *A Study in Keith*. Vimeo, 2009. Disponível em: <https://vimeo.com/2433947>. Citado 6 vezes nas páginas v, xiii, 34, 36, 37 e 63.

SWENSON, J. *The Rolling stone jazz record guide*. Rolling Stone Press, 1985. 219 p. ISBN 039472643–X. Disponível em: <https://openlibrary.org/books/OL2867249M/The_Rolling_stone_jazz_record_guide>. Nenhuma citação no texto.

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 ix, 47 e 49.

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 10 vezes nas páginas xii, xiii, 22, 23, 24, 25, 26, 28, 29 e 63.

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.