UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Instituto de Informática
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MEPSOM
Método de Ensino de Programação Sônica para Músicos

por

ELOI FERNANDO FRITSCH

Tese de Doutorado submetida à avaliação como requisito
parcial na obtenção do grau de Doutor em Ciência da Computação

Orientadores

Dra. Rosa Maria Viccari
Instituto de Informática

Dr. Antônio Carlos Borges Cunha
Instituto de Artes

Porto Alegre, Maio de 2002.

AGRADECIMENTOS

Gostaria de agradecer muito aos meus orientadores Dr ª Rosa Maria Vicari e Dr. Antônio Carlos Borges Cunha. Ao compositor/pesquisador Dr. Eduardo Reck Miranda. Aos meus colegas do Laboratório de Computação e Música da UFRGS Luciano Vargas Flores, Evandro Manara Milleto Susana Ester Krüger e Rodolfo Wulfhorst. Aos meus bolsistas Rafael Vanoni Polanczyk, Jose Maurício Schaefer Poyastro, Tales Eduardo Riedel de Lima, Eduardo Flores, Roges Grandhi e Tiago Rubin. À minha mãe Odete Biazus Fritsch, meu pai Eloy Fritsch e meu irmão Rui Afonso Fritsch, à minha esposa Lauren Veronese e à minha filha Deborah Fritsch. À banda APOCALYPSE. Aos meus colegas professores do Instituto de Informática da UFRGS, aos meus colegas músicos e compositores do Instituto de Artes da UFRGS, aos meus colegas professores da Universidade Luterana do Brasil e aos meus colegas da Universidade de Caxias do Sul, em especial Alexandre Moretto Ribeiro. Aos meus bolsistas no Laboratório de Música Eletroacústica da UFRGS Rafael de Oliveira e Priscila Medina. À todos os meus alunos do curso de Bacharelado em Composição da UFRGS que participaram das avaliações do MEPSOM. À todos os meus alunos do Programa de Extensão em Música Eletrônica que participaram da avaliação do MEPSOM. Aos funcionários do Instituto de Informática da UFRGS. Ao CNPq e à FAPERGS por financiar a pesquisa e fornecer recursos para a implantação dos Laboratórios LC&M e Música Eletroacústica na UFRGS.

"Dedico esta TESE de DOUTORADO
em memória de meu pai ELOY FRITSCH"

ELOI FERNANDO FRITSCH

Sumário

Índice de Figuras

Lista de Abreviaturas

DSP – Processamento Digital de Sinais
EPCMs – Estações de Programação de Computadores para Música.
LC&M – Laboratório de Computação e Música da UFRGS.
LME – Laboratório de Música Eletroacústica da UFRGS.
MEPSOM – Método de Ensino de Programação Sônica para a Música.
MIDI – Musical Instrument Digital Interface.
MIDILAB – Laboratório MIDI da UFRGS.
SETMUS – Sistema Especialista para Teoria Musical
SeVEM - Separador de Vozes em Execuções Musicais Polifônicas
STI – Sistema de Treinamento de Intervalos
STR – Sistema de Treinamento Ritmico
UFRGS – Universidade Federal do Rio Grande do Sul
UI – User Interface

Lista de Tabelas

Resumo

Esta Tese de Doutorado apresenta o MEPSOM - Método de Ensino de Programação Sônica de Computadores para Músicos. O MEPSOM consiste em um sistema de computação que disponibiliza um conjunto de atividades para programação de software musical composto de exemplos e exercícios. O método foi idealizado para ser uma ferramenta de auxílio ao professor em cursos de Computação Musical, disponibilizando recursos didáticos para o ensino de programação nas áreas de composição e educação musical.

O MEPSOM foi implementado sob a forma de programas de computador e utilizado em cursos de Computação Musical na UFRGS. Nesta Tese de Doutorado apresentamos o projeto e a organização do MESPCM, a implementação do método, relatos de sua aplicação e os resultados obtidos. Também expomos a utilização do método em laboratório, através de estudo de caso, e os resultados da sua avaliação por estudantes que participaram de pesquisas de levantamento. Por fim, a partir da análise dos dados obtidos, sugerimos um conjunto de aspectos considerados relevantes para futuras aplicações do MEPSOM.

Palavras-chave: computação e música, informática na educação, ensino de programação sônica.

TITLE: "A Method for Teaching Audio Software Programming to Musicians"

Abstract

The current thesis presents MEPSOM – Método de Ensino de Programação Sônica para Músicos (A Method for Teaching Audio Software Programming to Musicians). MEPSOM consists of a computing system that puts available a set of activities for musical software programming composed of examples and exercises. The method was idealized as a tool to aid teachers in computer music courses, by supplying didactic resources for programming teaching in areas such as composition and music education.

MEPSOM was carried out through the implementation of computer programs. It was applied in computer music courses at UFRGS (Federal University of Rio Grande do Sul). In this work we present MEPSOM design and organization, as well as the method implementation, reports on method application, and results obtained. We report the method application in laboratory by means of case-studies, and display results of the survey we carried out with students. Eventually, following the analysis of data obtained, we suggest some aspects considered significant for future applications of MEPSOM.

Keywords: computer music, computer and education, audio software programming.

1 Introdução

Desde 1993 o LC&M pesquisa e desenvolve software musical de cunho educacional. Os projetos desenvolvidos na área de Computação Musical tornam viável a criação de aplicações na área de ensino de música direcionadas aos jovens músicos. Os sistemas de computação auxiliam não só no desenvolvimento da literatura e da prática musical, mas também na criação de software musical.

A pesquisa em software musical realizada no LC&M – Laboratório de Computação e Música é resultado de um trabalho em conjunto entre o Instituto de Informática e o Instituto de Artes da UFRGS.

Para auxiliar o músico a aprender programação de computadores, com objetivo de elaborar software musical, foi criado o MEPSOM – Método de Ensino de Programação Sônica para Músicos. A palavra "Método" é definida como sendo o caminho para chegar a um fim [FER 86]. Logo, para este trabalho, o termo "método de ensino" corresponde ao caminho que o aluno necessita seguir para aprender determinado conteúdo. O expressão " Programação Sônica" significa a criação de software para controle, organização e geração de sons através de processamento por computador. Portanto, Método de Ensino de Programação de Computadores para Músicos pode ser definido como "o caminho que o músico deve seguir para aprender a criação de software para controle, organização e geração de sons através do computador".

O desenvolvimento do MEPSOM foi baseado em trabalhos anteriores sobre software educacional para a música desenvolvidos pelo LC&M. Dentre eles podemos citar [FRI 94] , [FRI 95], [BEC 95], STI [FRI 96] [FLO 2000a], STR [FRI 98] e SeVEM [WUL 01], MMAS [WUL 01].

O MEPSOM é formado por um conjunto de programas organizados em níveis que constituem uma fonte de exemplos e exercícios para utilização em Laboratórios de Computação Musical. Esses programas foram desenvolvidos para serem utilizados por professores de Computação Musical e estudantes, através do uso em instrumentos MIDI[1] ligados ao computador.

O objetivo principal dessa Tese de Doutorado é projetar, implementar e aplicar o MEPSOM em situações reais de ensino de programação de computadores para músicos.

1.1 Contribuições

Esta pesquisa contribui, principalmente, para as áreas de computação e música. As contribuições computacionais são nas áreas de engenharia de software, informática na educação e multimídia, destacando-se:

- o desenvolvimento de soluções gráficas e visuais para algoritmos existentes em linguagens procedurais, a criação de programas com ênfase na interface visual e sonora;

[1] MIDI – Musical Instrument Digital Interface – Protocolo de comunicação entre instrumentos musicais e computadores.

- a verificação da adequação das linguagens escolhidas para a implementação do MESPCM;

- a criação de um sistema de computação inédito para área de informática na educação, utilizando o paradigma de projeto centrado no usuário;

- a contribuição com os trabalhos já desenvolvidos na área de ensino de programação sônica, visto que muitos deles não são específicos nem adaptados para músicos;

As contribuições de ordem musical estão compreendidas nas áreas da música eletroacústica, composição e educação musical e são:

- a criação de um método de ensino inédito de programação para a área de música eletroacústica, utilizando recursos computacionais modernos para facilitar o processo de aprendizagem dos músicos no estudo de programação de aplicações musicais;

- a desmistificação do uso de programação de computadores como recurso para criação de material musical para composição;

- a apresentação de algoritmos para composição auxiliada por computador de uma forma visual com ênfase nos resultados sonoros;

- facilitação do processo de ensino de programação de aplicações na área da educação musical, e ainda como recurso para motivar educadores e compositores a utilizarem o computador na educação musical;

Esta Tese de Doutorado está dividida em 3 etapas. O primeira apresenta a motivação, os trabalhos relacionados o projeto e a organização do MEPSOM. A segunda apresenta ferramentas de programação para a música, escolha do suporte tecnológico e a metodologia empregada na criação dos programas do MEPSOM e a implementação do MEPSOM. A última etapa apresenta o ensino de programação para músicos, fatores que possibilitaram a criação, aplicação e avaliação do MEPSOM. Por fim, são apresentadas as conclusões e trabalhos futuros que envolvem o método. A Tese de Doutorado ainda disponibiliza anexos que contêm o método de ensino na íntegra.

1.2 Aspectos Interdisciplinares

Os módulos que constituem o MEPSOM estão inseridos num contexto interdisciplinar que é formado por diferentes abordagens, tais como: educacional, tecnológico-computacional e musical. A programação sônica utilizada nesta pesquisa está fortemente baseada nesses três enfoques utilizando pois, conhecimentos interdisciplinares. Na área educacional propomos ensinar o músico a realizar tarefas através do computador, com o emprego de programação. No enfoque computacional existe a preocupação com a interface sonora e visual, a formalização do pensamento em instruções e fluxogramas gráficos para a construção de programas musicais. A área tecnológica da Computação e Música está fortemente baseada em técnicas de síntese,

processamento de áudiodigital e MIDI. Nesta pesquisa, a área musical foi delimitada através do enfoque nas subáreas: composição e educação, com o auxílio do computador.

É necessário que o músico programador tenha um conhecimento prévio dos conteúdos citados. Por exemplo, os programas não tem a intenção de ensinar MIDI, mas, sim , de ensinar o músico a programar utilizando esse recurso. Também é necessário que o músico tenha experiência em um instrumento, de preferência o teclado, e tenha conhecimentos básicos das áreas de Composição e Educação Musical. A figura 1 mostra como esses conteúdos se relacionam sob a forma de pré-requisitos. Portanto, é necessário que o aluno tenha vivenciado exepriências para a aquisição de conhecimento em Computação Musical e áreas correlatas, a fim de desenvolver sua capacidade de programação através do MEPSOM.

2 Projeto e Organização

2.1 Trabalhos Relacionados

Na literatura estrangeira encontramos sugestões para a organização de cursos na área de música e tecnologia. [DEA 97] propõe um modelo de disciplina sobre música e tecnologia para cursos de graduação em música, a partir dos requerimentos do NASM (*National Association of Schools of Music*). Essa associação coloca a tecnologia como uma das seis principais competências dos estudantes de música: "*tecnologia:* por meio de estudo e experiência em laboratório, os estudantes devem ser familiarizados com as capacidades da tecnologia em relação à composição, execução, análise, ensino e pesquisa".

No Brasil, existe uma proposta anterior de disciplina de computação musical para cursos de graduação em música e em informática [FRI 99] consistindo em um conjunto de conteúdos que podem ser adaptados às especificidades de ambos os cursos. Na área musical, o enfoque reside no uso de ferramentas para edição de partituras, composição, síntese e educação musical. O diferencial da proposta anterior para a atual reside na utilização de programação sônica, flexibilizando, desse modo, o emprego do computador na música. Logo, ao invés da utilização de programas convecionais para a música, o compositor desenvolverá seus próprios programas de acordo com suas necessidades.

A complexidade provinda de linguagens como "C" e Pascal pode tornar-se um empecilho ao músico que deseja aprender programação para música, o qual visa a construir suas próprias ferramentas computacionais para a música de maneira eficiente e rápida. Linguagens de Programação como "C" e Pascal requerem tempo para serem compreendidas. Em geral, o tempo de que o músico dispõe para estudar uma linguagem de programação é insuficiente, o que pode levá-lo a desistir dessa tarefa.

O MEPSOM foi criado para que o músico aprenda a programar utilizando um planejamento de recursos adequados e uma orientação técnica apropriada.

O MEPSOM foi idealizado tendo como referência as obras, que possuem, entre seus objetivos, o ensino de programação de computadores para músicos. Dentre elas podemos citar [WIP 89], [WIN 98], [MIR 98], [MIR 01], [DOD 97], [MES 98] e [ROA 96]. Os tutoriais dos manuais das linguagens de programação consistem em outra fonte de referência para este trabalho, principalmente na fase de implementação. Foram utilizados tutoriais de linguagens visuais[2] para aplicações musicais como os de [ZIC 88], [DOB 98], [RED 88] e [GOO 90]. As obras de [MES 98] e [CON 88] são dirigidas a programadores de computador e destinam-se ao desenvolvimento de atividades mais técnicas como programação de protocolo de comunicação MIDI e rotinas de multimídia. Esse tipo de obra não possui uma parte introdutória de programação e, portanto, torna-se inadequada ao músico que está iniciando a programação de computadores. Por essa razão não pode ser comparada ao MEPSOM, pois prevê um conhecimento aprofundado em computação já nos primeiros exemplos. Certas obras são complexas e dedicadas à programação de um assunto específico na área musical, como, por exemplo, sobre notação musical [SEL 97].

[2] Linguagens Visuais possibilitam a programação de computadores através da combinação de elementos gráficos que constituirão o programa.

Algumas obras, incluindo manuais das linguagens, são restritas a aplicações que utilizam apenas uma linguagem de programação e não apresentam muitas aplicações na área de educação musical, como é o caso de [WIN 98], [ZIC 88], [DOB 98] e [RED 88]. Outras obras são mais abrangentes e exemplificam várias linguagens de programação para a música, como as de [MIR 98], [MIR 01], [DOD 97] e [ROA 96]. No entanto, apesar de essas obras apresentarem também conteúdos sobre o ensino de programação para a música, este não é seu objetivo principal. Elas visam a apresentar as ferramentas de computação musical existentes para áreas como síntese e composição. Os autores apresentam o assunto de forma didática, porém a complexidade própria do conteúdo exige o uso de fórmulas, de deduções matemáticas, bem como de conceitos a que os músicos não estão acostumados por não possuírem embasamento na área das Ciências Exatas.

O MEPSOM possui diferenças em relação às obras citadas. Em [WIP 89] o grande parte do conteúdo apresentado objetiva a programação de aplicações composicionais, utiizando linguagem procedural[3], enquanto o MEPSOM utiliza programação visual. Em [ZIC 88] e [WIN 98], a programação visual restringe-se a aplicações com a tecnologia MIDI, não apresentando soluções para programação de síntese sonora. Por outro lado [DOB 98] apresenta um método de ensino de audiodigital através do MSP, supondo que o músico já saiba programar Max[4]. O MEPSOM, por outro lado, abrange tanto a tecnologia de programação de comunicação entre instrumentos eletrônicos quanto a programação de audiodigital.

A obra de [RED 88] apresenta um tutorial de utilização do HyperMIDI, juntamente com o guia de referência. O autor não aborda o aspecto de audiodigital. As obras [MIR 98], [MIR 01], [DOD 97] e [ROA 96], apesar de apresentarem várias linguagens de forma didática, não são métodos de programação para a música. Ao contrário do MEPSOM, esses mesmos autores não apresentam exemplos e exercícios para a elaboração de programas de educação musical. Além disso, em [DOD 97] e [MIR 01] são apresentados algoritmos e trechos de programas, porém não existe tanta ênfase na programação visual quanto no MEPSOM.

De acordo com o tabela 1, podemos dispor de um panorama geral sobre os conteúdos abordados em obras que ensinam a programação de computadores para músicos. A tabela 1 apresenta os conteúdos do MEPSOM e demonstra a abrangência do método no tema:

" Programação Sônica de Computadores para Músicos" , em relação às outras obras existentes.

[3] Linguagem escrita através de instruções de forma estruturada e algorítmica, sem a utilização de programação visual.

[4] Linguagem de programação visual para música utilizada para a implementação do MEPSOM.

Tabela 1 – Comparação dos conteúdos de Programação abordados no MEPSOM com as demais obras consultadas

Obras que tratam sobre programação para Música	Elementos Básicos de Programação	Programação de Comunicação entre Computadores e Instrumentos	Programação com Audiodigital	Programação para Aplicações de Educação Musical	Programação para Aplicações de Composição
Tutorial Max Development Package [ZIC 88]	X	X			X
C Programming for MIDI. [CON 88]	X	X			
Tutorial HyperMIDI 1.1 [RED 88]		X			X
Automated Music Composition [WIP 89]	X				X
Computer Music Tutorial [ROA 96]	X	X	X		X
Computer music: synthesis, composition, and performance [DOD 97]	X	X	X		X
Maximun MIDI: music applications in C ++. [MES 98]		X			X
MSP – The Documentation [DOB 98]			X		X
Composing Interactive Music [WIN 98]	X	X			X
Computer Sound Synthesis for the Electronic Musician. [MIR 98]	X		X		X
Composing Music with Computers. [MIR 01]	X				X
MEPSOM	X	X	X	X	X

A tabela 2 apresenta cinco características fundamentais para o ensino de programação sônica de computadores para músicos.

- Programação e Prática para Iniciantes em Computação e Música: o MEPSOM possui um conteúdo introdutório de programação de computadores (ver item 2.1.1, Nível da Programação Preparatória), no qual são apresentados as noções inciais de programação sônica. Além disso, disponibiliza o software para a prática de programação que deve ser utilizado pelo aluno como uma ferramenta de auxílio ao seu aprendizado. Algumas obras consultadas não possuem a introdução e destinam-se a programadores com experiência. Outras apresentam conteúdos introdutórios sobre programação, mas os autores não proporcionam maneiras de estimular a prática e o desenvolvimento desse

conteúdo. As obras ainda não apresentam o software e o ambiente de programação para iniciantes como é o caso de [ROA 96], [RED 88] e [DOD 97].

- Programação Visual, Amigável e Interativa: o MEPSOM é baseado em programação visual que utiliza linguagens amigáveis e interativas para facilitar o aprendizado do músico através de exemplos. A maioria das obras consultadas não se preocupam com essa característica. Outras desenvolvem seus exemplos utilizando linguagens declarativas de uso geral, como a linguagem C.

- Ensino de Programação Baseado em Exemplos: O MEPSOM utiliza o processo de aprendizado através de exemplos (ver o item 2.3, O Processo de Aprendizado de Programação Sônica de Computadores através de Exemplos), em que é possível o músico utilizar o computador como um laboratório de experiências sonoras. A maioria das obras consultadas também utiliza esse princípio e fazem uso de exemplos de programas [WIN 98] e algoritmos genéricos com trechos de programas [MIR 98].

- Exercícios de Programação: O MEPSOM disponibiliza um vasto conjunto de exercícios com possíveis soluções. Praticamente, para cada exemplo há um exercício de programação objetivando a prática e desenvolvimento da lógica de programação do músico. A maioria das obras consultadas não dispõe de exercícios propostos e, apesar de disponibilizarem exemplos, não desafiam o aluno a investigar, praticar e fixar o conteúdo apresentado.

- Programação da Interface Sonora: O MEPSOM busca ensinar a programação de computadores para a música, destacando a utilização da interface sonora. A maioria dos exemplos do método, bem como os exercícios propostos, produzem um resultado sonoro. Algumas obras não se preocupam com a interface sonora, não disponibilizando exemplos que possam ser escutados em tempo real.

Tabela 2– **Comparação de características relevantes presentes no MEPSOM em relação a outras obras consultadas**

Obra que tratam sobre programação para Música	**Programação e prática para músicos iniciantes em computação e música.**	**Programação Visual, amigável e interativa.**	**Ensino de Programação baseada em exemplos**	**Exercícios de Programação**	**Programação de interface sonora**
Tutorial Max Development Package [ZIC 88]	**X**	**X**	**X**		**X**
C Programming for MIDI. [CON 88]			**X**		**X**
Tutorial HyperMIDI 1.1 [RED 88]		**X**	**X**		**X**
Automated Music Composition [WIP 89]	**X**		**X**	**X**	
Computer Music Tutorial [ROA 96]			**X**		**X**
Computer music: synthesis, composition, and performance [DOD 97]			**X**		**X**
Maximun MIDI: music applications in C ++. [MES 98]			**X**		**X**
MSP – The Documentation [DOB 98]			**X**	X	**X**
Composing Interactive Music [WIN 98]	**X**	**X**	**X**		**X**
Computer Sound Synthesis for the Electronic Musician. [MIR 98]	**X**		**X**		**X**
Composing Music with Computers. [MIR 01]	**X**		**X**		**X**
MEPSOM	**X**	**X**	**X**	**X**	**X**

As tabelas 1 e 2 permitem-nos apontar as diferenças entre o MEPSOM e as demais obras consultadas. A comparação deve ser realizada através da análise comparativa entre as tabelas. Por exemplo, a tabela 2, demonstra que a obra de [WIN 98] possui caraterísticas semelhantes as do MEPSOM, porém, a tabela 1 ressalta a diferença entre a abordagem de conteúdos entre os métodos. Em [WIN 98] não são apresentados os conteúdos de Programação com Audiodigital e nem Programação para Aplicações de Educação Musical, considerados ítens fundamentais no MEPSOM.

Acreditamos que o processo de implementação e validação do MEPSOM irá resultar numa contribuição significativa para a área de Computação Musical. O MEPSOM já está sendo utilizado na UFRGS, na disciplina de Programação de

Computadores para a Música, cujo os resultados são resumidamente apresentados no final deste texto.

2.2 Apresentação e Organização do MEPSOM

O MEPSOM é baseado em dois critérios: o técnico, constituído pela computação e música, tecnologia musical e informática; e o musical, constituído pelas áreas de composição e educação musical. O MEPSOM está organizado em módulos, de acordo com os critérios adotados para a divisão das áreas de conhecimento. Um módulo é um conjunto de conteúdos de uma área do conhecimento utilizada no MEPSOM. A figura 1 representa graficamente a organização geral do MEPSOM e suas divisões modulares.

Os quadros com cantos arredondados representam os Módulos do Nível de Conhecimento Básico, constituídos pelo conjunto de fundamentos computacionais, técnicos e musicais utilizados no MEPSOM. O Módulo Elementos Básicos da Programação de Computadores tem o aspecto de figura oval e aborda conceitos de programação tradicionais, como controle de fluxo e estruturas de dados.

As ligações entre os Níveis de Conhecimento Básico e Níveis de Programação Preparatória representam uma relação de dependência causal, ou seja, o método precisa ser seguido, pelo professor de computação musical, na ordem hierárquica em que está apresentado. O gráfico da figura 1 sugere que o estudante tenha conhecimento dos Módulos do Nível de Conhecimento Básico para depois cursar os Módulos de Programação Preparatória.

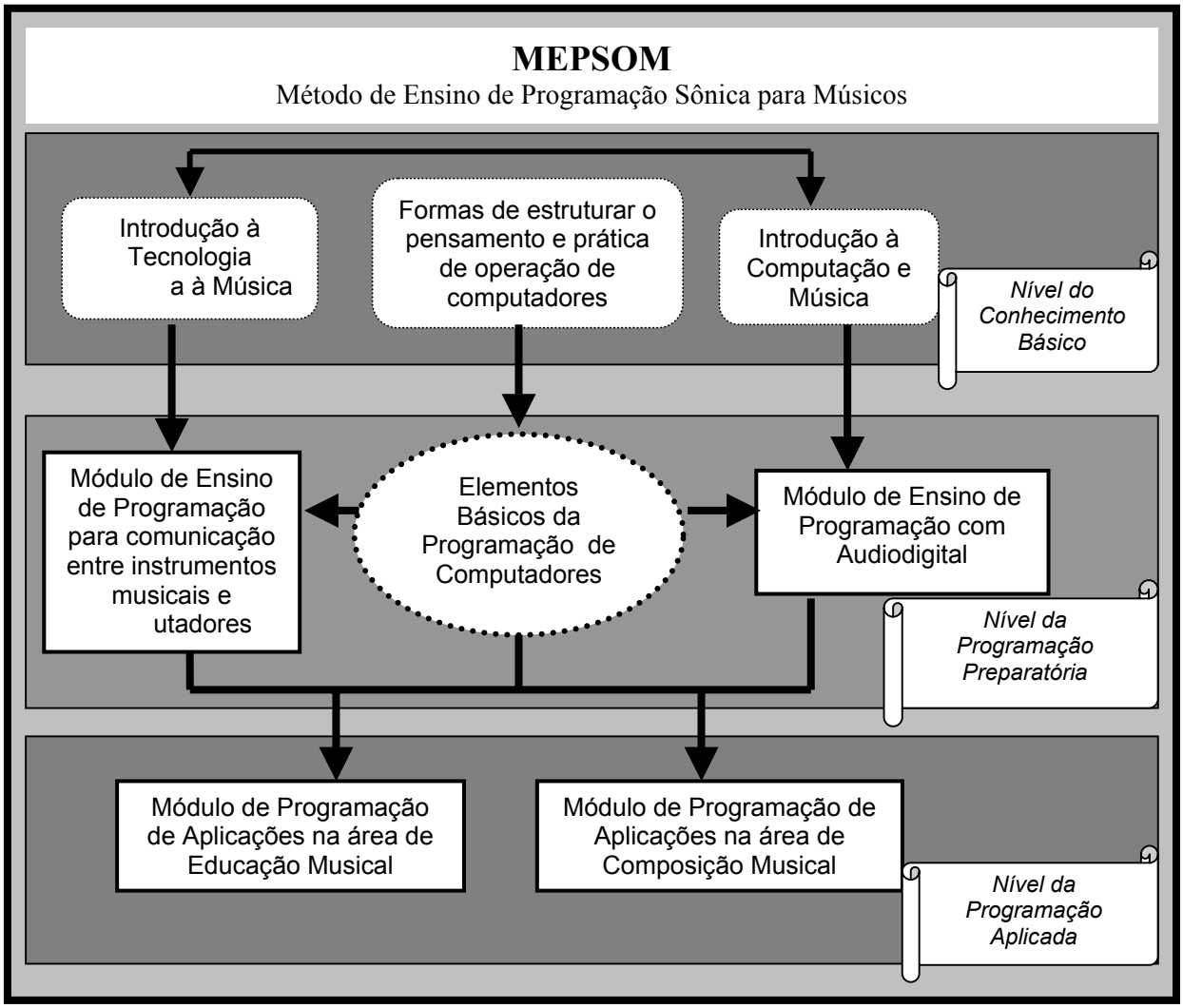


Figura 1 – A organização básica do MEPSOM

As ligações do módulo de Elementos Básicos da Programação de Computadores aos demais módulos demonstram que todos os conceitos da programação lhes são também pertinentes. Todos os módulos dos Níveis de Programação Preparatória e Aplicada utilizam os Elementos Básicos da Programação de Computadores. Portanto, este contém a informação básica necessária para que outros módulos possam desenvolver as tecnologias de comunicação entre instrumentos e o processamento de audiodigital, utilizando recursos da programação.

As setas que ligam os módulos do Nível da Programação Preparatória aos do Nível da Programação Aplicada demonstram que existe uma relação de dependência entre eles. Por exemplo, o Módulo de Programação de Aplicações na área de Composição Musical necessita dos conteúdos pertencentes a todos os módulos do Nível da Programação Preparatória. O mesmo acontece com os outros dois Módulos do Nível de Programação Aplicada.

Os Módulos do Nível de Programação Aplicada são terminais. Eles apresentam as implementações finais do Método em cada área. Nesse nível está todo o conhecimento do método sobre música utilizado para a solução de problemas específicos (ver exercícios nos anexos).

Através dessa organização, o MEPSOM oferece a possibilidade da inclusão de outras áreas da música no Nível de Programação Aplicada.

2.2.1 Nível de Conhecimento Básico

O conhecimento básico musical pressupõe que o músico tenha formação acadêmica ou equivalente. Para utilizar o MEPSOM, ele precisará de conhecimentos sobre composição, teoria e percepção, orquestração, harmonia e prática de instrumento.

Por outro lado, o nível de conhecimento básico em Tecnologia Aplicada à Música requer que o músico possua conhecimento sobre as tecnologias atuais que são empregadas nas mais diversas tarefas musicais. O conhecimento básico em matemática e lógica é utilizado no MEPSOM para a organização e estruturação do pensamento em algoritmos. Portanto, o conhecimento básico presente no MEPSOM está dividido em três áreas: Tecnologia Aplicada à Música, Formas de Estruturar o Pensamento e Computação e Música.

2.2.1.1 Introdução à Tecnologia Aplicada à Música

O músico programador precisa dominar a tecnologia musical que irá utilizar durante a elaboração dos programas. Necessita conhecer ligações MIDI, diferentes tipos de arquivos de áudio, taxas de amostragem digital, memória e instrumentos eletrônicos.[5] Se possível, deve sempre buscar o conhecimento no *estado da arte* com o objetivo de compreender novas tecnologias que poderão ser utilizadas para aplicações musicais. A figura 2 apresenta o conteúdo básico em Tecnologia Aplicada à Música, o qual deve ser conhecido antes de iniciado o estudo de Programação de Computadores para Música.

[5] Por exemplo, ao escolher a tecnologia MIDI como tecnologia de comunicação entre instrumentos eletrônicos, o músico precisa conhecer as principais mensagens, como *Note On*, *Note Off*, *Control Change*, *Program Change* e como utilizá-las.

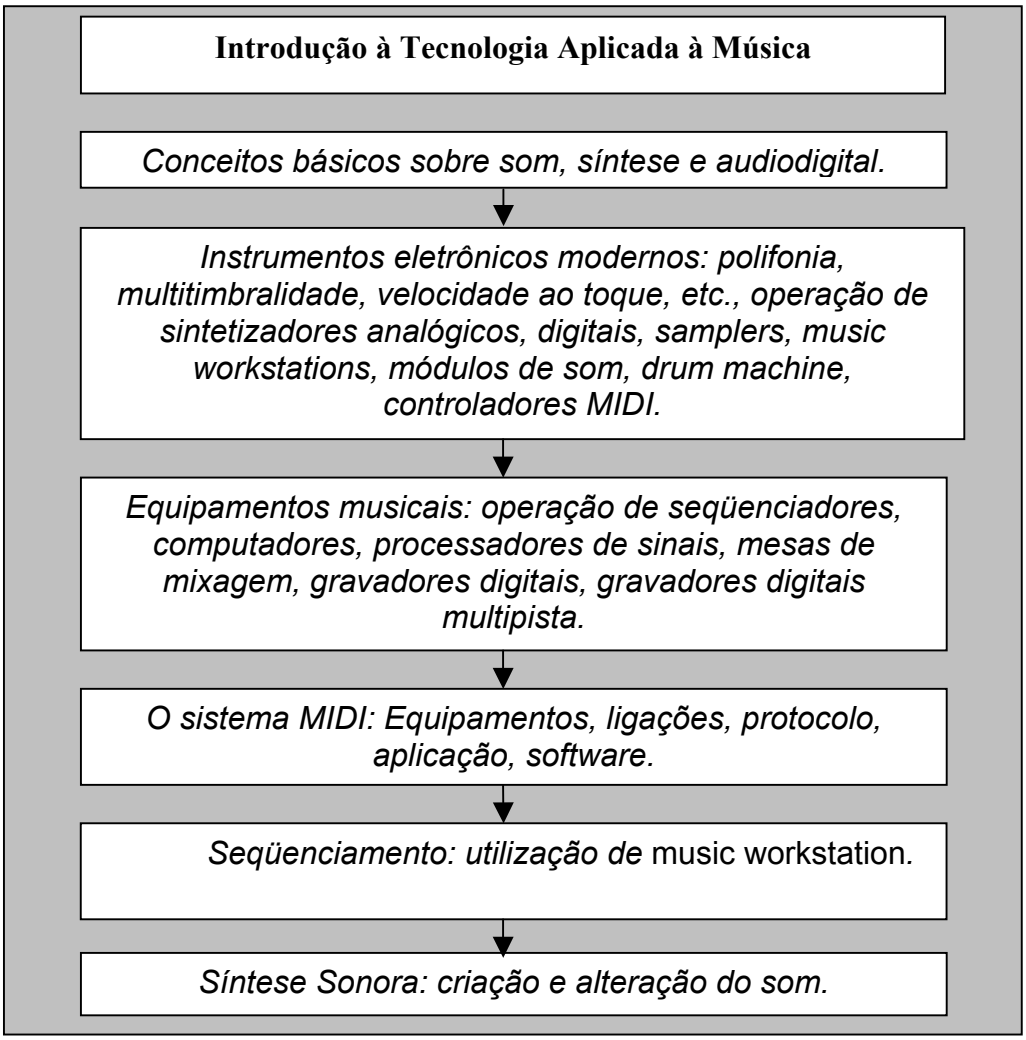


Figura 2 – Conhecimento tecnológico básico para a Programação de Computadores para a Música

Podemos observar, na figura 2, que existe uma seqüência para o aprendizado dos conteúdos indicado pelas flechas. Essa proposta de conteúdos foi utilizada na disciplina de Tecnologia Aplicada à Música do curso de graduação da UFRGS, nos anos de 2000 e 2001.

2.2.1.2 Formas de estruturar o pensamento e prática de operação de computadores

O conhecimento computacional necessário para a programação de computadores para música possui dois enfoques: o enfoque operacional e o enfoque lógico-matemático, vide figura 3.

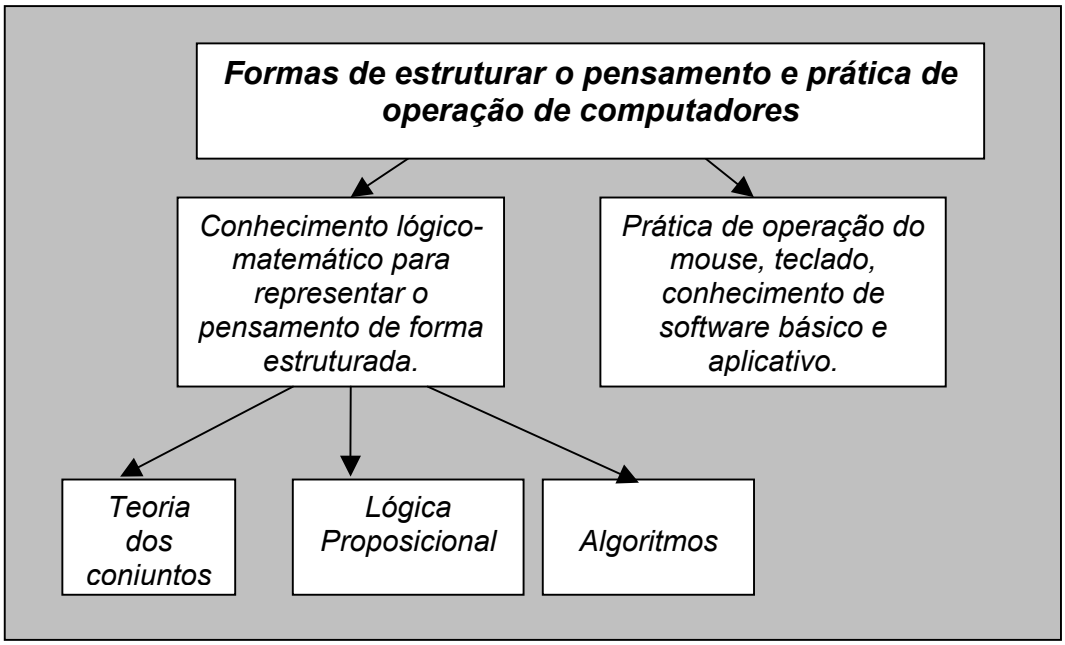


Figura 3 – Conhecimento lógico-matemático e prática de operação de computadores necessários para o aprendizado através do MEPSOM

O enfoque operacional está relacionado com a aptidão e a experiência que o músico possui para utilizar o computador como um usuário final. Quanto mais experiência na utilização de software musical, mais habilidade o músico terá em lidar com tarefas musicais no computador e, por conseqüência, terá maior compreensão ao tentar programar pela primeira vez. A agilidade nos processos operacionais de controle da interface do computador possibilita uma maior concentração na programação do computador. Através dessa destreza com o computador, o músico otimizará seu tempo, tornando desnecessários certos processos em que necessitaria pensar para efetuar operações no computador , tais como, "o que selecionar com o mouse?" ou "o que teclar agora?", utilizando este tempo no processo de programação. Atualmente, com a necessidade do emprego da Internet como meio de comunicação e dos editores de texto, como padrão para documentos, os músicos já possuem um contato mais freqüente com o computador. Portanto, mesmo que o músico ainda não tenha contato com software musical, o simples fato de ele utilizar a internet e um editor de textos colabora para a sua preparação no enfoque operacional.

Para o enfoque Computacional, segundo [MIR 01], é necessário o conhecimento lógico-matemático para que seja possível representar o pensamento de forma estruturada. Para isso o músico deverá estudar os conceitos básicos sobre teoria dos conjuntos, lógica e algoritmos. A teoria dos conjuntos tem, como operações mais importantes, a união e a interseção. O entendimento de matrizes é fundamental para a compreensão do armazenamento e recuperação de dados em vetores e tabelas. No conteúdo sobre lógica deve-se dar destaque aos operadores E, Ou e a Negação, por serem muito utilizados em programação. Os algoritmos são um conjunto de passos finitos utilizados na solução de um problema. Esses conceitos devem ser exercitados com os estudantes, pois são utilizados durante o MEPSOM. Maiores detalhes sobre esses conteúdos podem ser encontrados nos anexos desta Tese de Doutorado.

2.2.1.3 Introdução à Computação e Música

O músico programador deve conhecer e utilizar programas de música, para adquirir um conhecimento prévio a respeito dos principais produtos comerciais existentes nas áreas de edição de partituras, seqüenciamento, síntese, *sampler*, educação musical, acompanhamento e gravação de audiodigital (vide figura 3). Além desses produtos servirem de exemplo, eles propiciam um contato do músico com o ambiente de construção musical através do computador. Ainda, quando o músico estiver desenvolvendo seu próprio programa, poderá necessitar gravar os resultados, ou então, editar uma partitura no computador para registrar os resultados.

A figura 4 apresenta uma ordem seqüencial de conteúdos. Essa ordem é cronológica, não sendo, definitiva, pois dependente de vários fatores, tais como a qualidade e a quantidade de recursos disponíveis no laboratório de Música Eletroacústica para exemplos práticos e evolução das tecnologias para música. Por exemplo, o ensino do uso de Sintetizadores Virtuais deve ser abordado após o conteúdo sobre MIDI e seqüenciamento, já que os resultados obtidos da síntese poderão ser gravados num seqüenciador.

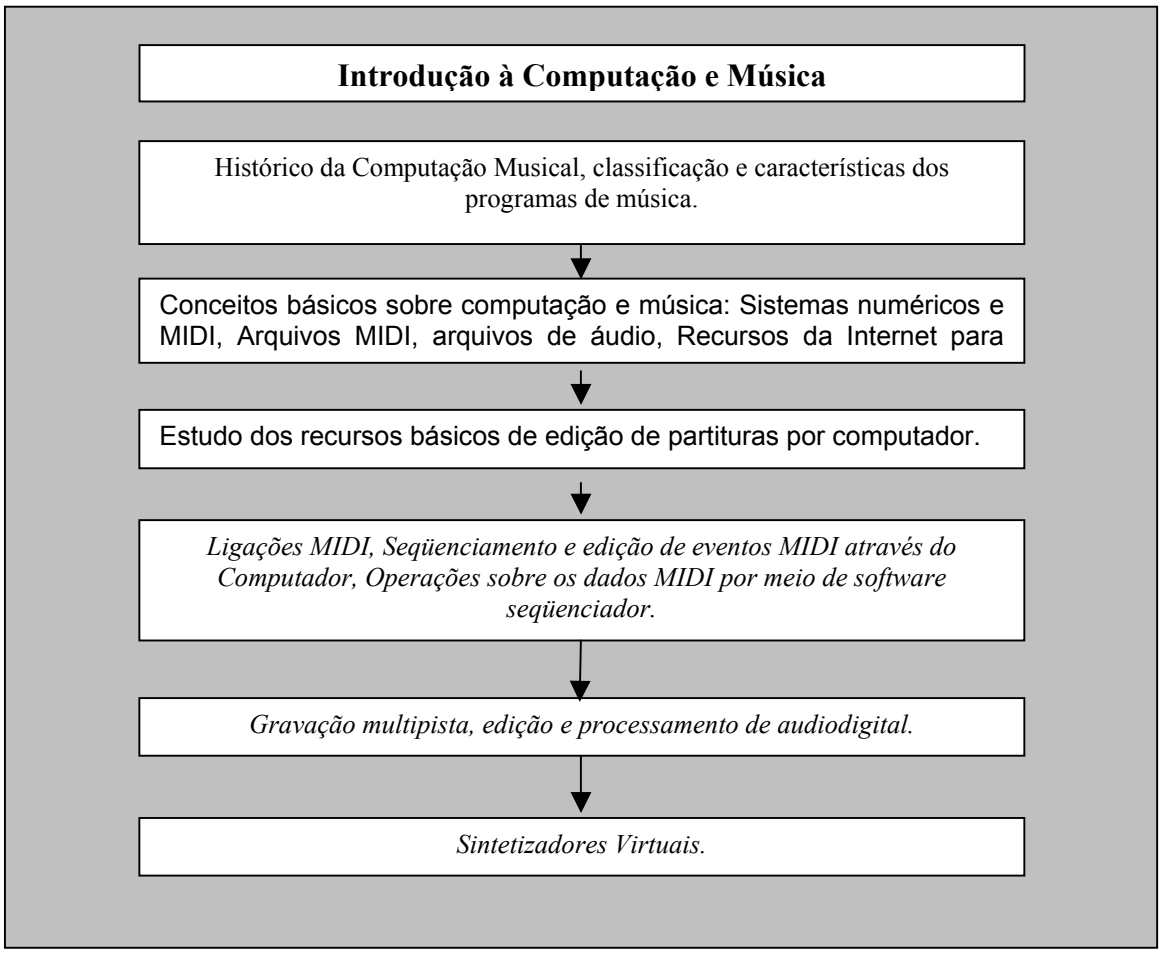


Figura 4 – Introdução à Computação e Música

2.2.2 Nível da Programação Preparatória

O Nível da Programação Preparatória tem a função de apresentar o conhecimento técnico necessário para a construção de um processo de aprendizagem em que o músico desenvolverá sua capacidade de programação. O Nível da Programação Preparatória, conforme figura 1, está dividido em três módulos: Ensino de Programação para a Comunicação entre Instrumentos Musicais e Computadores; Elementos Básicos da Programação de Computadores; e Ensino de Programação com Audiodigital. A seguir são apresentados os três módulos em detalhes.

2.2.2.1 Módulo de Ensino de Programação para Comunicação entre Instrumentos Musicais e Computadores

A programação de computadores para a música é um meio flexível e adequado para controlar instrumentos musicais eletrônicos através do computador. Este pode enviar e receber mensagens de controle e até ser o centro de uma rede de comunicação entre instrumentos e equipamentos dedicados à música eletrônica.

Este módulo do MEPSOM tem o objetivo de introduzir as principais funções de comunicação entre instrumentos musicais e computadores. A implementação dos exemplos e exercícios do MEPSOM utilizam o protocolo MIDI[6]. Portanto, como pode ser explicado na figura 5, este módulo apresenta um conjunto de exemplos e exercícios para ilustrar a programação para controle e comunicação de instrumentos musicais.

Para o ensino de programação de comunicação de dados entre computadores e sintetizadores, foram escolhidos exemplos simples para geração e organização de alturas e ritmos, e o controle da troca de timbres de um instrumento. Programas complexos e extensos dificultam o entendimento da programação, por isso os exemplos são adequados ao estudante que apresenta um nível de conhecimento intermediário na área.

[6] As informações MIDI transferidas de um equipamento (mestre) para outros (escravos) não carregam consigo nenhum sinal de áudio (som). Essas informações são mensagens que codificam as ações do músico, tais como o ato de pressionar uma tecla, ou o de soltar uma tecla. O som (áudio) conseqüente do ato de pressiona-la não é transferido via MIDI, mas, sim, gerado em cada um dos instrumentos da cadeia. Dessa forma, quando um instrumento "mestre' comanda outro via MIDI, basicamente ele "diz" ao outro quais as notas que devem ser tocadas, mas o som (áudio) do segundo instrumento será determinado pelo próprio "escravo".

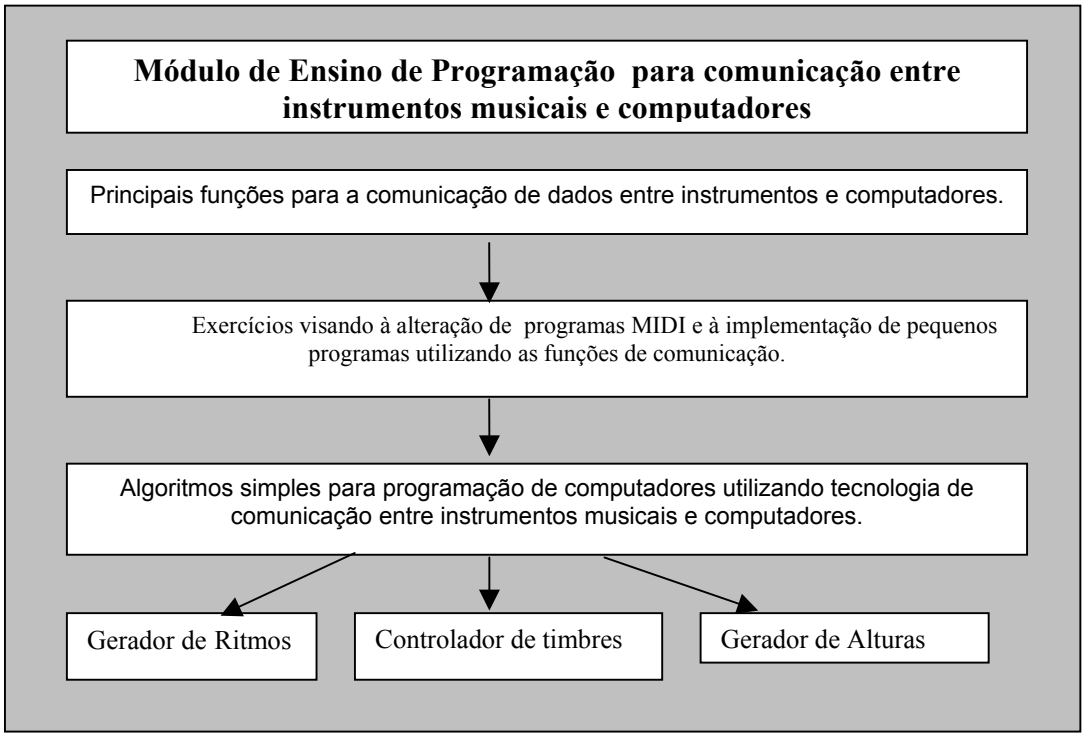


Figura 5 – Programação da comunicação entre instrumentos musicais e computadores

2.2.2.2 Módulo de Elementos Básicos da Programação de Computadores

Os elementos básicos da programação de computadores são agrupados nas seguintes categorias: operadores aritméticos e lógicos, atribuição de valores, controle de fluxo do programa, estruturas de dados, passagem de parâmetros e encapsulamento de rotinas (figura 6). Essas categorias são utilizadas pelos demais módulos do MEPSOM, pois são necessárias a quase todos os exemplos e exercícios. Ao invés de os programas do MEPSOM enfocarem somente os elementos básicos de programação de computadores isoladamente, eles enfocam os elementos básicos de programação de computadores no contexto da aplicação musical. Por exemplo, a construção condicional é apresentada para resolver um problema musical inserida num contexto em que o músico possa perceber a sua real necessidade.

O MEPSOM utiliza os elementos de programação e o conjunto das rotinas musicais para explicar ao músico como ele deve organizar e estruturar o pensamento para as tarefas de construção e alteração de programas. Conforme pode ser visto na figura 6, a ordem de apresentação dos conteúdos segue o fluxo das ligações. Após a apresentação dos elementos, o MEPSOM oferece exercícios práticos de programação com a finalidade de fixar o conteúdo e desenvolver a capacidade de programação visual do músico. Esses elementos são prioritários e indispensáveis para o ensino de programação para a música. Entretanto, o músico não deseja aprender os elementos de programação, mas sim deseja aprender como construir programas para auxiliá-lo nas tarefas musicais. Os elementos de programação são apenas o meio pelo qual o músico conseguirá transformar suas idéias e necessidades em programas úteis. Portanto, os elementos básicos de programação e as rotinas musicais desenvolvidas através do MEPSOM possibilitam que o músico desenvolva sua capacidade de criação, permitindo-lhe implementar seus próprios aplicativos musicais.

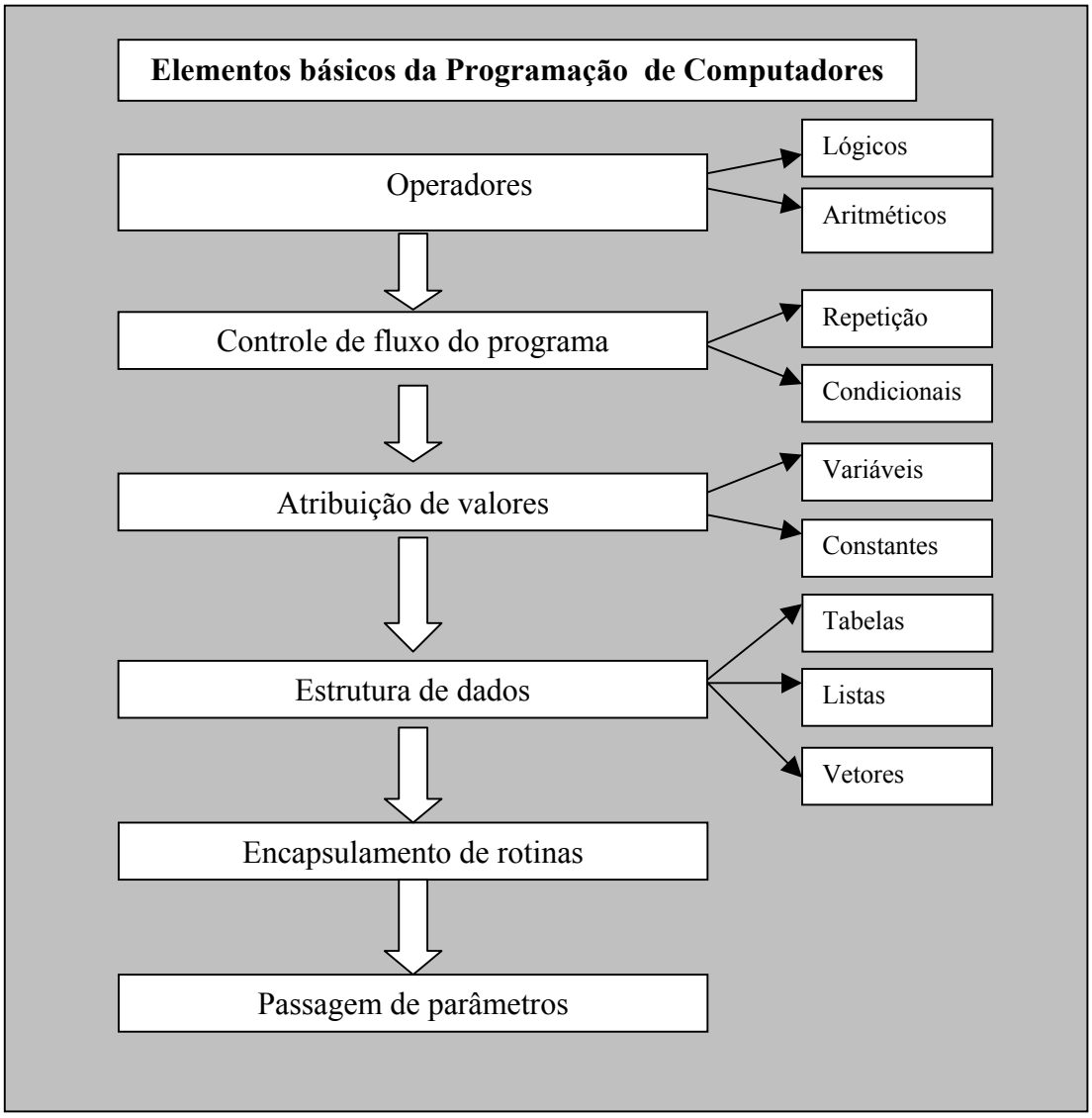


Figura 6 – Elementos Básicos da Programação de Computadores

A grande maioria das linguagens de programação para a música dispõe desses elementos de programação. Apesar do MEPSOM ter sido implementado em MaxMsp[7] [ZIC, 88] [DOB 98] e HyperCard/HyperMIDI[8] [RED 88] [GOO 90], ele independe da linguagem de programação para a implementação de seus exemplos e exercícios. Significa que o MEPSOM pode ser desenvolvido, em tese, para qualquer linguagem de programação apropriada ao desenvolvimento de aplicações musicais. Entretanto, o MEPSOM é direcionado para a utilização de linguagens visuais e amigáveis.

[7] MaxMsp é uma linguagem de programação visual para a música, síntese sonora e processamento de sinais.

[8] HyperCard/HyperMIDI é uma linguagem de programação visual para aplicações musicais que disponibiliza recursos de programação MIDI.

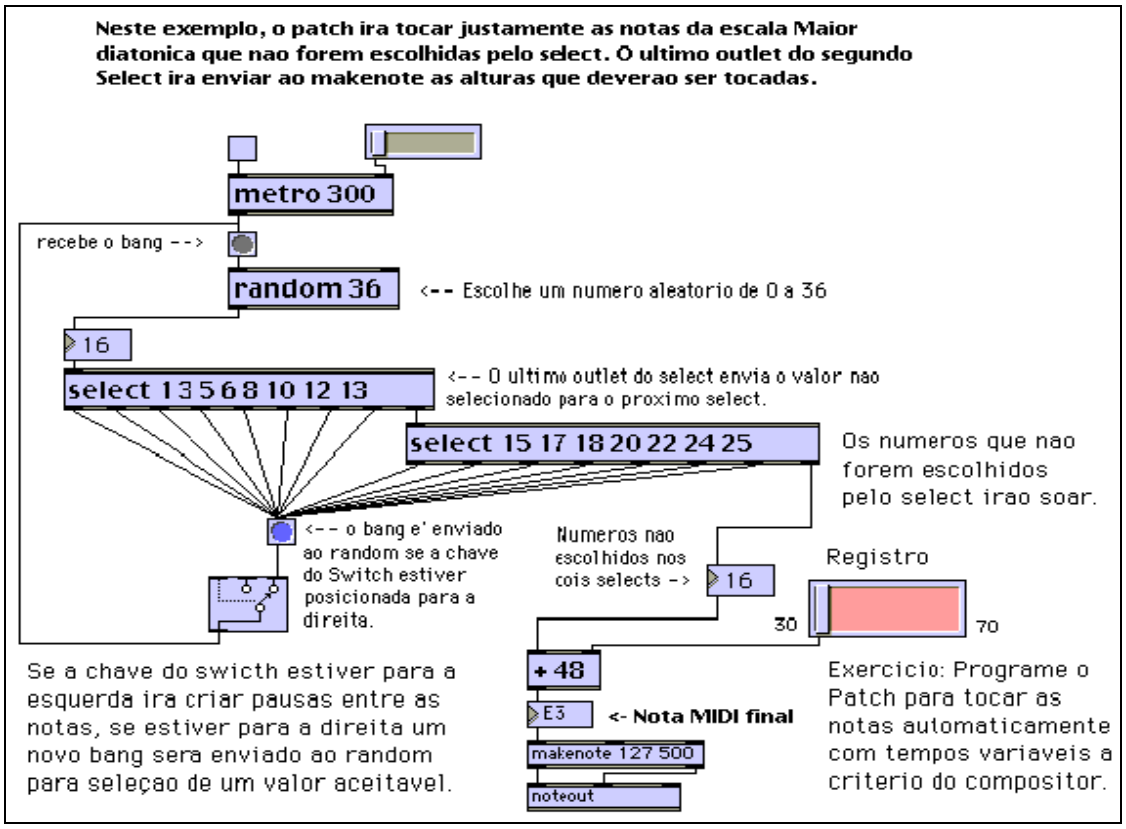


Figura 7 – Exemplo do emprego de Elementos Básicos da Programação de Computadores

Por exemplo, na figura 7, estão presentes vários elementos de programação, como estruturas de repetição e comandos condicionais. Os elementos de programação do MEPSOM são apresentados para resolver um problema musical inserido numa situação na qual o músico possa perceber a sua real necessidade.

2.2.2.3 Módulo de Ensino de Programação com Audiodigital

A programação de computadores utilizando rotinas de manipulação de audiodigital envolve o conhecimento teórico sobre síntese sonora [MIR 98].

O MEPSOM possui um módulo para o ensino de programação com audiodigital, porém pressupõe que o músico já possua um conhecimento técnico da área. Tal conhecimento é previsto nos módulos de conhecimentos básicos do MEPSOM.

O Módulo de Ensino de Programação com Audiodigital também utiliza o conhecimento provindo do Módulo de Elementos Básicos de Programação de Computadores.

O Módulo de Ensino de Programação com Audiodigital, conforme a figura 6, prevê exemplos e exercícios com as principais funções de síntese sonora, como osciladores, amplificadores e filtros. Cada elemento de síntese possui uma função já implementada em linguagem de programação. O que o MEPSOM se destina a fazer é explicar como essas funções podem ser agrupadas, organizadas, programadas e utilizadas para fins musicais.

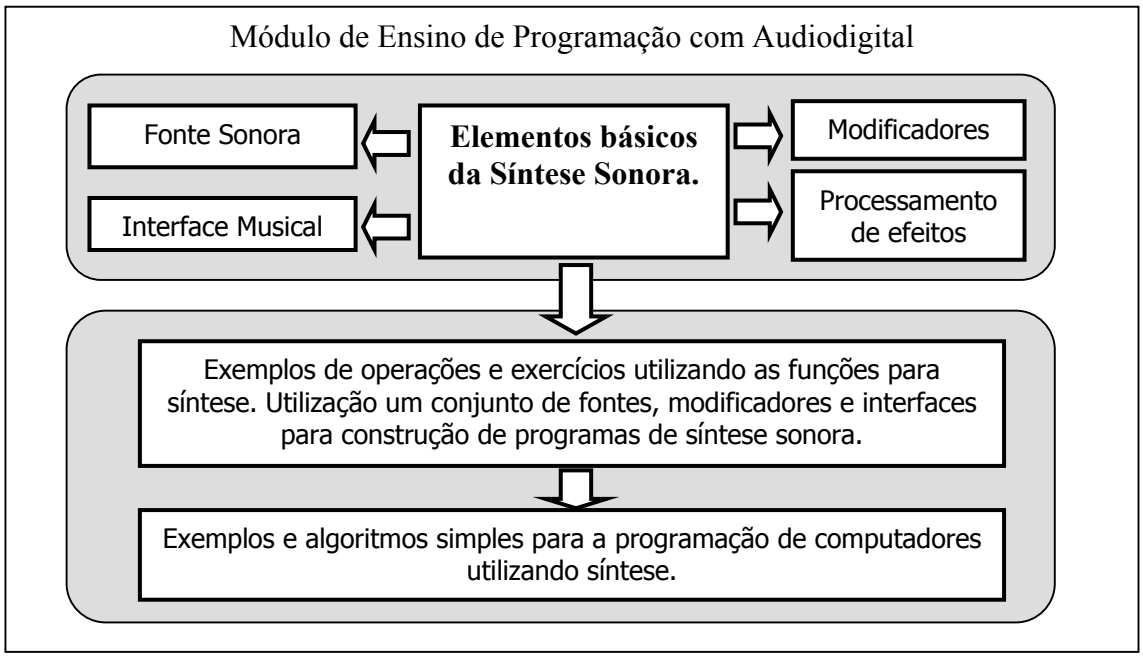


Figura 8 – Conteúdos sobre programação de síntese e audiodigital

A figura 8 apresenta as três fases do Módulo de Ensino de Programação com Audiodigital. A primeira fase destina-se a apresentar exemplos sobre os elementos básicos de síntese que são formados pela fonte, modificadores e interface musical. Este módulo ainda aborda a utilização de funções para processamento de efeitos, principalmente técnicas de espacialização. A segunda é constituída por exemplos e exercícios, utilizando os elementos básicos da síntese sonora. A terceira fase, no entanto, tem o objetivo de sugerir exercícios para o músico através da utilização de programas simples.

2.2.3 Nível de Programação Aplicada

O Nível de Programação aplicada à Musica consiste em exercitar, consolidar e aperfeiçoar todo o conhecimento provindo dos módulos de Conhecimento Básico e de Programação Preparatória através de sua aplicação em áreas distintas da música. As escolhidas para o MEPSOM são Composição Musical e Educação Musical.

2.2.3.1 Composição Musical

A Programação para Composição Musical foi desenvolvida considerando que o aluno já domine técnicas de organização e geração de materiais musicais. A adoção de métodos matemáticos para a composição utilizando técnicas estocásticas[9] e fractais[10] costuma ser muito empregada em programação de computadores para a geração de material musical e podendo essas técnicas serem úteis ao compositor. Portanto, quanto mais aprofundado o conhecimento em composição, maior serão suas possibilidades de desenvolvimento de algoritmos a serem programados de acordo com os critérios estéticos e intenções de expressividade desejados na composição.

[9] Métodos estocásticos podem produzir material através de processos não determinísticos utilizando as teorias da probabilidade.

[10] Um fractal pode ser entendido como construções musicais formadas pela iteração aproximada e infinita de padrões em escalas cada vez menores. A isto chamamos auto-similaridade.

Enquanto a composição por computador em laboratório através da pré-programação de tarefas musicais, visa à criação de música eletroacústica que só pode ser reproduzida em mídia digital, a composição em tempo real auxiliada por computador pode visar a uma performance com a interação de músicos. O MEPSOM dispõe de exemplos e exercícios para composição em laboratório e também para a composição em tempo real para performance.

Neste módulo são utilizados exemplos de programas de composição com o emprego das seguintes técnicas:

- Composição Interativa[11] transformando coordenadas da tela do computador em material musical, (vide figura 9).
- Composição automática[12] eleatória a partir de parâmetros iniciais de entrada.
- Composição automática a partir de materiais musicais (séries) fornecidos ao programa pelo compositor.
- Construção de um programas para síntese sonora com o objetivo de gerar e modificar material musical (vide figura 10).
- Composição através de métodos estocásticos.
- Composição através de fractais.

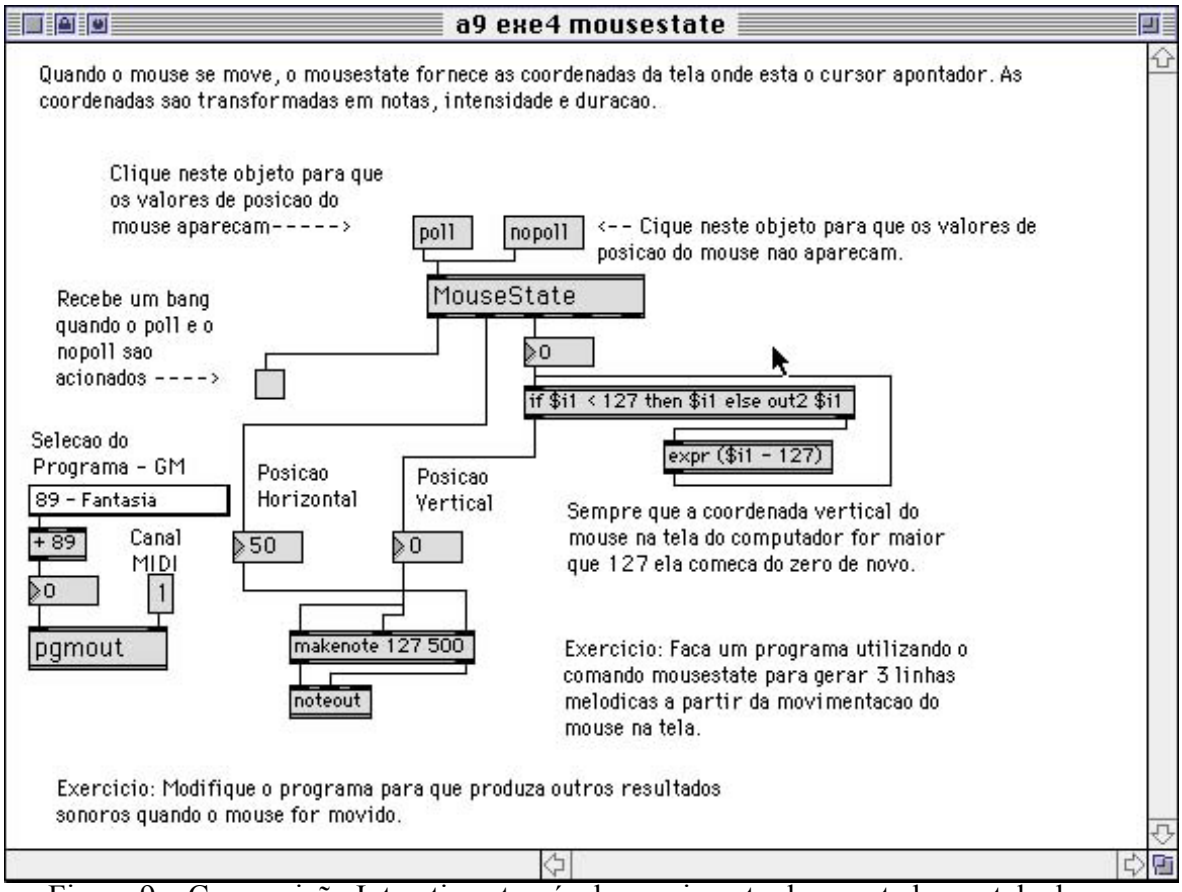


Figura 9 – Composição Interativa através do movimento do apontador na tela do computador

[11] A composição interativa permite a interferência do usuário durante o processo de criação. É uma operação limite entre a composição e a improvisação [YAV 92].

[12] A composição automática realiza toda a geração do material musical a partir de entradas fornecidas pelo usuário, não permitindo a interferência humana ao longo do processo de geração [YAV 92].

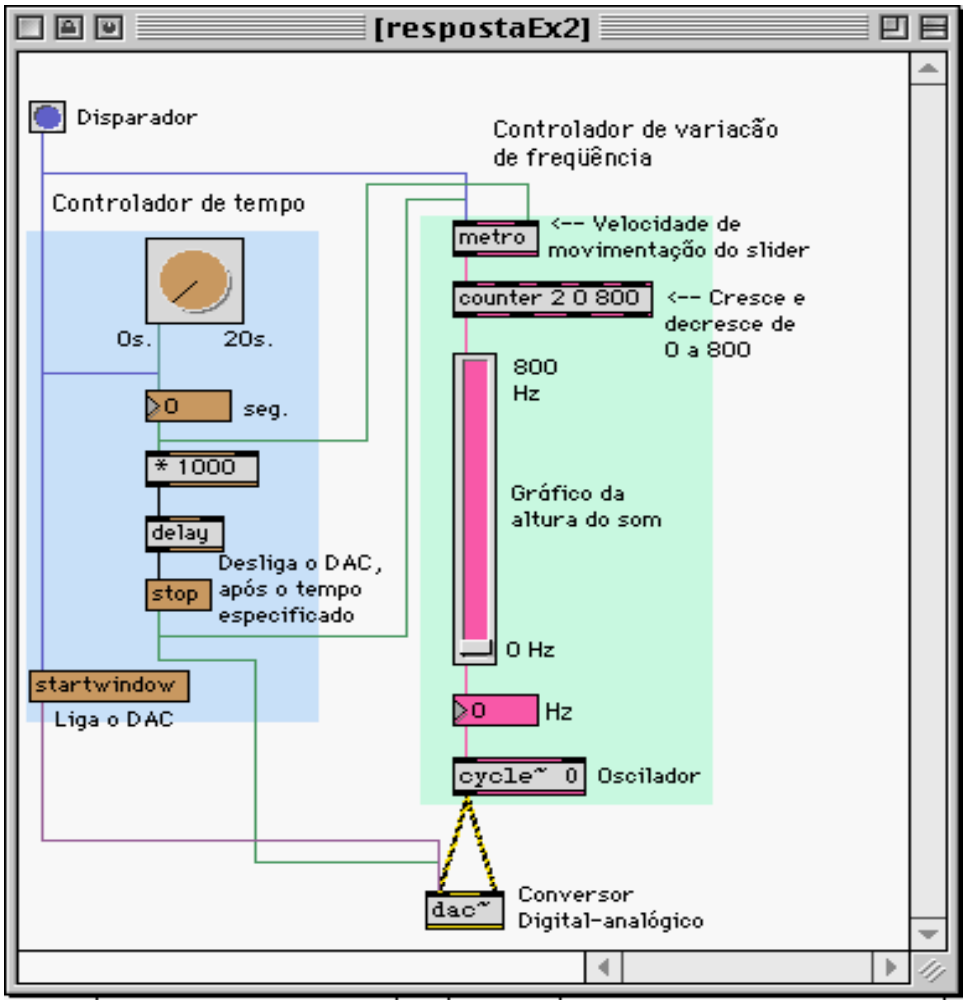


Figura 10 – Programa para controle de um oscilador

2.2.3.2 Educação Musical

O computador pode ser utilizado como uma ferramenta para o ensino/aprendizagem de música, sendo o desenvolvimento de um software de educação musical um projeto de caráter interdisciplinar. Entre as áreas de conhecimento envolvidas, podemos citar especializações da Informática, como Computação Musical, Multimídia, Informática na Educação, Interação Homem-Computador e Engenharia de Software; e áreas do domínio da Música, como Educação Musical e Psicologia da Música. O músico que deseja desenvolver software educacional deverá ter noções dessas áreas. Tais noções estão incluídas nos exemplos e exercícios deste módulo, principalmente as de como traduzir as concepções de educação musical em um software e de como proporcionar uma boa interação do aluno com o software educacional, de modo a não prejudicar o processo de ensino/aprendizado pretendido. Cabe salientar que essas noções foram igualmente aplicadas no próprio desenvolvimento dos programas que compõem o método.

Para estudar este módulo, o músico necessita ter conhecimento prévio dos métodos de ensino de música que, posteriormente, poderão ser adaptados para o computador. A importância da prática pedagógica e da experiência com alunos de música será fundamental para a criação dos programas educacionais. Quanto mais profundo for o conhecimento do programador em Educação Musical, maiores serão as

possibilidades de abstrair o conhecimento e representá-lo sob a forma de software. Nesse sentido, a metodologia de trabalho interdisciplinar do LC&M tem seguido as diretivas do Modelo (T)EC(L)A[13] [SWA 79], na criação de software de educação musical, com o objetivo de que o mesmo propicie o desenvolvimento musical da forma mais abrangente possível. Este conjunto de atividades tem sido o referencial para muitas pesquisas curriculares na educação musical brasileira [HEN 96a], [OLI 96]. A aplicação do Modelo (T)EC(L)A ocorre na definição do conteúdo do software, em atividades que serão oferecidas ao seu "usuário final" (o aluno). Para a forma de apresentação deste conteúdo (didática, estratégias de ensino), podemos observar as mais recentes propostas curriculares desenvolvidas na educação musical brasileira. Destaca-se a pesquisa do Currículo ALLI, realizada por Liane Hentschke e Alda Oliveira [HEN 96] a qual combina a Teoria Espiral de Desenvolvimento Musical [SWA 88] com o Modelo (T)EC(L)A.

Na prática, o software desenvolvido dessa maneira poderá apresentar várias partes, que poderão estar apresentadas na tela do computador como janelas, cada uma proporcionando uma determinada experiência musical de um certo tipo envolvendo o mesmo tópico do ensino de música. O software musical STR - Sistema de Treinamento Rítmico [FRI 98] é um exemplo de aplicação com referências ao Modelo (T)EC(L)A. O STR está descrito em [KRÜ 99]. A aplicação conjunta do Modelo (T)EC(L)A e a Teoria Espiral em aplicações da informática em educação musical está contemplada no Roteiro para Avaliação de Software para Educação Musical, desenvolvido por [KRÜ 2000].

É grande a importância dos conhecimentos de Interação Homem-Computador (IHC) para o futuro programador de software para educação musical. Sua aplicação se dará principalmente no momento da definição de como o aluno irá interagir com o programa, ou seja, no projeto da "interface com o usuário" (ou "IU"). Para a área de IHC, a "usabilidade" é uma qualidade desejável nas IUs, caracterizando-se pela facilidade no aprendizado e uso, taxa de erro mínima e alta atratividade dessas interfaces.

Na concepção de software educacional, mesmo que a IU não seja o único elemento responsável pelo caráter "educacional" do aplicativo, é imprescindível que haja uma preocupação redobrada com essa qualidade [VAL 2000]. Isso porque a interface com o usuário de um software que pretende auxiliar o ensino deve ser a mais transparente possível, evitando que o aluno gaste seu tempo aprendendo a usá-la, ao invés de aprender novos conhecimentos através da interface [WIN 2000]. Problemas na interação do aluno com a IU, denominados "problemas de usabilidade" [BEV 95], podem levar este aluno "a conclusões equivocadas ou errôneas, tornar o uso do computador uma experiência frustrante e até mesmo causar desinteresse pelo estudo" [WIN 2000]. Assim, uma IU educacional deve não somente ter características que enriqueçam o processo de ensino/aprendizagem, mas que, fundamentalmente, não o comprometam. Segundo [SQU 96], a integração entre fatores de usabilidade e de pedagogia não têm ocorrido. Os autores asseguram que "somente porque uma interface é fácil de usar não significa que a mesma seja programada adequadamente sob uma perspectiva educacional" e, por isso, é necessário atentar para "a forma pela qual a

[13] Modelo (T)EC(L)A: formado por atividades de Técnica, Execução, Composição, Literatura e Apreciação. As atividades de envolvimento direto *com* música são Execução, Composição e Apreciação. Técnica e Literatura são complementares (ensinam *sobre* música) por isso aparecem entre parênteses.

usabilidade e o aprendizado interagem" (p.15). Em particular, programas educacionais para a música trazem ainda mais desafios. As preocupações clássicas relativas à usabilidade são a flexibilidade de interação, robustez de interação e facilidade de aprendizado da IU. Além dessas, uma IU para um programa educacional requer ainda cuidados quanto à apresentação correta das informações musicais envolvidas tais como: a) geração de sons (notas musicais, características de timbre, amplitude, altura, etc.); b) edição e manipulação de signos musicais (pautas, claves, figuras rítmicas, etc.); c) controle da emissão de sons (ritmo, andamento, sincronização com elementos visuais, etc.), para que se atinjam com sucesso os objetivos educacionais. Alguns exemplos/exercícios que fazem parte do MEPSOM e são destinados ao ensino da programação de software de educação musical na área de Técnica musical, segundo o Modelo (T)EC(L)A), são:

- Completar a programação de um software para ditado intervalar, em que as alturas sejam executadas a partir de um instrumento externo ao computador.
- Completar a programação de um software para explicar como desenhar instrumentos na tela do computador e associa-los ao som correspondente.
- Completar a programação de um software para seqüenciamento e transposição musical (Figura 11).

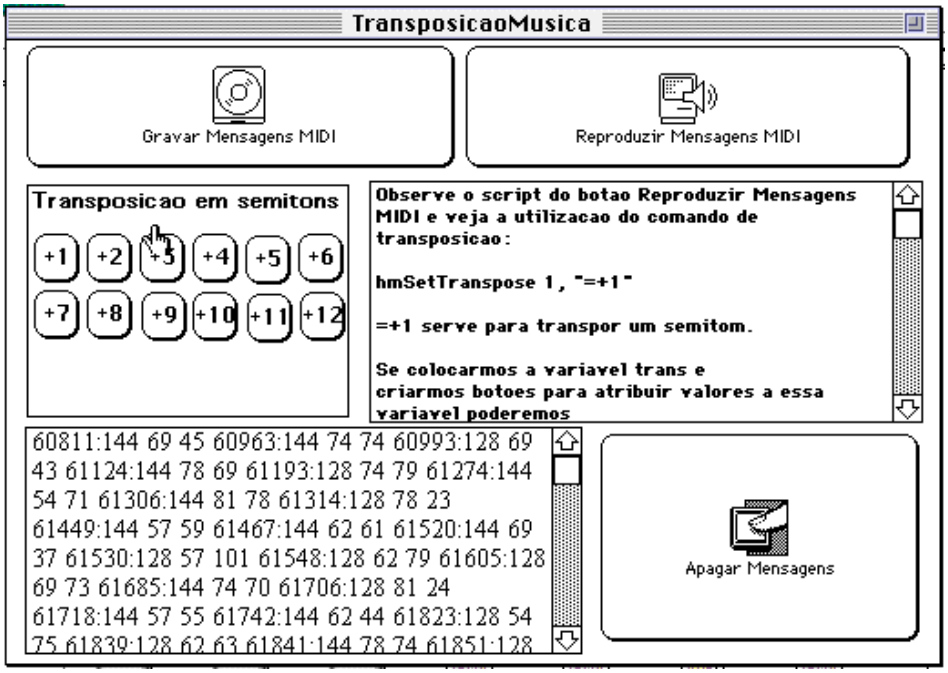


Figura 11 - Exercício para ensino de programação de software educacional.

Esses exercícios fornecem ao músico parte das ferramentas necessárias para a programação de software educacional, utilizando a comunicação entre o computador e o instrumento musical.

A utilização de software educacional em aulas de música proporciona experiências importantes, podendo ser utilizadas na tarefa de programação. O músico, tanto no papel de professor quanto de aluno, experimentará a utilização de pacotes já desenvolvidos para a finalidade do ensino de música, que poderão servir de exemplos

futuros para o projeto de novos programas em diferentes subáreas do conhecimento musical.

2.3 O Processo de Aprendizado de Programação Sônica de Computadores através de Exemplos

Programar é a arte de ensinar procedimentos ao computador. O processo de desenvolvimento de um método para ensinar músicos a programar através de uma interface visual e sonora está relacionado com os princípios do aprendizado humano e o aprendizado de máquina. O MEPSOM baseia-se no ensino através de exemplos, utilizando-os como forma de explicar idéias ao estudante. Em alguns casos, a necessidade de exemplos é um paradoxo, pois, de um ponto de vista lógico, os exemplos não adicionam nenhuma nova informação à explicação original. Se as regras são precisas e estão completas, a sua simples apresentação pelo professor e a memorização pelo estudante não deveria ser o suficiente?

A resposta é não, pois os professores às vezes não têm certeza absoluta de que estão apresentando regras totalmente completas e corretas, e os estudantes, por sua vez, não têm a certeza se estão aprendendo os princípios corretos. O uso de exemplos portanto, é recomendado na situação de aprendizagem. [LAW 85] relata um caso de sucesso com o aprendizado através de exemplos: uma criança com dificuldades em realizar adições com números escritos no papel (onde os números são abstratos) demonstrou competência em adicionar os valores de moedas ao necessitar comprar algo.

Lecionar é também para um professor uma experiência de aprendizado. Professores freqüentemente declaram que o processo de ensinar aprofunda mais seu entendimento sobre o conteúdo. Esse entendimento não raro é desenvolvido durante o processo de elaborar exemplos para auxiliar os estudantes na compreensão do conteúdo. Entretanto, o processo de programação é conhecido como uma situação de ensino dirigido. O MEPSOM apresenta um ambiente que estimula o ensino de programação de computadores através de exemplos. A figura 12 é um exemplo de programação visual e sonora do MEPSOM, sendo que, após, é sugerida a realização de um exercício para o aluno praticar o que aprendeu.

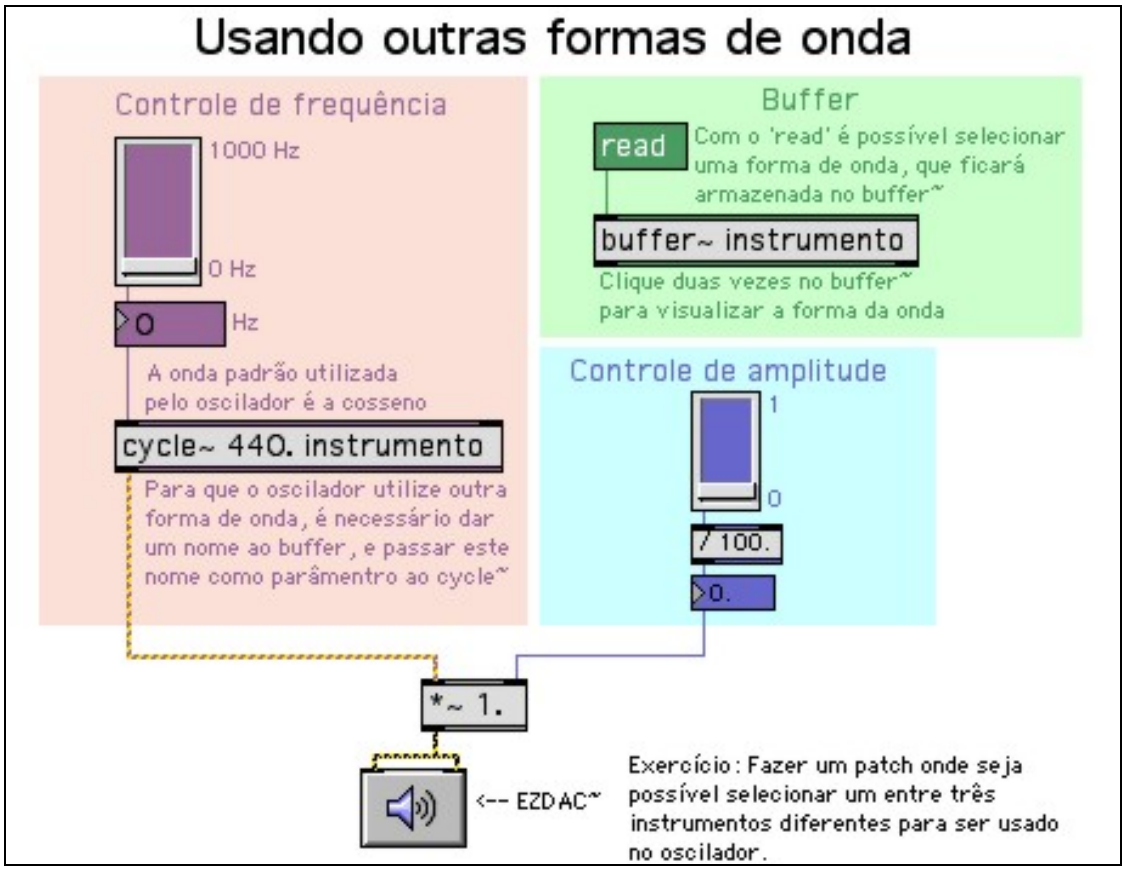


Figura 12 – Exemplo do MEPSOM com a formulação do exercício para o estudante

O MEPSOM prevê ainda que tanto o aluno quanto o professor tenham acesso às respostas. O professor deve dispor de pelo menos uma solução para o problema para poder comparar com a solução por ele obtida. O professor poderá utilizar uma ou mais soluções como resposta para determinado problema. Para que o aluno se sinta seguro ao utilizar o MEPSOM, este deve fornecer exemplos, exercícios e respostas sobre o conteúdo abordado em aula pelo professor. Todos os exemplos previstos no MEPSOM possuem exercícios com possíveis soluções. A figura 13 apresenta a solução para o exercício da figura 12.

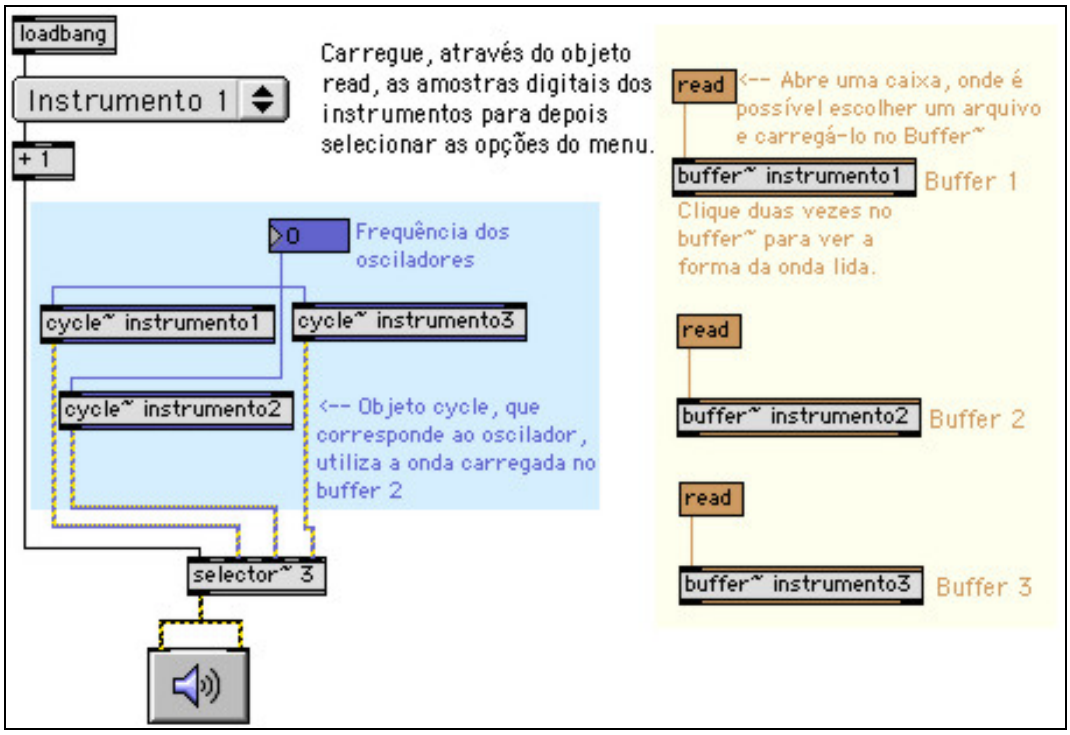


Figura 13 – Solução do exercício "Usando outras formas de Ondas"

2.4 Programação Visual

Segundo [CYP 93], o computador geralmente é visto como uma ferramenta para melhorar as habilidades cognitivas e analíticas. No entanto, neste trabalho, pretende-se que ele também contribua com as habilidades não-analíticas e com o pensamento intuitivo do músico. Por isso, para a implementação do MEPSOM, foi adotada a programação visual e sonora na intenção de desenvolver aplicações musicais.

A idéia de ensino através de gráficos não é nova. No passado, alguns estudiosos propuseram que gráficos gerados por computadores teriam um profundo efeito no aprendizado humano. Brown e Lewis escreveram em 1968: "Da mesma forma em que livros apóiam o pensamento linear e verbal do homem, máquinas irão apoiar gráfica e intuitivamente o seu processo mental". Semelhantemente, em 1969, Tony Oettinger escreveu: "Computadores são capazes de afetar profundamente a ciência ao aumentar a razão e intuição humana, assim como telescópios e microscópios aumentam a visão humana" [OET 69]. Essas afirmações nos conduzem à utilização de linguagens de programação visuais que permitam a utilização de gráficos e fluxogramas para apresentar, explicar e exercitar a programação sônica de computadores para a música.

2.4.1 O Controle do Músico Sobre a Criação de Programas

Na proposta do MEPSOM que descrevemos nesta Tese de Doutorado, o usuário começa resolvendo um exercício, fazendo um rascunho do projeto, não em papel ou em palavras, mas diretamente na tela do computador. Esse rascunho é baseado em exemplos e experiências anteriores e realizado através de um ambiente visual e sonoro de programação. Se as idéias do músico são um tanto quanto vagas, o processo de rascunho pode ajudá-lo a defini-las; se as idéias são bem definidas, elas podem ser rapidamente aceitas, rejeitadas ou melhoradas. Logo, o MEPSOM apresenta o ambiente de programação como um laboratório de experiências sonoras, no qual o aluno irá testar

e desenvolver suas soluções de programação. No processo de tradução das idéias do músico em linguagem de programação visual e sonora, nada é perdido, porque, além do professor de computação musical, o intermediário entre o músico e o produto final é um sistema gráfico de computador que fornece respostas imediatas através da emissão de sons. Como não há esperas, o músico está envolto num processo criativo no qual o investimento na produção de um programa é reduzido. Portanto, esse sistema enfatiza que uma produção não satisfatória possa ser facilmente rejeitada, e uma produção satisfatória possa ser trabalhada e melhorada.

2.4.2 A Metáfora de Ensaio

O MESPCM, por propiciar um grande ambiente para projeto de aplicações musicais, necessita de uma forte metáfora, na qual os conceitos "estranhos" de programação terão referências familiares e reais para os músicos. A metáfora deve servir como um guia para os músicos programadores, abstraindo a parte técnica pertencente à computação. Mesmo que isso não possa ser feito na sua totalidade, a tentativa de emprego de uma metáfora durante a aplicação do MEPSOM em aulas atenua o impacto tecnológico que o músico sofre ao se deparar com conhecimentos que fogem à sua área de domínio.

No MaxMsp, por exemplo, os elementos básicos de programação são tratados como objetos que interagem com outros objetos através do envio de mensagens. O MaxMsp tem vários tipos de objetos que respondem a uma grande variedade de mensagens. Assim como [CYP 93], nossa imersão em MaxMsp levou-nos a estender a metáfora "objeto-mensagem" para uma metáfora musical na qual os componentes básicos da programação são **intérpretes**. Esses intérpretes interagem uns com os outros formando a **Orquestra**. Nós chamamos o ambiente de programação de "**Sala de Ensaio**", e o processo de criação de um programa, de "**Ensaio**". Na figura 14 é apresentado um paralelo entre a metáfora de Ensaio e um programa em Max.

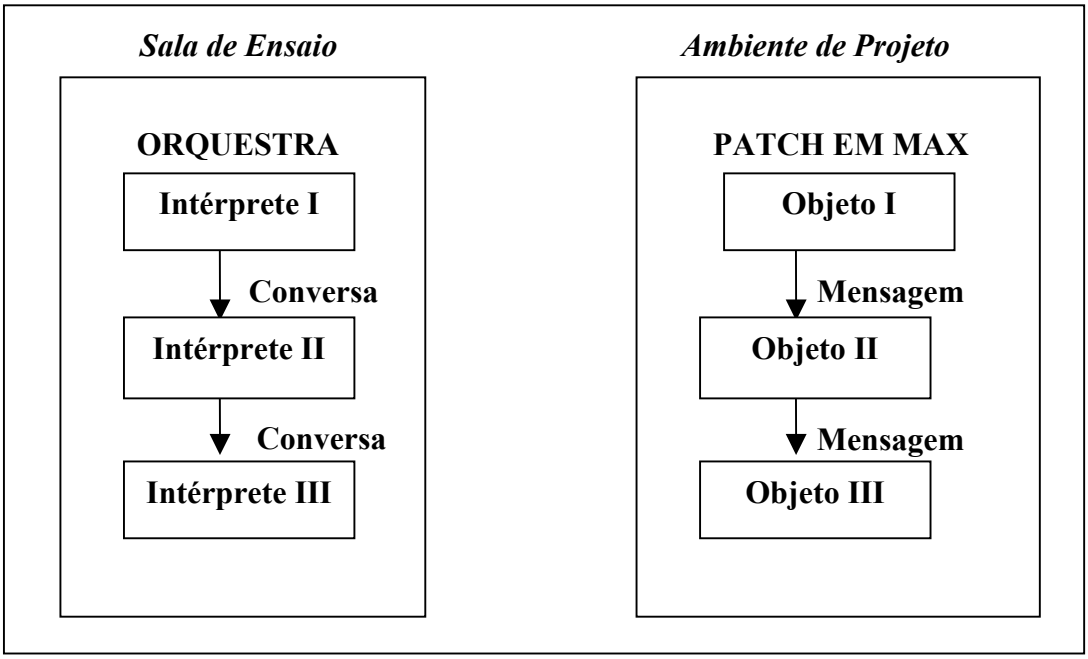


Figura 14 – A metáfora de um Ensaio comparada com um patch em MAX

Segundo [CYP 93], tudo no Mundo do Ensaio é visível; não há abstrações, e apenas coisas visíveis podem ser manipuladas. Quase todas as interações do usuário músico com a Orquestra ocorrem através da seleção (com o mouse) de um intérprete. Presumindo que um músico usuário tenha o embrião de uma idéia, a criação de um programa Orquestra envolve:

- Testar os intérpretes disponíveis, selecionando suas funções e observando suas respostas para determinar quais são apropriadas para viabilizar o planejamento de um programa.
- Selecionar os intérpretes e colocá-los na Orquestra.
- Formatar visualmente o programa através da alteração do tamanho ou da movimentação dos intérpretes, até que eles estejam com tamanho apropriado e no local desejado.
- Ensaiar o programa, mostrando a cada intérprete que ações ele deve tomar em resposta ao músico usuário ou às mensagens enviadas por outros intérpretes.
- Guardar a produção para futura reutilização.

Um aspecto importante sobre a Sala de Ensaio é que tudo é visível, e apenas coisas que podem ser vistas podem ser manipuladas. Ainda, ao invés de pensar de forma abstrata, como é necessário na maioria dos ambientes de programação, um músico está sempre pensando de forma concreta, selecionando um intérprete em particular que irá se comunicar com outro formando uma Orquestra. Após montar sua própria Orquestra, o músico, ao ouvi-la, irá avaliar o resultado instantâneo desse agrupamento de intérpretes. Portanto, pode-se afirmar que a aceitação inicial do MEPSOM, por músicos que desejam programar, deve-se ao seu enfoque concreto, visual e intuitivo.

À medida que os músicos usuários avançam no MEPSOM, criando Orquestras cada vez maiores, eles podem encontrar problemas com o espaço na tela e com a complexidade visual. Alguns desses problemas são tratados pela habilidade de encapsular uma Orquestra tornando-a disponível para os demais intérpretes. Além de

auxiliar no gerenciamento do espaço gráfico disponível da tela do computador, o músico ainda desenvolve sua capacidade de dividir um problema em partes menores, com soluções distintas do todo.

Enquanto músicos iniciantes na programação de computadores se beneficiam da concretividade, músicos mais experientes em programação, são favoreciudos por pensarem em termos mais gerais e abstratos. Em termos gerais, pelo fato de que todos os intérpretes conversam entre si através de uma mesma língua, eles são levados a pensar em termos abstratos pela manipulação de Listas e Repetições. Em algum ponto, a concretividade pode tornar-se mais uma barreira do que uma vantagem, dependendo, em parte, da aplicação musical que será desenvolvida. Por isso, o grande mérito da implementação do MEPSOM através de linguagens visuais reside na facilidade que a visualização representa para iniciantes. Ela torna possível a representação de pensamentos e intenções musicais sob a forma visual e sonora empregando fluxogramas e processamento de sinais. A metáfora de ensaio pode ser utilizada pelo professor de computação musical durante a utilização do MEPSOM.

As primeiras experiências com o MEPSOM, como veremos no item 4.4.4 Resultados, mostram que essas idéias sobre como fazer o potencial dos computadores acessível a não-programadores podem ser utilizadas com êxito. Programas visuais interativos podem e devem ser construídos dentro de um ambiente de programação orientado que disponibilize tais recursos. Além disso, para muitos algoritmos, a programação visual irá, eventualmente, substituir o atual processo de escrita textual de código. Por exemplo, o processo de síntese aditiva pode ser realizado com várias linguagens de programação declarativas para a música como csound e pcmusic. Porém, utilizando uma linguagem visual como MaxMsp ou OpenMusic [MIR 01], a programação torna-se mais interativa e acessível aos músicos. Um patch em MaxMsp feito para o MEPSOM para a síntese aditiva pode ser visto na figura 15.

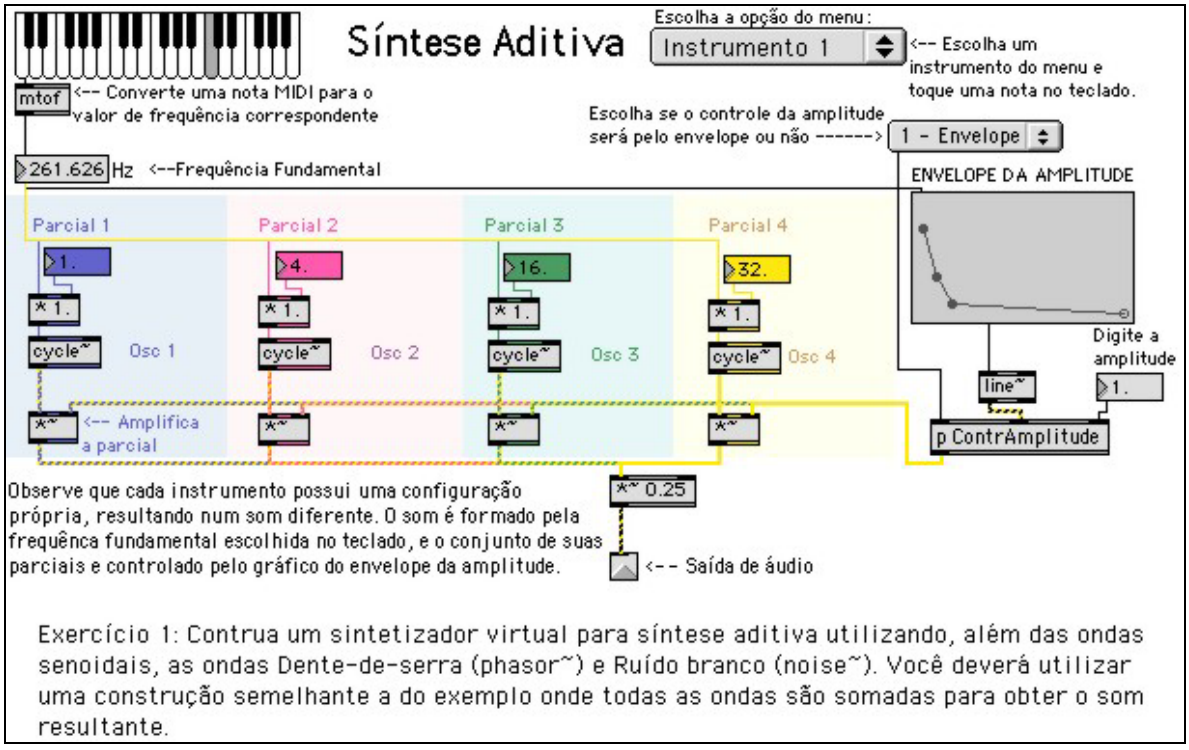


Figura 15 – Um exemplo de patch para síntese aditiva realizado num ambiente de programação visual

O sucesso no ensino depende, muitas vezes, da arte de apresentar bons exemplos aos estudantes, bons exemplos entendendo-se como aqueles que claramente ilustram as idéias que desejam ser explicadas pelo professor. Conhecendo as idéias, os estudantes têm a possibilidade de solucionar problemas. O MEPSOM faz uso dos seguintes princípios no ensino através de exemplos:

- Exemplos que mostram as similaridades e diferenças de uma idéia e idéias relacionadas ajudam a clarear princípios.
- Para aprender através do exemplo, é necessário conhecer quais as suas características mais importantes.
- Uma seqüência de exemplos deve iniciar sempre com um exemplo simples . Gradualmente, eles vão tornando-se mais complexos e, sempre no final do conteúdo, são apresentados casos de exceção.

3 Implementação do MEPSOM

Acredita-se que a implementação do MEPSOM, descrita neste texto, irá resultar numa contribuição significativa para a área de Computação Musical no Brasil. O MEPSOM já está sendo utilizado sob a forma de software musical na UFRGS, na disciplina de Programação de Computadores para a Música do Curso de Bacharelado em Composição, desde março de 2001.

3.1 Ferramentas de Programação para a Música

Atualmente existem várias aplicações musicais que utilizam o sistema MIDI. Programas comerciais, criados por programadores profissionais, são escritos em linguagem de alto nível. A linguagem "C" é uma das mais utilizadas. De um modo geral, qualquer linguagem de alto nível pode ser utilizada para desenvolver aplicativos musicais. Certas linguagens de programação apresentam mais recursos para a programação sônica e maior flexibilidade para a criação de software musical do que outras.

3.1.1 Linguagens de Programação de Alto Nível para a Música

Para programas escritos em linguagens de alto nível são necessários controladores (drivers) especiais, escritos na linguagem *Assembler*, dando total acesso ao hardware que conecta o computador à interface MIDI. Os programadores escrevem seus próprios *drivers* ou aproveitam o código dos que já existem (às vezes chamados de bibliotecas) para computadores específicos e interfaces MIDI.

A portabilidade dos programas é uma das principais vantagens do uso das linguagens de alto nível. Por exemplo, ao escrever em linguagem "C" utilizando a plataforma Macintosh, o programador poderá usar grande parte deste mesmo código para criar uma versão do programa para a plataforma Windows. Contudo, as rotinas de baixo nível para comunicação de dados, através de mensagens MIDI, dependem do hardware e apresentam menor grau de portabilidade entre diferentes plataformas. De fato, os *drivers* e os comandos utilizados para manipulação de instruções MIDI de baixo nível diferem, de acordo com a plataforma de execução. A linguagem utilizada para programação de aplicativos musicais bem como a plataforma de desenvolvimento, são escolhidas de acordo com a qualidade e a diversidade de rotinas para programação da comunicação MIDI.

3.1.2 Linguagens musicais embutidas em linguagens de programação

Linguagens genéricas podem utilizar sublinguagens embutidas que são especializadas em rotinas para programação musical. A vantagem desta implementação advém da possibilidade de a linguagem genérica estar sempre disponível para as sublinguagens. Uma linguagem de composição musical, por exemplo, pode sempre contar com a linguagem genérica para rotinas de tratamento aritmético de ponto flutuante, arquivos, entrada/saída, gerenciamento de janelas e outras tarefas utilitárias.

Segundo [ROA 96], uma linguagem funcional intrinsecamente aberta como Lisp encoraja a utilização de linguagens embutidas. De acordo com o mesmo autor, o músico pode utilizar uma linguagem genérica e invocar rotinas para realizar a síntese sonora. Por exemplo, linguagens como o Cscore e o Music 4C estão embutidas na linguagem de programação "C". Portanto, os compositores têm a liberdade de utilizar os recursos da linguagem "C" com o benefício do uso das rotinas de síntese sonora para a especificação da composição.

3.1.3 Linguagens de programação amigáveis

As linguagens *user-friendly* permitem aos programadores "não-profissionais" a chance de criar suas próprias aplicações musicais. A programação é de fácil aprendizado e permite a implementação de software através de um ambiente interativo e de uso intuitivo. Todavia, as linguagens *user-friendly* tendem a ser limitadas quanto à sua capacidade e potencial de programação. Além disso, a execução dos programas é mais lenta, visto que, na sua grande maioria, as linguagens *user-friendly* são linguagens interpretadas.

A linguagem interpretada apresenta vários benefícios para o ensino de programação de computadores para a música. Os programas escritos nessa tipo de linguagem são convertidos em linguagem de máquina ao mesmo tempo em que suas instruções estão sendo executadas. Portanto, enquanto o programa é executado, o computador pode informar os erros cometidos pelo programador.

O HyperCard é uma linguagem *user-friendly* interpretada, possuindo uma pilha (biblioteca) com uma linguagem musical embutida, chamada HyperMIDI. Essas características qualificam o *HyperCard/HyperMIDI* como uma linguagem de programação possível de ser empregada no ensino da programação sônica de computadores para músicos.

3.1.4 Linguagens de programação visual

A escolha de linguagens de programação que melhor possam servir o compositor durante o processo de construção de um programa musical deve levar em conta a forma em que o pensamento será formalizado, a interface visual e sonora e também o processo de elaboração do programa.

O pensamento pode ser formalizado através de algoritmos para exprimir o conjunto de passos finitos que devem ser seguidos para a solução de um problema. Os algoritmos, quando transformados em instruções de uma linguagem de programação, podem ser representados sob a forma gráfica ou sob a forma escrita. A programação visual possibilita que algoritmos gráficos, como fluxogramas, possam ser transcritos diretamente para o ambiente de programação, sem muitas modificações. A utilização de recursos gráficos possibilita que a solução de um problema possa ser mais facilmente projetada por músicos que não tenham um conhecimento mais aprofundado de programação de computadores.

O MEPSOM visa ao ensino de elementos básicos de programação de computadores, como estruturas de controle de fluxo e estruturas de dados. A utilização de uma linguagem visual facilita a representação da formalização do pensamento sob a forma de fluxograma. Ao representar o pensamento em forma de gráficos, linhas, botões, campos, tabelas e som, o músico está mais próximo da situação real na qual ele constrói a notação gráfica em papel para representar processos musicais. Isso é muito comum na música eletroacústica, em que o compositor define uma "partitura não

tradicional" fornecendo os processos musicais para a interpretação de sua obra através de gráficos. O MEPSOM foi projetado para aplicações de música eletroacústica e, por essa razão, o algoritmo utilizado para a criação de uma composição pode ser expresso graficamente de uma forma semelhante à escrita das linguagens visuais. Logo, o emprego de linguagens visuais para programação colaboram com a capacidade que o músico possui em representar de forma mais fácil processos musicais.

A elaboração de interfaces visuais através de procedimentos simples de programação já é bastante utilizada em software de autoria e linguagens amigáveis. Como exemplos de software musical desenvolvido através de linguagens visuais e amigáveis como o HyperCard e o Toolbook, podemos citar o SETMUS [FRI 95], STI [FRI 96] e STR [FRI 98]. Principalmente na década de noventa, linguagens para a música, como o MaxMsp, apresentavam recursos para a interface gráfica [ZIC 88] [DOB 98]. Isso possibilitou a construção de programas preocupados com a usabilidade e a interação do músico com o ambiente de operação do software. No entanto, só a capacidade de criação de interfaces gráficas não é suficiente para a programação de computadores para a música. Existe também a necessidade da criação da interface sonora, principalmente porque os programas nessa área são dedicados a aplicações que utilizam o som como o resultado final de um programa. Por isso é fundamental que, tanto a interface visual quanto a interface sonora, sejam projetadas em equilíbrio, prevendo sua utilização durante os procedimentos do programa musical.

O processo de elaboração dos programas pode ser facilitado, se a linguagem de programação for interpretada ou permitir um ambiente que auxilie o processo de escrita do programa. Em geral as linguagens visuais fornecem ambientes de auxílio à programação. Algumas delas possibilitam que o programa seja escrito num nível de abstração que permite aos programadores principiantes mais facilidade na elaboração de programas. Programas realizados em linguagens como MaxMsp assemelham-se aos próprios fluxogramas [ZIC 88] [DOB 98]. Por essa razão, em muitos casos não há a necessidade de um algoritmo prévio. O músico, devido às facilidades de representação e formalização do pensamento por meio do software, desenvolve seu programa diretamente no ambiente de programação gráfica.

O MEPSOM, ao contrário de outras obras de programação para a música disponíveis na literatura, como as de [WIP 89], [CON 88], [DOD 97], desenvolve toda a implementação com ênfase na utilização de interfaces sonoras e visuais. Os programas são elaborados visando à programação gráfica através da organização de linhas e objetos.

Por exemplo, algoritmos para geração de material musical para a composição podem ser apresentados em linguagem visual, objetivando facilitar a compreensão do algoritmo pelo músico. Como técnica de composição automática, podemos utilizar o método Random Walk Linear. Esse método pode ser aludido a um tecladista que, tocando inicialmente uma tecla escolhida à sorte, avança, aleatoriamente, uma (ou mais) tecla (s) acima ou abaixo da atual. Esse deslocamento, é chamado de passo. A figura 16 apresenta o algoritmo Random Walk na linguagem procedural BASIC [WIP 89], com simples geração numérica. Cada passo tem o valor 1, correspondendo a 1 semitom. A figura 17 apresenta uma adaptação do algoritmo Random Walk Linear para a linguagem visual Max, com geração sonora por MIDI. Podemos observar que, na versão da linguagem visual, o algoritmo torna-se mais inteligível. Logo, o grau de dificuldade para seu entendimento é menor. Vários algoritmos para composição auxiliada pó r computador, apresentados na linguagem procedural em [WIP 89], foram adaptados para a versão visual, com o objetivo de servirem de exemplos no MEPSOM.

```
        LASTLOC=7
        :
        RWL:
        D=1
        U=RND
        IF U<0.5 THEN D=-1
        S=LASTLOC+D
        IF S>15 THEN S=14
        IF S<1 THEN S=2
        LASTLOC=S
        CURRLOC=S
        RETURN
```

Figura 16 – Algoritmo Random Walk na linguagem Basic

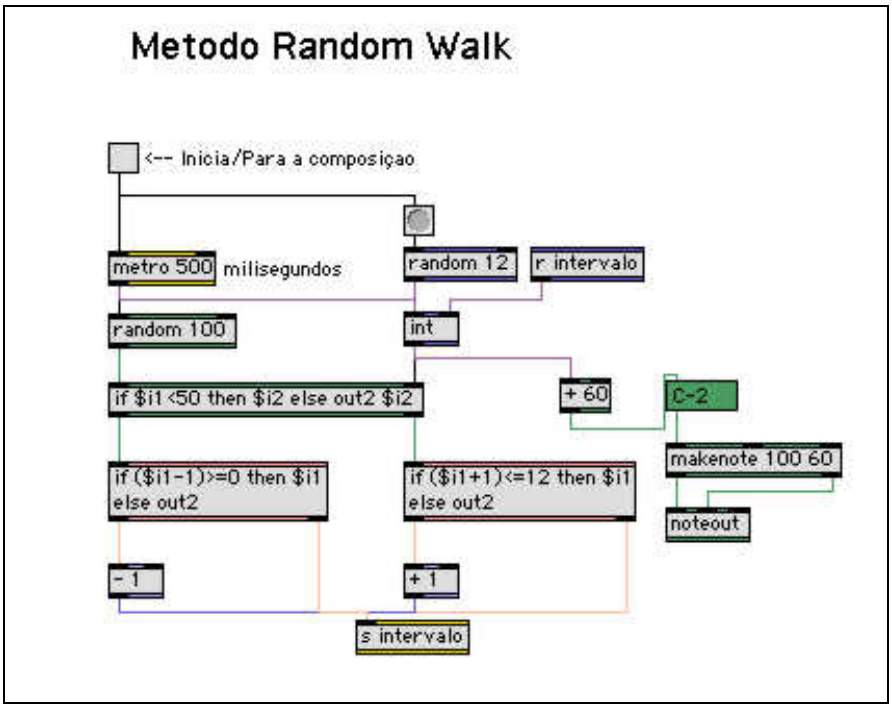


Figura 17 – Algoritmo Random Walk adaptado para a linguagem Max

Algoritmos mais complexos que o Random Walk, constituídos por fórmulas matemáticas, são de difícil compreensão à primeira vista. Contudo, podemos diminuir o grau de complexidade de algoritmos procedurais transformando-os em fluxogramas, que podem ser escritos e documentados diretamente em linguagens visuais com o objetivo de resultar numa maior clareza para o músico.

Outros exemplos comparativos entre algoritmos programados em diferentes linguagens de uso comum podem ser verificados nas figuras 18, 19, 20. O algoritmo embaralhar notas foi programado em Basic [WIP 89], Pascal e C, como segue, para exemplificar a programação através de linhas de comando.

```
DIM notas(1 TO 12) 'Cria a tabela de 12 notas
FOR INDICEDATABELA = 1 TO 12 'Percorre as posições da tabela...
 notas(INDICEDATABELA) = INDICEDATABELA + 59 '...armazenando o valor de cada notas
NEXT INDICEDATABELA 'Passa para o próximo valor da tabela

INICIO:
PRINT "Escolha uma operação: Parar Tocar Embaralhar Reordenar" 'Apresenta o menu
INPUT "", ESCOLHA$ 'Lê a opção escolhida
IF ESCOLHA$ = "Parar" THEN GOSUB SILENCIAR 'Caso Parar, vá para Silenciar
IF ESCOLHA$ = "Tocar" THEN GOSUB SOAR 'Caso Tocar, vá para Soar
IF ESCOLHA$ = "Embaralhar" THEN 'Caso Embaralhar...
 FOR INDICEDATABELA = 1 TO 12 STEP 1'...percorre a tabela...
  POSICAOSORTEADA = (INT(11 * RND)) + 1'...sorteia um posição da tabela...
  AUXILIAR = notas(INDICEDATABELA) '...guarda o valor da posição atual...
  notas(INDICEDATABELA) = notas(POSICAOSORTEADA) '...põe o valor da posição sorteada na posição atual...
  notas(POSICAOSORTEADA) = AUXILIAR '...efetua a troca, colocando o valor da posição atual na posição sorteada...
 NEXT INDICEDATABELA '...passa para o próximo valor da tabela
END IF
IF ESCOLHA$ = "Reordenar" THEN GOSUB Reordenar 'Caso Reordenar, vá para Reordenar
FOR INDICEDATABELA = 1 TO 12 'Percorre a tabela...
 PRINT notas(INDICEDATABELA) '...apresentando as notas em ordem
NEXT INDICEDATABELA
GOTO INICIO 'Recomeça o programa, reapresentando o menu de opções

SILENCIAR: 'Declaração da subrotina que cancela a operação em andamento
RETURN

SOAR: 'Subrotina que faz soar as notas da tabela
 DURACAO = 60
 PAUSA = 500
 FOR INDICEDATABELA = 1 TO 12 STEP 1 'Percorre a tabela
  SOUND 440 / (2 ^ ((69 - notas(INDICEDATABELA)) / 12)), DURACAO / 1000 * 18.2 'Faz soar a nota atual, com a sua duração
  SOUND 0, PAUSA / 1000 * 18.2 'Espera a pausa especificada entre as notas
 NEXT INDICEDATABELA
RETURN

Reordenar: 'Subrotina para reordenar a tabela
FOR VEZES = 11 TO 1 STEP -1 'Percorre as porções não ordenadas da tabela
  FOR INDICEDATABELA = 1 TO VEZES STEP 1 'Percorre toda a porção não ordenada
   IF notas(INDICEDATABELA) > notas(INDICEDATABELA + 1) THEN 'Caso o valor atual seja maior que o seguinte...
    AUXILIAR = notas(INDICEDATABELA) '...guarda o valor da posição atual...
    notas(INDICEDATABELA) = notas(INDICEDATABELA + 1) '...armazena o valor da próxima posição na posição atual...
    notas(INDICEDATABELA + 1) = AUXILIAR '...termina a troca, armazenando o valor da posição atual na próxima posição
   END IF '...senão, esse par de valores já se encontra em ordem crescente
  NEXT INDICEDATABELA
 NEXT VEZES
RETURN
```

Figura 18 - Embaralhador codificado em Basic (Microsoft QuickBasic 4.5)

```
Program Embaralhador_de_Notas;
Uses crt;
Const Num_Notas=12; {Constante para o número de notas}
Type Nota=0..127; {Define o tipo nota como um intervalo}
Type Tabela_de_Notas=Array [1..Num_notas] of Nota; {Define uma tabela 12 notas}

Procedure Silenciar; {Subrotina para parar uma operação em andamento}
Begin
End;

Procedure Soar(Tabela:Tabela_de_Notas;Duracao,Pausa:Integer); {Subrotina para tocar a tabela de notas, especificando
a duração e a pausa entre elas}
Var Indice_da_Tabela: 1..Num_Notas; {Variavel para percorrer a tabela}
Begin
 For Indice_da_Tabela:=1 to Num_Notas do {Efetua um loop a fim de percorrer a tabela}
 Begin
  Sound(Round(440/Exp(Ln(2)*((69-Tabela[Indice_da_Tabela])/12)))); {Faz soar a nota da posição atual}
  Delay(Duracao); {Espera até que a duração da nota tenha sido concuída}
  NoSound; {Desliga a nota}
  Delay(Pausa); {Espera a pausa especificada}
 End;
End;

Procedure Reordenar(Var Tabela:Tabela_de_Notas); {Subrotina para reordenamento da tabela}
Var Auxiliar: Nota; {Variavel auxiliar para a operação de troca}
   Vezes,Indice_da_Tabela: Integer; {Variaveis controladoras de loop}
Begin
 For Vezes:=Num_Notas-1 downto 1 do {Especifica a porção da tabela ainda não ordenada}
  For Indice_da_Tabela:=1 to Vezes do {Percorre a porção não ordenada da tabela}
   If (Tabela[Indice_da_Tabela]>Tabela[Indice_da_Tabela+1]) Then {Caso o valor atual seja maior que o seguinte...}
    Begin
     Auxiliar:=Tabela[Indice_da_Tabela]; {...então armazena o valor da posição atual...}
     Tabela[Indice_da_Tabela]:=Tabela[Indice_da_Tabela+1]; {...armazena o valor da posição seguinte na posição
atual...}
     Tabela[Indice_da_Tabela+1]:=Auxiliar; {... e armazena na posição posterior o valor da atual}
    End; {...senão o par já está em ordem crescente}
End;

Var Notas: Tabela_de_Notas; {Tabela das notas}
Indice_da_Tabela: 1..Num_Notas; {Indice da tabela}
Escolha: String; {Nome da operação escolhida}
Auxiliar,Posicao_Sorteada: Nota; {Variaveis auxiliares para o embaralhamento}
Begin
 For Indice_da_Tabela:=1 to Num_Notas do {Percorre a lista...}
  Notas[Indice_da_Tabela]:=Indice_da_Tabela+59; {...armazenando a nota}
 While True do {Loop que permite reapresentar o menu ao usuário}
 Begin
  Writeln('Escolha uma operação: Parar Tocar Embaralhar Reordenar'); {Apresenta as opções do menu}
  Readln(Escolha); {Faz a leitura da operação escolhida}
  If Escolha='Parar' Then Silenciar Else {Caso Parar, vá para Silenciar}
  If Escolha='Tocar' Then Soar(Notas,60,500) Else {Caso Tocar, vá para Soar}
  If Escolha='Embaralhar' Then {Caso Embaralhar...}
  For Indice_da_Tabela:=1 to Num_Notas do {...percorre toda a tabela...}
   Begin
    Posicao_Sorteada:=Random(11)+1; {...sorteia um posição de troca...}
    Auxiliar:=Notas[Indice_da_Tabela]; {...armazena o valor da posição atual...}
    Notas[Indice_da_Tabela]:=Notas[Posicao_Sorteada]; {...põe o valor da posição sorteada na posição atual...}
    Notas[Posicao_Sorteada]:=Auxiliar; {...troca as posições, armazenando o valor da posição atual na posição sorteada}
   End
  Else
  If Escolha='Reordenar' Then Reordenar(Notas); {Caso Reordenar, vá para Reordenar}
   For Indice_da_Tabela:=1 to Num_Notas do
    Writeln(Notas[Indice_da_Tabela]); {Percorre a lista a fim de apresentar a ordem da tabela de notas}
 End;
End.
```

Figura 19 - Embaralhador codificado em Pascal (Borland Pascal 7.0)

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<pc.h>

void Silencia(void); //Função que paralisa a operação em andamento
void Soar(unsigned char *Tabela,unsigned int Duracao,unsigned int Pausa); //Função que faz soar as notas da tabela, especificando a duração e a pausa entre elas
void Reordenar(unsigned char *Tabela); //Função para reordenamento da tabela
int main(void)
{
 unsigned char Notas[12],Indice_da_Tabela,Auxiliar,Posicao_Sorteada; //Tabela de notas,índice da tabela, e variaveis auxiliares para o embaralhamento
 char Escolha[11]; //Armazena o nome da operação escolhida
 for(Indice_da_Tabela=0;Indice_da_Tabela<12;Indice_da_Tabela++) //Percorre a tabela...
  Notas[Indice_da_Tabela]=Indice_da_Tabela+60; //...armazenando as notas
 for(;;) //Loop que permite reapresentar o menu de opções ao usuário
 {
  printf("Escolha uma operação: Parar Tocar Embaralhar Reordenar\n"); //Apresenta o menu
  scanf("%s",Escolha); //Faz a leitura da operação desejada
  if (strcmp(Escolha,"Parar")==0) Silencia(); else //Caso Parar, vá para Silenciar
  if (strcmp(Escolha,"Tocar")==0) Soar(Notas,60,500); else //Caso Tocar, vá para Soar
  if (strcmp(Escolha,"Embaralhar")==0) //Caso Embaralhar...
   for(Indice_da_Tabela=0;Indice_da_Tabela<12;Indice_da_Tabela++) //...percorre a lista...
   {
    Posicao_Sorteada=(random() % 12); //...sorteia uma posição da tabela...
    Auxiliar=Notas[Indice_da_Tabela]; //...guarda o valor da posição atual...
    Notas[Indice_da_Tabela]=Notas[Posicao_Sorteada]; //...põe o valor da posição sorteada na posição atual...
    Notas[Posicao_Sorteada]=Auxiliar; //...e efetua a troca, armazenando na posição sorteada o valor da posição atual
   }
  else
  if  (strcmp(Escolha,"Reordenar")==0) Reordenar(Notas); //Caso Reordenar, vá para Reordenar
  for(Indice_da_Tabela=0;Indice_da_Tabela<12;Indice_da_Tabela++) //Percorre a tabela...
   printf("%d\n",Notas[Indice_da_Tabela]); //...apresentando as notas em ordem
 }
 return 0; //Termina o programa
}
void Silencia(void)
{
}
void Soar(unsigned char *Tabela,unsigned int Duracao,unsigned int Pausa)
{
 unsigned char Indice_da_Tabela; //Armazena a posição atual da tabela

 for(Indice_da_Tabela=0;Indice_da_Tabela<12;Indice_da_Tabela++) //Percorre a tabela
 {
  sound((signed int)(440.0/pow(2.0,(69.0-Tabela[Indice_da_Tabela])/12.0))); //Faz soar a nota
  delay(Duracao); //Espera a duração informada...
  nosound(); //...e desliga a nota
  delay(Pausa); //Espera a pausa entre as notas antes de passar para a próxima
 }
}
void Reordenar(unsigned char *Tabela)
{
 unsigned char Auxiliar; //Variavel auxiliar para o ordenamento
 int Vezes,Indice_da_Tabela; //Contadores para o ordenamento
 for(Vezes=10;Vezes>0;Vezes--) //Informa a parte ainda não ordenada da tabela
  for(Indice_da_Tabela=0;Indice_da_Tabela<=Vezes;Indice_da_Tabela++) //Percorre a parte ainda não ordenada
   if (Tabela[Indice_da_Tabela]>Tabela[Indice_da_Tabela+1]) //Caso o valor atula seja maior que o seguinte
   {
    Auxiliar=Tabela[Indice_da_Tabela]; //Salva o valor da posição atual da tabela
    Tabela[Indice_da_Tabela]=Tabela[Indice_da_Tabela+1]; //Armazena na posição atual o valor da posição sorteada
    Tabela[Indice_da_Tabela+1]=Auxiliar; //termina a troca, armazenando na próxima posição o valor atual da tabela
   }
 }  //senão este par já está em ordem crescente
```

Figura 20 - Embaralhador codificado em C (DJ Delories' DJGPP 2.03)

O mesmo algoritmo é apresentado na figura 21, através de um exemplo desenvolvido para o MEPSOM, utilizando linguagem visual. Podemos comparar a diferença na forma da apresentação da solução do problema "embaralhar notas".

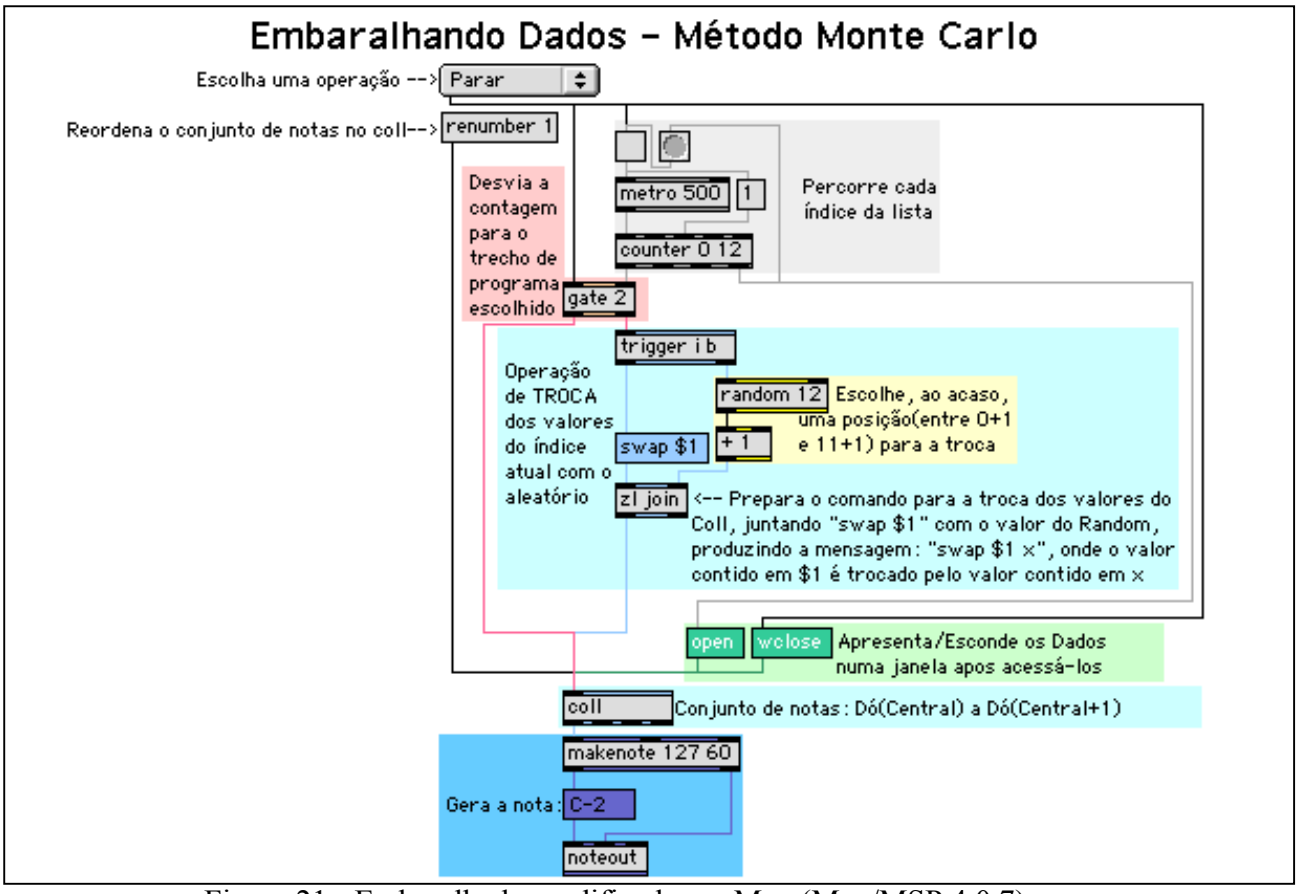


Figura 21 - Embaralhador codificado em Max (Max/MSP 4.0.7)

3.1.5 Linguagens Dedicadas a Aplicações Musicais

Outra abordagem para a programação de computadores para a música é o uso das linguagens dedicadas a essa função as quais, são na sua grande maioria, direcionadas a aplicações de síntese sonora e visam a criação e alteração do som por computador. Como exemplo de linguagem dedicada podemos citar o *csound* desenvolvida, por B. Vercoe e R.Karstnes. A obra de [MIR 98] apresenta várias linguagens dedicadas à síntese sonora, incluindo tutoriais [THO 97].

Linguagens baseadas em processamento de listas como a Nyquist podem ser utilizadas para implementação de algoritmos musicais para composição. Na figura 22, é apresentado o mesmo exemplo do algoritmo "embaralhar notas" programado em Nyquist.

```
	(setq notas nil) ;Cria a lista para as notas

	(setq indice-da-tabela 2) ;Aponta para a segunda posição da lista
	(push 60 notas) ;Armazena a nota/tecla 60 na primeira posição da lista
	(loop ;Efetua um loop a fim de percorrer toda a escala temperada
		(setq notas (append notas (list (+ indice-da-tabela 59)))) ;Calcula e armazena o valor da nota no início da lista juntamente com os valores já existentes
		(setq indice-da-tabela (+ indice-da-tabela 1)) ;Passa para a próxima nota
		(when (> indice-da-tabela 12) (return)) ;Verifica quando o loop deve terminar: depois de percorrer toda a escala musical
	)
	(defun Parar () () notas) ;Função para parar uma operação em andamento

	(defun Tocar (tabela duracao pausa) ;Função que toca a lista, especificando a duração das notas, e a pausa entre elas
		(setq indice-da-tabela 1) ;Aponta para o início da lista
		(loop ;Efetua um loop para percorrer a lista
			(setq auxiliar (car tabela)) ;Lê a primeira posição da lista
			(pop tabela) ;Retira a primeira posição da lista
			(play (osc auxiliar duracao)) ;Faz soar a nota com a duração especificada
			(play (osc 0 pausa)) ;Efetua a pausa entre as notas
		(setq tabela (append tabela (list auxiliar))) ;Repõe o valor, no fim da lista
			(setq indice-da-tabela (+ indice-da-tabela 1)) ;Passa para o próximo valor da lista
			(when (> indice-da-tabela 12) (return)) ;Verifica o fim da leitura da lista
		)
	    notas
	)
	(defun guarda (val pos tab) ;Função para armazenamento de um valor numa lista
		(setq cont 1) ;Aponta para o início da lista
		(loop ;Efetua um loop para percorrer a lista
			(setq aux (car tab)) ;Lê a primeira posição da lista
			(pop tab) ;Retira a primeira posição da lista
			(if (= cont pos) ;Se a posição atual for a escolhida...
				(setq tab (append tab (list val))) ;...armazena o valor na lista
				(setq tab (append tab (list aux))) ;...mas mantém na lista o antigo valor
			)
			(setq cont (+ cont 1)) ;Passa para o próximo valor da lista
			(when (> cont 12) (return tab)) ;Verifica o fim da leitura da lista
		)
	)
	(defun valor (pos tab) ;Função para leitura de uma posição de uma lista
		(setq cont 1) ;Aponta para o início da lista
		(loop ;Efetua um loop para percorrer a lista
			(setq aux (car tab)) ;Lê a primeira posição da lista
			(pop tab) ;Retira a primeira posição da lista
			(if (= cont pos) (setq res aux)) ;Se a posição atual for a escolhida, então lê o valor
			(setq tab (append tab (list aux))) ;Repõe o valor na lista
			(setq cont (+ cont 1)) ;Passa para o próximo valor da lista
			(when (> cont 12) (return res)) ;Verifica o fim da leitura da lista
		)
	)
	(defun Embaralhar (tabela) ;Função para o embaralhamento da lista
		(setq indice-da-tabela 1) ;Aponta para o início da lista
		(loop ;Efetua um loop para percorrer a lista
			(setq posicao-sorteada (+ (random 12) 1)) ;Sorteia um posição da lista
			;Efetua a troca de duas posições da lista
			(setq auxiliar (valor indice-da-tabela tabela)) ;Guarda o valor da posição atual
			(setq tabela (guarda (valor posicao-sorteada tabela) indice-da-tabela tabela)) ;Armazena o valor da posição sorteada na posição atual
			(setq tabela (guarda auxiliar posicao-sorteada tabela)) ;Guarda o valor da posição atual na posição sorteada
			(setq indice-da-tabela (+ indice-da-tabela 1)) ;Passa para o próximo valor da lista
			(when (> indice-da-tabela 12) (return tabela)) ;Verifica o fim da leitura da lista
		)
	    notas
	)
	(defun Reordenar (tabela) (sort tabela #'<) notas) ;Função para o reordenamento dos valores de uma tabela
	(print "Escolha uma opera‡Æo: Parar Tocar Embaralhar Reordenar") ;Apresenta o menu de opções ao usuário
```

Figura 22 - Embaralhador codificado em Nyquist/Lisp(David Betz's XLISP 2.0)

Comparando o exemplo em Nyquist, da figura 22, com o exemplo em Max, da figura 21, é possível notar grandes diferenças na apresentação do programa. Apesar do

Nyquist ser uma linguagem dedicada à aplicações musicais, ela não dispõe de uma interface visual. Isso cria uma dificuldade para o músico durante o processo de entendimento do exemplo. Como podemos perceber, o programa em Max é apresentado na forma de fluxograma enquanto que o programa em Nyquist necessita de um entendimento mais aprofundado sobre o funcionamento de listas e a lógica envolvida nesse processo O processamento com listas pode tornar-se de difícil compreensão para músicos iniciantes em programação.

O Max é um dos melhores exemplos de aplicações *user-friendly* dedicadas para a programação visual de aplicativos musicais. Com o aumento do desempenho e da capacidade da memória dos computadores, foi possível desenvolver programas que integram MIDI e DSP (*Digital Signal Processing*). Programas de computador que antes somente apresentavam tratamento de dados MIDI atualmente já incorporam audiodigital. Portanto, um método de ensinar programação sônica de computadores para músicos não se restringe apenas ao conjunto de programas que podem ser elaborados com base no protocolo MIDI, mas também àqueles que efetuam processamento de sinal de áudio. O MAX dispõe de objetos MIDI e de um conjunto de objetos de DSP intitulado MSP [DOB 98]. Logo, o Max é uma linguagem possível de ser utilizada em um método de programação sônica de computadores para músicos. Como o MEPSOM baseia-se no ensino através de exemplos, é necessário que as linguagens em que os exemplos são programados sejam o mais interativas, visuais e próximas possíveis à linguagem do músico usuário.

Linguagens como *csound* e *cmusic,* utilizadas para a síntese sonora, não possuem a mesma facilidade de utilização que a linguagem MaxMSP. Enquanto o *csound* e o *cmusic* utilizam o ambiente de linha de comando, o MSP apresenta uma interface gráfica com objetos que implementam os principais elementos para a síntese sonora. No MSP, ao invés de escrever linhas de comando, objetos gráficos são conectados para formar um programa completo. Além disso, a disponibilização de uma linguagem MIDI e outra para síntese, no mesmo ambiente, através de controle gráfico, possibilita a apresentação de exemplos ao aluno de uma maneira clara e interativa.

Com base nessas premissas, optamos em escolher duas linguagens para servirem de meio de implementação do MEPSOM:

HyperCard – com a biblioteca (pilha) *HyperMIDI* embutida.

Max – com a biblioteca (objetos) MSP embutida.

3.2 Escolha do suporte tecnológico para o MEPSOM

Tendo-se a preocupação de escolher a melhor solução tecnológica para proporcionar um aprendizado de maneira rápida e fácil, procurou-se explorar as características das linguagens visuais por estarem mais próximas da realidade do músico atual, conforme relatado nos itens 4.3 e 4.4. Para que seja possível implementar exemplos e exercícios com o objetivo de atender ao escopo do Nível de Programação aplicada do MEPSOM, este trabalho utilizou duas linguagens:

1)Linguagem *HyperCard/HyperMIDI* - linguagem genérica "amigável" para autoria, com rotinas MIDI embutidas, possibilitando a programação visual com *scripts* de comandos de linha.

2)Linguagem MaxMsp - linguagem visual de quarta geração inspirada no paradigma de orientação a objetos, dedicada à programação MIDI, processamento DSP e síntese sonora.

Apesar da existência de outras linguagens, como o OpenMusic[14], optou-se em utilizar principalmente o *MaxMsp* por ser um dos ambientes de programação para música mais empregados atualmente.

As linguagens *Max* e *HyperCard/HyperMIDI* possuem muitas diferenças. Enquanto a Linguagem Max é inspirada no paradigma de Orientação à Objetos, a linguagem *HyperCard/HyperMIDI* é declarativa. Ambas são "amigáveis" e disponibilizam muitos recursos para programação visual. Foram escolhidas para apresentar ao músico diferentes abordagens e características de programação. Ao aprender duas linguagens o músico terá maiores possibilidades de utilizar a que mais se adequar às suas necessidades. Para determinadas aplicações poderá ser o MaxMsp, para outras o *HyperCard/HyperMIDI*.

Na Tabela 3 são apresentados alguns comandos/funções empregados no MEPSOM aproximando particularidades de ensino de programação entre as linguagens MAX e HyperCard.

[14] A Linguagem OpenMusic foi desenvolvida por Gèrard Assayag e Carlos Agon no IRCAM. Consiste num sistema para programação de música, podendo ser considerada tanto uma linguagem de programação quanto um software para composição musical. Possui uma interface visual para a linguagem Lisp, disponibilizando uma grande variedade de rotinas e estruturas abstratas representadas por ícones. Os músicos podem construir seus programas em OpenMusic, simplesmente selecionando ícones e arrastando-os para a área de trabalho e, de forma similar ao Max, ligando seus *inlets* e *outlets*. Maiores detalhes sobre o OpenMusic podem ser encontrados na obra de [MIR 01]. O OpenMusic não foi utilizado durante a fase de implementação do MEPSOM, pois não foi possível obter uma versão em tempo hábil.

Tabela 3 - Quadro comparativo entre os objetos e os comandos *Max/HyperCard* - programação convencional

Tipo de Comando/Função/Objeto	MAX	HYPERCARD
Declaração de Variáveis	Não possui	Global/local
Passagem de parâmetros	Inlet, outlet	
Atribuição	$1 em mensagens $i1 em objetos IF e expr	Put x into var
Condicional	If if encadeado	If if encadeado
Repetição	Metro Clocker tempo	Repeat x times Repeat while
Seleção e controle de fluxo	Gate Select Split Switch Change counter	If
Expressão Aritmética	expr	Put expressão into var
Armazenamento de dados	Table Coll Int	Field
Modularização	Através de patchers	Através de Cards
Interface	Slider Dial Table preset	Card Button Field
Randomização	random	random
Atraso	delay	delay
Listas	Pack Umpack	Só por implementação

Ambas as linguagens disponibilizam vários Comandos/Objetos para o tratamento de mensagens MIDI. Essa característica das linguagens é de grande valia para o músico programador, pois ele irá implementar seu aplicativo programando em alto nível . No entanto, existem diferenças marcantes entre as duas linguagens com relação à utilização dos comandos MIDI. A linguagem HyperCard possui uma outra linguagem embutida, que é o HyperMIDI e, por essa razão, todos os comandos iniciam com as letras “Hm” [RED 98]. Nessa linguagem os comandos necessitam especificar parâmetros que exigem uma compreensão, por parte do músico, em termos de alocação de memória e processamento de dados. Por outro lado, o Max foi projetado para MIDI. Portanto, certas operações de mais baixo nível que precisam ser realizadas no HyperMIDI pelo músico são transparentes no MAX.

Tabela 4 - Quadro comparativo entre os objetos Max e os comandos *HyperMIDI*- programação MIDI

Tipo de Comando /Objeto MIDI	MAX	HYPERMIDI
Abre o ambiente MIDI e determina quantidade de memória a ser alocada.	Automático	hmOpenMIDI
Finaliza o ambiente MIDI	Automático	hmCloseMIDI
Recebe mensagens MIDI	Midiin Notein Pgin Bendin Touchin Polyin Ctlin	hmReadMIDI
Envia Mensagens MIDI	Midiout Noteout Pgout Bendout Touchout Polyout Ctlout	hmWriteMIDI
Filtros de entrada/saída MIDI	Stripnote makenote	hmSetFilter
Modificadores de saída MIDI	makenote	hmSetChannel hmSetTranspose hmSetVelocity hmSetOffset
Construtores de mensagens MIDI	makenote	hmBuilder
Conversores de dados MIDI	Midiparse, midiformat	hmConvert
Seqüenciadores	Seq, mtr	Só por implementação

3.2.1 Características da linguagem HyperCard/HyperMIDI

A filosofia de programação visual em HyperCard mostrou ser mais facilmente entendida pelos alunos, pois assemelha-se muito à programação em HTML, ao implementar os conceitos de *HyperLink*, utilizados pelos browsers na internet. A grande maioria dos músicos que participaram das aulas já utilizavam internet e por essa razão identificaram-se com a interface de navegação formada por cartões e botões.

A figura 23 mostra as principais características da linguagem *HyperCard/HyperMIDI* que foram levadas em consideração para a escolha da linguagem, como uma das possíveis de serem utilizadas para a implementação do MEPSOM.

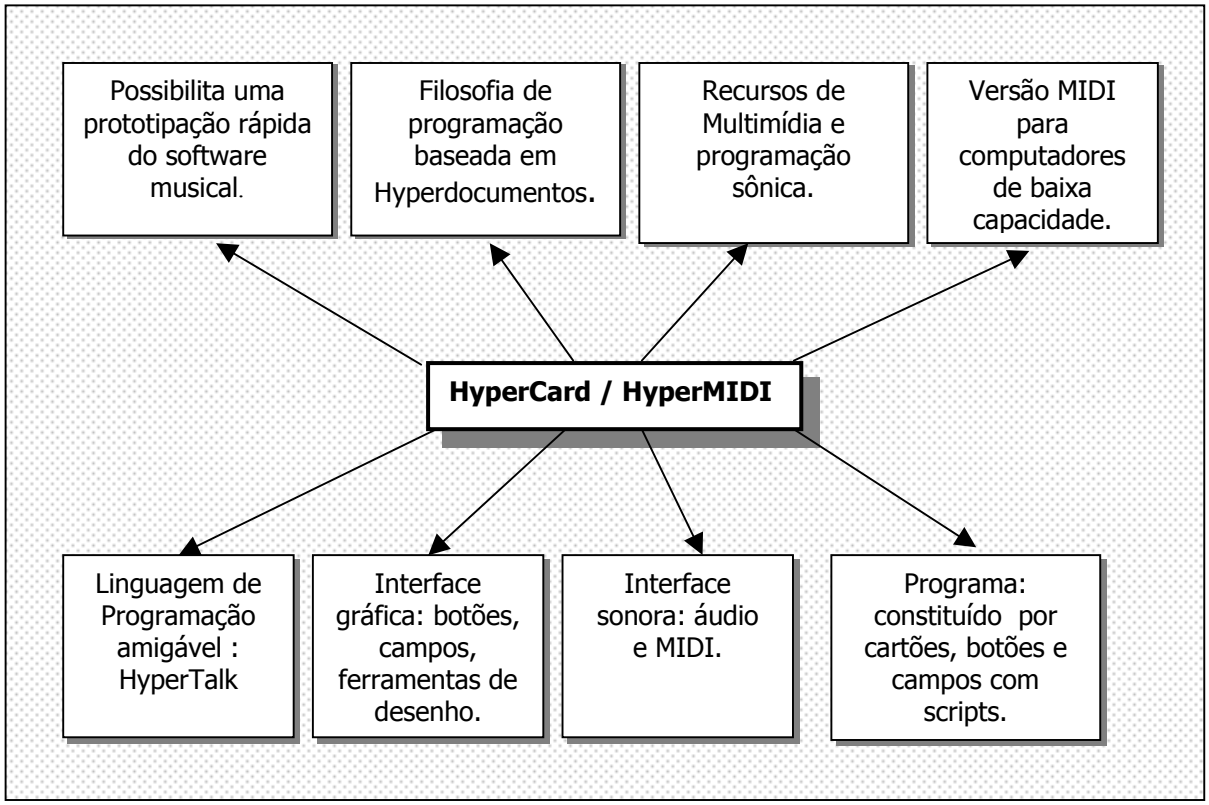


Figura 23 – Características da linguagem *HyperCard/HyperMIDI* importantes para a implementação do MEPSOM

3.2.2 Características da linguagem MaxMsp

A linguagem MaxMsp tem a grande vantagem de estar sendo continuamente atualizada com novas versões, a que foram incorporados novos objetos para processamento de sinais e síntese sonora inexistentes no HyperCard. Por isso, durante o desenvolvimento do MEPSOM para o conteúdo de síntese, está sendo utilizada a linguagem MaxMsp. A Figura 24 mostra as principais características dessa linguagem escolhida para a implementação do método.

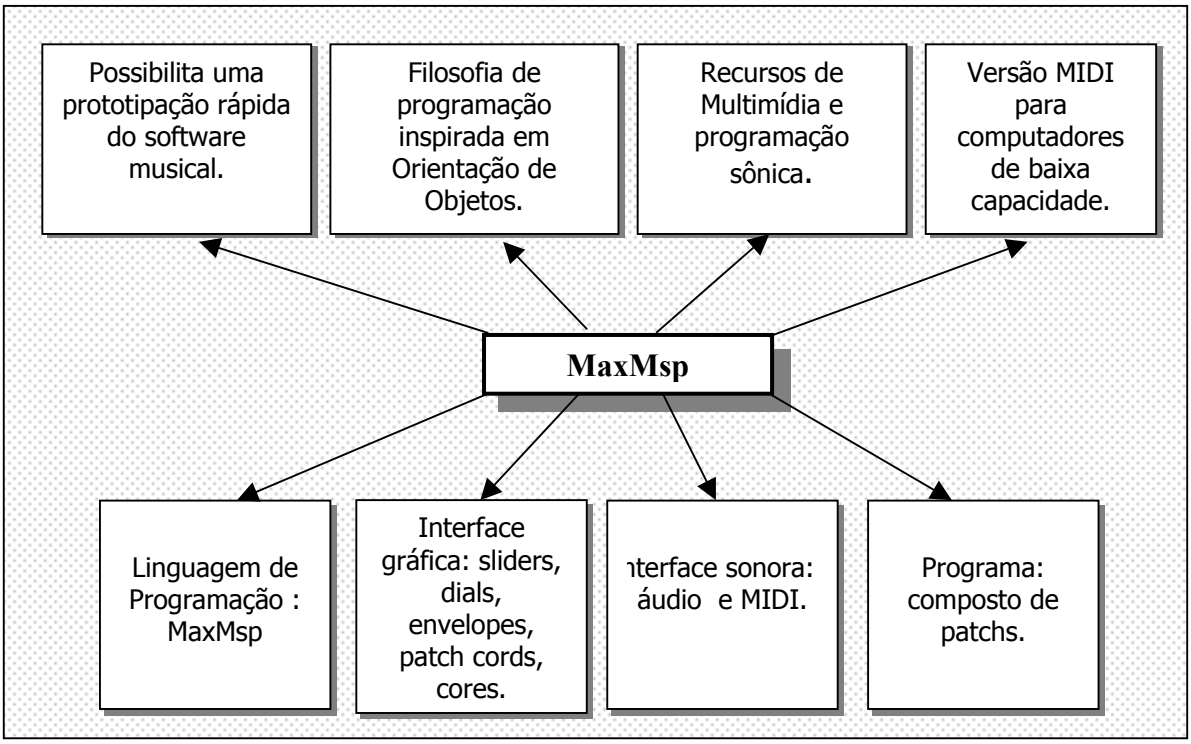


Figura 24 – Características do MaxMsp importantes para a implementação do MEPSOM

3.3 Metodologia empregada na criação dos programas do MEPSOM

3.3.1 Projeto Centrado no Músico

O Projeto Centrado no Músico é uma proposta de adaptação para o MEPSOM do Projeto Centrado no Usuário (PCU) [NOR 86] que, segundo [GOL 85], caracteriza-se por 3 aspectos principais:

- Processo cíclico de desenvolvimento
- Ênfase nos usuários e suas tarefas
- Avaliação empírica dos programas

O PCU é uma metodologia usualmente empregada no projeto da interface com o usuário (IU) de softwares. Entretanto, consideramos que a extensão proposta pode ser aplicada ao processo de desenvolvimento de software educacional como um todo (não só para a sua IU). Os próprios autores em [WIN 2000] defendem que "o projeto da IU de um software educacional se confunde muito com a concepção do software como um todo. Isso porque um dos aspectos mais importantes do software educacional é a interação do aprendiz com o sistema e o diálogo que é estabelecido entre as partes. Assim, a separação entre o que é a IU do software e o que é o software em si com freqüência não é fácil".

Conforme pode ser observado, na figura 25, adaptada de [WIN 2000], o produto final é fruto de uma contínua construção de programas musicais, que são avaliados em laboratório durante os cursos, alterados e novamente avaliados de forma cíclica, visando a um aperfeiçoamento dos mesmos.

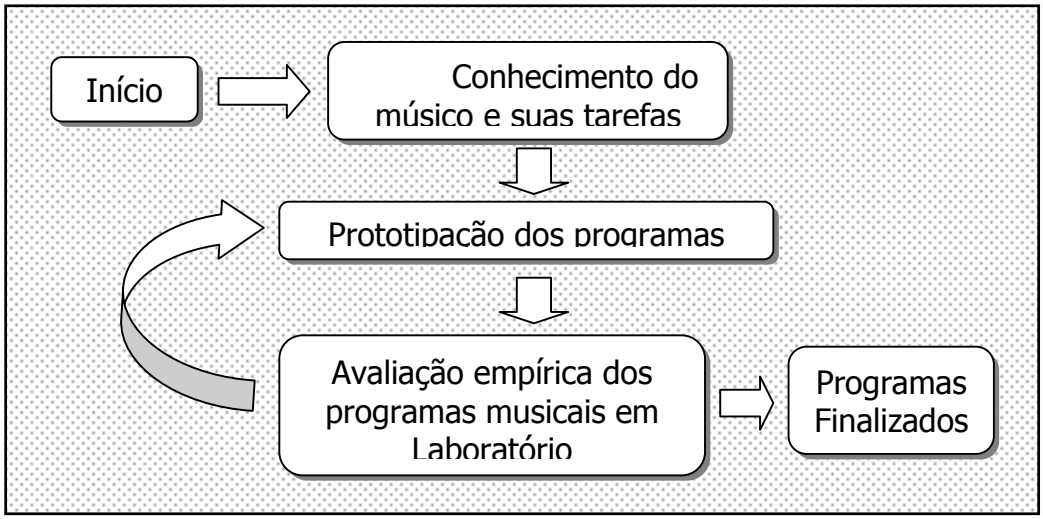


Figura 25 – Projeto de desenvolvimento dos programas do MEPSOM

3.3.1.1 Conhecer os músicos reais e suas tarefas

Consiste na criação de um perfil para detalhar todos os aspectos que se relacionam com o músico: computação musical, conhecimentos prévios, área de atuação na música, experiências anteriores com computação musical e outros programas, faixa etária e escolaridade. De posse de informações como essas, pode-se perceber como se dará o processo de aprendizagem do músico. Com a realização de cursos de Programação de Computadores para a Música está sendo possível avaliar e aprimorar o MEPSOM. A interação dos alunos com os programas resulta em questionamentos a respeito do conteúdo. Através de novas experiências do professor com alunos, novos exercícios são criados. Toda essa interação permite que seja traçado um perfil de usuário, conhecendo, portanto, a especificidade dos objetivos de cada um em aplicar a programação na confecção de programas. Nessa elaboração notam-se as diferentes soluções de cada aluno para o mesmo problema e a forma como desenvolvem o conhecimento através do MEPSOM.

3.3.1.2 Prototipação do MEPSOM

Nesta fase foi implementada uma versão simples de alguns dos programas do MEPSOM, para experimentos e possíveis soluções. Esses protótipos também serviram para dar ao aluno uma noção dos trabalhos desenvolvidos ao longo do curso. Utilizaram-se, nesta fase, protótipos funcionais feitos em *HyperCard/HyperMDI* e MAX.

3.3.1.3 Método de Avaliação do MEPSOM

A avaliação é considerada uma das mais importantes etapas do desenvolvimento, pois visa, sobretudo, a identificar problemas que possam comprometer a interação do usuário com o MEPSOM.

Nesta avaliação foi adotado o Método de *Survey* (ou Estudo de Levantamento) de Corte Transversal, que é caracterizado por uma única coleta de dados de uma "porção da população de interesse (...), na expectativa de que os indivíduos examinados

propiciem informação relativamente descritiva de uma população inteira" [CAS 92], (p.116).

Para a avaliação do MEPSOM, foram utilizados dois instrumentos de avaliação: um questionário aberto e um questionário fechado. [CAS 92] define questionário como "um conjunto de questões ou declarações impressas" que são respondidas de forma escrita (p.118).

No questionário aberto, os respondentes podem utilizar suas próprias palavras para as respostas (ibid. p.118). No questionário fechado, os itens são formados por categorias ou questões com opções de respostas [NEW 98], compostas por cinco alternativas. Os resultados podem ser verificados no capítulo 3 – Aplicação e Avaliação do MESPCM.

3.4 Programação do MEPSOM

O MEPSOM está sendo implementado de acordo com o projeto (figura 1) que divide o método em níveis e módulos.

As linguagens *MaxMsp* e *HyperCard/HyperMIDI*, conforme os motivos já mencionados, foram selecionadas para a inplementação dos exemplos e exercícios do MEPSOM. O critério de seleção adotado levou em conta as seguintes características das linguagens: objetos e rotinas para manipulação de áudio e MIDI, programação visual, linguagem de programação simples e amigável.

Conforme a figura 26, a linguagem HperCard/HyperMIDI foi utilizada na implementação de apenas dois módulos: Elementos Básicos de Programação e Educação Musical. A Linguagem MaxMsp foi empregada em quatro módulos, todos, menos no Módulo de Educação Musical. Logo o Módulo de Elementos Básicos de Programação foi implementado em duas linguagens.

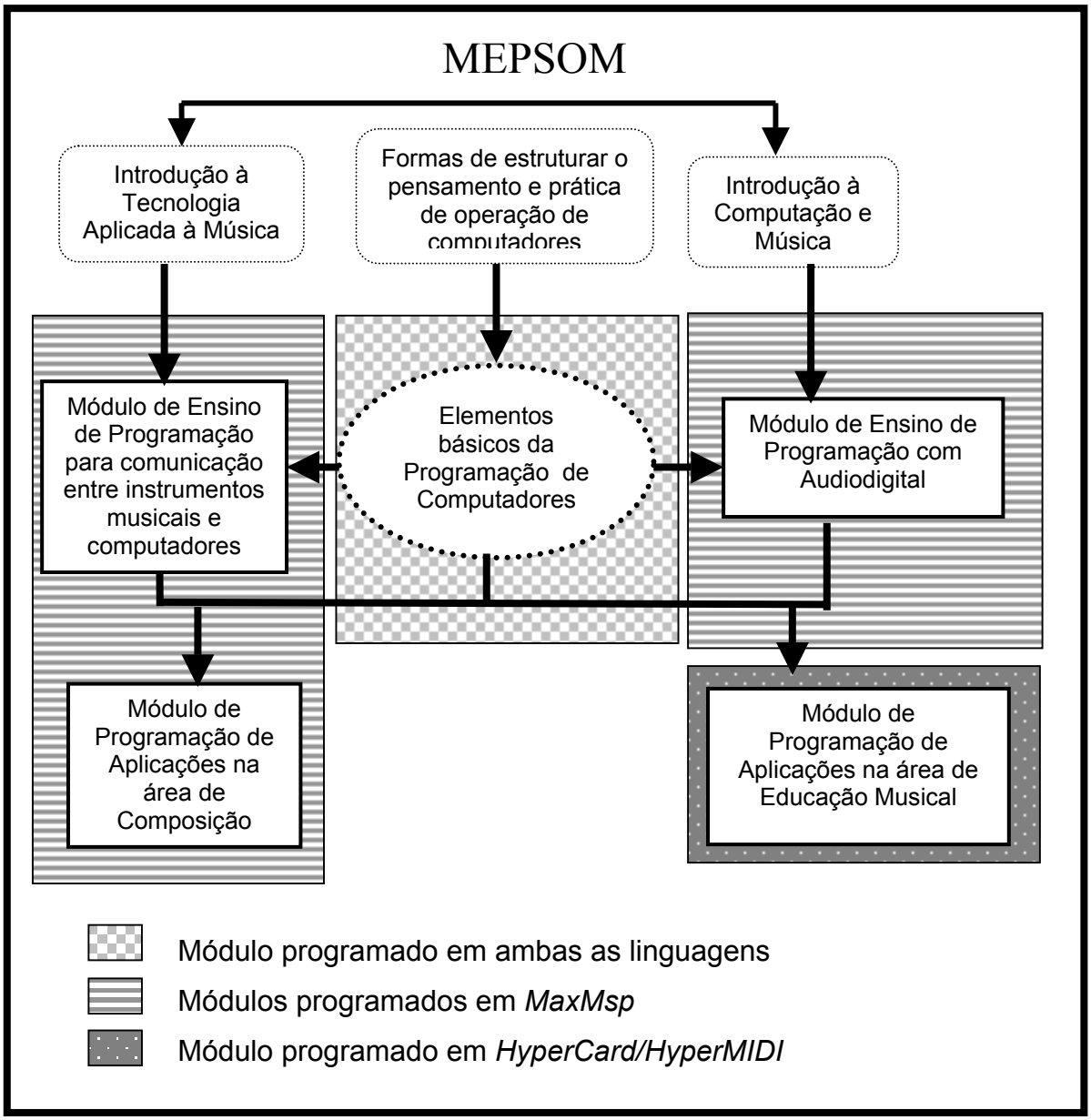


Figura 26 – Plano de distribuição de Módulos do MEPSOM para a programação nas linguagens MaxMsp e HyperMIDI

A tela inicial do MEPSOM, figura 27, foi criada para a abertura dos programas em Max. Após o estudante visualizar a tela inicial, dará início ao método e irá basear-se em exemplos para solucionar os exercícios.

Figura 27 – Tela inicial do MEPSOM

3.4.1 Implementação do Nível da Programação Preparatória

3.4.1.1 Implementação do Módulo de Ensino de Programação para comunicação entre instrumentos musicais e computadores

Este módulo do MEPSOM tem o objetivo de introduzir as principais funções de comunicação entre instrumentos musicais e computadores.

Para que seja possível a comunicação entre instrumentos musicais eletrônicos e computadores, é necessária a utilização de um protocolo de comunicação específico para a música. O padrão MIDI é o protocolo atual mais utilizado em aplicações musicais e, por essa razão, está presente em linguagens de programação, sendo implementado em *hardware* musical. Para a implementação desta Tese, foi utilizado o padrão MIDI conforme a tabela M.

Para o ensino de programação de comunicação de dados entre computadores e instrumentos musicais eletrônicos, foram escolhidos exemplos simples para geração e organização de alturas de notas e ritmos e o controle da troca de timbres de um instrumento.

Neste módulo foram elaborados programas que reúnem algumas das principais funções de troca de informações entre instrumentos eletrônicos e computadores. A implementação foi realizada em Max, devido à existência de objetos específicos para a troca de mensagens MIDI, conforme pode ser observado na tabela 4. Assim, podemos destacar os seguintes programas:

- Comunicação com o Instrumento MIDI – Note On e Note Off
- Execução de notas com diferentes intensidades e durações
- Programação de acordes
- Programação de canais MIDI
- Controle de sons de percussão
- Programação do envio e recebimento de Mensagens de Mudança de Programa
- Programação do envio e recebimento de Mensagens de Mudança de Controle

Um exemplo de *patch* que explica instruções para a programação da comunicação MIDI pode ser visualizado na figura 28.

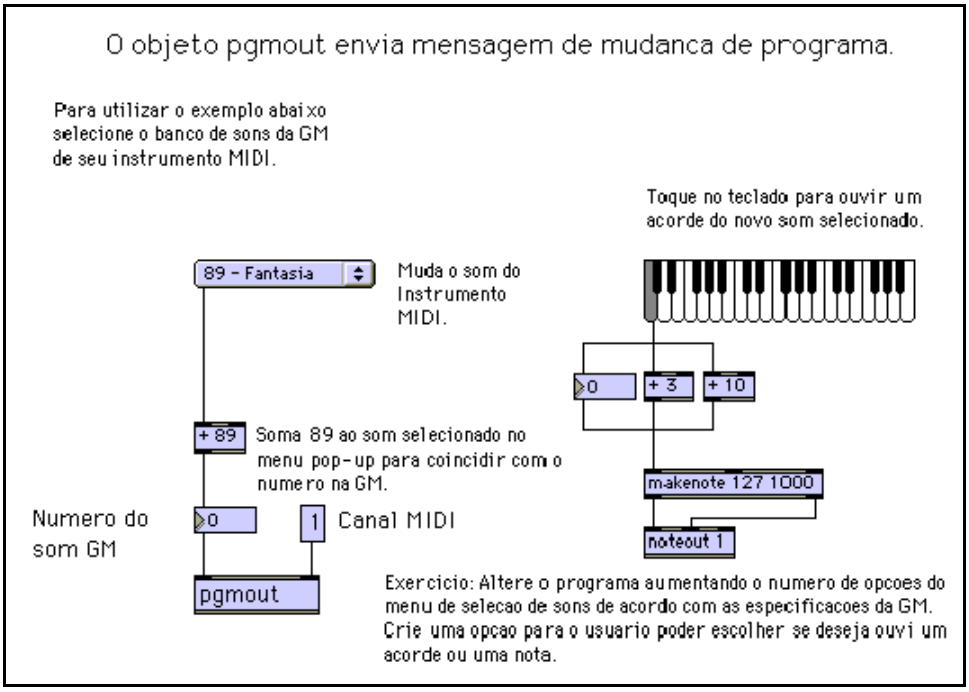


Figura 28 - Objetos de comunicação MIDI

Apesar de o ensino de programação do controle de sintetizadores externos ser parte integrante do tema desta pesquisa, a geração sonora de uma fonte externa pode ser substituída por um ou mais instrumentos construídos em Msp. Portanto, o resultado sonoro de um programa pode ser enviado para uma fonte externa de som, ou executado internamente no computador, através de síntese. A relevância no uso de uma técnica ou outra na emissão dos sons poderá ser uma decisão técnica ou estética.

3.4.1.2 Implementação do Módulo de Elementos básicos da Programação de Computadores

De acordo com o Capítulo1, os elementos básicos da programação de computadores são agrupados nas seguintes categorias: controle de fluxo do programa, estruturas de dados, passagem de parâmetros e encapsulamento de rotinas. Essas categorias são utilizadas pelos demais módulos do MEPSOM, pois são necessárias em todos os exemplos e exercícios. Foram escolhidos os seguintes elementos básicos de programação: operadores lógicos e matemáticos, controle de fluxo, atribuição de valores a variáveis, estrutura de dados, encapsulamento de rotinas e passagem de parâmetros. O MEPSOM foi programado em duas linguagens. A mais empregada foi o MaxMsp. Exemplos do MEPSOM desse módulo são:

- Operadores aritméticos: adicionar intervalos.
- Seleção: Seleção de notas, Seleção de opções, Escolha de notas, Seleção de limites numéricos.
- Condicional: Seleção de notas ou intensidades, programação de decisões, programação de decisões encadeadas.

- Repetição: Programando um metrônomo, seleção automática de valores de notas, execução automática de uma seqüência de acordes, contagem automática.
- Atribuição de valores: armazenamento de valores de altura.
- Tabelas: Armazenando e recuperando informações musicais em tabelas.
- Listas: Agrupar elementos numa lista, separar elementos de uma lista.
- Encapsulamento e passagem de parâmetros: Geração de alturas randomicamente.

A grande maioria das linguagens de programação para a música dispõe desses elementos de programação. Apesar de o MEPSOM estar utilizando as linguagens MaxMsp e *HyperCard/HyperMIDI*, ele independe de linguagens de programação para a implementação de seus exemplos e exercícios. Significa que o MEPSOM pode ser desenvolvido, em tese, para qualquer linguagem de programação apropriada ao desenvolvimento de aplicações musicais. Entretanto, conforme já explicado neste trabalho, o MEPSOM é direcionado para a programação de linguagens visuais e amigáveis.

Um exemplo de *patch* para ensino das instrução de seleção pode ser visualizado na figura 29. Neste exemplo, o objeto *gate* é utilizado para selecionar qual nota será tocada. As notas estão agrupadas em quatro conjuntos. O usuário seleciona qual a nota e qual o conjunto de notas, depois clica no *bang* para que a mensagem *note on* seja enviada.

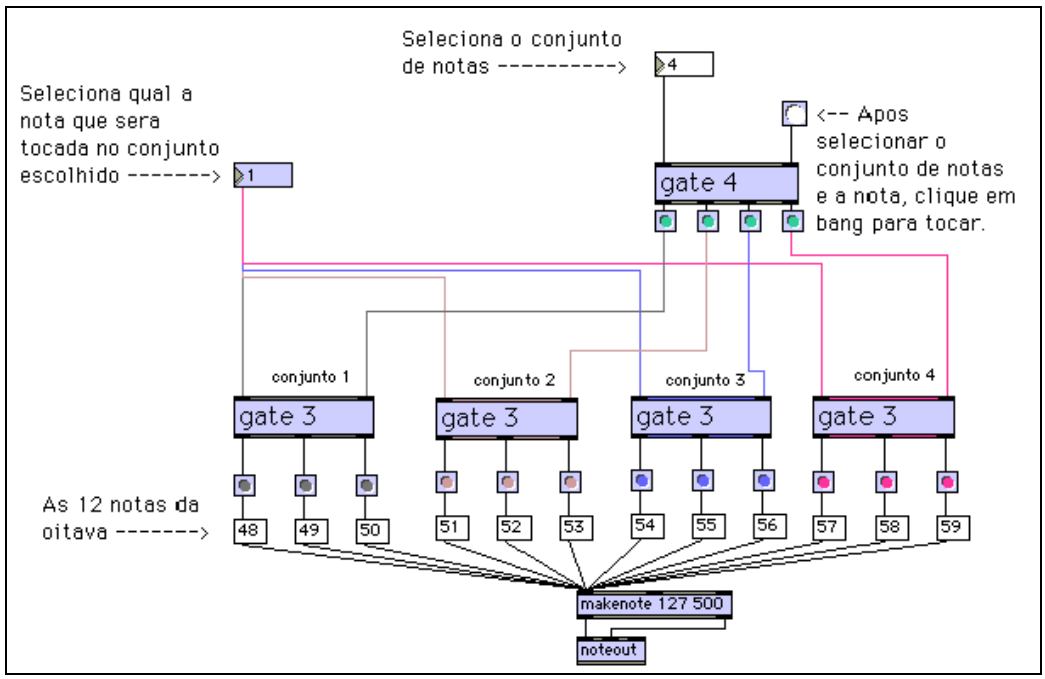


Figura 29 – *Patch* Educacional para o ensino da instrução de seleção em programação visual

O encapsulamento de rotinas é exemplificado no MEPSOM de forma um pouco diferente em cada linguagem. No MaxMsp, o encapsulamento é utilizado através do emprego do objeto *patch;* e em *HyperCard/HyperMIDI*, utilizando botões e programação de *scripts*. A figura 30 mostra um exemplo de programação utilizando encapsulamento de rotinas e passagem de parâmetros. O programa permite a emissão apenas de notas que possuam a numeração MIDI par. Isso é realizado através de uma

escolha aleatória de números pares, utilizando o objeto random. Neste tipo de programação modular, o músico aprende técnicas que irão ser muito úteis na construção de programas mais extensos. Também percebe a necessidade de dividir um problema para solucioná-lo em partes. O exercício, logo abaixo do exemplo, sugere que o músico modifique o algoritmo para tocar apenas os valores ímpares da numeração MIDI e agrupe-os sob a forma de acordes. Esse tipo de atividade desafia o aluno a entender o algoritmo, proporcionando uma compreensão total do programa e suas partes.

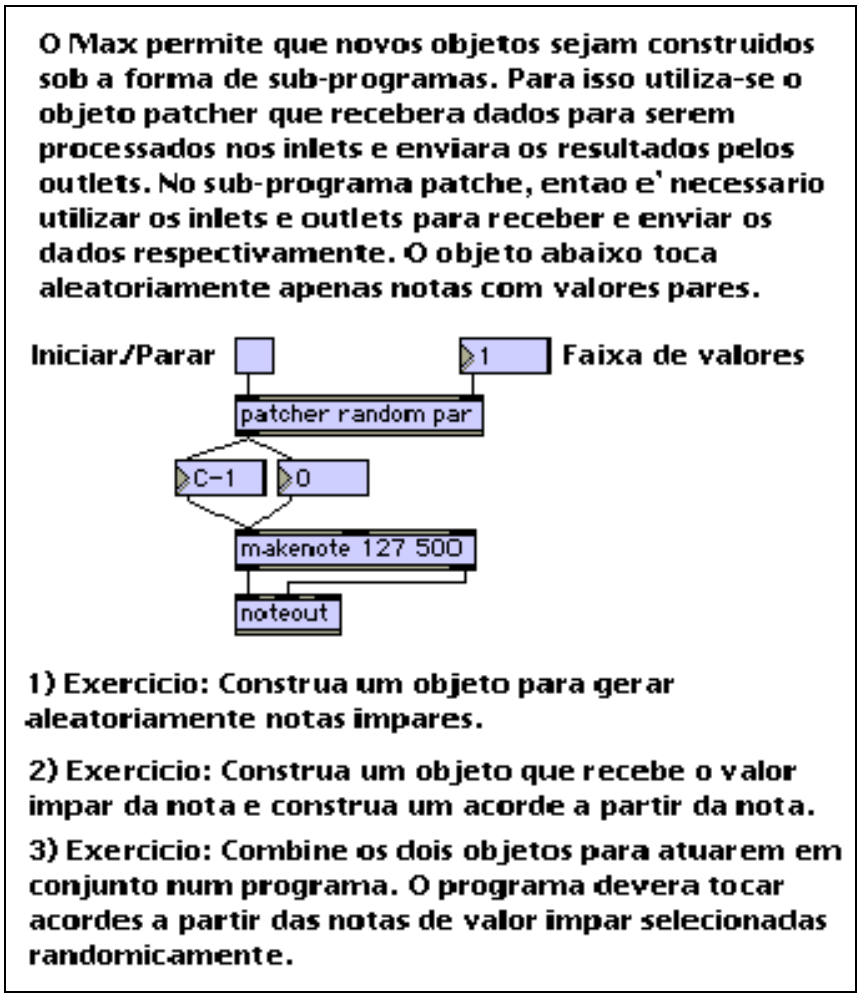


Figura 30 – Exemplo de programação utilizando o encapsulamento de rotinas e passagem de parâmetros

3.4.1.3 Implementação do Módulo de Ensino de Programação com Audiodigital

O Módulo de Ensino de Programação com Audiodigital prevê exemplos e exercícios com as principais funções de síntese sonora, como osciladores, amplificadores e filtros. Conforme já mencionado, cada elemento de síntese possui uma função já implementada em linguagem de programação. O objetivo principal desse módulo é explicar como essas funções podem ser agrupadas, programadas e utilizadas para fins musicais.

O Módulo de Ensino de Programação com Audiodigital é formado por três fases. A primeira destina-se a apresentar exemplos sobre os elementos básicos de

síntese: fonte, modificadores e interface musical. Este módulo ainda aborda a utilização de funções para processamento de efeitos, principalmente técnicas de espacialização.

A segunda fase apresenta exemplos e exercícios, utilizando os elementos básicos da síntese sonora.

A Figura 31 apresenta um exemplo que explica a utilização de envelopes para controle da amplitude do som num transcurso de tempo.

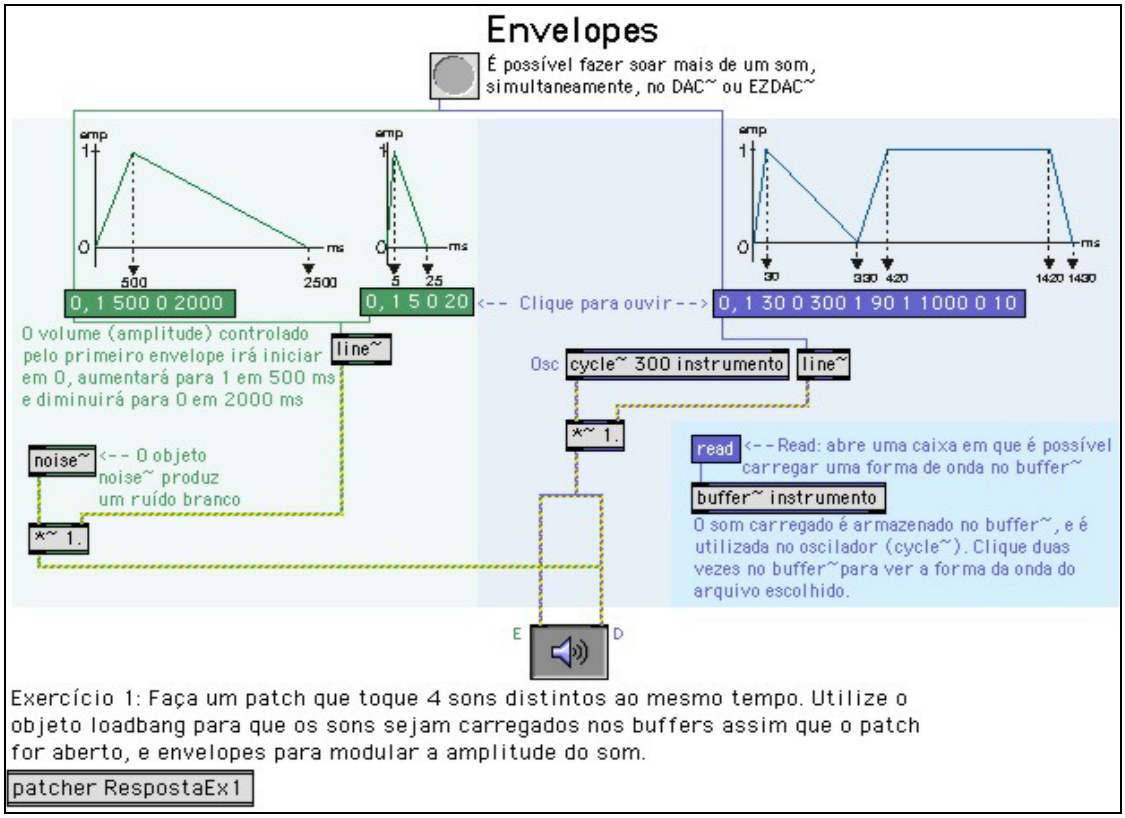


Figura 31 – Exemplo de programação utilizando síntese sonora

A terceira fase do Módulo de Ensino de Programação com Audiodigital é constituída por exemplos mais elaborados de combinações dos vários elementos de síntese sonora. A figura 32 mostra o exemplo que explica ao estudante como fazer um sintetizador através de programação visual. O sintetizador combina objetos Max e Msp no mesmo *patch* para gerar notas MIDI através do teclado virtual que serão convertidas em freqüências, filtradas e amplificadas.

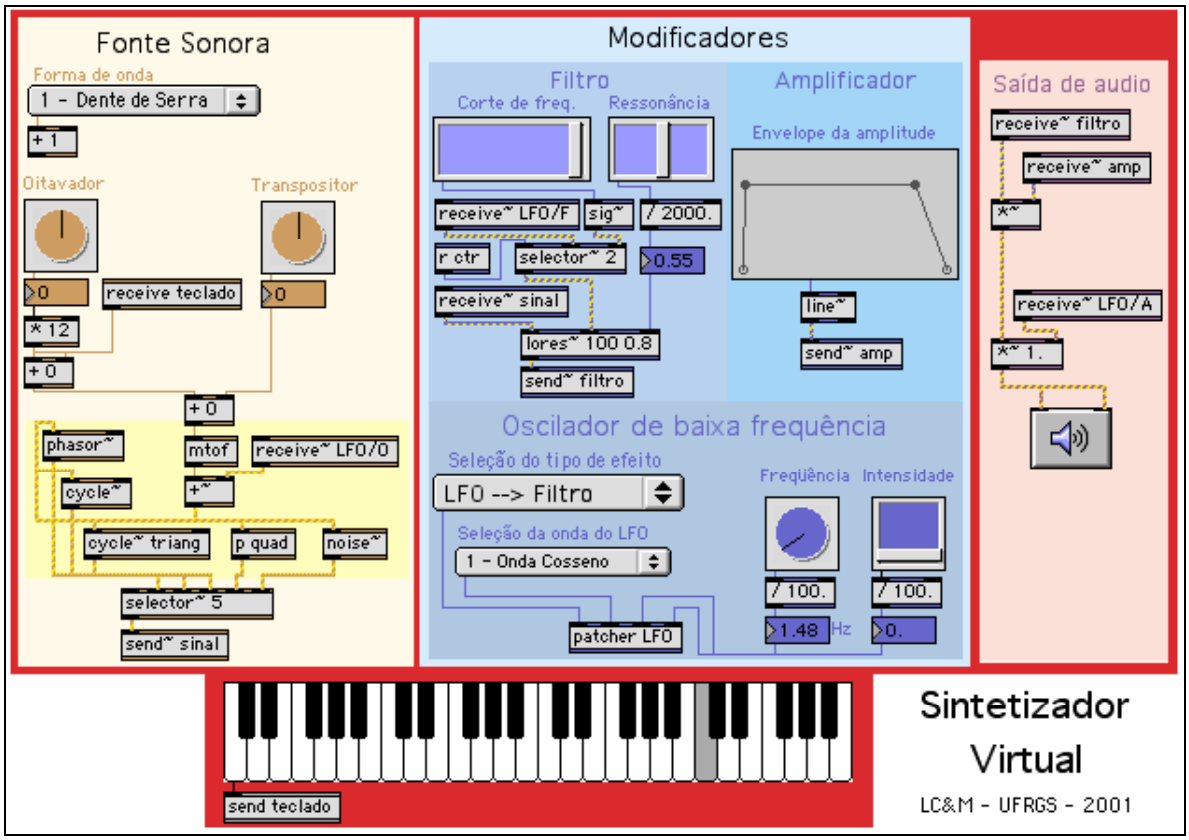


Figura 32 – Exemplo de um Sintetizador Subtrativo Virtual

3.4.2 Implementação do Nível da Programação Aplicada

O Nível de Programação aplicada à Musica consiste em exercitar, consolidar e aperfeiçoar todo o conhecimento adquirido nos módulos de Conhecimento Básico e de Programação Preparatória, através de sua aplicação em áreas distintas da música, tais como, Educação e Composição Musical.

3.4.3 Implementação do Módulo de Programação para a Composição Musical

A Programação para Composição Musical foi desenvolvida considerando que o aluno já domine técnicas de organização e geração de materiais musicais.

Neste módulo são utilizados exemplos de programas de composição com o emprego das seguintes técnicas:

- Composição Interativa, transformando coordenadas da tela do computador em material musical.
- Composição automática e aleatória a partir de parâmetros iniciais de entrada, vide Figura 33.
- Composição automática a partir de materiais musicais (séries) fornecidos ao programa pelo compositor.
- Construção de um programas para síntese sonora, com o objetivo de gerar e modificar material musical.
- Composição através de métodos estocásticos.
- Composição através de fractais.

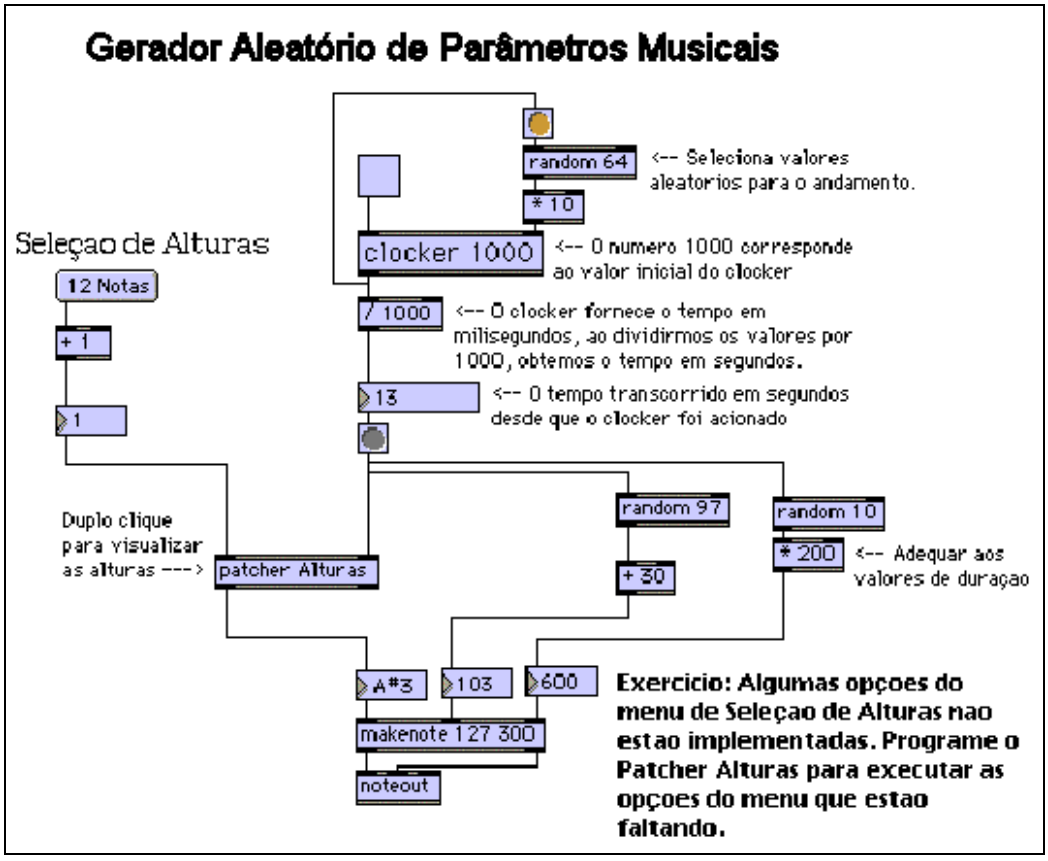


Figura 33 – *Patch* educacional para composição

3.4.3.1 Módulo de Educação Musical

O Módulo de Educação Musical foi programado em *Hypercard/HyperMIDI*. A seguir apresentaremos alguns exemplos e exercícios destinados ao ensino da programação de software de educação musical na área de teoria e percepção:

- Completar a programação de um *software* para ditado intervalar em que as alturas sejam executadas a partir de um instrumento externo ao computador.
- Completar a programação de um *software* para explicar como desenhar instrumentos na tela do computador e associa-los ao som correspondente (figura 34).
- Completar a programação de um *software* para seqüenciamento e transposição musical .
- Elaborar um *software* para educação musical utilizando o conhecimento dos programas anteriores.

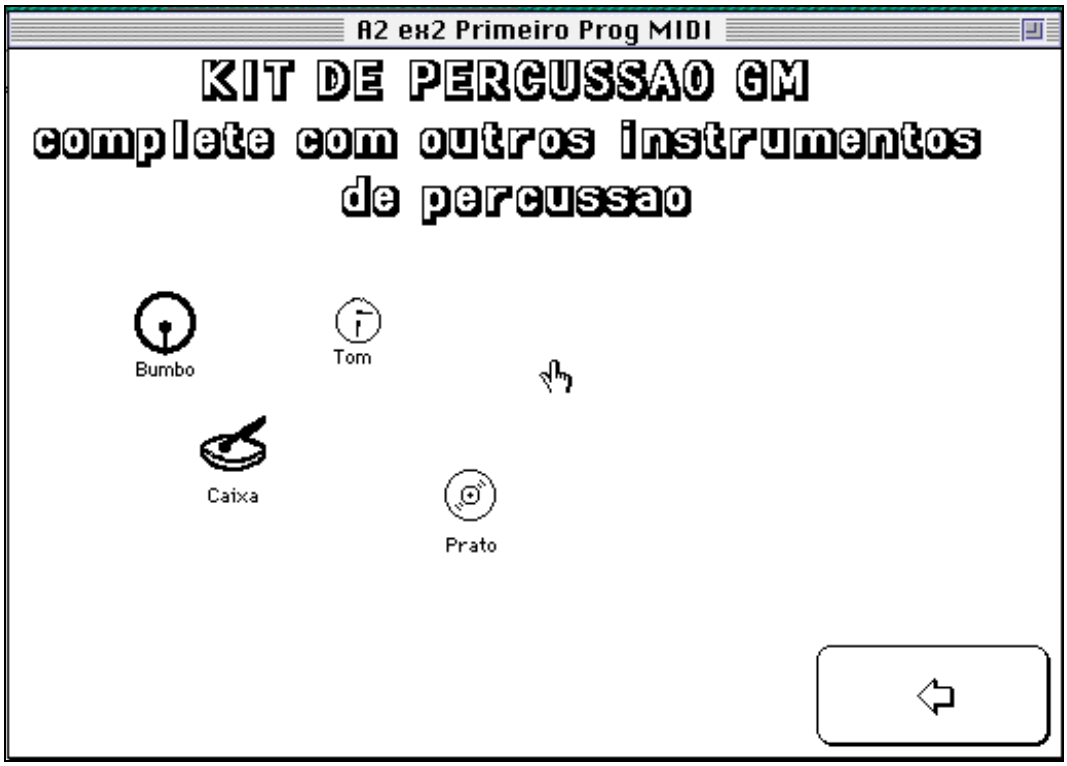


Figura 34 – Programa *HyperCard* para desenhar e programar instrumentos

4 Aplicação e Avaliação de MEPSOM

Neste capítulo relatamos os resultados da pesquisa sobre a aplicação do MEPSOM (Método de Ensino de Programação Sônica para Músicos) em aulas de programação de software para diversas subáreas da música. A pesquisa teve como objetivo principal a obtenção de dados para a criação e aprimoramento de software musical. Os programas desenvolvidos nesta pesquisa são utilizados como exemplos e para atividades em aulas práticas de programação sônica em laboratório.

A aplicação e avaliação do MEPSOM foi realizada com apoio em duas metodologias de pesquisa: (a) estudos de caso: aplicação em quatro cursos, com turmas diferentes de alunos; (b) pesquisas de levantamento: coleta de dados sobre o Método dos alunos participantes destes cursos, por meio de dois questionários.

Através da experimentação dos programas que compõem o MEPSOM em Laboratório, foi possível obter dados a respeito do ensino de Programação de Computadores para Músicos. Puderam ser efetuadas várias inferências a respeito de como aperfeiçoar os programas e aplicá-los da melhor forma possível, visando ao desenvolvimento do músico na área de programação. Os alunos contribuíram com a elaboração desses programas através de sugestões, avaliações por meio de questionários, comentários com relação à dificuldade na resolução de problemas, utilização dos programas em laboratório e capacidade de programar.

4.1 O ensino de programação para músicos

Na literatura da informática educacional e da educação musical estrangeira encontramos algumas sugestões para a organização de cursos nesta área [DEA 97] propõe um modelo de disciplina sobre música e tecnologia para cursos de graduação em música, a partir dos requerimentos do NASM (*National Association of Schools of Music*). Essa associação coloca a tecnologia como uma das seis principais competências dos estudantes de música "*tecnologia:* por meio de estudo e experiência em laboratório, os estudantes devem ser familiarizados com as capacidades da tecnologia em relação à composição, execução, análise, ensino e pesquisa" (ibid. pp.17, grifo dos autores). A partir de cinco questões relacionadas a fatores pedagógicos e logísticos, os autores sugerem os parâmetros para a implementação deste tipo de curso:

(1) Quais habilidades computacionais são essenciais para todos os graduandos?
(2) O treinamento computacional deve variar de área para área (por ex., de composição para educação musical) ou um conjunto de habilidades seria suficiente para todos?
(3) Todos os graduandos deveriam ser obrigados a fazer ao menos uma disciplina em técnicas computacionais [cursos de informática] ou, por outro lado, as experiências computacionais deveriam ser incorporadas em alguns cursos de música?
(4) Os estudantes de música devem ser enviados a outro departamento no *campus* para o treino computacional?
(5) Em um currículo de computação musical, como os professores podem lidar com as diferenças de habilidades computacionais encontradas em estudantes que estão entrando no curso? (p.18).

No Brasil, uma proposta anterior de uma disciplina de Computação Musical para cursos de graduação em música e em informática [FRI 99] consistia em um conjunto de conteúdos que podiam ser adaptados às especificidades de ambos os cursos. Na área musical, o enfoque reside na criação e uso de ferramentas para edição de partituras, composição, criação de novos sons e educação musical. O diferencial da proposta do autor e a aqui descrita reside na flexibilização do computador como ferramenta para composição e educação musical através da programação.

De maneira ampla, consideramos que o MEPSOM (Método de Ensino de Programação Sônica para Músicos) é uma proposta de ensino vinculado à teoria de aprendizagem construtivista. O Construtivismo Cognitivo Individual, advindo dos estudos de Piaget, entende a aprendizagem como um processo prático, no qual interagem a acomodação e assimilação do conhecimento, o que proporciona a modificação das estruturas cognitivas internas [SQU 94]. Esse processo é denominado "construção do conhecimento", e não, soma ou acúmulo de conhecimentos, por ser a síntese obtida por meio do processo de acomodação e assimilação [BEC 93], (p.57). Segundo este autor, essa construção ocorre por força da ação do sujeito sobre o objeto – ou meio físico e social – e pelo retorno ou repercussões dessa ação sobre o sujeito. O conhecimento dá-se por interação ou pelas trocas do organismo com o meio. A ação do sujeito sobre o objeto é entendida como ação assimiladora que transforma o objeto. As repercussões dessa ação, ou ação de retorno do objeto sobre o sujeito, enquanto implicam uma ação transformadora do sujeito sobre si mesmo ou sobre seus esquemas de ação/operação, são entendidas como ação acomodadora. Assimilação é a ação transformadora do sujeito sobre o objeto. Acomodação é a ação transformadora do sujeito sobre si mesmo (ibid. p.62).

Os aspectos que relacionam o MEPSOM à teoria construtivista são o aprendizado através da prática de programação, a atuação do professor como orientador do processo de ensino / aprendizagem e o desenvolvimento da capacidade de programação do aluno através do raciocínio lógico-matemático e da criatividade.

4.2 Fatores que possibilitaram a criação do MEPSOM

A proposta de criação do MEPSOM tornou-se viável devido à sua elaboração em Laboratórios de Computação Musical. A criação do MIDILAB – Laboratório MIDI, possibilitou que estudantes do curso de música que desejavam aprender sobre as novas tecnologias aplicadas à música encontrassem tais oportunidades. Observamos na Figura 35 que o MEPSOM foi criado devido a um conjunto de atividades que ocorreram na década de noventa e que possibilitaram um ambiente mais favorável às pesquisas. Essas atividades prepararam a comunidade e geraram um *background* de conhecimento que, posteriormente, foi aproveitado no Método.

Conforme pode ser visto na figura 1, as pesquisas realizadas no LC&M do Instituto de Informática da UFRGS resultaram nos seguintes programas musicais, com aplicações na área de educação: SETMUS – Sistema Especialista para Teoria Musical [FRI 95], STI – Sistema de Treinamento de Intervalos [FRI 96] [FLO 2000], STR – Sistema de Treinamento Ritmico [FRI 98] e o SeVEM - Separador de Vozes em Execuções Musicais Polifônicas [WUL 01].

Durante o processo de desenvolvimento desses programas foram realizados estudos sobre Linguagens de Programação para Música, técnicas de programação multimídia e aprimoramento de interfaces visuais e sonoras. A avaliação deses estudos,

auxiliou na correção de falhas de conteúdo, programação e interface dos programas [FLO 01].

Na área de ensino de programação e de avaliação do MEPSOM, destacaram-se os Cursos de Extensão em Tecnologia Aplicada à Música, ministrados no Departamento de Música da UFRGS para músicos interessados em aprender Computação Musical. Essa preparação apresentou aos músicos conteúdos da área de música e tecnologia, capacitando-os para o curso de Programação de Computadores para a Música. Os Cursos de Extensão e as Disciplinas de Graduação nesta área serviram para avaliar o Método de Programação de Computadores para Músicos.

Por fim, a infra-estrutura criada para as pesquisas (Laboratórios LC&M no Instituto de Informática, MIDILAB e LME no Instituto de Artes) possibilitou a implementação de software musical, o desenvolvimento, a aplicação e a avaliação do MEPSOM.

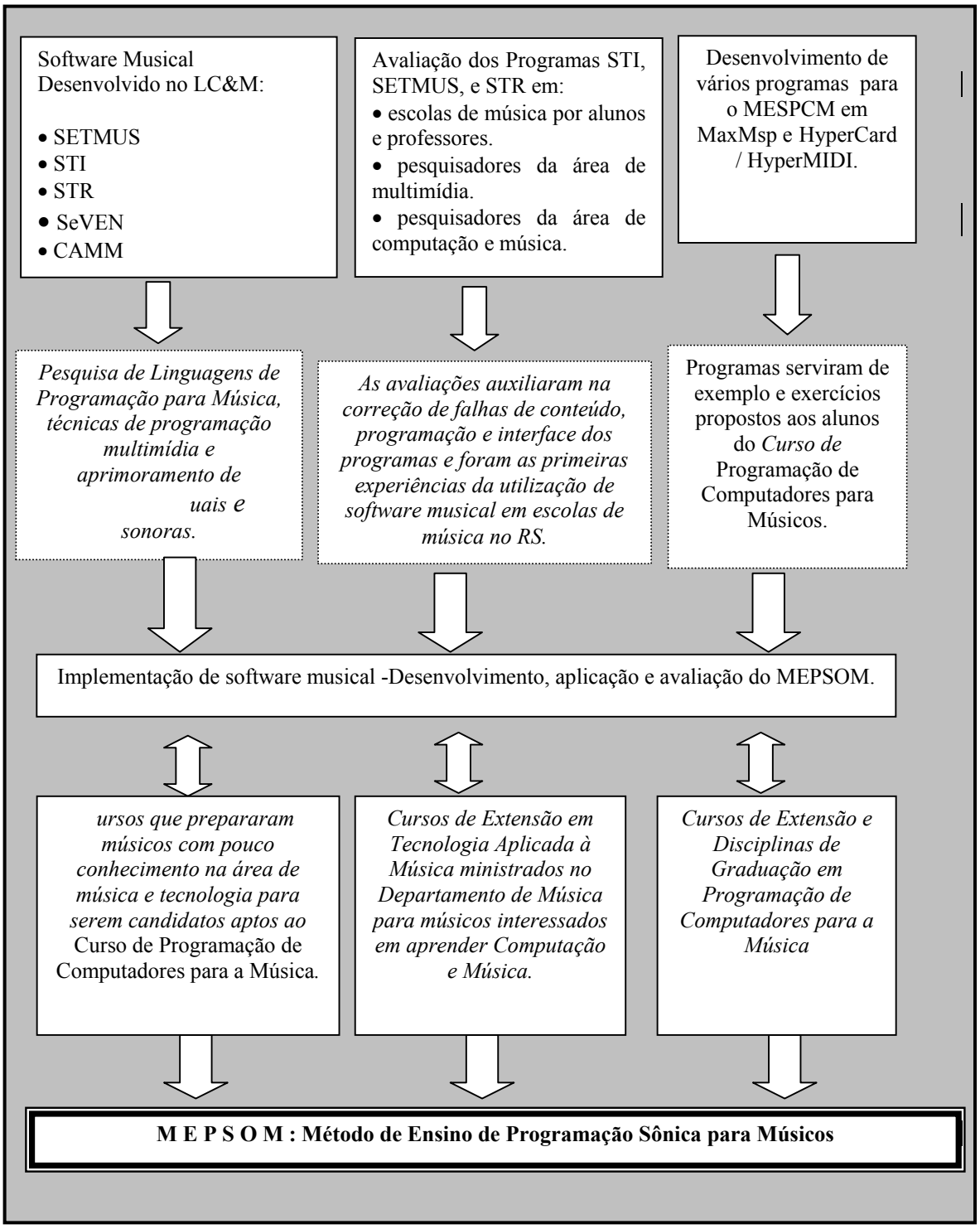


Figura 35 - Fatores que possibilitaram a criação do MEPSOM

Obs.: As caixas pontilhadas referem-se aos resultados dos programas conduzidos anteriormente.

Os programas que fazem parte do MEPSOM podem ser utilizados pelos professores de Computação Musical e os estudantes, sob a forma de exemplos e atividades práticas, utilizando instrumentos MIDI ligados ao computador. Os programas

estão sendo criados com o objetivo de permitir que o músico desenvolva sua capacidade de programação para aplicá-la na criação de software musical.

4.3 Aplicação do MEPSOM

4.3.1 Metodologia

Na pesquisa aqui relatada empregamos a metodologia de "Pesquisa de Desenvolvimento" [RIC 96], que enfatiza o "estudo do projeto, desenvolvimento, ou dos processos de avaliação, ferramentas ou modelos" e leva a um "novo *design,* desenvolvimento e avaliação de procedimentos e modelos, e das condições que facilitam seu uso" e, geralmente, propicia a generalização das conclusões (p.1217). Pode incluir diversas e distintas fases, caracterizadas pelo relato e análise sucessiva de um conjunto de dados e, principalmente, "podem ser conduzidos estudos secundários para analisar e definir o problema instrucional, para especificar o conteúdo, ou para determinar a validade e confiabilidade do instrumento" (ibid. p.1218). De acordo com [TRI 87], dependendo do objeto de estudo, "a coleta e a análise dos dados não são divisões estanques. As informações que se recolhem geralmente são interpretadas, e, isto pode originar a exigência de novas buscas de dados" (p.131). Como se trata do desenvolvimento e avaliação de um método para ensino de programação, esta metodologia foi considerada pertinente.

A aplicação do MEPSOM em cursos de Programação de Computadores para a Música ocorreu em quatro etapas ou cursos. Os programas do MEPSOM foram gradualmente implementados de acordo com o projeto (Figura 1) e utilizados em laboratório pelos alunos do Programa de Extensão em Música Eletrônica do Instituto de Artes da UFRGS. Os programas foram construídos e aperfeiçoados de acordo com os dados obtidos a partir dos estudos de caso. A Tabela 5 apresenta os dados referentes aos quatro cursos.

Tabela 5 – Etapas de desenvolvimento e aplicação do MEPSOM

Itens	1ª Etapa	2ª Etapa	3ª Etapa	4ª Etapa
Curso / Disciplina	Extensão: Programação de Computadores para a Música	Extensão: Programação de Computadores para a Música	Extensão: Programação de Computadores para a Música	Graduação: Programação de Computadores para a Música
Número de alunos que iniciaram o curso	7	4	5	3
Número de alunos que finalizaram o curso	4	2	5	3
Número de aulas introdutórias sobre MIDI	2	1	2	1
Número de aulas *HyperCard/HyperMIDI*	8	4	4	4
Número de aulas Max	6	11	10	11
Número de encontros semanais	1	1	2	1
Total de encontros	16	16	16	16
Duração de cada encontro	1 hora/aula	2:30 hora/aula	2:30 hora/aula	3 hora/aula
Duração do curso	4 meses	4 meses	2 meses	4 meses
Carga horária	20 horas/aula	40 horas/aula	40 horas/aula	48 horas/aula

Os programas do MEPSOM foram gradualmente aperfeiçoados. Cada uma das etapas visava a diferentes objetivos. Nas primeiras etapas, conforme mostra a Tabela 6, os objetivos consistem em levantar informações para a prototipação dos programas. A partir da terceira etapa, os programas foram avaliados e refinados.

Tabela 6 - Principais Objetivos durante a elaboração dos exemplos e exercícios do MEPSOM

1ª Etapa	2ª Etapa	3ª Etapa	4ª Etapa
• Conhecimento dos músicos e as tarefas que desejam realizar em computação e música. • Prototipação de programas, dando ênfase ao desenvolvimento de exemplos e exercícios para a programação preparatória.	• Prototipação de programas, dando ênfase ao desenvolvimento de exemplos e exercícios para a programação preparatória. • Alteração dos protótipos criados na primeira etapa conforme resposta dos alunos em Laboratório.	• Avaliar os protótipos criados na primeira e segunda etapas. • Prototipação de programas, dando ênfase ao desenvolvimento de exemplos e exercícios para aplicação nas áreas de educação musical e composição. • Refinamento dos protótipos e criação de mais exemplos e exercícios nas áreas de educação musical e composição.	• Avaliação dos exemplos e exercícios do MEPSOM. • Refinamento dos Protótipos.

4.3.2 Amostra

Para a seleção dos indivíduos que participaram dos cursos foi adotada a amostragem não probabilística, cujas diversas técnicas não incluem todos os membros do público alvo na amostragem [CAS 92]. Dentre essas técnicas, optamos pela Amostragem por Objetivo, que se caracteriza pela "busca de sujeitos que possuem uma característica de interesse específico à pesquisa" [CAS 92], (p.117). Isso foi necessário devido à constatação de dados obtidos em questionários preliminares ministrados a alunos dos Cursos de Extensão da UFRGS. O perfil de aluno que tem conhecimentos de informática e música ainda é raro no Brasil, e, quando os têm, são muito divergentes (por exemplo, conhecimento acentuado de música, e básico, em informática, ou vice-versa).

Assim sendo, o principal critério de seleção dos estudantes dos Cursos seria o conhecimento mais próximo possível do *Nível de Conhecimento Básico* do MEPSOM (Figura 2). Portanto, os candidatos deveriam comprovar conhecimento elementar nas áreas de Música, Tecnologia Aplicada à Música, Computação e Música e Informática, e idade acima de 15 anos.

Foi, assim, elaborado um formulário para levantamento do grau de desenvolvimento musical/tecnológico dos alunos candidatos, visando a identificar aqueles que contemplassem o perfil desejado para participar dos Cursos. Dos 49 questionários de seleção respondidos, foram escolhidos 19 alunos que preenchiam as características necessárias. Dos 19 alunos que iniciaram as 4 turmas, 14 completaram o curso e responderam aos questionários de avaliação.

4.3.3 Obtenção de Dados para Avaliação

Durante o desenvolvimento do MEPSOM (Tabela 3), várias inferências foram realizadas, principalmente pela aplicação dos programas em situações reais de aprendizado. As aulas foram planejadas e após cada uma foi efetuado um relatório, o qual foi posteriormente analisado.

As conclusões obtidas após cada Curso (etapa), permitiram que os programas e a metodologia (didática) de ensino fossem continuamente aprimorados com relação a vários aspectos, como consistência, interface visual e sonora, conteúdo, adequação à linguagem do músico, organização, clareza e simplicidade.

4.4 Avaliação do MEPSOM

4.4.1 Metodologia

A Metodologia de Pesquisa de Desenvolvimento permite o uso de métodos secundários, de acordo com os objetivos do estudo. Nesta fase adotamos o Método de *Survey* (ou Estudo de Levantamento) de Corte Transversal, que é caracterizado por uma única coleta de dados de uma "porção da população de interesse (...), na expectativa de que os indivíduos examinados propiciem informação relativamente descritiva de uma população inteira" [CAS 92], (p.116).

4.4.2 Amostra

Todos os 14 indivíduos que completaram o curso responderam aos questionários de avaliação.

4.4.3 Instrumentos de Coleta de Dados

Para a avaliação do MEPSOM, foram utilizados dois instrumentos de avaliação: um questionário aberto e um questionário fechado. [CAS 92] define questionário como "um conjunto de questões ou declarações impressas" que são respondidas de forma escrita (p.118).

No questionário aberto, os respondentes podem utilizar suas próprias palavras para as respostas (ibid. p.118). O questionário aberto utilizado nesta pesquisa consistiu de uma ficha de resposta com uma única questão, avaliando o MEPSOM: "o que você pode dizer sobre o MEPSOM?". Dessa forma, oferecemos aos alunos um espaço para que demonstrassem pontos favoráveis do MEPSOM e efetuassem críticas e sugestões, sem indução a questões específicas.

No questionário fechado, os itens são formados por categorias ou questões com opções de respostas [NEW 98], compostas por cinco alternativas. Foram efetuadas análises qualitativas das respostas do questionário aberto e dos relatos dos alunos; e quantitativas, das respostas do questionário fechado.

4.4.4 Resultados

4.4.4.1 Questionários fechados

Durante a utilização dos programas no Laboratório, foi possível verificar vários itens que colaboraram para o desenvolvimento de exemplos e exercícios. Dentre eles podemos citar: a aceitação dos exemplos e exercícios pelos alunos, o nível de facilidade em resolver os problemas, o tempo de resolução dos problemas, a quantidade de informação adicional necessária além daquela disponível no MEPSOM, a quantidade de estudo extra-classe, o interesse dos alunos em aprender e resolver os exercícios, a curiosidade em relação a certos tópicos do conteúdo de programação e a eficiência em aprender com o auxílio do MEPSOM.

Para esta Tese foram selecionadas seis questões que sintetizam o resultado das avaliações com o questionário fechado. Apesar de as avaliações terem sido realizadas sem o Módulo de Programação com Audiodigital, os resultados indicam o sucesso do MEPSOM utilizado pelos músicos no Laboratório de Música Eletroacústica da UFRGS.

As respostas dos músicos aparecem sob a forma de gráficos, indicando o percentual de cada opção escolhida, conforme as figuras 36, 37, 38 e 39.

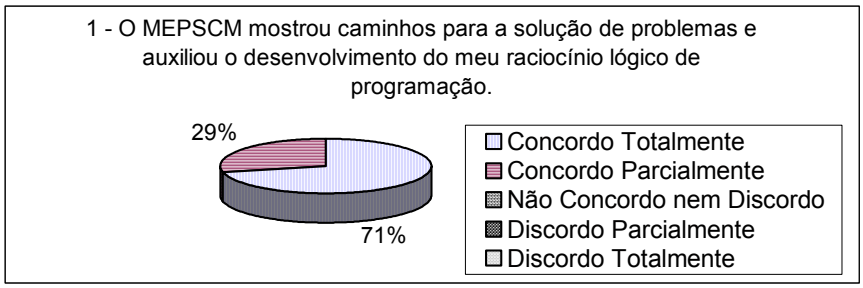


Figura 36 – Primeiro Gráfico resultante do somatório das avaliações das quatro turmas de Programação de Computadores para a Música

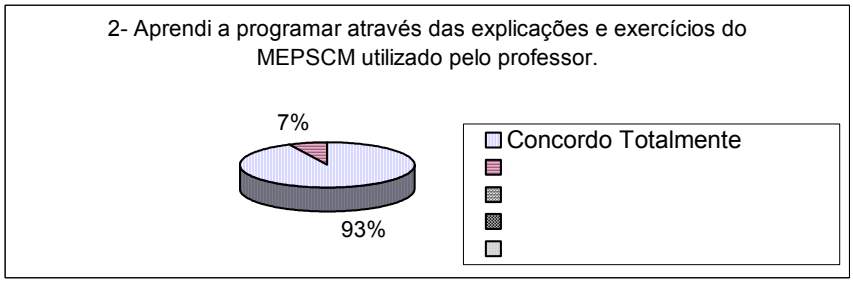


Figura 37 – Segundo Gráfico resultante do somatório das avaliações das quatro turmas de Programação de Computadores para a Música

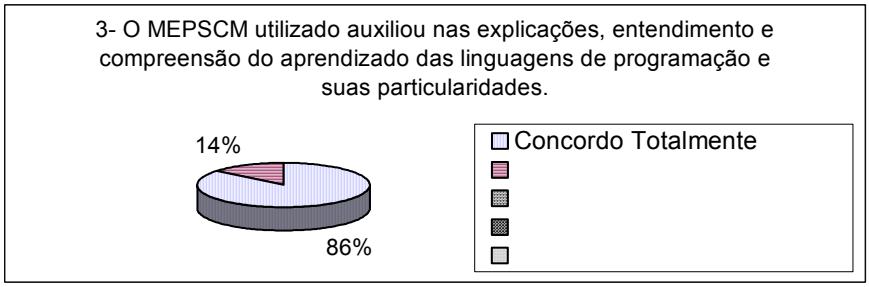


Figura 38 – Terceiro Gráfico resultante do somatório das avaliações das quatro turmas de Programação de Computadores para a Música

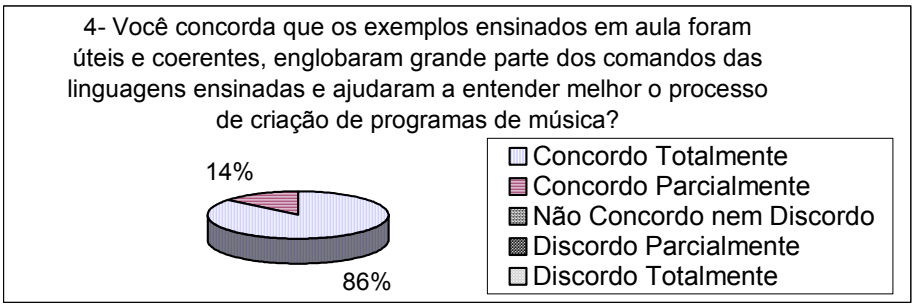


Figura 39 – Quarto Gráfico resultante do somatório das avaliações das quatro turmas de Programação de Computadores para a Música

4.4.4.2 Questionários abertos

De acordo com as várias observações e sugestões dos alunos ao longo dos cursos (quatro etapas), foi possível obter um conjunto de conclusões e melhorias a serem feitas em praticamente todos os pontos da aplicação do MEPSOM. As questões levantadas pelos alunos vão desde a carga horária necessária até mais clareza em exercícios e exemplos desenvolvidos para o método. A seguir sintetizamos, nas tabelas 7, 8, 9 e 10, os principais pontos levantados em cada etapa da avaliação, através de questionários abertos e anotações realizadas durante as aulas.

Tabela 7 – Resumo das observações sobre a 1ª etapa da utilização do MEPSOM no curso de extensão Programação de Computadores para a Música da UFRGS

1ª Etapa
• O método a ser desenvolvido precisa ser empregado numa carga horária superior a 20 horas/aula. ▪ Os exemplos e exercícios devem, sempre que possível, apresentar uma interface sonora, pois os interessados em aprender a programar aplicativos são músicos. ▪ As linguagens de programação MIDI devem ser de alto nível e com um alto grau de abstração do conhecimento musical. ▪ Alguns músicos que possuem menos contato com a tecnologia digital precisam de um tempo de adaptação no ambiente MIDI, a fim de se familiarizarem com a linguagem técnica utilizada. ▪ Os músicos precisam programar vários exemplos através de exercícios dirigidos e passar a maior parte do tempo possível em contato com o ambiente formado pelo computador, software e equipamento musical interligados. ▪ Os programas que constituem o método devem ser claros, bem documentados, abordarem os principais tópicos da programação MIDI, apresentar resultados sonoros e visuais úteis para o entendimento do conteúdo apresentado. ▪ Programas complexos devem ser abordados em partes e baseados no conhecimento musical, a fim de, sempre que possível, falar a mesma linguagem que o músico, mesmo nos momentos mais técnicos e próprios da Ciência da Computação. ▪ Os exercícios devem apresentar ênfase na área de interesse musical do aluno, como composição, educação musical, performance, etc.

Tabela 8– Resumo das observações sobre a 2ª etapa da utilização do MEPSOM no curso de extensão Programação de Computadores para a Música da UFRGS

2ª Etapa – Prototipação Max e Avaliação
• Em relação ao primeiro teste houve um aumento tanto na qualidade quanto na quantidade de conteúdos apresentados. • O MAX mostrou ser mais flexível para aplicações musicais relacionadas à composição e performance do que o HyperCard/HyperMIDI. • O HyperMIDI mostrou ter uma interface mais apropriada para o ensino de programação voltada ao desenvolvimento de programas educacionais para a música. • Os alunos não mostraram grandes dificuldades na resolução dos exercícios, devido ao avanço gradual dos conteúdos e ao reforço de revisão permanente, fruto das próprias linguagens de programação utilizadas. • O número de 40 horas/aula melhorou o rendimento dos alunos que, apesar de realizarem encontros semanais, aprenderam a programar nas suas linguagens. • O estudo da programação de aplicações musicais através de exemplos e exercícios dirigidos mostrou ser eficiente, pois o aluno acostuma-se, desde a primeira aula, a realizar continuamente exercícios com um grau de dificuldade e tempo de duração diferenciado. Com isso o aluno desenvolve sua capacidade de pensamento lógico e solução de problemas através de programação.

Tabela 9 – Resumo das observações sobre a 3ª etapa da utilização do MEPSOM no curso de extensão Programação de Computadores para a Música da UFRGS

3ª Etapa
• Em relação aos testes anteriores houve um aumento tanto na qualidade quanto na quantidade de conteúdos ensinados. • Apesar de o maior número de aulas terem sido destinadas ao Max, os alunos mostraram interesse pelas duas linguagens de programação. • Os alunos resolveram os exercícios do HyperCard/HyperMIDI com facilidade, principalmente nas três primeiras aulas. Em parte, pela familiaridade que já possuíam com os browsers de navegação na internet. • O número de 40 horas aula mostrou ser insuficiente para alguns alunos, pois estes tinham pouco tempo de estudo em horários extraclasse e não possuíam equipamentos Macintosh para exercitar em casa. • Com o refinamento dos exercícios propostos no MEPSOM, pela inclusão das respostas que faltavam e pela adição de novas explicações, os alunos demonstraram menos dúvidas para o professor do que nas edições anteriores do curso. • Foi a turma com o maior número de alunos. Todos acompanharam o andamento normal das aulas propostas e realizaram todos os exercícios em sala de aula. O MEPSOM mostrou ser um complemento importante para aulas em grupo.

Tabela 10 – Resumo das observações sobre a 4ª etapa da utilização do MEPSOM na disciplina de Programação de Computadores para a Música do curso de Bacharelado em Composição da UFRGS

4ª Etapa
• Os alunos do curso de Bacharelado em Música utilizaram o MEPSOM para aprender a programação. Utilizaram programas feitos por eles próprios para elaborarem composições eletroacústicas, mostrando que é possível empregar o MEPSOM para ensinar programação aos músicos de forma eficiente. • Maior carga horária e maior dedicação dos alunos ao curso colaborou para um melhor resultado. O MEPSOM foi utilizado em todas as aulas pelos alunos que aprenderam a desenvolver aplicativos musicais nas linguagens ensinadas, mesmo sem um embasamento anterior na área de programação. • Os alunos mostraram interesse pela disciplina e seguiram o roteiro do MEPSOM realizando todos os exercícios propostos no método. Logo, concluímos que o MEPSOM colaborou para o aprendizado rápido dos alunos. • Alguns exercícios do roteiro ainda deverão ser aperfeiçoados, principalmente com relação à interface gráfica e à forma como o conteúdo musical é tratado. • Até agora o MEPSOM foi apresentado como um conjunto de exemplos e exercícios, no entanto deverá ser aperfeiçoado para dois sistemas individuais, face à particularidade das linguagens abordadas. • Uma vez elaborada a parte do método relativa à tecnologia MIDI, o MEPSOM poderá incorporar os conteúdos de DSP com a elaboração de programas em MSP. • Esta foi a primeira utilização do MEPSOM numa turma de Bacharelado em Composição Musical. Todos acompanharam o andamento normal das aulas propostas e realizaram a grande maioria dos exercícios em sala de aula. O MEPSOM mostrou ser um importante aliado do professor de computação e música, tornando a aula fundamentalmente prática e com aplicações dirigidas às necessidades dos alunos de composição eletroacústica.

5 Proposta de Aplicação do MEPSOM

Conforme as experiências relatadas, a avaliação realizada e as conclusões obtidas pela aplicação do método, podemos formular uma primeira versão de aplicação do MEPSOM, que ainda será apurado e aperfeiçoado em avaliações futuras.

5.1 Perfil dos alunos

De acordo com o laboratório criado para a construção e validação do método, o número de alunos participantes recomendado é de 4 a 8, por turma, sendo aconselhável que haja apenas 1 por estação de trabalho. O conhecimento prévio dos alunos deve ser coerente com o Nível de Conhecimento Básico do MEPSOM (Figura 2). Portanto, os pré-requisitos necessários para os músicos realizarem o MEPSOM são conhecimento em:

- Introdução à Tecnologia Aplicada à Música
- Formas de estruturar o pensamento e prática de operação de computadores
- Introdução à Computação e Música

5.2 Perfil do Professor

O professor de Computação Musical deverá dominar programação visual de computadores e conhecer composição e educação musical. Além disso, deverá estar interado com o conteúdo do método e conhecer os exemplos e exercícios propostos.

5.3 Equipamento (Hardware e Software)

O Método estipula que deva ser criado um laboratório para as aulas serem ministradas contendo 4 ou 8 EPCMs - Estações de Programação de Computadores para a Música; cada EPCM, composta de um Macintosh Performa 7200 ou superior; uma interface MIDI 1x1 - 1 entrada e 1 saída; um fone de ouvido profissional; um sintetizador polifônico, multiparte e multitimbral dotado de sistema MIDI e GM; e uma placa de áudio de alta fidelidade. A pesquisa aqui relatada utilizou apenas a tecnologia MIDI e computadores Macinstosh LCIII e, por essa razão, não avaliou o Módulo de Ensino de Programação com Audiodigital[15]. As linguagens de programação utilizadas foram o MaxMsp e o HyperCard/HyperMIDI. No entanto, em tese, qualquer linguagem de programação de computadores que possua recursos para desenvolver software musical poderia ser utilizada. Por outro lado, para o MEPSOM, aconselha-se utilizar linguagens visuais e amigáveis; pela facilidade de programação.

[15] Na época em que foi realizada a avaliação a implementação do Módulo Programação com Audiodigital não estava finalizada. Além disso, o equipamento computacional não permitia o funcionamento da tecnologia de audiodigital.

5.4 Programa e Conteúdo

Conforme a Figura 2, observamos que os conteúdos são desenvolvidos de acordo com os Módulos do MEPSOM. Portanto, este método é apresentado ao aluno por meio de um material didático composto de disquete com exemplos e exercícios em MaxMsp e HyperCard/HyperMIDI, e um polígrafo com o seguinte conteúdo[16]:

- Revisão sobre MIDI e Audiodigital
- Programação para comunicação entre instrumentos musicais e computadores
- Elementos básicos da Programação de Computadores
- Programação com Audiodigital
- Programação de Aplicações na área de Educação Musical
- Programação de Aplicações na área da Composição Musical.

5.5 Duração das aulas

As etapas foram ministradas com diferentes períodos e cargas horárias, tendo sido utilizadas durações e freqüências de encontros diferentes para verificação do desempenho dos alunos. De acordo com a avaliação das quatro etapas utilizadas para realizare o MEPSOM, é possível formular uma sugestão de período, freqüência e duração das aulas. O Método prevê 50 horas aula em 20 encontros, com a duração de 2 horas e 30 minutos. O curso deverá ocorrer em 2 meses com 2 encontros semanais. No entanto, a tendência é o aumento da carga horária, devido ao aumento do conteúdo e ao resultado dos questionários de avaliação com relação a este item.

[16] Os conteúdos estão disponíveis nos anexos desta TESE DE DOUTORADO.

O MÉTODO

A seguir será apresentada a documentação do MEPSOM destinada aos estudantes de Programação Sônica de Computadores para Música. A apresentação segue o gráfico da Organização do MEPSOM (figura 1).

6 NÍVEL DE CONHECIMENTO BÁSICO

O conhecimento básico presente no MEPSOM está dividido em três módulos: Tecnologia Aplicada à Música, Formas de Estruturar o Pensamento e Computação e Música.

6.1 MÓDULO DE INTRODUÇÃO À TECNOLOGIA APLICADA À MÚSICA

Neste módulo são apresentadas tecnologias utilizadas na área de música eletrônica como ligações MIDI, áudio digital, síntese e instrumentos eletrônicos

6.1.1 Histórico da Tecnologia Musical

A intenção em relatar os marcos do avanço tecnológico é de ressaltar a importância que os novos instrumentos eletrônicos adquiriram ao longo dos anos, principalmente a partir da década de 70, e as alterações que estes causaram na maneira de fazer música. O avanço da ciência na área da música é marcante a tal ponto de utilizar inteligência artificial, redes neurais e realidade virtual na construção de sistemas inovadores e que permitem uma grande flexibilidade ao usuário.

Nossa viagem inicia com a invenção do telefone por Alexandre Graham Bell (1876) considerado como o marco inicial do desenvolvimento da música eletrônica. Esta invenção estabeleceu que o som podia ser convertido em sinal elétrico e vice-versa. Por sua vez, a invenção do gramofone, que rapidamente se seguiu, estabeleceu as possibilidades de armazenamento e alteração do som.

Por volta de 1906 Thaddeus Cahil mostrou seu dinafone - “telharmonium", o primeiro instrumento que produzia som por meios elétricos. A geração do som era feita a partir de dínamos e a transmissão por meio de cabos telefônicos [ROA 96].

Por volta de 1915, um outro passo importante veio a ser dado com a invenção do oscilador a válvula, por Lee De Forest. O oscilador, que representa a base para a geração do som eletrônico, tornava possível a geração de freqüência a partir de sinais elétricos, e, conseqüentemente, a construção de instrumentos eletrônicos mais fáceis de manejar. O primeiro desses foi desenvolvido pelo russo Leon Termen, em 1919/1920 e foi posteriormente melhorado por volta da década de trinta. Este instrumento, o “Theremin", usava um oscilador que era controlado pelo movimento da mão do executante em volta de uma área vertical. Outros instrumentos eletrônicos rapidamente o seguiram. O inventor alemão Jorg Mager introduziu alguns deles na década de trinta. O ``Ondas Martenot" foi criado pelo francês Maurice Martenot e o ``Trautonium" pelo alemão Friedrich Trautwein, ambos em 1928. Neste mesmo ano o americano Lores Hammond produziu o primeiro órgão elétrico [CHA 97] .

Alguns compositores estavam voltados para a utilização e desenvolvimento de técnicas de composição que tratavam do som de uma maneira diferente da utilizada pelos compositores convencionais. Essas composições eram baseadas em gravações de sons pré-existentes, posteriormente trasnformados a partir de processos de alteração de rotação, superposição de sons ou fragmentos sonoros, execução em sentido inverso, etc. Isto foi chamado de **música concreta.** Pierre Schaeffer, compôs varias peças de Musique Concrète em colaboração com Pierre Henry, que, posteriormente, o sucedeu no cargo. Symphonie por um Homme Seul (1949-1950) foi uma das primeiras obras eletrônicas apresentadas ao público e um dos principais trabalhos dos dois compositores. [CHA 97].

Em 1952, foi criado em Köln, na Alemanha, o segundo estúdio de música que, em oposição aos princípios da música concreta, trabalhava exclusivamente por meios eletrônicos sem a interferência de sons naturais. Isto foi chamado de **música eletrônica**. Karlheinz Stockhausen foi o principal compositor deste estúdio e, ao contrário de Pierre Schaeffer, não se preocupava em transformar sons naturais, mas sim em criar música eletrônica a fim de sintetizar o som a partir de freqüências puras. A primeira composição eletrônica foi Studie I (1953) seguida de Studie II (1954) [CHA 97].

A utilização simultânea de sonoridades concretas e sons eletrônicos resultou na **música eletroacústica**, que tem na peça Gesang der Jüngling (1955-56) de Stockhausen, o exemplo pioneiro e mais significativo. Nesta peça, Stockhausen gravou a voz de um menino soprano que cantou versos que se intercalaram com texturas eletrônicas. Gensag der Jüngling foi composta para ser executada em 5 alto-falantes em movimento para projetar os sons no espaço. A mediação entre Paris e Colônia estabeleceu-se nesta obra de Stockhausen pois este utilizou tanto sons naturais gravados quanto sons gerados eletronicamente. Stockhausen compôs várias outras peças eltroacústicas como Kontakte (1960), Mikrophonie (1964-65) e Telemusik (1966). Em 1968 realiza Kurzwellen (Ondas curtas) onde o material composicional é fornecido por ondas curtas de rádio. O uso de aparelhos de rádio com o papel de instrumentos já havia sido utilizada por John Cage em sua composição Imaginary Landscape n. 4 para doze receptores em 1951 [CHA 97].

Os compositores de música eletroacústica geram suas composições em estúdios e laboratórios. As audições são realizadas através da reprodução das composições em fita. Este tipo de técnica foi chamado de “tape-music" ou música para fita magnética. Tal música, independente do processo de composição, seja ele acústico ou eletrônico, tem como particularidade o fato de que a sua execução em público só poderia ser feita por meio de reprodutores de fita e amplificação eletrônica. A sua execução era exatamente a criação do compositor, eliminando com isso a idéia do instrumento/intérprete. O compositor fazia os dois papéis utilizando-se de audições para mostrar suas obras, já que essas eram concebidas em laboratórios musicais.

Outro compositor de destaque na música eletroacústica foi Varèse. Uma de suas grandes obras foi Poème Électronique que foi executada em 1958 através de vários canais independentes ligados à um grande número de alto falantes nos Pavilhões da Philips em Bruxelas. As sonoridades reconhecíveis, como a voz de uma soprano acompanhada de coro, de sinos e de um órgão, juntaram-se a outras totalmente eletrônicas, resultando numa espacialização global da música e afirmando os concertos de música eletroacústica através da “tape-music”.

A pesquisa em música eletroacústica é realizada tanto em instituições particulares quanto públicas. Entre elas estão os centros internacionais de pesquisa Utrecht, Standford, MIT, e o IRCAM dirigido por Boulez em Paris [CHA 97].

Ainda na década de 50, a partir da antiga idéia de criar sons usando a eletricidade, Herbert Belar e Harry Olsen inventaram o Mark II RCA Music Synthesizer, o primeiro sintetizador controlado por voltagem. Deste instrumento somente um modelo foi fabricado. Milton Babbitt realizou as mais importantes composições com o Mark II Music Synthesizer dentre elas Compositions for Synthesizer (1961) e Esembles for Synthesizer (1964) para tape music. O Mark II representa o final da era dos primeiros instrumentos eletrônicos.

Max Mathews, considerado o pai da Música Computacional, desenvolveu no Bell Labs, em Nova Jersey, o primeiro programa de computador para a música em 1957 num computador de grande porte. O programa chamado Music I tinha uma única voz, uma forma de onda triangular, não possui ADSR e só controlava a afinação, intensidade e duração. O Music I deu origem a uma série de programas musicais como Music II, Music III e o Music IV. Estes programas abriram espaço para uma "avalanche" de novos programas musicais de todas as categorias que foram criados a partir de 1976 com a difusão dos microcomputadores e a utilização de linguagens de programação de alto nível e grande portabilidade.

Na década de 50, Robert Moog era um engenheiro que montava e vendia Theremins. Na década de 60 ele iniciou a construção de novos equipamentos que resolveu chamar de sintetizadores. Ele foi o responsável pela invenção dos primeiros sintetizadores modulares Moog que foram mundialmente difundidos através dos primeiros tecladistas que utilizaram sintetizadores em suas gravações e performances como Keith Emerson (Lucky Man) e Wendy Carlos (Switch on Bach). A chegada do sintetizador Moog e outros instrumentos eletrônicos semelhantes ao mercado, em 1964, representou uma revolução nas técnicas da música eletrônica. Os compositores não precisavam mais de tantas horas nos estúdios preparando e editando o seu material, como o caso de Stockhausen que utilizou 18 meses para produzir os poucos minutos da obra Gesang der Jüngling. Os sintetizadores possibilitaram o surgimento de novos gêneros musicais baseados em suas possibilidades tímbricas e de controle eletrônico dos parâmetros musicais. Os músicos então, utilizavam a síntese para criar seus próprios sons exigidos em suas composições e que até então não eram possíveis de serem produzidos pelos instrumentos convencionais.

A esta altura o interesse maior voltava-se pela execução ao vivo de instrumentos e equipamentos musicais eletrônicos que viria a ser chamada "live electronic music". Em meados da década de 60 multiplicavam-se os conjuntos de música eletrônica ao vivo, entre eles Sonic Arts Union dos EUA, o conjunto de Stockhausen e a Música Eletrônica Viva de Roma.

A música popular começou a assimilar a música eletrônica de vanguarda e grupos ingleses de rock progressivo da década de 70 como ELP, Yes e Gênesis começaram a utilizar sintetizadores em suas performances. A utilização da orquestra sinfônica, coral e sintetizadores na música popular foi difundida principalmente por Rick Wakeman com a obra Viagem ao Centro da Terra obtendo uma sonoridade ímpar até então na música popular. A utilização só de equipamentos eletrônicos originou também a música popular eletrônica tendo como expoentes Kraftwerk , Jean Michel Jarre e Vangelis Papathanassiou. Este último recebendo um Oscar nos EUA pela trilha sonora do filme Chariot's of Fire realizada apenas com piano e sintetizadores abrindo ainda mais o caminho para a música eletrônica. Até então, apenas compositores eruditos com trilhas para orquestra haviam alcançado esse prêmio.

Os sintetizadores evoluiram e deixaram de ser teclados analógicos com osciladores e potenciômetros para alteração do som para se tornarem uma incrível linha de produtos que vão desde módulos de som até computadores com software que

os tornam sintetizadores. Grandes empresas fabricantes de sintetizadores ganharam o mercado como a Roland, Yamaha e a Korg. Com base nos conceitos herdados dos sintetizadores analógicos modulares, vários tipos de síntese como FM, aditiva, subtrativa, linear, modelagem fisica foram desenvolvidas para os sintetizadores digitais modernos [YAV 91].

Na década de 70, já existiam ligações entre os instrumentos musicais, mas foi só em 1983 que a MIDI - Musical Instrument Digital Interface - surgiu para padronizar as comunicações entre instrumentos eletrônicos. Como o computador já havia sido usado para a música, mas sem uma ligação efetiva com instrumentos do meio externo, este foi rapidamente empregado na ligação com sintetizadores através do interfaceamento MIDI. Essa utilização da MIDI originou uma revolução no campo da música.

A criação dos microcomputadores pessoais e sua difusão nos anos 80 facilitou a configuração de um sistema MIDI completo pois barateou e simplificou a aquisição do hardware básico para aplicações em sistemas musicais. Logo após o padrão MIDI ser difundido, fabricantes de software iniciaram uma produção em grande escala dos mais diversos aplicativos musicais que utilizavam a comunicação entre instrumentos e o computador. No final de 1986, uma grande linha de produtos já existia.

Os computadores PC multimídia tornaram o padrão MIDI e a gravação digital em computador ainda mais difundida por já apresentarem esses recursos inclusos no preço final do computador.

6.1.2 A Tecnologia Digital na Música

Atualmente, com o aumento da capacidade de processamento e o barateamento da memória utilizada nos equipamentos, foram criados sintetizadores virtuais como o Virtual Sound Canvas da Roland e sistemas para gravação de áudio digital em discos rígidos.

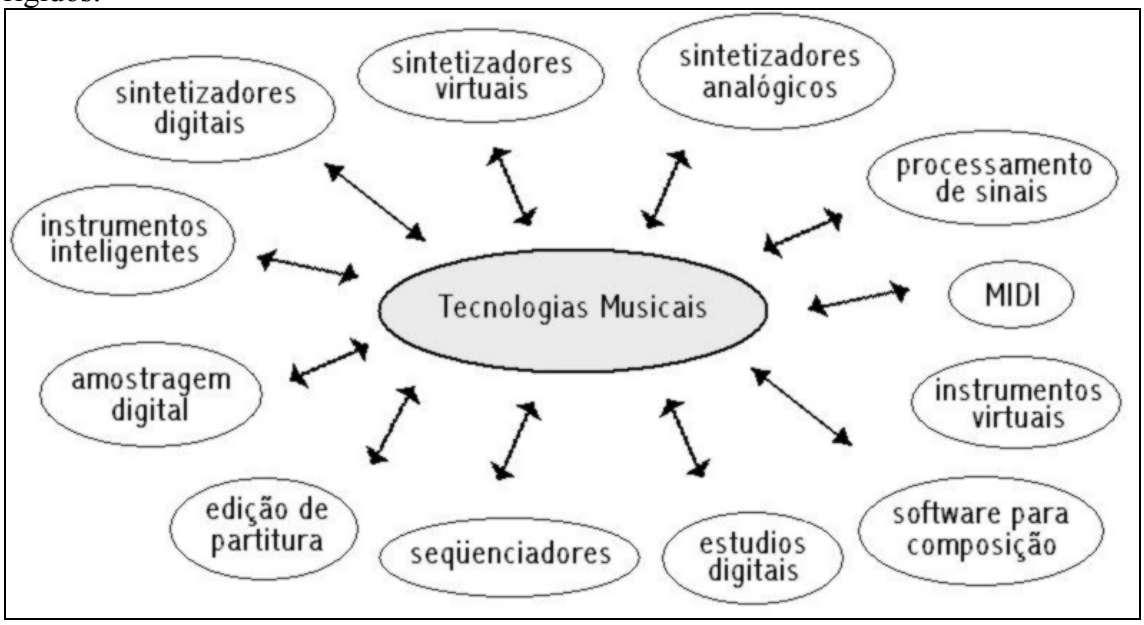


Figura 40 - Tecnologias musicais atualmente disponíveis.

A música por computador está sendo amplamente utilizada na internet através dos recursos de áudio digital compactado e MIDI. Instrumentos baseados em realidade virtual, laser e inteligência artificial já são uma realidade em laboratórios e mega-

espetáculos de música eletrônica. A gravação de grupos musicais pela internet e a educação musical à distância tornam-se realidade.

Os recursos de gravação evoluíram muito sendo que o músico pode gravar músicas em seu home studio economizando tempo e dinheiro. A figura 41 apresenta um estúdio composto por vários itens representando a tecnologia disponível na área da música eletrônica.

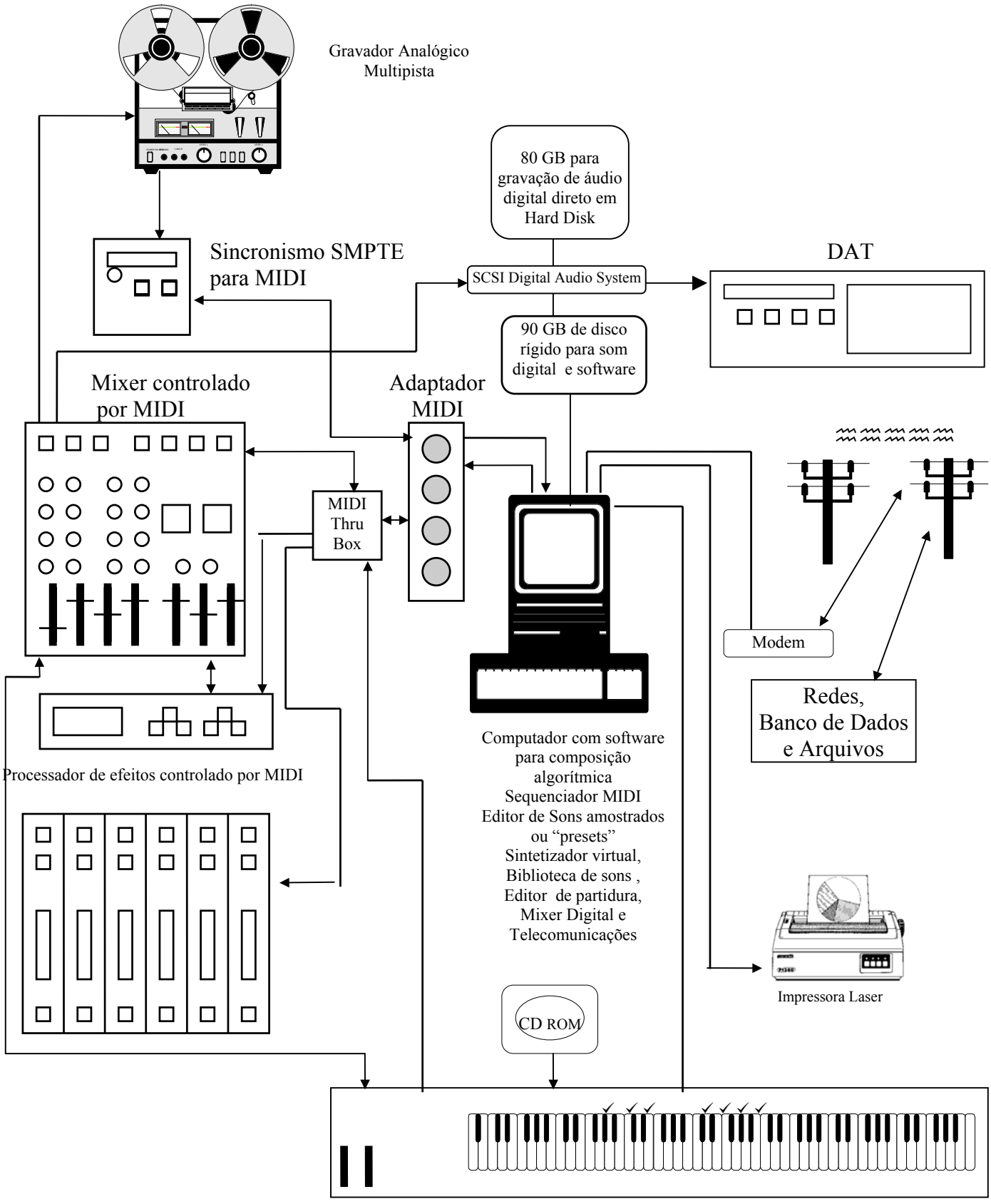


Figura 41 - Estúdio MIDI composto de vários equipamentos interligados

6.1.2.1 Os Instrumentos Musicais Modernos

Praticamente todos os instrumentos musicais (eletrônicos) de hoje possuem interface MIDI, ou seja, são dotados dos conectores apropriados e implementam funções de comunicação MIDI. Dessa forma, os equipamentos de um *home studio* ou de um estúdio profissional devem ser interligados e comandados através de MIDI. A seguir apresentaremos as principais características dos instrumentos modernos:

6.1.2.2 Multitimbralidade

No início do uso do MIDI, o músico que desejasse criar um arranjo que utilizasse diversos timbres simultaneamente teria que usar diversos instrumentos de forma que cada um executasse um dos timbres do arranjo. Assim, por exemplo, o instrumento A tocaria a parte dos violinos, o instrumento B tocaria a parte do baixo, o instrumento C tocaria a parte dos metais, e assim por diante.

Nesse caso, o arranjo seria efetuado em um seqüenciador, que posteriormente enviaria os comandos de execução musical (mensagens de notas, pedais, etc) aos instrumentos, ligados em cadeia. O seqüenciador pode ser imaginado como um equipamento portátil específico, ou então um computador equipado com interface MIDI e rodando um software para seqüenciamento.

Naquela época, para que se pudesse realizar esse tipo de aplicação era necessário um grande investimento na compra de diversos instrumentos. Mas, graças ao barateamento dos componentes e equipamentos digitais, a indústria começou a oferecer equipamentos que incorporavam dentro de si mais do que um único instrumento, e que, via MIDI, podiam funcionar como se fossem diversos instrumentos separados. A esses equipamentos dá-se o nome de multitimbrais; sua característica principal é a de poderem se subdividir internamente em diversos instrumentos independentes. No começo, esses equipamentos possuíam duas a quatro partes timbrais, mas hoje há equipamentos de 16 a 32 partes timbrais, o que significa que é possível criar e executar todo um arranjo usando-se apenas um equipamento desses.

6.1.2.3 Polifonia

Em um instrumento multitimbral, os recursos de geração de sons são compartilhados por todas as partes timbrais. Dessa forma, se a capacidade máxima de polifonia do instrumento (quantidade de sons diferentes que podem ser produzidos simultaneamente) é de, digamos, 16 vozes (notas), a soma de todas as notas executadas simultaneamente pelas partes timbrais não pode exceder a 16 notas. Nesse caso, há um critério de 'utilização' das vozes, em geral dando-se prioridade à nota mais recente. Isso significa que, considerando o exemplo, caso as partes timbrais já estejam tocando um total de 16 notas, e chegando um comando para executar uma 17ª nota, a nota mais antiga será eliminada (silenciada) para dar vez à mais recente. A limitação do número de vozes existe porque os recursos de geração de som (circuitos ou software) têm um limite (quantidade de circuitos ou velocidade de processamento).

6.1.2.4 Sensibilidade à velocidade

Este parâmetro refere-se à capacidade que o sintetizador tem de responder à variação da velocidade com a qual seu teclado é pressionado. Os fabricantes de

sintetizadores normalmente referem-se a este parâmetro como "key velocity" ou velocidade da tecla.

Isto permite um controle de determinados parâmetros do patch, de acordo com a dinâmica aplicada à execução. Por exemplo, ao se tocar suavemente (pianíssimo) obtém-se um nível de volume diferente de quando se toca mais fortemente (fortíssímo). Em alguns casos, é possível até selecionar um timbre diferente, de acordo com a velocidade com a qual se pressiona a tecla - esta característica é chamada "velocity switching" ou comutação por velocidade.

6.1.2.5 Aftertouch

Em tradução livre, aftertouch pode ser compreendido como pós-pressionamento da tecla, ou seja, um efeito conseguido quando se executa uma nota e, depois de obtido o efeito normal desta ação, sem que a tecla seja solta, pressionamo-la mais fortemente.

Não se deve confundir "aftertouch" com "key velocity": o efeito de aftertouch só é obtido depois que a nota gerou um determinado timbre, sujeito inclusive ao efeito de key velocity.

Normalmente, a ação do aftertouch se faz sentir na profundidade de modulação do LFO, quer seja no DVA (ou VCA), no DVF (ou VCF), ou ainda no WG (ou VCO). Assim sendo, conseguem-se efeitos de trêmolo, vibrato e wah-wah, selecionando-se o bloco no qual o LFO será aplicado.

6.1.2.6 Pitch Bend

É o desvio de tom. Normalmente, uma alavanca ou controle do tipo circular, responsável pela alteração da freqüência fundamental do timbre executando, tanto para mais quanto para menos.

Praticamente todos os sintetizadores modernos permitem inclusive selecionar a extensão do Pitch Bend, podendo- se ir de meio tom até uma oitava ou mais.

6.1.2.7 Modulação

Quando o Pitch Bend é em forma de alavanca, a Modulação normalmente está junto dele. Se o controle do Pitch Bend é por meio de acionador circular, a Modulação fica ao seu lado, em outro controle do mesmo tipo. O efeito da Modulação é análogo ao do Aftertouch, podendo-se até dizer que este último é a transposição do comando de Modulação para o teclado.

6.1.2.8 Pedais de controle

Podem ser divididos em dois grupos - os de acionamento ON/OFF (liga/desliga) e os de acionamento variável.

Os do tipo ON/OFF são usados normalmente para "sustain", para troca de patches e outros efeitos que possam ser comandados pela simples ação de ligar ou desligar. Já os de acionamento variável são usados no controle de volume, de modulação e seus derivados - vibrato, trêmolo e wah-wah -, bem como qualquer outro efeito que exija o controle de forma contínua.

6.1.3 Tipos de equipamentos

Instrumentos musicais eletrônicos	Estações Musicais Sintetizadores Teclados Arranjadores Módulos de Som Guitarra Sintetizadora Sampler Bateria eletrônica Computador + Software para Performance
Equipamentos musicais	Seqüenciadores Computadores Processadores de Sinais Mesas de Mixagem Gravadores digitais MIDI Patchbays Computador + Software para Gravação

6.1.3.1 SINTETIZADORES

Os sintetizadores são instrumentos eletrônicos que revolucionaram a música por possuírem a possibilidade de criar novos sons e alterar sons pré-existentes. Os sintetizadores podem ser classificados em analógicos, digitais e híbridos.

Os sintetizadores analógicos foram os primeiros a surgir e utilizavam osciladores controlados por tensão elétrica para fazer a alteração do som. Os primeiros eram monofônicos e em geral possuiam vários potenciômetros para o músico realizar a síntese em tempo real. Possuiam nenhuma ou pouca memória para armazenamento dos "patchs" programados. Podemos citar como exemplo os sintetizadores analógicos: Minimoog, Korg MS-20, Roland System 100. São exemplos de sintetizadores analógicos polifônicos: Yamaha CS-80, Prophet-5.

Os sintetizadores híbridos possuiam parte das funções controladas por tensão e parte controlada por um processador digital. Os sons produzidos pelos sintetizadores do início da década de 80 como o Roland JX-3P e o Roland Júpiter 8 são exemplos de sintetizadores híbridos. Estes sintetizadores já não apresentavam flutuações e oscilações na freqüência, e sendo assim, mantinham a afinação constante.

Nos sintetizadores digitais o som é representado por uma seqüência de números (amostras - "samples"). O sintetizador possui uma memória com amostras digitais das ondas sonoras responsáveis pelo timbre de cada instrumento a ser criado.

Ex: Nordlead III, Novation SuperNova II, Korg MS-2000, Minimoog, Roland JP-8000, Moog Modular.

6.1.3.2 ESTAÇÕES MUSICAIS ("MUSIC WORKSTATIONS")

As estações musicais são teclados que incorporam pelo menos três elementos básicos:

Sintetizador
Seqüenciador
Processador de efeitos

A primeira workstation de sucesso foi o KORG M1 MusicWorkstation em 1989 que incorporou estes três elementos utilizando a tecnologia digital. Devido a grande vendagem desse produto outros fabricantes seguiram a mesma fórmula adotada pela Korg e vários novos modelos foram sendo fabricados por quase todas as marcas que já haviam lançado sintetizadores digitais [RAT 97].

O conceito de estação de trabalho musical pode ser ampliado para um computador com um software seqüenciador com conexão MIDI para um teclado multitimbral. Porém o termo é mais utilizado para teclados que constituim-se verdadeiros estúdios de produção musical como o Roland XP-60, Korg Triton, Kurzweil K2600X, e o Ensoniq ZR-76. Algumas workstations ainda mais completas possuem ainda outros recursos como Arpejador automático, "drum machine" e sampler.

6.1.3.3 Teclados Arranjadores

No princípio eles eram chamados de "órgãos eletrônicos". Depois, com a inclusão de ritmos eletrônicos, passaram a ser chamados de "órgãos com acompanhamento automático". Hoje em dia, com a imensa gama de timbres e variados recursos que possuem, são genericamente chamados de "teclados arranjadores", ou simplesmente teclados. De qualquer forma, pode-se dizer que os teclados arranjadores se tornaram o instrumento musical familiar moderno, assim como o piano o foi por muito tempo – muitas pessoas se aproximaram da música ao tomar contato com um destes instrumentos.

De um modo geral, um bom teclado atual apresenta os seguintes recursos de fábrica:

- Timbres (presets) padrão General MIDI;
- Seqüenciadores;
- Recursos para a edição de sons, permitindo a alteração dos sons originais;
- Processadores de efeitos (normalmente incluindo os efeitos chorus, reverber e delay, entre outros);

Acompanhamentos automáticos, que nada mais são do que seqüências de padrões rítmicos, harmônicos e melódicos agrupados por "estilos" musicais (dance music, samba, jazz latino, etc.);

Dispositivos de leitura/gravação de discos flexíveis (drive) de 3 ½ pol., permitindo o armazenamento de novos sons criados pelo usuário, acompanhamentos, ritmos e configurações genéricas.

Ex: Roland VA-7, Roland G-1000.

6.1.3.4 Bateria Eletrônica ("Drum Machines")

Baterias Eletrônicas são módulos de som especializados destinados à produção de sons percussivos (como bateria e percussão em geral) e partes rítmicas. Assim como outros módulos de som, elas podem empregar sons sintetizados ou "sampleados" (digitalizados), sendo que os sons gerados pela bateria eletrônica são específicos para a emulação de diversos tipos de percussão.

Uma bateria eletrônica típica pode fornecer um bom número de instrumentos simulados, como caixas, tom-tons, bumbos, pelo menos um chipô (aberto e fechado), pratos de ataque, pandeiros, etc. Além disso, praticamente todas as drum machines possuem um seqüenciador incorporado, permitindo a edição de pequenas seções

rítmicas (normalmente de 1 a 4 compassos). O seqüenciador pode então executar o trecho criado de forma contínua, gerando um padrão rítmico (pattern); a partir da união seqüencial de vários padrões pode-se criar uma trilha (“song”) e armazená-la na memória ou disco para uso posterior [RAT 97].

Toda bateria eletrônica possui teclas ou “pads”, utilizadas para se armazenar as seqüências ou mesmo para se executar os sons em tempo real. Baterias eletrônicas de boa qualidade possui teclas ou “pads” sensíveis (Velocity-sensitive) a diferentes pressões, o que ocasiona uma resposta sonora mais realista, já que assim pode-se controlar a dinâmica dos ataques. Estes valores de dinâmica, bem como o instrumento simulado (ou seja, qual peça de percussão está sendo tocada) são representados por um número no padrão MIDI, o que significa que os mesmos podem ser emitidos por qualquer instrumento MIDI compatível, como um piano digital, por exemplo.

Ex: Roland TR 909, Roland DR-770.

6.1.3.5 Sampler

Os “samplers" trabalham com amostras do som obtidas através de um processo de digitalização sonora onde uma onda sonora analógica é transformada em digital. Posteriormente ao som ter sido amostrado, ele pode ser alterado, sofrer uma conversão digital-analógica e ser reproduzido. Muitos equipamentos musicais, após o processo de PCM (“Pulse Code Modulation”) ter sido desenvolvido, começaram a trabalhar com sons amostrados por ter uma fidelidade maior que os sons puramente sintetizados. Os equipamentos mais utilizados atualmente são chamados de “Music Workstations" e reúnem todos os requisitos básicos para um profissional fazer qualquer tipo de produção musical. Estas estações musicais não deixam de ser computadores dedicados à música e incorporam, na maioria das vezes, um processador de efeitos, um seqüenciador e um sintetizador que trabalham sobre sons PCM amostrados. Tudo é controlado pelo usuário através de uma tela de cristal líquido com comandos sob a forma de menu.

Ex. Korg Triton, Kurzweil K2600, S600 Studio Sampler AKAI, Samplecell II, Gigasampler.

6.1.3.6 Controladores MIDI

Os controladores MIDI são equipamentos destinados a enviar mensagens MIDI para uma fonte sonora. São projetados para servirem como interface sonora entre o músico e o módulo de som. Os controladores mais comuns são teclados, geralmente sem geradores de som, que são utilizados em estúdio, performances, etc. Porém existem vários controladores MIDI sob a forma de outros instrumentos como pedaleiras MIDI, Violoncelo, Violino, Instrumentos de sopro, etc.

Os controladores MIDI para sopro oferecem o mesmo aspecto do instrumento de sopro acústico e características aproximadas para que o músico deste instrumento possa utilizá-lo de maneira similar empregando sua técnica e expressividade particular do instrumento de sopro dificilmente obtida a partir de teclados. Eles permitem que os saxofonistas e clarinetistas utilizem módulo específicos e até mesmo sintetizadores que possuam conexão para controladores MIDI de sopro.

Os controladores MIDI para guitarra, também chamados de guitarras sintetizadoras. O adaptador MIDI pode ser colocado numa guitarra convencional onde receptores ficam por debaixo das cordas da guitarra. Um cabo fino faz a ligação entre o instrumento e o módulo de som. Isso possibilita que o guitarrista execute sua técnica através de seu próprio instrumento ao invés de realizar seu trabalho com teclados.Também existem guitarras com um design futurista específicas para este tipo de conexão MIDI.

Controladores MIDI para Quarteto de cordas podem ser adaptadores instalados no instrumentos ou instrumentos específicos destinados à conexão MIDI. O sistema ZETA MIDI é muito empregado em instrumentos de corda. A interface Zeta recebe sinais de áudio e analiza os sinais em termos de altura e amplitude. Os dados são processados e enviados para qualquer módulo multitimbral ou sintetizador. É possível também selecionar diferentes canais MIDI para cada corda obtendo um timbre diferente para cada corda.

Os controladores MIDI para percussão já são muito comuns tanto em estúdio quanto em espetáculos pela facilidade de apresentar os sons prontos dispensando horas no processo de captação e mixagem de instrumentos de percussão acústicos. Como os sons são amostrados digitalmente ou calculados por modelagem física, aproximam-se muito dos reais. Outra grande vantagem na sua utilização é que outros sons além dos convencionais podem ser utilizados de acordo com o gerador de som utilizado. Geralmente nas baterias convencionais utiliza-se “triggers” para o reconhecimento do toque do baterista que são transmitidos ao módulo de som dedicado a esta função. Também são muito utilizadas baterias eletrônicas específicas para ligação com módulos de som para a percussão. Estas baterias possuim “Pads” que substituim todas as peças originais para a bateria, incluindo os pratos. Uma utilização muito comum em espetáculos é a bateria híbrida, onde algumas peças são amplificadas e outras são “trigadas” ou então substituídas por “Pads”. Porém nem todas as baterias digitais vem com componentes separados. Alguns controladores MIDI foram feitos sob a forma de pequenos “Pads” agrupados numa única unidade.

Conversores de altura para MIDI (Pitch-to-MIDI converters) são uma categoria de equipamentos que possibilitam a conversão da voz humana, piano, harpa e outros instrumentos acústicos em mensagens MIDI. MidiVox é um produto inovador que funciona melhor que os demais Pitch-toMIDI converters porque, ao contrário de converter audio digital em MIDI ele possui um biosensor que determina o volume e a freqüência que da voz do cantor convertendo esta informação para Mensagens MIDI. Possui opções para voz masculina e fenina/criança, dois tipos de volume um variante e outro fixo.

Pedaleira MIDI é um equipamento que simula a pedaleira convencional de órgão apresentando uma uma mais oitavas e capacidade de conexão MIDI a qualquer módulo de som. Dessa forma é possível adicionar linhas de baixo com a utilização dos pés na música que também está sendo executada pelas mãos. Geralmente estes instrumentos possuem oitavador e sistema para envio de mudança de programa.

Exitem outros controladores MIDI mais exóticos como o WaveAccess que possibilita a captação de sinais do corpo provindos do cérebro, coração e músculos que são enviados ao computador, processados e transformados em mensagens MIDI. A Harpa Laser também possibilita o reconhecimento da interrupção do fluxo do raio pela mão transformado em mensagem MIDI que pode ser enviada ao módulo de som. Dispositivos adaptados no corpo humano também possibilitam a leitura dos movimentos da dança e a trasnformação dos dados obtidos em eventos MIDI.

Exemplos:

Teclados	Roland AX-1, Fatar Studio 2001, Roland A-33.
Sopro	Yamaha WX11, Akai EWI3000.
Guitarra	Roland GR-09.
Cordas	Zeta MIDI System
Triggers para bateria	Roland PD-7, Yamaha EP75, Roland PAD-80 Octapad II, Drum Kit 3.5
Pedaleira MIDI	FATAR MP-1

6.1.3.7 Sequenciador

Um seqüenciador é um dispositivo de gravação e edição de eventos musicais, em geral MIDI. Alguns seqüenciadores existem sob a forma de software para micro- computadores enquanto outros são construídos em hardware. Os seqüenciadores ("sequencers") podem estar embutidos num teclado e servir como um gravador digital para as músicas a serem registradas. Existem também aparelhos dedicados ao seqüenciamento que servem apenas para esta finalidade. Geralmente são usados ligados via MIDI a outros instrumentos como sintetizadores, “samplers", baterias eletrônicas ("drum machine"). O seqüenciador pode gravar o que está sendo tocado nestes instrumentos e depois pode executar automaticamente através de mensagens MIDI enviadas para os instrumentos interligados. Atualmente é um dos recursos tecnológicos da música mais utilizados em estúdios de gravação e produção musical devido à facilidade e ao baixo custo.

Ex. Roland MC-50, Cubave VST Score, seqüenciador de MusicWorkstations Ensoniq ZR-76.

6.1.4 MIDI

MIDI é a sigla de *Musical Instrument Digital Interface*, um padrão de comunicação de dados criado em 1983 por um acordo entre diversos fabricantes de instrumentos musicais norte-americanos e japoneses, para possibilitar a transferência de informações entre instrumentos musicais e computadores.

O padrão MIDI usa a moderna tecnologia digital, codificando as informações em dados binários (bits) que são transferidos por meio de uma linha física (cabo MIDI) de um equipamento para outro. Todos os detalhes técnicos sobre os códigos e os circuitos básicos necessários à comunicação MIDI são definidos na *MIDI Specification 1.0,* publicada pela lnternational MIDI Association (EUA).

Embora o objetivo principal do MIDI seja transferir informações de caráter musical, tais como a execução de notas e outros controles de expressão, seu uso tem sido expandido de tal forma que atualmente o MIDI é usado não só para a transferência de dados relativos à execução musical propriamente dita, mas também para outras aplicações de alguma forma correlatas à música, como controle de iluminação e equipamentos de áudio [LEH 93].

Na forma mais simples de se usar MIDI, um músico pode controlar um teclado a partir de outro (controle remoto), pois todas as suas atitudes de execução musical são codificadas e transmitidas pelo cabo MIDI para o outro teclado, que recebe, decodifica e executa aquelas atitudes. Assim, numa implementação bastante simples, é possível tocar dois ou mais instrumentos a partir de um deles, o que possibilita, entre outras coisas, a superposição dos sons dos vários instrumentos controlados simultaneamente.

É importante frisar que as informações MIDI transferidas de um equipamento (mestre) para outros (escravos) não carregam consigo nenhum sinal de áudio (som). Essas informações são mensagens que codificam as ações do músico, tais como o ato de pressionar uma tecla, ou o de soltar uma tecla. O som (áudio) conseqüente do ato de pressionar uma tecla não é transferido via MIDI, mas sim gerado em cada um dos instrumentos da cadeia. Dessa forma, quando um instrumento "mestre" comanda outro via MIDI, basicamente ele "diz" ao outro quais as notas que devem ser tocadas, mas o som (áudio) do segundo instrumento será determinado pelo próprio "escravo".

6.1.4.1 Principais características do sistema MIDI

- Sistema digital de comunicação de dados.
- Transmissão serial assíncrona.
- Taxa de transmissão: 31.250 bits/Seg.
- Comunicação unilateral.
- Permite padronizar a comunicação entre os instrumentos (protocolo MIDI).
- Facilita a ligação entre instrumentos e computadores.
- Permite a edição de qualquer evento musical através de software.
- Permite tocar vários instrumentos simultaneamente.
- Permite gerar arquivos MIDI Standard.

6.1.4.2 Evento MIDI

Um evento MIDI é gerado pela ação do músico ao tocar o instrumento MIDI.

Figura 42 – Evento MIDI

Ao ser pressionada uma tecla, além do instrumento MIDI soar a nota com o timbre previamente escolhido, ele codifica o evento em bytes e envia os bits serialmente para a porta MIDI OUT.

Figura 43 – Evento em bytes

Os bits são formados por apenas dois números: o zero e o um. Por isso utiliza-se a base binária para representação de informação digital.

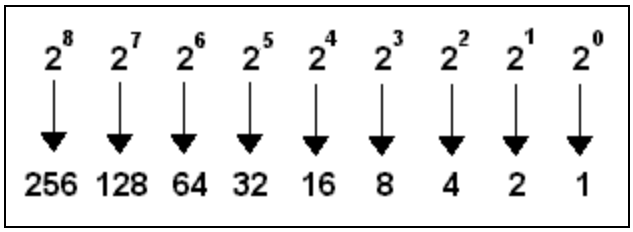


Figura 44 – Representação em binário

Exemplo de números em binário:
0 = 0
1 = 1
10 = 2
11 = 3
100 = 4
101 = 5
110 = 6
111 = 7
1000 = 8
1001 = 9
1010 = 10

Exemplo de conversão de binário para decimal:
$[10101]_2 = 16+4+1=21= \ [21]_{10}$

Exemplo de conversão de decimal para binário

$[17]_{10}$ = divisões sucessivas do 17 por 2. O resto da divisão, no sentido contrário, é o n º em binário = $[10001]_2$

A Base Hexadecimal

Formada por 16 símbolos: 0 1 2 3 4 5 6 7 8 9 A B C D E F

A Base Hexadecimal é muito utilizada para a representação numérica digital pois, além de ser múltiplo de dois, ela utiliza mais símbolos para representar os números diminuindo , portanto, a utilização de casas para representar a informação.

A3h = $(Ax16^1) + (3x16^0) = [163]_{10}$

Converter da base Hexadecimal para a base Binária

ABCh = $[1010\ 1011\ 1100]_2$

A B Ch
/ \ \
1010 1011 1100

Converter da base Binária para a base Hexadecimal

[101 0111]2 = 57h

$[101\ 0111]_2$
/ \
5 7 h

Exemplos de conversão de bases numéricas:

a)$[110110]_2$ = 32+16+4+2=54 = $[54]_{10}$

b)$[AB]_{16} = (10x16^1)+(11x16^0) = [171]_{10}$

c)$[11010101]_2$ = (13) (5) = $[D5]_{16}$

d)$[101010]_2$ = (2)(10) = $[2A]_{16}$

6.1.4.3 Mensagem MIDI

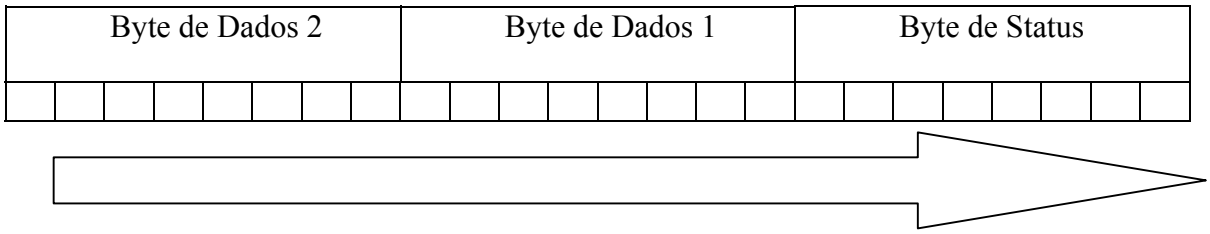


Figura 45 – Mensagem MIDI

TRANSMISSÃO SERIAL ASSÍNCRONA 31.250 bits / seg

As informaçõessão transmitidas por mensagens digitais, cujo tamanho (em bytes – conjunto de 8 bits) varia, dependendo da finalidade. Todas as mensagens contém um código inicial, chamado de *Byte de Status,* que determina o teor da mensagem. Algumas mensagens contêm ainda um ou mais *Bytes de Dados,* que complementam a informação que está sendo transmitida [PEN 95].

6.1.4.4 LIGAÇÕES

Os conectores MIDI são 3:

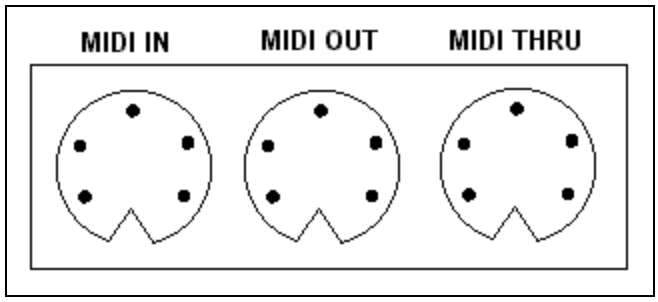


Figura 46 – Conectores MIDI

MIDI IN	É uma entrada, por onde o equipamento recebe as mensagens;
MIDI OUT	É uma saída, por onde o equipamento transmite as mensagens geradas por ele próprio;
MIDI THRU	É uma saída, por onde o equipamento retransmite as mensagens recebidas por MIDI IN (não geradas por ele).

Cabos MIDI:

- Tomadas do tipo DIN de 5 pinos.
- Cabos blindados, com 2 condutores e a malha de blindagem.
- Cabos com mais de 15 metros podem trazer problemas de perda de sinal.

6.1.4.5 Limite da transmissão MIDI:

Quando se utiliza um grande número de equipamentos conectados em cadeia e um volume considerável de mensagens, poderá ocorrer atrasos na transmissão de eventos.

Exemplo de conexão MIDI:

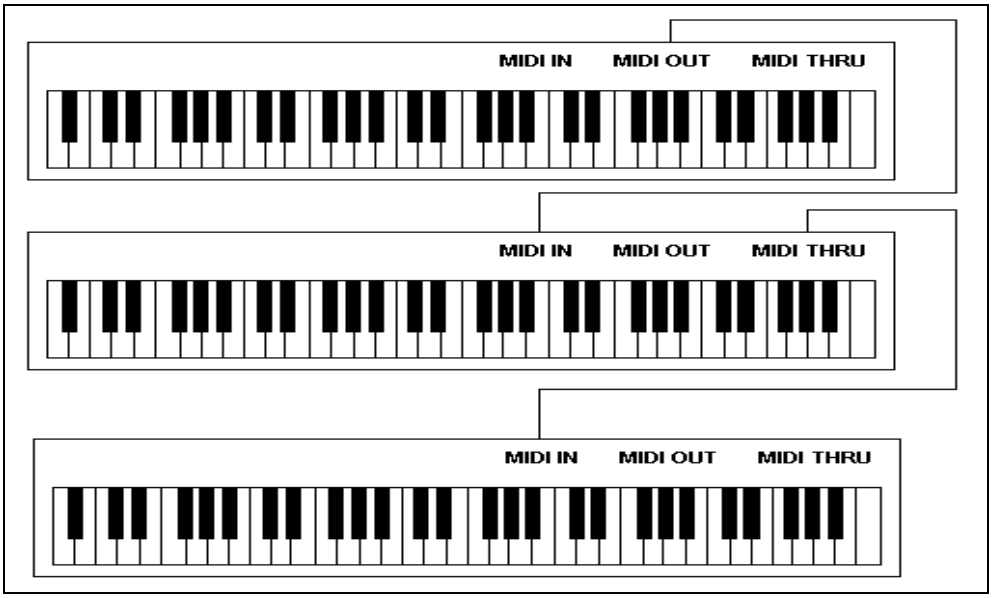


Figura 47 – Exemplo de Conexão MIDI

A ligação entre os instrumentos pode ser feita da seguinte forma. O instrumento mestre transmite por MIDI OUT os dados gerados nele próprio (gerados pela execução do músico em seu teclado). O instrumento escravo recebe os dados do mestre através de sua entrada MIDI IN, executa os comandos se estiver habilitado para isso e, executando ou não os comandos, retransmite por MIDI THRU todas as mensagens que recebe. O instrumento escravo 2, por sua vez, recebe por MIDI IN as mensagens do mestre (que passaram pelo escravo 1). Se houvesse mais instrumentos, estes poderiam ser incluídos sucessivamente no encadeamento, usando o esquema de ligação THRU-IN-THRU-IN, etc. Além disso, nada impede que um desses escravos possa ser o mestre de um encadeamento secundário.

Como podemos perceber, a concepção do encadeamento MIDI permite a implementação de sistemas complexos, com vários instrumentos encadeados em série, comandados por um deles [PEN 95].

6.1.4.6 Canais MIDI

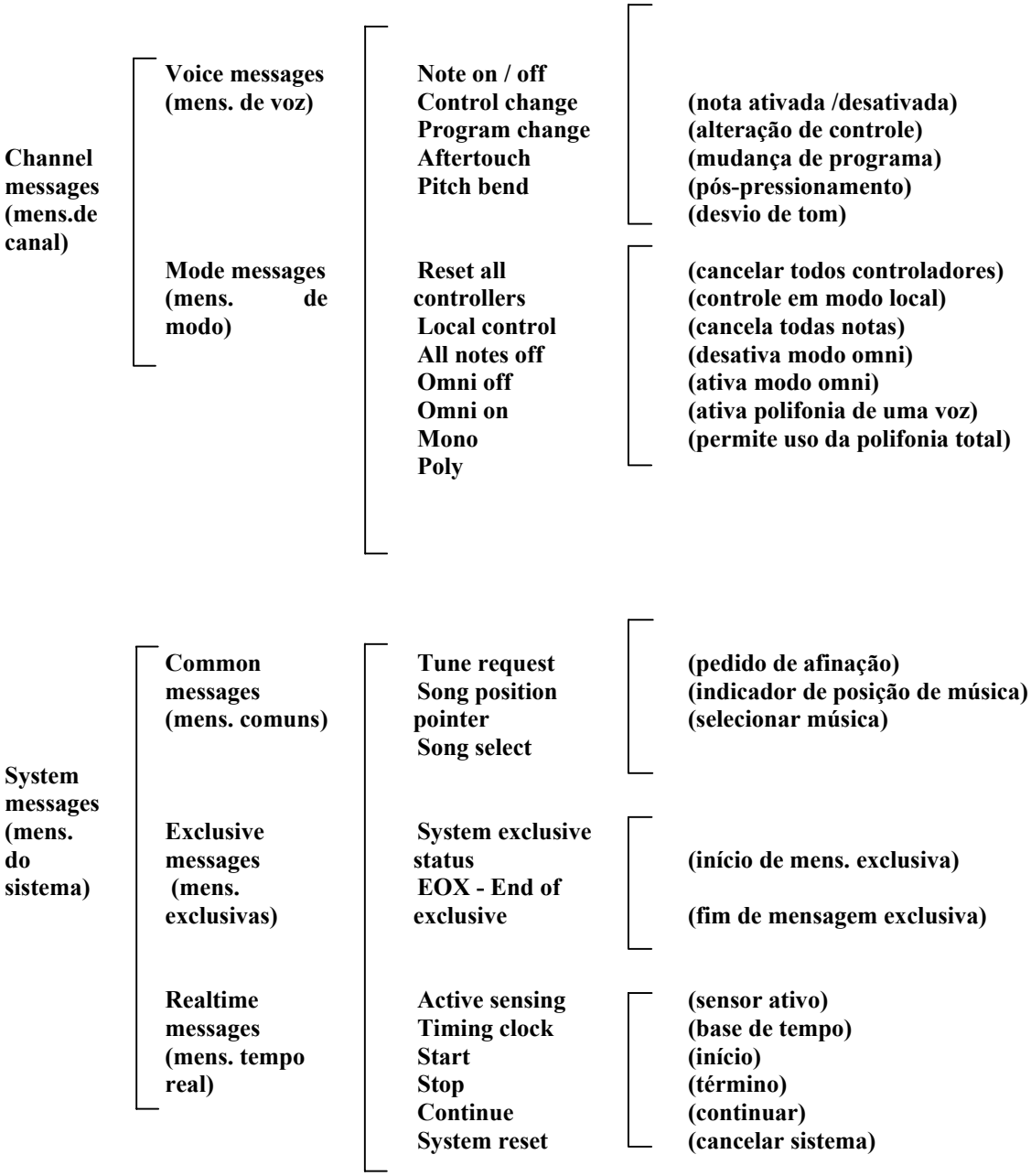
Um dos recursos mais poderosos do MIDI é a possibilidade de se endereçar mensagens para determinados equipamentos, de uma forma discriminada. Isso é feito através de 16 canais de MIDI, algo que funciona de forma semelhante aos canais de televisão. Quando um instrumento transmite uma mensagem (por exemplo, uma nota pressionada), inclui nesta um código que identifica um canal (no caso, um canal de transmissão). Para que um outro instrumento possa executar o comando contido naquela mensagem, é necessário, em princípio, que ele esteja ajustado para receber no

canal de mesmo número daquele codificado na mensagem. Se isso não ocorrer, o instrumento ignora a mensagem, pois ela não está destinada a ele (ao seu canal de recepção). Com os 16 canais de MIDI, pode-se transmitir informações diferentes para até 16 instrumentos, simultaneamente, através do mesmo cabo, de forma que é possível ter-se até 16 instrumentos comandados por um mestre (um seqüenciador, por exemplo) que lhes transmite todas as notas a serem executadas, como se fossem 16 músicos tocando seus instrumentos, individualmente [YOU 96]. Algumas interfaces , entretanto, possuem mais do que um par de conectores, oferecendo múltiplas saídas que permitem expandir além do limite dos 16 canais, possibilitando ter-se um sistema de 16 canais em cada uma das saídas MIDI Out. Dessa forma, usando-se uma interface com duas saídas MIDI Out, por exemplo, pode-se endereçar até 32 instrumentos diferentes, tocando notas totalmente independentes, pois uma das saídas constitui-se em um sistema próprio de 16 canais (pode-se ter um equipamento A conectado à MIDI Out 1 e recebendo pelo canal 1, e ter um equipamento B recebendo outras informações diferentes também pelo canal 1, mas pela MIDI Out 2). É importante certificar-se se a interface possui realmente saídas *independentes,* pois pode acontecer das MIDI Out serem duplicadas, isto é: os mesmos dados são transmitidos igualmente por todas as MIDI Out (esse último caso não caracteriza *saídas múltiplas).* Evidentemente, para se operar as saídas múltiplas é necessário que o software usado suporte tal recurso (o que acontece com a maioria deles) [PEN 95].

6.1.4.7 lnterfaces MIDI

Existem dois tipos de interfaces MIDI: internas e externas. Uma interface interna é a placa eletrônica que é inserida em um dos conectores (slots) internos de expansão, por onde o computador entrega e acessa dados na interface. Na face externa da placa estão os conectores MIDI, que permitem interligá-la aos demais instrumentos musicais [CAM 97]. Na interface externa, por sua vez, o circuito eletrônico está contido em uma pequena caixa, que, dependendo do modelo, é ligada ao computador através da porta paralela (normalmente usada pelo mouse ou pelo modem). As interfaces internas em geral são mais baratas, pois dispensam acabamento (caixa, pintura, etc), enquanto as interfaces externas são a solução para usuários de computadores portáteis (laptops e notebooks), que não dispõem de conectores internos de expansão e não podem por isso usar placas internas [LEH 93].

6.1.4.8 Classificação dos Eventos MIDI (MIDI Especification 1.0)



6.1.4.9 Classificação Alternativa

Segundo [RAT 97] é possível apresentar as mensagens MIDI através de outra classificação:

Mensagens de Execução Musical – Mensagens de Voz
Mensagens de Auxílio à Execução – Mensagens de Modo
Mensagens de Tempo Real
Mensagens Comuns
Mensagens de Transferência de Sons – Mensagens Exclusivas (Sys-EX)

As Mensagens de Canal

São formadas por eventos relacionados diretamente à execução musical, como o ato de pressionar e soltar teclas, pedais (de sustain e de volume), as variações de volume e de posição do som no estéreo, a seleção de timbres, etc. Essas mensagens são transmitidas usando-se *canais de MIDI,* o que possibilita o endereçamento individual das mensagens.

Mensagens de Voz

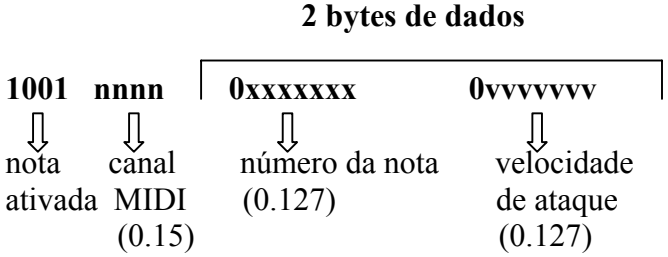
São as mais usadas, pois carregam as informações referentes à execução musical (notas, pedais, etc) e outros comandos diretamente ligados à execução (variação de volume, seleção de timbres, etc).

Liga Nota - Note On: informação transmitida quando uma tecla é pressionada, iniciando a execução de uma nota.

O primeiro byte de dados corresponde ao número da nota que foi pressionada pelo músico. Os valores vão de 0 até 127 sendo que o 60 é o Do central do piano (C4). O número 127 corresponde a nota Sol mais aguda do piano (G9).

O segundo byte de dados informa a velocidade (intensidade) com que a nota foi pressionada.

Formato da Mensagem



Runnig Status

Quando os BYTES DE STATUS são idênticos entre as mensagens MIDI, ela tornam-se redundantes acarretando um desperdício em tempo de transmissão. Foi estipulado pelo padrão MIDI que se o BYTE DE STATUS atual é igual ao último byte transmitido, ele pode ser suprimido da mensagem, sendo enviado apenas o BYTE DE DADOS [LEH 93].

Ex.: SEM RUNNING STATUS	COM RUNNING STATUS
6.1.4.9.1.1 NOTE ON 1	60 64 NOTE ON 1
60 64	
NOTE ON 1 52 100	52 100
NOTE ON 1 38 64	38 64
	ECONOMIZA 2 BYTES

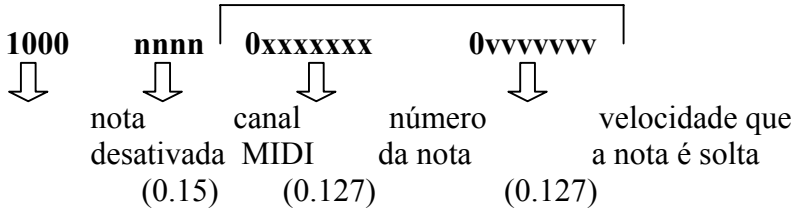
Nota ativada com velocidade = 0 é interpretada como uma mensagem de nota desativada. Trabalhando com Running Status isso possibilita a economia de tempo de transmissão pois ao invés de se usar outro BYTE DE STATUS para indicar a nota destivada, pode-se utilizar a nota ativada com velocidade = 0.

EX.: NOTE ON 1 60 100 – LIGA NOTA
NOTE ON 1 60 0 – DESLIGA NOTA

COM RUNNING STATUS:
NOTE ON 1 60 100
60 0
ECONOMIZA UM BYTE

Desliga Nota - Note Off. informação transmitida quando uma tecla é solta, finalizando a execução de uma nota [RAT 92].

Formato da mensagem



Os equipamentos MIDI devem reconhecer tanto a nota ativada com velocidade 0 quanto o evento da nota desativada.

Mudança de Controle (Control Change)

Transmitida quando um dispositivo de controle (volume, pan, pedal, etc) é acionado; informa a posição do controle (de 0 a 127) [LEH 93].

Os dois tipos de mensagem de mudança de controle são:

Controles Contínuos (pedal de volume)

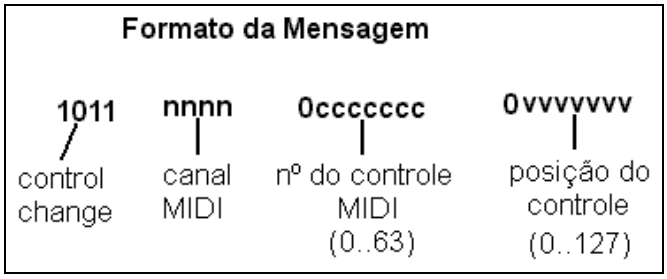


Figura 48 – Controles Contínuos (pedal de volume)

Liga/Desliga (pedal sustain)

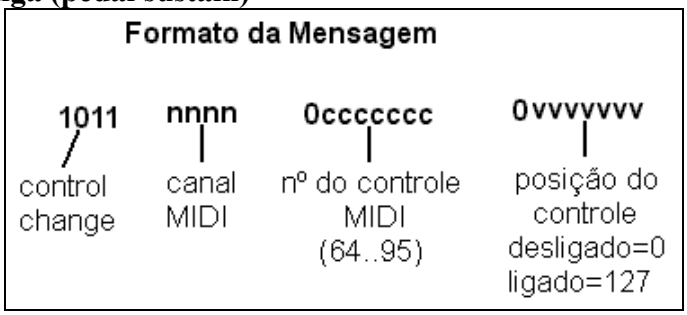


Figura 49 – Controles Contínuos (pedal sustain)

Exemplo de controle contínuo

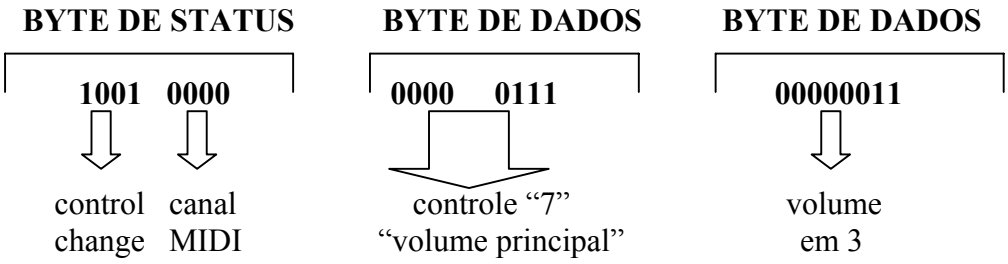


Tabela 11 - Mensagens de Control Change

CÓDIGO	DESCRIÇÃO DO CONTROLE
0	Seleção de Banco de Programas – MSB (bank select)
1	Roda/Alavanca de Modulação – MSB (modulation wheel)
2	Controle por Sopro - MSB (breath control)
4	Pedal - MSB (foot controller)
5	Tempo do Portamento - MSB (portamento time)
6	Entrada de Dados – MSB (data entry)
7	Volume Principal – MSB (main volume)
8	Equilíbrio - MSB (balance)
10	Pan - MSB (pan)
11	Controle de Expressão - MSB (expression controller)
16-19	Controles de Uso Geral 1 a 4 - MSB (general purpose)
20-63	LSB dos controles de códigos 0 a 31
64	Pedal de Sustain (sustain pedal)
65	Liga/Desliga Portamento (portamento on/of)
66	Pedal de Sustenuto (sustenuto pedal)
67	Pedal Abafador (soft pedal)
69	Pedal de Sustain 2 (hold 2)
80-83	Chaves de Uso Geral 5 a 8 (general purpose)
91	Profundidade de Efeito Externo (extemal effect depth)
92	Profundidade do Tremolo (tremolo depth)
93	Profundidade do Chorus (chorus depth)
94	Profundidade do Batimento (celeste detune depth)
95	Profundidade do Phaser (phaser depth)
96	Incremento (data increment)
97	Decremento (data decrement)
98	Número do Parâmetro - LSB (parameter number)
99	Número do Parâmetro – MSB (parameter number)
100	Número do Parâmetro - LSB (parameter number)
101	Número do Parâmetro - MSB (parameter number)

Mudança de Programa - Program Change: esta mensagem **é** transmitida quando é selecionado um timbre (piano, flauta, baixo, etc).

Patch = preset = program = tone = sound = voice são denominações empregadas para o conceito de programa e podem ser encarados, dependendo do fabricante, como sinônimos. A mudança de programa serve para trocar o preset do instrumento. Quando é escolhido um novo som, é gerado uma mensagem MIDI correspondente e enviada ao MIDI OUT [LEH 93].

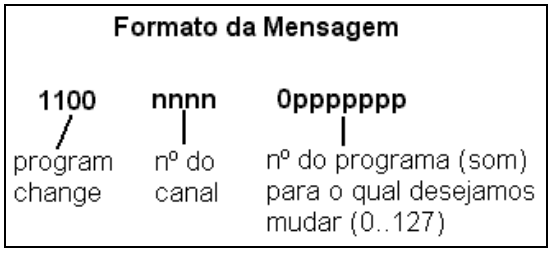


Figura 50 – Formato da Mensagem

Alguns fabricantes de sintetizadores iniciam a numeração de seus programas com o número 1 ao invés de 0. Dessa forma os timbres estarão deslocados de uma unidade em relação à mudança de programa recebida. Por exemplo, ao receber um program change = 10, o sintetizador irá selecionar o programa número 11.

O código MIDI permite a seleção de um programa num banco com 128, porém os instrumentos modernos possuem bem mais que 128 sons. Para solucionar este problema pode-se criar bancos de som com até no máximo 128 programas que poderão ser acessados por meio de dois comandos de program change. O primeiro comando vai selecionar o banco e o segundo o programa. Pode-se utilizar também a General MIDI que estipula 128 sons [LEH 93].

Evento de Pressão no Teclado (channel aftertouch)

É transmitida quando é feita pressão sobre o teclado, com alguma tecla já pressionada (esse controle pode ser usado para diversas finalidades); informa a pressão global feita sobre o teclado. O aftertouch pode ser programado para controlar a intensidade, controlar o oscilador de modulação, etc.

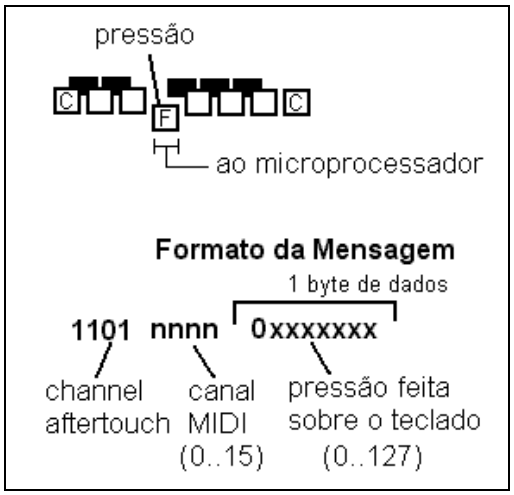


Figura 51 – Evento de pressão no teclado

Pressão de tecla individual (polyphonic aftertouch)

Transmite informações a respeito dos valores de pressão detectados por sensores individuais em cada tecla, indicando a pressão individual sobre a tecla. Esta mensagem é composta de três partes. Além do canal MIDI, da nota, transmite também a pressão exercida sobre a tecla que foi pressionada.

O Polyphonic Aftertouch acarreta uma grande quantidade de bytes a mais na transmissão MIDI pois cada nota possui uma própria representação da pressão, por isso muitos equipamentos limitam-se a usar o CHANNEL AFTERTOUCH [RAT 92].

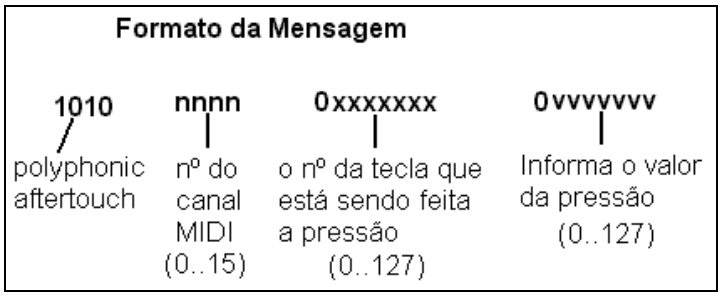


Figura 52 – Pressão de tecla individual

Variação do Pitchbender – Pitch Bender change: é uma função dos instrumentos eletrônicos que produz uma alteração na afinação da nota, variando a sua frequência. Quando a alavanca é movida ela gera um evento MIDI com o valor da alteração da posição da alavanca. [RAT 92].

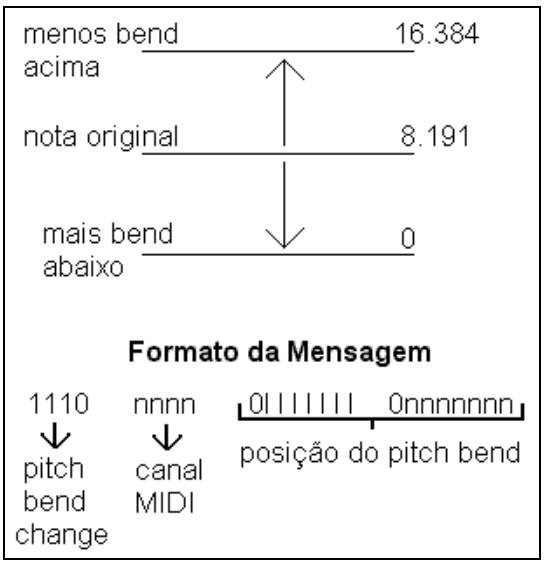


Figura 53 – Variação do Pitchbender

Mensagens de Modo

São usadas quando se deseja fazer com que um instrumento mude seu modo de operação.

Controle Local (local control)

Serve para mandar o instrumento controlado desconectar ou conectar seu teclado às suas próprias vozes [RAT 92].

Formato da Mensagem

1011	**nnnn**	**01111010**	**0vvvvvvv**
⇩	⇩	⇩	⇩
mesmo da msg. de control change	canal MIDI (básico)	desativa notas	se v for = 0 desabilita se v for = 127 habilita

Desativa Notas (all notes off)

Todas as notas que estejam soando em um instrumento irão parar repentinamente de soar. Serve para cessar notas que estejam sendo executadas pela perda do controle das mensagens normais de note off [LEH 93].

Formato da mensagem

1011	**nnnn**	**01111011**	**0vvvvvvv**
⇩	⇩	⇩	⇩
mesmo da msg. de control change	canal MIDI (básico)	desativa notas	v = 0

OMNI OFF/ON MONO/POLY

Os modos de operação são quatro, como resultado da combinação de dois parâ*metros: Poly / Mono e Omni On*/Off.

O primeiro define como o instrumento executará as notas que lhe forem enviadas via MIDI: caso esteja atuando como Poly (polifônico), ele poderá executar notas simultâneas (por exemplo: acordes); caso esteja atuando como *Mono* (monofônico), só poderá tocar uma nota de cada vez [RAT 92].

O segundo parâmetro define a forma como o instrumento interpreta os canais: caso esteja em *Omni On,* executará qualquer mensagem de canal, independentemente do número do seu canal de recepção; caso esteja em *Omni* Off, somente executará as mensagens que lhe forem transmitidas no canal de mesmo número do seu canal. Assim, os quatro modos MIDI possíveis são definidos como:

Modo 1: Omni On / Poly
Modo 2: Omni On / Mono
Modo 3: Omni Off / Poly
Modo 4: Omni Off / Mono

As mensagens de modo são o resultado da combinação de dois perâmetros POLY/MONO e OMNI ON/OFF onde o BYTE DE STATUS permanece sempre o mesmo e o BYTE DE DADOS varia conforme segue:

POLY – Polifônico (o instrumento fica apto a executar as notas simultâneamente).
Formato do Byte de Dados – 01111111 0vvvvvvv = 0

MONO – *Monofônico (o instrumento só pode tocar uma nota por vez).*
Formato do Byte de Dados – 01111110 0vvvvvvv = 0

OMNI ON – *Executará qualquer mensagem de canal, independente do número do canal de recepção.*
Formato do Byte de Dados – 01111101 0vvvvvvv = 0

OMNI OFF – *Executará somente as mensagens que lhe forem transmitidas no canal de mesmo número do seu canal de recepção.*
Formato do Byte de Dados – 01111100 0vvvvvvv = 0

Mensagens do Sistema

Tratam de eventos não-musicais, mas de relevante importância para o funcionamento de um sistema MIDI *complexo.*

Mensagens Comuns (Common Messages)

São mensagens utilizadas quando se deseja controlar vários equipamentos que estejam executando a mesma música, como é o caso típico dos seqüenciadores.

Pedido de Afinação (Tune Request) – 111 0110 - Controla a afinação de um instrumento MIDI de forma remota. Muito utilizada nos sintetizadores com o a afinação controlada por voltagem (VCO) que necessitavam receber comandos para afinar ou mudar a afinação nos intervalos de uma performance.

Indicador de Posição de Música (Song Position Pointer) – 1111 0010 + 2 Bytes de Dados com os valores de início da execução. Permite indicar o ponto de início da execução de um arquivo MIDI gravado. Serve também para sincronizar diversos equipamentos que façam parte da execução de uma mesma música quando se começa a execução dela num ponto diferente de seu prórpio início [RAT 97].

Mensagens Exclusivas (Exlusive Messages)

Também chamadas mensagens Sys-Ex, são para uso exclusivo dos fabricantes, que podem usá-las para várias finalidades, como, por exemplo, para a transferência de dados internos dos instrumentos (os dados responsáveis pela geração dos sons, por exemplo). Essas mensagens também têm sido usadas para a expansão das aplicações do MIDI, como controle de iluminação e de máquinas de áudio e vídeo [RAT 92].

Ex.: Com dois instrumentos MIDI de mesmo fabricante é possível copiar um programa (PATCH) de um instrumento para outro.

Formato da mensagem – As mensagens de SYS-EX possuem uma formatação mais "aberta" proporcionando a inclusão de qualquer tipo de código dentro da mesma msg.

Os dados das mensagens exclusivas podem ser armazenados num seqüenciador. Neste caso as mensagens de sistema exclusivo devem ser enviadas (de um seqüenciador para um instrumento) antes da seqüência ser disparada pois nenhum outro evento MIDI deve ser transmitido aos equipamentos enquanto durar a mensagem SYS-EX.

Eox – Fim da Mensagem Exclusiva – 1111 0111 - Utilizado para finalizar uma msg exclusiva. As mensagens Sys-Ex são muito utilizadas nos Editores e as Bibliotecas Sonoras. Estes aplicativos musicais lêem os programas de som dos instrumentos para a memória do computador permitem a edição e organização de parâmetros. Devido a limitação dos painéis de equipamentos musicais, os sons podem ser editados na tela do computador graficamente e depois serem enviados novamente através de mensagens SYS-EX para os instrumentos de origem.

Padrão MIDI de transferência de Amostras (MIDI SAMPLE DUMP):

A transferência dos dados externos dos sons digitalizados pelo sampler é possível de ser realizada por mensagens exclusivas padronizadas.

Mensagens de Tempo Real (Real Time)

As mensagens de Real Time (Start, Stop, MIDI Clock) existem para assegurar a sincronização de seqüenciadores, baterias eletrônicas e outros equipamentos que estejam executando música ao mesmo tempo.

Atividade (active sensing) – 1111 1110 - Os equipamentos escravos ao passarem 300 milesegundos sem receber mensagens do mestre assumirão que a linha de comunicação foi interrompida de alguma forma, e se desligarão silenciando todas as suas vozes, aguardando então uma mensagem.

Timing clock – 1111 1000 - É uma mensagem MIDI que obtém o sincronismo de equipamentos mesmo em passagens mais rápidas da música. Durante o tempo de uma semínima devem ser enviados pelo sequênciador 24 mensagens de MIDI Clock (24 ppq, ou 24 pulsos por semínima).

Indicar Sequência (start) – 1111 1010 - Um sequencer slave ou drum machine só pode iniciar uma seqüência após receber uma mensagem específica para isso. Mesmo que um sequenciador esteja recebendo MIDI Clocks através da linha MIDI, ele não fará nada sem que receba um comando de Iniciar.

Parar Sequência (stop) – 1111 1100 - Interrompe imediatamente a sequência, fazendo com que todas as mensagens de MIDI Clock posteriores sejam ignoradas.

Continuar Sequência (continue) – 1111 1011 - Reinicia sequência após o recebimento da mensagem. Os comandos de INICIAR e CONTINUAR não executam qualquer função de MIDI Clocks, ou seja, o recebimento dessa mensagem apenas prepara o equipamento para a função, que só será realizada no recebimento da próxima mensagem de MIDI Clock.

Menssagem de Reinicialização (Reset) – 11111111 - É um alarme geral que só deve ser enviado em caso extremo. Ao enviar esta mensagem, os equipamentos escravos executarão:

Ajuste para MODO1 (OMNI ON/POLY), ajuste para LOCAL CONTROL, Seleção de todas as vozes, ajuste do posicionador da seqüência para 0, parada da execução, cancelamento do STATUS corrente (Running Status), reinicialização do instrumento.

6.1.4.10 MIDI INPLEMENTATION CHART

É uma tabela com todas as informações que o instrumento MIDI pode receber e enviar. Apesar do código MIDI ser padrão as características dos instrumentos eletrônicos variam e portanto estes diferem quanto ao recebimento e funcionalidade de mensagens MIDI. Por isso cada instrumento possui uma tabela (geralmente no manual do usuário) para relacionar todas as características do recebimento e envio de mensagens MIDI [LEH 93].

Function	Tx	Rx	Remarks
Basic default	1 – 16 each	1 – 16 each	Memorized
Channel Changed	1 – 16 each	1 – 16 each	
Mode Default	Mode 3	Mode 3,4	
Messages	X	X	
Altered	*********		
Note	24 – 108	0 – 127	
Number True voice	*********	0 – 127	
Velocity Note ON	o v = 1 127	o v – 1 127	
Note OFF	o v = 1 127	o v = 1 127	
Aftertouch Key´s	X	X	
Ch´s	* 1	* 1	
Pitch Bender	o	o	9 bit resolution
1	o	o	Modulation
2	x	*1	Breath
5	x	o	Portamento time
38,6	x	o	Data Entry LSB,
7	*1	*1	MSB
10	*1	x	Volume
Control	o	*1	Pan
64	x	o	Hold1
Change	x	x	Portamento
65			
100,101	o	o	
121			Reset All Controller
Progam	*1	*1	
Change True #	*********	0-127	
System Exclusive	o	*1	
Song Pos	x	x	
System Song Sel	x	x	
Common Tune	x	x	
System Clock	x	x	
Real Time Commands	x	x	
Local ON/OFF	X	O	
Aux All Notes OFF	x	o	
Messages Acitve Sense	o	o	
Reset	x	x	
Notes	* 1 Pode ser alterado de o para x manualmente e memorizado		

Mode 1 : OMNI ON, POLY **Mode 2 : OMNI ON, MONO** **o : Yes**
Mode 3 : OMNI OFF, POLY **Mode 4 : OMNI OFF, MONO** **x : No**

A interpretação da MIDI Implementation Chart é feita da seguinte forma:

o : Mensagens que podem ser transmitidas ou recebidas

x : Mensagens que não podem ser transmitidas ou recebidas.

Basic channel (canal básico): É a gama de canais, onde um pode ser escolhido para receber ou transmitir dados MIDI. Este canal pode ser mantido memorizado, mesmo quando o equipamento é desligado ("Memorized" no campo "Remarks").

Mode (modo): Os modernos equipamentos MIDI trabalham em modo 3 - OMNI OFF / POLY - como default (configuração de fábrica).

Messages (mensagens): mostra em que modos o equipamento pode transmitir ou receber mensagens.

Altered (alterado): mostra para qual modo serão alteradas as mensagens recebidas em um modo no qual o equipamento não opera. Como esta característica só é válida para recepção, na coluna "Transmissão" aparecerá "*********".

Note number (número de nota): Especifica os números de notas que podem ser transmitidos ou recebidos pelo equipamento.

Dentro deste campo, existe ainda o parâmetro "True Voice" que representa, dentro da faixa que pode ser recebida, aquela que efetivamente terá a geração de um som. Como isto só se aplica à recepção, na coluna de transmissão aparecerá "**********".

Velocity (velocidade): É a faixa de valores nos quais as mensagens Note on e Note off podem ser transmitidas ou recebidas. Cabe lembrar que nem todos os equipamentos podem reconhecer a velocidade de Note off.

Aftertouch (pós-pressionamento): Simplesmente mostra se o equipamento é capaz ou não de reconhecer ou transmitir esta mensagem. É feita ainda uma distinção entre "channel aftertouch" e "key pressure aftertouch". Lembramos que a grande maioria dos equipamentos não transmite nem reconhece o "key pressure".

Pitch bender : Mostra se o equipamento reconhece e transmite esta mensagem e a faixa de valores de trabalho. No campo "remarks" é normalmente mostrada a resolução em bits do valor - quanto maior a resolução, maior será a precisão na geração/interpretação do valor.

Control change (alteração de controle): Indica quais as mensagens de controle que podem ser transmitidas ou recebidas pelo equipamento. Nas mensagens cuja definição não é padronizada, é usual que o fabricante especifique o efeito causado por essa mensagem, no campo "remarks".

Program change (mudança de programa): Mostra se o equipamento reconhece e transmite esta mensagem e a faixa de valores de trabalho. Como subdivisão desta mensagem, temos o "True #" (número real), que representa a quantidade real de

programas existentes internamente ao equipamento e que podem ser interpretados e executados. Como esta característica só é válida para recepção, na coluna "Transmissão" aparecerá "********". Caso se deseje receber um program change fora da faixa de true #, deve-se recorrer ao artifício do "mapping", ou mapeamento de programas (conforme já comentado na análise da mensagem de program change).

System exclusive (sistema exclusivo): Indica se o equipamento reconhece e transmite mensagens em *system exclusive*. É prática indicar na coluna "remarks", quais são os dados utilizados pelo *system exclusive.*

System common (sistema comum): Específica se o equipamento transmite e recebe as mensagens do *system common.*

System real time (sistema em tempo real): Especifica se o equipamento transmite e recebe as mensagens do *system real time*.

Auxiliary messages (mensagens auxiliares): Especifica como o equipamento transmite e recebe as mensagens de modo (local control on/off, all notes off, active sensing, reset, omni on/off, poly e mono).

6.1.4.11 SEQÜENCIAMENTO MIDI

Seqüenciar é gravar mensagens MIDI geradas pela execução do músico ao realizar uma performance no instrumento. Todas as mensagens MIDI podem ser armazenadas e depois reproduzidas pelo seqüenciador.

Modos de Seqüenciamento MIDI

Gravação em tempo real

É o tipo de gravação MIDI mais usado. Nele, o músico dispara a gravação e, ouvindo um metrônomo gerado pelo seqüenciador, toca no instrumento a música a ser gravada.

Ajusta-se adequadamente o valor do andamentoe define-se o tipo de metrônomo a ser usado, Pode-se ainda definir uma contagem inicial (count-in) do metrônomo, antes de começar a gravação, o que é útil para se tomar o andamento. Outro ajuste importante é o da métrica do compasso (time signature), o qual, se estiver errado, fará com que as notas gravadas sejam representadas nos tempos errados em relação ao que foi tocado. É aconselhável indicar ao seqüenciador também a tonalidade da música, para que sua representação fique correta, principalmente se a seqüência for posteriormente salva em arquivo Standard MIDI File para ser lido por um software editor de partituras.

Para iniciar a gravação, o primeiro passo é escolher uma trilha (preferencialmente vazia, sem nada gravado). Em alguns seqüenciadores, a gravação apaga totalmente o conteúdo anterior da trilha; em outros, agravação atual é mixada (merge) com o conteúdo existente. Estando tudo pronto, pode-se disparar a gravação (REC) e, no andamento do metrônomo, executar a parte da música que se deseja.

É importante atentar para o fato de que, em princípio, o seqüenciador gravará tudo o que for tocado. Assim, execução de pedais, movimentação de controle de volume, aftertouch, etc, tudo será armazenado como parte da seqüência. A maioria dos

seqüenciadores possui uma opção de filtragem de eventos (record filter), que permite determinar quais os eventos MIDI que serão realmente gravados. Isso permite o músico escolher quais mensagens MIDI deseja armazenar na sua seqüência. Estes parâmetros devem ser selecionados antes da gravação MIDI [LEH 93].

Overdub

É denominado overdub o processo de adicionar novas partes a um material musical. No seqüenciador isso é possível, sendo recomendável usar uma outra trilha (vazia) para colocar a outra parte. Digamos que você tenha gravado a parte do piano, por exemplo, e agora quer adicionar a linha do baixo. Então você deverá usar um canal MIDI diferente do que está gravado o piano. Dessa forma voce ouvirá o piano para referência e gravará o baixo.

Durante o processo de overdub MIDI, para que se possa ouvir tanto o material já gravado como a execução atual, é necessário que haja mais de um instrumento conectado à saída MIDI Out da interface, ou então que o instrumento seja multitimbral (o que é mais comum hoje em dia).

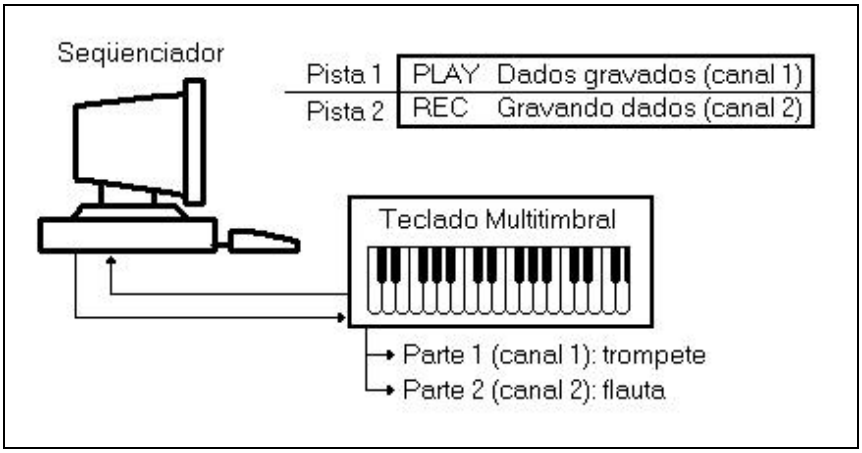


Figura 54 – Sequenciador

No processo de overdub, pode-se gravar um novo material ao mesmo tempo que se ouve o material já gravado.

A gravação da nova parte segue o mesmo procedimento já descrito para a gravação em tempo real, com a diferença de que, caso haja intrumento para isso, além do metrônomo, será ouvido o material já gravado.

Gravação com pontos de Punch-in e Punch-out:

Outra facilidade que há no seqüenciador é a possibilidade de se "remendar" trechos que foram gravados de forma incorreta, por meio de um recurso normalmente chamado de punch-in/out. Para usá-lo, indicam-se os pontos de início e de fim do trecho a ser remendado. A gravação começa, então, de qualquer ponto antes do trecho em questão. O seqüenciador não grava os eventos MIDI que estejam antes do ponto de

início do remendo (punch-in), e grava o que for tocado (substituindo o material antigo daquele trecho) até atingir o ponto de fim do remendo (punch-out).

Gravando passo a passo

Para os usuários menos habilidosos musicalmente, ou quando as passagens instrumentais são muito difíceis de se executar numa gravação, pode-se recorrer à facilidade da gravação passo a passo. Nesse processo, ao invés de executar as notas em um instrumento, via interface MIDI, o músico escreve nota por nota, usando o mouse ou o teclado do computador.

Para isso, pode-se escolher qualquer uma das vistas que for mais conveniente ou agradável para o músico (piano-roll, partituta, lista de eventos) e inserir as notas desejadas, indicando sempre todos os seus parâmetros (posição, altura, intensidade e duração). Esse processo é trabalhoso, mas possibilita enorme precisão dos dados.

É comum usar-se os dois processos em combinação. Grava-se em tempo real tudo o que for mais fácil; em seguida entra-se com algumas notas passo a passo, ao mesmo tempo que se corrige a notas que tenham sido tocadas de modo incorreto. A entrada de alguns eventos, como controle de volume (Control Change 7) e mudança de timbre (Program Change) também são freqüentemente feitos passo a passo.

Gravação de padrões / gravação em loop

Um processo de gravação de notas muito usado em baterias eletrônicas é a gravação em loop. Na gravação em loop um determinado trecho fica tocando indefinidamente,e as notas que são executadas no instrumento vão sendo gravadas na trilha, misturando-se (merge) com aquelas que já estão gravadas. A gravação em loop permite que se adicione notas em um trecho, sucessivamente, sem parar a gravação.

A cada repetição do loop, o músico pode ouvir as novas notas que foram gravadas, já adicionadas (misturadas) àquelas que já estavam gravadas. Caso as notas da última passagem não tenham ficado boas, em geral, é possível apagá-las sem destruir as anteriores.

Esse recurso é muito útil, principalmente para a criação de compassos e padrões rítmicos de bateria, onde é comum gravar-se adicionando peças individualmente.

Gravação multicanal

Na gravação multicanal (multichannel ou multitrack recording) o seqüenciador pode gravar simultaneamente notas de diversos canais MIDI, colocando-as em trilhas diferentes, de acordo com o número do canal. Ela é muito útil quando são usados dois ou mais instrumentos ao mesmo tempo (uma "jam session", por exemplo) transmitindo em canais diferentes, quando então a interface deverá possuir uma entrada MIDI In para cada instrumento. Outra aplicação é quando se quer gravar em tempo real uma seqüencia de outro seqüenciador que contém eventos em diversos canais, ou uma execuçccão produzida em um teclado arranjador (que transmite cada parte do arranjo em um canal diferente).

Edição de Eventos MIDI

Os recursos de edição são as maiores ferramentas do seqüenciador. Com eles, é possível corrigir, mudar, recriar, cortar, colar e muito mais. Um item importante no que se refere à edição é a possibilidade de se reverter o processo, através da função chamada de desfazer (undo). Essa função torna o processo não destrutivo, possibilitando "voltar" quando a edição não ficar boa [LEH 93].

Filtragem de eventos

Na maioria dos seqüenciadores, o usuário pode optar se quer que as operações de edição manipulem todos os eventos existentes em um determinado trecho marcado ou apenas alguns daqueles eventos. Isso é possível quando existe o recurso de filtragem de eventos, uma opção que possibilita ao usuário discriminar quais dos eventos do trecho marcado serão afetados pelo comando de edição.

Quantização

O recurso de quantização é útil quando as notas executadas estão fora dos tempos corretos (atravessadas). No processo de quantização, marca-se o trecho onde deverá ser efetuada a edição e define-se a figura da nota musical para a qual devem ser aproximadas as notas gravadas.

Ao quantizar as notas, o seqüenciador as move para a posição mais próxima da figura de nota escolhida como referência, de forma que todas as notas ficarão perfeitamente posicionadas. Um software seqüenciador possibilita que os eventos MIDI gravados possam ser visualizados de forma gráfica na tela do computador. Essa possibilidade é uma grande vantagem sobre outros seqüenciadores, como aqueles embutidos em MusicWorkstations, pois reflete em grande facilidade na hora da edição e visualização da música. Por exemplo, No piano-roll de um software seqüenciador as notas são representadas por tarjas, cujos comprimentos indicam suas durações; e suas posições verticais definem sua altura (em relação ao teclado). As linhas verticais indicam os tempos (posições) das semínimas.

A escolha da figura de quantização correta é muito importante pois se escolhermos uma figura maior ou menor do que a necessária teremos erros de quantização, pois as notas seriam posicionadas em tempos incorretos. Assim, a figura de quantização deve ser compatível com a as figuras usadas na melodia, de forma que se uma melodia está em colcheias, por exemplo, a figura de quantização deve ser colcheia. Se houver uma passagem em semicolcheias no meio, já não será possível usar a figura de colcheia para quantizar todo o trecho (ou se quantiza por partes: trechos em colcheia e trechos em semicolcheia) ou então tenta-se quantizar pela menor figura. O inverso também é perigoso: se for usada como referência de quantização uma figura de tempo muito menor do que a cadência do trecho, pode acontecer das notas serem também posicionadas erradamente, quebrando os tempos.

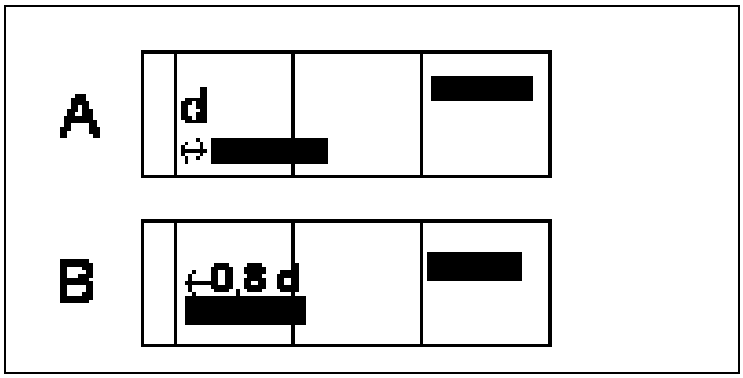


Figura 55 - Quantização

A quantização percentual, ao invés de reposicionar as notas no tempo exatamente correto, aproxima-se percentualmente daqueles tempos. No exemplo acima, foi definido um percentual de acerto de 80%, de forma que as notas em B foram aproximadas dos tempos corretos apenas 80% da distância total.

A quantização é aplicada com o objetivo de melhorar uma execução, tornando-a mais precisa. Entretanto, às vezes o material quantizado acaba soando pior do que o que era originalmente e, nesses casos, é melhor reverter (desfazer) a operação, e tentar acertar as notas de alguma outra forma. Em trechos mais complicados, onde há notas com diversas cadências, pode ser mais adequado acertá-las manualmente, uma por uma.

Músicas totalmente quantizadas tendem a soar muito artificiais, muito mecânicas. Uma forma de se evitar essa mecanização da música é usar uma quantização percentual, um recurso disponível na maioria dos seqüenciadores profissionais. Esse tipo de quantização, ao invés de acertar totalmente as notas, apenas aproxima-as da figura de referência, num percentual de aproximação definido pelo usuário.

Usando-se um percentual de acerto de 80%, por exemplo, todas as notas são aproximadas apenas 80% do total, caso fosse usada a quantização total. Dessa forma, melhora-se a execução, mas sem perder a flutuação de tempo, o chamado "swing". Há ainda seqüenciadores que permitem gerar um swing a partir de notas quantizadas, num processo quase que inverso ao da quantização, onde as notas são posicionadas ligeiramente fora dos tempos certos.

Transposição

A transposição é a função que altera a altura das notas. Na realidade, como a altura de uma nota é representada por seu número, o processo de transposição resume-se a somar ou subtrair determinado valor (em semitons) dos números das notas.

6.1.5 A Física do Som

6.1.5.1 O Som

Diariamente, nós ouvimos uma grande quantidade de sons , vozes, sons de portas abrindo e fechando, passos, ruído dos motores dos automóveis, chuva e música. Em outras palavras, vivemos toda a nossa vida rodeados por sons. O som não é algo que podemos ver com nossos olhos. Então, o que é isso que chamamos de som? Tomando como exemplo o som de uma campainha, temos que quando a energia cinética é aplicada a ela, através de um martelo, ocorre uma "deformação" da campainha, gerando desta maneira a energia que trata de devolver a campainha ao seu estado original. Começa, então, uma repetição periódica de deformações e restaurações. A isto chamamos de vibração. Estas vibrações produzem mudanças de pressão do ar, resultando em seções de ar que são mais densas e outras que são rarefeitas, ocorrendo sucessivamente uma depois da outra e expandindo-se. Estas são chamadas ondas de condensadas e rarefeitas. O processo é similar ao que conhecemos quando jogamos uma pedra dentro da água, a qual produz ondas circulares em sua superfície. Estas ondas condensadas e rarefeitas são propagadas para dentro do ouvido humano e irão vibrar o tímpano. As vibrações são captadas pelas terminações nervosas, de maneira que nós as escutamos como sons. Se os corpos que vibram são diferentes, também será diferente a classe de vibração que produzem, isto significa que escutamos distintas classes de sons. No espaço, onde não existe ar, também não existem sons. O som é, portanto, a vibração do ar.

Formas de Onda

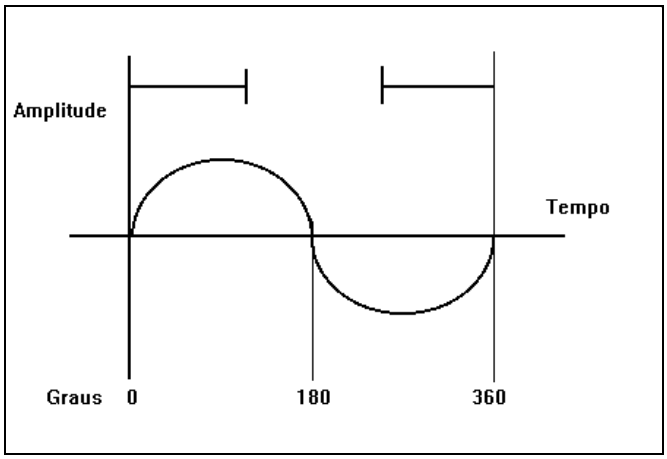


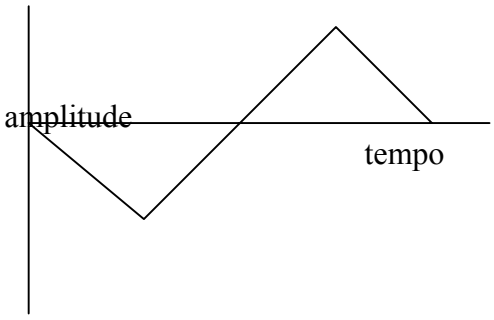
Figura 56 - Onda Sonora

Os sons não podem ser vistos porque são vibrações do ar. Para melhor compreendermos o que são ondas sonoras podemos tomar como exemplo o mecanismo de um microfone usado como meio de captar sons. Um microfone converte o som em sinais elétricos que se pode transmitir a um amplificador. Estes sinais elétricos são simples conversões elétricas das vibrações do ar, que são trocados na pressão atmosférica. Quando estas mudanças são apresentadas em forma gráfica, podem ser interpretadas como "formas de ondas". Um osciloscópio é um aparelho prático que também converte sons em sinais elétricos mostrando-os em forma de onda (Figura 56) em uma tela de televisão. O osciloscópio é um aparelho muito empregado para a visualização da forma da onda pois estas diferem enormemente de acordo com o som.

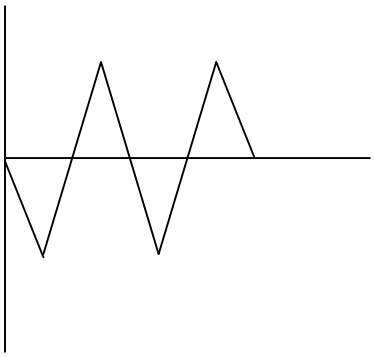
6.1.5.2 Os Elementos Básicos do Som

Altura

Quando se pulsa as teclas de um piano, nota-se que os sons são mais agudos quanto mais tocados a direita e mais graves quanto mais se tocar a esquerda. Essa "altura" de um som, ou seja, se é alto ou baixo, é chamado "altura tonal". Quando são comparados em um osciloscópio sons com alturas tonais diferentes, nota-se que difere o número de ondas por unidade de tempo. Quanto mais agudo é o som, maior será o número de ondas, quanto mais grave é o som, menor será o número de ondas. Portanto, o número de ondas é o número de vibrações que produzem o som. Quanto mais agudo o som, maior o número de vibrações por unidade de tempo, quanto mais grave o som, menor o número. Por exemplo, o número de ondas obtidos das vibrações das cordas de um violino é maior do que o número de ondas produzidas ao se tocar um contra-baixo. O número de vibrações dentro do intervalo de um segundo é geralmente chamado freqüência e expresso em unidades chamadas Hz (Hertz). 100 Hz indica que a vibração ocorre com uma freqüência de 100 vezes por segundo. Quanto maior o número de Hertz, mais agudo é o som. Dobrando a freqüência de um som, este é elevado em uma oitava.



Onda sonora



Onda com o dobro da freqüência

A faixa de freqüências que podem ser ouvidas por um ouvido humano é de aproximadamente 20 Hz para os sons mais graves e de até 15000 Hz para os sons mais agudos. A altura tonal de um som depende do número de ondas por unidade de tempo, ou seja, da freqüência e da vibração, e é mais aguda quanto mais alta a freqüência.

Capacidade de perceber a freqüência:

(Som Baixo) 20 Hz
:
:
(Som Alto) 20.000 Hz (20 KHz)
(K=mil)

As freqüências são classificadas em baixas, médias e altas:

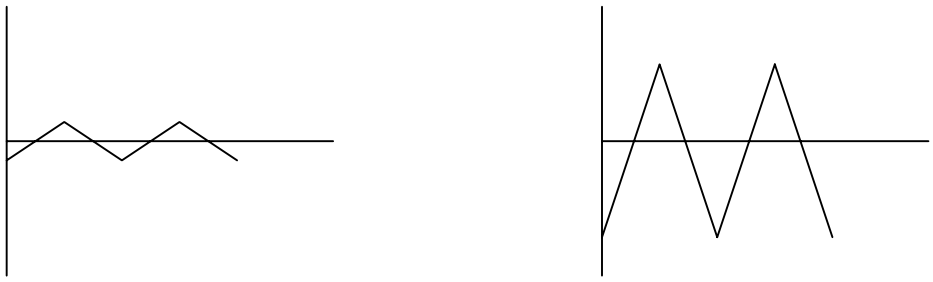
Baixa – até 300 Hz	*Média – 300 Hz até 2 KHz*	*Alta – 2 KHz ...*

A banda de freqüência de alguns instrumentos musicais.

Instrumento	Freqüência
Bumbo – Bateria	60 – 120 Hz
Caixa – Bateria	100 – 700 Hz
Pratos	200 – 1500 Hz
Baixo	70 – 500 Hz
Piano (Registro Baixo)	180 – 500 Hz
Piano (Registro Alto)	900 – 2000 Hz

Volume

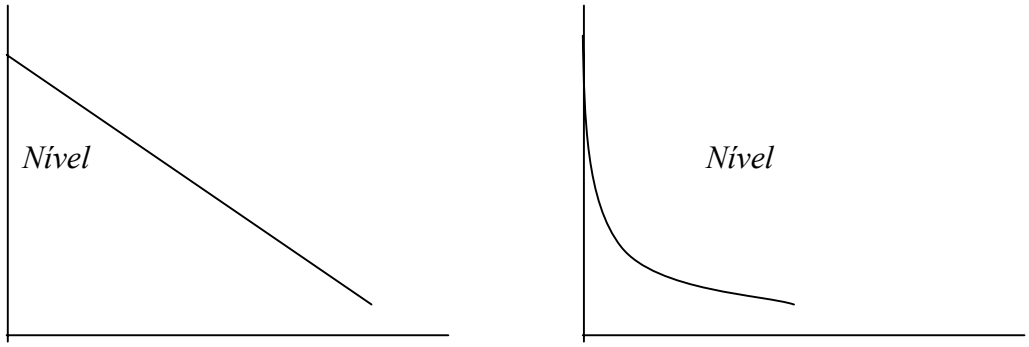
Se uma tecla de piano é pulsada fortemente, o som produzido será forte. Se for pulsada suavemente, o som produzido será fraco. As denominações alto e baixo devem ser utilizadas de maneira correta apenas para designar a altura tonal e não o volume. Através de um osciloscópio, a mudança no volume de um som pode ser vista como uma diferença na altura das ondas. A altura de uma onda chama-se "amplitude". Quanto maior a amplitude, mais forte é o som. Portanto o volume de um som é determinado pela amplitude (altura da onda). Ao contrário do que é falado coloquialmente, um som alto, do ponto de vista da amplitude corresponde a um som forte. Já um som baixo, corresponde a um som fraco.



Ondas com diferentes amplitudes

Quando desejamos comparar duas grandezas quaisquer de mesmo tipo, no campo da eletrônica e do áudio, utilizamos uma unidade chamada decibel (dB).

Para a compreensão do decibel é necessário examinar os logaritmos e as escalas logarítmicas. O logaritmo (log) é uma função matemática que reduz grandes valores numéricos em pequenos valores numéricos onde os números podem ser mais facilmente manuseados. Por aumentarem exponencialmente, eles expressam nosso sentido de percepção mais precisamente do que uma curva linear.



Escala linear Escala logarítmica

O logaritmo pode ser definido como segue:

$$\text{Se } a = 10^{b} \text{ então } b = \log a$$

Para achar o logaritmo de um valor deve-se olhar nas tabelas de logaritmos ou utilizar a calculadora. No entanto em aplicações para o som utiliza-se a base 10. Sendo assim, o valor do logaritmo pode ser encontrado facilmente. Simplesmente escreve-se o número na sua forma exponencial e o expoente será o logaritmo. Por exemplo, o número 10.000 é 10^4 na forma exponencial, então seu logaritmo é 4 [CAR 92].

Ex. Log 1 = 0 porque $10^0 = 1$
Log 10 = 1 porque $10^1 = 10$
Log 100 = 2 porque $10^2 = 100$
Log 1000 = 3 porque $10^3 = 1000$

Os valores dos logaritmos são negativos para potências de 10 menores do que 1.

Ex. Log 0,1 = -1 porque $10^{-1} = 0,1$
Log 0,01 = -2 poque $10^{-2} = 0,01$
Log 0,001 = -3 porque $10^{-3} = 0,001$

No som mais suave e silencioso que podemos ouvir o limite de audição é 0 dB SPL (Sound Pressure Level). A conversação em um restaurante pode ter uma média de nível de audição em torno de 60 dB SPL. Uma avenida com muito movimento pode ter uma média de audição de 85 dB SPL. O limite máximo do ouvido humano onde existe uma audição com dor vai de 125 a 130 dB SPL [CAR 92].

dB SPL – decibéis (Sound Pressure Level)

0 dB	Limite mínimo da audição.
10 dB	Conversação em voz baixa.
40 dB	Rua sem tráfego.
85 dB	Tráfego intenso.
120 dB	Turbina do avião.
130 dB	Limite máximo (dor).

Uma relação entre grandezas expressa em decibéis é dada pela seguinte fórmula:

$$G\,(dB) = 10 \log \; G / G_{ref}$$

Onde

G(dB) é a relação em decibel
G é a grandeza a ser medida
G_{ref} é o valor padrão de referência da grandeza

A sensação do ouvido humano em relação a amplitude é logarítmica. Para que tenhamos a sensação do dobro da intensidade sonora é necessário que a pressão sonora tenha sido multiplicada por 10. Por isso que são usados potenciômetros logarítmicos em painéis de vários equipamentos de áudio.

Um amplificador de áudio pode ter, por exemplo, uma escala em dB. Isto não significa que a comparação é feita do som que sai das caixas acústicas com o limite de audição, mas sim do nível de referência da potência elétrica que é 1 Watt.

Teoricamente todo o amplificador deveria ter 120 dB que é o máximo que o ouvido humano suporta, porém isso necessitaria de uma grande potência em watt RMS como pode ser observado a seguir:

Valor de Referência = 1 Watt RMS

Relação	6.1.5.2.1.1.1 Fórmula	6.1.5.2.1.1.2 Cálculo	Resultado (dB)
1:1	10 Log 1/1 =	0 x 10 =	0 dB
2:1	10 Log 2/1 =	0,3 x 10 =	3 dB
3:1	10 Log 3/1 =	0,4 x10 =	4 db
4:1	10 Log 4/1 =	0,6 x10=	6 dB
5:1	10 Log 5/1 =	0,69 x10=	6,9 dB
6:1	10 Log 6/1 =	0,78 x10=	7,8 dB
7:1	10 Log 7/1 =	0,84 x10=	8,4 dB
8:1	10 Log 8/1 =	0,93 x10=	9,3 dB
9:1	10 Log 9/1 =	0,95 x10=	9,5 dB
10:1	10 Log 10/1 =	1 x10=	10 dB
100:1	10 Log 100/1 =	2 x10=	20 dB
1000:1	10 Log 1000/1 =	3x10 =	30 dB
10.000:1	10 Log 10.000/1 =	4x10 =	40 dB
100.000:1	10 Log 100.000/1 =	5x10 =	50 dB
1.000.000:1	10 Log 1.000.000/1 =	6x10 =	60 dB
10.000.000:1	10 Log 10.000.000/1 =	7x10 =	70 dB
100.000.000:1	10 Log 100.000.000/1 =	8x10 =	80 dB
1.000.000.000:1	10 Log 1.000.000.000/1 =	9x10 =	90 dB
10.000.000.000:1	10 Log 10.000.000.000/1 =	10x10 =	100 dB
100.000.000.000:1	10 Log 100.000.000.000/1 =	11x10 =	110 dB
1.000.000.000.000:1	10 Log 1.000.000.000.000/1 =	12x10 =	120 dB

Existem vários valores padrões de referência no qual potência e voltagem são comparados usando decibéis conforme mostra a tabela abaixo:

Tipo do decibel (elétrico)	Referência Padrão
DBm	1 milliwatt
DBu	0,775 volts RMS
DBV	1 volt RMS

Decibéis que referenciam potências são usados para circuitos que apresentam uma significante quantidade de corrente provinda da fonte de tensão elétrica (voltagem). Se a medida da potência é igual a potência de referência ($G_1 = G_{ref}$) a diferença é 0 dB, ou seja, 0 dB é o nível de referência.

$$10 \log (G_1 / G_{ref}) = 10 \log 1 \quad (\text{A divisão de } G_1 \text{ por } G_{ref} = 1)$$
$$= 10 \times 0$$
$$= 0 \text{ dB}$$

Se a potência medida é duas vezes o valor de referência ($2G_1 = G_{ref}$), a diferença é de aproximadamente 3 dB.

$$10 \log (G_1 / P_{ref}) = 10 \log 2$$
$$= 10 \times 0{,}301$$

$= 3{,}01$ dB

Se a potência medida é 10 vezes o valor referência ($10\ G_1 = G_{ref}$) diferença é de 10 dB.

$$10 \log (G_1 / G_{ref}) = 10 \log 10$$
$$= 10 \times 1$$
$$= 10 \text{ dB}$$

A referência de potência mais comum é denotada por dBm e o valor de potência P_{ref} é 1 milliwatt (mW), ou 0,001 watt. Então 0dBm = 1 mW.

Esse tipo de decibel é empregado principalmente quando medimos pequenos valores de potência como as que existem nos equipamentos de áudio profissionais.

Decibel x Voltagem

Decibéis que referenciam voltagem são usados principalmente em equipamentos semi-profissionais e sintetizadores. O nível de referência continua sendo o 0 dB. Todavia, se a voltagem medida é o dobro do valor de referência ($V_1 = V_0$), a diferença é 6 dB, e não 3 dB como nos decibéis que referenciam a potência.

$$20 \log (V_1/V_0) = 20 \log 2$$
$$= 20 \times 0{,}301$$
$$= 6{,}02 \text{ dB}$$

Este tipo comum de decibel referenciando a voltagem é denotado por dBu. A voltagem de referência para o dBu é 0,775 V. Ou seja, 0 dBu = 0,775 V.

Outro tipo de decibel usado para referenciar voltagem é denotado por dBV, para o qual a voltagem de referência é 1 V. Ou seja, 0 dBV = 1V.

Exemplos de aplicação do decibel

Exemplo 1 – Os violinos.

Num ambiente temos um som produzido por um violinista na intensidade de 3dB. Se somarmos mais um violinista tocando na mesma intensidade teremos 6dB. Se outros dois violinistas começarem a tocar juntos com os outros dois teremos 9dB. Quando mais quatro violinistas iniciarem a tocar também com a mesma intensidade, teremos uma medida de 12 dB. Por isso que a orquestra precisa de tantos violinos para produzir um som mais intenso.

Exemplo 2 - Potência dos amplificadores.

Um amplificador tem a potência de 100 Watt RMS correspondendo a uma saída de 20 dB watt (relacionado a 1 Watt). Se a potência desse amplificador for somada a potência de um outro de também 100 watt RMS, teremos 23 dB Watt (pois 10 log 200 = 23 dB Watt). Se mais dois amplificadores de mesma potência forem somados aos outros

dois então teremos 26 dB Watt (pois 10 log 400 = 26 db Watt). Se tivermos 8 amplificadores de 100 watts RMS a resultante será 29 dB watt (pois 10 log 800 = 29 db Watt).

Exemplo 3 – A intensidade sonora ao ar livre.

Uma caixa amplificada acústica fornece 80 dB SPL (em relação a 1 watt a 1 metro de distância). A dissipação ao ar livre a cada 10 metros é de 6 dB de perda. Logo a 10 metros da fonte de projeção sonora, ao medirmos com um decibelímetro, teremos 74 dB.

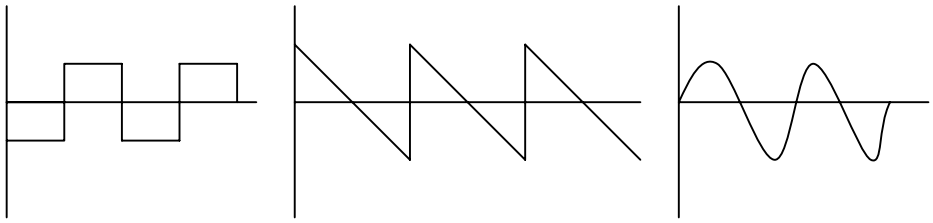
Ligamos um aplificador de 100 Watts RMS com 20 dB (pois 10 log 100 Watt RMS = 20 dB Watt) a esta caixa amplificada obtendo 100 dB.

Qual será a potência em dB que uma platéia à 20 m de distância ouvirá?

Solução: 80 dB + 20 dB – 12 dB (pois a cada 10 metros temos a perda é de 6 db) = 88 dB.

Timbre

Um clarinete e uma flauta não produzem o mesmo som ao serem tocados com a mesma altura tonal e o mesmo volume. Isso se deve ao fato de haver um fator distinto a mais para o som além da altura tonal e o volume; este fator é o timbre. Observando-se sons com timbres diferentes , através de um osciloscópio, nota-se que as formas de onda diferem entre si. De um modo geral, formas de ondas arredondadas produzem um timbre mais suave enquanto que as formas de ondas ponte-agudas dão um timbre mais penetrante e estridente.



Onda retangular Onda Dente-de-serra Onda senoidal

Na Tabela 12 são mostradas as três formas de onda, os timbres resultantes e os instrumentos característicos em cada caso.

Tabela 12 - Formas de ondas e seus timbres característicos

Forma de Onda	Timbre	Instrumento
Onda retangular	*complexo*	*clarinete, oboé*
Onda dente de serra	*claro*	*violino,trompete*
Onda senoidal	*suave*	*flauta,assobio*

Os sons complexos são formados por ondas simples (ondas senoidais) que são os harmônicos e a fundamental. Os harmônicos ou parciais são múltiplos da freqüência básica ou fundamental. A fundamental de um som complexo é o componente de mais

baixa freqüência. A freqüência fundamental e seus harmônicos constituem a série harmônica. A resultante é uma forma de onda que representa um som complexo.

Na Tabela 13 está a representação matemática correspondente a Figura onde podemos encontrar exemplos de fundamental e seus harmônicos.

Tabela 13 - A fundamental e seus harmônicos

p1	P2	p3	p4	p5	resultante
fundamental	Parcial 2	parcial 3	parcial 4	parcial 5	série
Senxt	1/2sen2xt	1/3sen3xt	1/4sen4xt	1/5sen5xt	p1+p2+p3+p4+p5

onde x = 2 π f
f= freqüência
t = tempo

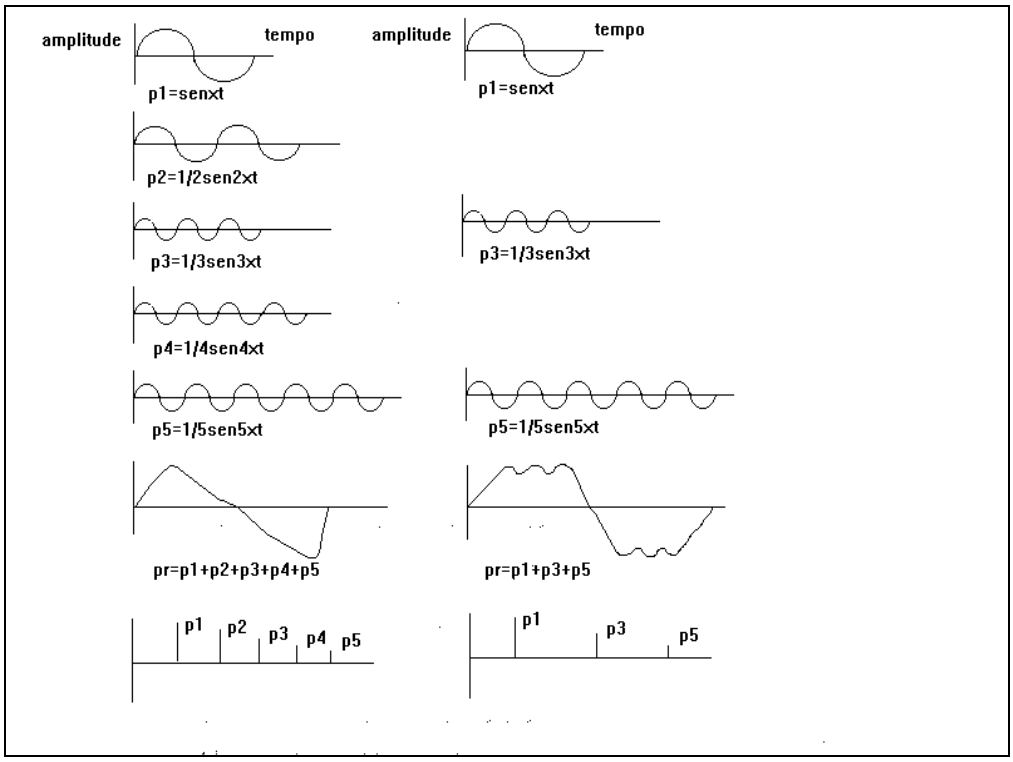


Figura 57 – A Fundamental e suas Parciais

A estrutura de uma onda produzida por um instrumento musical pode ser representada por um espectro gráfico. Na figura 57, as barras verticais representam a amplitude da fundamental p1 e a dos parciais. Com base nesta figura é possível concluir que podem ser desenvolvidas várias formas de onda a partir das estruturas harmônicas mais complexas.

Tons Puros:

Chamam-se de tons puros os sons que não têm em absoluto nenhum outro componente (tais como harmônicos) e consistem em uma só freqüência simples. A forma de onda de um tom puro sempre é uma onda senoidal Os timbres de um

diapasão afinador ou toque do telefone são quase tons puros (ondas senoidais perfeitas), mas esta classe de timbre não existe no mundo natural. Portanto, os tons puros só podem ser criados artificialmente, por exemplo, eletronicamente. Chama-se de harmônicas as freqüências que são múltiplos integrais de uma onda básica. O Timbre (forma de onda) é determinado pelos componentes harmônicos.

Envolvente da Onda (Envelope)

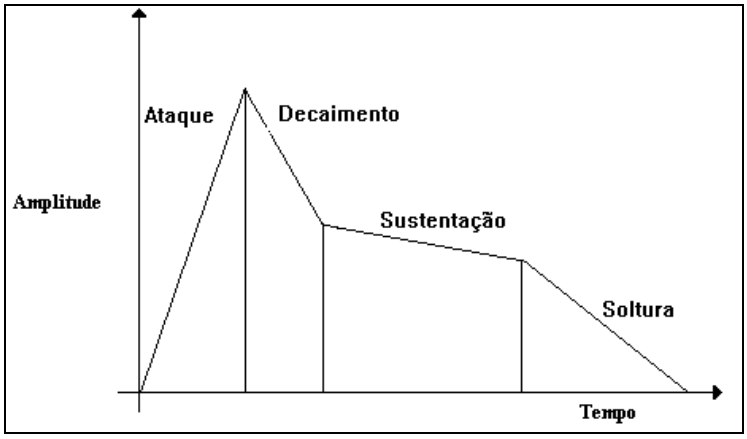


Figura 58 - Envolvente da Onda.

Além dos três fatores básicos do som explicados anteriormente, altura tonal, volume do som e o timbre, há outro fator importante que determina o som. Trata-se da variação de som em um transcurso de tempo. Mais precisamente, a variação de cada um dos três elementos em um transcurso de tempo que vai desde o começo do som até um ponto do tempo onde desaparece completamente. Se um violino for tocado com arco, por exemplo, geralmente o volume do som aumenta gradualmente e também o timbre e altura tonal trocam ligeiramente. Estas trocas em um tempo são o que determinam o timbre característico de um violino. Por outro lado, o som de caída de um piano é um caimento contínuo, sem este caimento seria muito difícil distingüi-lo do som de uma flauta. Estas variações em um transcurso de tempo são chamadas envolventes (envelopes).Envolventes são, portanto, as mudanças de altura tonal, volume e timbre em um transcurso de tempo.

6.1.6 Síntese Sonora

Geralmente a primeira idéia que vem à uma pessoa pouco familiarizada com os aspectos relativos ao sintetizador é a utilização como simulador de instrumentos utilizados na musica clássica como, por exemplo, o som das cordas da orquestra. Mas a verdadeira função do sintetizador não é de imitar, e sim a de criar sons totalmente inéditos e irrealizáveis por outros métodos. A imitação é uma função adicional do sintetizador e, portanto, não se pode esperar que os sons sejam exatamente iguais aos instrumentos os quais se pretende imitar ou simular.

Existem várias técnicas que podem ser utilizadas para sintetizar sons. Muitas delas utilizam o modelo de fonte e modificadores do som. O uso desse modelo fica mais claro na síntese subtrativa, mas ele pode ser aplicado a outros métodos de síntese como a S & S (Sample and Synthesis), Waveshaping (Distorção de fase), ou modelagem física, por exemplo. Alguns métodos de síntese são mais complexos como a síntese FM (freqüência modulada).

Uma metáfora de modelo desses métodos mais complexos pode ser muito difícil de compreender. Esta é uma das razões pela qual os métodos mais fáceis de compreender como a síntese subtrativa e a S & S obtiveram um sucesso comercial. Os sintetizadores digitais implementam diferentes técnicas de síntese sonora. No entanto, a síntese subtrativa é comum a quase todos os sintetizadores comerciais. Certas sintetizadores apresentam mais de uma forma de síntese e até combinações delas.

6.1.6.1 Síntese Digital

A tecnologia digital pode ser entendida como a representação numérica de sinais e o uso de computadores para o processamento dos mesmos.

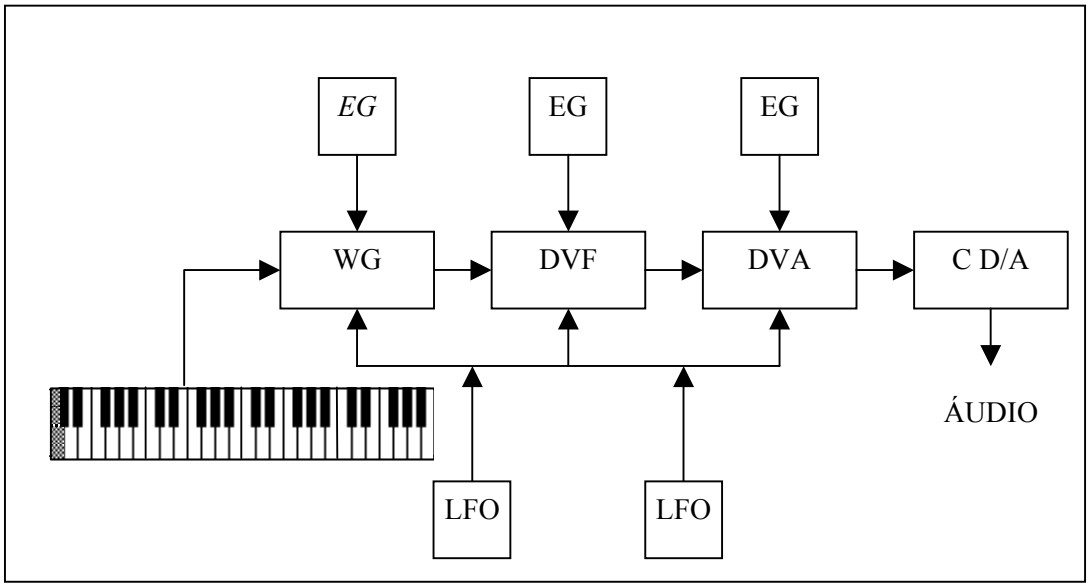


Figura 59 – Esquema básico de um sintetizador digital

6.1.6.2 Síntese Subtrativa

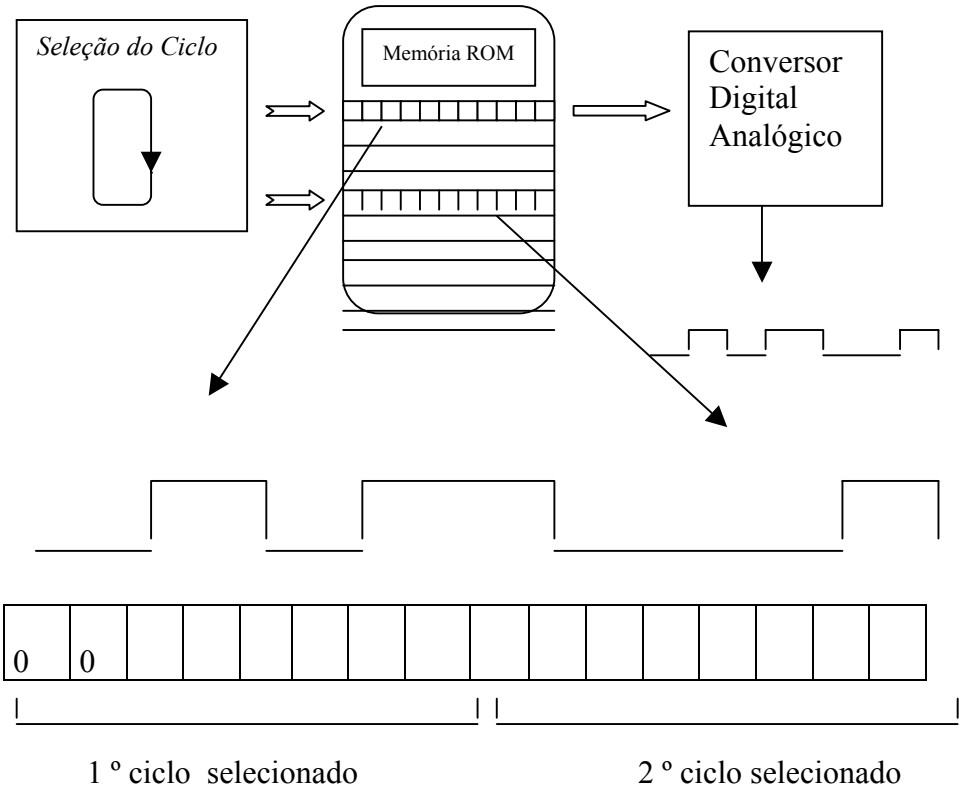
A síntese subtrativa parte de um som rico em harmônicos que é filtrado e seu conteúdo harmônico retirado.

Os sons originais são, tradicionalmente, formas de ondas matemáticas tais como quadrada, dente-de-serra e triangular, sendo que os modernos sintetizadores subtrativos disponibilizam amostras digitais ao invés de um ciclo de forma de onda. A filtragem tende a ser um filtro resonante de passa-baixa onde as mudanças no ponto de corte da freqüência (cut-off frequency) desse filtro produzem o som de filtro "sweep" que é característico da síntese subtrativa.

Fonte Sonora

Num sintetizador podemos obter a fonte sonora a partir de um gerador de ondas (Wave Generator) ou a partir de entradas externas.

O gerador de ondas possui uma tabela de ondas onde estas são representadas numericamente. Várias formas de ondas podem ser armazenadas no sintetizador para depois serem dinamicamente selecionadas em tempo real. A síntese por Tabela de Ondas amostradas digitalmente, utiliza a memória como parte integral do processo de síntese, pois, o ciclo usado é dinamicamente selecionado pelo controle de memória. Um ciclo de forma de onda é armazenado na memória de um chip e sucessivos valores são carregados da memória e enviados para o Conversor Digital Analógico onde é produzida a saída da forma de onda. Um sintetizador wavetable utiliza vários ciclos de ondas localizados na memória.



Expressões como Vector Sintese (Síntese Vetorial) e Linear Arithmetic (Aritmética Linear) são utilizadas pelos fabricantes para referenciar pequenas diferenças na abordagem da síntese por Tabela de Ondas (Wavetable).

Atualmente o método mais utilizado em sintetizadores digitais é o sampling – ou amostras digitais de sons armazenados na memória sob a forma de números.

Modificadores do Som

A função de um módulo de modificação do sinal é de modular o sinal da amostra digital. Quando o sinal passa através do módulo a amplitude, o timbre, ou outros aspectos do sinal serão modificados de forma a resultar em um som mais apropriado com o contexto musical. Podemos fazer uma analogia comparando com o processo que o corpo da guitarra modifica o som produzido pela vibração das cordas.

DCF - Digital Controlled Filter (Filtro Controlado Digitalmente) ou TVF-Time Variant Filter (Filtro Controlado por Variação de Tempo)

Nos sintetizadores da linha Roland este modificador é chamado de TVF (Time Variant Filter). É o circuito que corresponde ao elemento básico do som, o timbre. Este filtro modifica o timbre ao acentuar ou filtrar certos harmônicos das formas de ondas criadas pelo WG (gerador de ondas). O DCF pode ser considerado como a parte mais importante de um sintetizador.

O timbre é produzido pela presença de harmônicos de diversas freqüências. O filtro permite determinar faixas de freqüências que podem passar ou não produzindo modificação no brilho e na qualidade tonal do som.

Tipos de Filtro

HPF (High Pass Filter) - Este filtro "passa alta" corta as freqüências graves, e deve ser usado para criar sons brilhantes e sem “massa”.

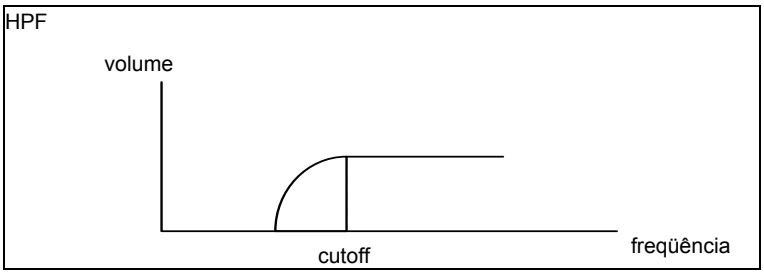


Figura 60– *High Passs Filter*

BPF (Band Pass Filter) - O filtro "passa banda" permite escolher uma região de freqüências a ser ouvida. Ele enfatiza as freqüências médias.

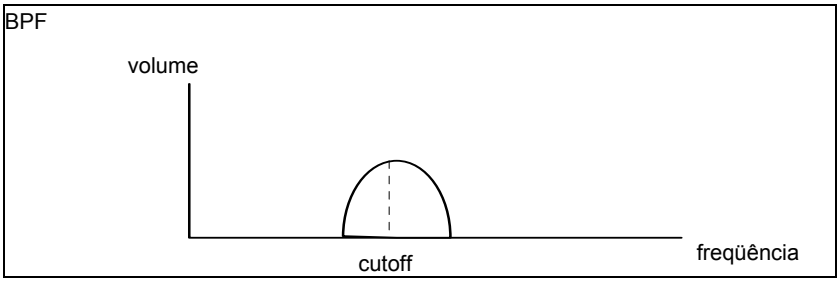


Figura 61 – *Band Pass Filter*

LPF (Low Pass Filter) - O filtro "passa baixa" permite cortar freqüências altas. Na verdade este é o filtro mais usado, e permite tornar os timbres mais suaves.

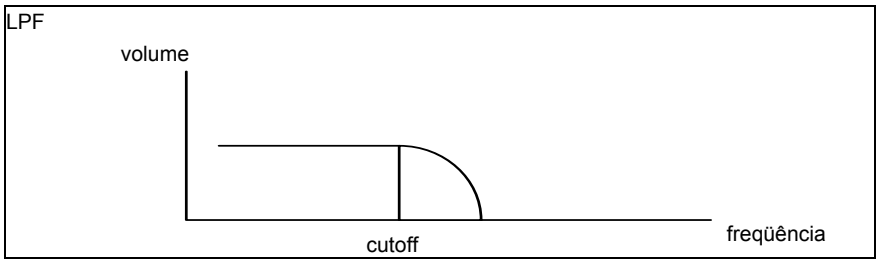


Figura 62 – *Low Pass Filter*

Freqüência de Corte - CUTOFF FREQ

Com o ajuste da freqüência de corte, se determina em que harmônicos do som o filtro irá atuar. Modificando a freqüência de corte, você controla o brilho do som. Elevando este controle, a freqüência de corte será feita nos registros mais altos do som.

Ajuste do ponto de corte para HPF:

Quando o filtro for HPF, e o valor de cutoff for elevado, você terá menos parciais graves, e o som será mais brilhante. Ao mesmo tempo o volume será menos intenso. Com ajustes muito altos, algumas formas de onda não produzirão nenhum som.

As figuras 63, 64, 65 e 66 mostram exemplos gráficos da modificação da onda quadrada pela posição do ponto de corte usando o HPF.

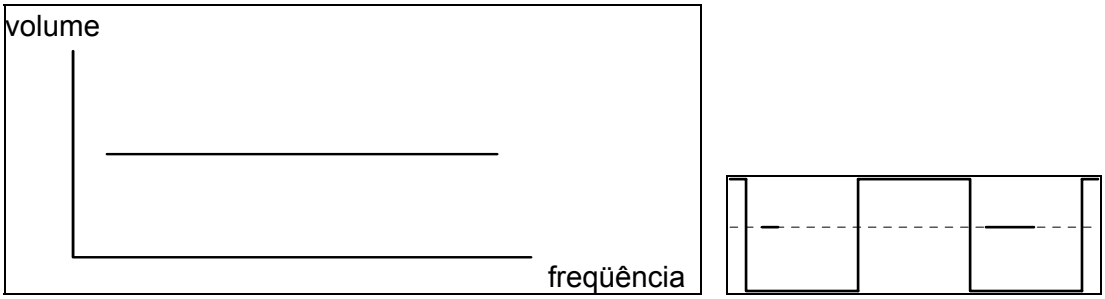


Figura 63 – Onda sem corte de freqüência

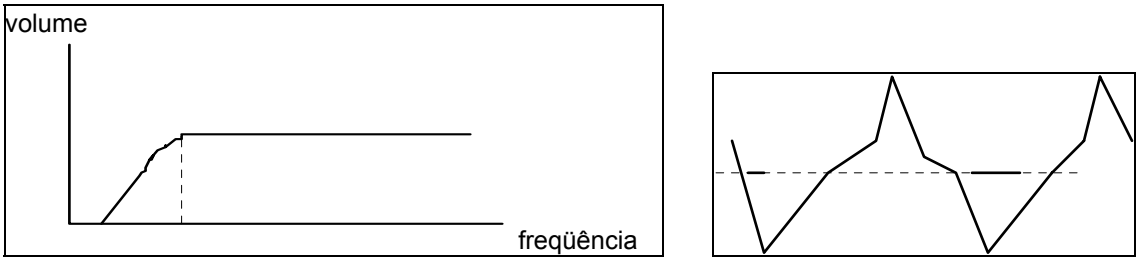


Figura 64 – Onda submetida ao filtro HPF com corte de freqüência

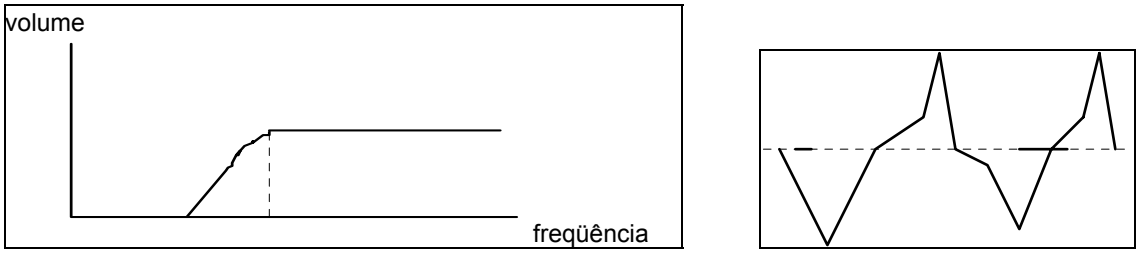


Figura 65 – Aumento no ponto de corte da freqüência

Figura 66 – Onda resultante do corte de freqüência utilizando o HPF

Ajuste do ponto de corte para BPF:

Quando o filtro for BPF, apenas as freqüências da região de corte "cutoff" serão ouvidas. Com ajustes altos neste filtro pode ser que nenhum som seja ouvido.

As figuras 67, 68, 69 e 70 mostram a modificação da onda pela posição do ponto de corte usando o BPF.

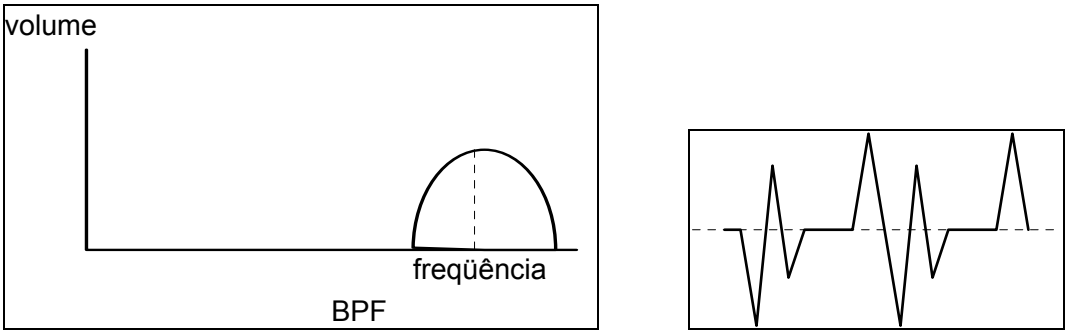


Figura 67 – Onda submetida ao BPF com corte de freqüências graves

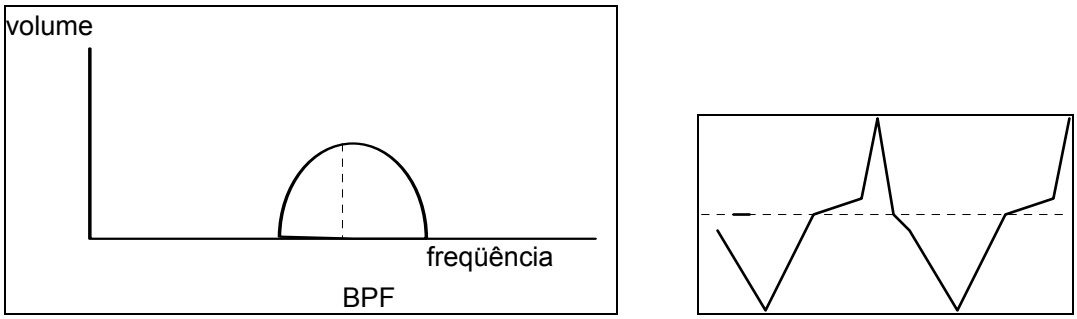


Figura 68 – Onda submetida ao filtro BPF com corte de freqüências agudas e graves

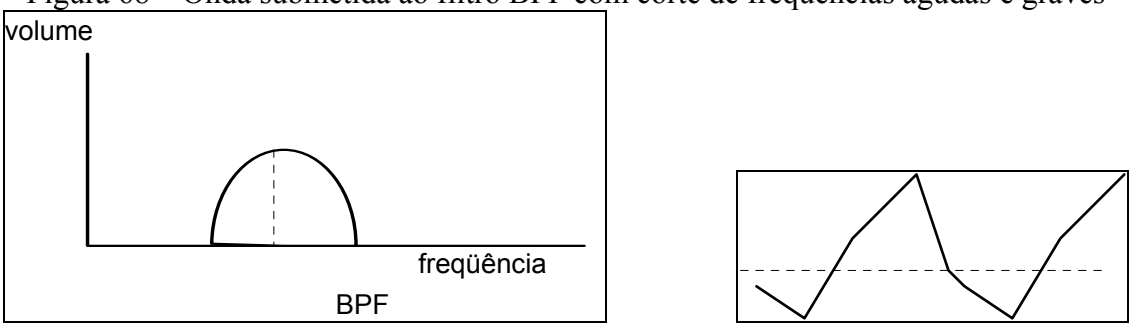


Figura 69 – Onda submetida ao filtro BPF com corte de freqüências agudas e graves

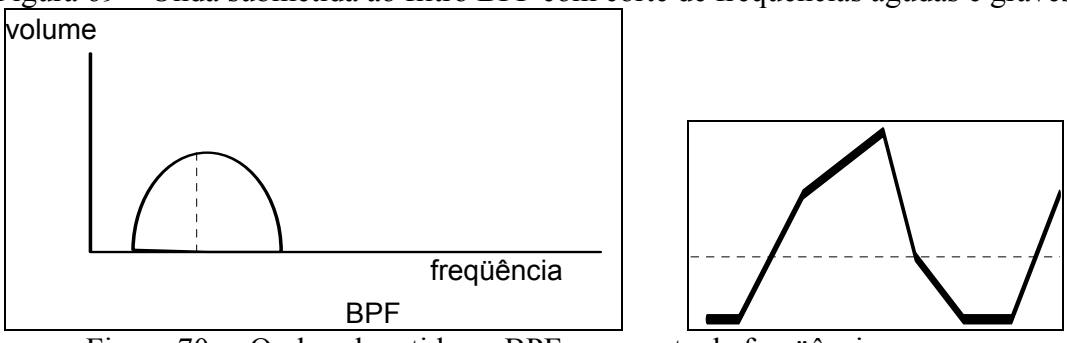


Figura 70 – Onda submetida ao BPF com corte de freqüências graves

Ajuste do ponto de corte para LPF:

Quando o filtro for LPF, abaixando o cutoff você elimina harmônicos agudos, tornando o som mais suave. Ao mesmo tempo ocorre um decaimento do volume. As figuras 71, 72, 73 e 74 mostram a modificação da onda quadrada pela posição do ponto de corte usando o LPF.

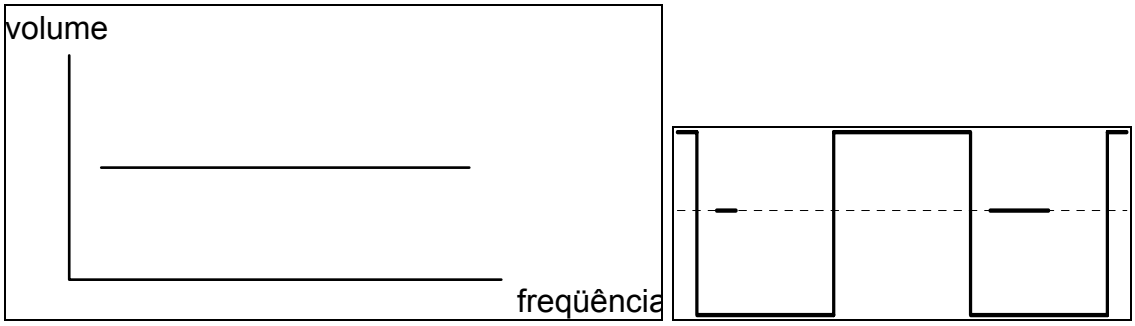


Figura 71 – Onda sem corte de freqüência

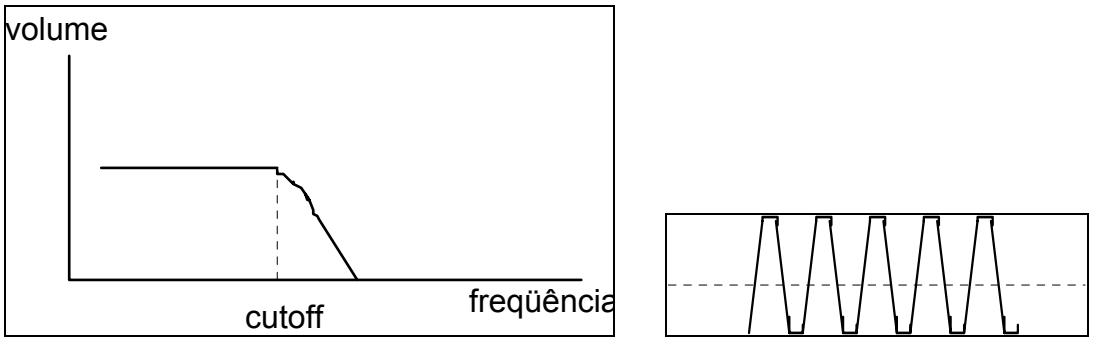


Figura 72 – Onda submetida ao filtro LPF com corte de freqüências agudas

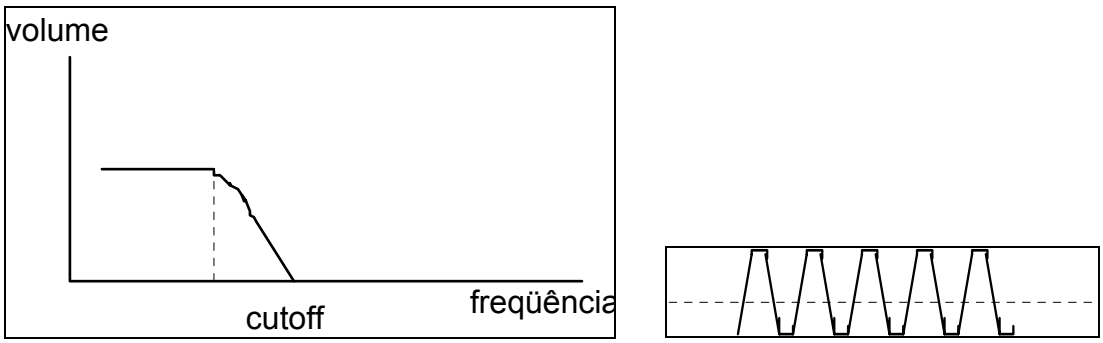


Figura 73 – Aumento no ponto de corte da freqüência

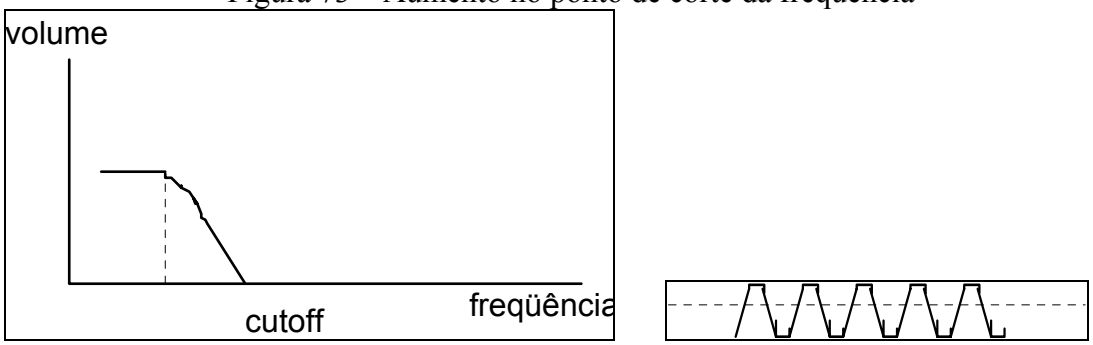


Figura 74 – Onda resultante após o corte das freqüências agudas

2.2.1.3 RESONÂNCIA – RESONANCE or Q

Resonância é o pico ou a acentuação da resposta de freqüência de um filtro numa freqüência específica. Elevando este valor, os harmônicos da região de cutoff são enfatizados produzindo um som característico. Com valores altos, um novo som é produzido além dos sons dos osciladores.

Exemplos gráficos de alteração da região de corte pelo incremento ou decremento da resonância:

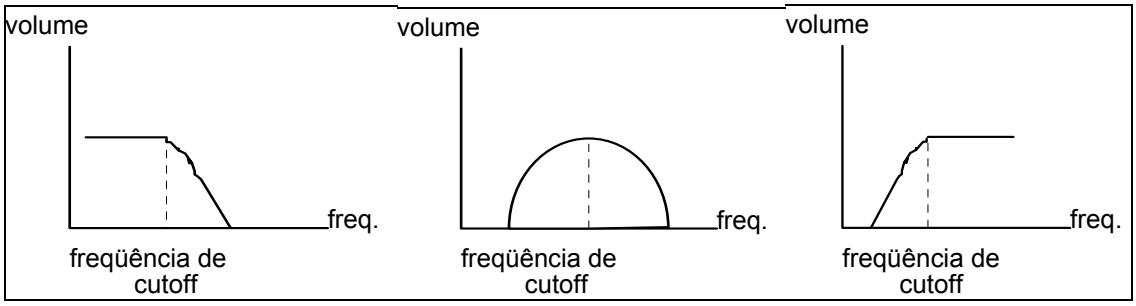


Figura 75 – Corte de freqüência sem Ressonância

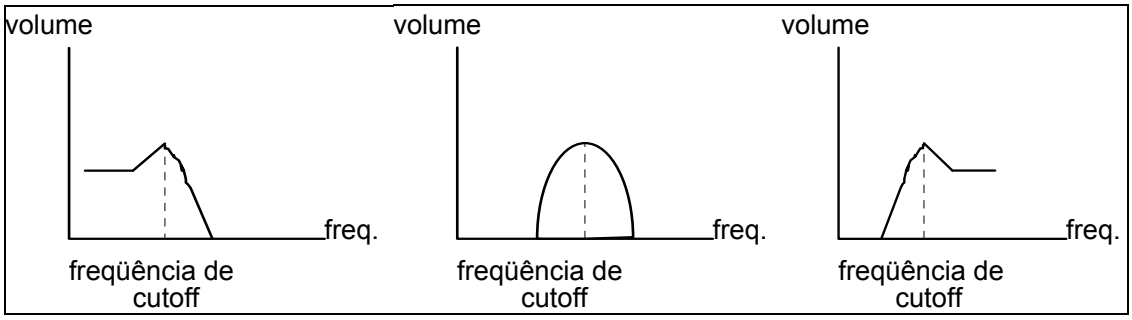


Figura 76 – Corte de Freqüência com Ressonância

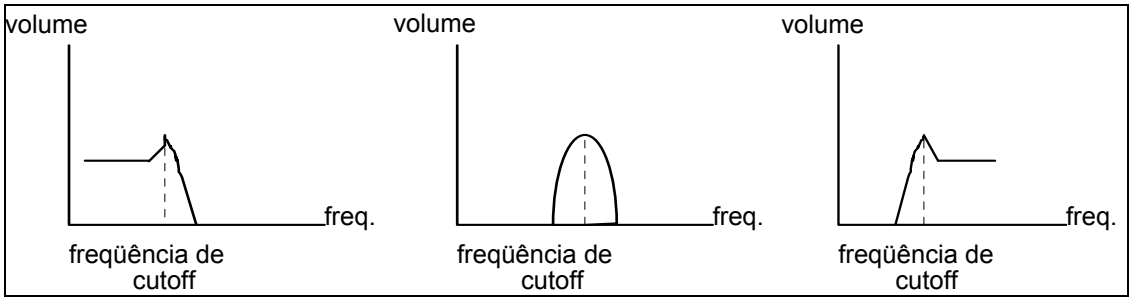


Figura 77 – Aumento da Ressonância no ponto de corte

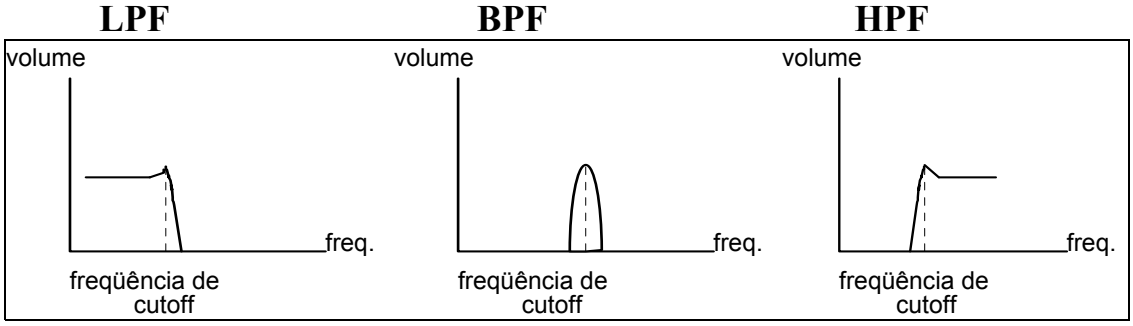


Figura 78 – Ressonância no ponto de corte

2.2.2 DCA - Digital Controlled Amplifier (Amplificador Controlado Digitalmente) ou TVA – Time Variant Amplifier (Amplificador controlado por Variação de Tempo)

Nos sintetizadores da linha Roland este modificador é chamado de TVA (time Variant Amplifier). É o circuito que corresponde ao elemento básico do som: o volume. Ele é responsável por controlar a intensidade do volume do som criado pelo WG e DCF. O DCA modifica a amplitude, e por conseqüência a intensidade do sinal de áudio. O DCA é muito utilizado em combinação com o ADSR a fim de controlar a amplitude de um som num transcurso de tempo.

2.2.3 EG – GERADOR DE ENVELOPE - Envelope Generator ou ADSR (Attack-Decay-Sustain-Release).

Controla a mudança de volume e timbre num transcurso de tempo**,** em outras palavras, controla o envelope. A curva básica do envelope consiste de quatro elementos - ADSR - que podem ser controlados independentemente.

Quando um teclado é utilizado para iniciar um envelope, dois sinais separados são produzidos. O sinal de gate indica quando uma tecla é pressionada ou solta (duração do envelope) e o sinal de trigger serve para iniciar a operação de ativar o envelope.

2.2.3.1 Controle do volume num transcurso de tempo (ADSR-DCA)

Estes parâmetros ajustam o tempo de ataque e decaimento, o valor de sustentação e o tempo de release do envelope de volume de amplificação. Em geral existe um potenciômetro para cada variável ADSR. Alguns sintetizadores podem ter até dois ou mais geradores de envelopes para serem usados por diferentes módulos.

Gráfico de ADSR:

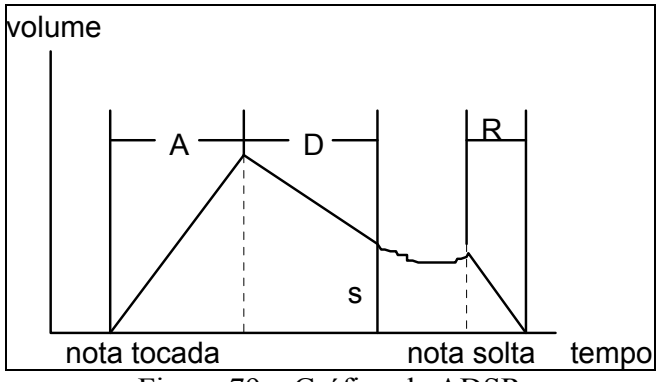


Figura 79 – Gráfico de ADSR

A: Tempo decorrido entre tecla acionada e máximo de volume.

D: Tempo entre o máximo de volume de som sustentado.

S: Volume do som sustentado estável – duração do volume enquanto a tecla estiver pressionada.

R: Tempo que leva para o volume entre a tecla solta e o fim do volume.

2.2.3.2 Controle da altura num transcurso de tempo (ADSR-WG)

Os parâmetros ADSR ajustam a freqüência da onda responsável pela altura tonal (afinação).

2.2.3.3 Controle do timbre num transcurso de tempo (ADSR-DCF)

Os parâmetros ADSR ajustam os tempos de ataque e caimento, volume de sustentação e tempo de release no envelope do filtro.

2.2.4 LFO - Low Frequency Oscillator (Oscilador de Baixa Freqüência)

O LFO (Low Frequency Oscillator) gera uma forma de onda com freqüência abaixo da capacidade de audição. É usada somente como fonte de controle de voltagem.

Quando a saída de um módulo de LFO (com forma de onda triangular) é ligada à entrada de um módulo de WG, a freqüência de áudio do WG começará a alternar para cima e para baixo de acordo com a faixa de freqüências do LFO. Essas flutuações periódicas de uma freqüência de áudio numa faixa abaixo de 20 vezes por segundo é chamada vibrato.

Por outro lado, se a saída de um LFO utilizando uma onda triangular for ligada à entrada de um VCA, então a modulação da amplitude, chamada de trêmolo, é produzida.

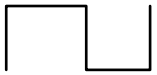
Um efeito de “wah-wah” pode ser obtido se o CUTOFF FREQUENCY de um filtro for controlado por um LFO utilizando uma onda senoidal ou triangular.

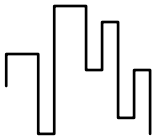
2.2.4.1 FORMAS DE ONDA DO LFO - WAVEFORM

O oscilador de baixa freqüência, em geral, utiliza 4 formas de onda. O som será modulado de acordo com a característica da forma de onda selecionada no LFO**.**

 (triangular): O som é modulado continuamente. Esta forma de onda é adequada para vibrato, etc.

 (dente de serra): Quando o som chega no valor mínimo, ele volta ao máximo imediatamente e recomeça a cair.

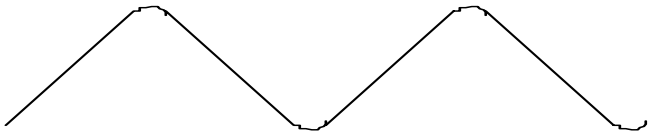
 (quadrada): O som alterna entre dois ajustes.

 (aleatória): O som alterna aleatoriamente entre vários ajustes.

2.2.4.2 Velocidade de Modulação – FREQUENCY / RATE

Esta opção tem a função de ajustar a velocidade de modulação de LFO.

Freqüência de modulação



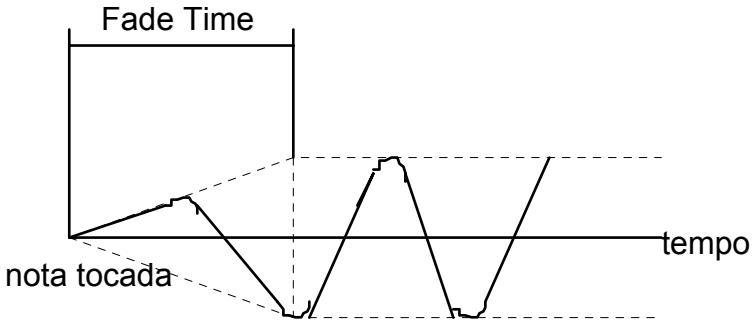
Dobro da freqüência

Ex 2. Aumentando o ajuste da velocidade (RATE) é possível obter um som semelhante a um telefone.

6.1.6.2.1 2.2.4.3 ATRASO NO INÍCIO DA MODULAÇÃO - DELAY TIME/ FADE TIME

Estas modulações de altura, timbre e amplitude obtidas a partir do LFO, são aplicadas com maior freqüência em notas longas inspiradas na forma de tocar instrumentos acústicos como, por exemplo, o violino. As notas iniciam sem vibrato e

após um tempo começam a vibrar. Isso resulta numa expressividade maior. Por essa razão os LFOs possuem a possibilidade de selecionar o tempo de atraso (delay time/fade). Esta função permite ajustar o intervalo de tempo desde que a tecla é pressionada até o aparecimento do efeito máximo de LFO.



6.1.6.3 RESUMO DOS PRINCIPAIS PARÂMETROS DA SÍNTESE

Qualidade do Som	Termo Técnico	Parâmetro do Sintetizador
Timbre	Conteúdo harmônico	Wave Generator
Brilho	Amplitude de harmônicos de alta freqüência	Filter Cutoff
Mudança de volume	Amplitude dinâmica	DCA Envelope
Mudança de tonalidade	Filtragem dinâmica	DCF Envelope
Vibrato	Modulação da nota	LFO aplicado no DCO (altura)
Tremolo	Modulação de amplitude	LFO aplicado no DCA (volume)
Altura tonal	Freqüência	Afinação do Oscilador (WG)
Velocidade com que as notas iniciam	Tempo de Ataque	DCA Envelope Attack
Velocidade com que as notas param	Tempo de Soltura	DCA Envelope Release
Percussividade	Tempo de Caimento	DCA Envelope Decay

6.1.7 Métodos de Síntese

Existem várias técnicas que podem ser utilizadas para sintetizar sons. Muitas delas utilizam o modelo de fonte e modificadores do som. O uso desse modelo fica mais claro na síntese subtrativa, mas ele pode ser aplicado a outros métodos de síntese como a S & S (Sample and Synthesis) ou modelagem física, por exemplo. Alguns métodos de síntese são mais complexos como a FM.

Uma metáfora de modelo desses métodos mais complexos pode ser muito difícil de compreender. Esta é uma das razões pela qual os métodos mais fáceis de compreender como a síntese subtrativa e a S & S obtiveram um sucesso comercial.

6.1.7.1 Síntese Analógica

Analógico refere-se ao uso de sinal de áudio que pode ser processado por filtros e amplificadores.

Subtrativa

A síntese subtrativa parte de um som rico em harmônicos que é filtrado e seu conteúdo harmônico retirado.

Os sons originais são, tradicionalmente, formas de ondas matemáticas tais como quadrada, dente-de-serra e triangular, sendo que os modernos sintetizadores subtrativos disponibilizam amostras digitais ao invés de um ciclo de forma de onda. A filtragem tende a ser um filtro resonante de passa-baixa onde as mudanças no ponto de corte da freqüência (cut-off frequency) desse filtro produzem o som de filtro "sweep" que é característico da síntese subtrativa.

Aditiva

A síntese aditiva adiciona ondas senoidais com diferentes freqüências para produzir o timbre final. O principal problema com este método é a complexidade no controle de uma grande quantidade de ondas senoidais.

6.1.7.2 Síntese Híbrida

A denominação de síntese híbrida é atribuida aos sintetizadores que não são completamente analógicos ou digitais. Esses sintetizadores foram de grande importância no início da década de oitenta quando houve a tranformação da tecnologia analógica para digital. Estas características híbridas se mantiveram em várias linhas de sintetizadores comerciais.

Com o crescimento atual do interesse pela síntese analógica a partir da década de 90, surpreendentemente alguns instrumentos começaram a ser projetados inspirados nos analógicos e muitas vezes, incorporando métodos da síntese híbrida. Métodos de síntese que combinam mais de uma técnica de síntese para produzir sons são denominados métodos compostos de síntese.

É possível dividir os sintetizadores híbridos em diferentes grupos de acordo com as suas partes digitais e analógicas:

I - Controle Digital de Parâmetros de síntese analógica.

II - Controle Digital do Oscilador (DCO) com o restante do instrumento analógico.

III - Oscilador Digital com modificadores analógicos e com controle digital dos parâmetros analógicos.

O método predominante de síntese usa geração digital do som e controle dos parâmetros com filtros e envolventes analógicos.

Osciladores Controlados Digitalmente

DCO's são as unidades digitais geradoras de som equivalentes aos VCOs. Elas podem ser consideradas como um caso especial de um oscilador básico de um ciclo sendo que só podem produzir as ondas clássicas como senoidal, quadrada, retangular e dente-de-serra. Os DCO's foram projetados para substituir os VCOs pois estes não eram estáveis na mudança de temperatura e causavam flutuações na altura tonal.

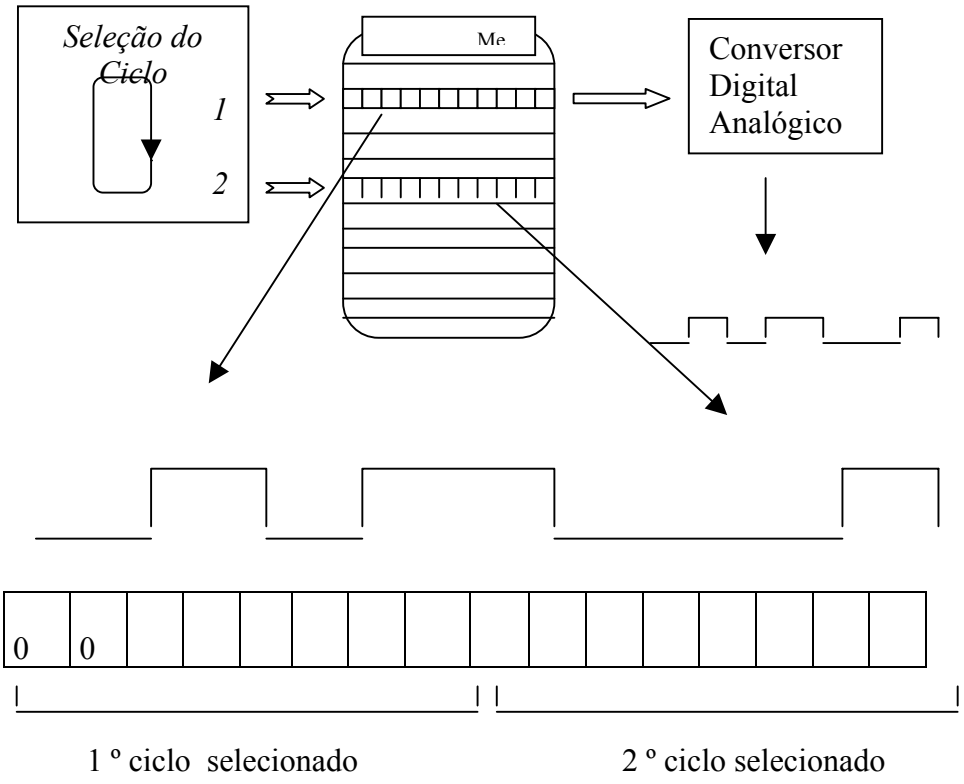
O primeiro tipo de DCO utilizado em sintetizadores usava um microprocessador para monitorar a afinação dos VCOs e afiná-los quando necessário.

A Segunda abordagem de DCO's em sintetizadores era baseada na idéia usada em chips de osciladores de órgãos elétricos que consistia, resumidamente, em um oscilador mestre controlado por cristal de quartzo que gerava freqüências suficientes para afinar as todas as notas do sintetizador.

Síntese por Tabela de Ondas (Wavetable)

A síntese por tabela de ondas estende a idéia de síntese subtrativa pois possibilita formas de ondas mais sofisticadas. Várias formas de onda podem ser armazenadas no sintetizador para depois serem dinamicamente selecionadas em tempo real.

É a forma de síntese mais comum em sintetizadores comerciais, computadores e cartões de sons. A síntese por Tabela de Ondas utiliza a memória como parte integral do processo de síntese pois o ciclo usado é dinamicamente selecionado pelo controle de memória. Um ciclo de forma de onda é armazenado na memória de um chip e sucessivos valores são carregados da memória e enviados para o Conversor Digital Analógico onde é produzida a saída da forma de onda. Um sintetizador wavetable utiliza vários ciclos de ondas localizados na memória.



Expressões como Vector Sintese (Síntese Vetorial) e Linear Arithmetic (Aritmética Linear) são utilizadas pelos fabricantes para referenciar pequenas diferenças na abordagem da síntese por Tabela de Ondas (Wavetable).

Existem pelo menos 4 métodos que são comumente empregrados na síntese por tabelas de ondas: single wavecycle (ondas de um ciclo), multiple wavecycles (múltiplos ciclos de ondas), sampling (amostragem) e interpolação de tabelas.

Ondas de Um Ciclo

Osciladores de um ciclo produzem formas de onda fixas, como os sintetizadores analógicos, embora exista muito mais opções na seleção de forma de ondas. O método mais simples do controle da forma de onda é provavelmente é o controle da largura de pulso. O pulso das formas de onda possuem em geral dois níveis: positivo e negativo em relação ao ponto central zero. No entanto, existem métodos que utilizam múltiplos níveis.

Múltiplos Ciclos de Onda

Neste método a Tabela de Ondas contém mais de um ciclo de onda sonora. O oscilador multi-ciclo possui vários ciclos de forma de onda que são gerados em seqüência, onde o mesmo conjunto de ciclos de onda é repetido continuamente. Ao contrário dos sons sintéticos gerados pelas formas de ondas, as amostras são na sua grande maioria de gravações de instrumentos “naturais”. Os Osciladores Multi-ciclo normalmente fazem a seleção de apenas um ciclo de forma de onda que é adicionada aos seguintes elementos:

- Concatenação de formas matemáticas: ciclo de onda senoidal, triangular, quadrada e dente-de serra em seqüência.
- Variação de simetria matemática e outras formas.
- Modulação no comprimento da onda (PWM) das formas de onda que mudam seu conteúdo harmônico ao longo do tempo.
- Formas de onda que mudam seu conteúdo harmônico ao longo do tempo, mas não numa seqüência regular e progressiva como a gerada pela PWM.
- Ruído onde mais ciclos propiciam um ruído ainda mais branco do que um oscilador de apenas um ciclo.

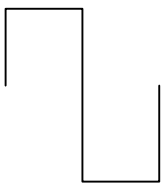
Amostragem (sampling)

A diferença básica entre o método de Múltiplos Ciclos de Onda e a Amostragem é que a Amostragem utiliza tabelas de ondas mais longas. Mesmo que a abordagem do método de Múltiplos Ciclos de Onda trabalhe com mais de um ciclo de onda, o tamanho da tabela de ondas é relativamente pequena e necessita de várias repetições (loops) para produzir uma nota de curta duração.

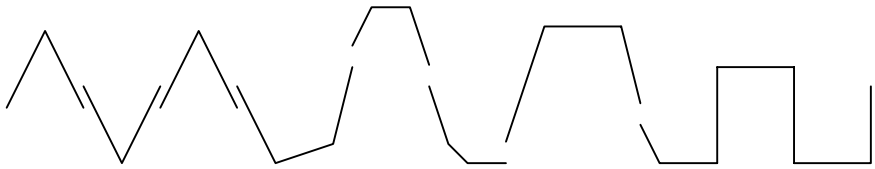
Interpolação de Tabelas

É um método que combina elementos da abordagem de Múltiplos Ciclos de Onda com os da abordagem da Amostragem digital. A interpolação vai gradualmente mudando a saída da amostra digital de uma tabela de ondas para a saída da amostra digital de outra tabela com o objetivo de produzir sons que variam ao longo do tempo. É um método usado para criar efeitos de "Sound Morphing".

OndaInicial



Onda final



Interpolação entre duas formas de onda

6.1.7.3 Síntese Digital

A tecnologia digital pode ser entendida como a representação numérica de sinais e o uso de computadores para o processamento dos mesmos. Os métodos de síntese digital são mais variados do que os métodos de síntese analógica sendo que novas maneiras de construir sons continuam sendo criadas e aprimoradas.

FM – Freqüência Modulada

Criada na Universidade de Standford por John Chowning é baseada nos mesmos princípios usados na transmissão de rádios FM. O primeiro experimento realizado por Chowning foi empregar um sinal de áudio (chamado de modulador) para controlar a freqüência de um oscilador (chamado de portador) para produzir sons ricos em harmônicos.

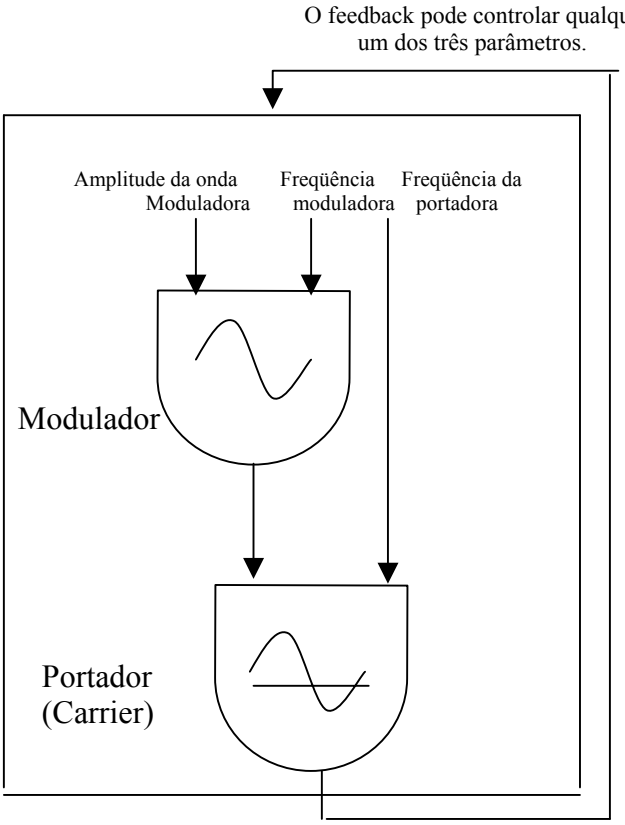
A abordagem mais básica de um instrumento baseado em síntese FM é composta de dois osciladores chamados de modulador (modulator) e portador (carrier). Esta simples estrutura é capaz de produzir uma grande quantidade de timbres distintos. Instrumentos FM mais complexos podem empregar vários moduladores e vários portadores combinados de várias maneiras [MIR 98].

A Yamaha introduziu sintetizadores comerciais no mercado com base na síntese FM. Mecanismos de FM foram encapsulados em módulos que só possibilitam manipulação de funções em alto nível chamados operadores. O usuário só tinha acesso a um número limitado de parâmetros para o controle dos operadores. Cada operador consiste basicamente de um gerador de envolvente e um oscilador que pode ser usado como portador (carrier) ou modulador (modulator).

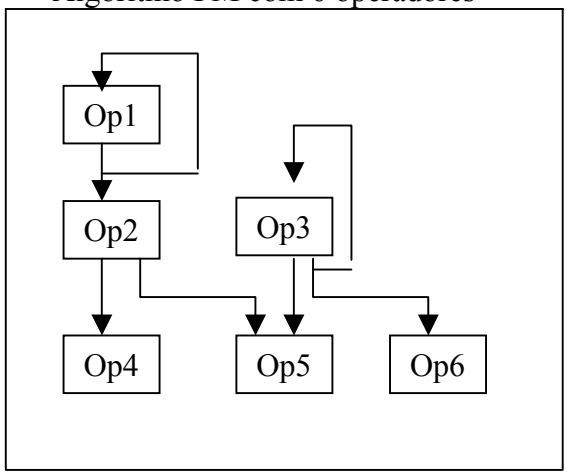
Os sintetizadores da Yamaha da linha DX são programados pela interconexão apropriada de um número de operadores a fim de formar um algoritmo. A Yamaha utiliza alguns tipos importantes de algoritmos: aditivos, pares, pilhas, múltiplos portadores e múltiplos moduladores. Até 6 operadores podem ser recombinados a fim de criar novos patchs a serem armazenados na memória. O primeiro protótipo de instrumento FM foi o Yamaha GS1 com 8 operadores, depois veio o GS2 com 4 operadores e a linha DX, em geral, com 6 operadores.

A seguir é apresentado o esquema de um operador de um sintetizador FM da Yamaha constituído por um gerador de envolvente e um oscilador. En seguida é apresentado um algoritmo FM com 6 operadores [MIR 98].

Operador FM composto por um Modulador e um Portador.



Algoritmo FM com 6 operadores



Síntese Waveshaping ou Distorção Não Linear

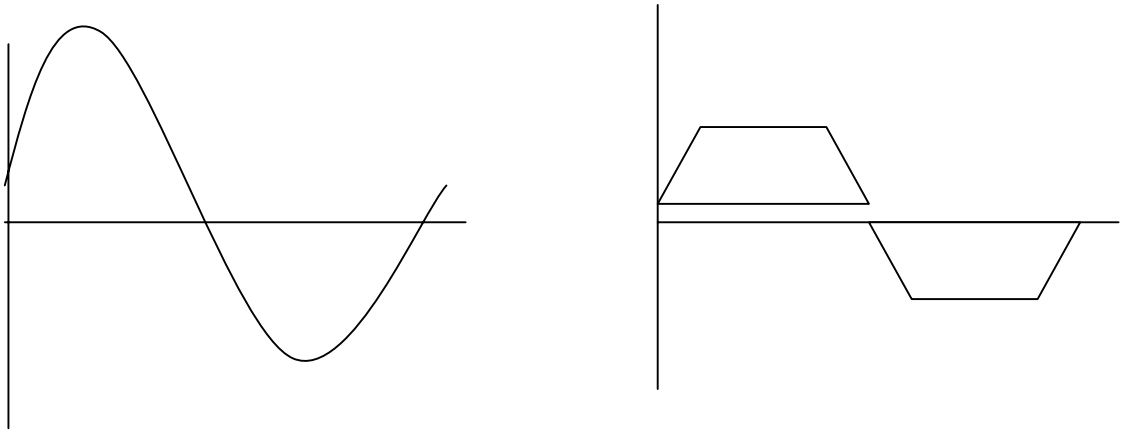
Este método de síntese consiste em compor um espectro pela aplicação de distorção à uma onda simples (senoidal). A técnica consiste em passar o som através de um unidade que distorce sua forma de onda de acordo com os parâmetros escolhidos pelo sintetista. A unidade de distorção é chamada de Waveshaper (formador de onda) e as especificações do sintetista são chamadas funções de transferência. Jean-Claude Riset realizou a primeira experiência com Síntese Waveshaping ainda na década de 60.

Para o entendimento do princípio da síntese Waveshaping utiliza-se a metáfora do amplificador a válvula.

Quando um som de cordas passa por um amplificador a válvula que tem seu volume elevado ao máximo, este amplificador irá saturar o som e cortar os picos. Se a amplitude do som é aumentado na sua origem, após entrar no amplificador, então os picos da saída irão ser cortados cada vez mais.

a) Ondas com maior amplitude fornecem um timbre mais brilhante.

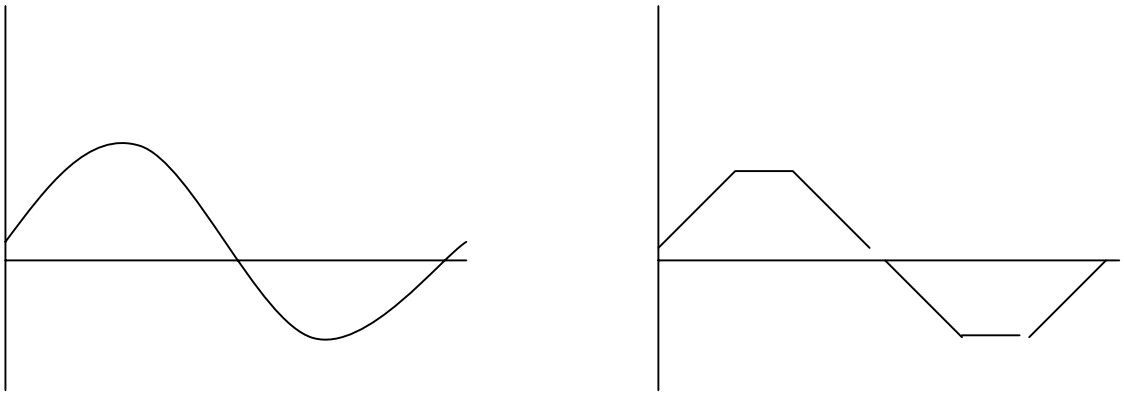
amplitude



freqüência

b) Ondas com menor amplitude fornecem um timbre menos brilhante.

amplitude



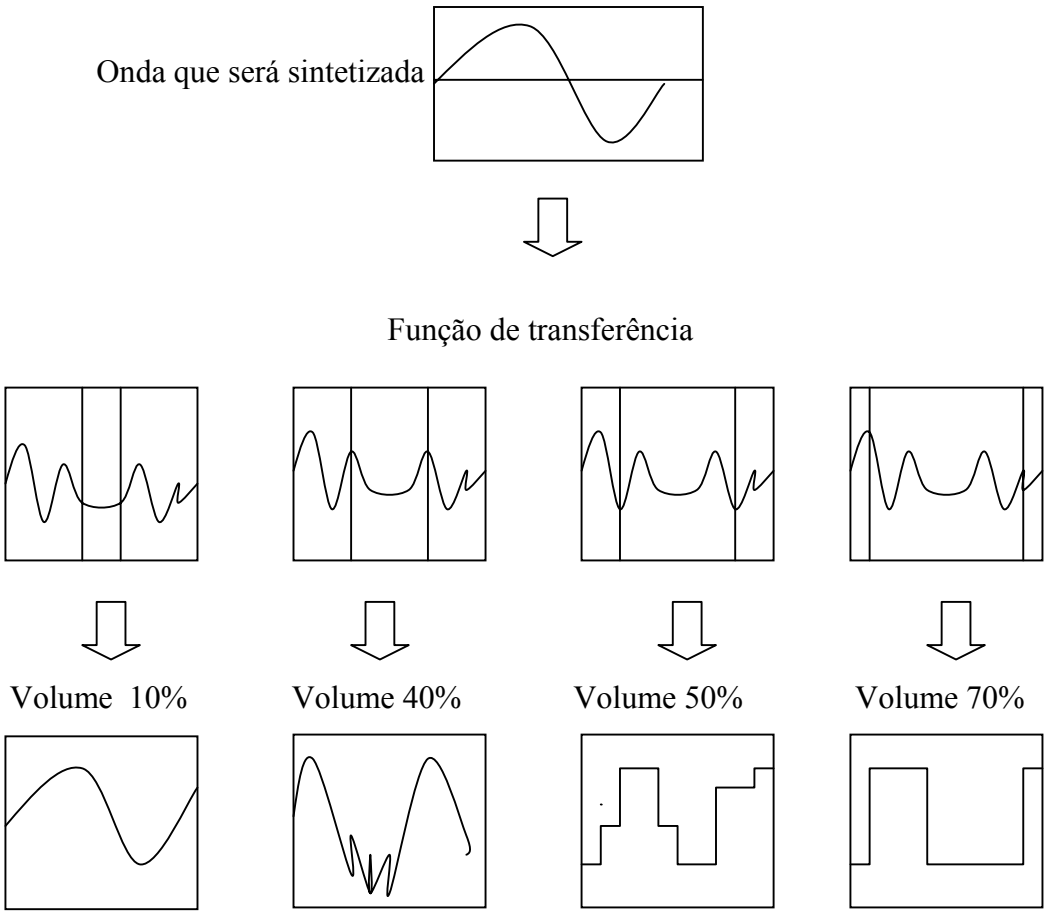
freqüência

Na Síntese Waveshaping, a riqueza do espectro harmônico resultante é proporcional à amplitude do sinal de entrada. Distorcendo o padrão da forma de onda seu conteúdo harmônico é alterado, na maioria dos casos mais harmônicos são adicionados do que subtraídos. Quando a síntese FM estava no pico de sua popularidade na metade da década de 80 a Casio utilizou técnicas de síntese baseadas na Waveshaping na linha de sintetizadores CZ, porém denominaram de síntese por distorção de fase. Alguns fabricantes utilizaram a síntese Waveshaping com muitas limitações. Por exemplo, a série de teclados Korg 01, embora sendo um instrumento S

& S, implementava técnicas de síntese Waveshaping, mas era uma forma muito limitada de função de transferência não linear.

Na síntese Waveshaping, ao contrário de um filtro de passa-baixa (low-pass filter), os harmônicos não precisam ser adicionados numa seqüência progressiva, podendo mudar para outras formas, tornando-se mais interessantes de ouvir.

No esquema abaixo, uma onda senoidal é submetida a uma função de transferência. As formas de onda resultantes mudaram em função do aumento do nível de entrada.

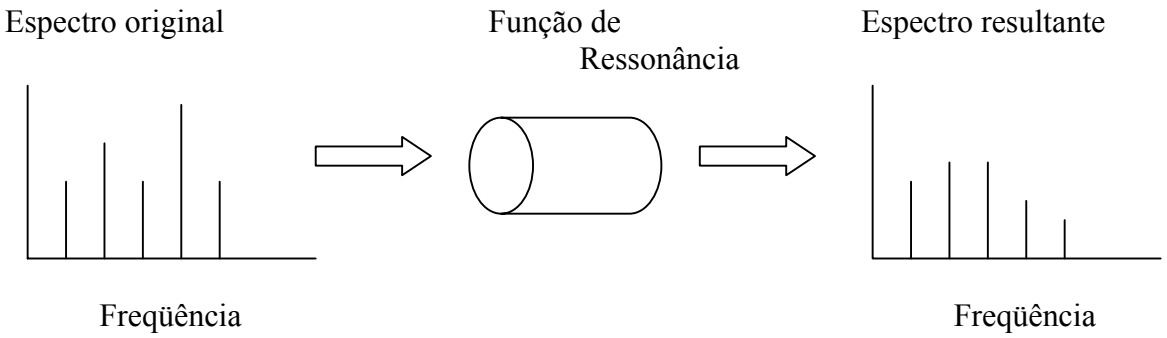


Síntese por Modelagem Física

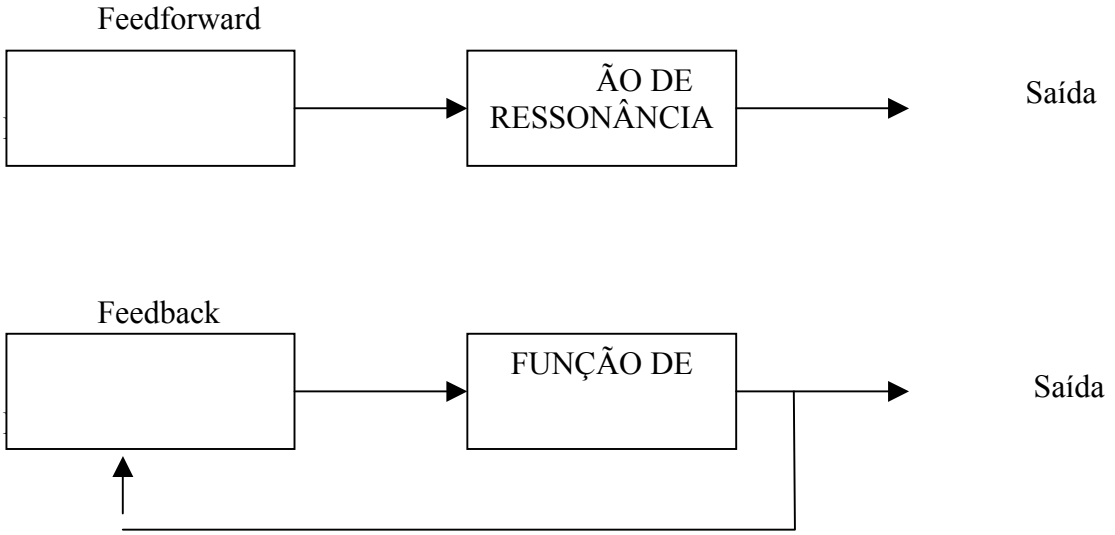
Este método é muito usado para simular instrumentos acústicos incluindo a expressividade do instrumento como o sopro dos metais e o trastejado das cordas. Métodos de modelagem física simplificados possibilitam a utilização dessa técnica na indústria para a produção de sintetizadores e módulos de som. O primeiro exemplo de uso comercial da modelagem física foi o Bontempi-Farfisa's MARS em 1990 seguido pela Yamaha que produziu o VL1 em 1993.

Na sua abordagem funcional, a Modelagem Física é baseada no princípio que o comportamento de um instrumento é determinado por dois componentes principais: a fonte e a função de resonância. O papel do componente fonte é produzir um sinal puro que será modificado pela função de ressonância. Uma variação no espectro pode ser

obtida pela variação nas propriedades acústicas da função de ressonância. Por exemplo, podemos soprar num tubo cilíndrico que este irá produzir um som que irá variar de acordo com as dimensões do tubo [MIR 98].



O ressonante atua como um banco de filtros aplicados à um sinal fonte não linear (noise). A interação entre a fonte e os componentes ressonantes dividem-se em duas categorias: feedforward e feedback [MIR 98].



O *Feedback* produz freqüentemente resultados mais convincentes porque essa interação entre a fonte e o ressonante existem nos instrumentos acústicos. Esta interação de *feedback* pode criar efeitos que ouvimos como os detalhes de uma performance. Por exemplo, a freqüência de vibração da palheta do saxofone é fortemente influenciada pelo *feedback* acústico da ressonância do tubo do instrumento [MIR 98].

A síntese por modelagem física não técnicas de amostragem digital para imitar instrumentos acústicos, ao invés disso, calcula todas as características do som através de fórmulas matemáticas possibilitando recriar os sons nos mínimos detalhes.

6.1.8 ÁUDIO DIGITAL

6.1.8.1 Gravação do som

Se o sinal do microfone for introduzido em um gravador de fita magnética, seu circuito transformará o sinal elétrico magnético (preservando ainda as características oscilatórias originais do som), que pode então ser registrado na fita (graças ao alinhamento magnético das partículas metálicas existentes na superfície da fita). Uma vez gravado na fita, esse sinal magnético pode ser reproduzido, desde que se efetue o processo inverso, recompondo o sinal elétrico a partir do sinal magnético registrado na fita, e depois transformando o sinal elétrico em acústico, através de um sistema composto de amplificador e alto-falantes [YAV 91].

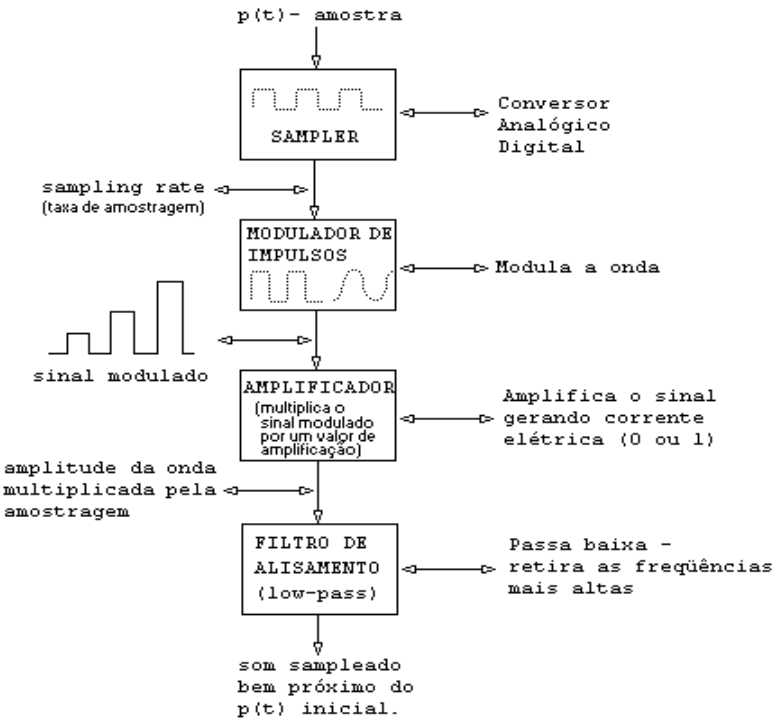
Pode-se representar as oscilações de uma corda, por exemplo, usando-se um gráfico que mostre as variações que ocorrem na pressão do ar, no decorrer do tempo. O mesmo gráfico serve também para representar as variações da tensão elétrica no tempo (no caso do sinal convertido pelo microfone) ou as variações do campo magnético da fita no tempo (no caso do sinal gravado), pois todos esses sinais mantêm as mesmas características oscilatórias.

O processo de gravação usado pelos gravadores convencionais de fita magnética (cassete ou rolo) possui certos inconvenientes, alguns deles inerentes ao próprio meio usado para a gravação, que é a fita magnética. Por exemplo, quando o sinal elétrico (representando a onda sonora) é registrado na fita (convertido em sinal magnético) há uma perda de fidelidade, com a distorção do sinal original e a eliminação de alguns componentes harmônicos de alta freqüência. Além disso, existe ainda a adição de ruído gerado pela própria fita [CAR 92].

Como solução alternativa a esse processo de gravação convencional, existem hoje processos mais aprimorados para o registro de sinal de áudio, que usam tecnologia digital.

6.1.8.2 Princípios Básicos do Processo de Amostragem Digital

O processo de representar numericamente o som é chamado de digitalização. O som é a variação da pressão do ar. Sendo assim, a forma de produzir um determinado som depende da maneira como a pressão do ar varia. O som da nota Dó central de um piano tem uma variação periódica de pressão que repete 256 vezes em um segundo. Isto significa que sua freqüência de vibração é de 256Hz.



6.1.8.3 Seqüência do processo de amostragem.

O alto-falante produz variações de pressão do ar por intermédio do movimento vibratório do cone de papel, em que uma bobina de fios é submetida a um campo magnético. Isso só acontece porque uma voltagem elétrica é aplica a bobina. Os computadores são máquinas digitais que trabalham somente com o zero e o um. Para um computador recriar um fenômeno analógico como o som, o computador tem que primeiro converter os dados analógico em digitais. Isto é feito através do circuito denomidado conversor analógico-digital (ADC). Quando os sons gravados digitalmente precisam ser reproduzidos, se faz necessário um processo inverso. O conversor digital-analógico (DAC) transforma os dados digitais novamente em analógicos. Todo o circuito analógico para a produção de som, seja um toca disco a laser, um "digital audio tape" (DAT) ou um computador, realiza a conversão digital para analógica a fim de reproduzir o som [CAR 92].

"Sampling" é a técnica de fazer um determinado número de amostragens de uma certa forma de onda em um determinado espaço de tempo. O processo de amostragem é realizado pela determinação de N pontos de amplitude da onda e a representação destes pontos por números que sejam proporcionais as amplitudes. Na figura 80, cada um dos números contidos na memória do computador é a amplitude da onda em um único instante de tempo. Uma função contendo freqüências limitadas pode ser representada por uma seqüência de números [YAV 91].

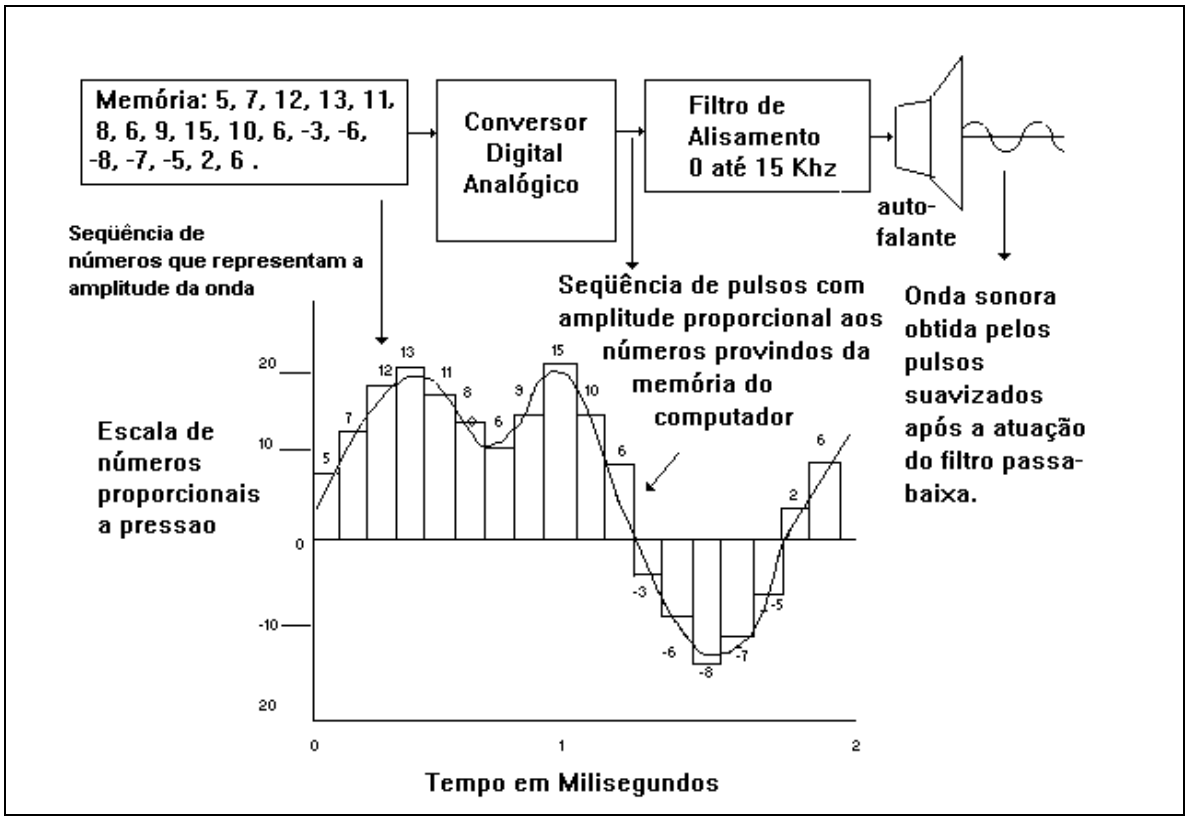


Figura 80 – Seqüência do processo de amostragem

6.1.8.4 O processo de amostragem.

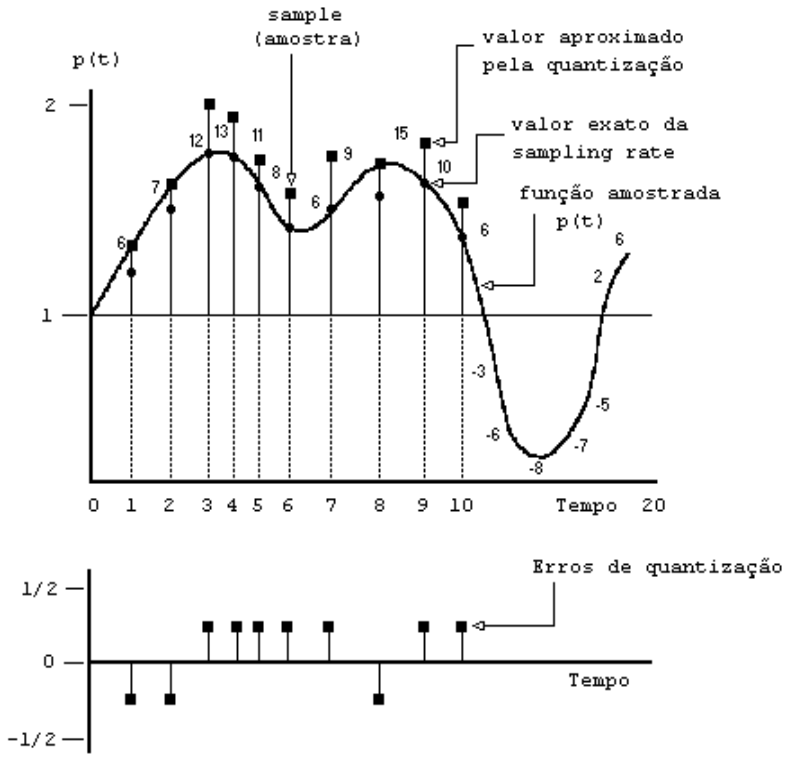
A taxa de amostragem ("sampling rate" - SR) deve ser no mínimo o dobro da freqüência que esta sendo amostrada, ou seja, a freqüência amostrada deve estar entre 0 e a SR/2. Logo, uma função que contenha freqüências neste limite pode ser representada por amostragem digital. Ao exceder este limite, o processo de "sampling" gera erros denominados de "foldovers". Tais erros são distorções e reflexões de freqüências ouvidas na escala entre 0 e a SR.

O erro foldover, visto no gráfico abaixo, pode ser calculado usando a fórmula

Fr= SR - F

onde :

Fr é a freqüência "foldover"
Sr é a taxa de amostragem
F é a freqüência maior que SR/2 e que foi amostrada gerando o erro.
Os "foldovers" podem ser ouvidos como harmônicos enfatizados ou distorções do som.



6.1.8.5 Quantização

O processo de quantização, mostrado na figura anterior, é usado quando o nível dinâmico (amplitude) de uma determinada amostra fica entre dois pontos. Então a quantização consiste em arredondar o nível dinâmico para o ponto mais próximo.

O computador da figura contém apenas dois dígitos. Dessa forma todos os números contidos na memória e que representam as amplitudes entre 10.5 e 11.5 serão representados pelo número 11. Se o computador em questão utilizasse até três dígitos, então os números entre amplitudes 10.65 e 10.75 seriam quantizadas para 10.70. Da mesma forma para um nível maior de detalhamento na representação das amostras.

6.1.8.6 Qualidade do Som

Três fatores predominantes determinam a qualidade do som:

Número de amostras tiradas por segundo também chamada de sample rate. Quanto maior este número melhor o detalhamento e mais fiél ficará o som.

Número de sons usados para cada amostra também chamado de a resolução da amostra. Neste caso quanto maior a resolução, mais precisa será a representação do som original no momento em que a amostra foi tirada. Os sistemas digitais usam *números binários,* cujos algarismos (dígitos) são chamados de *bits,* e podem ter valor "0" ou "1". Quanto maior for o número de bits, melhor será a capacidade de resolução do equipamento, mas, infelizmente, também mais alto será o seu custo. Quanto mais bits houver, mais números podem ser representados, conforme mostra a tabela abaixo.

Quantidade de bits	*Números representados*
3	*8*
4	*16*
8	*256*
10	*1.024*
12	*4.096*
14	*16.384*
16	*65.536*

E qual seria a resolução ideal? Obviamente, aquela que use o maior número possível de bits. Mas, como isso tem um custo, é necessário optar pela situação otimizada.

O projeto e a qualidade do circuito de áudio usado para filtragem e amplificação do som.

Um sampler profissional tem a capacidade de fazer 44.100 amostras por segundo, ou seja, produz uma taxa de amostragem de 44,1 kilohertz (milhares de ciclos por segundo, abreviado por KHz). Isso também acontece no caso de outros equipamentos como computadores destinados ao tratamento do som. Alguns equipamentos, no entanto, fazem amostragens de mais baixa qualidade com taxas de 22khz. Neste caso a freqüência que pode ser gravada com precisão é menor. A taxa de amostragem digital tem que ser sempre pelo menos o dobro da freqüência amostrada para que o som seja gravado digitalmente com precisão. Este limite máximo é chamado de freqüência de Nyquist, em honra ao cientista que descobriu o fenômeno. Especificamente em termos de computador, a taxa de amostra afeta a qualidade do som porque determina a precisão com a qual o computador detecta variações dinâmicas, ou seja, mudanças na altura do som que está sendo examinado. Quanto mais bits forem atribuídos a cada amostra, maior será a precisão com a qual o som é medido. Um equipamento profissional usa uma taxa de 16 bits à 24 bits, permitindo o reconhecimento e a reprodução de milhares de níveis de alturas diferentes. Em geral com a utilização de placas de expansão qualquer computador atual pode ser transformado num equipamento profissional de amostragem digital fornecendo outros vários recursos como a edição de sons pré-gravados.

Após a apresentação desses parâmetros fundamentais sobre a digitalização de áudio, vejamos alguns números práticos interessantes. Tomemos o exemplo da amostra de um sinal de áudio com duração de 1 segundo. Se for usada uma freqüência de amostragem igual a 44.1 kHz, significa que serão efetuadas 44.100 amostragens do sinal. Se as amostragens são representadas por números de 16 bits (2 bytes), serão usados 44.100 x 2 bytes, ou seja, 88.200 bytes. Dessa forma, podemos calcular que para digitalizar 1 minuto de música com essa qualidade, seriam necessários 5.292.000 bytes, ou seja, mais ou menos 5 Megabytes (MB) de memória. Um disco CD, por exemplo, que pode conter cerca de 75 minutos de música digitalizada (44,1 kHz, 16 bits), possui capacidade de armazenamento de 650 MB.

Exemplo:

Freqüência de amostragem. 44,1 KHz
Resolução – 16 bits
44.100 * 16 = 705.600 bits ou 88.200 byte (86,15 kb)
Para digitalizar um minuto de música com qualidade de CD.
88.200 * 60 = 5.292.000 bytes ou +/- 5 MB
(10 Mb em estéreo)

6.1.8.7 Placas de Áudio

A placa mais popular do mercado é a Creative Labs Sound Blaster Awe 64 Gold Card, produzida pela Creative Labs. Além de possuir um sintetizador interno, a *Sound Blaster Pro* também pode funcionar como interface MIDI, sendo que para isso é necessário acoplar um adaptador opcional *(MIDI Kit)* com o qual a placa pode então *falar* MIDI com outros instrumentos externos e, assim, um software seqüenciador, por exemplo, pode comandá-los, usando a *Sound Blaster Pro* como interface.

Outras placas de áudio, como a Multisound (Turtle Beach) e a RAP – 10 (Roland), também possuem sintetizadores e interface MIDI, além da parte de áudio digital.

6.1.8.8 Formatação dos dados de Áudio

Algumas placas de áudio utilizam processo de compactação de dados, de forma a reduzir a quantidade de bytes que representam digitalmente o som, e, por conseqüência, o tamanho dos arquivos onde esses dados são armazenados.

O padrão adotado pela Microsoft para o arquivamento de dados de áudio é o arquivo do tipo WAVE (extensão.WAV), que suporta diferentes formatos de áudio digital: amostragem PCM com resoluções de 8 ou 16 bits, mono ou estéreo, e podendo utilizar um dentre três valores de freqüência de amostragem: 11,025 kHz, 22,05 klU ou 44,1 kHz. O formato WAVE, portanto, tem sido adotado por todos os softwares de áudio atuais, bem como é suportado por todas as placas.

O fato de uma placa suportar diferentes tipos de formatos é muito importante, pois pode ter grande utilidade nas aplicações utilizadas. Dessa forma, é importante que se tenha esses aspectos em mente quando se trabalha com áudio digital pois, dependendo da aplicação, é mais interessante piorar a qualidade em favor da economia de bytes. Um exemplo seria o caso da gravação de uma narração a ser usada em um software aplicativo de multimídia: como o som certamente será reproduzido em alto-falantes de baixa fidelidade, pode-se reduzir a resolução e/ou a freqüência de amostragem, pois a qualidade final não seria tão comprometida. Já no caso da digitalização de sons mais complexos, como peças musicais ou determinados efeitos sonoros, isso pode não ser adequado.

Portanto, é muito importante observar o tipo de aplicação e refletir sobre a relação *custo- benefício* de cada situação, para que se possa obter a melhor qualidade necessária dentro do melhor preço possível. Por essa razão, algumas placas oferecem diferentes opções quanto à qualidade do áudio digitalizado.

Alguns detalhes relativos às técnicas mais comuns de amostragem merecem ser analisados mais detalhadamente: eles se relacionam com o multi-sampling (ou amostragem múltipla) e com os loops **(**ou laços de repetição).

Como já citado anteriormente, sabemos que a reprodução de um sinal amostrado se dá com uma velocidade de acesso à wave table determinada pela tecla pressionada. Numa primeira abordagem, poderíamos imaginar que, uma vez tomada uma única

amostra de um determinado som - por exemplo, de um coral - seria possível reproduzi-la no sample playback transportada para qualquer outra freqüência.

Isto na verdade não acontece, pois uma amostra tomada num determinado tom, quando transportada alguns tons acima ou abaixo, passa a perder suas características próprias e o resultado disso irá soar como se tivéssemos um disco gravado em 45 rpm e o tocássemos em 78 rpm ou em 33 rpm.

A solução para esse problema é dada realizando-se várias amostragens do som original, executado em diversos tons, estabelecendo a cada amostra uma região bem delimitada do teclado, na qual a transposição do tom original pode ser realizada sem problemas de degeneração.

Em determinados equipamentos, a região de transição do teclado pode ser claramente percebida, ao se executar em sample playback uma escala de um som amostrado, pois quando o tom começa a perder sua clareza, logo em seguida já se pode observar uma sensível melhora na qualidade do timbre, devido à mudança do tom amostrado. Existem, atualmente, alguns samplers que, no intuito de minimizar o efeito de se sentirem as regiões de transição, permitem que se sobreponham as amostras do final de uma região com o início da seguinte - esta técnica é conhecida como "cross-fading".

Com relação à execução de loops, temos sua aplicação típica em sons de órgão ou de strings (ou cordas). Como estes sons, na grande maioria dos casos, precisam ser sustentados por vários segundos, já pensou no tamanho de memória necessário para armazenar, por exemplo, 10 amostras de 15 segundos cada?

O que ocorre, na realidade, é um processo bem mais engenhoso, que consiste em se reproduzir a amostra desde o ponto onde ela se inicia até o seu final e, caso a tecla ainda permaneça pressionada, retorna-se a um ponto predeterminado do espaço de tempo da amostra, repetindo-se o processo até a liberação da tecla. Alguns tipos de samplers permitem que, além da escolha dos pontos de início e término do loop, se possa escolher também um dentre vários tipos de loop ou até mesmo um loop a ser efetuado depois que a tecla é liberada [CAR 92].

Um erro cometido constantemente é a confusão entre um equipamento sampler e outro sample playback: o primeiro pode digitalizar formas de onda e armazená-las na memória, o outro pode criar sons, usando como base uma forma de onda que foi digitalizada e previamente armazenada na memória pelo fabricante. Portanto, todo sampler é obviamente um sample playback, pois pode tratar e executar as formas de onda que ele próprio armazenou; mas um sample playback pode somente tratar e executar formas de onda que já se encontrem em sua memória, sejam elas pré-gravadas ou carregadas via disquete, cartão ou transferência direta de um outro sintetizador ou sampler. Lembramos que tanto os samplers como os sintetizadores podem ser encontrados em forma de racks, o que permite que em um espaço bem reduzido possamos ter acesso a uma grande variedade de timbres diferentes, tocando em um único teclado.

Gravadores Digitais {
- Computadores + Placa
- Computadores dedicados
- DAT (Digital Audio Tape)
- ADAT (Digital Audio Tape Multitrack)
- Minidisk
- Gravador de CD

6.1.8.9 Processamento do Som

Equalização do som

Conjunto de filtros sintonizados, agrupados de forma a cobrir toda a faixa de freqüências de áudio, cada um deles com capacidade de operar amplificando ou atenuando o sinal de entrada [CAR 92].

A análise do sinal de áudio é feita não como forma de onda, mas como um espectro de freqüências que mostra seu conteúdo harmônico.

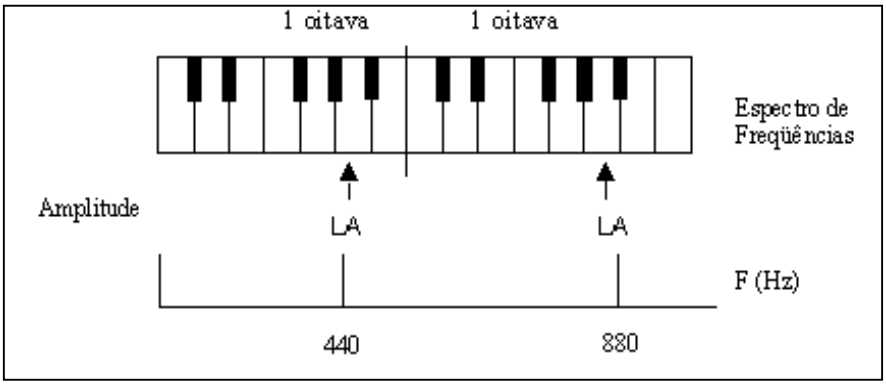


Figura 81 – Equalização do Som

Cada um dos componentes do timbre pode ser tratado de forma individualizada.

O filtro sintonizado permite a escolha de uma determinada freqüência contida num certo timbre.

A resposta em freqüência de um filtro sintonizado possui três características principais.

Ganho: Fator de amplificação que o filtro impõe ao sinal de entrada. É expressa em dB (decibéis e pode ser compreendido como o volume aplicado a uma dada freqüência.

Freqüência Central: Freqüência na qual o filtro apresenta seu ganho máximo.

Largura de faixa (Banda Passante): Região de freqüências onde a amplificação do filtro se encontra dentro de uma variação de até 3 dB, abaixo do ganho da freqüência central.

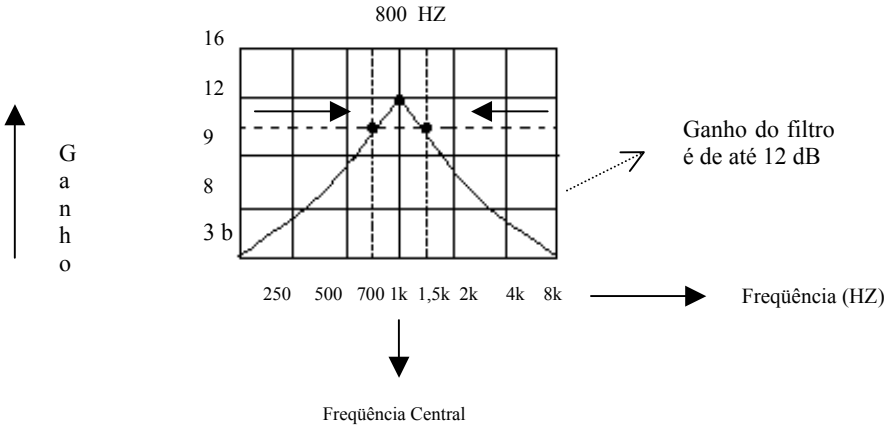
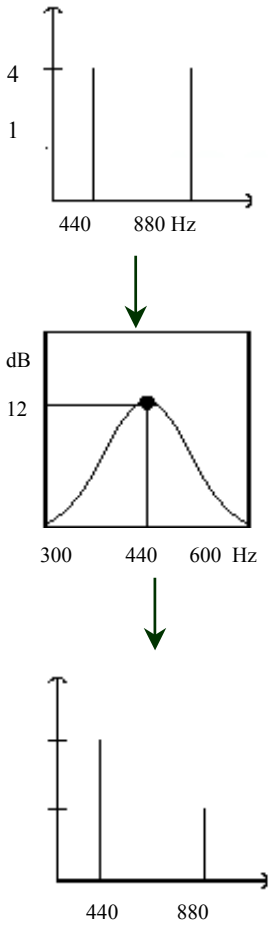


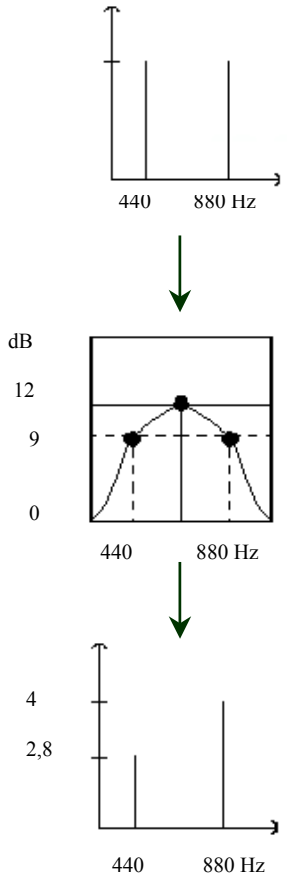
Figura 82 – Resposta em freqüência de um filtro

Respostas de Filtros Sintonizados

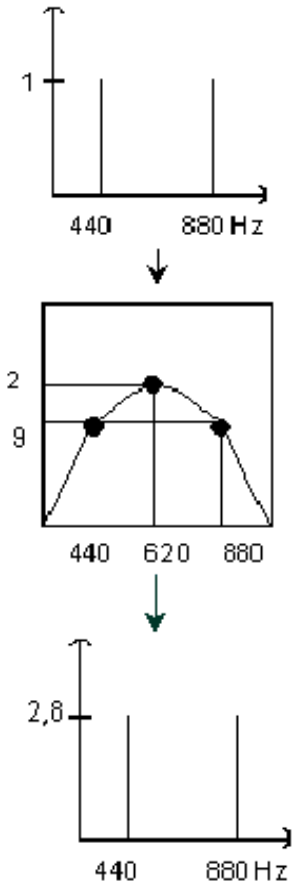
a) Filtro muito seletivo em uma das freqüências



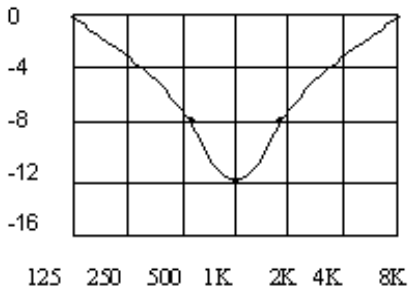
b) Amplificação pouco seletiva de uma das freqüências

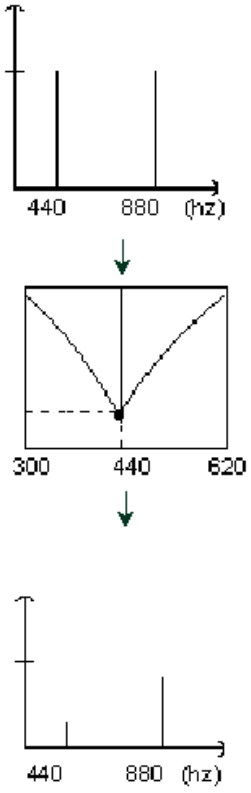


c) Filtro pouco seletivo na média das freqüências



Outra possibilidade de atuação de um filtro sintonizado é a atenuação do sinal de entrada, ao invés da amplificação. Valem para este caso todas as características anteriores citadas, levando-se em conta que o sinal da saída será sempre menor que o sinal de entrada.





EQUALIZADORES

Gráficos: Os filtros estão sintonizados em freqüências fixas de operação disponibilizando o ajuste do ganho e atenuação em cada freqüência.

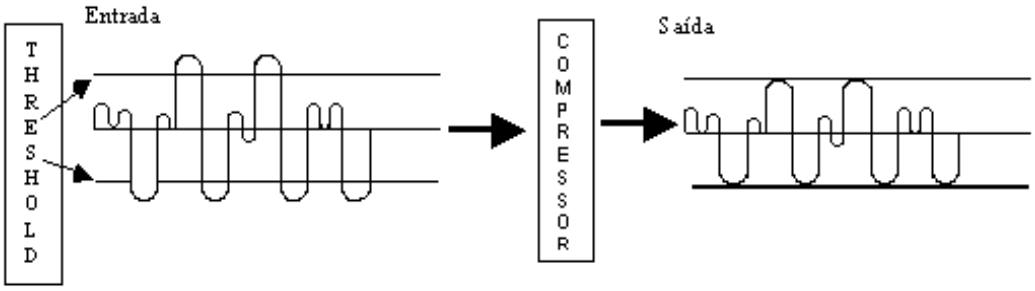
Paramétricos: Disponibiliza o posicionamento em freqüências variadas.

Compressão de Áudio

Compressor: Realiza operações sobre a soma fazendo com que porções de maior intensidade de um deteminado programa musical sejam atenuadas, ou seja, tenham sua amplitude reduzida.

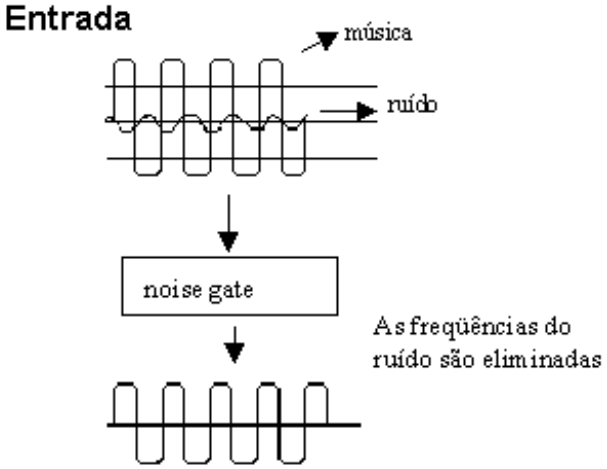
Threshold (nível de limiar): É o nível de sinal a partir do qual o sinal de entrada começa a ser atenuado pelo compressor [CAR 92].

Exemplo de atenuação do compressor:



Redutor de Ruído (Noise Gate)

O redutor de ruídos tem a função de suprimir um porção de sinal que entra abaixo do nível ajustável, eliminando juntamente com o sinal, o ruído indesejado. O nome GATE se justifica pela abertura ou o fechamento de uma passagem para determinado nível de sinal [CAR 92].



6.2 MÓDULO: FORMAS DE ESTRUTURAR PENSAMENTO E OPERAÇÃO DE COMPUTADORES

O conhecimento computacional necessário para a programação de computadores para música possui dois enfoques: o enfoque operacional e o enfoque lógico-matemático.

O enfoque operacional está relacionado com a aptidão e a experiência que o músico possui para utilizar o computador como um usuário final. Por ser um conhecimento prático, uma explanação sobre o assunto neste anexo de nada adiantaria. Para aprender a operar computadores, o usuário necessita ter contato com o ambiente e os programas que são mais utilizados para desenvolver sua capacidade de operação.

Para o enfoque Computacional, segundo [MIR 2001], é necessário o conhecimento lógico-matemático para que seja possível representar o pensamento de forma estruturada. A seguir, com base na obra de [MIR 2001], é apresentado um conjunto de conceitos lógico-matemáticos que devem ser de conhecimento do músico antes de iniciar o estudo de programação através do MEPSOM.

6.2.1 Matrizes

Uma matriz é um arranjo de números em colunas e linhas. Por exemplo:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

Esta é uma matriz 3 x 4, pois possui 3 linhas e 4 colunas.

Um número em uma matriz é chamado de elemento e é referido na forma de E C X L (coluna x linha). Uma matriz que possui o mesmo número de colunas e de linhas é uma matriz quadrada, e, neste tipo de matriz, a linha de número que vai do primeiro elemento até o último chama-se diagonal.

Quando as colunas e linhas são invertidas a matriz resultante passa a ser chamada de matriz transposta. Por exemplo:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \qquad A' = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}$$

A utilização de matrizes em computação e música é freqüente, principalmente quando parâmetros musicais precisam ser armazenados sob a forma numérica. É possível modificar esses parâmetros musicais numéricos através de operações matemáticas[MIR 2001]. A adição/subtração de elementos de matrizes é realizada pela simples soma de seus elementos um a um, por exemplo.

$$\begin{bmatrix} 2 & 2 \\ 3 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 3 \\ 5 & 5 \end{bmatrix}$$

A multiplicação de matrizes pode ser realizada entre duas matrizes ou entre uma matriz e um número simples, por exemplo:

$$\begin{bmatrix} 2 & 3 & 1 \\ 5 & 7 & 10 \end{bmatrix} \times 2 = \begin{bmatrix} 4 & 6 & 2 \\ 10 & 14 & 20 \end{bmatrix}$$

A multiplicação entre matrizes é feito quando o número de linhas da primeira matriz é igual ao número de colunas da segunda matriz. Uma vez que isto ocorre, os seguintes cálculos podem ser efetuados: a multiplicação de cada elemento das linha da primeira matriz pelo seu elemento correspondente nas colunas da segunda matriz, os elementos resultantes desta multiplicação devem ser somados para formarem um único elemento. Exemplo:

$$\begin{bmatrix} 3 & 7 & 1 \\ 2 & 4 & 0 \end{bmatrix} \times \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 6+14+2 \\ 4+8+0 \end{bmatrix} = \begin{bmatrix} 22 \\ 12 \end{bmatrix}$$

6.2.2 Teoria dos Conjuntos

Na Matemática, um conjunto é uma coleção de itens – tecnicamente chamados de *elementos* do conjunto. A importância da teoria dos conjuntos está no fato de que se pode trabalhar com diferentes ramos de modelagem matemática sob apenas um guarda-chuva. Isto facilita muita a conexão entre diferentes áreas, como, por exemplo, computação e música.

A teoria dos conjuntos é um excelente meio de se trabalhar com problemas que envolvem grandes quantidades de informações que precisam ser examinadas estruturalmente de diferentes modos. Um uso típico em computação envolve grandes bancos de dados. Nesse caso, a teoria dos conjuntos é útil pois auxilia a estruturar as várias categorias de informação pela análise de algumas informações [MIR 2001]. Por exemplo, um único arquivo de banco de dados pode conter o conjunto de instrumentos musicais utilizados em uma determinada peça, outro arquivo pode conter o conjunto de peças musicais e por aí em diante.

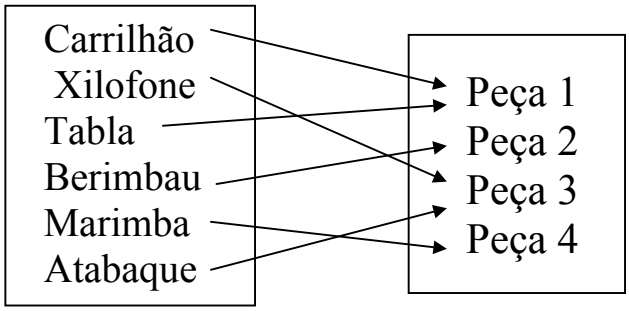


Figura 83 - Teoria dos conjuntos aplicada a estruturação de dados relacionados à música.

Uma condição importante da teoria dos conjuntos é a de que não haja repetição de membros dentro de um conjunto.

Quanto à notação matemática, conjuntos são denotados por letras maiúsculas e os elementos de um conjunto por letras minúsculas. O símbolo $\in$ significa "pertence à" ou "é um elemento de", e o símbolo $\notin$ "não pertence à" ou "não é um elemento de". Por exemplo, $x \in C$ lê-se "o elemento x pertence ao conjunto C", enquanto que $x \notin C$ é lido "x não pertence ao conjunto C" [MIR 2001].

6.2.2.1 Operações com Conjuntos

Existem quatro operações básicas entre conjuntos: **união**, **interseção**, **diferença** e **complemento**.

Escreve-se $X \cup Y$, a união de dois conjuntos X e Y. E esta união resulta em no conjunto de todos os elementos de ambos os conjuntos X e Y. Se X={1,2,3} e Y={4,5,6}, então $A \cup X = \{1,2,3,4,5,6\}$.

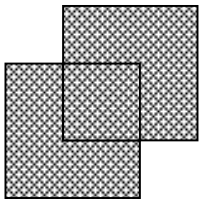


Figura 84 – União de dois conjuntos

Escreve-se $X \cap Y$, a interseção de dois conjuntos X e Y. Esta interseção resulta num conjunto que contém todos elementos que X e Y possuem em comum. Se X={1,2,3} e Y={2,3,4}, então $X \cap Y = \{2,3\}$. Caso os conjuntos envolvidos na operação não possuam nenhum elemento em comum, $X \cap Y = \varnothing$.

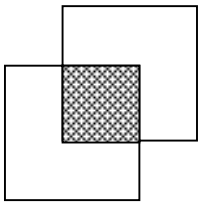


Figura 85 – Interseção de dois conjuntos

A diferença entre dois conjuntos pode ser representada tanto como X – Y ou como X / Y, ela é composta por todos elementos de X que não estão em Y. Por exemplo, se X={1,2,3} e Y={2,3,4}, então $X - Y = \{1\}$.

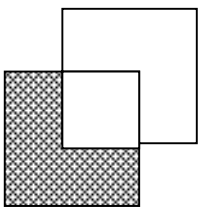


Figura 86 – Diferença entre dois conjuntos

O complemento de um conjunto X é denotado como X` e consiste em todos os elementos do respectivo conjunto universo os quais não estão contidos em X. Ou seja, X` = U – X. Por exemplo, se X={1,2,3,4,5} e U={3,4,5,6,7}, então X`={6,7}.

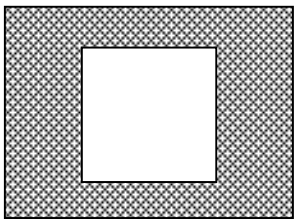


Figura 87 – Complemento de um conjunto

6.2.2.2 Álgebra de Conjuntos

Para compreender como as quatro operações básicas com conjuntos podem ser aplicadas para mais de um ou dois operandos, é necessário fazer uma revisão de conceitos básicos da matemática.

Adição e multiplicação são operações *comutativas*, ou seja, a ordem dos fatores não altera o produto. As operações de união e de interseção são semelhantemente comutativas:

$X \cap Y = Y \cap X$
$X \cup Y = Y \cup X$

A multiplicação também é *distributiva* sob a adição, o que significa que 2 x (3 + 5) = (2 x 3) + (2 x 5). Na teoria de conjuntos, uniões e interseções são distributivas entre si. Por exemplo:

$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$

Há, também, a lei de *associatividade* de adição e multiplicação que afirma que (2 + 3) + 5 é equivalente a 2 + (3 + 5). Isto se aplica também à união e à interseção:

$X \cup (Y \cup Z) = (X \cup Y) \cup Z$
$X \cap (Y \cap Z) = (X \cap Y) \cap Z$

Além das propriedades comutativa, distributiva e associativa existem outras seis a serem consideradas, estas estão resumidas na tabela abaixo:

Complemento	$X \cup X` = U$	$X \cap X` = U$
Idempotência	$X \cup X = X$	$X \cap X = X$
Identidade	$X \cup \varnothing = X$	$X \cap U = X$
Lei do Zero	$X \cup U = U$	$X \cap \varnothing = \varnothing$
Lei de De Morgan	$(X \cup Y)` = X` \cap Y`$	$(X \cap Y)` = X` \cap Y`$

Involução $(X`)` = X$

6.2.3 Lógica

Nesta seção, serão introduzidos os conceitos de *Álgebra Booleana* e de *Dedução Lógica*.

Na Lógica, expressões são frases declarativas que são *verdadeiras* ou *falsas*, nunca podendo ser ambas ao mesmo tempo. Semelhantemente à Teoria dos Conjuntos onde um elemento pertence ou não a um conjunto.

Quando se trabalha com Álgebra Booleana usa-se apenas dois valores: 1 (verdadeiro) ou 0 (falso). Estes valores são empregados em expressões muito impráticas de serem escritas literalmente, por isso, foram adotadas convenções para abrevia-las. Os símbolos mais comumente usados são:

$\forall$	"para todo(s)"
$\exists$	"existe"
$\neg$	indica negação
$\wedge$	"e"
$\vee$	"ou"

Portanto, ao invés de escrever "todos os valores de y para que 2y seja um número maior que zero", escrevemos " ? y | 2y > 0 ".

Os últimos três símbolos da tabela acima são chamados de *operadores lógicos* e são muito semelhantes às operações algébricas da Teoria dos Conjuntos. Um exemplo disto é: um elemento pertence à $X \cup Y$ se está em $X \vee Y$, e por aí em diante. Aqui expressões compostas resultantes de operações com operadores lógicos só podem ter valor 1 ou 0, como se fossem expressões simples. Porém, a verdade ou a falsidade de uma expressão composta depende de cada uma das expressões simples deste composto.

As regras para se trabalhar com expressões Booleanas compostas estão resumidas em *tabelas de verdade*. Estas tabelas para as três operações lógicas básicas "ou", "e" e "não" são:

X	y	$x \vee y$
1	1	1
1	0	1
0	1	1
0	0	0

x	y	$x \wedge y$
1	1	1
1	0	0
0	1	0
0	0	0

x	$\neg$ x
1	0
0	1

As propriedades algébricas definidas para a teoria dos conjuntos também se aplicam na álgebra Booleana. Neste caso o símbolo $\wedge$ representa $\cap$, o símbolo $\vee$ representa $\cup$, o símbolo de verdadeiro representa o conjunto universo e uma expressão sempre falsa representa um conjunto vazio.

Comutação	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
Associação	$x \vee (y \vee z) = (a \vee y) \vee z$	$x \wedge (y \wedge z) = (a \wedge y) \wedge z$
Distribuição	$x \wedge (y \vee z) = (x \vee y) \wedge z$	$x \vee (y \wedge z) = (x \vee y) \wedge z$
Complemento	$x \vee \neg x = 0$	$x \wedge \neg x = 0$
Idempotência	$x \vee x = x$	$x \wedge x = x$
Identidade	$x \vee 1 = x$	$x \wedge 1 = x$
Lei do Zero	$a \vee 0 = 0$	$a \wedge 0 = 0$
Lei de De Morgan	$\neg (x \vee y) = \neg x \wedge \neg y$	$\neg (x \wedge y) = \neg x \vee \neg y$

Pode-se usar o exemplo de um banco de dados para demonstrar a aplicação da álgebra Booleana no dia-a-dia. Imagine uma vídeo locadora com as seguintes categorias de filmes:

A = ação	B = aventura	C = policial
D = comédia	E = romance	F = ficção científica

Estas categorias devem interagir no sentido de que um único cliente possa gostar de mais de uma delas. Aqui, A(x) representa "x gosta de filmes de ação", o mesmo para as outras categorias. Então, A(Pedro) = 1 significa que Pedro gosta de filmes de ação e C(Paula) = 0 significa que Paula não gosta de filmes policiais. Supomos que esta locadora queira listar os clientes que gostam das categorias A e B mas não gostam de E e F. Como bancos de dados normalmente suportam buscas no formato de expressões booleanas podemos expressar esta situação como $A(x) \wedge B(x) \wedge \neg E(x) \wedge \neg F(x)$. Isto é uma tática muito comum na área de banco de dados.

6.2.4 Combinação dos Conceitos vistos

Podemos combinar a utilização de Lógica e Matrizes. Por exemplo, é possível que matrizes trabalhem com elementos diferentes de números como, por exemplo, expressões booleanas [MIR 2001]. Ao trabalharmos com estas devemos nos ater aos princípios de lógica booleana discutidos anteriormente, sendo que os operadores $\wedge$ e $\vee$ agem como multiplicadores e somadores, respectivamente. Por exemplo:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \wedge \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} (1 \wedge 1) \vee (1 \wedge 0) & (1 \wedge 0) \vee (1 \wedge 1) \\ (1 \wedge 1) \vee (0 \wedge 0) & (1 \wedge 0) \vee (0 \wedge 1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

6.3 MÓDULO DE INTRODUÇÃO À COMPUTAÇÃO E MÚSICA

No Módulo de Introdução à Computação e Música são apresentadas as principais categorias de sofware musical, características e utilização.

O avanço tecnológico é tão veloz que dificilmente um profissional consegue acompanhar todas as novidades lançadas no mercado, o que dirá assimilá-las e empregá-las no dia a dia. É sobre este aspecto que este texto enfoca a necessidade da atualização dos músicos no campo da tecnologia musical e, particularmente, na área da computação e música.

A inclusão de disciplinas em cursos de graduação em música estão sendo sugeridas pela nova Lei de Diretrizes e Bases para a Área de Música, em termos de "implantação da tecnologia" nos cursos de Composição, Execução e Educação Musical, visando contemplar o desenvolvimento do graduando nas "dimensões sociais, científicas, tecnológicas, artísticas e culturais" (CEEARTES, 1997).

No campo da informática existe uma forte interação com outras áreas do conhecimento gerando trabalhos interdisciplinares com bastante freqüência. Cada uma dessas áreas que usufruem de métodos algorítmicos, passíveis de processamento, necessitam de uma certa interação do informata com o problema em questão, a fim de entendê-lo melhor para propor soluções dentro de um universo mais específico de outra área de atuação. A interação com a área musical é inevitável e cada vez mais intensa, principalmente pelo advento dos sistemas multimídia e programas musicais.

6.3.1 O Despreparo do Músico Face à Tecnologia Digital

Uma parte dos compositores fazem uso das novidades disponíveis no milionário mercado de instrumentos musicais e correlatos. Com o desenvolvimento das telecomunicações e da tecnologia digital aliada à informática, existe um crescente número de músicos interessados em lidar com computadores e software musical [HUN 97]. Em geral, os músicos possuem pouco conhecimento na área de informática, sendo que outra grande fatia não tem o preparo técnico para entender manuais e livros específicos sobre música computacional. Em contrapartida, existe uma preocupação entre os fabricantes de instrumentos e software musicais em aprimorar a interface com o usuário, padronizando alguns sistemas, como o General MIDI (GM), e tornar a utilização de outros mais interativa.

A figura 88 mostra os recursos disponíveis e qual a tecnologia atual que o músico deve dominar para poder realizar sua interpretação ou composição.

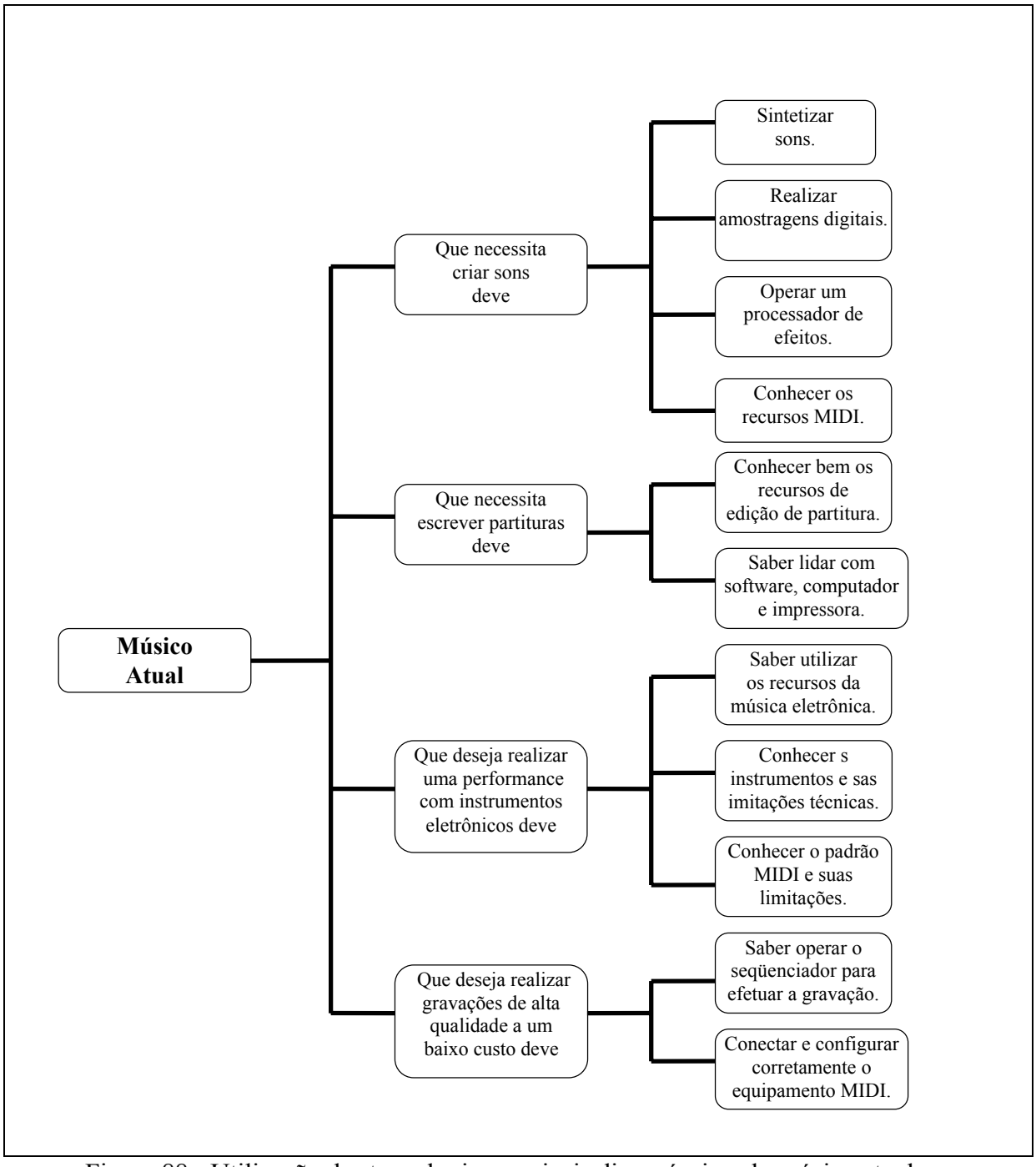


Figura 88 - Utilização das tecnologias musicais disponíveis pelo músico atual..

Recursos de síntese, amostragem digital e processamento de efeitos servem para o compositor criar seus próprios sons a serem utilizados como material musical em suas obras. Recursos de edição de partitura são muito úteis para o músico pois fornecem todas as ferramentas necessárias para a edição dos mais diversos símbolos musicais. Os instrumentistas podem fazer uso da tecnologia digital na execução de peças musicais e até mesmo verdadeiras sinfonias. Os instrumentos eletrônicos, muitas vezes computadores dedicados, fornecem um desafio para o instrumentista que deseja explorar o máximo de suas possibilidades para obter o melhor som. Muitas vezes o músico brasileiro adquire instrumentos digitais importados com manuais extremamente técnicos, geralmente na língua inglesa e pouco didáticos dificultando um melhor entendimento na operação do instrumento. Os recursos de gravação evoluíram muito sendo que o músico

pode gravar músicas em seu próprio computador e reproduzí-las automaticamente num estúdio profissional, economizando, dessa forma, o tempo de gravação.

Este conjunto de recursos tecnológicos que auxiliam o músico atual podem sugerir vários questionamentos como:

Até onde o músico deve saber sobre informática para começar a usufruir das vantagens obtidas pelo uso do computador?

Deverá o músico estudar apenas a informática relacionada à sua área de atuação de acordo com sua especialidade ou algo mais abrangente que englobe outras atividades dentro do campo musical? (Deal & Taylor, 1997).

Os conteúdos de Computação e Música deverão ser ministrados por professores com formação em música ou informática?

Tais questões nos levam a refletir sobre a interdisciplinaridade que a informática causa ao interagir com outras áreas do conhecimento e de como profissionais com formação em computação desenvolvem, com muita freqüência, seus trabalhos aplicados às demais áreas do conhecimento humano.

A não utilização de uma filosofia de ensino voltada para os avanços da tecnologia musical nos conservatórios brasileiros, aliada ao despreparo na área tecnológica e de informática dos profissionais formados, está originando uma utilização inadequada dos recursos tecnológicos e uma desinformação por parte de alguns músicos atuais. Para o ensino de tecnologias digitais no campo da música, a solução não está só em criar uma disciplina isolada com poucos recursos tecnológicos, mas sim mudar o enfoque para um ensino direcionado também ao mercado de trabalho de produções musicais que envolvam o emprego de recursos tecnológicos.

6.3.2 Software Musical

6.3.2.1 Editores de Som e Bibliotecas Sonoras

Os Gerenciadores de Bibliotecas Sonoras tem a função de controlar uma base de dados sonora que deve estar interligada a um sistema MIDI de comunicação entre instrumentos musicais. Geralmente arquiva sons que foram programados e possibilita a recuperação de uma forma otimizada. Pode guardar uma seqüência de sons de instrumentos diferentes e selecioná-los de acordo com a vontade do usuário. Os programas para a edição de sons são, em sua grande maioria, específicos para um determinado equipamento musical. Servem basicamente para facilitar a sintetização sonora. Os programas apresentam telas com gráficos representando as ondas sonoras que estão sendo alteradas e demais padrões musicais que regem a constituição do som.

6.3.2.2 Softtware Seqüênciador

A seguir apresentaremos o funcionamento e a operação básica do software seqüenciador CUBASE.

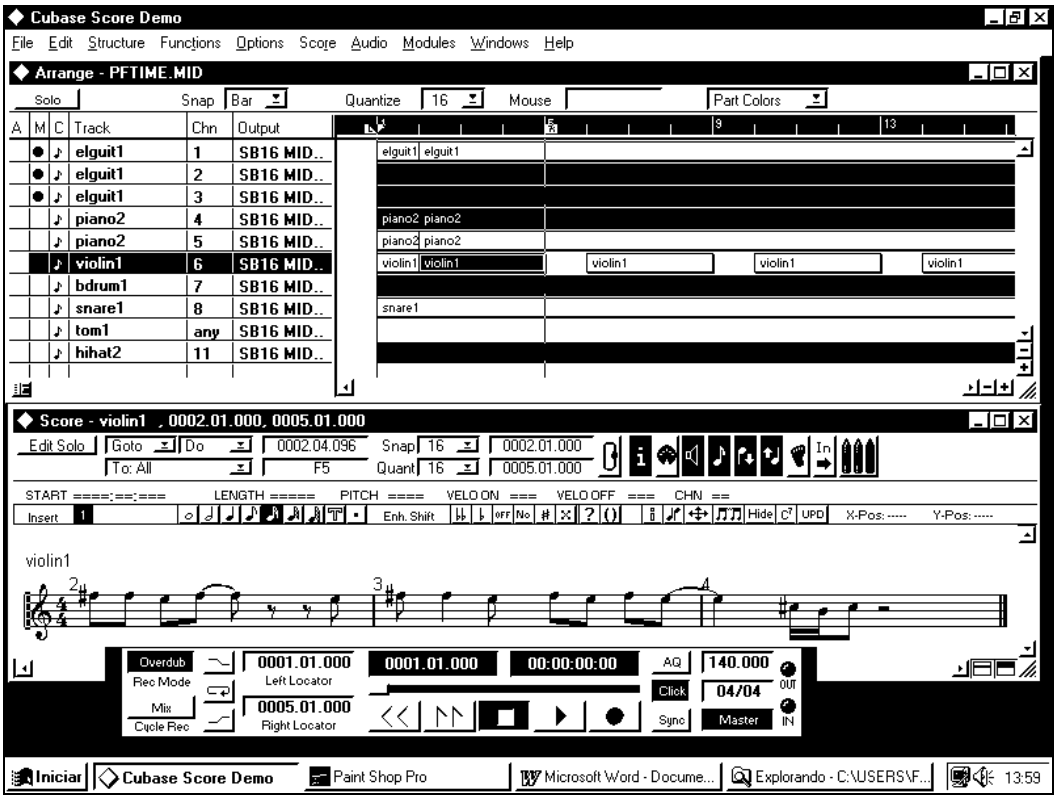


Figura 89 - Software Seqüênciador

Gravação em tempo real

É o tipo mais usado. Nele, o músico dispara a gravação e, ouvindo um metrônomo gerado pelo seqüenciador, toca no instrumento controlador (que está conectado à entrada MIDI In da interface) a música a ser gravada.

Ajusta-se adequadamente o valor do andamentoe define-se o tipo de metrônomo a ser usado, que pode ser interno (tocando pelo auto-falante do computador) ou MIDI (disparando uma nota de um instrumento conectado à MIDI Out da interface, normalmente um timbre de percussão). Pode-se ainda definir uma contagem inicial (count-in) do metrônomo, antes de começar a gravação, o que é útil para se tomar o andamento. Outro ajuste importante é o da métrica do compasso (time signature), o qual, se estiver errado, fará com que as notas gravadas sejam representadas nos tempos errados em relação ao que tocaram. É aconselhável indicar ao seqüenciador também a tonalidade da música, para que sua representação fique correta, principalmente se a seqüência for posteriormente salva em arquivo Standard MIDI File para ser lido por outro software (um editor de partituras, por exemplo) [HEY 96].

Para iniciar a gravação, o primeiro passo é escolher uma trilha (preferencialmente vazia, sem nada gravado). Em alguns seqüenciadores, a gravação apaga totalmente o conteúdo anterior da trilha; em outros, agravação atual é mixada (merge) com o conteúdo existente. Estando tudo pronto, pode-se disparar a gravação (REC) e, no andamento do metrônomo, executar a parte da música que se deseja.

É importante atentar para o fato de que, em princípio, o seqüenciador gravará tudo que o instrumento transmitir. Assim, execução de pedais, movimentação de controle de volume, aftertouch, etc, tudo será armazenado como parte da seqüência. A maioria dos seqüenciadores possui uma opção de filtragem de eventos (record filter), que permite determinar quais os eventos MIDI que serão realmente gravados.

Overdub

É denominado overdub o processo de adicionar novas partes a um material musical. No seqüenciador isso é possível, sendo recomendávelmente usar uma outra trilha (vazia) para colocar a outra parte. Digamos que você tenha gravado a parte do piano, por exemplo, e agora quer adicionar a linha do baixo. Para isso, é preciso saber qual o canal de MIDI das notas gravadas (em pricípio, é o canal de transmição usado durante a gravação, mas pode ser alterado por edição ou por recanalização em tempo real, nos parâmetros da própria trilha).

Durante o processo de overdub MIDI, para que se possa ouvir tanto o material já gravado como a execução atual, é necessário que haja mais de um instrumento conectado à saída MIDI Out da interface, ou então que o instrumento seja multitimbral (o que é mais comum hoje em dia).

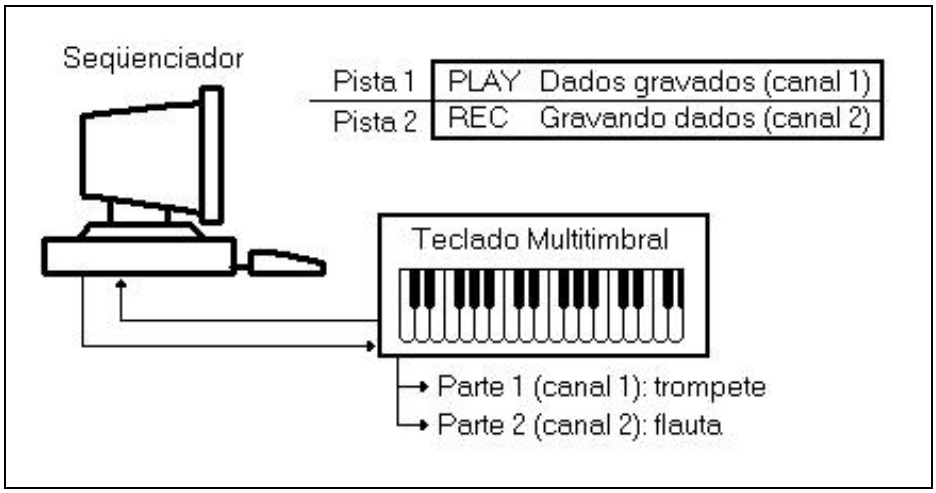


Figura 90 - Overdub

No processo de overdub, pode-se gravar um novo material ao mesmo tempo que se ouve o material já gravado.

A gravação da nova parte segue o mesmo procedimento já descrito para a gravação em tempo real, com a diferença de que, caso haja intrumento para isso, além do metrônomo, será ouvido o material já gravado.

Outra facilidade que há no seqüenciador é a possibilidade de se "remendar" trechos que foram gravados de forma incorreta, por meio de um recurso normalmente chamado de punch-in/out. Para usá-lo, indicam-se os pontos de início e de fim do trecho a ser remendado. A gravação começa, então, de qualquer ponto antes do trecho em questão. O seqüenciador não grava qualquer evento antes que atinja o ponto de início do remendo (punch-in), e grava (substituindo o material antigo daquele trecho) até atingir o ponto de fim do remendo (punch-out), devidamente "remendado".

Gravando passo a passo

Para os usuários menos habilidosos musicalmente, ou quando as passagens instrumentais são muito difíceis de se executar numa gravação, pode-se recorrer à facilidade da gravação passo a passo. Nesse processo, ao invés de executar as notas em um instrumento, via interface MIDI, o músico escreve nota por nota, usando o mouse ou o teclado do computador.

Para isso, pode-se escolher qualquer uma das vistas que for mais conveniente ou agradável para o músico (piano-roll, partituta, lista de eventos) e inserir as notas desejadas, indicando sempre todos os seus parâmetros (posição, altura, intensidade e duração). Esse processo é trabalhoso, mas possibilita enorme precisão dos dados e dispensa o uso de interface e instrumento MIDI (mas aí também não é possível ouvir o material gravado).

É comum usar-se os dois processos em combinação. Grava-se em tempo real tudo o que for mais fácil; em seguida entra-se com algumas notas passo a passo, ao mesmo tempo que se corrige a notas que tenham sido tocadas de modo incorreto. A entrada de alguns eventos, como controle de volume (Control Change 7) e mudança de timbre (Program Change) também são freqüentemente feitos passo a passo.

Gravação de padrões / gravação em loop

Um processo de gravação de notas muito usado em baterias eletrônicas é a gravação em loop, isto é, determinado trecho fica tocando indefinidamente, e as notas que são executadas no instrumento vão sendo gravadas na trilha, misturando-se (merge) com aquelas que já estão gravadas. A gravação em loop permite que se adicione notas em um trecho, sucessivamente, sem parar a gravação.

A cada repetição do loop, o músico pode ouvir as novas notas que foram gravadas, já adicionadas (misturadas) àquelas que já estavam gravadas. Caso as notas da última passagem não tenham ficado boas, em geral, é possível apagá-las sem destruir as anteriores.

Esse recurso é muito útil, principalmente para a criação de compassos e padrões rítmicos de bateria, onde é comum gravar-se adicionando peças individualmente.

Gravação multicanal

Na gravação multicanal (multichannel ou multitrack recording) o seqüenciador pode gravar simultaneamente notas de diversos canais MIDI, colocando-as em trilhas diferentes, de acordo com o número do canal. Ela é muito útil quando são usados dois ou mais instrumentos ao mesmo tempo (uma "jam session", por exemplo) transmitindo em canais diferentes, quando então a interface deverá possuir uma entrada MIDI In para cada instrumento. Outra aplicação é quando se quer gravar em tempo real uma seqüencia de outro seqüenciador que contém eventos em diversos canais, ou uma execuçccão produzida em um teclado arranjador (que transmite cada parte do arranjo em um canal diferente).

Edição

Os recursos de edição são as maiores ferramentas do seqüenciador. Com eles, é possível corrigir, mudar, recriar, cortar, colar e muito mais. Um item importante no que se refere à edição é a possibilidade de se reverter o processo, através da função chamada de desfazer (undo). Essa função torna o processo não destrutivo, possibilitando "voltar" quando a edição não ficar boa.

Cortar, Copiar e Colar

Esses proccedimentos presentes no seqüenciador já são bastante conhecidos por aqueles que usam softwares editores de textos e editores gráficos.

Filtragem de eventos

Na maioria dos seqüenciadores, o usuário pode optar se quer que as operações de edição manipulem todos os eventos existentes em um determinado trecho marcado ou apenas alguns daqueles eventos. Isso é possível quando existe o recurso de filtragem de eventos, uma opção que possibilita ao usuário discriminar quais dos eventos do trecho marcado serão afetados pelo comando de edição.

Quantização

O recurso de quantização é útil quando as notas executadas estão fora dos tempos corretos (atravessadas). No processo de quantização, marca-se o trecho onde deverá ser efetuada a edição e define-se a figura da nota musical para a qual devem ser aproximadas as notas gravadas.

Ao quantizar as notas, o seqüenciador as move para a posição mais próxima da figura de nota escolhida como referência, de forma que todas as notas ficarão perfeitamente posicionadas. No piano-roll as notas são representadas por tarjas, cujos comprimentos indicam suas durações; e suas posições verticais definem sua altura (em relação ao teclado). As linhas verticais indicam os tempos (posições) das semínimas. No quadro superior está a melodia como foi gravada, sendo possível observar as imperfeições de tempo ocorridas na execução, em relação às posições exatas de semínimas (ou frações suas). No quadro inferior é mostrada a melodia já quantizada, tomando-se como figura de referência a colcheia. Observe também que não só as posições das notas foram acertadas, mas também as suas durações, de forma que o fim de cada nota está agora acabando exatamente no fim da duração de uma colcheia.

A escolha da figura de quantização correta é muito importante. Imagine que no exemplo anterior tivesse sido escolhida a figura da semínima, ao invés da colcheia. Nesse caso, haveria um erro de quantização, pois as notas seriam posicionadas em tempos de semínimas (no gráfico, as linhas verticais), jamais em tempos de colcheias e, poR isso, diversas notas acabariam posicionadas no lugar errado (a terceira nota, por exemplo, seria puxada para a linha vertical da esquerda, alterando totalmente sua posição). Assim, a figura de quantização deve ser compatível com a cadência usada na melodia, de forma que se uma melodia está em colcheias, a figura de quantização deve ser colcheia. Se houver uma passagem em semicolcheias no meio, já não será possível usar a figura de colcheia para quantizar todo o trecho (ou se quantiza por partes: trechos em colcheia e trechos em semicolcheia) ou então tenta-se quantizar pela menor figura. O inverso também é perigoso: se for usada como referência de quantização uma figura de tempo muito menor do que a cadência do trecho, pode acontecer das notas serem também posicionadas erradamente, quebrando os tempos.

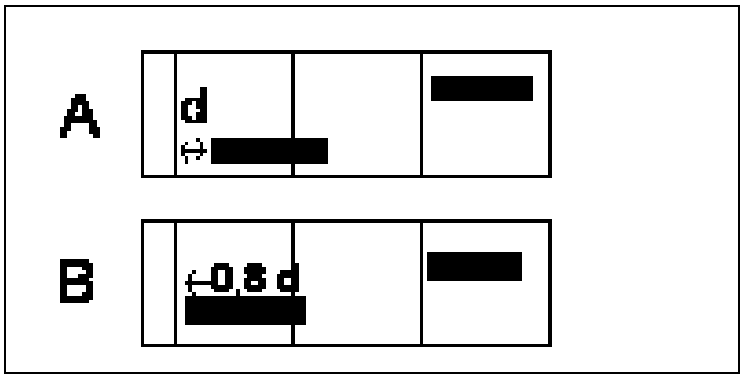


Figura 91 - Quantização

A quantização percentual, ao invés de reposicionar as notas no tempo exatamente correto, aproxima-se percentualmente daqueles tempos. No exemplo acima, foi definido um percentual de acerto de 80%, de forma que as notas em B foram aproximadas dos tempos corretos apenas 80% da distância total.

A quantização é aplicada com o objetivo de melhorar uma execução, tornando-a mais precisa. Entretanto, às vezes o material quantizado acaba soando pior do que o que era originalmente e, nesses casos, é melhor reverter (desfazer) a operação, e tentar acertar as notas de alguma outra forma. Em trechos mais complicados, onde há notas com diversas cadências, pode ser mais adequado acertá-las manualmente, uma por uma.

Músicas totalmente quantizadas tendem a soar muito artificiais, muito mecânicas. Uma forma de se evitar essa mecanização da música é usar uma quantização percentual, um recurso disponível na maioria dos seqüenciadores profissionais. Esse tipo de quantização, ao invés de acertar totalmente as notas, apenas aproxima-as da figura de referência, num percentual de aproximação definido pelo usuário.

Usando-se um percentual de acerto de 80%, por exemplo, todas as notas são aproximadas apenas 80% do total, caso fosse usada a quantização total. Dessa forma, melhora-se a execução, mas sem perder a flutuação de tempo, o chamado swing. Há ainda seqüenciadores que permitem gerar-se um swing a partir de notas quantizadas, num processo quase que inverso ao da quantização, onde as notas são posicionadas ligeiramente fora dos tempos certos.

Substituição de eventos

É possível produzir no próprio seqüenciador o que faz o compressor, dispositivo de áudio que efetua uma compressão dinâmica da amplitude dos sons de uma música, alterando suas intensidades e aproximando todos de um nível médio (abaixando os mais intensos, e aumentando os mais fracos). No seqüenciador, isso é possível alterando-se as intensidades das notas (key velocity).

Transposição

A transposição é a função que altera a altura das notas. Na realidade, como a altura de uma nota é representada por seu número, o processo de transposição resume-se a somar ou subtrair determinado valor (em semitons) dos números das notas.

6.3.2.3 Software Editor de Partituras

Editores de Partitura

Os Editores de Partituras, tem por função produzir partituras em notação musical tradicional possibilitando sua edição e posterior impressão em papel. São de grande valia para editoras e livros de música. Também possuem sua importância junto aos compositores que utilizam a notação musical tradicional para escrever suas peças. Os músicos e compositores atuais deveriam saber utilizar os recursos de edição de partitura pois esta técnica deveria ser tão comum para eles quanto o editor de textos é para os escritores atuais. No entanto, muitos profissionais na área da música ainda possuem uma visão conservadora dificultando uma maior difusão desta tecnologia. Neste tópico, apresentaremos o funcionamento e a operação do software editor de partituras Encore.

Num software editor de partituras é possível escrever uma partitura a partir de três modos:

- **manualmente**, quando se escreve com o mouse cada símbolo da partitura ou toca-se o teclado virtual;
- **via MIDI**, quando se grava a música a partir de um instrumento MIDI
- **via arquivo MIDI**, quando se carrega para o Editor de partituras, através de um arquivo Standard MIDI File as notas e demais execuções de uma música criada em outro software (em geral, um seqüenciador).

Sendo um software para edição gráfica, o *Editor de partituras* oferece recursos poderosos de manipulação de objetos gráficos e por isso como também usa procedimentos típicos de softwares desta natureza.

Uma página escrita com música no *Editor de partituras* é composta de sistemas, que agrupam uma ou mais pautas (pentagramas), e é possível definir tamanhos e posições de acordo com a necessidade.

Acima da janela principal do *Editor de partituras* há uma barra de botões, onde se pode selecionar as ferramentas que se quer usar, bem como controlar a gravação e a execução MIDI da música escrita na pauta e virar páginas da música.

As ferramentas usadas nas operações de escrita e edição são a *seta* (para apontar e selecionar, o lápis (para inserir e desenhar símbolos) e a *borracha* (para apagar símbolos).

O botão, *Voice* permite escolher quais as vozes da pauta que serão habilitadas para edição, execução e gravação. Alguns editores de partituras permitem ter-se oito vozes em cada pauta, de forma que se pode trabalhar independentemente com apenas uma delas, sem alterar as demais de um mesmo pentagrama, Esse recurso é muito interessante em arranjos para naipes ou corais, onde pode-se ter uma pauta com todas as vozes (ideal para o regente) ou então extrair cada uma delas separadamente, para distribuir aos instrumentistas ou cantores.

Iremos tomar como exemplo um dos editores de partitura mais utilizados na autalidade, o Encore.

No caso do Encore, a primeira opção da lista aberta pelo botão Voice indica *All Voices,* isto é, todas as vozes da pauta estão selecionadas (para edição, escrita e

gravação). Pode-se também selecionar apenas uma das oito, de forma que a escrita, edição e/ou gravação só atue sobre aquela determinada voz que está selecionada.

O botão Record (representado por um círculo vermelho) permite que se grave uma música via MIDI e o botão Play (representado por um triângulo verde) que se toque a música que está escrita na partitura, desde que se tenha todos os recursos MIDI (interface, instrumento) disponíveis ao computador.

O botão, *Stop* (representado por um quadrado preto) permite que se pare a execução da música. O botão *Thru* ativa ou não a retransmissão pela saída MIDI Out dos dados recebidos pela entrada *MIDI* ln, conforme configurado no quadro *MIDI Setup* , e no próprio botão é mostrada a letra da porta e o número do canal que está sendo usado para o *Thru.*

O botão *M* indica o número do compasso que se está apontando, e permite saltar (*Jump)* imediatamente para determinado compasso da música. No quadro, indica-se em *Measure*, o número do compasso para onde se quer ir, pode-se ir direto para o último compasso clicando-se no botão com o símbolo de *duplo travessão.*

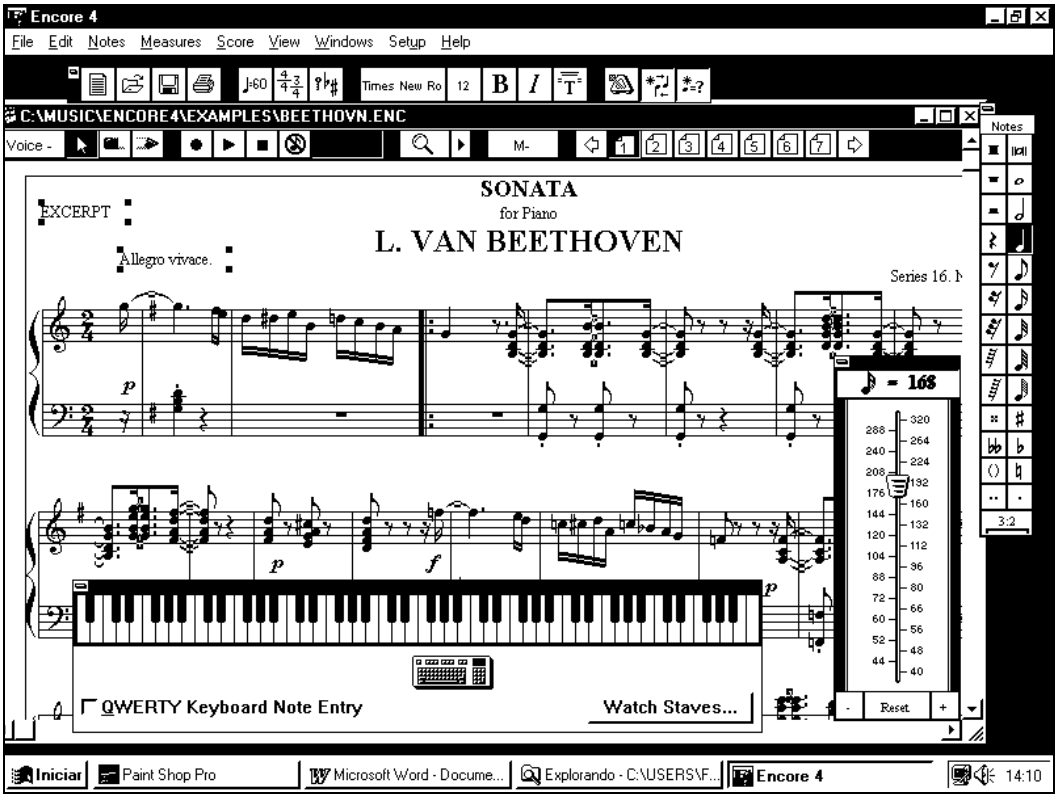


Figura 92 – Software editor de partitura

O botão com o ícone de uma placa de trânsito "proibido", quando clicado pelo mouse, transmite uma mensagem de “All Notes Off”, que desliga quaisquer "notas presas" que estejam ocorrendo no instrumento MIDI. Ao se clicar com o mouse sobre o botão de Zoom (ícone de lente de aumento), a seta do cursor se transforma em uma lente de aumento, e apontadas para a o objeto ou a região central do local que se deseja ampliar, e em seguida clicando o mouse, faz com que o trecho ou objeto apontado seja mostrado ampliado. Para restaurar a imagem à suas dimensões normais, clica-se novamente no botão de **Zoom,** Pode-se também ativar e desativar a função Zoom pressionando-se a tecia **Z.**

Pode-se definir a magnitude da ampliação, criando no pequeno botão de seta, localizado à direita do botão de **Zoom,** Fazendo-se isso, abre uma janela de opções, dividida em duas partes. Na parte superior estão as opções de restauração de dimensões *(Restore Level)* isto é, as dimensões que a imagem retomará ao se desativar o zoom:

Fít Paqe: ajusta as dimensões da imagem de forma que a partitura caiba inteira na tela;

Fit Width: ajusta as dimensões da imagem de forma que a largura total da partitura caiba na largura total da tela;

Actual Size: mostra a partitura no tamanho que será impressa, levando em consideração as eventuais reduções ou ampliações definidas em *Page Settíngs;*

Constant Scale: mostra todas as pautas com as mesmas dimensões, desconsiderando as alterações efetuadas em *Score Settings,* mas preservando o tamanho (Size) individual de cada pauta definido em *Staff Sheet.*

Na parte inferior do quadro de opções, estão os níveis de aproximação *(Zoom Level):*

x2: aumenta duas vezes o tamanho da imagem;

x3: aumenta três vezes o tamanho da imagem;

x4: aumenta quatro vezes o tamanho da imagem;

Custom: aumenta o tamanho da imagem na proporção definida pelo usuário, permitindo valores de 25% a 400%.

Para ampliar a imagem (*zoom in)* em um nível, pressione juntas as teclas Shift e Z, para reduzir a imagem *(zoom out)* em um nível, pressione **Shift, Ctrl** e **Z**.

Keyboard Window / Tempo Window

A janela *Keyboard Window* do Encore permite que se "toque" MIDI virtualmente, através do teclado do computador ou pelo mouse, usando como referência um teclado gráfico. Já a janela *Tempo Window* permite ao usuário alterar o andamento da música em tempo-real. Esses recursos estão detalhados mais adiante, na seção específica sobre o menu *Window.*

Arquivos do Editor de partituras e MIDI Files

Embora seja um software para música, o *editor de partituras é* um software de edição gráfica, isto é, sua finalidade é a apresentação visual da notação musical. Para mostrar os diversos símbolos musicais, o *editor de* utiliza parâmetros próprios. Isso quer dizer que, ao se salvar a partitura de uma música feita no editor de partituras, diversos dados têm que ser adicionados ao arquivo, além das informações padronizadas de notas e demais eventos "puramente MIDI". Para isso, o Editor de partituras usa um formato de arquivo próprio. No caso do Encore, por exemplo, é utilizada a extensão ENC que preserva todas as informações necessárias à notação musical.

O Editor de partituras pode carregar e salvar arquivos em formato universal Standard MIDI Files (extensão MID) mas, a não ser que se queira transferir trabalhos do *editor de partituras* para outros softwares (seqüenciadores, por exemplo), deve-se sempre salvar os arquivos no formato próprio do editor de partituras, pois os arquivos Standard MIDI Files não preservam qualquer símbolo gráfico, mas apenas dados dos eventos MIDI e algumas informações adicionais (voltadas para o MIDI). Dessa forma,

ao se salvar uma música do *Editor de partituras* em formato MID, este não terá mais qualquer dos detalhes puramente gráficos.

Por outro lado, como os demais softwares não podem usar o formato próprio do *Editor de partituras* (ENC), a única maneira de se transferir uma música entre eles é usando o padrão Standard MIDI File. Assim, quando se quer escrever pelo *Editor de partituras* a partitura de uma música composta em um seqüenciador, basta salvar essa música pelo seqüenciador em formato MID e carregá-la no *Editor de partituras.* A partir daí, a música pode ser editada graficamente, e então deverá ser usado o formato ENC, para preservar os símbolos gráficos que foram adicionados aos eventos MIDI do arquivo original.

Editor de partituras também pode carregar e salvar arquivos no formato MTS, usado pelos seqüenciadores *Master Tracks Pro e* Trax, da Passport, e arquivos com extensão MUS, usado pelo software *Music Time,* também da Passport.

Edição básica com o mouse

Pode-se alterar notas e símbolos diretamente com o mouse. Para mudar a altura de uma nota, basta selecionar o botão de seta, e com o mouse clicar e arrastar verticalmente a cabeça da nota até a nova altura desejada, o então soltá-la.

Pode-se também mover qualquer outro objeto gráfico de posição, clicando sobre ele e arrastando-o até a posição desejada. No caso de notas de um acorde, ao se clicar em uma delas e arrastá-la para a direita ou esquerda, todas as notas do acorde serão arrastadas juntas.

Para apagar um símbolo (nota, pausa, texto, etc), basta selecionar o botão da borracha e clicar com o mouse no meio do símbolo que se quer apagar.

Para se escrever uma nota basta selecionar o botão do lápis, abrir a paleta de notas (menu *Windows)* e selecionar a figura da nota que se quer inserir (no Encore, se quiser inserir uma pausa daquela figura, basta teclar R). A seta do mouse se transformará na figura da nota selecionada (ou da pausa, se tiver sido teclado R). Então, posicione a nota (ou pausa) onde deseja, clique o mouse e a figura será inserida na pauta. Se o compasso já estiver completo (não há mais tempos vazios) ou o tempo vazio for menor do que a figura que se quer incluir, não será possível inserir a nota (ou pausa), sendo necessário apagar alguma nota ou pausa, para fazer espaço. Para poder incluir notas ou pausas além da capacidade de tempos do compasso, é preciso desativar a opção *Auto Space* (menu *Setup).*

Entrando com as notas via MIDI

Pode-se entrar com as notas no Editor de partituras usando um teclado MIDI, onde se pode executar a música em tempo-real (no metrônomo) ou então teclá-las uma a uma.

Ao iniciar uma nova música no Encore, é possível trabalhar com quatro formatos diferentes de pautas:

Piano: é o tipo de pauta que combina dois pentagramas; esse é o único tipo de pauta que permite a uma haste unir notas nos dois pentagramas;

Piano-Vocal: usa a pauta de piano (descrita acima) com mais um pentagrama da clave de fá, acima da pauta de piano;

Single Staves: usa um único pentagrama; podem ser criados até 64 pautas dessas;

Template: essa opção só está disponível se houver um arquivo com nome TEMPLATE no diretório do *Editor de partituras;*

O arquivo "template" serve como ponto de partida para o *Editor de partituras.* Se seus trabalhos normalmente são diferentes das condições iniciais padronizadas do Editor de partituras (compasso 414, tonalidade de dó, pauta de piano, etc) crie uma partitura vazia com as características que você usa e salve-a no diretório do *Editor de partituras* com o nome de TEMPLATE.ENC. Assim, toda vez que você abrir o *Editor de partituras,* ele usará este arquivo como básico.

Staves per system (pautas por sistema) indica quantos pentagramas haverá em cada sistema. Este parâmetro só atua quando se usa *Sinqle Staves.*

Systems per paqe (sistemas por página): indica quantos grupos (sistemas) de pentagramas haverá em cada página.

Measures per system (compassos por sistema): indica quantos compassos haverá em cada sistema, isto é, quantos compassos haverá em cada "linha" da partitura.

Um sistema é um grupo de pautas de instrumentos diferentes. Esses instrumentos estão listados nas linhas do *Staff Sheet (menu Windows). O* número de sistemas por página ("systems per page") multiplicado pelo número de pautas por sistema ("staves per system") não pode exceder 64. *O* Editor de partituras permite que se abra mais de um arquivo, de forma que se pode ter diversas janelas de trabalho abertas, simultaneamente. Isso é útil quando se quer copiar trechos de uma música para outra, ou mesmo consultar uma música enquanto se edita outra. Entretanto, quanto mais janelas estiverem abertas, menos memória estará disponível no computador, o que pode reduzir seu desempenho. Você pode abrir diversos arquivos em uma mesma ação de *Open:* na janela de arquivos selecione todos os que quer abrir, clique OK e cada um ocupará uma janela de trabalho do *Editor de partituras.*

Marcando Trechos e Manipulando Símbolos

Como todo software gráfico, Editor de partituras também possui técnicas e procedimentos específicos para a manipulação dos símbolos e objetos que compõem a notação musical da partitura. Basicamente, pode-se atuar sobre notas, compassos, pautas e símbolos gráficos.

Seleção

Na maioria das vezes, para se efetuar uma edição é necessário primeiramente selecionar o trecho que se quer editar. *Para efetuar a seleção de qualquer trecho, nota, compasso ou pauta, é necessário que esteja selecionada a seta (A), e não a borracha (E) ou o lápis (P).*

Selecionando notas e pausas individualmente

Para selecionar uma única nota ou pausa, há duas maneiras diferentes. A primeira é pressionando a tecla Shift e em seguida clicando sobre a cabeça da nota (ou pausa), o que fará surgir um pequeno quadrado escuro sobre ela. A segunda maneira é clicando e arrastando a seta de forma que o retângulo preto cubra a cabeça da nota (ou pausa).

A primeira maneira é indicada quando se quer selecionar diversas notas ou pausas não adjacentes, individualmente. Nesse caso, a tecla Shift deve ser mentida

pressionada enquanto se clica sobre as cabeças das notas (ou pausas) que devem ser selecionadas. Para desfazer a(s) seleção(ões), basta clicar com o mouse uma vez em qualquer ponto fora da área selecionada. Isso vale para qualquer tipo de seleção (notas, pausas, compassos ou pautas).

Selecionando compassos

Em alguns casos, pode-se querer efetuar algum tipo de edição em todo um compasso (às vezes, em um número de compasso inteiros).

Para selecionar um compasso inteiro, basta clicar duas vezes (rapidamente) sobre algum ponto do mesmo (sem clicar sobre uma nota). o compasso inteiro será envolvido por um retângulo preto (vídeo inverso).

Para selecionar diversos compassos, contíguos ou não, basta manter a tecla Shift pressionada e selecionar cada compasso desejado (clicando duas vezes, rapidamente, sobre algum ponto seu). Os compassos selecionados serão envolvidos por retângulos pretos (vídeo inverso).

Sempre que se quiser selecionar mais do que uma nota individual ou mais do que um compasso, use o recurso da tecla *Shift***,** mantendo-a pressionada enquanto seleciona. Mesmo depois de já ter feito as seleções, se quiser selecionar mais itens, basta pressionar *Shift* e selecionar.

Selecionando pautas

Para selecionar uma única linha de uma pauta, aponta-se para um ponto fora do pentagrama, à esquerda da pauta, e então dá-se um único clique com mouse. Com isso, toda aquela linha daquela pauta ficará selecionada (mostrada em vídeo inverso).

Para selecionar todas as linhas de uma pauta, a partir de determinada linha (inclusive), basta apontar à esquerda (fora do pentagrama) da primeira linha que se quer selecionar e clicar duas vezes, rapidamente. Com isso, todas as linhas a partir da linha onde se clicou (incluindo ela) serão selecionadas. Para selecionar todas as linhas de uma pauta na música, basta clicar duas vezes, à esquerda da primeira linha daquela pauta.

Movendo notas e pausas

Pode-se mover notas de uma posição para outra, havendo duas situações diferentes: movimento vertical (mudança de altura da nota) e movimento horizontal (mudança de posição).

No primeiro caso, basta clicar e, sem soltar o botão do mouse, arrastar a nota para sua nova altura (posição vertical no pentagrama). Se houver um instrumento MIDI conectado ao computador, poder-se-á ouvir a nota tocando na posição (altura) original e depois na nova posição. Quando quiser mover a nota ou pause verticalmente, mova-a logo na vertical, pois se a ela for movida na horizontal, o Editor de partituras assumirá que se deseja movê-la assim, e não será possível movê-la na vertical, a não ser que se solte o mouse e reinicie o movimento.

Para mover a nota ou pausa na horizontal, o procedimento é semelhante, só que o movimento deve ser feito somente na horizontal. Se a nota estiver em um acorde, todas as notas do acorde serão movidas juntes (só soará a nota que foi clicada e arrastada),

Movendo um grupo de notas na horizontal

Há dois recursos bastante úteis para se mover horizontalmente grupos de notas dentro de um compasso.

Para se mover todas as notas de um compasso, "espremendo-as" para a direita ou para a esquerda, basta manter pressionada a tecla *Ctrl* e, clicar com o botão da direita do mouse e arrastar qualquer nota do compasso. O grupo de notas será "espremido" na direção do movimento, empurrando as notas que estejam entre a nota que está sendo arrastada e a extremidade do compasso do lado do movimento.

Pode-se efetuar o mesmo movimento para todas as notas daquele compasso de todas as pautas do sistema. Para isso, basta manter pressionada as teclas *Ctrl* e *Shift*, clicar com o botão da direita do mouse e arrastar qualquer nota de qualquer das pautas. Em cada compasso, os grupos de notas serão igualmente "espremidos" na direção do movimento, empurrando as notas que estejam entre a nota que está sendo arrastada e a extremidade do compasso do lado do movimento.

Editando traços de união de notas

Pode-se alterar a inclinação dos traços que unem as hastes das notas. Para isso, basta apontar para a extremidade do traço que se quer mudar a inclinação, clicar e arrastá-lo para a nova inclinação.

Para mover o traço para cima ou para baixo, sem alterar a inclinação, basta apontar para o meio do traço, clicar e arrastá-lo para a nova posição.

Editando duração de notas

Há um procedimento rápido para se alterar a duração de notas: selecione a(s) nota(s) que deseja alterar e então tecle o novo valor da duração *(l = semibreve, 2 = mínima, 3 = semínima, etc).* Se quiser transformar a nota em pausa, tecle R (teclando R novamente faz a pausa se transformar em nota).

Para que a alteração de duração na pauta não altere a duração real da execução MIDI, faça o procedimento acima mantendo pressionada a tecla *Shift.*

Editando arcos de ligaduras

Para alterar o posicionamento e a curvatura dos arcos de ligadura, mas para isso é conveniente primeiramente habilitar a visualização dos *pontos de controle,* através dos quais se pode mover e remodelar os arcos.

Para visualizar os pontos de controle, é necessário chamar a função *Show / Hide,* do menu View, Esta função também pode ser chamada teclando-se **Ctrl** e **H**.

Ao chamar a função *Show / Hide,* aparece um quadro, cujas opções serão descritas detalhadamente na Seção que aborda o menu *View*. Para o momento, o que nos interessa é a opção *Control Points,* que deve ser marcada com X para que os pontos de controle dos arcos de ligadura sejam mostrados.

Uma vez marcada esta opção (e fechando o quadro Show/ H*ide*), todos os arcos de ligadura (e alguns outros símbolos) passarão a exibir pequeninos quadrados pretos, que são os *pontos de controle.* Pode-se também visualizar os *pontos de controle* teclando *Ctrl e*).

Para alterar o formato e a curvatura de um arco, basta posicionar a seta do mouse sobre o ponto de controle desejado e arrastá-lo à nova posição, mudando o formato do arco.

Movendo barras de compasso

Embora Editor de partituras posicione automaticamente as barras de divisão de compassos, pode acontecer que seja necessário alterar a posição de uma ou mais barras, de forma a tomar a notação mais estética. Para fazer isso, basta apontar com o mouse no extremo superior da barra, clicar e, sem soltar o botão, arrastar a barra horizontalmente até a posição desejada.

As notas e pausas serão ajustadas nos compassos separados pela barra movida, comprimindo-se e espaçando-se, automaticamente.

Movendo pautas

Pode-se também mover pautas inteiras (horizontal e/ou verticalmente), bastando para isso apontar a extremidade (esquerda ou direita) da primeira linha do pentagrama da pauta, clicar sem soltar o botão do mouse e arrastar a pauta (enquanto se estiver arrastando a pauta, esta será mostrada na posição original e um retângulo cinza mostrará a posição atual em que se está arrastando a mesma).

Ao mover uma pauta de posição, as demais pautas (linhas) do sistema abaixo da que foi movida são igualmente movidas, de forma que a distancia entre elas permanece imutável. Se for movida uma pauta que não seja a superior, todas as pautas acima dela permanecerão em suas posições originais, e as abaixo dela se moverão conjuntamente (mantendo as distâncias originais).

Se for mantida pressionada a tecla Ctrl enquanto se move uma pauta, o novo posicionamento será aplicado à todas as páginas da partitura.

Movendo textos

Também os textos escritos na partitura podem ser movidos de suas posições originais. Para mover qualquer texto, basta apontar sobre ele, clicar e arrastá-lo até a posição desejada. Caso não se consiga apontar para o texto, pode-se visualizar a caixa de texto, habilitando a visualização dos *pontos de controle (função Show / Hide, menu View)*. Toda caixa de texto é demarcada por quatro *pontos de controle.*

Para mover as sílabas da letra da música, é usado procedimento semelhante, mas só é possível movê-las na horizontal (veja na próxima seção, *Paletas de Símbolos,* como editar a letra da música).

Movendo símbolos

Além de notas, pausas, barras de compassos, pautas e textos, outros símbolos também podem ter suas posições alteradas usando-se o mouse. Para mover ou ajustar a inclinação do colchete de quiálteras, usa-se o mesmo procedimento já descrito para ajustar o traço de união de notas.

Pode-se mover na horizontal as cifras da harmonia, com o mouse, da mesma forma que foi descrita anteriormente para mover as sílabas da letra da música (veja na próxima seção, Paletas de Símbolos, como editar as cifras de harmonia).

Selecionando as figuras da partitura

O Editor de partituras oferece uma enorme variedade de símbolos para serem usados na notação musical. Estes símbolos estão organizados em grupos, e cada grupo pode ser acessado por meio de uma *paleta* própria. Uma paleta, na realidade, nada mais é do que uma janela onde estão dispostos botões com os símbolos correspondentes. Para se escolher um determinado símbolo, basta abrir a paleta em que ele se encontra, e clicar sobre o botão correspondente.

Para abrir as janelas de paleta deve-se chamar a função Palette *(menu Windows) e* selecionar a desejada dentre as dez disponíveis. Uma vez aberta a janela de paletas (não importa com qual paleta), pode-se "virar" todas elas, clicando-se com o mouse uma vez sobre o nome da paleta.

A janela de paletas pode ser posicionada em qualquer lugar da tela, bastando arrastá-la clicando no quadrado azul, no alto à direita. Para fecha-la, basta clicar duas vezes no quadrado de "gaveta", no alto à esquerda.

Notes

A paleta Notes contém os símbolos de notas musicais e respectivas pausas, além dos sinais de alteração ou acidentes (sustenido, bemol, dobrado sustenido, dobrado bemol, bequadro) e dos pontos de aumento e colchete de quiáltera.

Para selecionar uma figura da paleta, basta clicar com o mouse sobre o botão respectivo (que mudará de cor, indicando que está ativo). Com isso, a seta do mouse se transformará na figura correspondente ao botão clicado e pode-se então inserir aquela figura onde se deseja, na pauta.

Para selecionar uma nota pontuada, é necessário clicar no botão da figura da nota e em seguida no botão com o ponto de aumento desejado (um ponto ou dois pontos).

Para selecionar quiálteras (tripiets), é necessário clicar no botão da figura da nota e em seguida no botão que mostra o colchete de quiáltera. O padrão de quiáltera é 3:2 (três notas no lugar de duas), mas é possível definir outras proporções, bastando para isso clicar duas vezes, rapidamente, no botão com o colchete de quiáltera, o que fará abrir um pequeno quadro *(Choose Tuplets)* onde devem ser definidas "quantas notas no lugar de quantas" (5:4, 7:6, etc). Para grupar ou desagrupar quiálteras num mesmo travessão, use a função *Notes / Beam Notes.*

Para se inserir os sinais de alteração, é necessário selecioná-los na paleta (a seta do mouse se transformará no sinal selecionado) e em seguida clicar o sinal sobre a nota desejada, que ele será grafada com aquele sinal.

É importante atentar para o fato de que uma nota ou pausa só poderá ser inserida em determinado compasso se houver "espaço rítmico" para ela., isto é, se todos os tempos já estão completos (ou o tempo da figura que se quer inserir não cabe no tempo que falta), a nota ou pausa não poderá ser inserida naquele compasso a não ser que seja eliminada (com a borracha) alguma(s) outra(s) nota(s) ou pausa(s)@ para dar espaço.

Clefs

A paleta Clef contém os símbolos de claves. Para selecionar uma clave da paleta, basta clicar com o mouse sobre o botão respectivo (que mudará de cor, indicando que está ativo). Com isso, a seta do mouse se transformará na clave correspondente ao

botão clicado e pode-se então inserir aquela *clave* no local onde se desejar alterar a clave de referência, em qualquer Ponto de qualquer compasso, o que fará mudar todas as claves naquela pauta daquele ponto em diante, a não ser que já haja alguma outra clave inserida na pauta.

Graphics

A paleta *Graphics* permite a escrita da letra da música, textos diversos, cifras de acorde, posições dos dedos no violão e também símbolos gráficos para criar marcações úteis com finalidade didática. Para selecionar um tipo de símbolo gráfico ou de texto da paleta, basta clicar com o mouse sobre o botão respectivo (que mudará de cor, indicando que está ativo). Com isso, a seta do mouse se transformará no símbolo correspondente ao botão clicado e pode-se então inserir aquele símbolo ou texto na partitura.

Letra da música

O botão *L* do Encore permite que se escreva a letra da música *(lyrics)* na partitura. Ao pressioná-lo, será mostrada uma seta → à direita e abaixo da primeira pauta. Com o mouse, pode-se arrastar esta seta verticalmente, para definir a posição vertical (linha) onde a letra será escrita, que pode estar acima ou abaixo da pauta.

Uma vez escolhida essa posição, basta clicar com a seta do mouse sobre a nota onde se quer iniciar a letra, e o *Editor de partituras* irá posicionar o cursor (linha vertical) abaixo da mesma, mostrando que a letra pode ser digitada naquele ponto. Digita-se então a letra, sendo que ao se digitar um hífen (-) ou espaço, o cursor avança para a próxima nota da pauta (acima). Dessa forma, pode-se separar perfeitamente as sílabas das palavras.

Para corrigir qualquer sílaba ou palavra já digitada, basta apontar com o mouse para a mesma, que o cursor será posicionado nela. permitindo alterá-la.

Para cada pauta, pode haver até oito linhas de letra, cada linha usando uma voz diferente, independentemente da(s) voz(es) usada(s) pelas notas na pauta. É importante atentar para o fato de que, para cada pauta, a linha de letra que usa a voz l estará sempre acima da linha de letra que usa a voz 2, que estará acima da linha de letra da

Se uma nota estiver ligada à anterior, ela será automaticamente saltada, isto é, a próxima sílaba ou palavra não poderá ser escrita sob ela, a não ser que se pressione *Shift* e barra de espaço, o que fará o cursor parar embaixo da nota ligada.

Para escrever mais linhas de letra, sob a mesma pauta, selecione uma outra voz (clique no botão *Voice*). Para poder usar o espaço ou o hífen entre palavras sem avançar para a próxima nota, pressione Ctrl enquanto tecia a barra de espaço ou a tecla de hífen.

Para alterar a fonte (formato) de letra usada na letra da música, clique no botão do lápis (*P*) e chame a função *Font* (no menu *Text*, que só aparece quando o botão do lápis está selecionado). No quadro das fontes, escolha o tipo, tamanho e demais características para as letras. Se quiser mudar apenas uma sílaba, seleciones (aponte o cursor para ela, e arraste o mouse da direita para a esquerda), e então chame a função *Font* (que s6 atuará sobre a sílaba selecionada).

Texto

O botão *T* permite que se escreva qualquer texto na partitura. Daí então, basta apontar e clicar com o mouse no local onde se quer iniciar o texto, o que fará surgir uma pequena *caixa de texto,* com um cursor vertical piscante.

Para aumentar a *caixa de texto,* basta clicar no pequeno quadrado existente em sua extremidade direita inferior e arrastá-lo até a posição onde se quer que vá o texto. Feito isso, escreve-se o texto normalmente.

Para alterar a fonte (formato) de letra usada no texto, é preciso selecionar o texto que se quer editar (aponte o cursor para após a última letra do texto e arraste o mouse da direita para a esquerda). Chame então a função *Font* (no menu Text, que só aparece quando o botão do lápis está selecionado) e no quadro das fontes, escolha o tipo, tamanho e demais características para as letras.

Algumas fontes são maiores do que outras, de forma que poderão não caber na caixa de texto, que deverá ser aumentada.

Para redimensionar a *caixa de texto,* basta habilitar a visualização dos *pontos de controle (função Show / Hide,* no menu *View)* e mover qualquer desses pontos, conforme desejar.

Cifras

Os botões *C* e *G* permitem que se escrevam cifras de acordes na partitura. O botão *C* permite indicar a cifra somente, enquanto o botão *G* permite não só indicar a cifra, mas também um desenho com a posição do acorde no braço do violão. Ao clicar no botão *C* ou *G*, é mostrada uma seta → que indica a posição vertical onde será escrita a cifra (mesmo procedimento para inserção da letra da música). Basta então apontar e clicar com o mouse no local onde se quer inserir a cifra, quando então será mostrado um quadro *(Choose Chord)* onde é possível definir o tipo de acorde.

Em *Root*, deve-se definir a tônica do acorde. Se for marcada com *X* a opção *Bass* pode-se definir um baixo alternativo para o acorde (se não for marcada, o baixo será a tônica). Em *Type*, seleciona-se o tipo de acorde (se o acorde desejado não estiver nas opções, deve-se marcar com X a opção *Custom,* e escrever sua cifra no campo correspondente.

Figuras geométricas

Os cinco botões inferiores da paleta *Graphics* possibilitam desenhar figuras geométricas de retângulos, retas, retângulos com vértices arredondados e círculos ou elipses. No último botão de baixo, pode-se selecionar o tipo de linha a ser usado nessas figuras (tracejada ou contínua de diversas espessuras).

A utilidade dessas figuras é muito grande, principalmente para marcar trechos ou partes interessantes, para efeitos didáticos, por exemplo.

Tools

Esta paleta contém diversos símbolos de notação musical que representam condições especiais ou técnicas de execução da música. Algumas dessas indicações afetam a execução musical, como é o caso da indicação de andamento e as variações de dinâmica.

Arcos

Os dois primeiros botões de cima selecionam figuras de arcos *(slurs)*, normalmente usadas para indicar que as notas sob eles devem ser executadas ligeiramente ligadas. Para desenhar um arco, após clicar no botão correspondente e

selecionar o botão do lápis (*P*), deve-se definir os três pontos básicos do arco. Primeiro os dois da extremidade (indicados por pequenas cruzes).

Andamento

O segundo botão da primeira coluna permite indicar alterações de andamento *(tempo) no* decorrer da música (essa indicação pode atuar na execução MIDI). Para inserir uma indicação de andamento, é necessário selecionar o botão correspondente (quando então a seta do mouse se transforma em cruz), apontar e clicar na posição da partitura onde se quer inseri-la. Nesse momento, será apresentado uma janela *(Set Tempo Marking)* com diversos parâmetros que definirão a representação visual da alteração de andamento.

Selecionando-se a opção *Listesso,* significa que o valor da figura cujo botão estiver ativado será usado como nova referência para o andamento da música, a partir da posição em que for inserida a mudança de andamento, o que quer dizer que o andamento não deverá mudar, mesmo que haja uma mudança de (métrica de) compasso.

Se a opção estiver marcada com X, a execução MIDI obedecerá às variações de andamento definidas com esta função. O botão Font permite escolher o tipo de letra a ser usado na indicação de andamento.

Dinâmica

O botão com o símbolo de *crescendo* (<) permite indicar alterações de dinâmica na pauta. (essa indicação pode atuar também na execução MIDI). Para inserir uma indicação de dinâmica, é necessário selecionar o botão correspondente do mouse se transforma em cruz). Em seguida, para se indicar um *crescendo,* deve-se apontar e clicar na posição da partitura onde se quer iniciá-lo e arrastar o símbolo para a direita, abrindo-o para ter um formato de <. Para se indicar um *dimínuendo,* deve-se apontar e clicar na posição da partitura onde se quer terminar o símbolo e arrastá-io para a esquerda, abdndo-o de forma que tenha um formato de >.

Para se definir as características da variação de dinâmica na execução MIDI, deve-se abrir uma janela de parâmetros *(Set Dynamic Range),* o que é feito clicando-se duas vezes, rapidamente, no botão <.

Na opção *Hairpins Chanqe Velocities by* pode-se indicar a variação de intensidade (em *key velocity)* que ocorrerá na execução MIDI ao ser executado um trecho com um sinal de crescendo ou diminuendo. Essa variação só ocorrerá se a opção *Play iairpins* estiver marcada com um X. A indicação de dinâmica está vinculada à pauta em que ela está representada. Em algumas partituras de conjunto de cordas, é comum grafar estas variações em apenas uma das pautas, de forma que, num caso desses, se forem extraídas pautas individuais, deve-se antes copiar os símbolos de dinâmica para cada uma delas, caso contrário somente uma das pautas extraídas ficará com tais indicações.

Trinado

Há dois botões que permitem a inserção de símbolo de trinado (tr).

Para criar o símbolo do Trinado, basta clicar em um dos botões (a seta do mouse se transformará em uma pequena cruz) e então apontar para a posição em que se quer inserir o símbolo. Nesse ponto, deve-se clicar o mouse (surgirá o símbolo ir) e, sem

soltar o botão, arrastá-lo horizontalmente para a direita, criando a linha ondulada no comprimento que se quiser. Este símbolo não afeta a execução MIDI.

Arpejo

Há dois botões que permitem a inserção de símbolo de arpejo (linha ondulada, normalmente na vertical).

Para criar o símbolo do Arpejo, basta clicar em um dos botões (a seta do mouse se transformará em uma pequena cruz) e então apontar para a posição em que se quer inserir o símbolo. Nesse ponto, deve-se clicar o mouse (surgirá o símbolo ir) e, sem soltar o botão, artastá-lo na direção em que se quer gerar a linha ondulada, desenhandoa comprimento que se quiser (a linha ondulada pode ser desenhada em qualquer direção). Este símbolo não afeta a execução MIDI.

Parênteses

O botão de parênteses permite que se envolva qualquer objeto (nota, símbolo, texto) com eles.

Para criar os parênteses, basta clicar naquele botão, apontar para a extremidade superior esquerda da área que deseja envolver, clicar e sem soltar o botão do mouse, arrastá-lo até a extremidade inferior direita da área que deseja envolver, e então soltar o botão do mouse. Os parênteses serão representados em dimensões proporcionais à área envolvida.

Colchetes

O botão de colchete [permite que este símbolo seja inserido em qualquer lugar da pauta. Para isso, basta clicar naquele botão, apontar para a extremidade superior de onde se quer posicioná-lo, clicar e sem soltar o botão do mouse, arrastá-lo verticalmente até a extremidade inferior, e então soltar o botão do mouse.

Pedal

Há quatro botões para inserção de símbolos de pedal, que permitem que estes símbolos sejam inseridos em qualquer lugar da pauta, para indicar os momentos em que se deve pressionar e soltar o pedal (sustain ou abafador). Para inserir os símbolos de Ped e basta clicar no botão correspondente, apontar onde quer posicioná-lo e clicar.

Dessa forma, serão geradas mensagens MIDI de controle 64 com valor 127 (pedal de sustain pressionado) para a indicação Ped, e mensagens MIDI de controle 64 com valor O (pedal de sustain solto) para a indicação.

O que tem um símbolo semelhante a um pequeno *W*, serve para indicar que o pedal deve ser ficar pressionado, mas em sérios momentos deve ser solto parcialmente. Para desenhar este símbolo, basta clicar no botão correspondente, apontar para a extremidade esquerda de onde quer posicioná-lo, clicar e sem soltar o botão do mouse, arrastá-lo horizontalmente até o ponto desejado e então soltar o botão do mouse, citando uma linha do tipo. Para marcar as indicações onde o pedal é solto parcialmente, basta apontar sobre a linha desenhada e clicar, formando então um .

O outro botão, semelhante a um *″N″* ao contrário, serve para indicar pressionamentos rápidos do pedal. Para inserir este símbolo, basta apontar onde se quer

posicioná-lo, clicar e, sem soltar o botão do mouse, arrastá-lo para a direita e para baixo, de forma a dimensionar o símbolo na altura e largura desejadas, e então soltar o botão do mouse.

Oitava

Os quatro botões seguintes servem para inserir indicações de "uma oitava acima" e "uma oitava abaixo". Para inserir essas indicações, seleciona-se o botão desejado (8va, para, "oitava acima" e 8vb para "oitava abaixo"), aponta-se e clica-se onde inicia o trecho oitavado e o Editor de partituras então indicará oitavamento a partir deste ponto. Para delimitar o fim - - -, ou - - J (fecham do trecho oitavado, basta selecionar o botão de 8v8 e Bv respectivamente) e clicar no ponto onde encerra o trecho em questão.

MIDI

Este botão permite que sejam inseridos eventos MIDI do tipo *program change (mudança* de timbre) e contro/ler(controles de volume, pan, etc).

Para inserir um evento MIDI, basta clicar neste botão, apontar para o ponto da música onde se quer inseri-lo e clicar. Nesse instante, será mostrada a janela *MIDI Graphic Item,* onde é possível escolher o tipo de evento. Deve-se indicar o número (do *program change* ou do *controíler) e* o valor do *controller,* nos campos correspondentes. O nome do controle ou programa selecionado será mostrado acima (ex: Volume = controle 7). O botão Font permite escolher o tipo de letra que será usado para indicar o evento MIDI na partitura.

Para quem não sabe os números dos controles e dos timbres dos instrumentos, basta clicar no botão Change, que será mostrada uma lista dos mesmos, com seus respectivos números.

No caso dos controles (comandos de *contrai change),* o quadro *Choose Controiler* mostra os nomes dos que já têm definição na especificação MIDI. Basta clicar na planilha no nome do controle desejado, que seu número será automaticamente preenchido em *Number.* Depois é só fechar o quadro, e confirmar o evento na janela *MIDI Graphic Item.*

No caso dos timbres (comandos de *program change),* o quadro *Choose Instrument* mostra os nomes dos disponíveis no equipamento selecionado no quadro (em *Device).* Há diversos equipamentos na lista de Device, mas pode-se adicionar outros, caso seja necessário (abrindo uma nova lista por meio da opção Add *New* Device ou então carregando uma lista de um arquivo, por meio da opção Load Devíce File). Qualquer lista de tímbres de um equipamento (Device) também pode ser salva individualmente como um arquivo no disco.

7 NÍVEL DE PROGRAMAÇÃO PREPARATÓRIA

Neste módulo será abordada a Linguagem de Programação Max, para plataforma Macintosh, que se justifica pela facilidade do seu uso e por implementar os conceitos do protocolo MIDI e programação de computadores para música.

Desta forma, com a descrição das características do Max, seus principais objetos, exemplos práticos e exercícios ministrados em aula, ao término do curso o aluno terá plenas condições de aplicar os conhecimentos adquiridos no curso para desenvolver seus próprios programas apresentando soluções para problemas específicos quer na edição, execução ou composição musical assistidas por computador. Soluções essas que estão além daquelas tradicionalmente conhecidas como editores de partitura, timbre e seqüenciamento de sons.

Assim, o presente método surge para preencher essa lacuna a muito existente, despertanto no aluno de música o gosto pela programação de maneira agradável e interativa.

7.1 MÓDULO DE ELEMENTOS BÁSICOS DE PROGRAMAÇÃO DE COMPUTADORES

Os exemplos e exercícios práticos, que constituem o Módulo de Elementos Básicos de Programação de Computadores, foram implementados com base nas obras de [ZIC 88], [PUC 90] e [WIN 98].

O Max foi desenvolvido no *Institut de Recherche et de Coordination Acoustique / Musique* (IRCAM) em Paris, em meados de 1986. O autor principal do Max é Miller Puckette. Natural de Tennessee, Puckette estudou matemática e tem estado à frente nas pesquisas aplicadas à música computacional em tempo real. As origens do Max estão numa linguagem não-gráfica que foi desenvolvida para controlar o poderoso sintetizador 4x do IRCAM. Posteriormente, esta linguagem foi modificada e implementada como um ambiente gráfico para MIDI no Macintosh passando a chamar-se Max. A partir de 1989, David Zicarelli tem desenvolvido as novas versões do Max.

Max permite o controle de seu equipamento MIDI de maneira flexível, sendo possível criar aplicações para composição e improvisação musical, acompanhamento automático, envio de comandos para sintetizadores, modificação de *presets* ou qualquer coisa que seja possível criar via MIDI. Como o Max transforma todas as informações de controle em uma simples cadeia de números, você pode modificar praticamente qualquer parâmetro.

O Max adota os melhores aspectos das outras linguagens de programação e combina elas em um pacote gerado especificamente para aplicações de computação em tempo real. O Max será utilizado como ambiente de desenvolvimento de algoritmos para a composição pois possui ainda as seguintes características:

- uma interface gráfica intuitiva
- uma coleção de objetos gráficos para construir interfaces
- possibilidade de geração de aplicações “stand-alone”
- possibilidade de extensão dos objetos com a programação em C.

Com estas possibilidades é possível preocupar-se com a programação de algoritmos de composição em alto nível já que o MAX permite a utilização de bibliotecas de objetos especialmente projetadas para auxiliar a programação da composição musical.

7.1.1 Objetos

Os programas em Max são criados pela ligação de objetos graficamente na tela do computador. Objetos são algoritmos que realizam ações. A programação em Max é modular, sendo composta pela ligação destes vários objetos. Construir programas grandes e complexos consiste num processo de adicionar e conectar vários pequenos módulos.

Os objetos recebem mensagens do teclado do computador e mouse, de instrumentos MIDI ou de outros objetos Max. Uma mensagem é um conjunto de dados (números) enviados a um objeto para serem processados, ou instruções (palavras) enviadas a um objeto para descrever como ele funciona. Cada objeto é projetado para executar uma tarefa bem definida.

A conexão entre os objetos é desenhada graficamente através do mouse que conecta a saída de um objeto com a entrada de outro. As linhas que interligam os objetos permitem que as mensagens sejam transferidas entre eles.

Um programa em Max é chamado de *patch*, inspirado nos programas sonoros resultantes das ligações realizadas em sintetizadores analógicos. Um *patch* pode possuir um ou mais *sub-patches* (sub-programas). Logo, o termo *patch* refere-se a toda a coleção de pequenos objetos e sub-programas.

Os programas são criados na *patcher window*, onde os objetos padrão são escolhidos a partir de uma "palette" localizada no topo da tela. A "palette" possibilita a escolha dos objetos via mouse somente durante o modo de edição.

Um objeto pode ser encarado como um caixa preta onde os dados entram, são processados e saem, sem que o usuário tenha conhecimento de como as operações ocorreram internamente. . Isto simplifica muito a manipulação com altos níveis de programação, permitindo ao usuário ater-se mais à concepção musical. Em Max existe um grupo de objetos chamado interface objects, no qual cada objeto usa um ícone gráfico para representar suas funções. Eles servem para criar a interface do usuário e todas as operações realizadas por este em tempo real. Vários tipos de *sliders* aumentam e diminuem valores quando movidos para cima ou para baixo, *toggle switches* são usados para selecionar uma de duas entradas ou para iniciar ou finalizar processos, *bang button* envia uma mensagem de saída "*bang*", que significa "execute".

7.1.2 Transmissão de Mensagens

Barras pretas duplas acima e abaixo do nome do objeto possuem regiões onde linhas podem ser conectadas ao objeto. A conexão da linha ao objeto permite que este envie ou receba mensagens. O *outlet* ("tomada de saída") do objeto está sempre localizado embaixo do objeto, enquanto o *inlet* ("tomada de entrada") está sempre em cima do objeto. Alguns objetos podem ter vários *inlets* e *outlets,* dependendo de sua função. No exemplo da figura 93 a mensagem "Msg" é enviada pelo *outlet* do objeto de texto diretamente para o *inlet* do objeto *print*.

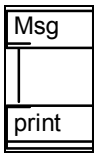


Figura 93 – Envio de Mensagens

Cada *outlet* e *inlet* nos objetos do Max possui uma função específica que pode ser verificada no menu ao selecionarmos a opção "*Assistance Option*".

7.1.3 Argumentos e Parâmetros

Cada objeto possui um número limitado de parâmetros que freqüentemente são variáveis que mudam durante uma execução. No caso do objeto *noteout*, este cria uma nota MIDI com os parâmetros de afinação, velocidade e canal MIDI. Alguns parâmetros utilizam valores *default*. Por exemplo, caso o parâmetro para o canal MIDI não seja especificado, o *default* é o canal MIDI 1.

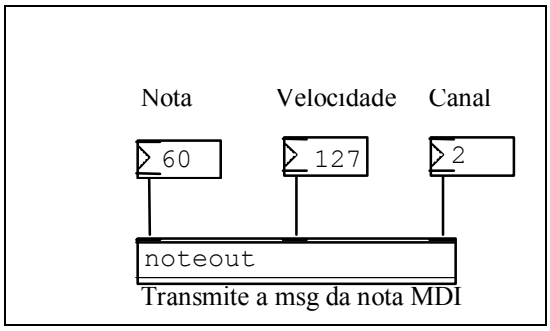


Figura 94 – Transmissão de mensagens MIDI

Alguns parâmetros podem ser escritos diretamente dentro da caixa de um objeto como argumentos. Qualquer dado novo recebido no inlet irá sobrepor os argumentos escritos. Para a maioria dos objetos, o uso de argumentos é opcional, enquanto alguns objetos necessitam de argumentos para funcionar. Argumentos podem ser sempre usados quando os parâmetros dos valores são constantes, significando que eles nunca irão mudar durante o curso do programa.

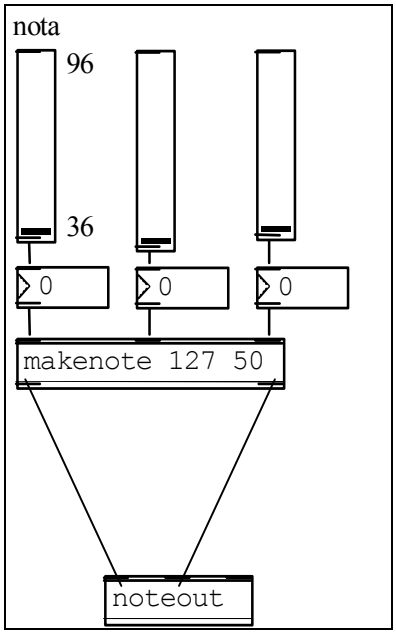


Figura 95 – Uso de parâmetros em Max

A figura 95 exemplifica uma utilização para o emprego de parâmetros no Max. Cada número que sai do *slider* é combinado com uma velocidade por *makenote* (127). A altura ("pitch") e a velocidade são então enviados a *noteout*. 50 milissegundos após o recebimento de cada nota, *makenote* envia o mesmo *pitch* novamente. A velocidade e a duração podem ser especificadas por mudanças no *slider*.

7.1.4 Tipos de Mensagens

Mensagem é qualquer informação transmitida de um objeto para outro. Uma mensagem pode ser enviada através da saída de um objeto, de um objeto de interface do usuário ou de uma caixa de mensagens "message box". Uma "message box" armazena e mostra qualquer tipo de mensagem podendo enviá-la através de um *bang* ou de um clique com o mouse.

Números – tipo inteiro ou tipo ponto flutuante.

Palavras – são também chamadas de símbolos no Max. Muitos objetos recebem símbolos que são mensagens de controle usadas para descrever a operação do objeto. Por exemplo, quando o objeto delay recebe a mensagem stop, ele imediatamente pára o processo de delay, impedindo qualquer saída adicional.

Listas – uma lista sempre inicia com um número e consiste em dois ou mais números ou símbolos separados por espaços. A ordem dos itens é importante e geralmente representa a ordem de parâmetros de inlets de um objeto que irá receber a lista. Por exemplo, se a lista [62 100 2] é enviada ao objeto noteout, este irá interpretar o primeiro item como sendo a nota, o segundo como sendo a velocidade e o terceiro como sendo o canal MIDI. Esta é a mesma ordem de parâmetros associada com os três inlets do objeto noteout.

Bangs – uma mensagem de bang pode ser enviada clicando-se no botão bang ou clicando-se na palavra bang no "message box". É uma mensagem especial que significa "execute".

Objetos de interface são otimizados por mensagens de controle de tempo real do teclado ou mouse do Macintosh. Por exemplo, o objeto number box mostra os números que ele envia e recebe. Isso pode ser controlado fisicamente clicando e arrastando o mouse, enviando para fora um fluxo contínuo de números. Um "message box", por comparação, é um objeto usado para mostrar uma única mensagem. Enquanto uma "message box" sempre mostra a mensagem atual, a maioria de outros objetos de interface do usuário são mais ativas, graficamente mostrando como os objetos são enviados.

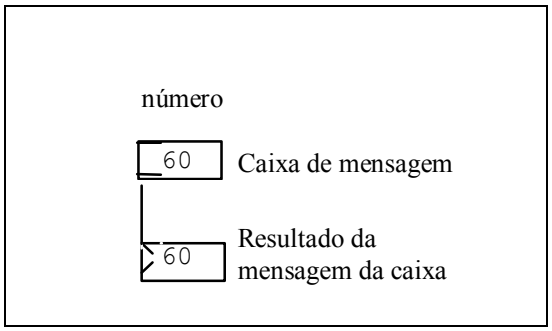


Figura 96 – Exemplo de envio de valores em Max

Existem inúmeras maneiras de enviar mensagens em Max. Se um número é uma constante, uma caixa de mensagem pode ser usada, desde que a caixa de mensagem retenha seu valor cada vez que o programa for usado. O objeto *int*, todavia, é designado para guardar um único número inteiro. Quando os valores precisam ser selecionados para mudar o comportamento em tempo real do programa, então a caixa de número – "number box" – é ideal. As caixas de número não salvam seus valores após sair do programa. Elas são inicializadas automaticamente com o valor *default* zero.

Em Max é possível limitar os valores a fim de filtrar dados inaceitáveis ou então descrever uma faixa desejada. Caixas de números usadas para enviar informações de nota e velocidade podem ser limitadas no mínimo de 0 e no máximo de 127, já que estes são os únicos valores possíveis para nota e velocidade. Isso pode ser feito utilizando-se a opção *Get Info* do menu do Max. É possível selecionar a faixa de um *slider*, assim como automaticamente multiplicar valores para colocá-los numa faixa desejada usando *Get Info*.

7.1.5 Ordem de execução

O Max é rápido o suficiente para dar a impressão de que muitas coisas estão acontecendo ao mesmo tempo. No entanto, um *patch* em Max executa somente um passo de cada vez. Max escalona a ordem dos eventos da direita para a esquerda, ou seja, objetos que aparecem à direita de outros objetos serão executados primeiro. Objetos em Max que tenham mais de um *inlet* também irão receber os dados na ordem da direita para a esquerda. Se um único *outlet* enviar dados para mais de um objeto, ele o fará na ordem da direita para a esquerda. Quando acontecer de *inlets* de dois ou mais objetos estarem perfeitamente alinhados na vertical, o objeto mais abaixo irá ser executado primeiro, seguindo a ordem de baixo para cima.

7.1.6 Paleta de Objetos do Max no Modo de Edição

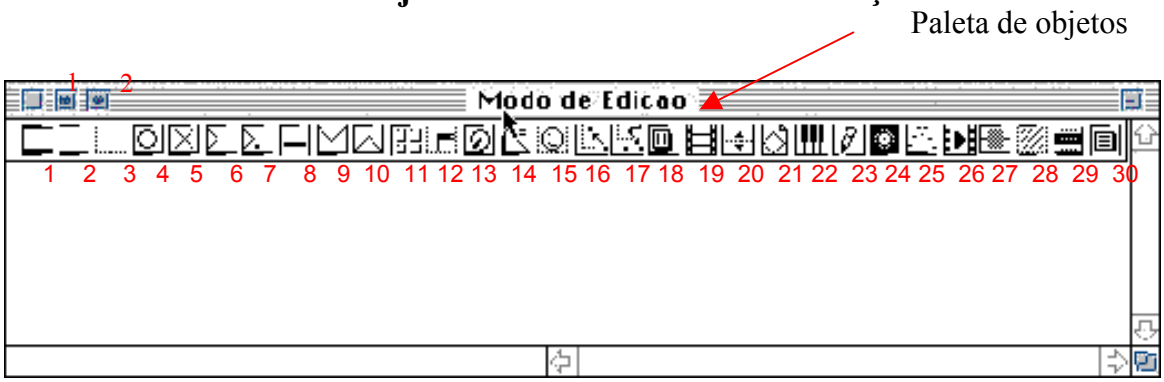


Figura 97 – Janela do Max em modo de edição

1. **Object Box** - abre uma lista de Comandos pré-definidos do Max
2. **Message Box** – envia uma mensagem para um objeto do Max
3. **Comment** – utilizado para comentários e títulos em um programa
4. **Button** – envia um disparo ou bang ao ser clicado
5. **Toggle** – alterna entre ligado e desligado (1 e 0)
6. **Number Box** – mostra e envia um número de saída
7. **Float Number Box** – mostra e envia números decimais de saída
8. **Slider -** envia números ao mover-se a barra verticalmente na tela
9. **Inlet** – recebe uma mensagem na entrada de um subprogramas
10. **Outlet** – envia uma mensagem de saída de um programa (janela)
11. **Preset** – salva e recupera ajustes dos objetos
12. **Patcher-in-Box** – cria um subprograma dentro de um programa
13. **Dial** - envia números ao girar-se o botão na tela
14. **Script-Driven Envelope** – Envelopa dados dentro do objeto Funnel
15. **Picture From PICT File** – mostra um arquivo PICT em uma caixa
16. **Graphic Switch** – recebe uma mensagem de entrada em uma das duas tomadas de entrada
17. **Graphic Gate** – passa a mensagem de entrada para uma das tomadas de saída
18. **Horizontal Slider -** envia números ao mover-se a barra horizontalmente na tela
19. **Quick Time Movie** – reproduz um filme QuickTime em uma janela
20. **Increment / Decriment** – incrementa e decrementa valores numéricos de uma caixa
21. **Icon Switch** – alterna ícones mostrados (0-7)
22. **Keyboard Slider** - envia notas ou valores correspondentes para um objeto
23. **LCD: Draw Into Patcher Window-** - desenha formas, linhas e texto em uma caixa
24. **LED** – mostra os estados ligado e desligado em cores
25. **MultiSlider** – pode funcionar como um arranjo de sliders
26. **Quick Time Play Controller** – controlador do andamento de filmes QuickTime
27. **Range Bar** – seleciona e mostra a faixa de dois valores
28. **Transparent Button** – usado por tras de um texto ou figura
29. **Pop-up Menu** – utilizado para criar-se a lista de um menu
30. **Vertical Slider** – mostra valores recebidos
31. **Barra de Status** – indica o nome ao clicar-se nos objetos da Paleta de Objetos
32. **Cadeado** - habilita/desabilita o modo de edição

7.1.7 AULA 1 – Transmitindo notas

O MEPSOM – Método de Ensino de Programação Sônica de Computadores para Músicos, apresenta um roteiro para o estudante observar exemplos e realizar exercícios propostos de programação. O aluno deverá ler com muito cuidado as explicações e apontamentos nos patchs para um perfeito entendimento da utilização dos objetos em conjunto.

A seguir apresentaremos os primeiros exemplos práticos em MAX que deverão ser utilizados num computador Macintosh Performa 6200 com 32 MB de memória RAM e 500 MB de memória em HD, ou superior. A versão inicial do Max utilizada para a elaboração do método foi a 3.0b6 de Agosto de 1994. Posteriormente foi realizado um upgrade para uma versão atual.

No exemplo da figura 98, quando o objeto ***noteout*** recebe um número na sua entrada da esquerda, usa este número como o valor da nota, combina a nota com uma velocidade e um número de canal, e transmite uma mensagem MIDI de nota. A nota, a velocidade e o canal também podem ser recebidos juntos na forma de uma lista na entrada da esquerda. Clique sobre os quadros para ligar as notas e depois sobre os quadros para desligar as notas. Após entender o programa, resolva o exercício.

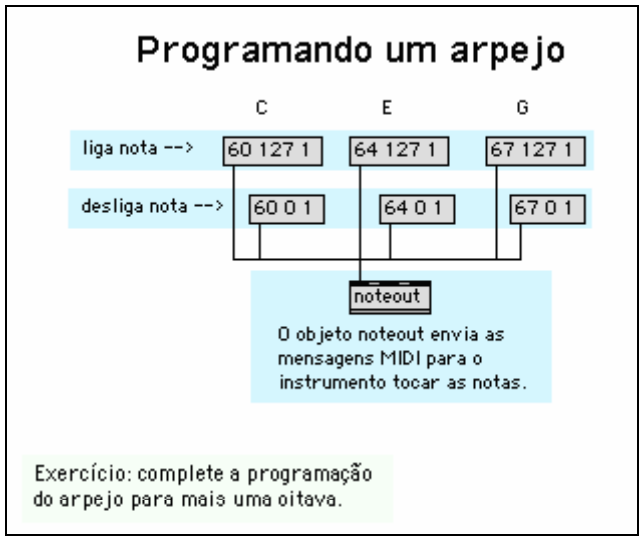


Figura 98 – Envio de mensagens de *NoteOn* e *NoteOff*

A figura 99 apresenta a resposta para o exercício. Perceba que os mesmos princípios utilizados no patch da figura 98 foram usados nesta resposta. Para completar o arpejo com mais uma oitava, foi adicionado o valor 12 ao 60, representando o deslocamento para uma oitava acima da nota original.

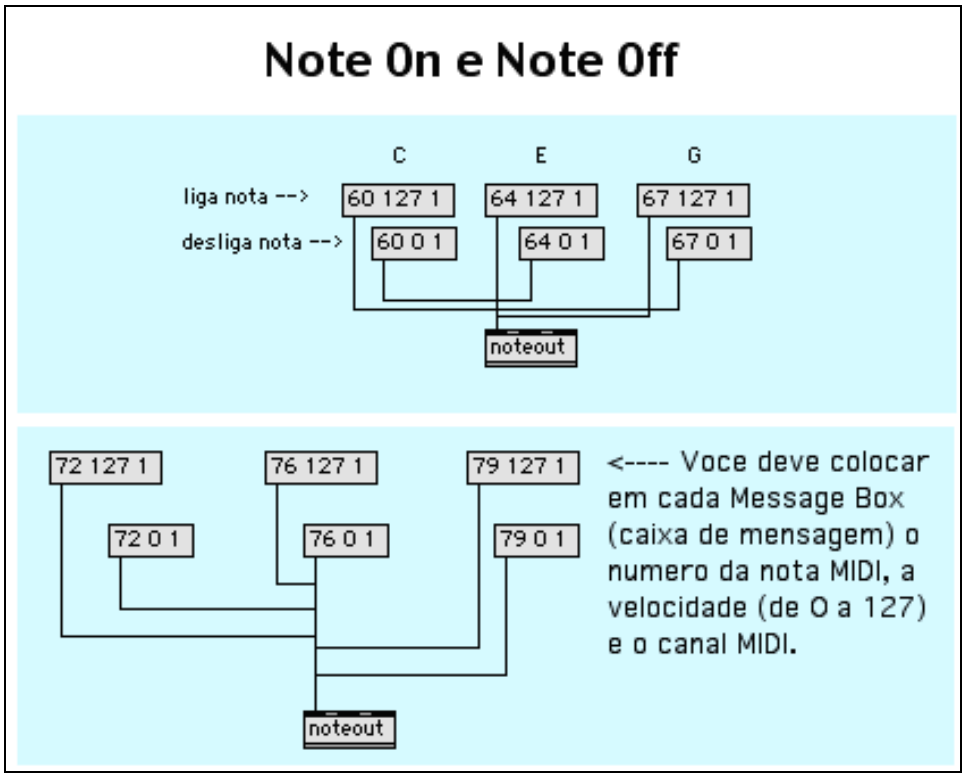


Figura 99 – *Note On e Note Off*

Os três exemplos a seguir ilustram o funcionamento do objeto *makenote*. Para cada *Note on* o ***makenote*** gera uma mensagem de *Note off*. Observe como existem três formas diferentes para resolvermos o envio de uma nota, sua velocidade e duração através do MAX. Estude cada uma das formas de utilização do objeto ***makenote*** e resolva o exercício.

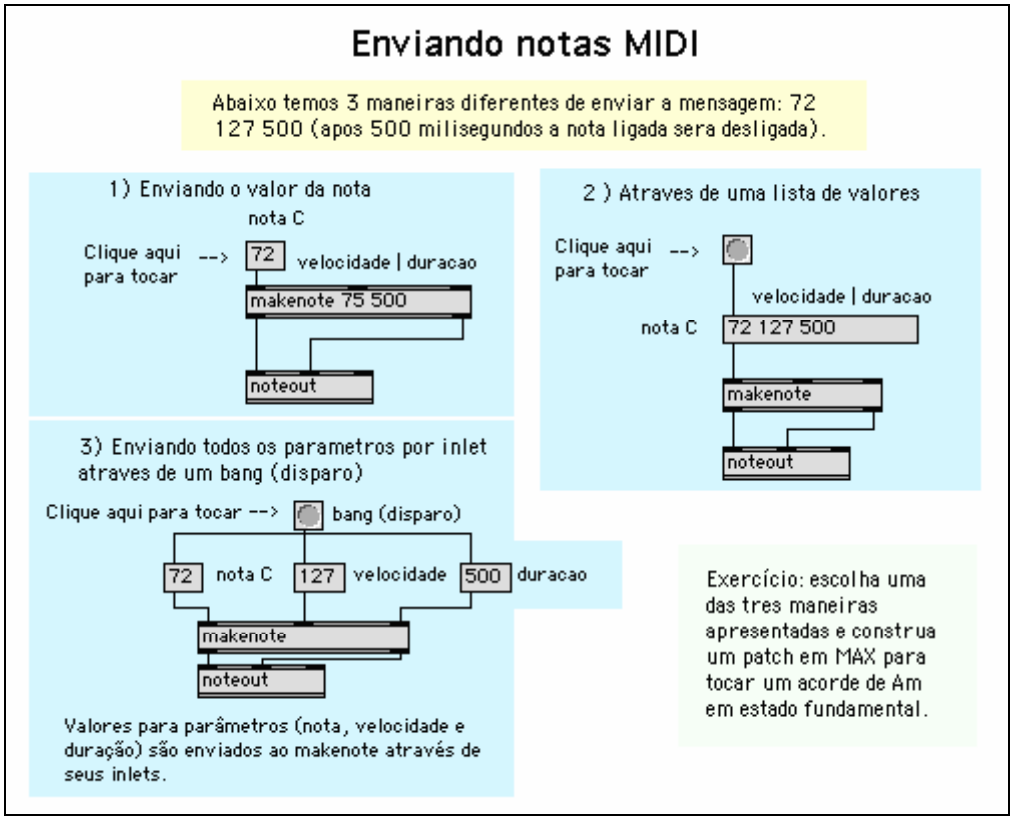


Figura 100 – Três formas de envio de mensagens MIDI utilizando o *makenote*

Uma das possíveis soluções para o exercício proposto é apresentada na figura 101. Observe que a solução foi baseada no exemplo 2 da figura 100. Foi utilizada, portanto, a técnica do envio de uma lista de valores ao makenote.

Observe que na resposta da figura 101 foram colocadas as cifras para identificar a numeração MIDI. Esses comentários sempre devem ser feitos para tornar o programa claro para quem não o fez mas desejam entendê-lo.

O objeto ***comment*** ("comentário"), uma caixa pontilhada na *palette*, não tem efeito no funcionamento de um programa, sendo simplesmente uma maneira de se pôr texto em uma janela *Patcher*.

Use ***comment*** para:

- Rotular objetos no *patch*, tais como "chave liga/desliga";
- Dar instruções ao usuário, como "Clique aqui".
- Explicar de que forma o programa funciona, ou como um determinado item do programa funciona. Isso pode não ser muito útil para o usuário final, mas é útil para você, o programador.

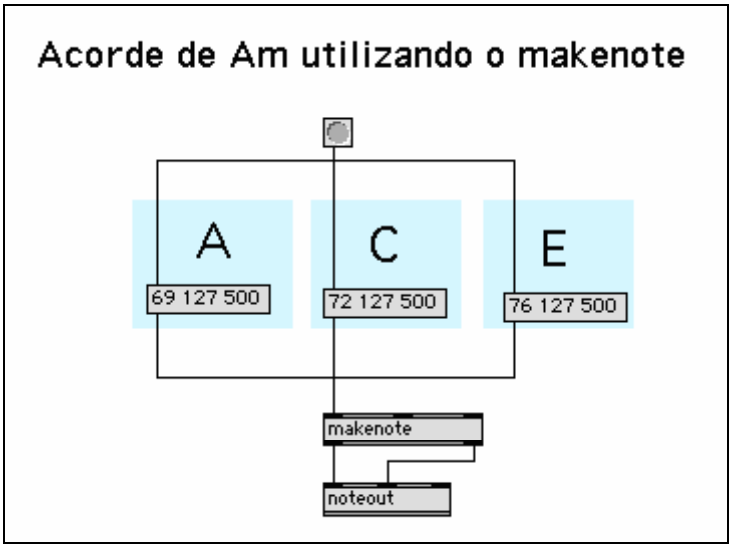


Figura 101 – Programando um acorde

Até agora enviamos notas isoladas ao makenote. Se enviarmos uma lista de números obteremos um acorde. Veja como fazer na figura 102.

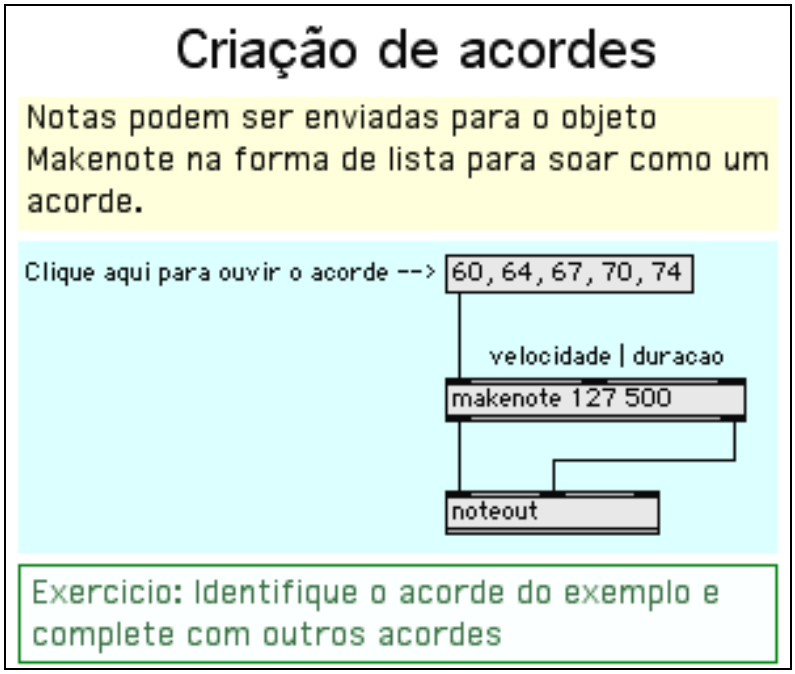


Figura 102 – Lista de notas soando como um acorde

A Vírgula (,) é usada para separar diferentes mensagens em uma caixa de mensagem (*message box*) e enviá-las para a saída uma após a outra.

7.1.8 AULA 2 – Aritmética e execução

Na figura 103, apresentamos os objetos **gswitch** e **bangbang**. O **gswitch** é um objeto que pode possui dois *inlets*, porém, abre apenas um deles por vez. A tomada de entrada que estiver sendo apontada pela seta é a tomada *aberta*, e as mensagens recebidas por ela passam até o *outlet*. As mensagens recebidas pela tomada *fechada* são ignoradas. O *objeto* **bangbang** envia um *bang* (disparo) pelas suas saídas. A ordem de envio dos disparos é da direita para a esquerda. Após utilizar o programa, resolva o exercício.

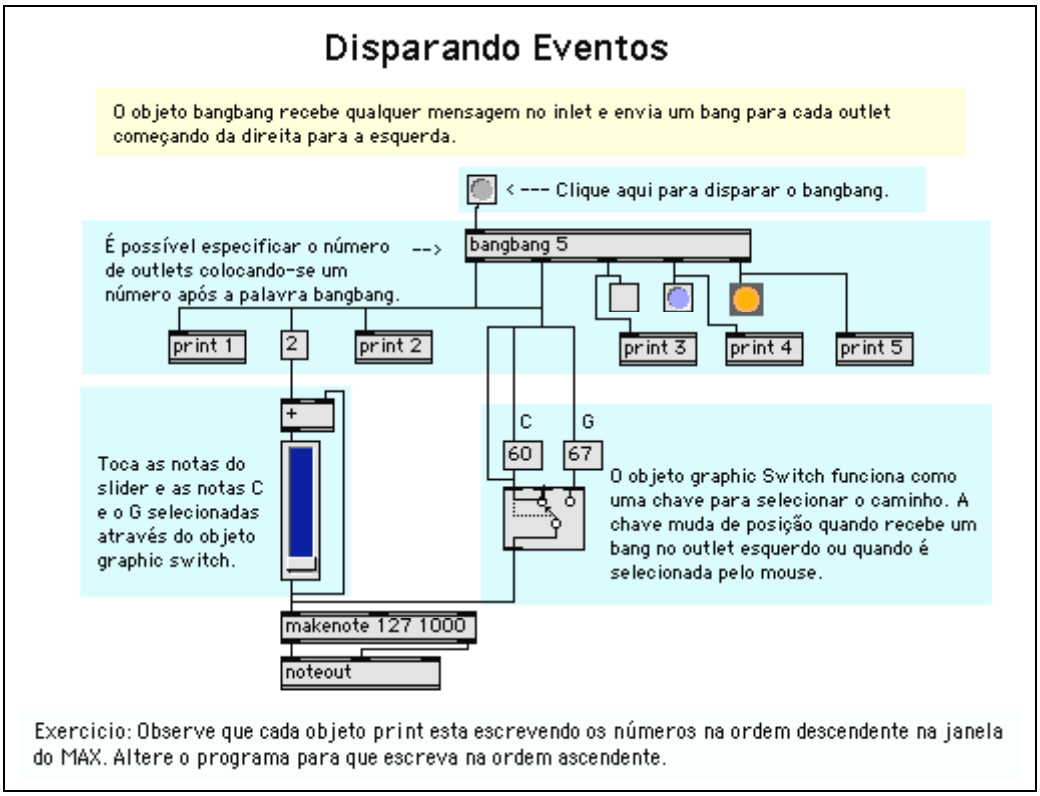


Figura 103 – Disparando eventos

A figura 104 apresenta a solução do exercício. Observe que basta inverter a ordem de impressão dos números para que sejam escritos em ordem ascendente.

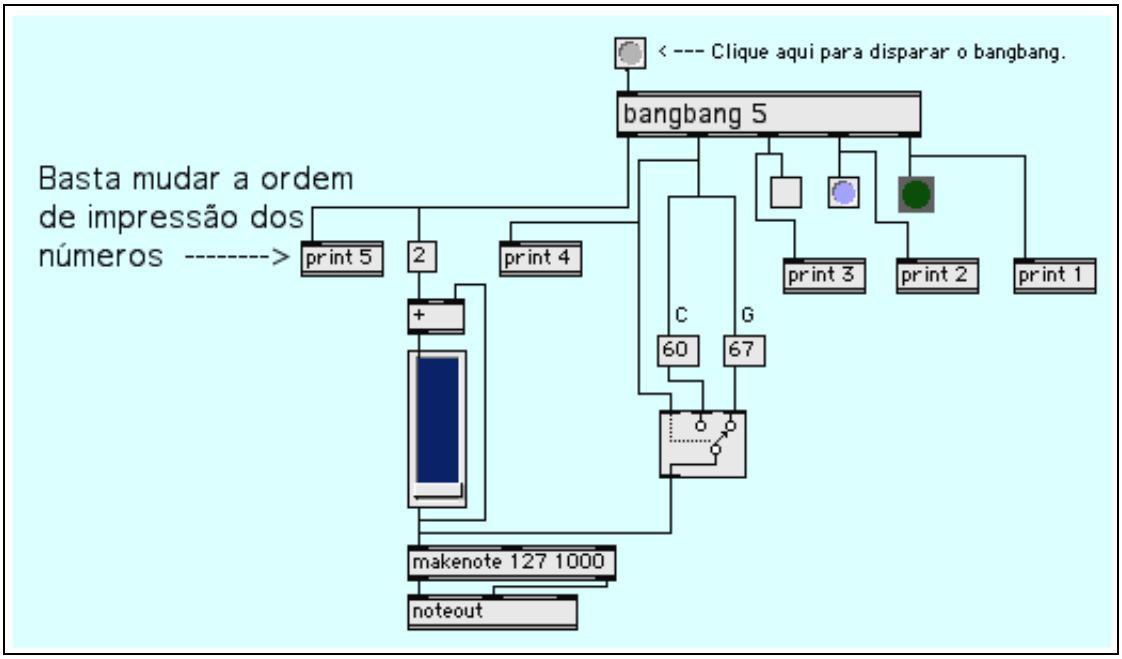


Figura 104 – Invertendo a ordem de execução

Na figura 105 apresentamos um programa que realiza a soma contínua de números. Leia atentamente os apontamentos no *patch* e tente resolver o exercício.

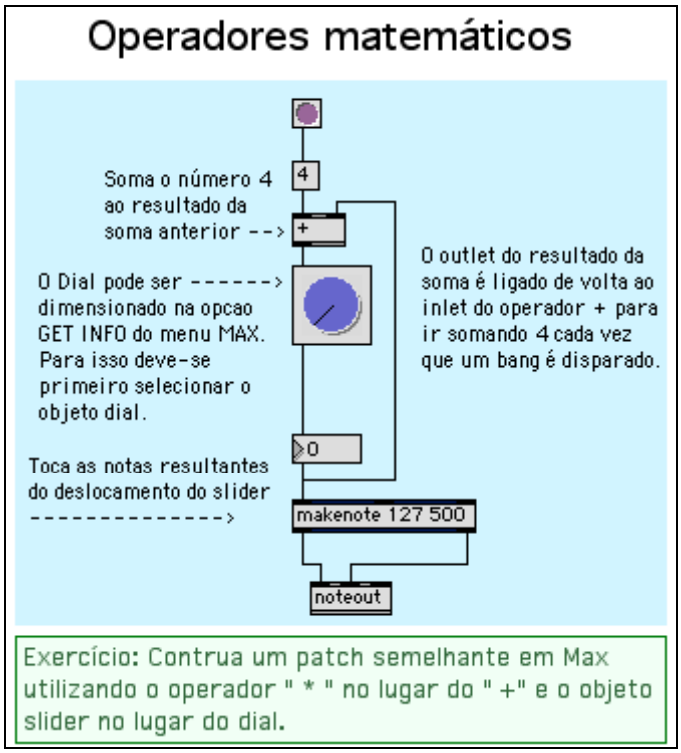


Figura 105 – Realizando adições

A solução do problema proposto é apresentada na figura 106. Observe que o operador matemático de adição foi trocado pelo de multiplicação e o objeto dial foi trocado pelo objeto slider.

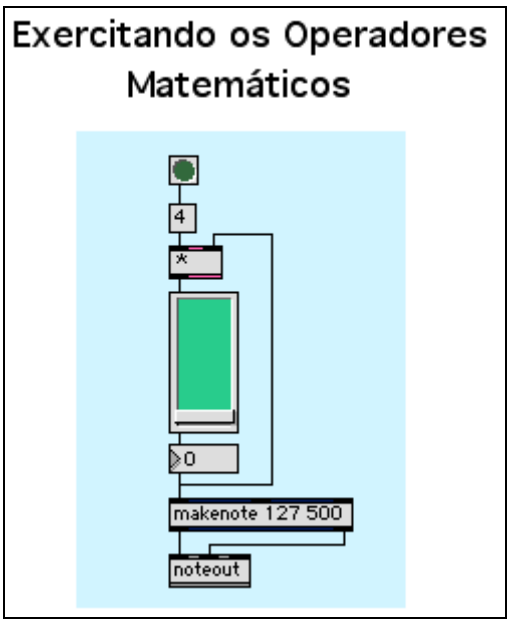


Figura 106 – Exercitando os Operadores Matemáticos

7.1.9 AULA 3 – Estruturas de seleção de dados

No exemplo da figura 107, apresentamos um patch composto pelo objeto Gate. O **Gate** é usado como um argumento que determina o número de saídas. Uma única saída é aberta quando o seu número é recebido na entrada da esquerda. Todas as outras são fechadas. No caso em que o número 0 (zero) for recebido na entrada da esquerda, todas as saídas serão fechadas.

Figura 107 – Selecionando através do objeto gate

A figura 108 mostra a resposta do exercício. Observe que o objeto Gate é ligado no terceiro *inlet* do *makenote*. O terceiro *inlet* é responsável por receber os valores das durações das notas que serão produzidas.

Utilizamos o tempo todo números na programação MIDI de computadores. Por isso o Max possui um objeto gráfico para obtenção desses números. O *Slider* permite a você o envio para a saída de um fluxo contínuo de números dentro de uma determinada faixa. O s*lider* também mostra e envia para a saída os números recebidos em seu *inlet*, tornando-se útil ao apresentar graficamente os números que passam através de si.

Ao se escolher **Get Info...** do menu Max, você pode modificar a faixa do *slider* (*Slider Range*), podendo também especificar um "multiplicador" (*Multiplier*), pelo qual todos os números serão multiplicados antes de serem enviados à saída, e um incremento (*Offset*), que será adicionado ao número, após a multiplicação.

Hslider é um *slider* horizontal e o ***uslider*** é um *slider* vertical. Estes objetos são similares ao ***slider*** convencional, mas com a diferença de re sponderem a um único clique do mouse (não é necessário arrastar) e de poderem ser dimensionados em virtualmente qualquer tamanho, independente da faixa. Além disso, estes objetos limitam os números que recebem à faixa determinada.

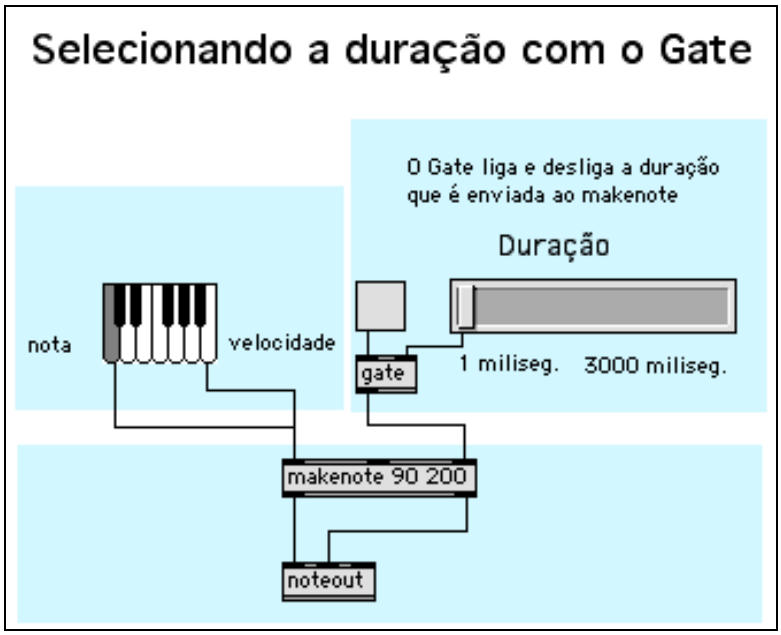


Figura 108 – Selecionando durações com o Gate

Na figura 109, o **Split** faz a seleção de valores. Se o número recebido em sua entrada está dentro do intervalo especificado, ele é enviado pela saída da esquerda. senão, ele é enviado pela saída da direita. Utilize o *patch* do exemplo para uma maior compreensão e em seguida resolva o exercício.

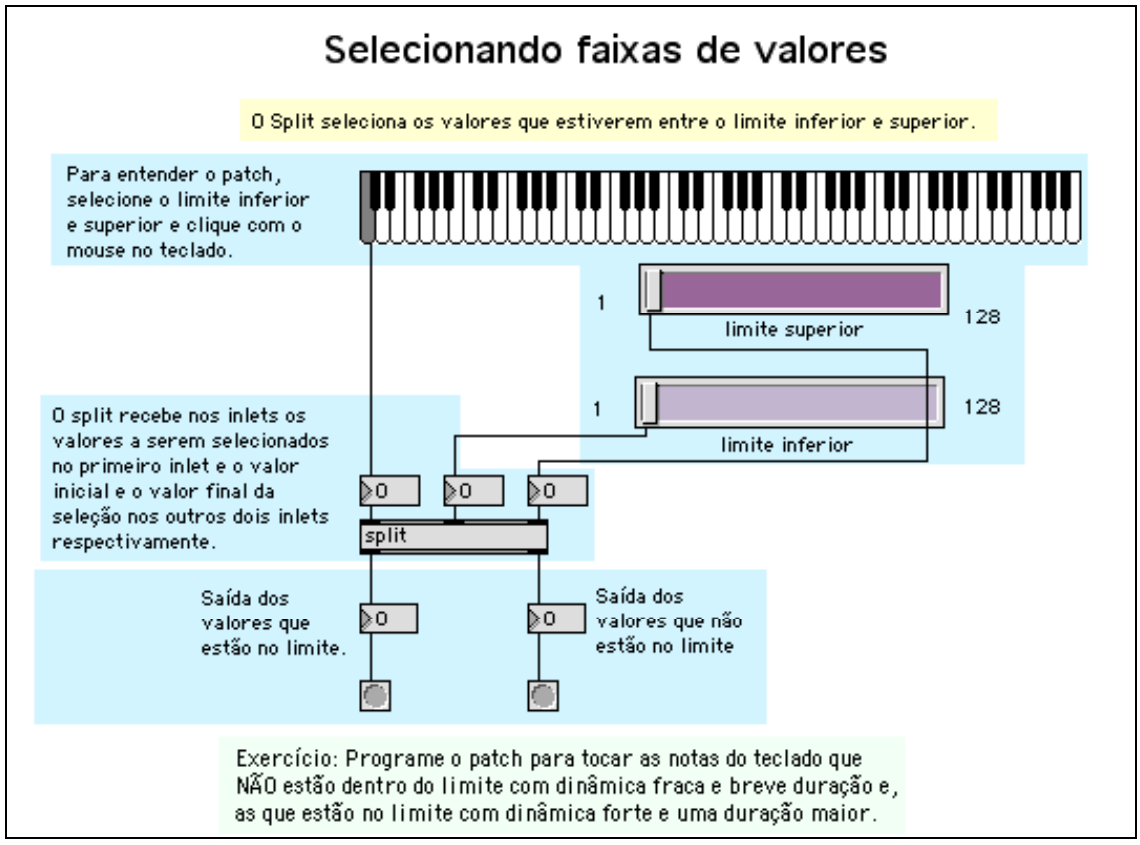


Figura 109 – Objeto split e hslider

Para solucionar o exercício proposto é necessário acrescentar o objeto *makenote*, este objeto possibilita que os valores selecionados sejam interpretados como valores de altura. A solução é apresentada na figura 110.

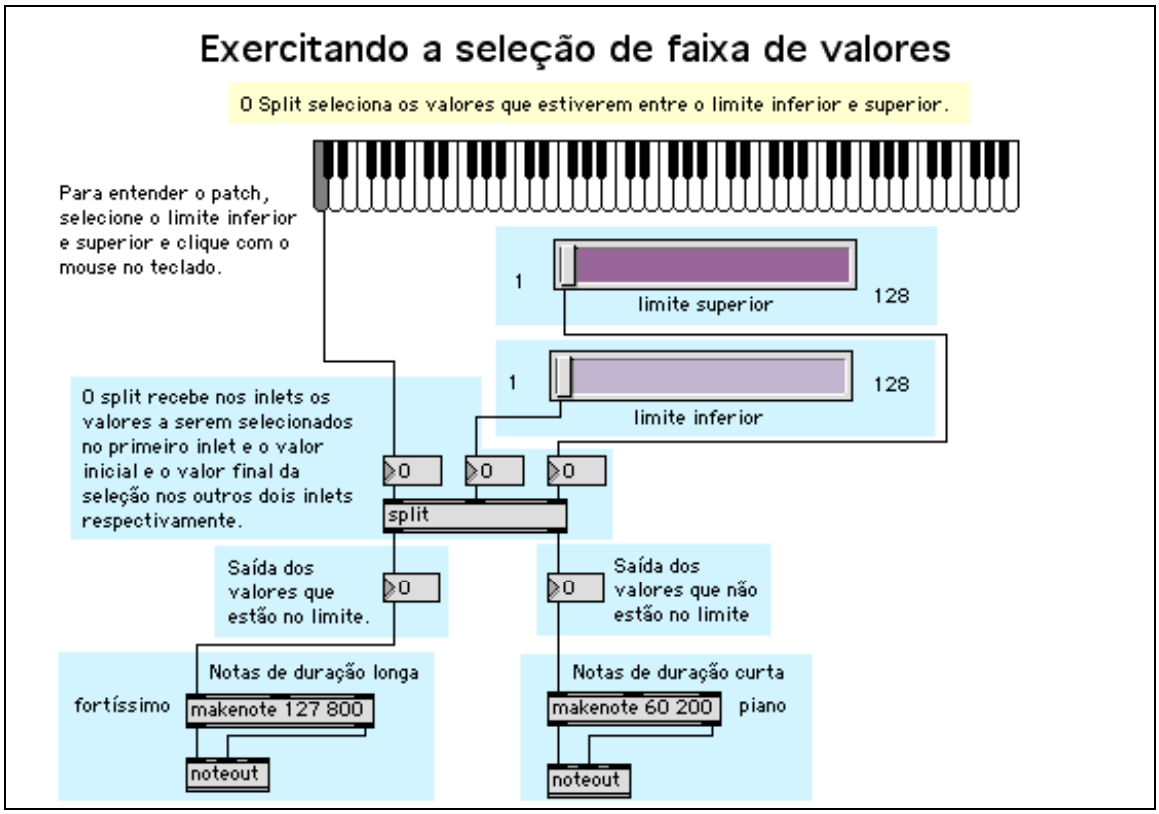


Figura 110 – Acrescentando o ***makenote*** ao *patch* de seleção

Outro objeto utilizado em programação de computadores, para realizar a seleção de valores é o switch. Estude o funcionamento do exemplo da figura 111 e compare com o funcionamento do exemplo da figura 107 que utiliza o objeto gate.

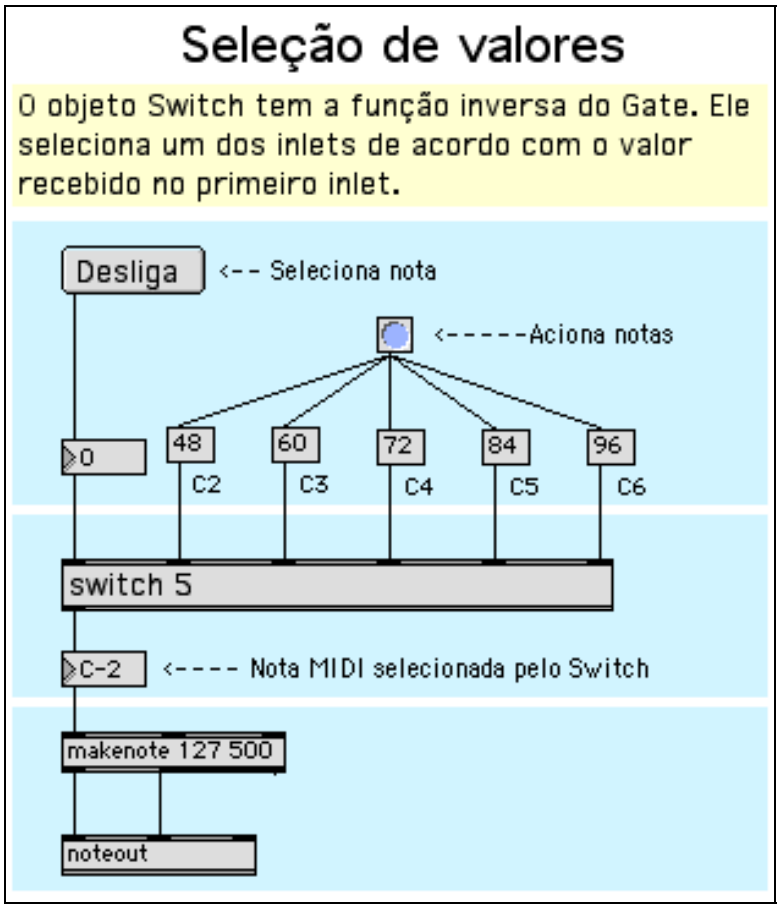


Figura 111 – Selecionando com o objeto *switch*

No exemplo da figura 112, a cada clique que o usuário realizar no botão de bang será sorteado um valor pelo objeto **random**. Este valor será enviado ao objeto **select** que irá selecionar, ou não, o valor. O valor da altura não selecionado pelo objeto **select** é enviado ao próximo objeto **select** pelo outlet direito. Os valores de altura que não forem selecionados pelos dois objetos **select** irão soar. Estude o programa abaixo e tente resolver o exercício.

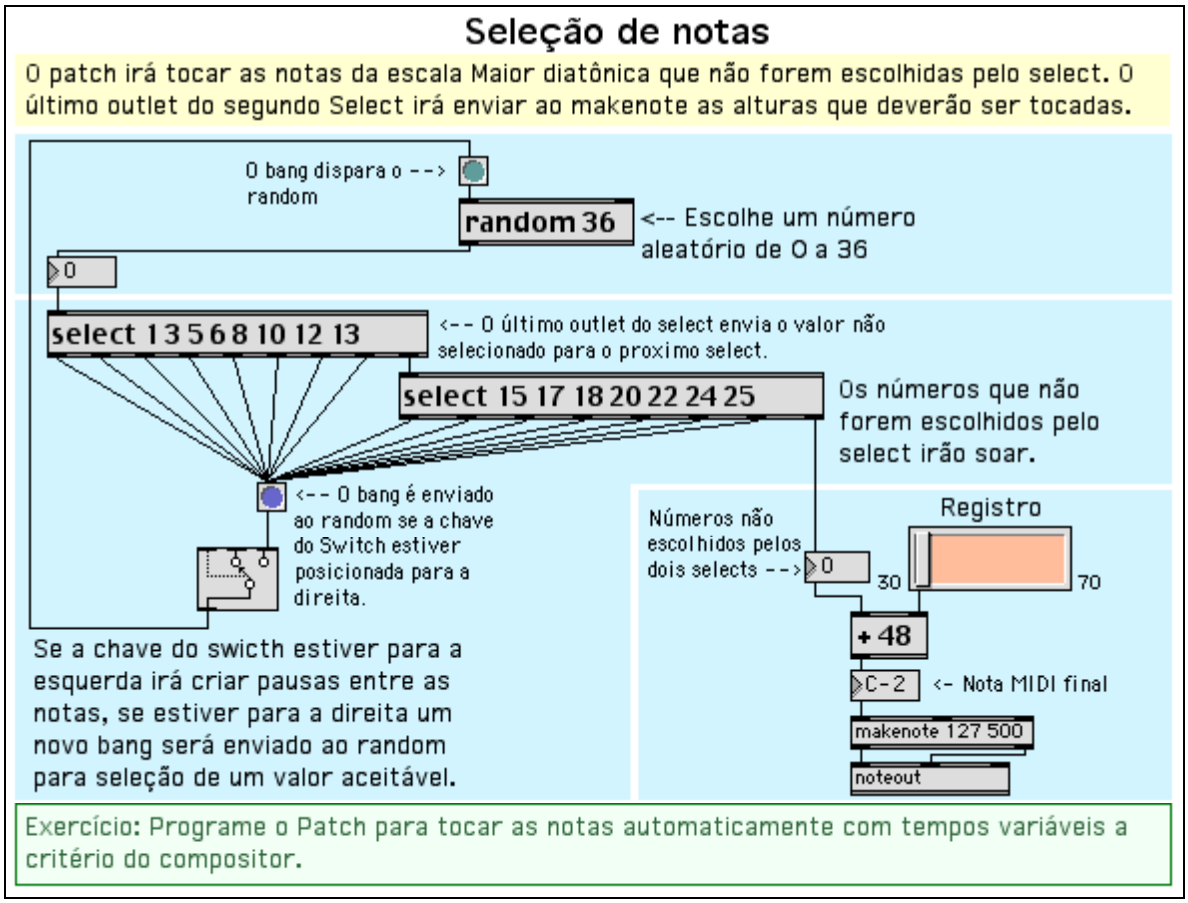


Figura 112 – Utilizando o objeto “select” para seleção de notas

Para solucionar este exercício é necessário utilizar o comando **metro**. O **metro** possibilita "automatizar" o programa através da execução de repetições automáticas dos procedimentos. Na resposta apresentada na figura 113, o metro inicia com um valor padrão de 300 milisegundos e pode ser ajustado pelo *slider* que está ligado ao seu *inlet* direito. Observe na figura 74 como o **metro** é empregado para automatizar o processo de execução do programa.

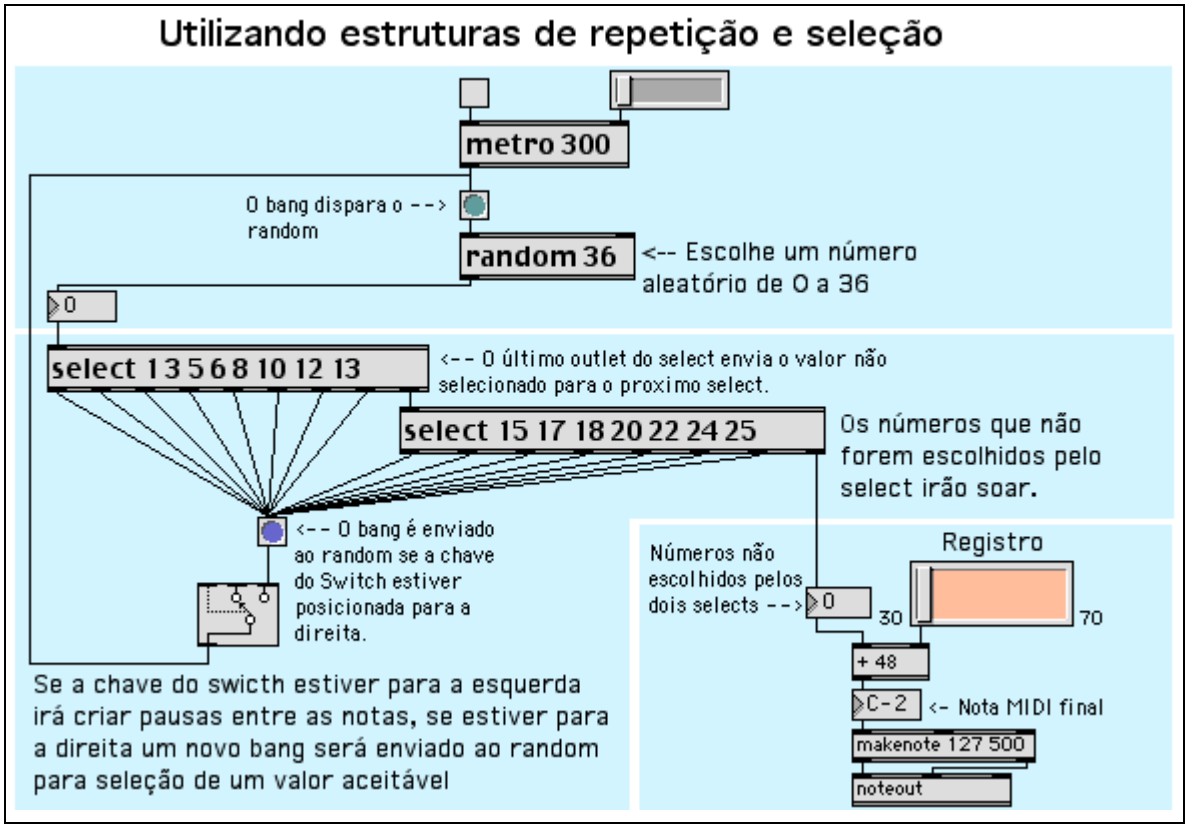


Figura 113 – Utilizando estruturas de repetição e seleção

7.1.10AULA 4 – Estruturas condicionais

Nas linguagens de programação de computadores como o MAX, dispomos de comandos que nos permitem escolher caminhos e dizer ao computador o que ele deve selecionar. No exemplo da figura 114, o comando **If** avalia uma declaração condicional na forma "***if*** *x* é verdadeiro ***then*** ("então") envie *y* ***else*** ("senão") envie *z*". A declaração condicional pode conter argumentos modificáveis (variáveis). Selecione notas em ambos os teclados e procure entender o funcionamento do programa executando e lendo atentamente as explicações. Após resolva o exercício.

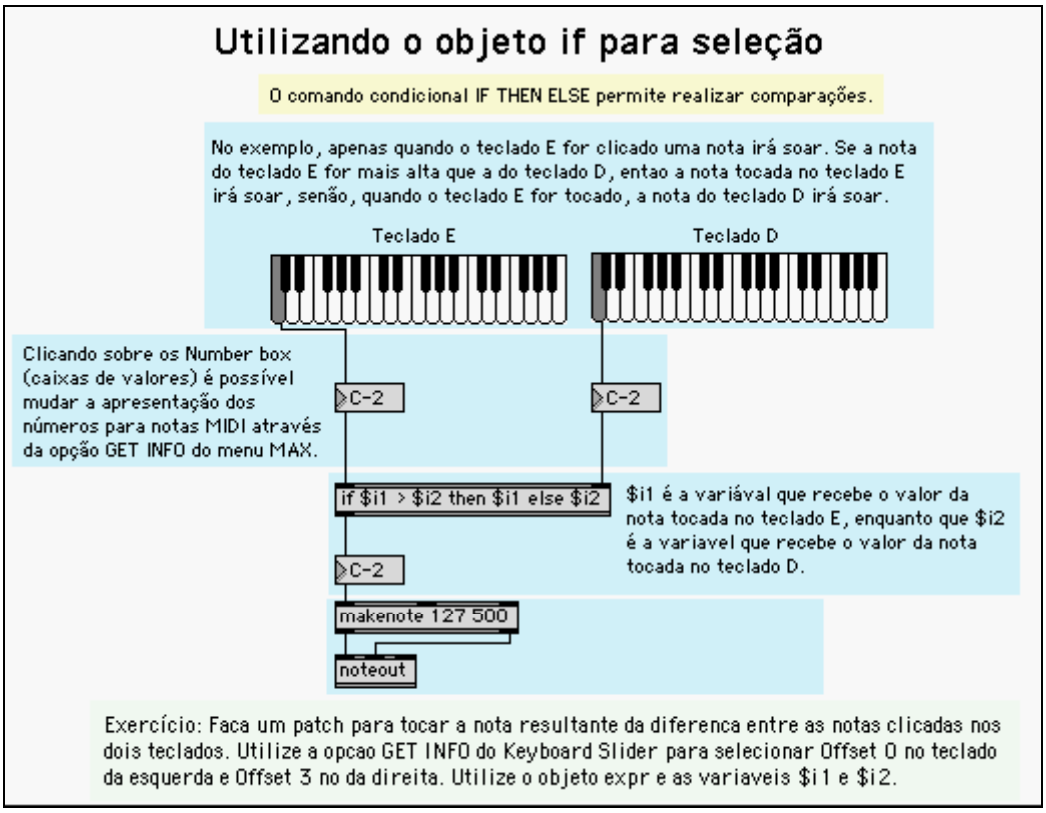


Figura 114 – O comando condicional *if*

Quando temos o ***Cifrão*** **($)**, seguido imediatamente por um número (como em $1), trata-se de um *argumento modificável*. Quando a caixa de mensagem recebe uma mensagem em sua tomada de entrada (*inlet*), cada argumento modificável é substituído pelo item correspondente na mensagem ($1 é substituído pelo primeiro item, $2 é substituído pelo segundo item, etc.) Se nenhum item estiver presente na mensagem para entrar na substituição, o valor previamente armazenado será utilizado. Se nenhum valor foi armazenado ainda, será usado o valor *default*(padrão) 0.

Para resolver o exercício proposto na figura 114, é necessário a utilização do objeto **expr.** Este objeto interpreta o argumento como sendo uma expressão matemática à maneira da linguagem C. Quando um número é recebido na tomada da esquerda, ***expr*** substitui os argumentos modificáveis (variáveis), avalia a expressão e envia o resultado.

A expressão no argumento de ***expr*** pode conter funções matemáticas em C tais como *pow( )* e *sin ( )*, podendo inclusive conter operadores relacionais (< , >).

A solução para a programação de uma expressão aritmética é apresentada na figura 115.

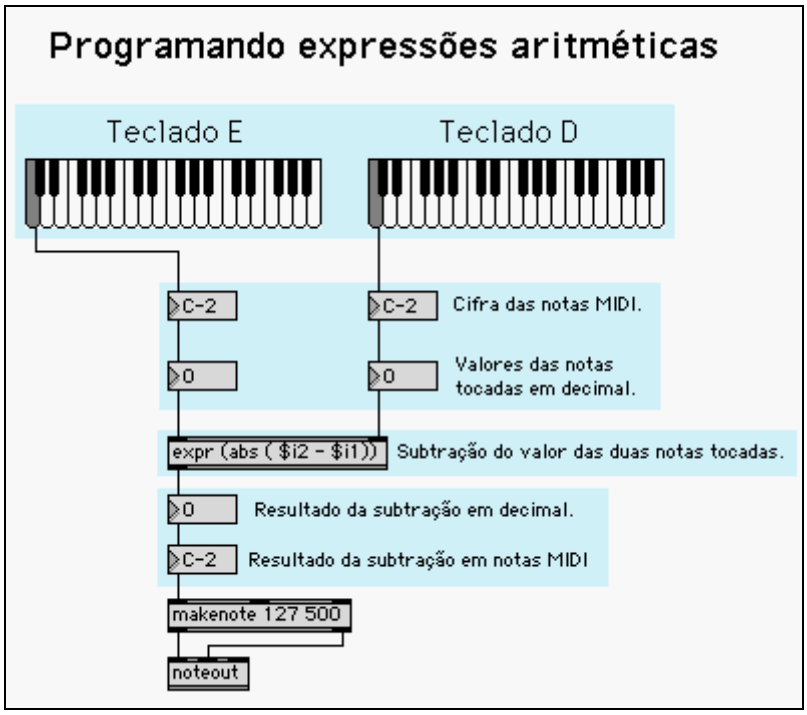


Figura 115 – Programação de uma expressão aritmética

No módulo de Composição Musical utilizaremos expressões musicais para a programação de funções probabilísticas.

O comando IF é muito utilizado em programação, por isso estudaremos *patchs* que exemplificam outras formas de utilização do comando condicional. No exemplo da figura 116 é apresentado outro exemplo do comando IF. Estude o programa e resolva o exercício proposto.

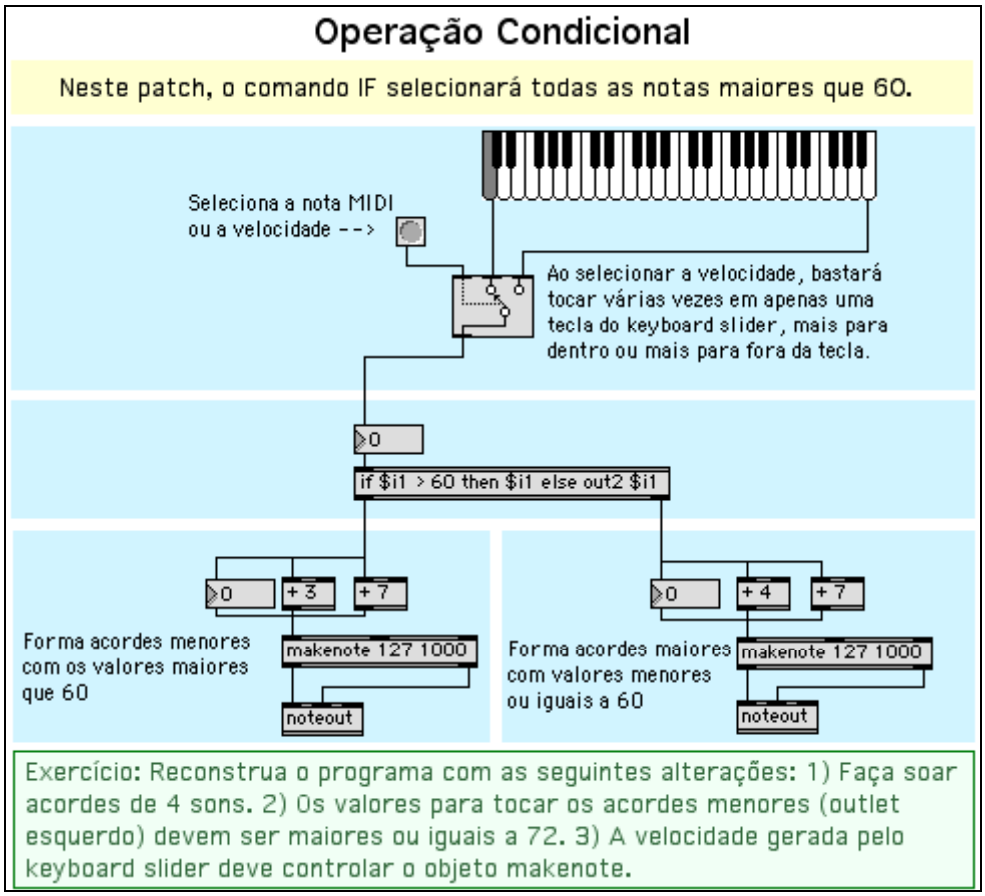


Figura 116 – Selecionando uma nota MIDI

Para receber as mensagens de notas MIDI geradas por um instrumento controlador externo, utilizamos o objeto *Notein*. O *Notein* espera apenas por uma mensagem MIDI de nota e, quando ela chega, *Notein* envia o número da tecla, a velocidade e o número do canal. Observe o exemplo da figura 117 e resolva o exercício.

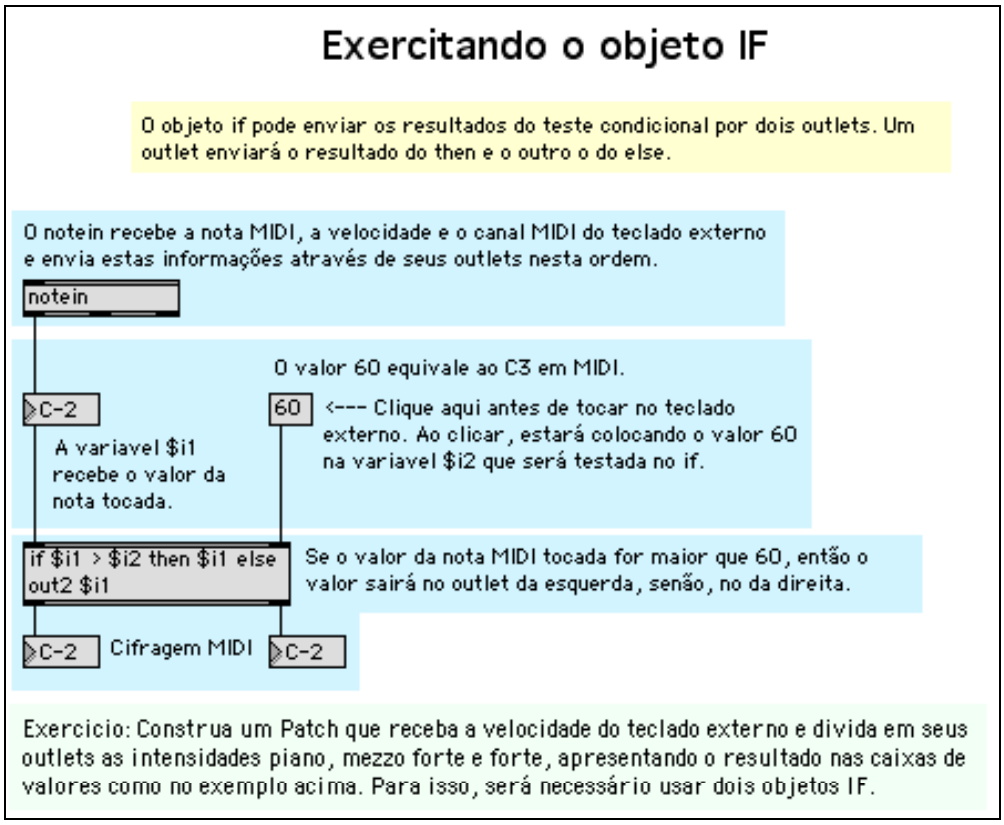


Figura 117 – Exercitando o objeto "if"

Para solucionar o exercício é necessário utilizar a técnica de "*Ifs* encadeados". Observe, na solução do exercício da figura 118, que o outlet à direita do objeto *if* envia o valor que não foi satisfeito pela condição para o próximo *if* para que seja testado novamente. Estude a solução do exercício e procure comparar com o seu programa.

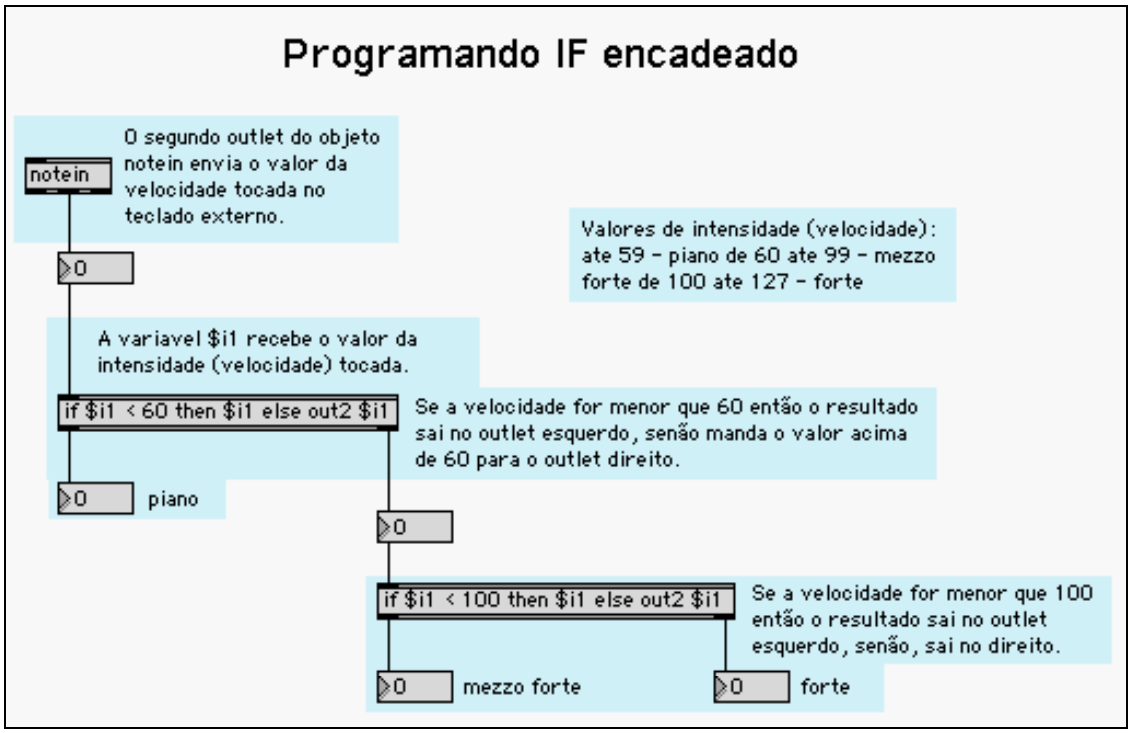


Figura 118 – Utilização de *ifs* encadeados

7.1.11AULA 5 – Estruturas de repetição

O exemplo da figura 119 serve para mostrar a utilização do objeto **metro** que funciona como uma espécie de metrônomo podendo ser ligado e desligado. O **metro** deve ser acompanhado de um parâmetro. No exemplo inserimos o número 1000 após a palavra *metro* na caixa de objeto. Este é o número de milissegundos entre as batidas do metrônomo. Um *metro* envia mensagens *bangs* repetidamente em intervalos regulares de tempo, até ser desligado.

O ***Toggle*** é uma caixa com um X, na *palette* de objetos. Funciona como um indicador ou interruptor entre dois estados: zero e não-zero. Este objeto pode receber um número ou um *bang* em sua tomada de entrada. Se o número for não-zero, ***toggle*** irá exibir um X e enviar o número para a saída. Se o número for 0, a caixa estará em branco e será enviado um zero para a tomada de saída. O objeto ***toggle*** deve ser usado como uma chave de liga/desliga.

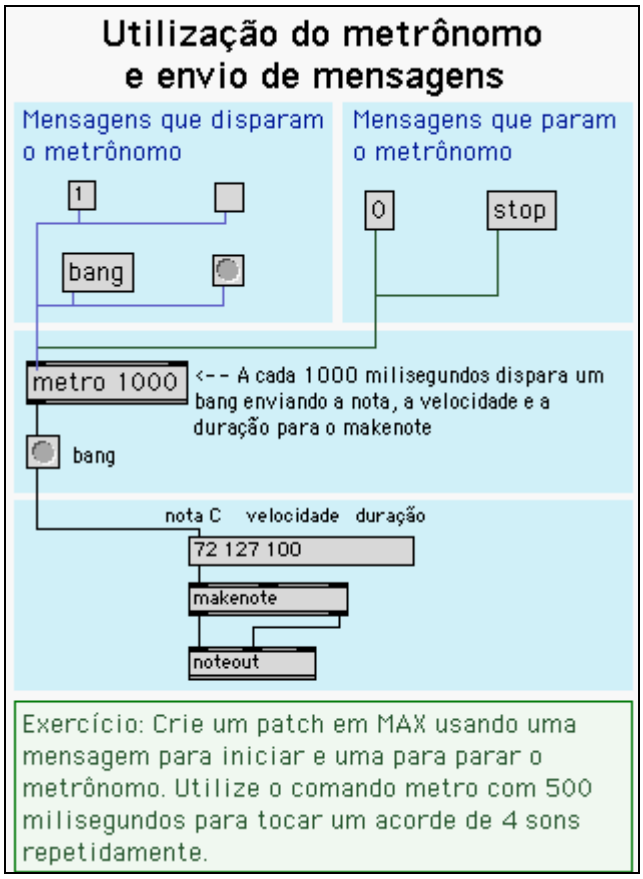


Figura 119 – Controle do tempo de envio de mensagens através do *metro*

A resposta para o exercício proposto é apresentada na figura 120. Foi utilizado um objeto **Toggle** para iniciar e para a execução do metro e quatro listas de valores, uma para cada nota que será tocada.

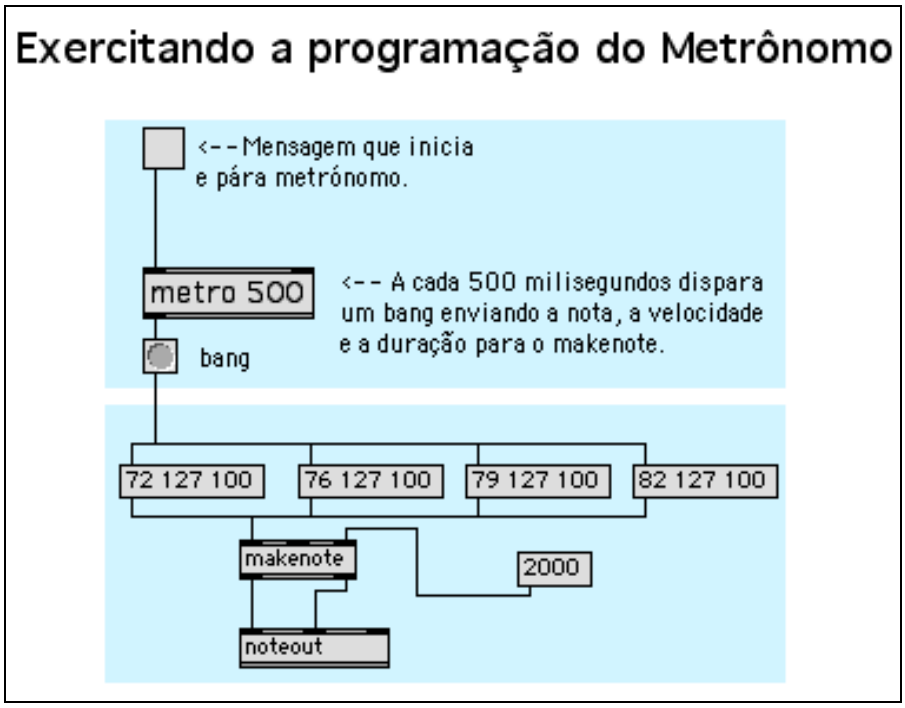


Figura 120 – Exemplo para tocar um acorde de quatro sons repetidamente

No exemplo da figura 121 é empregada uma técnica para contar ocorrências. Através da repetição, obtida pelo emprego do objeto **metro**, é possível criar um processo cíclico de execução em que podemos criar contadores. Os contadores servem para mostrar o número de ocorrências que foram realizadas durante ciclos de execução do programa. No programa abaixo são contadas quantas notas Do foram selecionadas e quantas notas não foram selecionadas pelo *select*. Observe a utilização do objeto **select**. Se ele selecionar uma das notas Do (nas oitavas estabelecidas) ele aciona um *bang* no *outlet* de saída que corresponde à posição do valor do objeto **select**. Toda vez que uma nota Do for selecionada pela contagem automática do programa, um acorde de Do irá soar e o valor do contador de notas Do selecionadas será incrementado de um. Utilize o programa para realizar a contagem automática e observe os resultados sonoros e os números nas caixas de valores. Após estudar o programa realize a alteração solicitada.

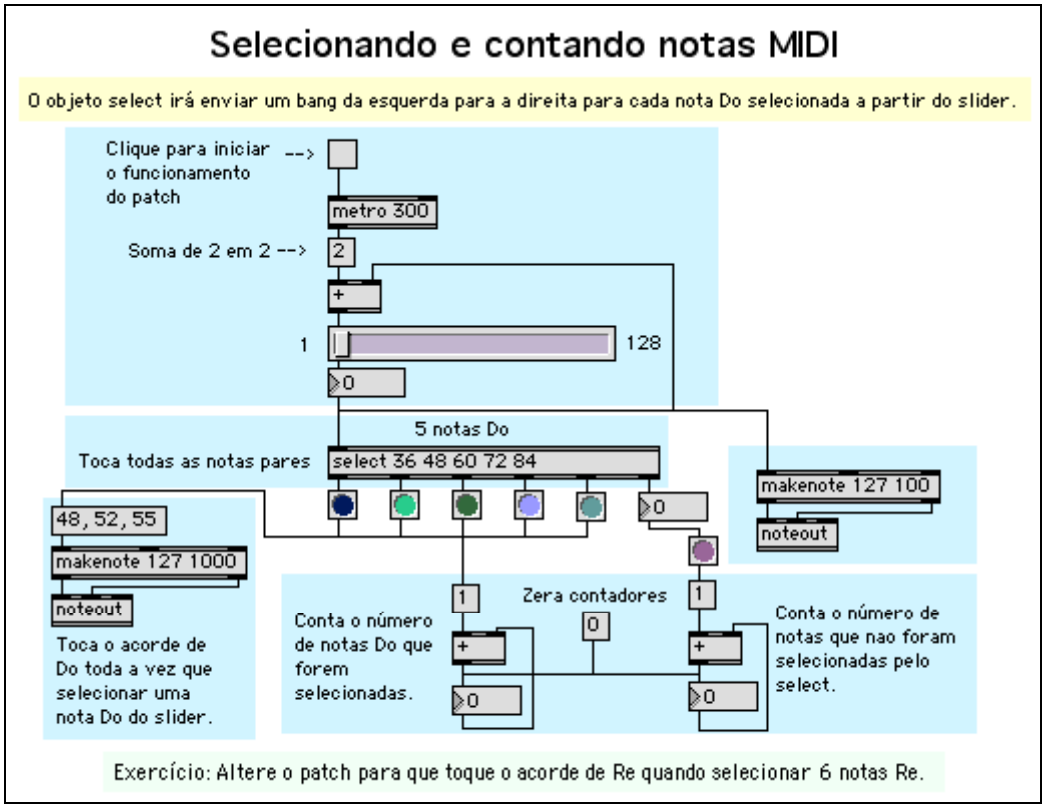


Figura 121 – Selecionando valores e contando as ocorrências

Para que o programa toque um acorde de Re ao invés de um acorde de Do, basta reprogramar os valores das notas MIDI no objeto **select.** A geração do acorde de Do também deverá ser reprogramada para que os números que formam o acorde de Re, sejam enviados ao **makenote**. Observe os valores das notas MIDI na resposta na figura 122 e compare com a sua solução.

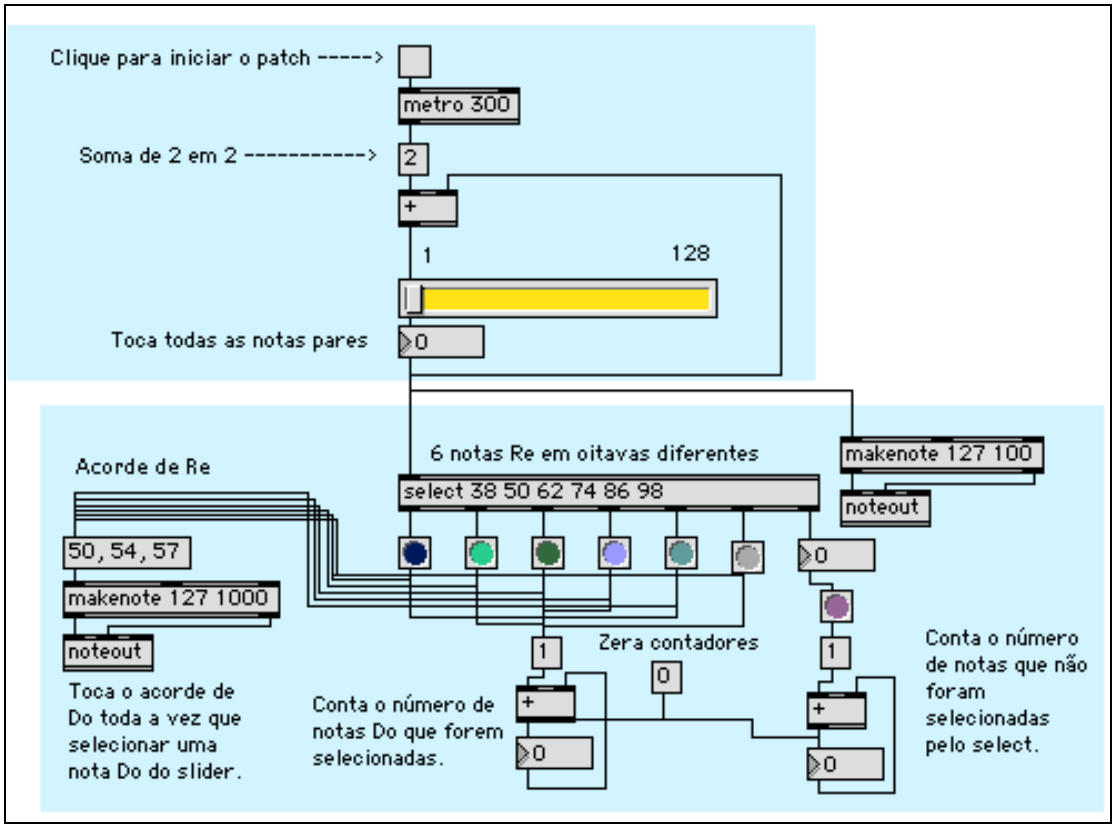


Figura 122 – Resposta do patch "Selecionando e contando notas MIDI"

No exemplo da figura 123 temos um seqüenciador monofônico de 9 passos. Procure utilizá-lo movendo os *sliders* e acionando o botão do canto superior esquerdo. Observe que temos um somador que irá incrementar os passos e enviar os números ao **gate** para selecionar as notas. É fundamental o entendimento do objeto **gate** e da estrutura formada pelo somador e pelo comando **if**. Leia atentamente as explicações contidas no *patch* e depois resolva o exercício proposto.

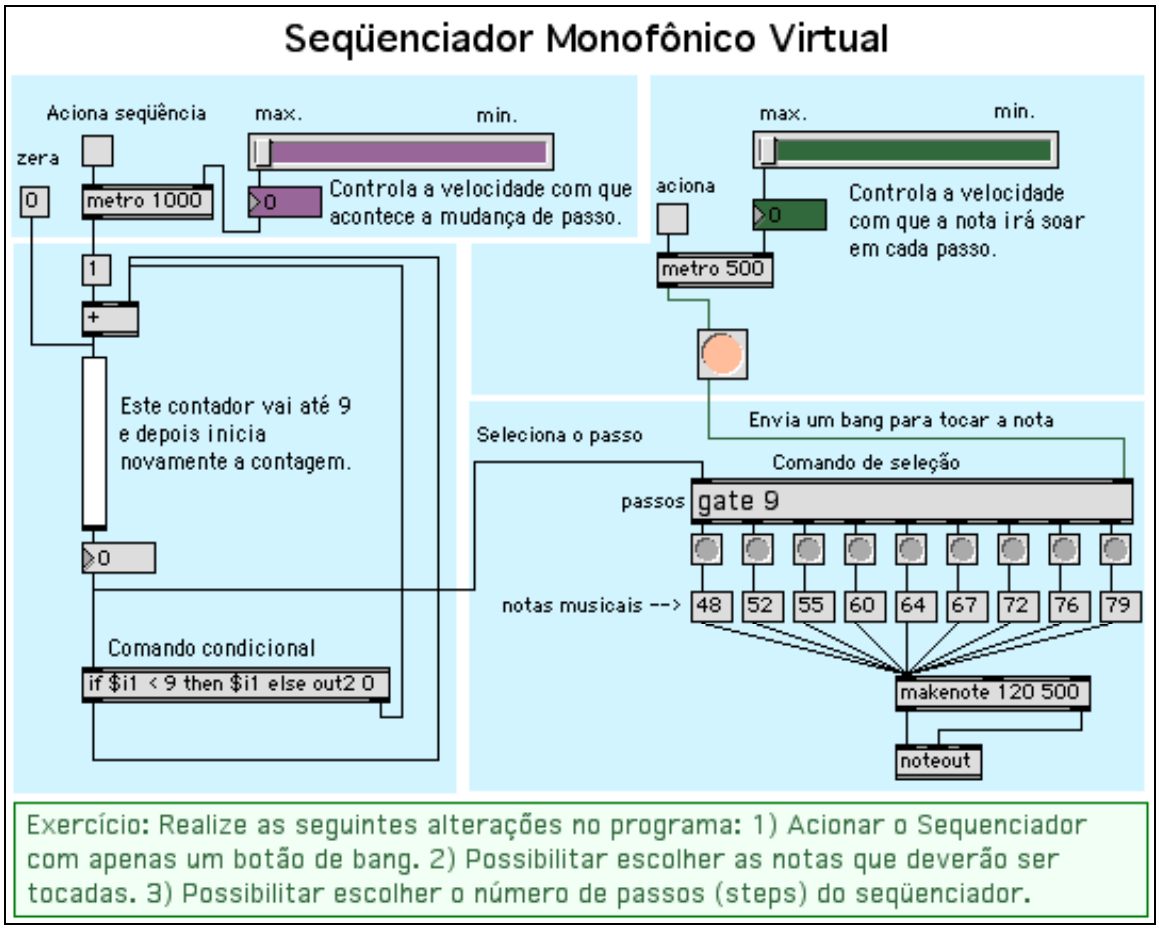


Figura 123 – Seqüenciador Monofônico Virtual

Para realizar as alterações sugeridas no exercício da figura 123 é necessário um perfeito entendimento do programa.

Para acionar o seqüenciador com apenas um botão é necessário utilizar apenas um **Toggle** ligado aos dois objetos **metro**.

Ao escutar o resultado sonoro do Seqüenciador Monofônico Virtual podemos perceber que as alturas são formadas por valores fixos. Para entender como mudar isso, observe, na figura 124 que, no lugar de caixas de mensagens, são programadas caixas numéricas onde o valor inicial das notas pode ser escolhido pelo compositor antes e durante a execução do programa.

Para possibilitar que o compositor controle o número de passos do seqüenciador deve-se ligar um *slider* (dimensionado com 9 passos) no *inlet* direito do objeto **if**. O objeto **if** deverá receber o valor do objeto *slider* e avaliá-lo. Para isso, é necessário que seja criada mais uma variável para receber o valor que irá entrar pelo *inlet* direito em que o *slider* for ligado. Estude a figura 124 para maiores detalhes referentes à solução do exercício.

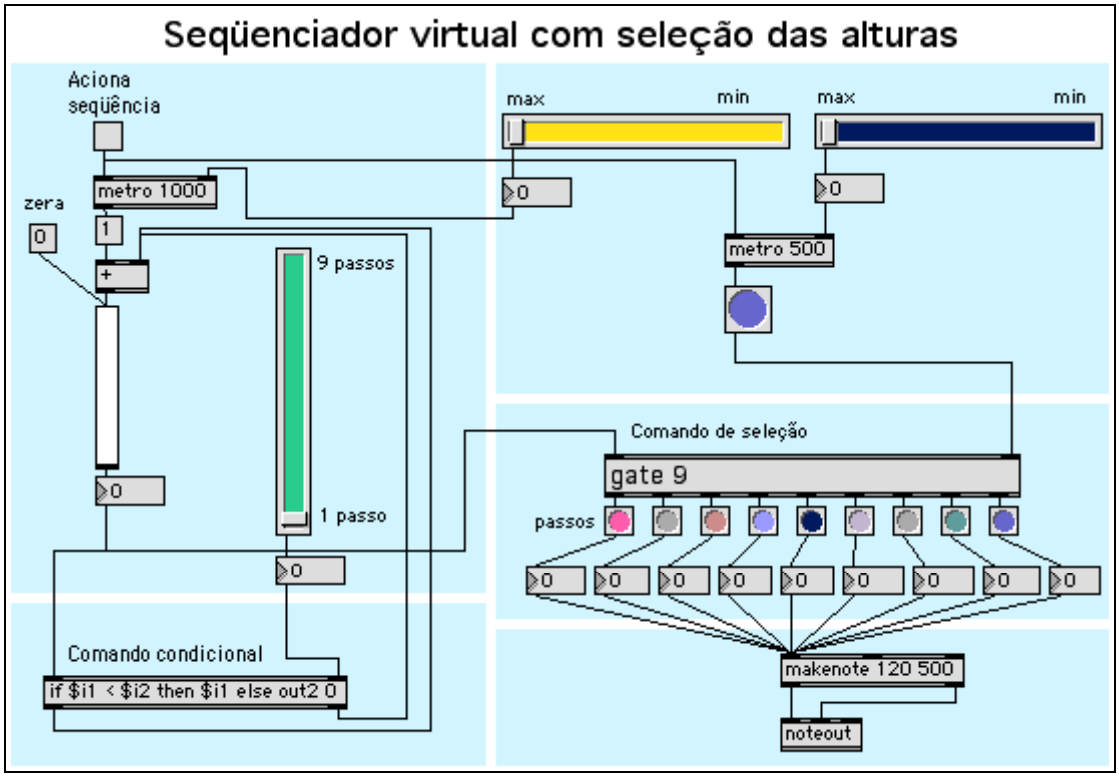


Figura 124 – Sequenciador virtual com Seleção das alturas

O exemplo da figura 125 apresenta a utilização de um contador em conjunto com o metrônomo. Até então, o contador era implementado pela utilização do objeto "+". Porém o MAX disponibiliza um objeto próprio para essa função que pode facillitar a programação. O objeto ***counter*** não é um objeto temporizador por si só, mas é freqüentemente usado em conjunto com ***metro***, porque ***counter*** conta o número de *bangs* que recebe. A combinação ***metro-counter*** é uma maneira eficaz de se incrementar ou decrementar um valor repetidamente. Os argumentos de ***counter*** são três: o primeiro especifica a direção da contagem (0 para "cima" e 1 para "baixo"), o segundo ajusta o valor mínimo da contagem e o terceiro ajusta o valor máximo.

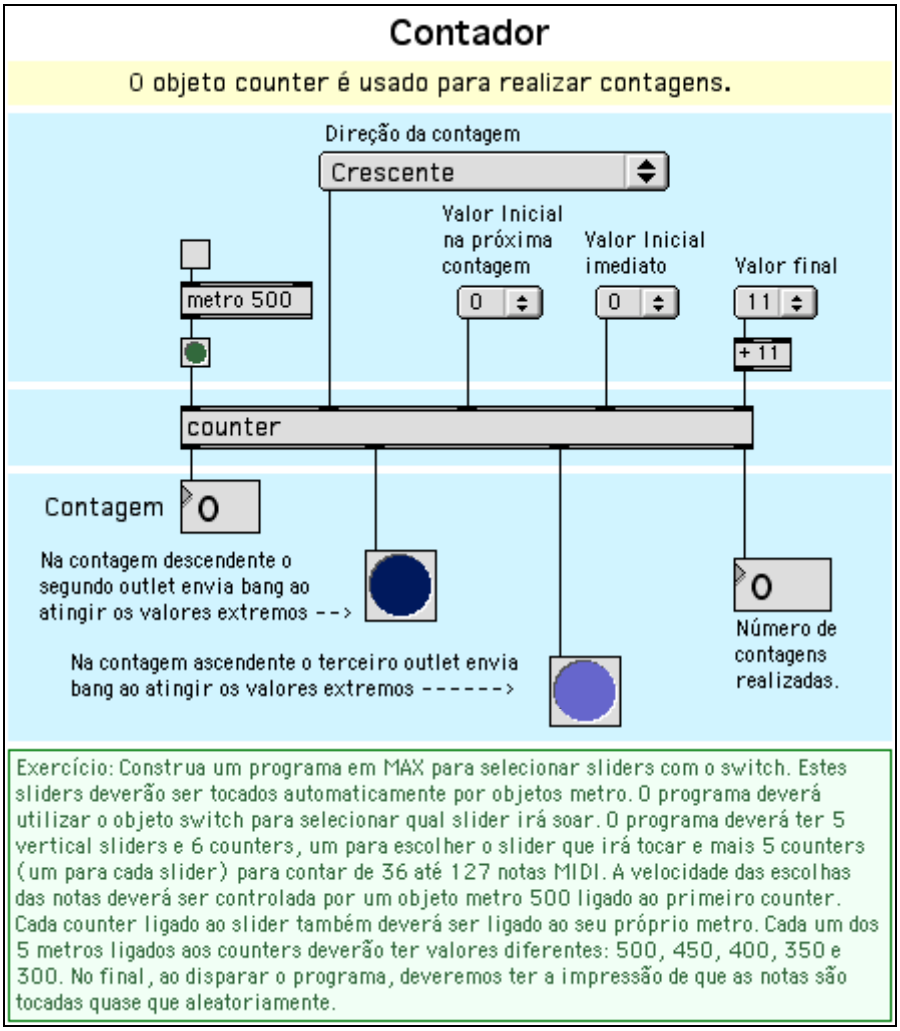


Figura 125 - Contador

Para solucionar esse exercício é necessário acompanhar o enunciado passo a passo e ir programando conforme as indicações. A figura 126 apresenta a solução do exercício. Nessa solução foram programados vários objetos **metro** com diferentes andamentos. A eles foram ligados os objetos **counter** que são selecionados pelo **switch** de acordo com os valores provindos do objeto **counter** mais à esquerda do *patch*.

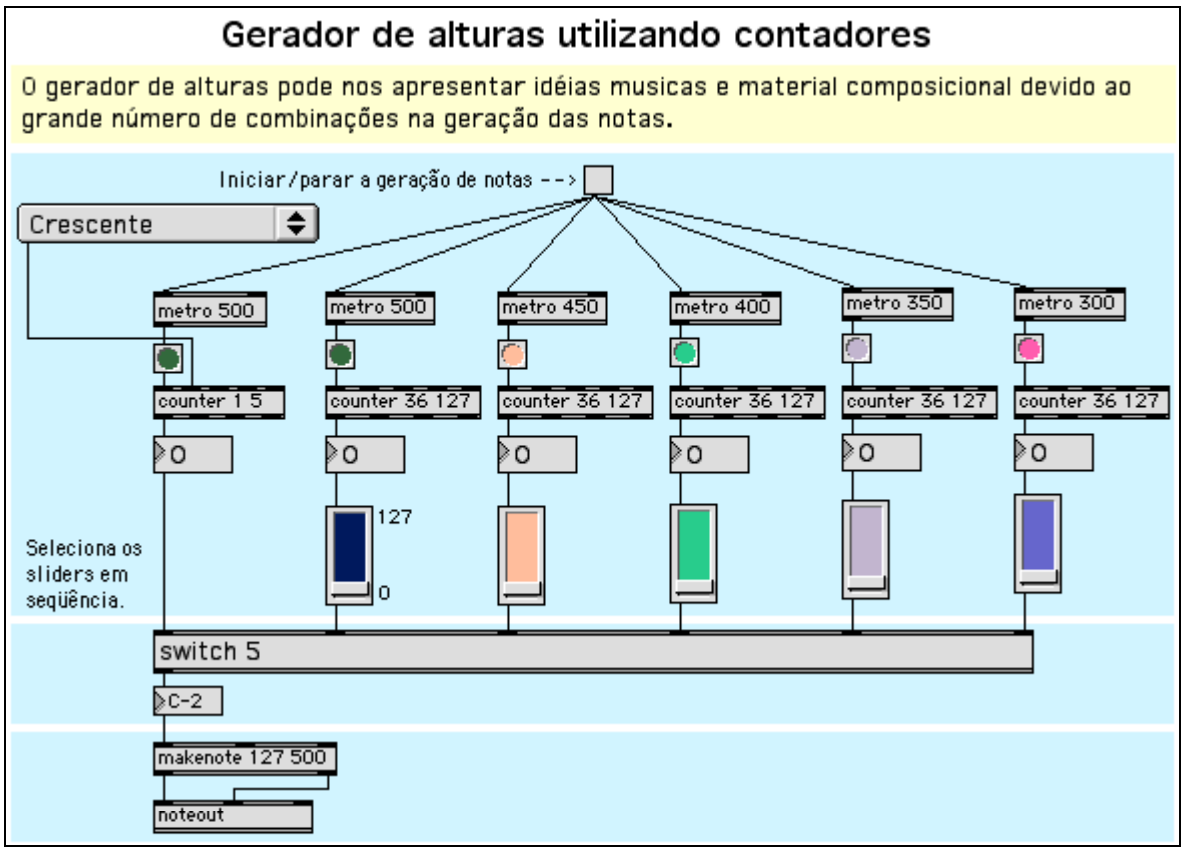


Figura 126 – Programação de contadores

Na figura 127 mostramos a utilização de um outro tipo de metrônomo em MAX que lida com parâmetros musicias ao invés da especificações em milissegundos requeridas por ***metro, clocker*** e ***line***. O primeiro argumento de ***tempo*** (ou um número recebido pela tomada central da esquerda) indica o tempo metronômico em termos de batidas por minuto – ou semínimas por minuto – tal qual um metrônomo tradicional. O segundo e terceiro argumentos (ou os números fornecidos às duas tomadas mais à direita) especificam que fração de uma semibreve o objeto ***tempo*** irá usar ao enviar as pulsações do metrônomo. Por exemplo, se o segundo e terceiro argumentos forem 1 e 16, a fração será 1/16 de uma semibreve e ***tempo*** enviará um número de 0 a 15 para cada semicolcheia, baseado no andamento de semínimas especificado. Uma fração de 2/3 enviaria tiques de quiálteras de mínima (um tique a cada 2/3 de semibreve), e assim por diante.

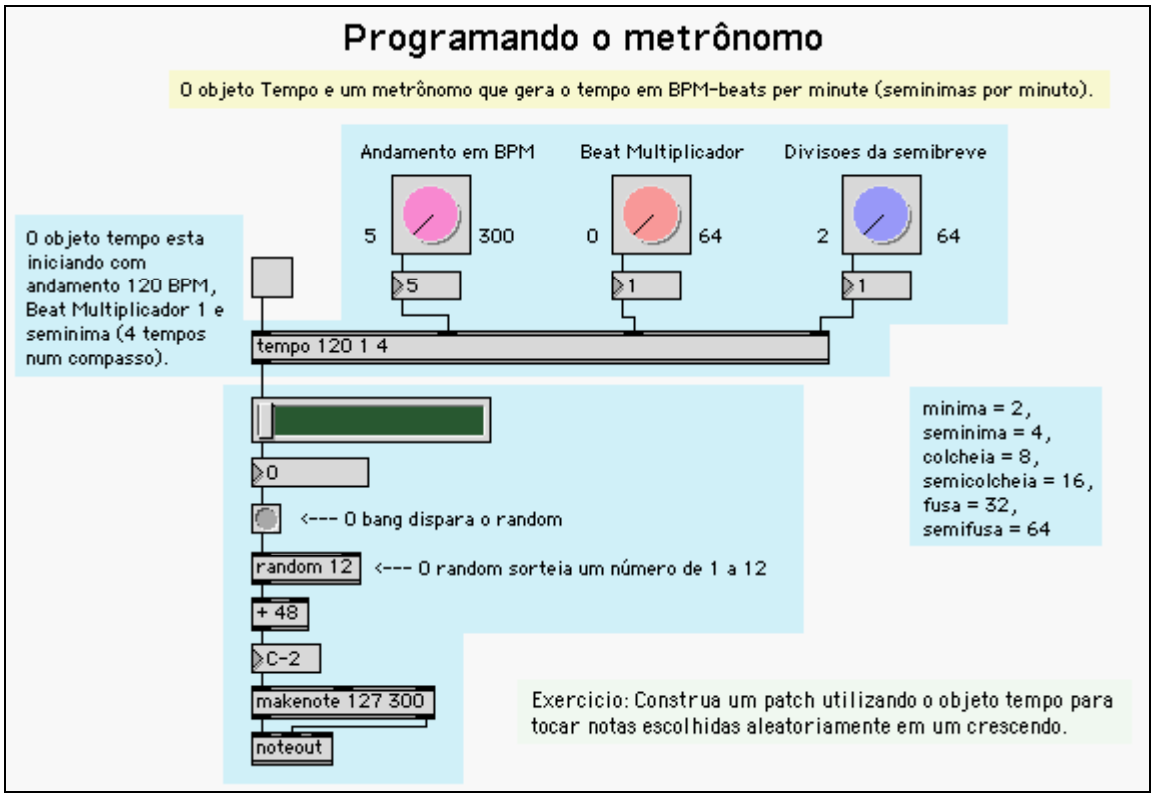


Figura 127 – Programando o metrônomo

Para solucionar esse problema é necessário a utilização do objeto random para a escolha aleatória das notas. A solução da figura 128 mostra também que o valor provindo do slider é multiplicado por 2 para chegar ao valor 128 (fortíssimo).

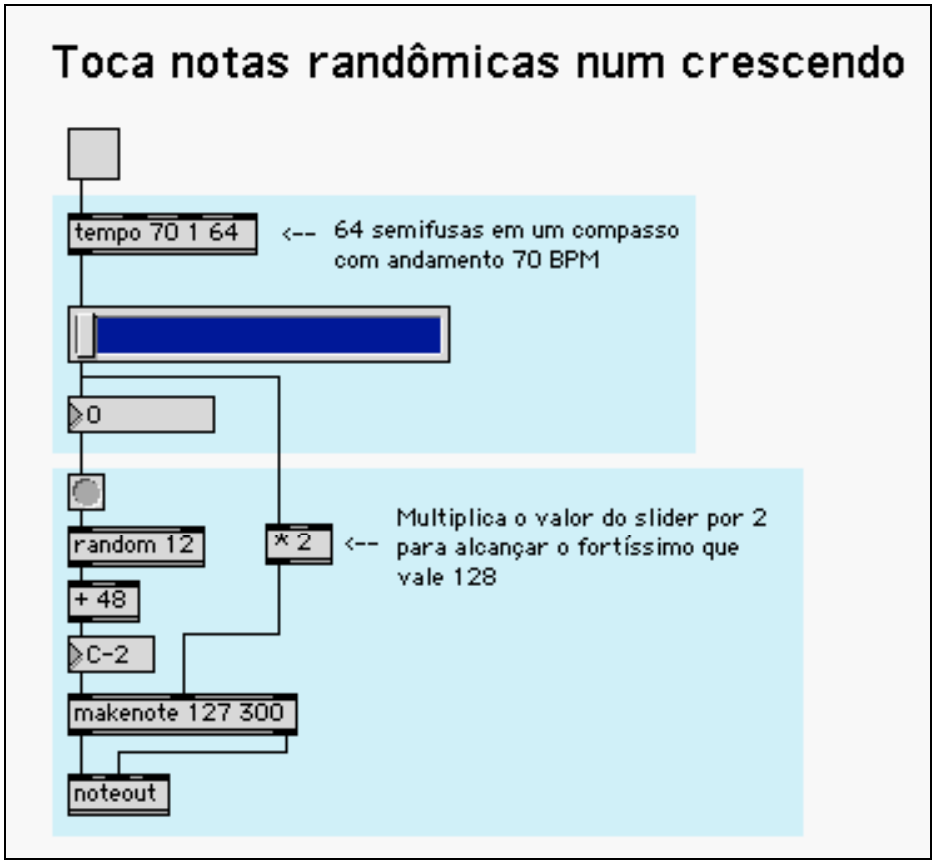


Figura 128 – Executando notas randômicas num crescendo

As linguagens de programação utilizam acumuladores para guardar dados que posteriormente serão utilizados pelo programa. O patch da figura 129 demonstra como o objeto ***Accum*** da linguagem MAX pode armazenar e retornar tanto um valor inteiro como real. O segundo inlet do objeto accum possibilita adições e o terceiro inlet possibilita multiplicações com a utilização de um fator.

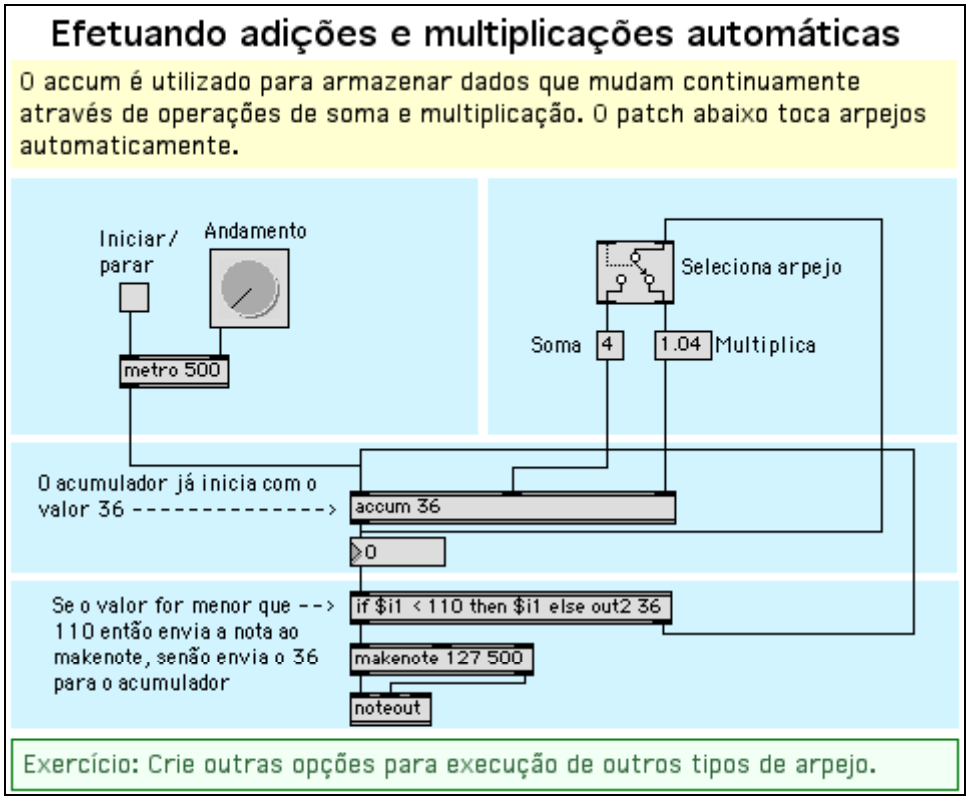


Figura 129 – Acumulando, somando e multiplicando valores

A solução do exercício é apresentada na figura 130. Observe a utilização do gate para possibilitar a escolha de outros fatores para adição e multiplicação. O programa irá gerar alturas diferentes dependendo do fator de adição e multiplicação selecionados.

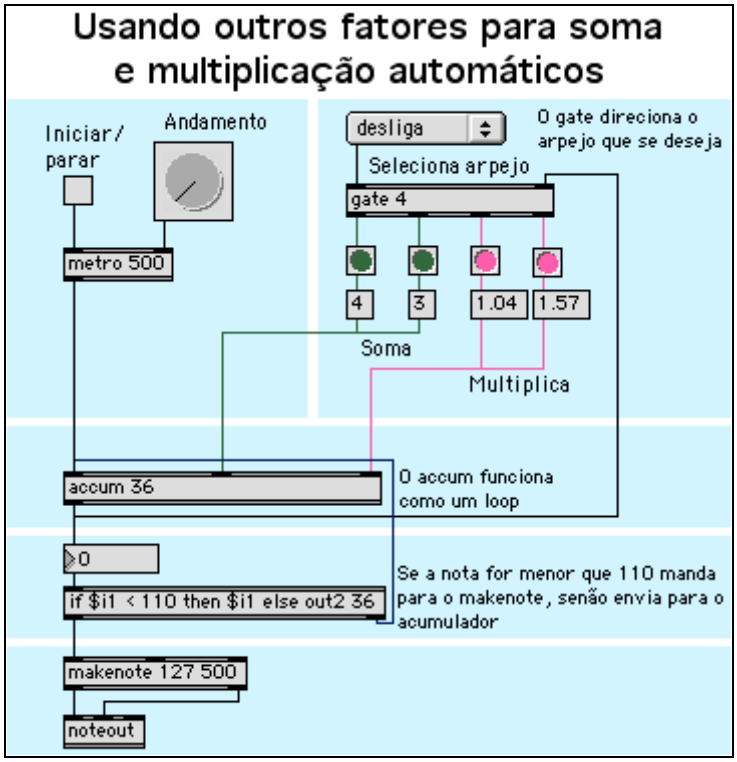


Figura 130 – Usando outros fatores de adição e multiplicação automáticos

7.1.12AULA 6 – Estrutuas de armazenamento

No exemplo 1 da aula 6 temos a utilização dos objetos INT e FLOAT que podem ser entendidos como "caixas" que guardam valores. Enquanto *FLOAT* armazena um número real (com ponto flutuante) o INT armazena um número inteiro. Execute o programa abaixo para entender como utilizar o armazenamento de dados.

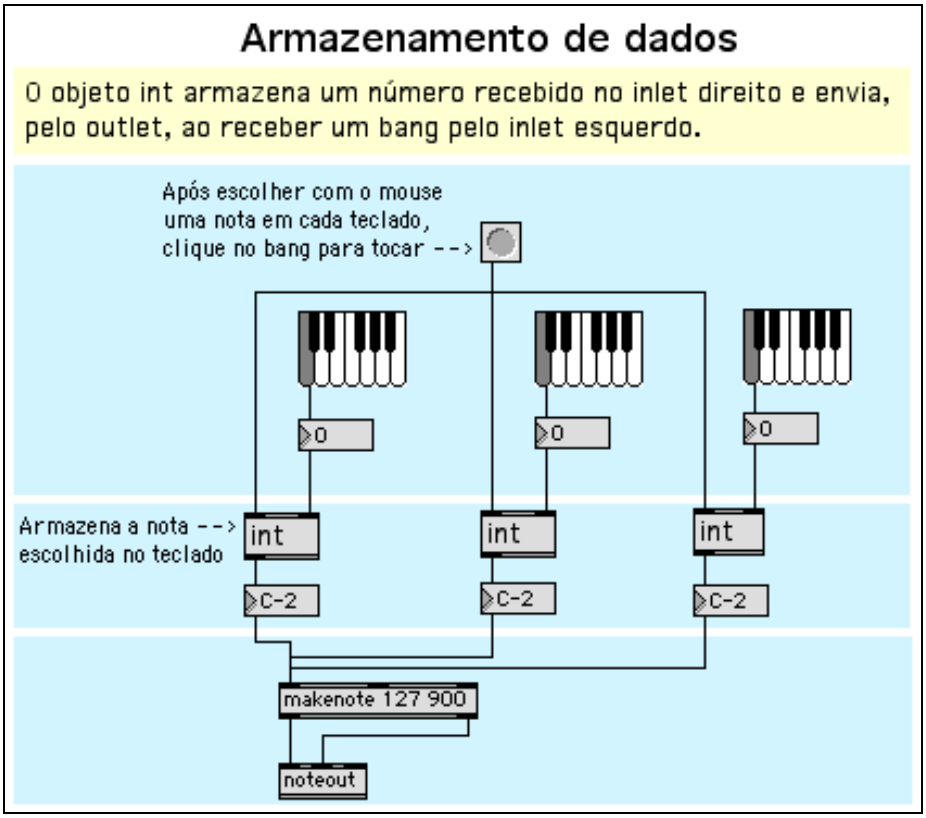


Figura 131 – Armazenamento de dados

Em programação costumamos utilizar estruturas para armazenamento de mais de um valor. No exemplo da figura 132, o objeto ***coll*** armazena qualquer tipo de mensagem, com um número ou símbolo como endereço. Os dados a serem armazenados podem ser recebidos como mensagens na tomada de entrada ou digitados na janela do editor de texto de ***coll***. Quando ***coll*** recebe um endereço em sua tomada, este é enviado para fora através do inlet da direita e a mensagem propriamente dita, armazenada naquele endereço, é enviada através da tomada da esquerda.

O conteúdo de um objeto ***coll*** pode ser armazenado como parte de um *patch* que o contenha ou como um arquivo separado. Um arquivo ***coll*** pode ser carregado em um objeto ***coll*** com a mensagem *read*, ou ainda digitando-se o nome do arquivo como argumento.

A mensagem enviada por ***coll*** pode ser analisada por outros objetos a fim de selecionar itens particulares da estrutura de dados. Além disso, itens individuais de dados podem ser enviados para fora ou alterados por certos comandos na tomada de entrada de ***coll***. Execute o patch da figura 93 e entenda o funcionamento do **coll**, depois resolva o exercício proposto.

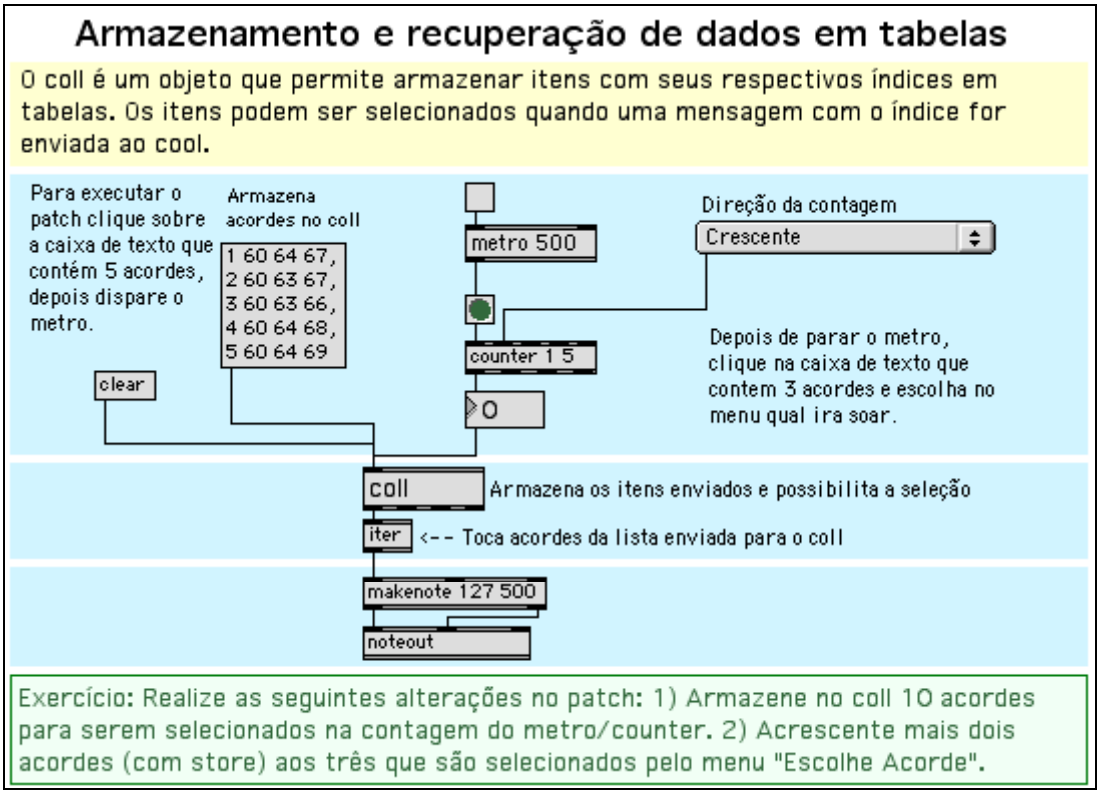


Figura 132 - Armazenamento e recuperação de dados numa tabela

A resposta do exercício é apresentada na figura 133. Verifique as alterações realizadas e compare com a sua solução.

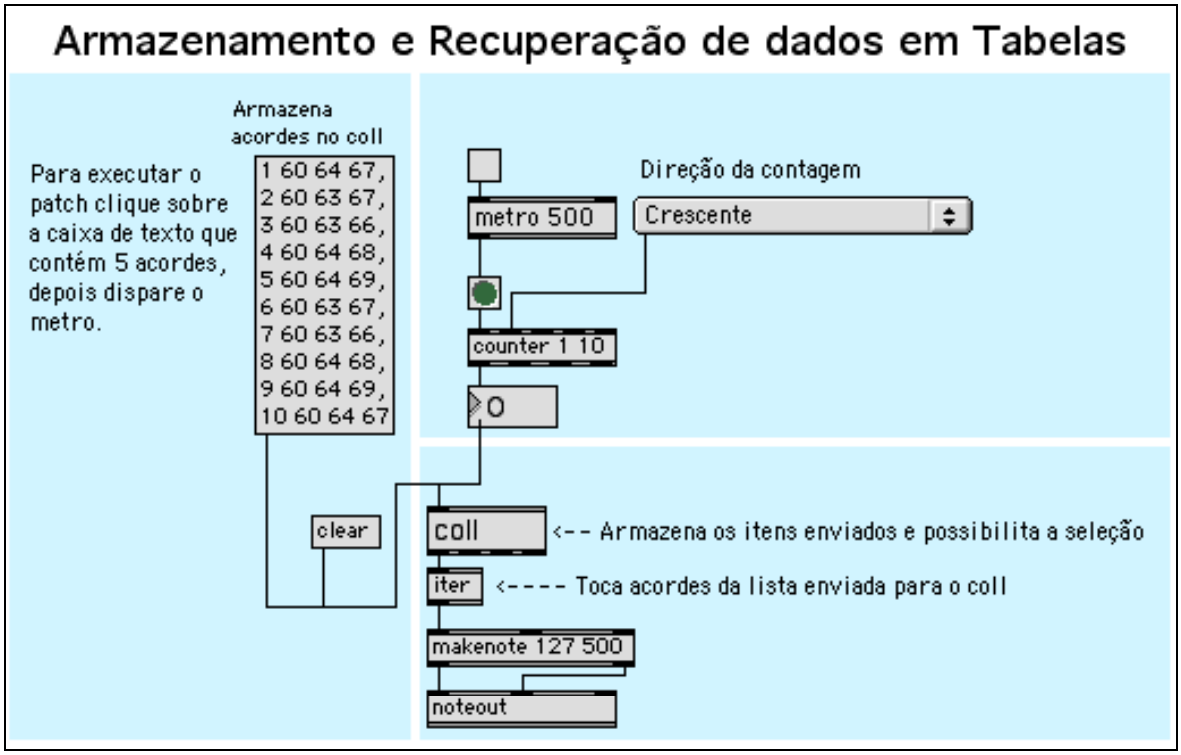


Figura 133 – Armazenamento e Recuperação de dados em Tabelas

O armazenamento de dados em tabelas pode ser exemplificado da figura 134. O objeto ***Table*** armazena e recupera uma matriz indexada de números. Você pode editar e visualizar graficamente os números armazenados dando um clique duplo no objeto ***table***. Os valores em ***table*** ("tabela") são normalmente descartados quando a janela *Patcher* é fechada, mas você pode salvá-los como parte do *patch* ao selecionar ***table***, escolher a opção **Get Info...** do menu do Max e marcar *Save with Patcher*. Você pode também salvar uma tabela como um arquivo separado, podendo então usá-la em um outro *patch*, criando um objeto ***table*** e digitando o nome do arquivo como argumento.

Para abrir uma nova janela *Table*, escolha a opção **Table** a partir do menu *New* ou crie um novo objeto ***table*** em uma janela *Patcher*.

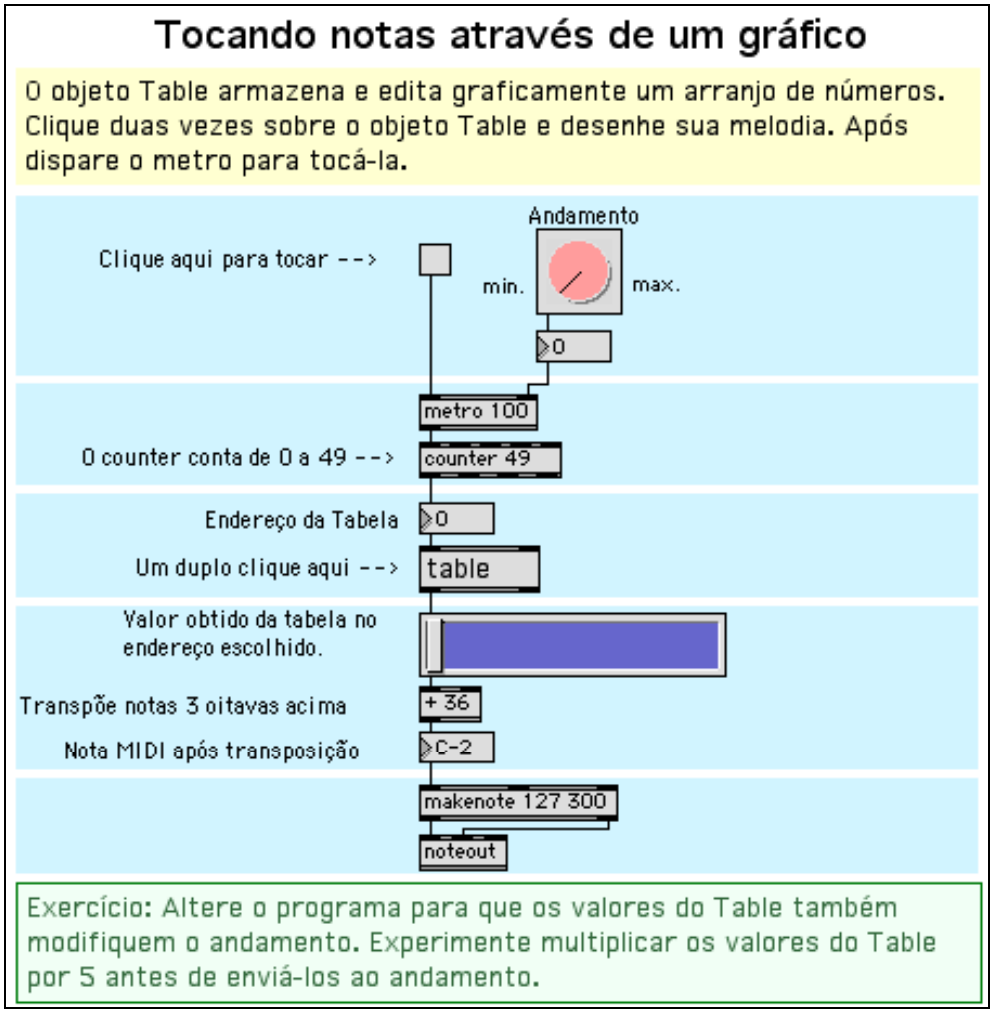


Figura 134 – Programando tabelas gráficas

A solução do exercício é apresentada na figura 135. Observe a alteração em relação ao programa anterior e compare com a sua solução.

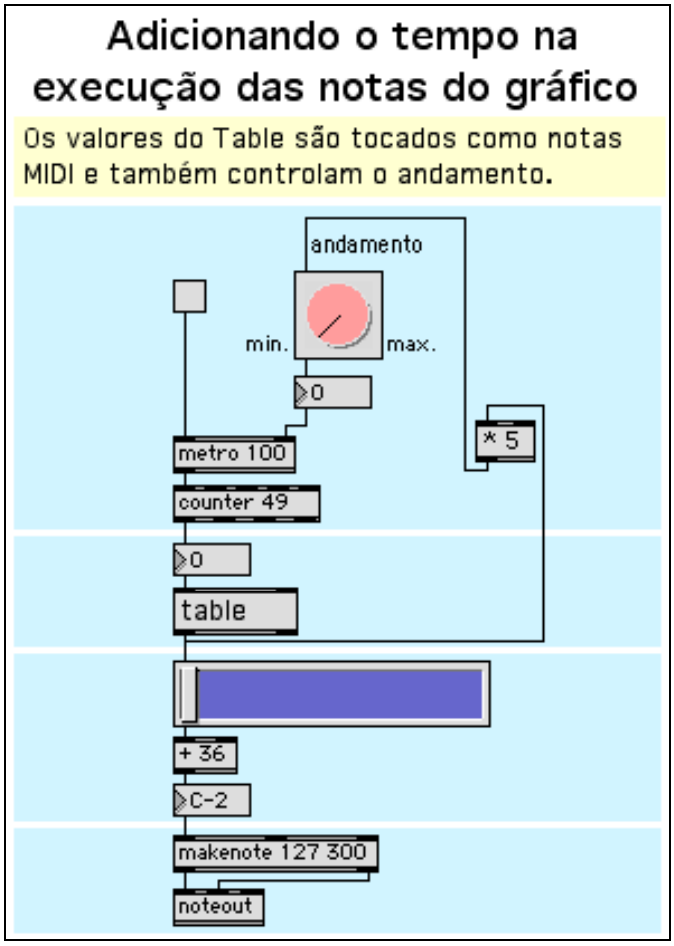


Figura 135 – Adicionando tempo na execução das notas do gráfico

O exemplo da figura 136 apresenta um *patch* que utiliza o objeto ***Preset***. O *Preset* permite que você armazene as configurações de qualquer outro objeto de interface em um determinado momento, para resgatar estas configurações posteriormente. Se o ***preset*** estiver conectado a objetos através de *patch cords,* o armazenamento e resgate das configurações serão feitos *somente* destes objetos. O conteúdo de uma tabela em ***table*** pode também ser memorizado pelo ***preset***, mas o ***table*** *deverá* estar conectado ao ***preset***. O objeto ***preset*** pode armazenar e resgatar até 256 coleções diferentes de configurações de todos os objetos de interface. Utilize o programa modificando as configurações de sliders e tables e armazenando no objeto preset, através da mensagem *store*. Após recupere as configurações, clicando no quadro, conforme está indicado no exemplo da figura 136.

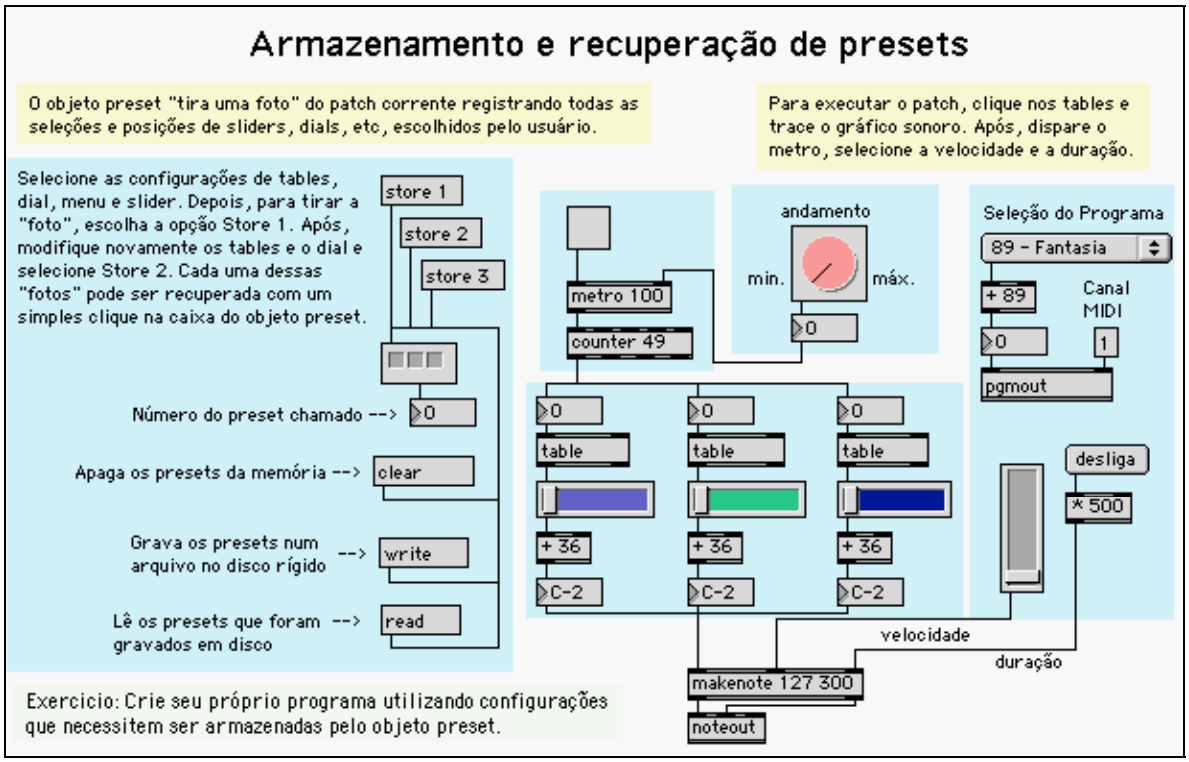


Figura 136 – Armazenamento e recuperação de presets

7.1.13AULA 7 – Listas e tipos de dados

O MAX disponibiliza tipos diferentes de dados. Observe no figura 137 o funcionamento do objeto ***Trigger*** que converte dados MIDI em *bangs*.

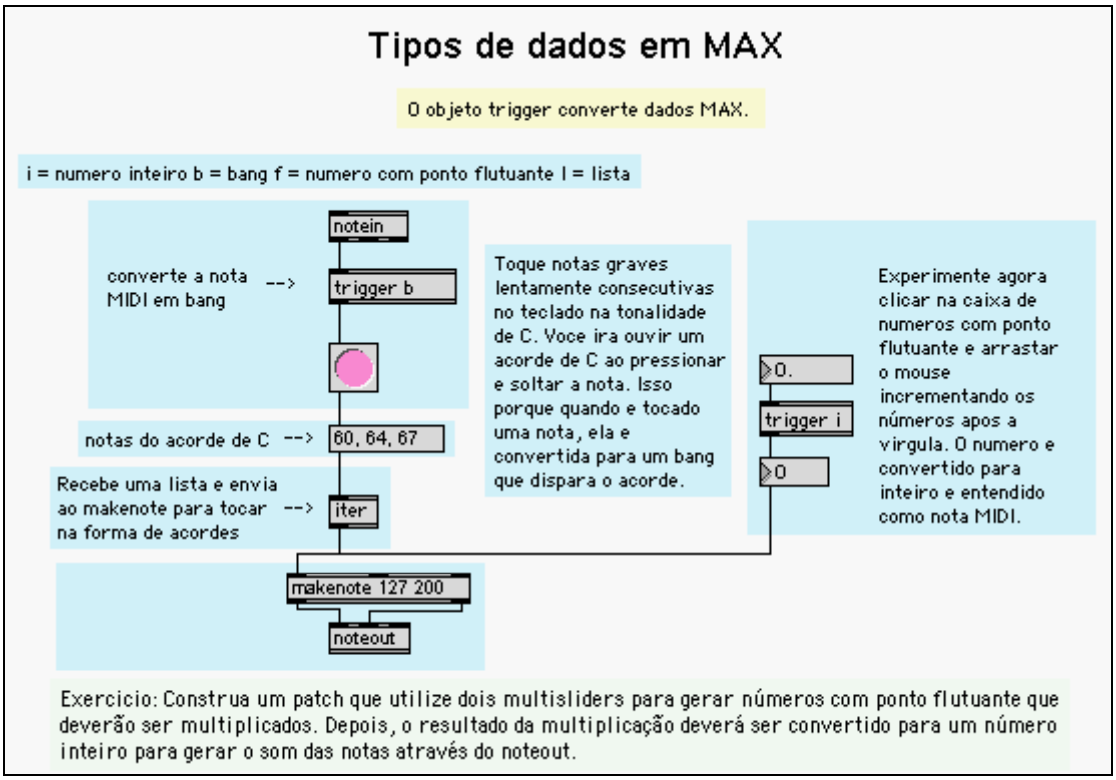


Figura 137 – Conversão de dados em Max

A solução do exercício é apresentada na figura 138. Nesta solução um número real obtido a partir da multiplicação de *multisliders* é convertido para um número inteiro para poder ser tocado como nota MIDI.

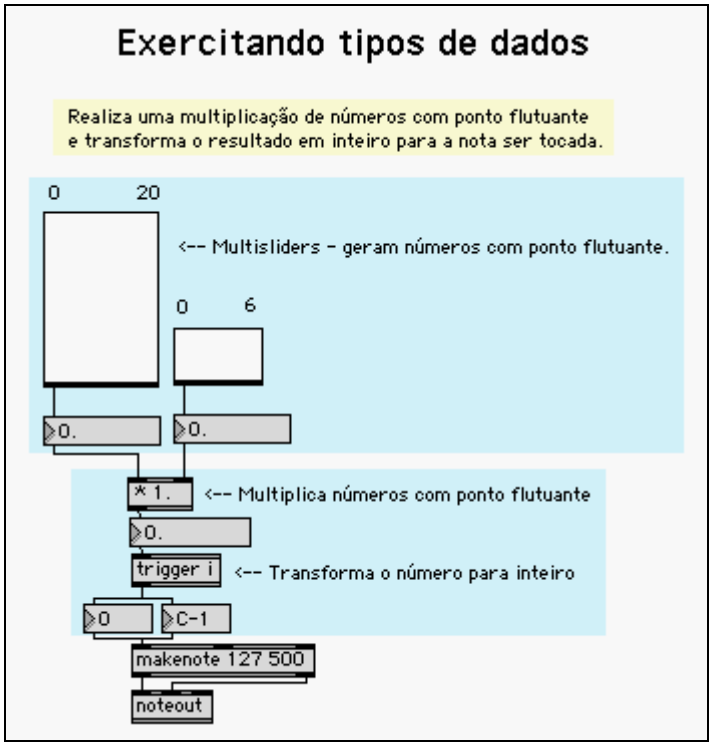


Figura 138 - Resposta do exercício sobre tipos de dados

O MAX possui funções para depuração de programas. É possivel ver a execução do *patch* passo-a-passo para entender construções mais complexas, e até mesmo achar erros de programação. Procure entender o fluxo dos dados do programa da figura 139 utilizando o depurador do MAX. Repare que o ***Stripnote*** filtra mensagens de desliga-nota, deixando passar apenas mensagens de liga-nota. Útil quando se quer obter dados apenas quando uma tecla do teclado é pressionada, mas não quando a tecla é solta. Nesse exemplo é calculado o pitch class e também o intervalo entre duas notas tocadas.

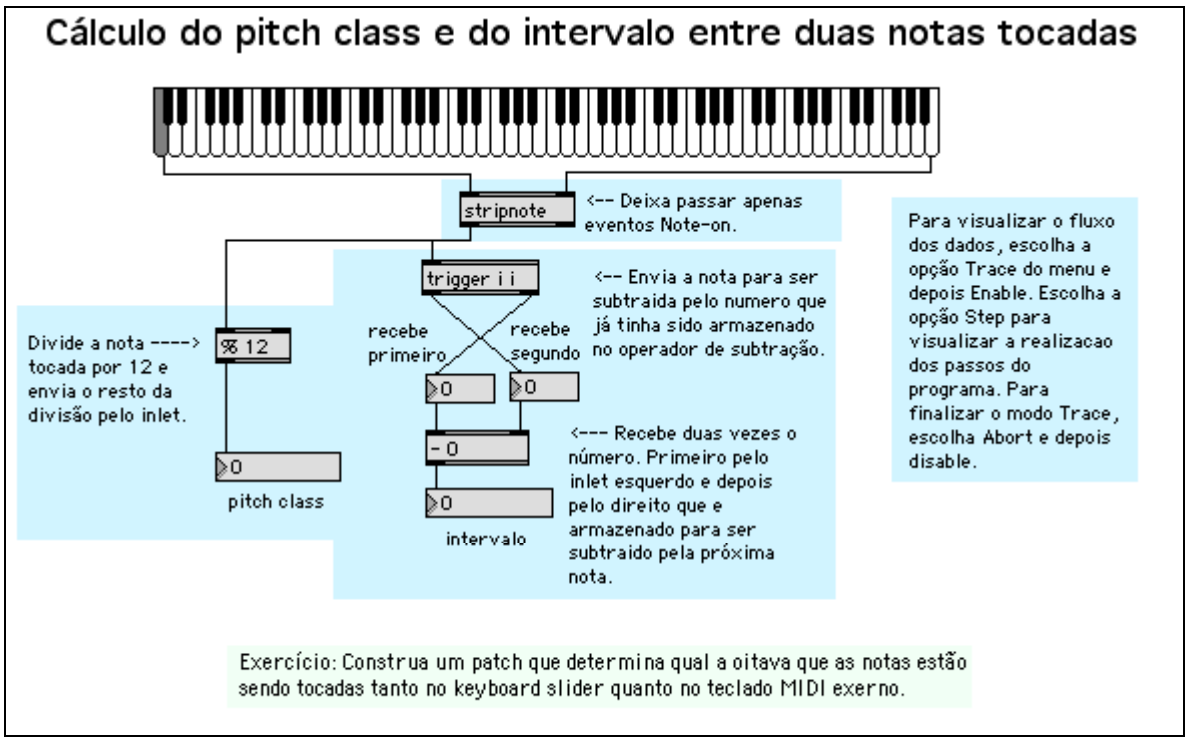


Figura 139 – Analise de intervalos e funcionamento do depurador do MAX

A solução do exercício é apresentada na figura 140. A nota MIDI provinda do kslider é dividida por 12 para a obtenção do número da oitava e subtrai de um para compensar a diferença entre a numeração técnica e a numeração musical.

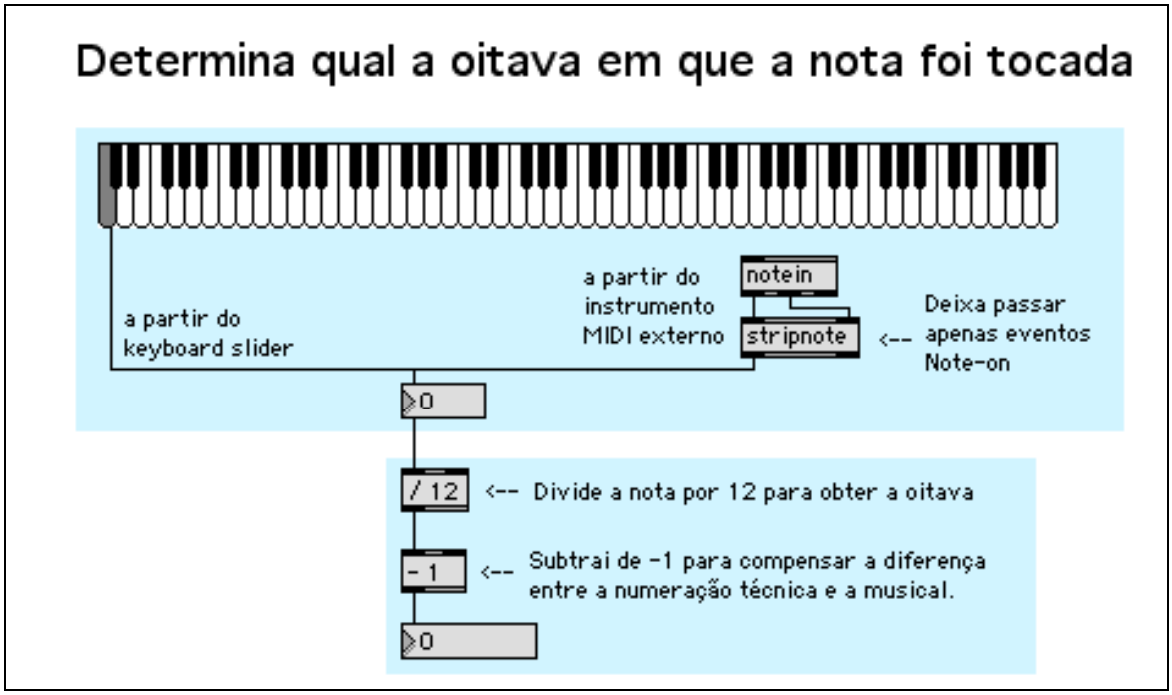


Figura 140 – Determinar a oitava em que a nota foi tocada

A utilização de listas é muito comum em programação de computadores. O objeto ***pack*** combina os números recebidos, em cada *inlet* em uma única lista de números. Ao receber um número ou um bang, envia o conjunto de números recebidos (como uma lista) pelo *outlet* da esquerda. Observe o exemplo de utilização de listas no programa da figura 141.

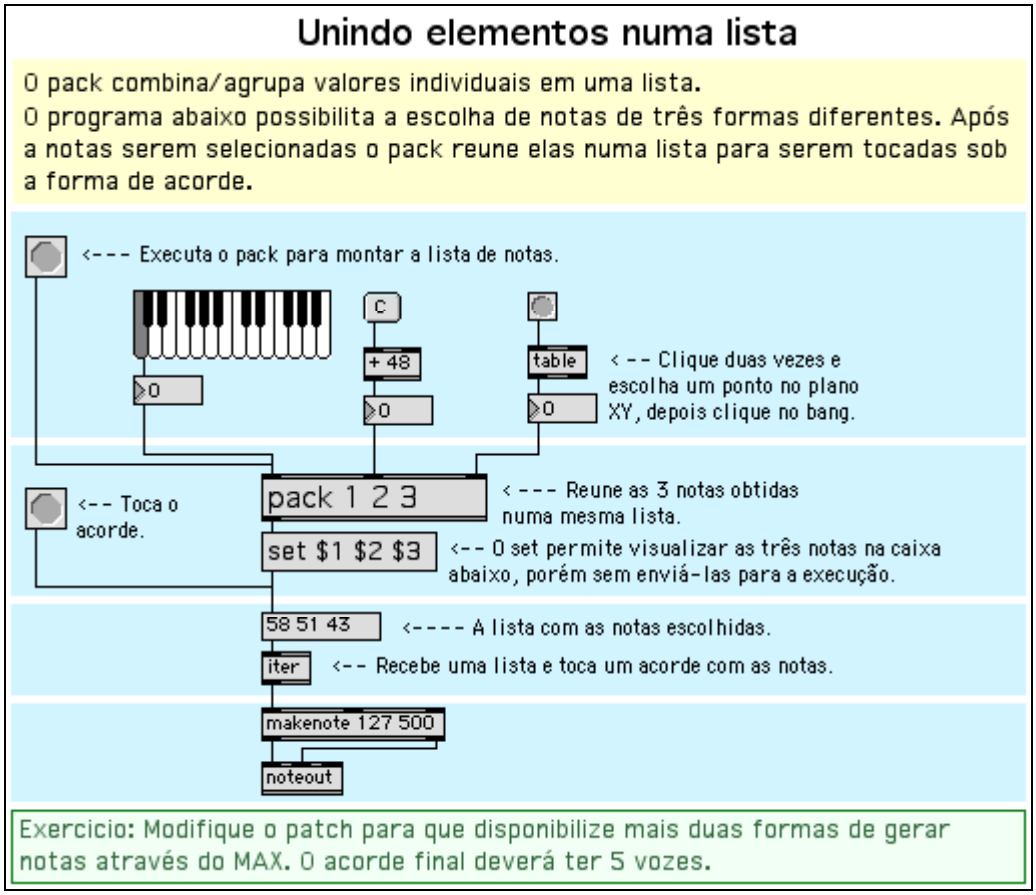


Figura 141 – Unindo elementos numa lista

A solução do exercício é apresentada na figura 142. Observe como o objeto pack e a mensagem set foram utilizados nessa solução.

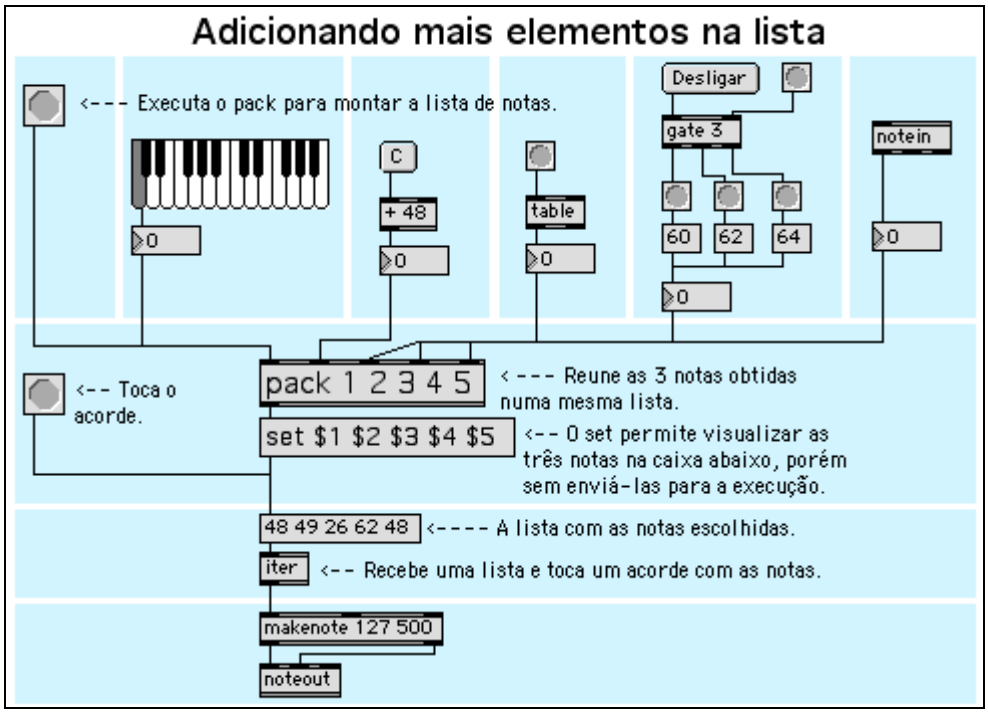


Figura 142 – Adicionando mais elementos na lista

O ***Unpack*** divide uma lista de números em seus elementos individuais (itens) e os envia para fora por diferentes tomadas de saída, da direita para a esquerda (ao contrário de ***iter***, que os envia para uma única tomada de saída).

Na figura 143 é apresentado um programa onde o usuário deve traçar com o mouse no quadro. As coordenadas de x e y são enviadas na forma de uma lista com dois elementos para o objeto **unpack**. O objeto **unpack** separa o primeiro elemento do segundo da lista e envia para o **makenote**. O **inlet** da esquerda do **makenote** recebe a nota e o **inlet** da direita recebe a duração. Estude o programa exemplo e tente resolver o exercício proposto.

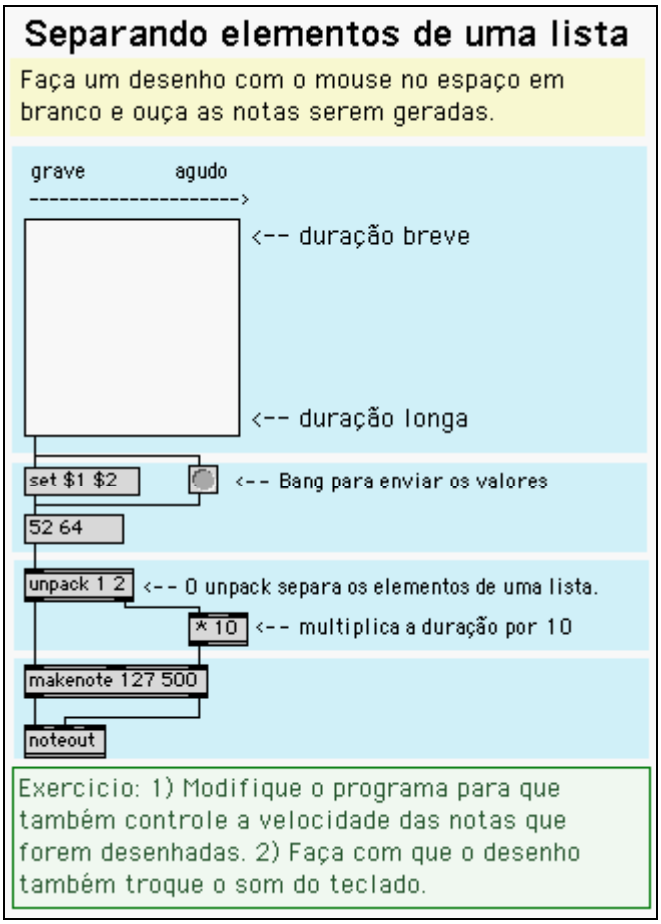


Figura 143 – Separando elementos de uma lista

7.1.14AULA 8 – Atrasos e subprogramas

Na figura 144 o exemplo demonstra como é possível criar sub-programas em MAX através do emprego do objeto **patcher**. Um sub-programa irá funcionar como um objeto MAX, com a diferença de que este é construído para uma função específica no *patch*. Este exemplo de sub-programa fornece, aleatoriamente, apenas números pares. O objeto **Random** devolve um número randômico (aleatório) e foi utilizado no sub-programa. Estude o exemplo e resolva os exercícios.

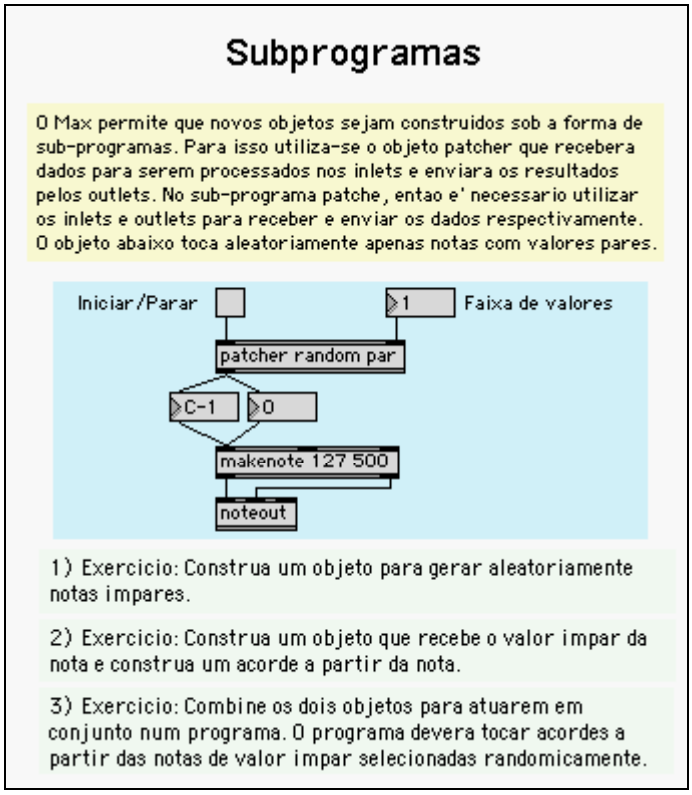


Figura 144 – Subprogramas

O objeto **Patcher** cria um *subpatch* dentro de um *patch*. O *subpatch* é salvo como parte do documento que contém o objeto ***patcher***. Se a janela do *subpatch* estiver aberta quando o *patch* for salvo, ela será aberta automaticamente na próxima vez que o programa for aberto. Pode-se trocar mensagens entre o *patch* principal e o *subpatch* com os objetos ***send*** e ***receive***, ou mesmo com os objetos ***inlet*** e ***outlet***. Quando objetos ***inlet*** e ***outlet*** são colocados em um *subpatch*, tomadas de entrada e de saída são automaticamente criadas no objeto ***patcher***. Na figura 145 é possível observar a ligação dos objetos do patcher *random par*.

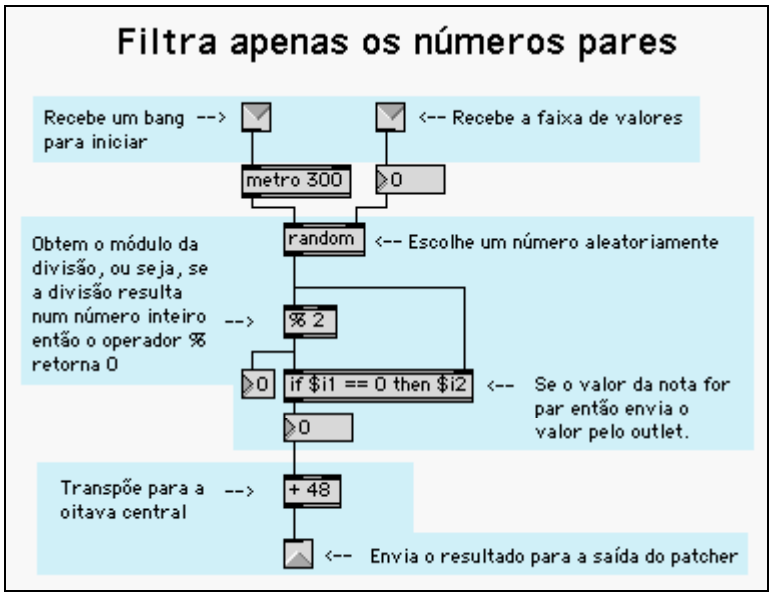


Figura 145 - Subpatch random par

O exercício da figura 145 sugere uma solução modular pois as soluções para os exercícios 1 e 2 podem ser encapsuladas em subprogramas. Um exemplo disso pode ser visto na figura 146 onde os subprogramas "random impar" e "acordes" são utilizados sob a forma de subprogramas. Portanto a resposta do exercício 3 é formada pela resposta dos exercícios 1 e 2.

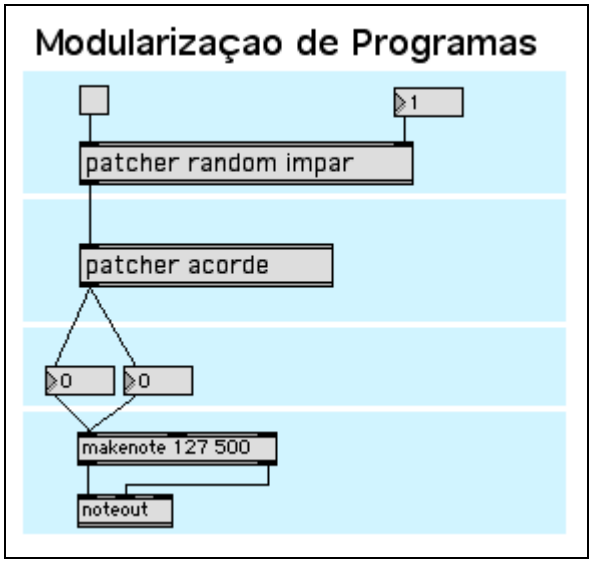


Figura 146 – Modularização de Programas

Para criar o subprograma “random impar” é aconselhável que o usuário siga a mesma lógica utilizada no subprograma “random par”. A figura 147 mostra a solução do exercício. Nela, apenas o comando condicional foi alterado para testar o resultado da divisão.

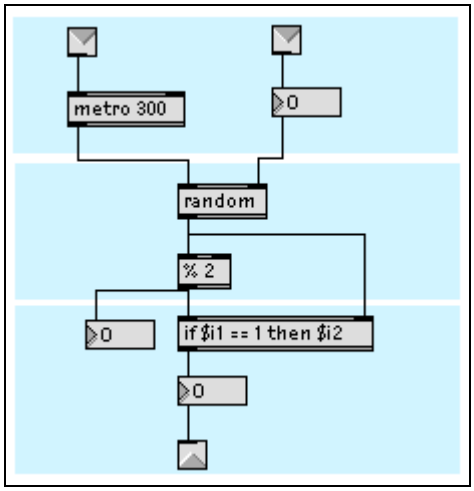


Figura 147 – Subpatch “random impar”

O subprograma “acordes” pode ser feito conforme a figura 148. Observe que são utilizados os valores das notas MIDI. Os valores são somados para compor o acorde.

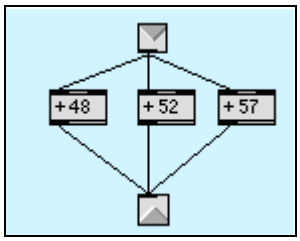


Figura 148 – Subpatch “acorde”

Na figura 149 podemos observar como fazer um patch para tocar diferentes rítmos de bateria. O patcher é composto por sub-patchers que estão programados com ritmos específicos.

O objeto ***Delay*** faz com que um único *bang* possa ser atrasado por uma quantidade específica de tempo. Se um segundo *bang* for recebido enquanto o primeiro estiver sendo atrasado, o primeiro bang é esquecido e o segundo é atrasado.

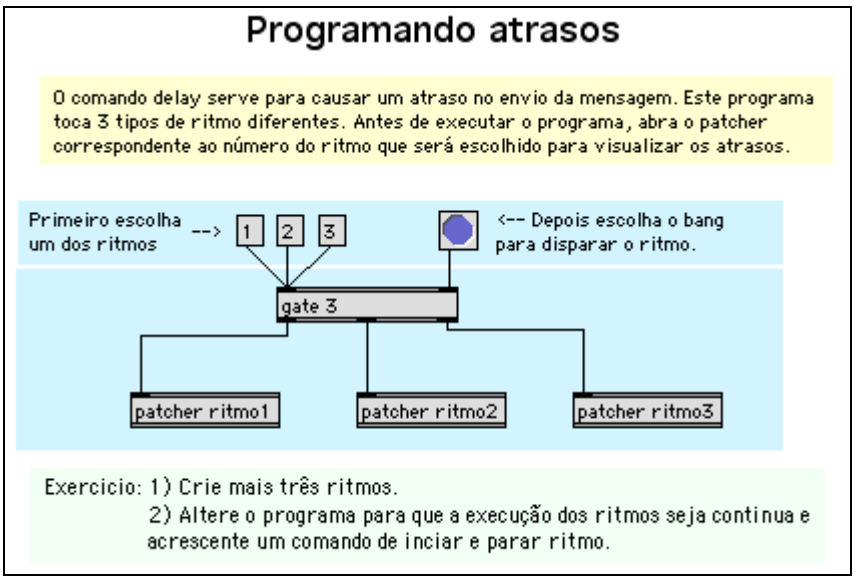


Figura 149 – Programando atrasos

As figuras 150, 151 e 152 apresentam os subprogramas (*subpatches)* ritmo1, ritmo2 e ritmo3 respectivamente.

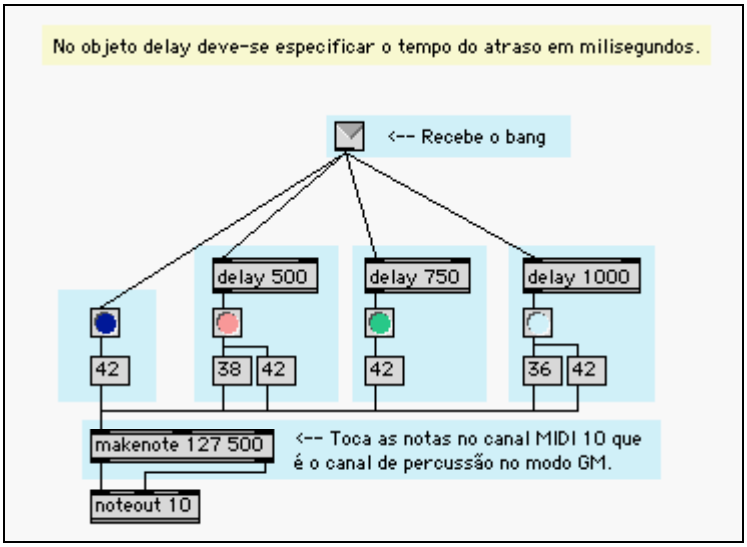


Figura 150 – Subprograma Ritmo 1

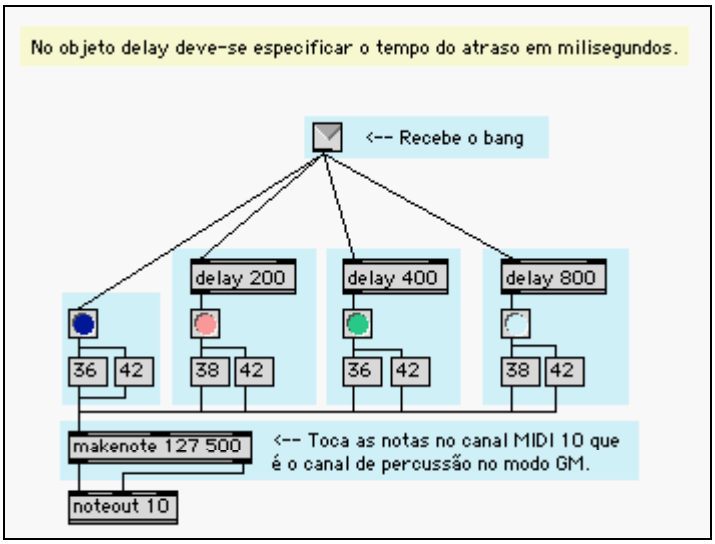


Figura 151 - Subprograma Ritmo 2

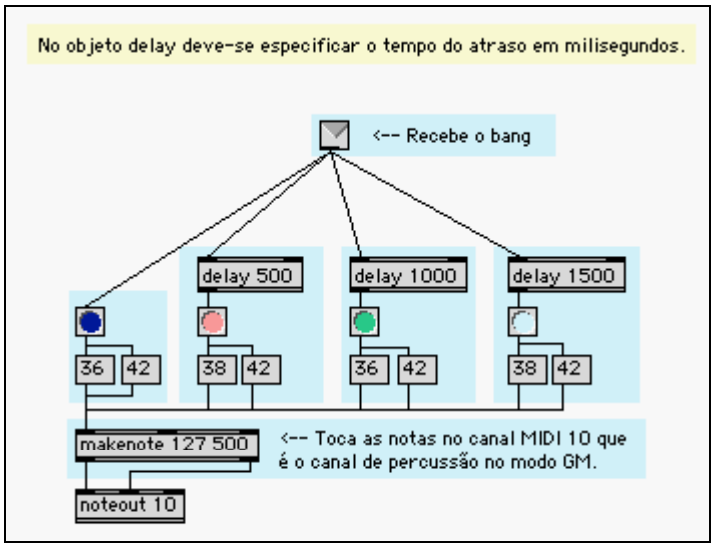


Figura 152 – Subprograma Ritmo 3

Uma possível solução para o exercício proposto é apresentado na figura 153. Observe que, no programa principal, o ritmo deve primeiro ser selecionado para depois ser acionado. Quando um dos ritmos é selecionado, a execução do programa passa para o *subpatch*.

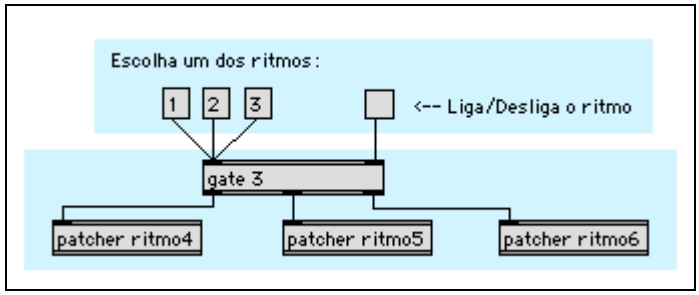


Figura 153 – Resposta do exercício sobre programação de ritmos

Os novos ritmos programados utilizam o metro para automatizar sua execução. As figuras 154, 155 e 156 apresentam os *subpatchers* utilizados no *patcher* principal da figura 63.

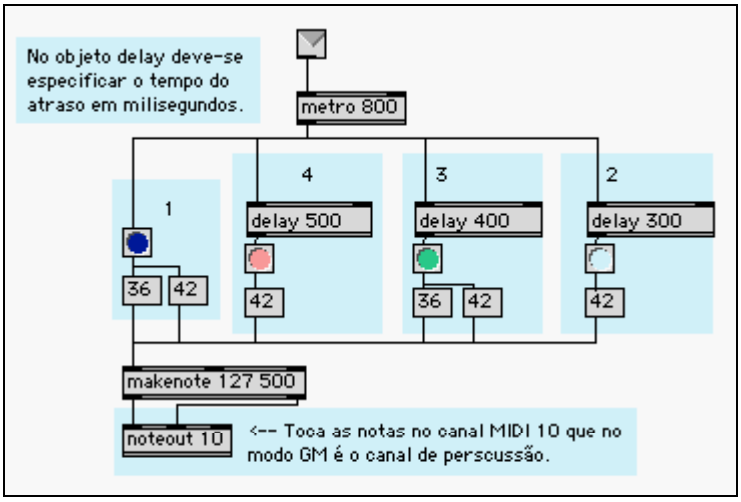


Figura 154 – Subprograma Ritmo 4

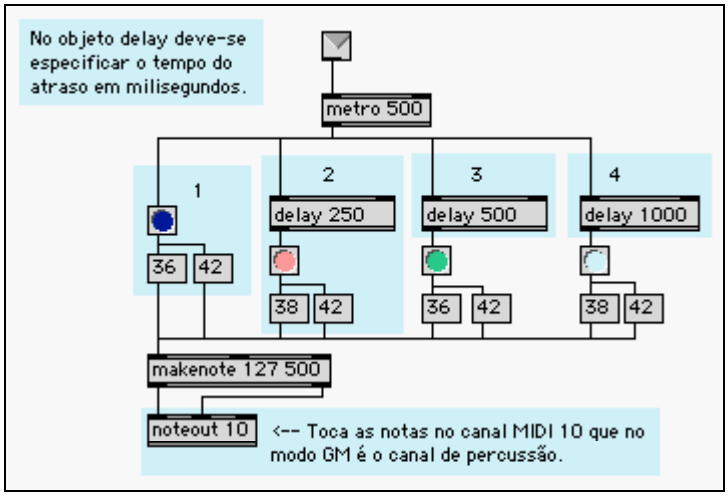


Figura 155 – Subprograma Ritmo 5

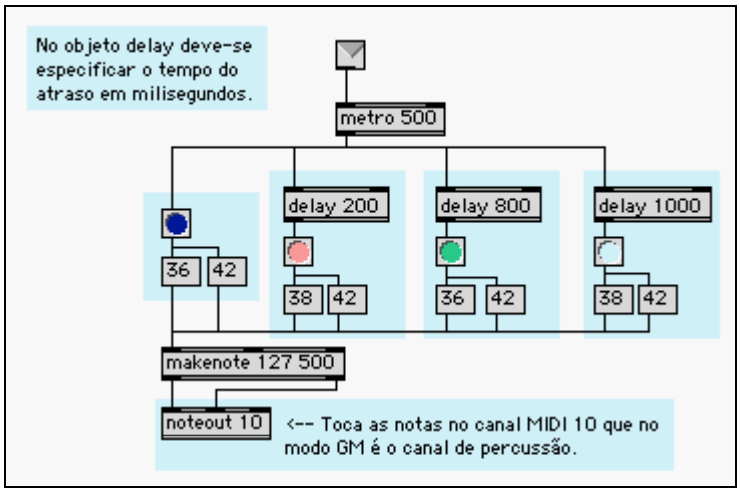


Figura 156 – Subprograma Ritmo 6

7.2 MÓDULO DE PROGRAMAÇÃO DE COMUNICAÇÃO ENTRE INSTRUMENTOS ELETRÔNICOS E COMPUTADORES

Os exemplos e exercícios práticos, que constituem o Módulo de Programação de Comunicação entre Instrumentos Eletrônicos e Computadores, foram implementados com base na obra de [ZIC 88] e [PUC 90].

O Max permite o controle do equipamento MIDI de maneira flexível, sendo possível criar aplicações para composição e improvisação musical, acompanhamento automático, envio de comandos para sintetizadores, modificação de *presets* ou qualquer coisa que seja possível criar via MIDI. Como o Max transforma todas as informações de controle em uma simples cadeia de números, você pode modificar praticamente qualquer parâmetro.

7.2.1 Manipulação MIDI

O Max recebe dados de equipamentos MIDI através do comando *midiin* e envia dados MIDI através do comando *midiout*. Após receber dados via *midiin*, o Max traduz todos os dados essenciais das mensagens MIDI e as apresenta para o usuário como números inteiros simples, que são usados em objetos MIDI. Estes objetos manipulam entradas e saídas do MAX e separam tipos específicos de dados da lista inteira de dados MIDI, tais como a nota ou o *pitch bend*. Dessa forma é mais simples manipular os dados e utilizar algoritmos para criar música. Após o processamento estar completo, Max recompõe a saída final na forma de mensagens MIDI completas para enviá-las a um dispositivo MIDI que irá interpretar as mensagens binárias. Os objetos MIDI do Max também podem especificar em qual canal MIDI será enviada a mensagem, colocando o número do canal após o comando *noteout*. Maiores informações sobre manipulação de dados MIDI através do MAX podem ser encontradas em [ZIC 88].

7.2.2 AULA 1 – Enviando e recebendo mensagens MIDI

Os exemplos da aula 1 englobam a programação em MAX para a troca de mensagens MIDI de voz entre o computador e o instrumento MIDI. Portanto, todos os exemplos dessa aula utilizam os objetos que realizam manipulação das mensagens MIDI. Para um melhor entendimento, revise os conceitos de MIDI antes de realizar os exercícios de programação.

A figura 157os mostra como, através do MAX, podemos enviar e receber mensagens nos 16 canais MIDI. É possível selecionar o canal MIDI desejado. Utilize o programa e depois tente resolver o exercício proposto.

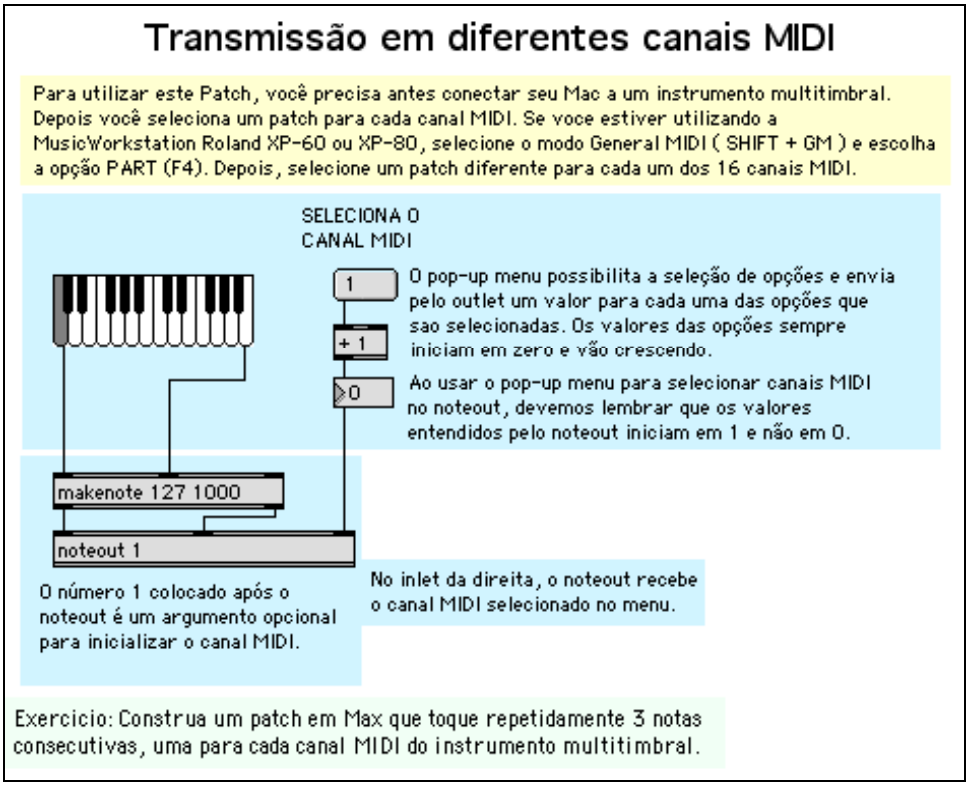


Figura 157 – Selecionando o canal MIDI

Para solucionar este exercício você precisará lembrar do comando de decisão (*if*), do comando de seleção (*gate*) e do comando de repetição (*metro*). Você tem mais de uma forma de resolver esse exercício. Após chegar a uma solução, procure compará-la com a solução apresentada na resposta do exercício da figura 158.

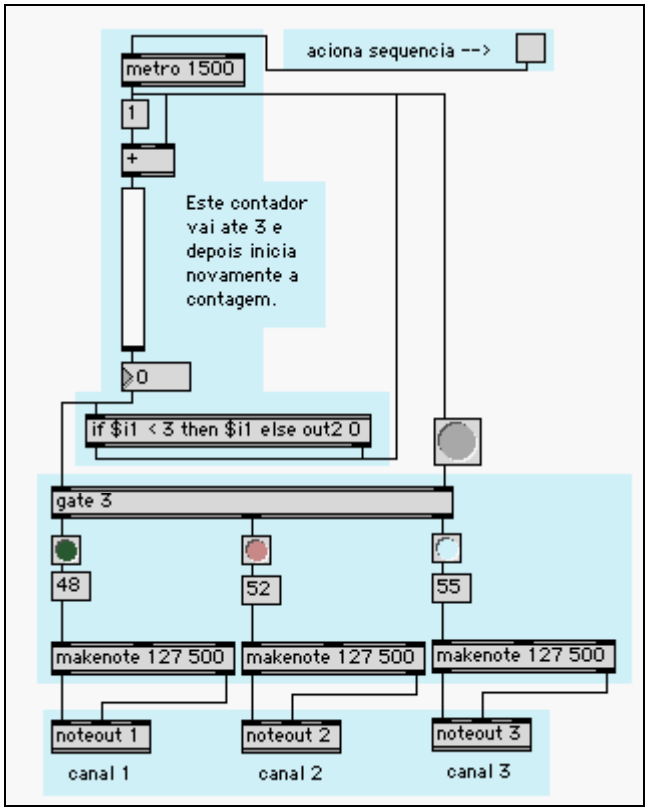


Figura 158 – Tocando notas repetidamente em diferentes canais MIDI

O exemplo da figura 159 mostra como é possível programar *patches* para que o computador controle a mudança de *presets* em seu instrumento MIDI. Para isso, deve ser utilizado o objeto ***pgmout*** que transmite mensagens de mudança de programa. Utilize este objeto sempre para mudar o *preset* de um sintetizador.

Ainda neste exemplo é utilizado o objeto ***menu*****.** O objeto ***menu*** é um menu *pop-up*, no qual os itens de menu (comandos) podem ser mensagens de qualquer tipo. ***Menu*** pode ser usado para a seleção de comandos por mouse e/ou para exibir mensagens. Quando um comando de ***menu*** é selecionado, seja com o mouse ou através de um número de item de menu recebido pelo *inlet*, o ***menu*** exibe o comando, envia a mensagem armazenada através do *outlet* direito e envia o número do item pelo *outlet* esquerdo.

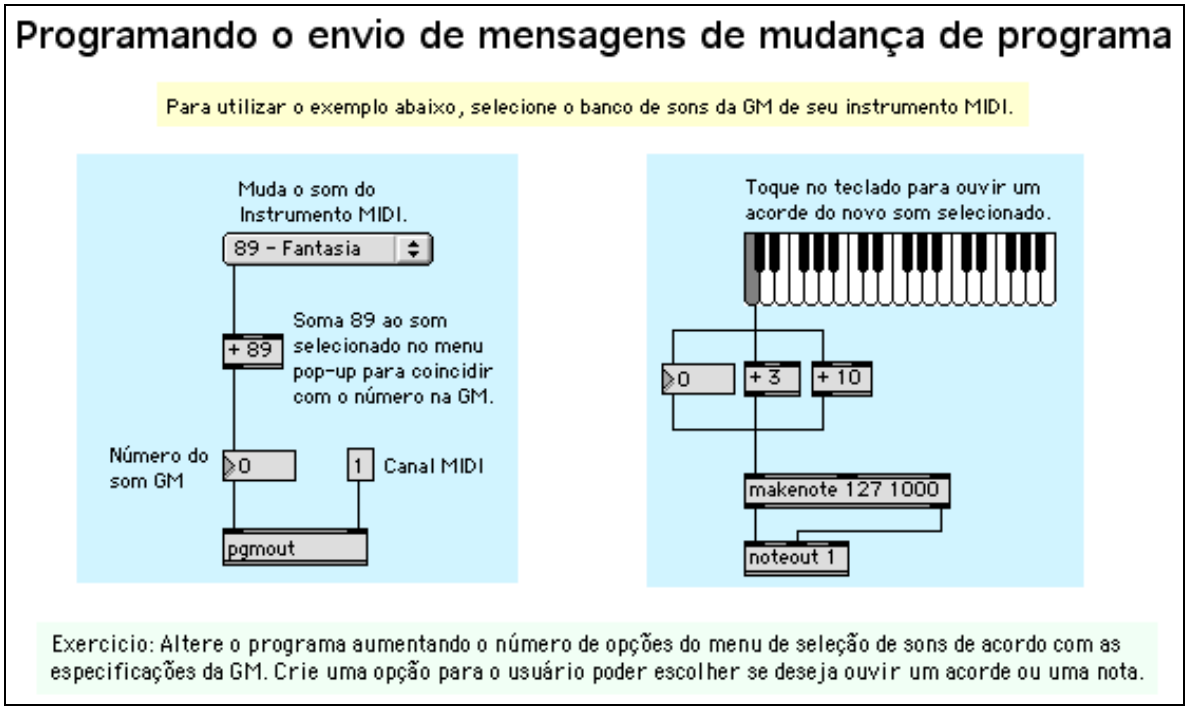


Figura 159 –Envio de mensagens *Program Change* para o instrumento MIDI

Na solução do exercício, pode ser utilizado o objeto *Gswitch* para alternar o envio da nota ou do acorde. Veja na figura 160 como os intervalos para formar o acorde são programados. Nesta resposta o objeto *pgmout* continua disponibilizando a mudança de programa através da seleção do menu.

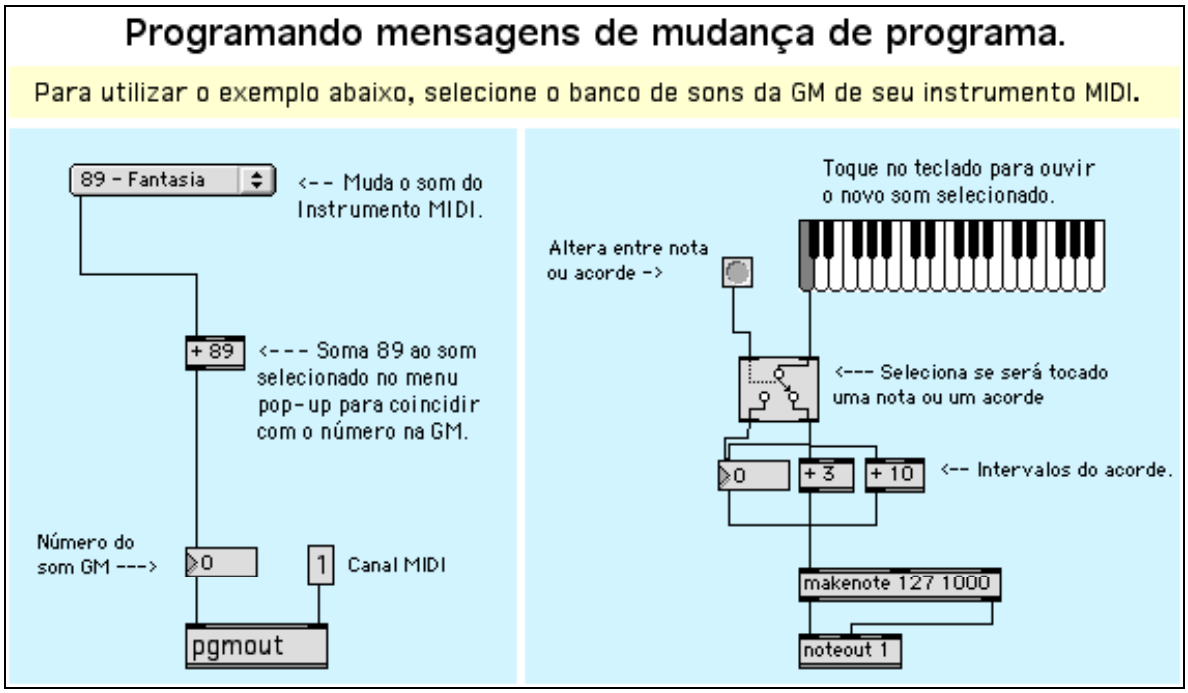


Figura 160 – Programando mensagens de mudança de programa

Além do objeto ***pgmout,*** que transmite mensagens MIDI de *Program Change*, o MAX disponibiliza o objeto ***pgmin***. Você pode utilizar este objeto para receber, do instrumento MIDI externo, as mensagens de *Program Change*. Utilize o programa exemplo da figura 161 e perceba como isso ocorre.

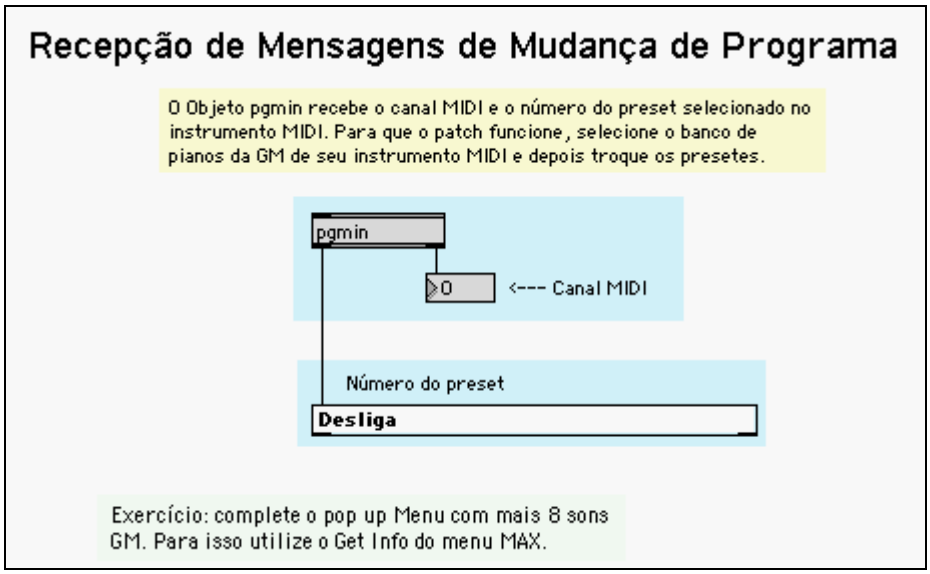


Figura 161 –Exemplo de recebimento de mensagem MIDI *Program change*

Existem várias mensagens MIDI de controle que podem ser selecionadas e alteradas através de programação. Tudo depende do conjunto de mensagens que o instrumento MIDI suporta. Isso pode ser verificado no *MIDI Implementation Chart* do manual do instrumento. Na figura 162, podemos acionar e controlar o portamento e o volume do teclado MIDI. O objeto ***clout*** envia a mensagem MIDI para troca de controle. Repare que cada *inlet* do ***clout*** recebe uma informação diferente. O primeiro *inlet* recebe a variação do controle. Se for controle contínuo, todos os valores do *dial* de 0 até 127 serão utilizados pelo *clout*. Se for controle liga e desliga, apenas os valores 0 (desliga) e 127 (liga) serão reconhecidos pelo ***clout***. O segundo *inlet* recebe o tipo do controle contínuo (se é portamento ou volume principal). O terceiro *inlet* recebe o valor do *Control Change*. Utilize o programa explorando todas as opções até perceber as mudanças no som e, logo após, resolva o exercício.

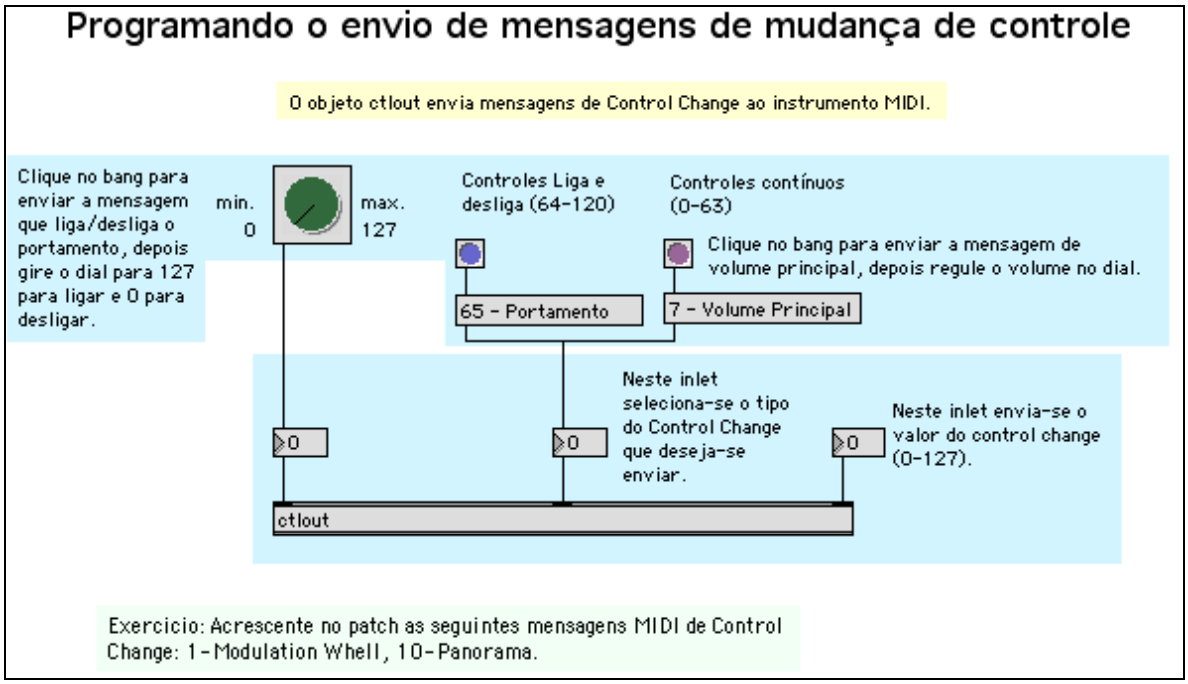


Figura 162 – Programando o envio de mensagens de mudança de controle

A figura 163 apresenta a resposta para o exercício. Observe as opções para *Modulation Whell* e Panorama. O *clout* foi programado para receber 4 tipos de mensagens de *control change*. Compare a resposta com a sua solução.

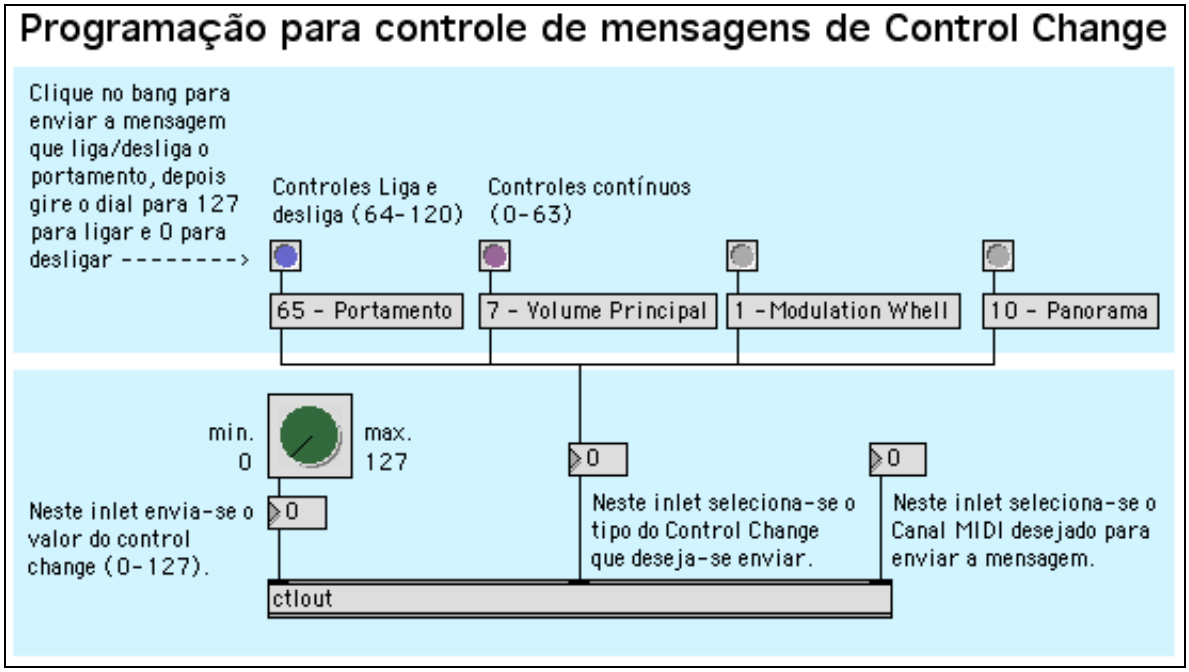


Figura 163 – Programação para controle de mensagens de *Control Change*

Outras mensagens MIDI também estão implementadas em Max sob a forma de objetos. Por exemplo, o ***bendin*** e o ***bendout*** recebem e enviam dados do *pitch bend* de um teclado. No entanto, para acessar todos os valores do *pitch bend*, utilizamos o objeto ***xbendout***. Utilize o exemplo da figura 164 e resolva o exercício.

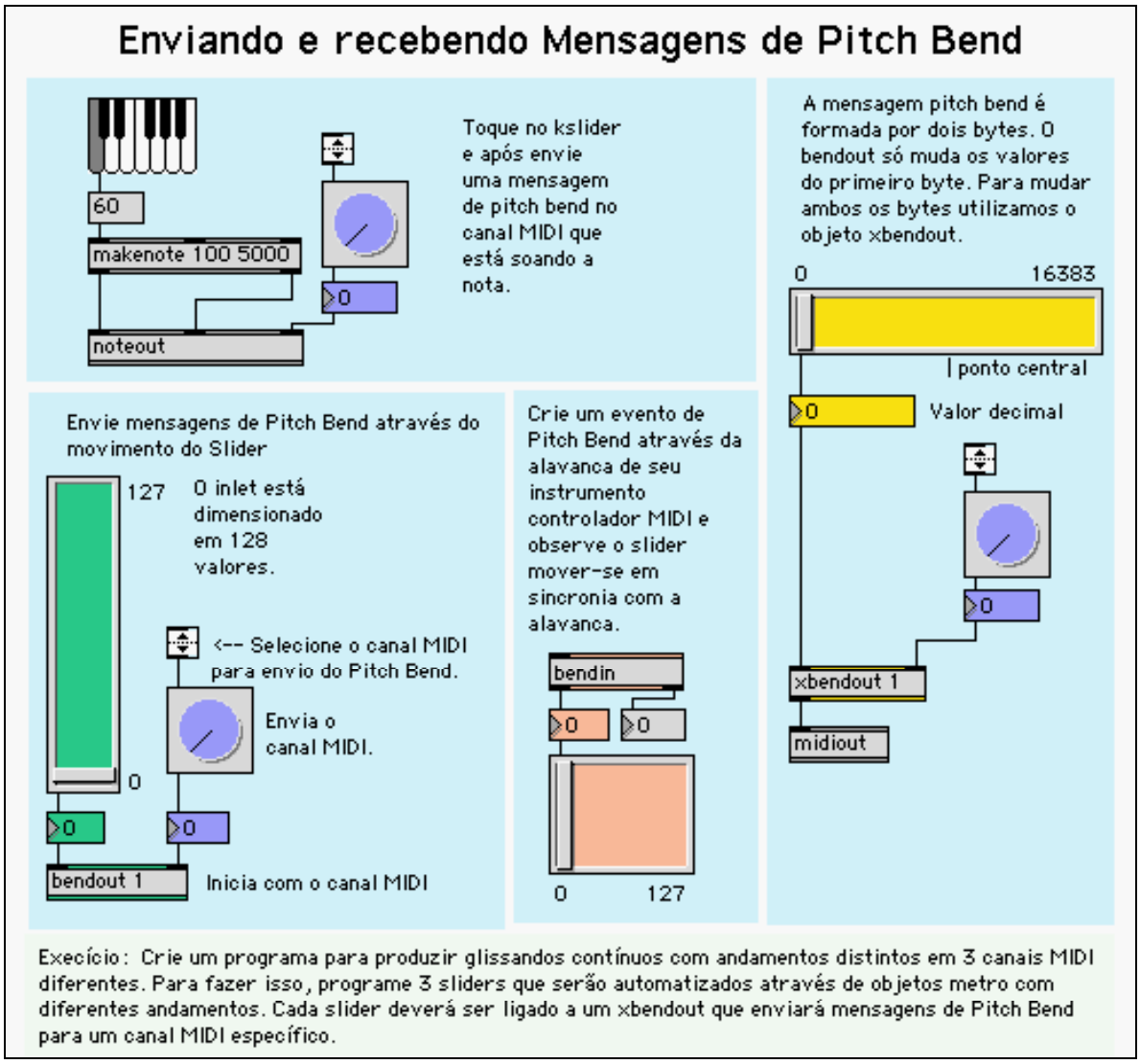


Figura 164 – Exercício com *pitch bender*

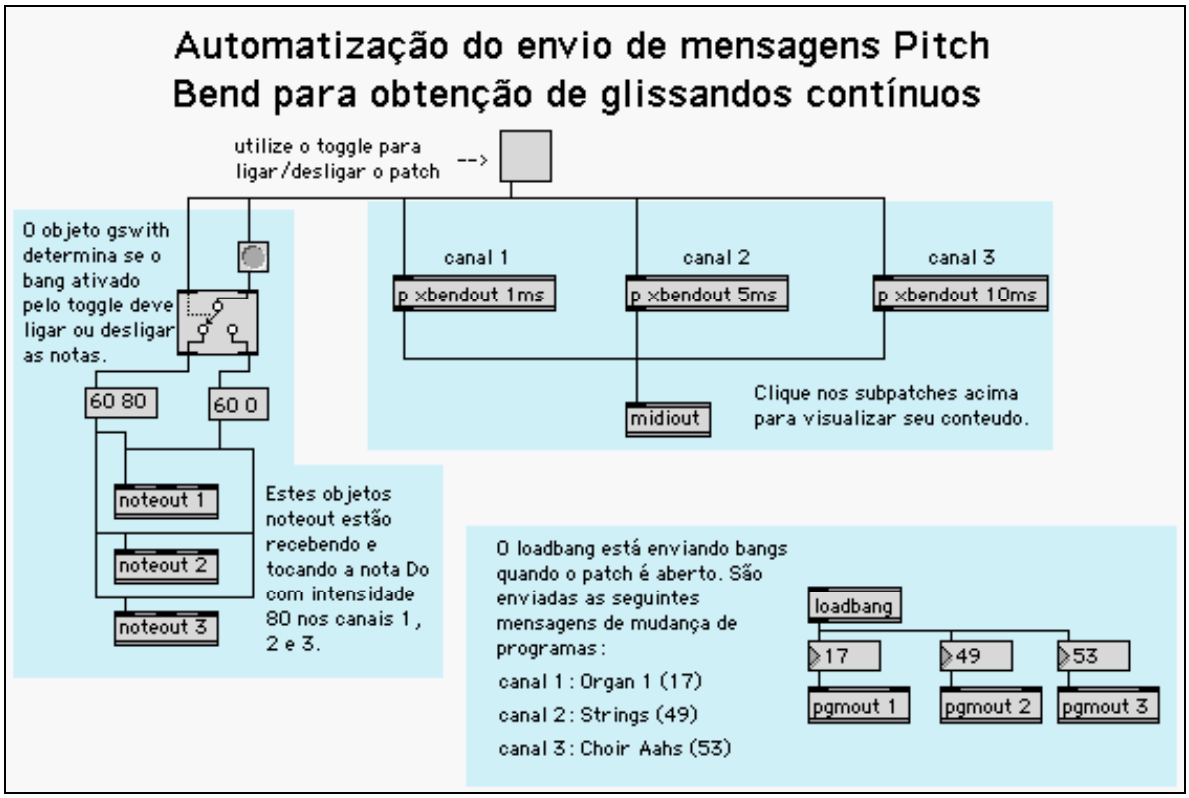


Figura 165 – Resposta do exercício com *pitch bender*

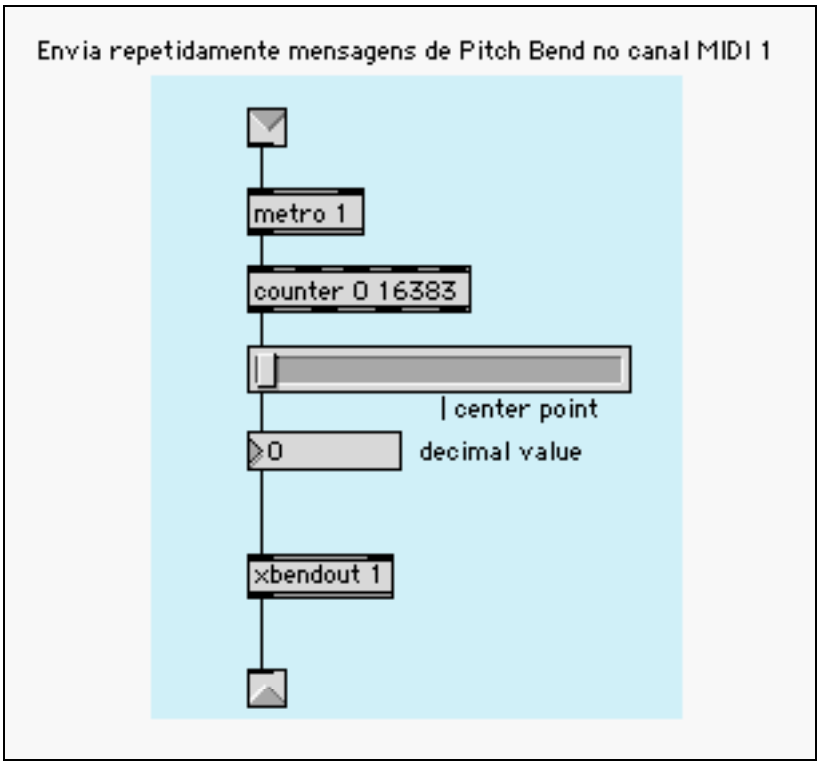


Figura 166 – Subpatch "xbendout 1ms", da resposta do exercício com *pitch bender*

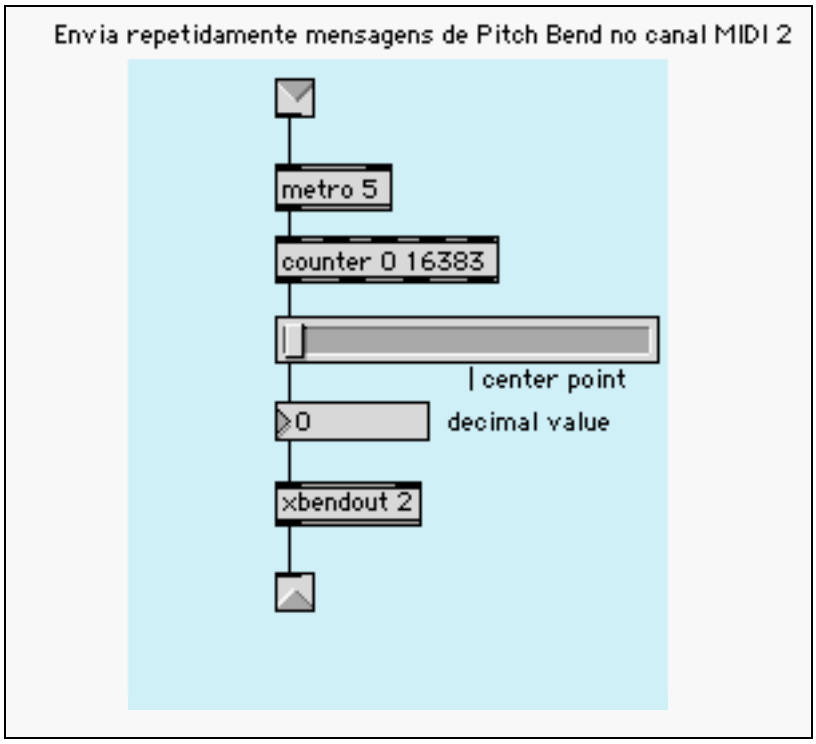


Figura 167 – Subpatch "xbendout 5ms", da resposta do exercício com *pitch bender*

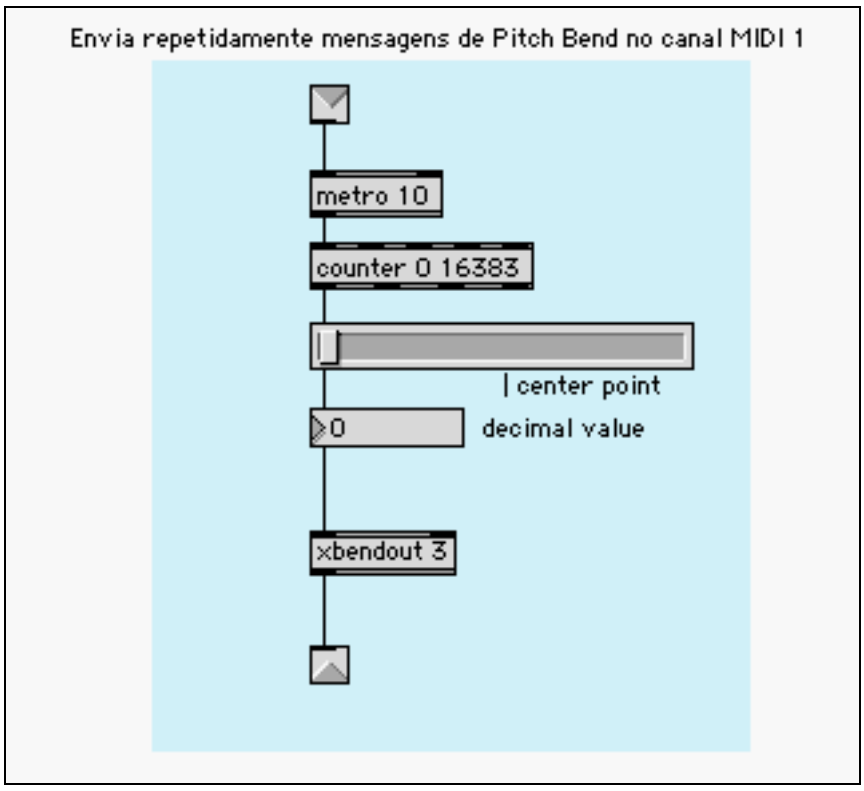


Figura 168 – Subpatch "xbendout 10ms", da resposta do exercício com *pitch bender*

7.2.3 AULA 2 – Programando com MIDI

Em geral, instrumentos de percussão são utilizados no canal MIDI 10. Na figura 169, o canal MIDI 10 deve ser selecionado para que os instrumentos de percussão de seu instrumento MIDI possam ser controlados pelos botões coloridos. Neste exemplo da figura 130, algumas ligações entre os objetos não estão visíveis para o usuário. Utilize o exemplo para aprender a tornar visíveis e invisíveis os objetos e ligações do *patch*.

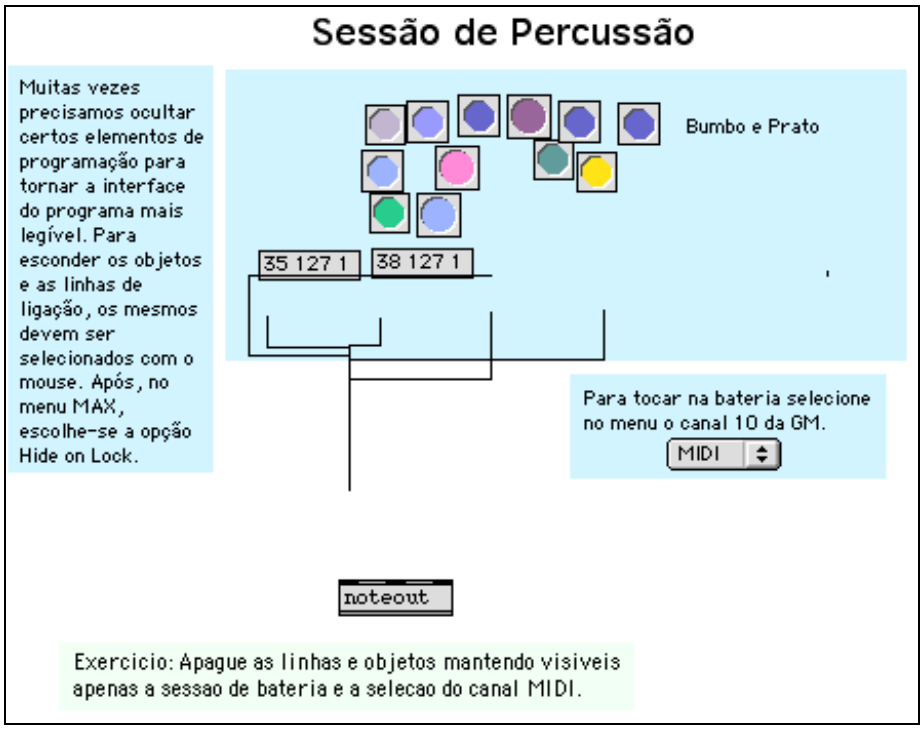


Figura 169 – Montando uma bateria

Sistemas de controle de instrumentos eletrônicos por computador podem ser programados utilizando o protocolo MIDI. Existem várias formas de controlar instrumentos eletrônicos de percussão remotamente. Uma das formas mais simples é através do próprio teclado do computador. Observe que no exemplo da figura 170 é utilizado o objeto ***key*** para receber o código da tecla e enviá-lo a um objeto ***select***. O objeto ***select***, por sua vez, seleciona apenas três notas MIDI que podem ser enviadas ao objeto ***makenote***. Utilize o programa para tocar ritmos de bateria pelo teclado do computador e resolva o exercício proposto.

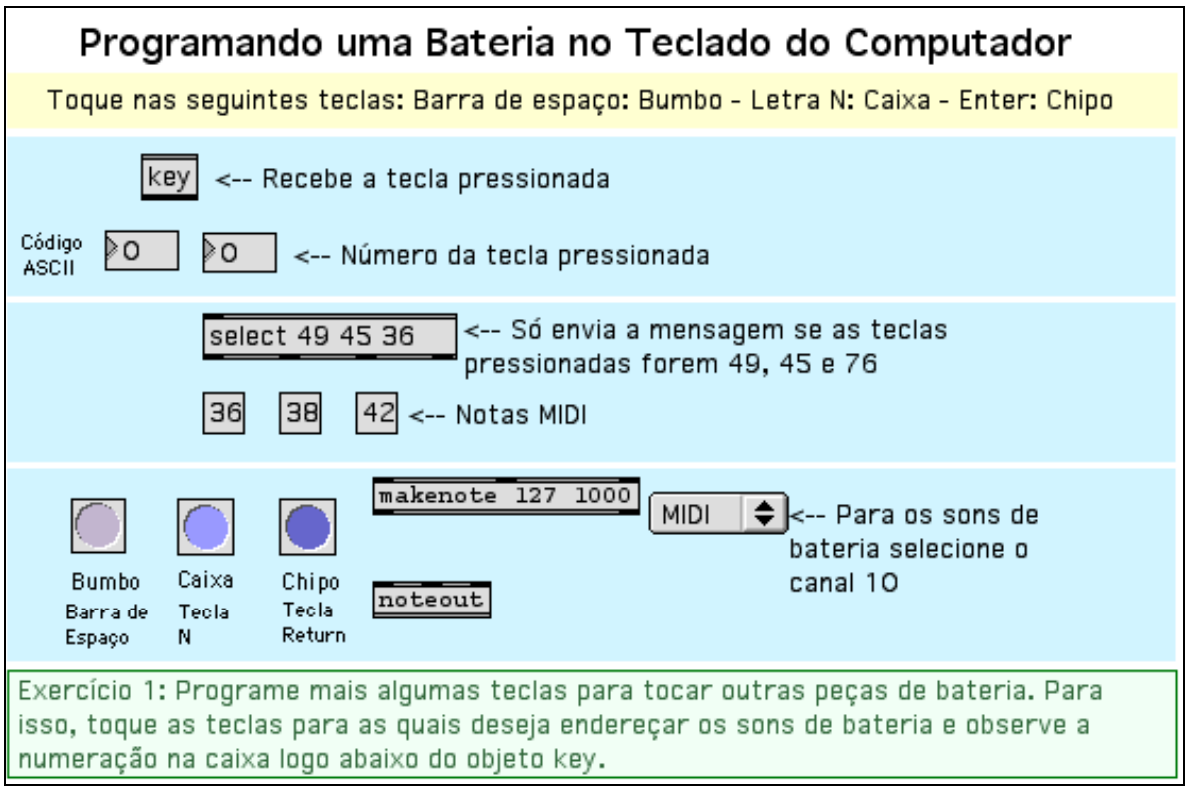


Figura 170 – Programando uma bateria no teclado do computador

7.2.4 AULA 3 – Programando um Seqüenciador

O *patch* mostrado na figura 171 apresenta uma implementação utilizando o objeto ***detonate***. Este objeto possui funções de seqüenciamento MIDI, ou seja, possibilita que eventos MIDI sejam gravados, reproduzidos, modificados através de uma janela (basta clicar no próprio objeto e esta aparece), salvar permanentemente o seu conteúdo em um arquivo MIDI ou ainda carregar um arquivo MIDI externo qualquer para manipulação junto ao MAX.

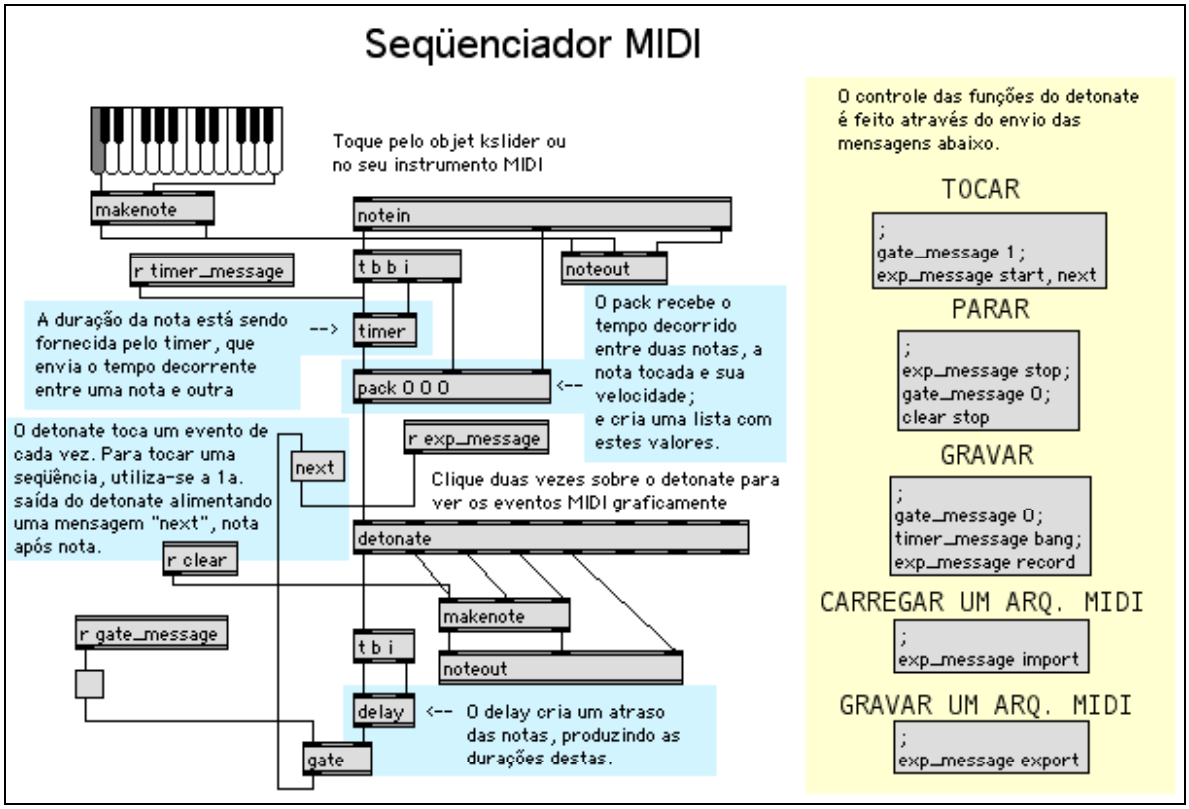


Figura 171 – Sequenciador MIDI

O *patch* da figura 171 possui, à direita, uma área de controle das principais funções do *detonate*, os comandos são recebidos pelos objetos ***receive*** (r) distribuídos na janela e passados ao ***detonate***.

Através do objeto ***kslider*** ou do seu instrumento MIDI externo, o usuário toca notas que serão recebidas pelo objeto ***trigger***, este, ao receber tais notas, as envia em forma de *bangs* ao ***timer*** e em forma de número inteiro ao objeto ***pack***. Um *bang* serve para ativar o ***timer*** e o outro *bang* – o do *inlet* esquerdo - faz com que o tempo de duração seja enviado ao ***pack***. Perceba que, como não foi usado o objeto ***stripnote*** junto ao *makenote*, tanto notas com velocidades positivas como notas com velocidade 0 (as *note-offs*) são enviadas ao ***trigger***. Então, quando o usuário apertar uma tecla do teclado, um *bang* será enviado e quando ele soltá-la, outro *bang* será enviado ao ***trigger***. O tempo decorrido entre estes dois eventos corresponde a duração da nota. Isto é necessário pois o ***detonate*** não possui entrada ideal para duração.

Continuando, o objeto ***pack*** forma uma lista com a duração da nota, a nota em valor MIDI e sua velocidade, e a envia ao ***detonate***, que, quando em modo de gravação, reconhecerá a lista como uma nota vinda de um instrumento MIDI e a gravará.

Para a reprodução de seu conteúdo, é enviada a mensagem “start,next” ao ***detonate***, isto faz com que ele toque a primeira nota que possui. Como o ***detonate*** não possui um comando que o faça tocar todas as notas que ele contém, é necessário adaptar alguns objetos para que isto seja possível. O objeto ***trigger*** aparece novamente recebendo as notas que o ***detonate*** toca, passando um *bang* e o valor da nota MIDI ao ***delay***. Este irá criar um atraso das notas, produzindo as durações destas, que é enviado ao ***gate*** que realimenta a mensagem “next” localizada logo acima do ***detonate*** e que faz com que a próxima nota seja reproduzida.

Cada nota realimenta esta cadeia de objetos causando um ciclo responsável pela reprodução precisa de todo o conteúdo do ***detonate***.

As opções de carregar e gravar um arquivo MIDI são mensagens de apenas um comando para o ***detonate*** (“import” e “export”, respectivamente). Ao clicar na primeira, surge uma janela onde o usuário deverá selecionar o arquivo MIDI que deseja carregar no ***detonate***. Na segunda, surge uma janela semelhante onde o usuário deve digitar o nome do novo arquivo que guardará o conteúdo do ***detonate***.

7.3 MÓDULO DE ENSINO DE PROGRAMAÇÃO COM ÁUDIO DIGITAL

Muitos benefícios podem ser obtidos através da representação digital do som, dentre os quais pode-se citar: síntese de novos sons realizada através de procedimentos matemáticos, técnicas de aplicação do processamento digital para sinais de áudio e gravação em alta fidelidade. O MSP é um software que disponibiliza um conjunto de ferramentas para integrar áudio digital e síntese aos objetos MIDI do Max. Através do MSP é possível projetar um instrumento para as necessidades do músico, construir um instrumento e ouvir os resultados em tempo real e integrar processamento de áudio aos programas para composição e performance. Os exemplos e exercícios práticos que constituem o Módulo de Ensino de Programação com Áudio Digital foram baseados na obra de [DOB 98].

7.3.1 AULA 1 - Introdução à Programação com Áudio Digital – MSP

7.3.1.1 Diferenças entre os objetos Max e os objetos MSP

Objetos MSP necessitam produzir uma quantidade de números suficiente para gerar um sinal de áudio de alta qualidade. Por exemplo, para gerar áudio na qualidade de CD é necessário produzir 44.100 números por segundo. Por isso os objetos MSP operam mais rapidamente do que os objetos padrões do Max que utilizam um escalonamento em milissegundos.

Um patch MSP deve ser entendido como um conjunto interconectado de objetos como uma rede de sinais e não como um patch Max no qual os eventos ocorrem num instante específico. No patch Max nada acontece até que ocorra um clique no mouse causando o envio de uma mensagem MIDI de um objeto a outro. Por outro lado, uma rede de sinais MSP está ativa desde o instante que é ligada até o instante que é desligada. Todos os seus objetos realizam uma comunicação constante para calcular os valores apropriados de um som. Portanto, durante o funcionamento de um patch MSP cada objeto gera constantemente valores numéricos para os objetos conectados em seus *outlets*. Existem diferenças na aparência e nas ligações entre objetos Max e MSP. Os nomes dos objetos MSP terminam com um til (~), como se fosse um ciclo de onda senoidal, para distingui-los dos objetos Max. Além disso as ligações entre os objetos (*patch cords*) possuem um traço pontilhado, ao contrário das ligações Max que são linhas simples.

7.3.1.2 Utilização de Sinais Digitais

Um som na forma digital é uma seqüência de números. Qualquer operação aritmética executada com esses números é uma forma de processamento de áudio. O MSP apresenta um objeto para multiplicar sinais e outro para somar. A operação de multiplicação resulta na amplificação do áudio. Por exemplo, a multiplicação de cada número de um sinal digital por dois resulta no dobro de sua amplitude, ou seja, um acréscimo de 6 dB. Multiplicando cada número de um sinal digital por um valor entre 0 e 1 será obtida a redução de sua amplitude. Por outro lado, a operação de adição resulta

na mixagem do áudio. Dados dois sinais de áudio, um novo sinal pode ser criado quando os primeiros números de cada sinal são adicionados aos números subseqüentes.

7.3.1.3 Paleta de Objetos do MSP no Modo de Edição

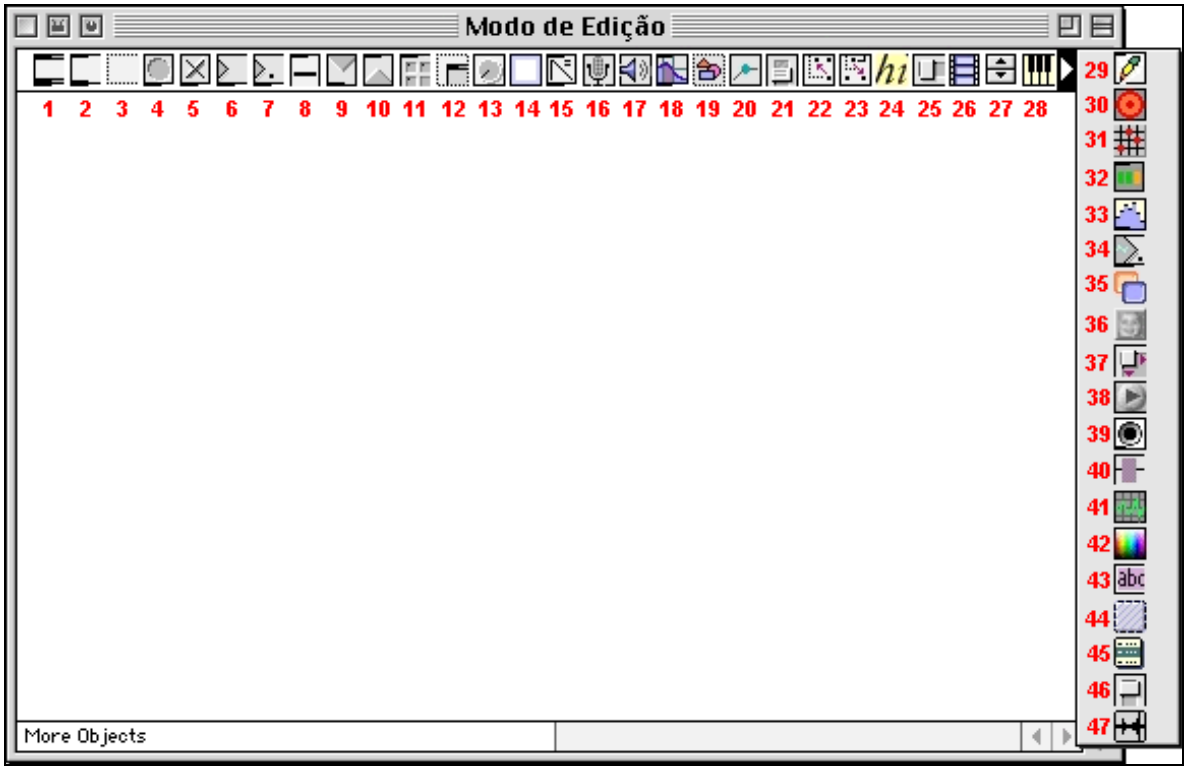


Figura 172 - Paleta de objetos do Max/MSP

1. **Object Box** – abre uma lista de Comandos pré-definidos do Max
2. **Message Box** – envia uma mensagem para um objeto do Max
3. **Comment** – utilizado para comentários e títulos em um programa
4. **Button** – envia um disparo ou bang ao ser clicado
5. **Toggle** – alterna entre ligado e desligado (1 e 0)
6. **Number Box** – mostra e envia um número de saída
7. **Float Number Box** – mostra e envia números decimais de saída
8. **Slider –** envia números ao mover-se a barra verticalmente na tela
9. **Inlet** – recebe uma mensagem na entrada de um subprograma
10. **Outlet** – envia uma mensagem de saída de um programa (janela)
11. **Preset** – salva e recupera ajustes dos objetos
12. **Patcher-in-Box** – cria um subprograma dentro de um programa
13. **Dial** – envia números ao girar-se o botão na tela
14. **Drop Region for Files** – define região onde é possível arrastar e soltar arquivos
15. **Script-Driven Envelope – e**nvelopa dados dentro do objeto **Funnel**
16. **Audio Input –** entrada de áudio simples de dois canais
17. **Audio Output –** saída de áudio simples de dois canais
18. **Filter Graph –** envelope para definição gráfica de um filtro
19. **Picture From PICT File –** mostra uma figura PICT numa caixa
20. **Breakpoint function Editor –** editor gráfico de envelopes
21. **Signal Level Fader –** *fader* logarítmico para sinais

22. **Graphic Switch –** recebe uma mensagem de entrada em uma das duas tomadas de entrada
23. **Graphic Gate -** passa a mensagem de entrada para uma das tomadas de saída
24. **Hint -** define área onde se verá uma dica de tela quando o cursor passar
25. **Horizontal Slider –** envia números ao mover-se a barra horizontalmente na tela
26. **QuickTime Movie –** reproduz um filme QuickTime em uma janela
27. **Increment / Decrement –** incrementa ou decrementa valores numéricos de uma caixa
28. **Keyboard Slider –** envia notas ou valores correspondentes para um objeto
29. **LCD –** desenha formas, linhas e texto em uma caixa
30. **LED –** mostra os estados ligado e desligado em cores
31. **Matrix Control -** matriz controlada pelo cursor
32. **Signal Level Meter –** medidor visual do nível de áudio
33. **MultiSlider –** pode funcionar como um arranjo de sliders
34. **Signal Monitor –** como a **number box**, mas aceita apenas sinais MSP
35. **Panel –** um painel retangular com bordas e tamanho ajustável
36. **Picture-based Control –** possibilita que uma figura seja usada como um **button**, **toggle** ou **dial**
37. **Picture-based Slider –** possibilita que uma figura seja usada como slider
38. **QuickTime Play Controller –** controlador de execução de filmes QuickTime
39. **Radio Group –** um grupo de botões de rádio para seleção
40. **Range Bar –** seleciona e mostra a faixa de dois valores
41. **Signal Scope –** funciona como um osciloscópio
42. **Color Swatch –** caixa onde é possível selecionar e exibir cores RGB
43. **Text Field –** campo para o usuário final entrar um texto no patch
44. **Transparent Button –** usado por trás de um texto ou figura
45. **Pop-up Menu –** utilizado para criar-se a lista de um menu
46. **Vertical Slider –** mostra valores recebidos
47. **Waveform Display –** visualiza e edita o conteúdo de um buffer

7.3.1.4 Programação com Áudio Digital

Um instrumento é formado por uma rede de objetos MSP. O conversor digital-analógico de um instrumento MSP é representado pelo objeto dac~. Para que seja possível ouvir o som, o dac~ deve receber em seu *inlet* um sinal, diferente de zero, de amplitude entre –1.0 e 1.0.

7.3.1.5 Unidades Fonte

São as unidades da síntese sonora que produzem o som. No MSP estão representadas pelos objetos osciladores : cycle~ , noise~ e o phasor~ .

O oscilador *wavetable* é representado pelo objeto cycle~ que é utilizado para produzir uma forma de onda periódica. O cycle~ lê ciclicamente uma lista de 512 valores de amplitude. Ao ler os 512 valores de amplitude, retorna ao primeiro valor para repetir toda a leitura novamente.

Conforme a figura 173, o objeto *~ tem o papel de amplificador. Para mudar a amplitude no MSP, multiplica-se cada amostra de áudio digital por qualquer número diferente de 1. A mensagem *startwindow* inicia o processamento de áudio na janela do *Patcher* que contiver um dac~ , e em qualquer dos *sub-patches* da janela. Porém desliga o áudio em todos os outros *Patches*.

Observe na figura 173 a utilização dos objetos até aqui explicados e tente acionar o oscilador e clicar nas caixas de valores para mudar a freqüência do oscilador e o volume. Depois tente resolver o exercício.

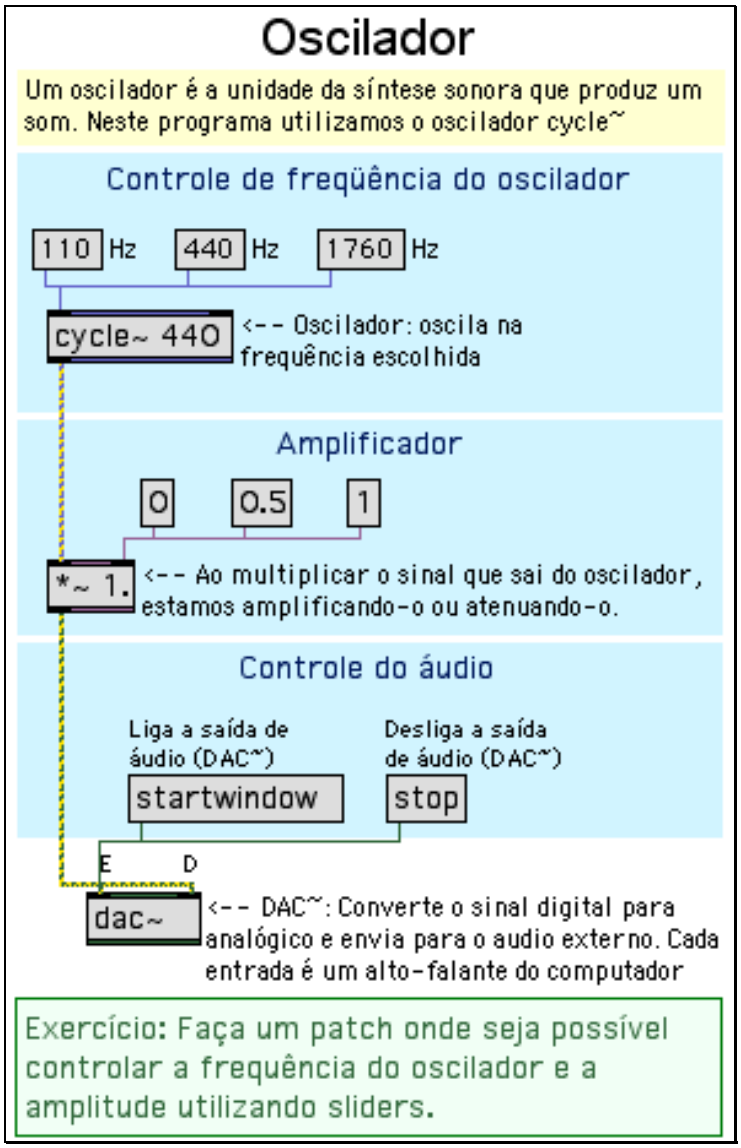


Figura 173 – Oscilador básico em MSP

Para solucionar esse primeiro exercício é necessário utilizar dois sliders. Um deles conectado ao objeto **cycle~** e o outro ao objeto *~ . No entanto os valores precisam estar na escala adequada. Portanto, o sinal de amplificação não pode ultrapassar o valor 1. Por isso divide-se o valor provindo do slider por 100 para depois enviá-lo ao amplificador conforme mostra a figura 174.

Figura 174 – Resposta do exerício sobre osciladores

Além da onda senoidal podemos trabalhar com outras formas de onda. A onda dente-de-serra é gerada pelo objeto **phasor~** que produz repetidamente, numa dada freqüência, sinais que aumentam linearmente de 0 até 1. Para obter uma onda quadrada deve-se enviar a saída do **phasor~** para o operador **>~0.5**. Para a primeira metade de cada ciclo de onda, a saída do phasor~ é menor que 0.5, então o objeto >~ envia o valor 0. Para a segunda metade do ciclo, a saída do **phasor~** é maior do que 0.5, então o objeto >~ envia o valor 1. Execute o exemplo da figura 175 e ouça o resultado.

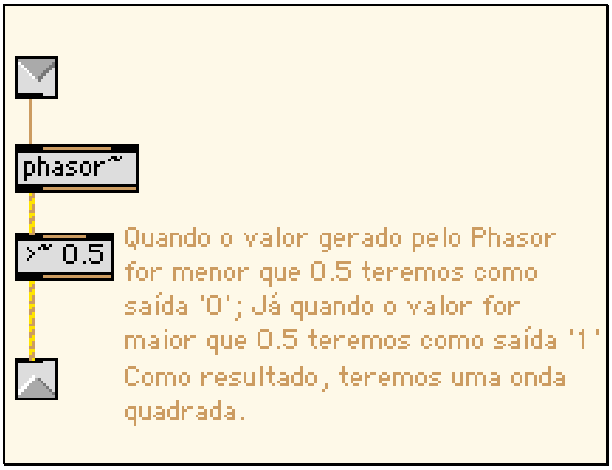


Figura 175 – Operando um **phasor~**

7.3.1.6 Mudanças de Amplitude

Mudanças drásticas nos valores de amplitude podem causar um *click*. Por isso é aconselhável utilizar um outro sinal para controlar a amplitude. Isso pode ser feito com o emprego do objeto **line~** que irá causar mudanças lineares nos valores da rede de sinais.

No exemplo da figura 176 fica claro o funcionamento do fade. Neste exemplo os envelopes de amplitude são determinados por listas de números enviados ao objeto **line~** que realiza a mudança linear de um número a outro em um determinado período de tempo.

O objeto **selector~** pode ser usado para escolher um dos vários sinais a ser enviado à saída do objeto ou interromper o fluxo dos sinais recebidos.

O ezdac~ é um botão utilizado para ligar e desligar o áudio no MSP.

Alguns objetos MSP existem somente para servirem de ligação entre o MSP e o Max traduzindo entre eles os controles e o áudio. Os principais objetos utilizados nesta ligação são o **sig~** , o **line~** e o **snapshot~** . Recebem mensagens MIDI em seus *inlets* e convertem-nas para a rede de sinais nos quais seus *outlets* podem estar conectados.

O objeto sig~ converte um número em um sinal constante. Ele recebe um número em seu *inlet* e envia o sinal correspondente ao valor. Isso é útil nos casos em que se deseja combinar valores constantes com sinais variáveis.

Observe a figura 176 e selecione a opção no menu para verificar o emprego de fades no MSP. Clique nas caixas de valores no quadro “Com Fade” e no quadro “Sem Fade” e ouça as diferenças no som.

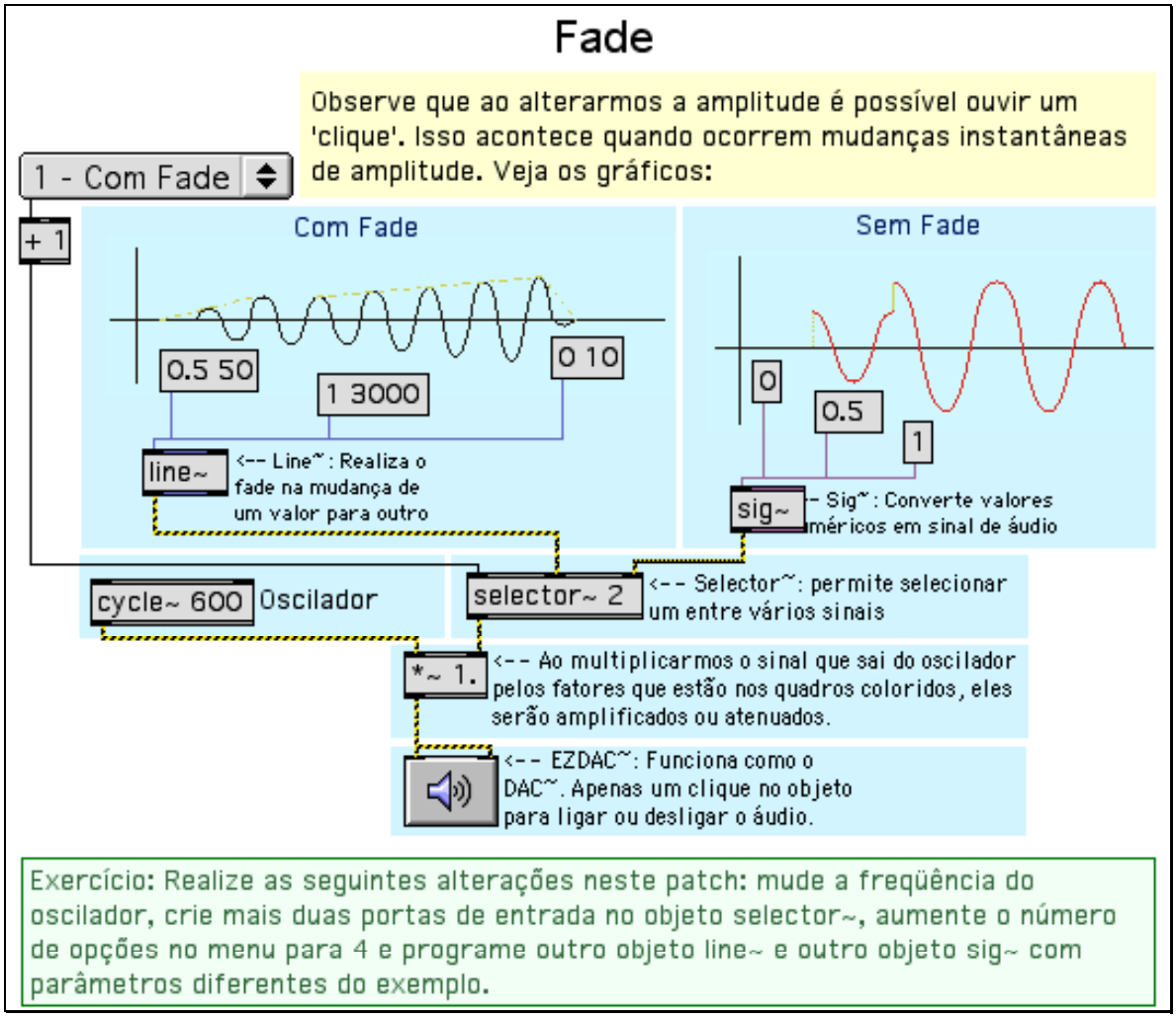


Figura 176 – A programação de Faders em MSP

Durante a adição de sinais é comum que os valores excedam o valor 1. Um sinal com amplitude superior ao valor 1 será distorcido pelo objeto **dac~**. Nestes casos é aconselhável que o valor seja ajustado antes de chegar ao amplificador, para nunca atingir o valor 1.

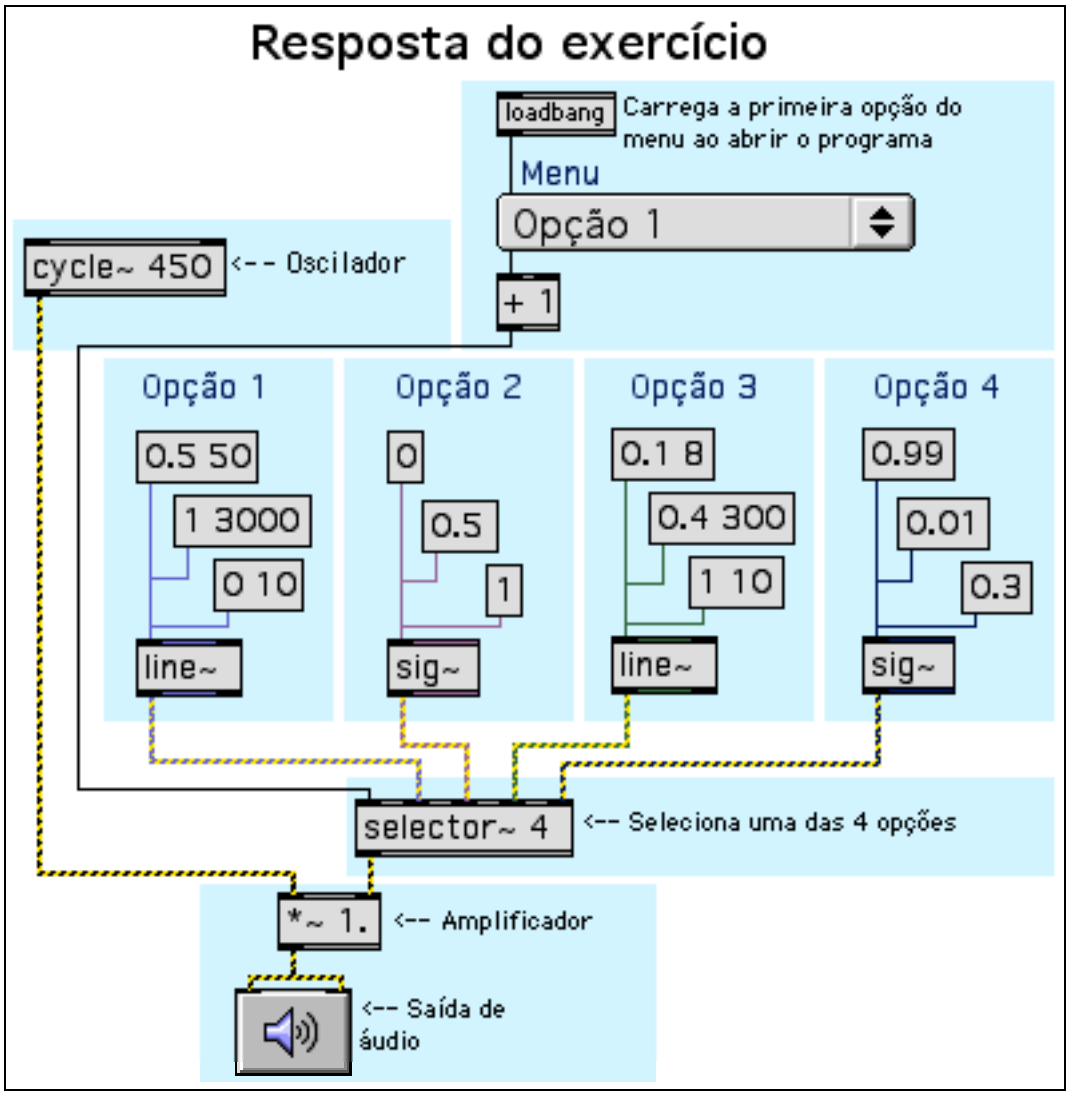


Figura 177 – Resposta do exercício com Faders

Conforme pode ser observado na figura 177, há duas opções com objetos **line~** e duas com objetos **sig~,**. que faz a transformação de um número em um sinal. Todas as opções com **line~** resultam em um fade gradual, enquanto que as opções com o objeto **sig~** resultam em um "click" sonoro.

Na opção 1, por exemplo, se o valor do **line~** for "0" e forem enviados os valores "0.5 50", o **line~** terá como saída números que irão aumentar seu valor de 0 até 0.5 num período de 50 milissegundos.

7.3.1.7 Aperfeiçoando o Oscilador

O objeto **buffer~** tem o papel de armazenar um som. Através de uma mensagem *read* é possível abrir uma caixa de diálogo para selecionar e carregar um arquivo de som que será armazenado no objeto **buffer~**. O objeto **cycle~** pode utilizar o arquivo de som armazenado no objeto **buffer~** desde que o nome do mesmo arquivo seja o argumento escrito no objeto **cycle~**. Neste caso, o **cycle~** não irá utilizar a onda senoidal padrão e sim o som carregado no objeto **buffer~**.

Observe na 178 como o arquivo de som *Instrumento* é carregado para o patch pelo objeto **buffer~** e como está sendo utilizado no objeto **cycle~**. Depois resolva o exercício proposto.

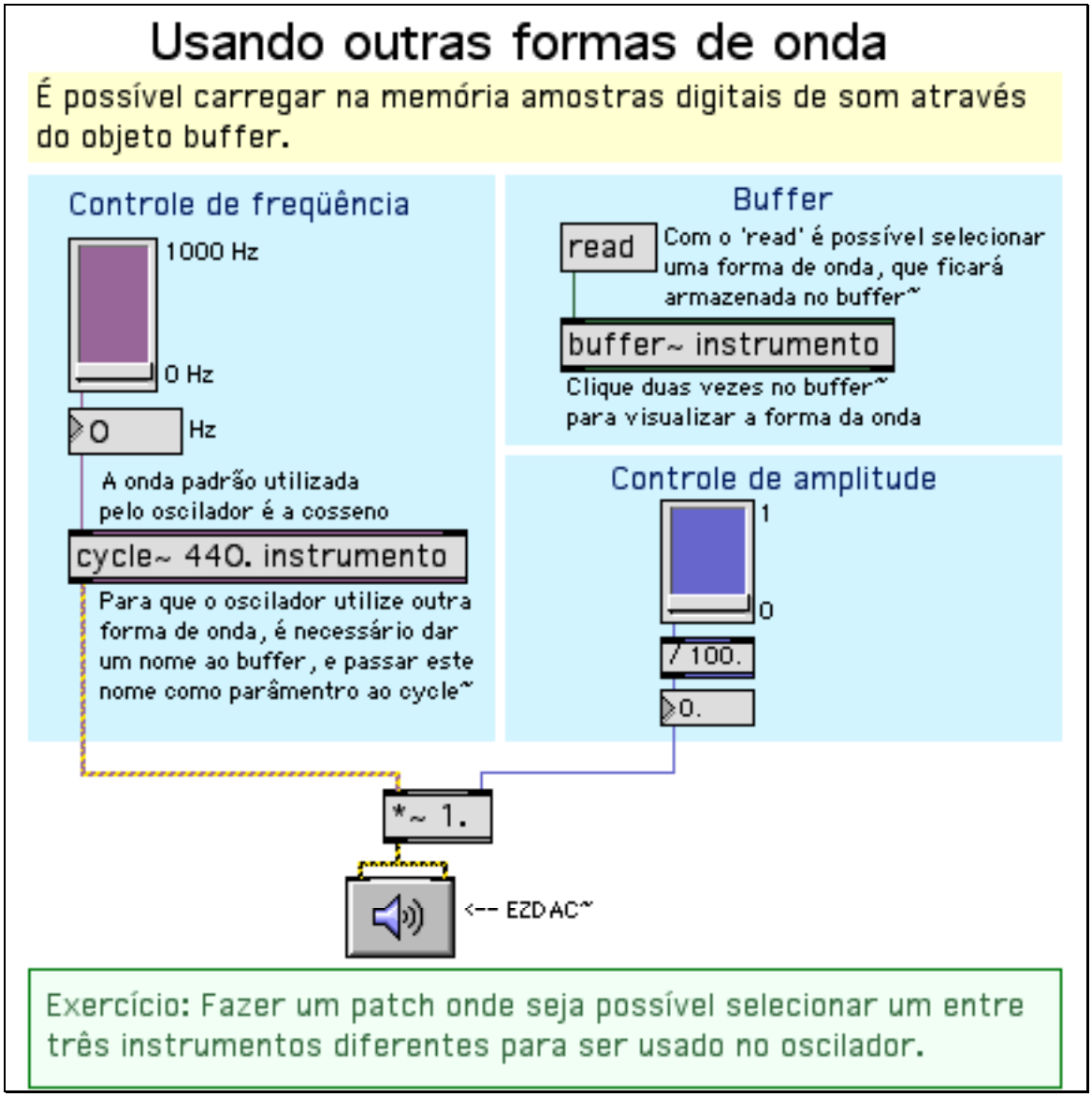


Figura 178 – Usando outras formas de onda

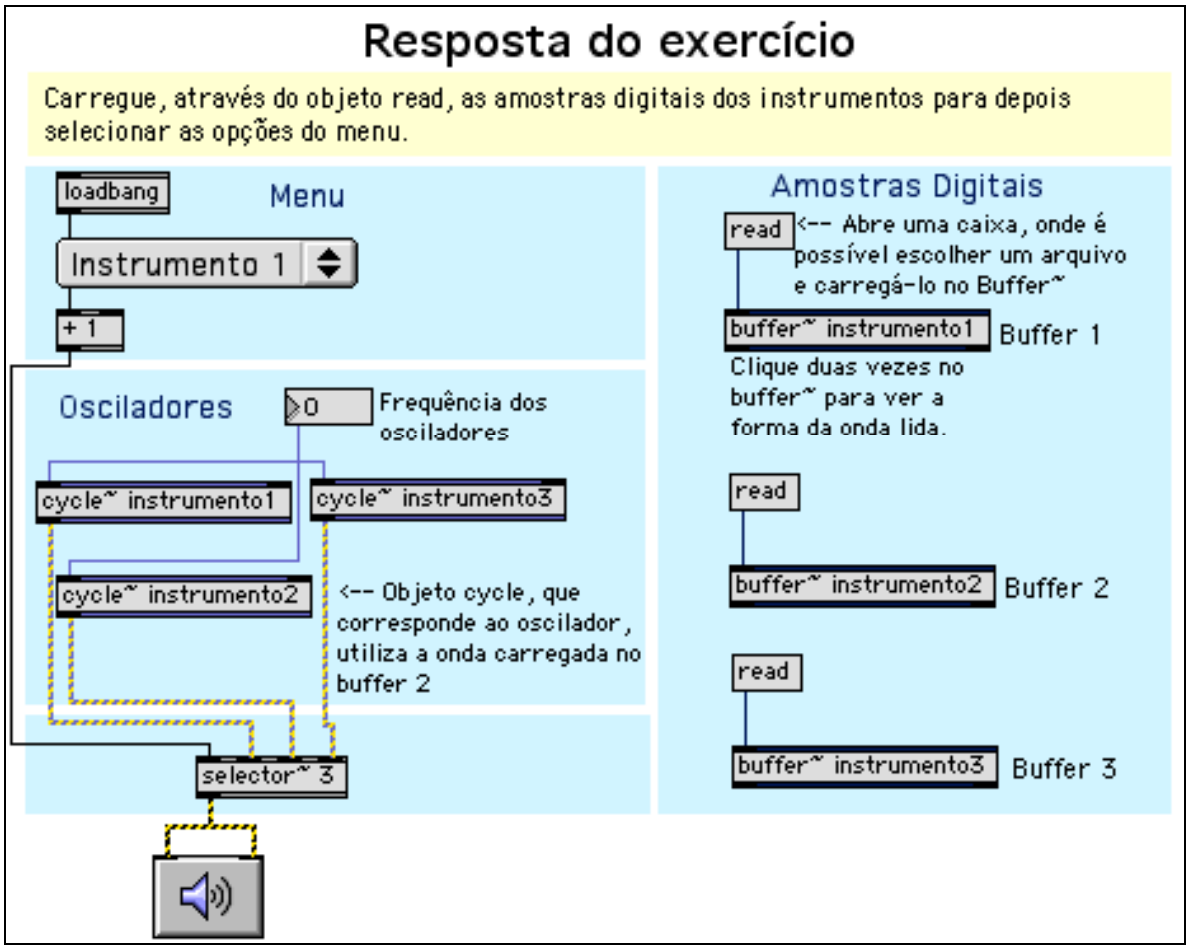


Figura 179 – Resposta do exercício de "Usando outras formas de Ondas"

Este exercício foi separado em duas partes fundamentais: uma que carrega os três instrumentos em três objetos **buffer~** separados e outra parte na qual o usuário pode selecionar o instrumento e a freqüência dos osciladores.

Para carregar três instrumentos separadamente, são necessários três conjuntos de objetos **buffer~**, cada um carregando o arquivo que é indicado com um único nome. Não se pode esquecer de que se deve enviar uma mensagem *read* para cada objeto **buffer~**.

Para selecionar qual dos instrumentos deseja-se ouvir, foi utilizado um objeto **umenu**, que envia o número referente ao instrumento selecionado a um objeto **selector~** e que permite somente a passagem do sinal de áudio do instrumento selecionado. O objeto **loadbang**, no topo do patch, envia um bang automaticamente quando o patch é carregado. Isto é fundamental para o funcionamento do programa.

7.3.2 AULA 2 – Modificadores do Som

Os modificadores têm a função de alterar o som provindo de uma fonte sonora. Em MSP são exemplos de modificadores os objetos **function**~ (envelope), ***~** (amplificador), **lores**~ e o **reson**~ (filtros).

7.3.2.1 Programação de um envelope

Um envelope pode ser formado pela combinação dos objetos **line**~ e ***~** com a finalidade de controlar a amplitude de um sinal. No exemplo da figura 180, o ruído branco é produzido pelo objeto **noise**~. Veja na figura 180 como podemos programar envelopes para controlar a amplitude da onda num transcurso de tempo. Em seguida, tente resolver o exercício.

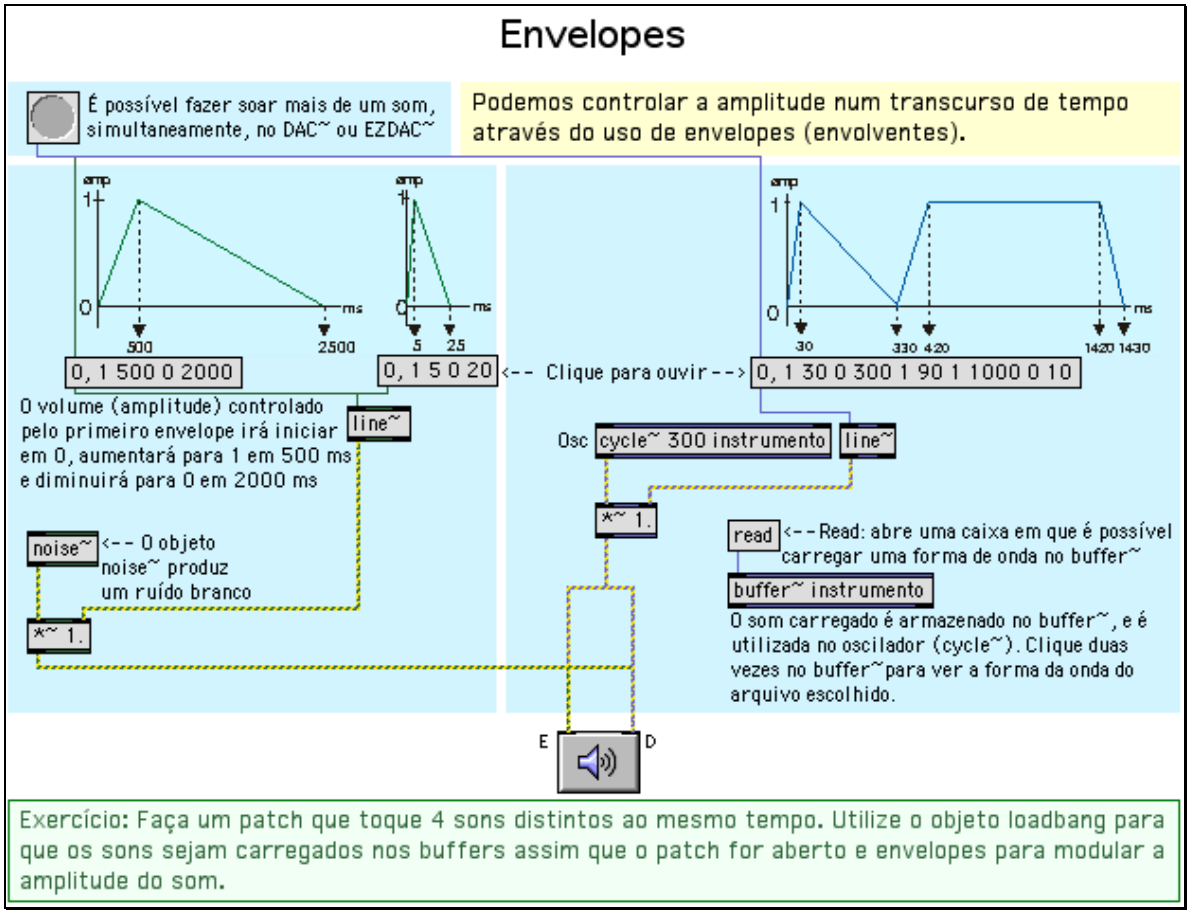


Figura 180 – A programação de envelopes para controle do volume num transcurso de tempo

Na figura 181, três instrumentos diferentes são carregados em três *buffers* utilizando o mesmo recurso do exercício de "Usando outras formas de ondas" da aula 1. Um objeto **loadbang~** envia um bang para três mensagens **read**, cada uma indicando um arquivo separado, que chegam aos seus respectivos objetos **buffer~** efetuando o carregamento dos arquivos sonoros.

Nos instrumentos 1, 2 e 3 é utilizado o objeto **cycle~** que usa o buffer anteriormente carregado, enviando o seu sinal para um objeto ***~1** que realiza a função de amplificador. O objeto **line~** recebe uma lista de parâmetros referentes aos valores do envelope e repassa ao amplificador. No instrumento quatro é utilizado o mesmo princípio, com a diferença de que é usado um objeto **noise~** como gerador de sinais sonoros ao invés de um **cycle~**.

Como o **ezdac~** tem um limite de amplitude sonora de 1, é conveniente fazermos com que este limite não seja ultrapassado. Para tanto, foi implementada uma multiplicação de 0.25 – que equivale a uma divisão por quatro – para permitir que os quatro instrumentos sejam amplificados igualmente. Observação: foi utilizada uma multiplicação pois uma divisão requer maior processamento por parte do computador.

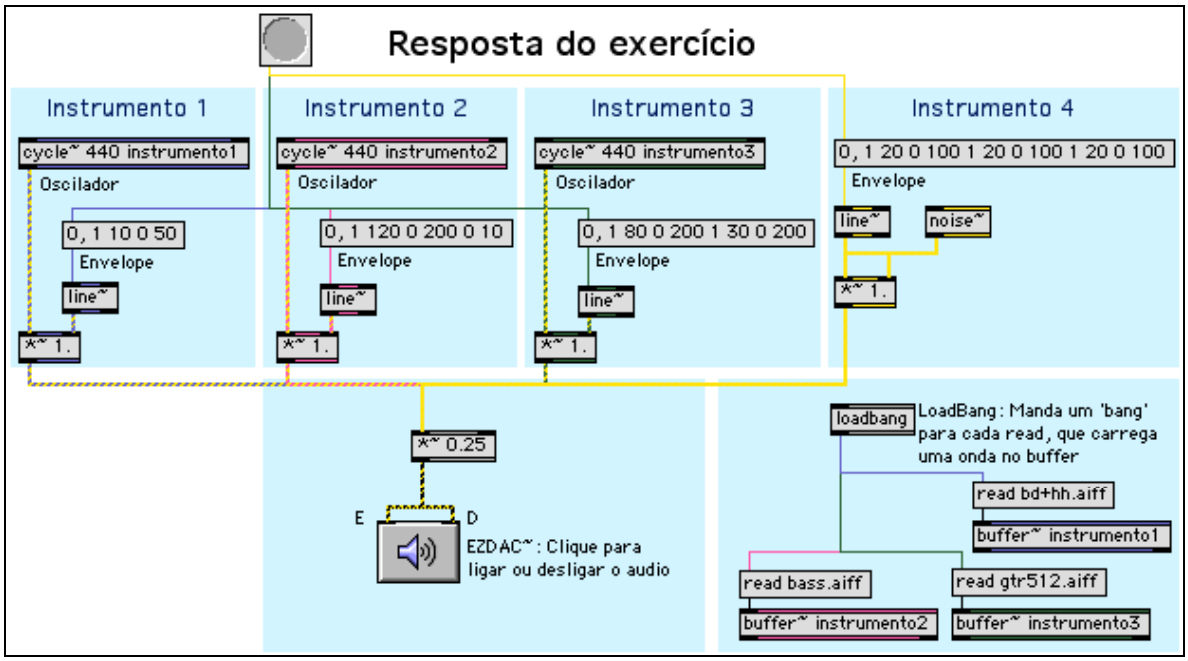


Figura 181 – Resposta do exercício com Envelopes

Apesar do principal uso do objeto **cycle~** ser destinado à geração de ondas de áudio periódicas , ele também pode ser usado para funções de controle de sub-áudio. O objeto **cycle~** permite ler qualquer amostra desejada da onda desde que sua freqüência seja mantida em 0 Hz e sua fase seja continuamente alterada entre 0 e 1. Observe na figura 182 como a fase do objeto **cycle~** pode ser mudada apropriadamente através do objeto **line~** . A utilização do objeto **phasor~** também é adequada para mudar a fase da onda do objeto **cycle~** já que tem o comportamento de uma onda dente-de-serra que percorre valores repetidamente de 0 a 1.

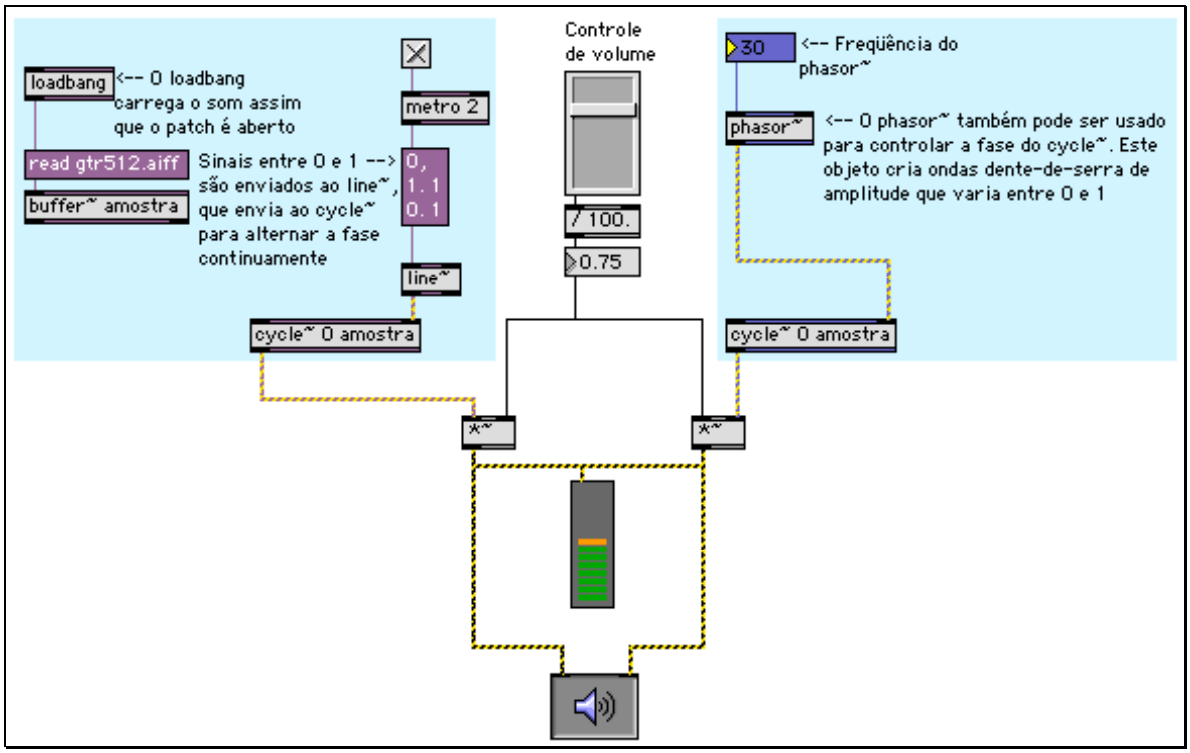


Figura 182 – Objeto **cycle~** controlando áudio

7.3.2.2 Programação de Filtros

Os objetos **reson~** e **lores~** são objetos que exercem a função de filtros sonoros. No exemplo da figura 183, o que for tocado no teclado MIDI será convertido para valores de freqüência. Esses valores de freqüência serão filtrados e depois amplificados. O objeto **reson~** é um filtro de passa-banda.

Neste exemplo os argumentos do filtro reson~ foram implementados através de hsliders. Juntamente com estes argumentos, ambos os filtros recebem o sinal através de objetos **receive~**. O sinal provém de um objeto kslider, os sinais MIDI são convertidos para valores de freqüência em *Hertz* pelo objeto mtof. Estes valores são fornecidos a um **phasor~** e o sinal proveniente deste phasor~ chega no objeto send~ para o envio de sinais.

Os objetos **send~** e **receive~** são similares aos homônimos do Max. Com eles é possível realizar conexões de sinais sem a utilização dos *patch cords.*

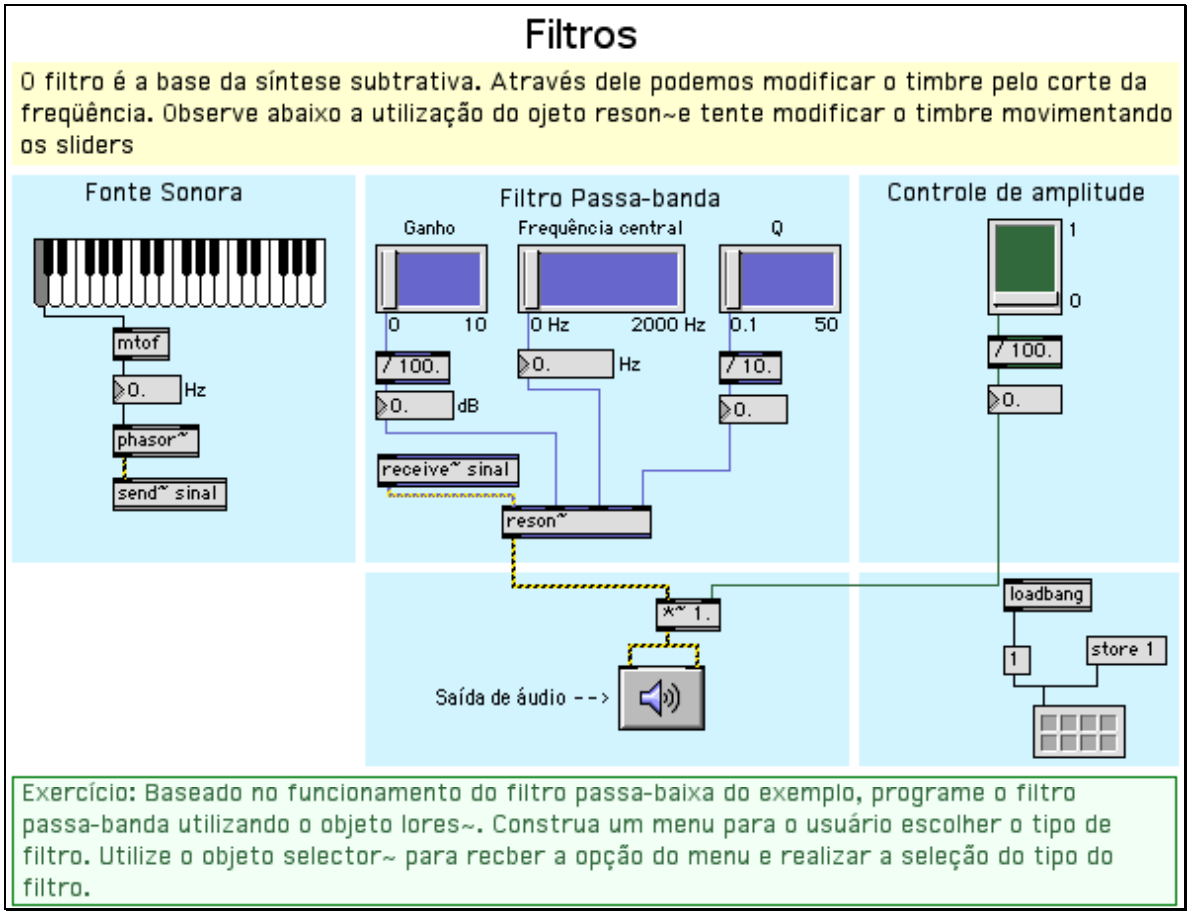


Figura 183 – Programação de Filtros

O filtro **lores~** é um filtro *passa-baixa*, que tem a função de "cortar" as freqüências altas. Os argumentos necessários são: o sinal a ser filtrado, o ponto de corte da freqüência em hertz e o "fator de ressonância" ou "Q" (que deve ter um valor entre 0 e 1).

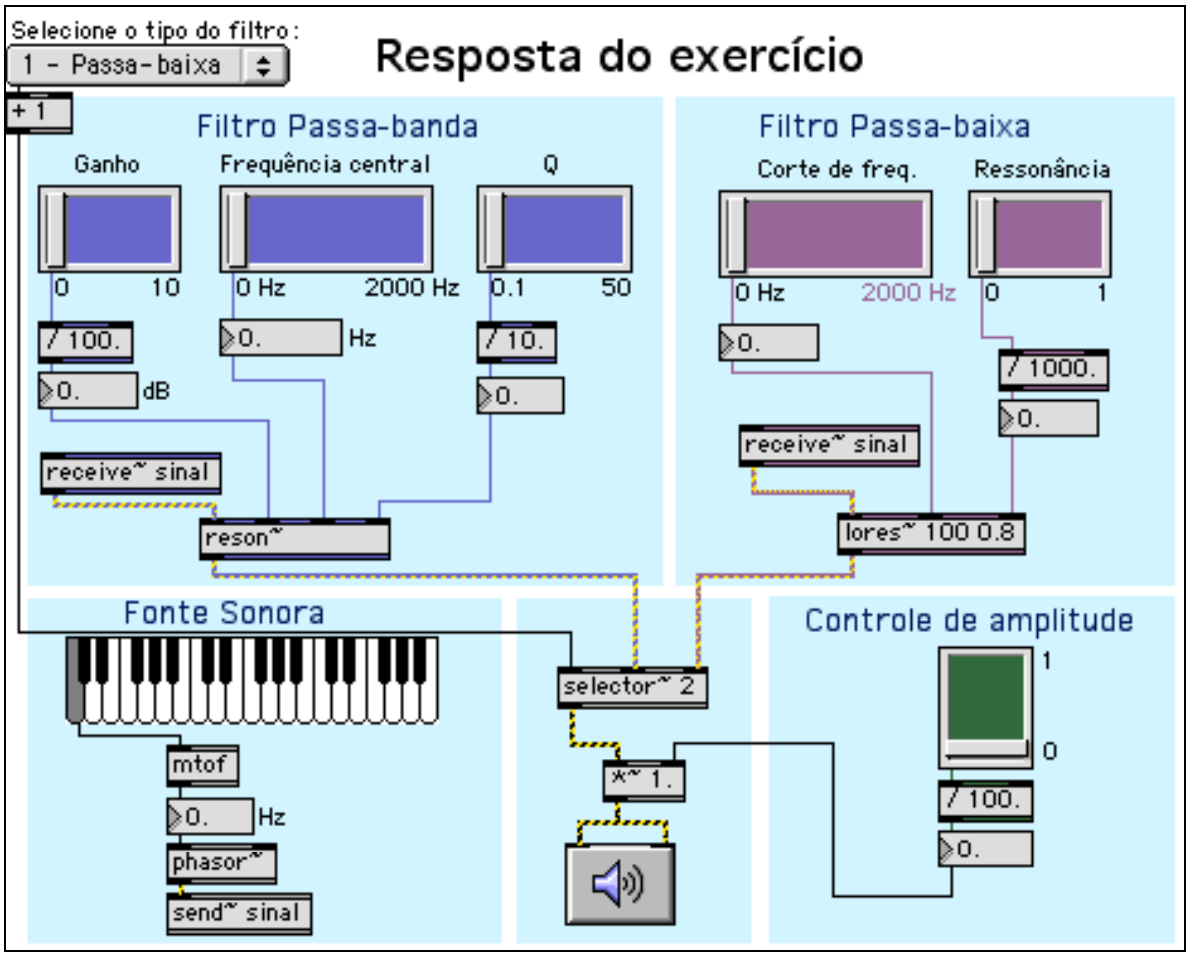


Figura 184 – Resposta do exercício com Filtros

7.3.2.3 Programando um Sintetizador Subtrativo

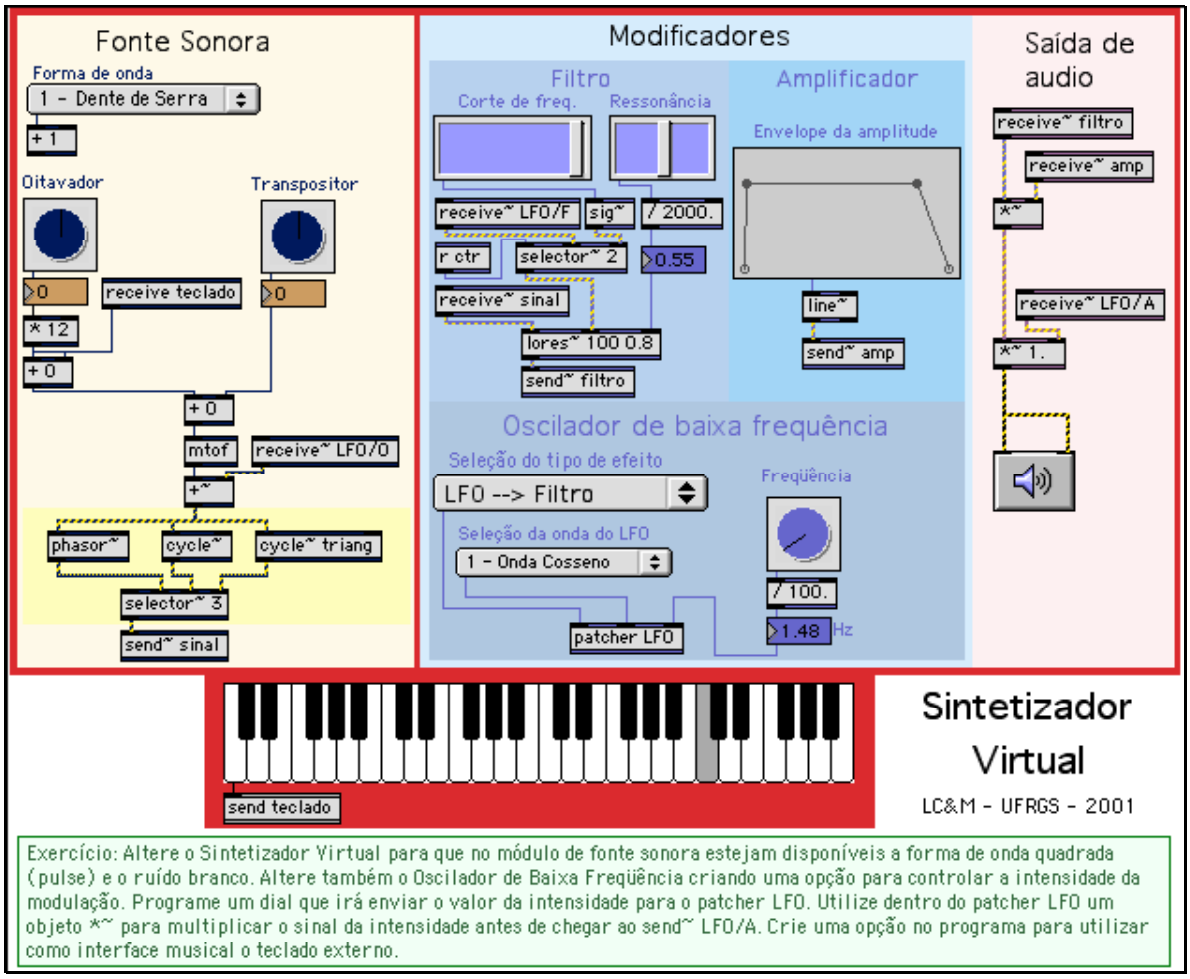


Figura 185 – Sintetizador Subtrativo Virtual

O Sintetizador Virtual da figura 185 funciona baseado em um sintetizador subtrativo analógico: a qualquer sinal de entrada aplica-se um processo de "filtragem" que permite, ou não, que algumas freqüências estejam presentes na saída sonora.

No Sintetizador Virtual, o sinal inicial é gerado no módulo Fonte Sonora. No menu, é possível escolher uma das seguintes formas da onda: dente-de-serra, senoidal ou triangular. Também é possível fazer transposição de notas ou utilizar o oitavador para deslocar a oitava central do **kslider**.

Todo sinal gerado na Fonte Sonora é enviado para o módulo Modificadores, através dos objetos **send~** e **receive~**. Neste módulo o sinal passa por um filtro passa-baixa (objeto **lores~**). Como visto anteriormente, o filtro passa-baixa que necessita de três argumentos: o sinal de entrada (provém da Fonte sonora), o ponto de corte da freqüência e o fator de ressonância.

O envelope presente no módulo Modificadores funciona como um amplificador e permite controlar o comportamento da amplitude do som num transcurso de tempo.

É possível utilizar um oscilador de baixa freqüência (LFO) para variar o valor do ponto de corte de freqüência automaticamente (criando um efeito de wah-wah), para alterar a amplitude ciclicamente (criando um efeito de trêmolo) ou ainda para mudar intermitentemente a freqüência provinda da Fonte Sonora (criando um efeito de vibrato).

Finalmente, o sinal filtrado e os valores do envelope são enviados para o último módulo, a Saída de Áudio (amplificador). Caso o LFO esteja configurado para ficar variando a amplitude (trêmolo), os valores apropriados de amplitude serão recebidos continuamente através do objeto **receive~LFO/A** e multiplicados por todo o sinal sonoro resultante, antes que seja enviado para o objeto **ezdac~**.

Após utilizar o Sintetizador Virtual para produzir diferentes sons, estudar seu funcionamento e sua programação, clique no patch Exercício no canto inferior esquerdo da tela e tente elaborar uma solução para o enunciado.

A solução do exercício proposto está na figura 186. Nela foram adicionadas mais duas formas de onda ao módulo Fonte Sonora: quadrada e ruído branco. Observe também que foi acrescentado um controle de intensidade ao oscilador de baixa freqüência. Ele é utilizado para controlar a intensidade da variação de amplitude quando o LFO estiver controlando o volume.

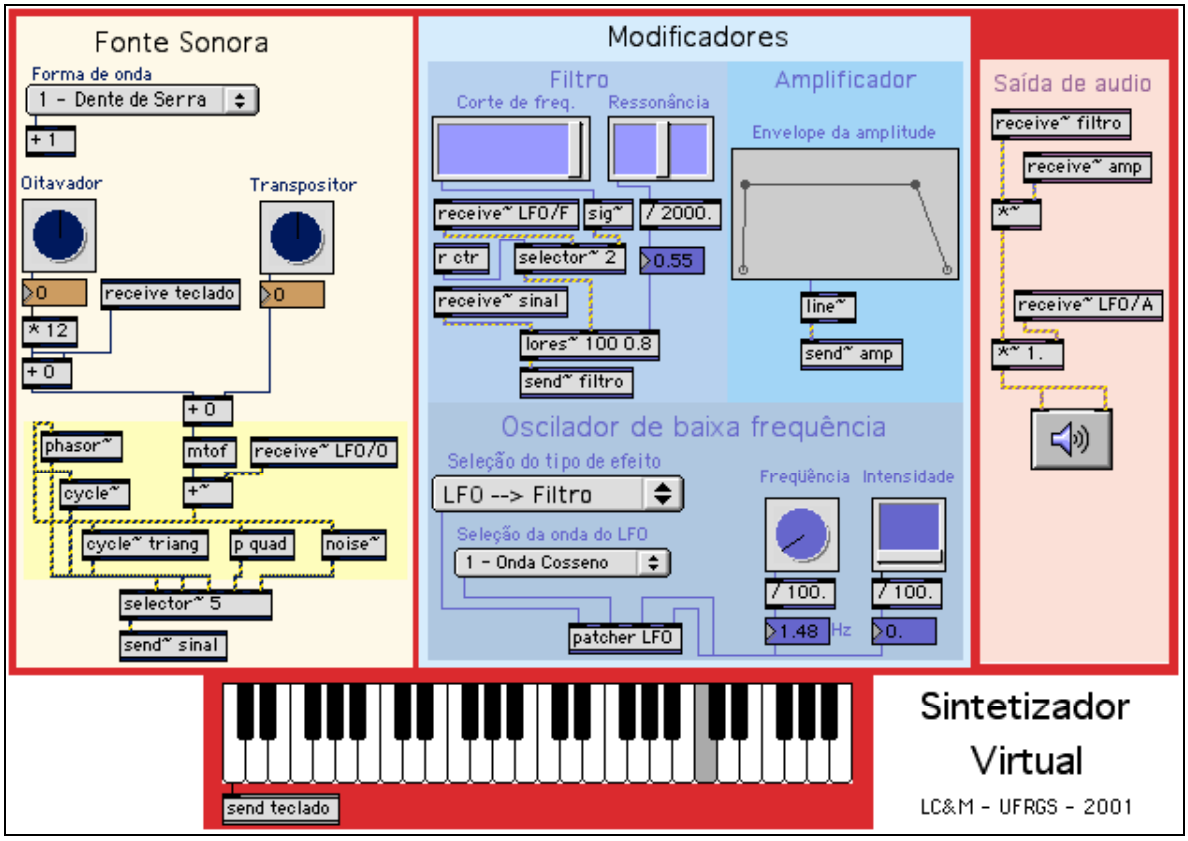


Figura 186 – Resposta do exercício com Sintetizador Subtrativo

7.3.3 AULA 3 - Programando Técnicas de Síntese Sonora

Para programar processos de síntese sonora é necessário um conhecimento teórico sobre as diversas técnicas de síntese. A figura 187 apresenta um patch explicativo sobre algumas técnicas mais conhecidas e que podem ser implementadas com facilidade através do MSP. Clique no menu e escolha a técnica de síntese desejada, leia a explicação e depois ouça e observe o osciloscópio que mostrará graficamente o comportamento das ondas sonoras.

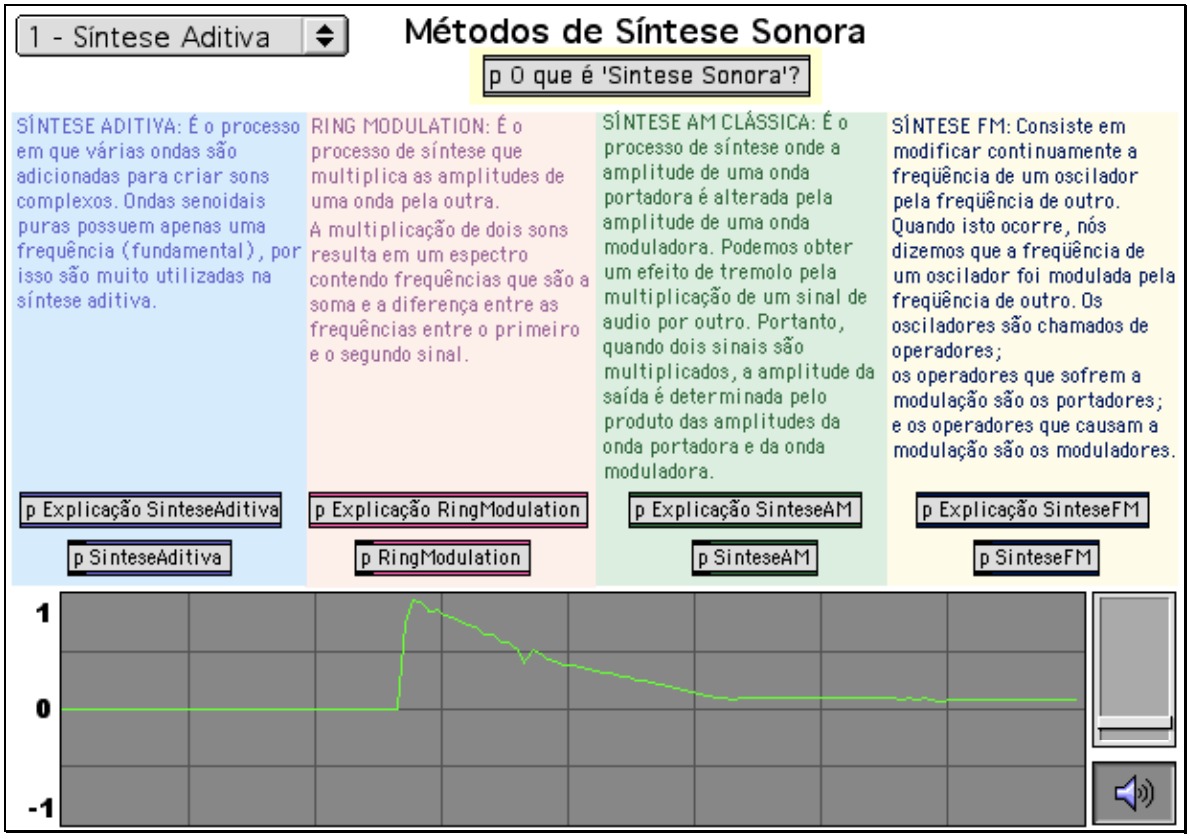


Figura 187 – Métodos de Síntese Sonora

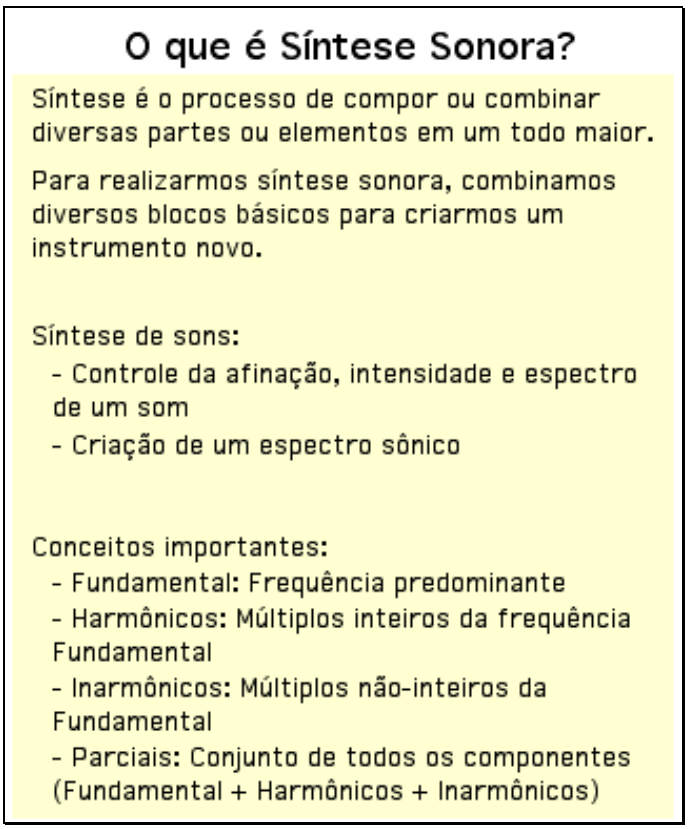


Figura 188 - Conceitos Básicos para Síntese Sonora

7.3.3.1 Síntese Aditiva

Síntese aditiva é o processo em que várias ondas simples são adicionadas para criar sons complexos. Ondas senoidais puras possuem apenas uma freqüência (a fundamental) e são muito utilizadas na síntese aditiva.

Ondas sonoras diferem-se por sua freqüência e por sua amplitude. Um agrupamento de ondas senoidais chama-se **espectro.** Ao examinarmos este espectro sonoro, chamamos a onda senoidal cuja freqüência é a mais baixa de **fundamental**. Esta é a freqüência que reconhecemos como o tom do som – ondas senoidais puras possuem apenas uma freqüência (a fundamental) e por isso são muito utilizadas na síntese aditiva. A partir desta onda fundamental, dizemos que todas as ondas cujas freqüências são múltiplas matemáticas desta fundamental são **harmônicas**, e que todas as ondas restantes, são as **parciais**. Para que os sons complexos tenham um timbre interessante, a amplitude das parciais deve mudar com um certo grau de independência.

Ao clicar no patch Síntese Aditiva um exemplo detalhado é apresentado. Para exemplificar a síntese aditiva, na figura 189, o objeto **function~** permite que o usuário construa graficamente formas de controle tais como envelopes de amplitude. Quando o objeto **function~** recebe um *bang*, ele envia ao objeto **line~** a forma gráfica traçada pelo usuário para a geração de um sinal correspondente.

Observe na figura 189 que a síntese aditiva implementada neste exemplo possui um controle de envelope da amplitude, um objeto **kslider** é usado para selecionar a freqüência e está ligado a 4 osciladores que geram as parciais. Estas parciais são amplificadas por objetos ***~** e a soma de todas elas é multiplicada por 0,25 para manter o volume no limite e evitar distorções.

Observe na figura 189 que a síntese aditiva é exemplificada por 4 ondas que são somadas para gerar um som mais complexo.

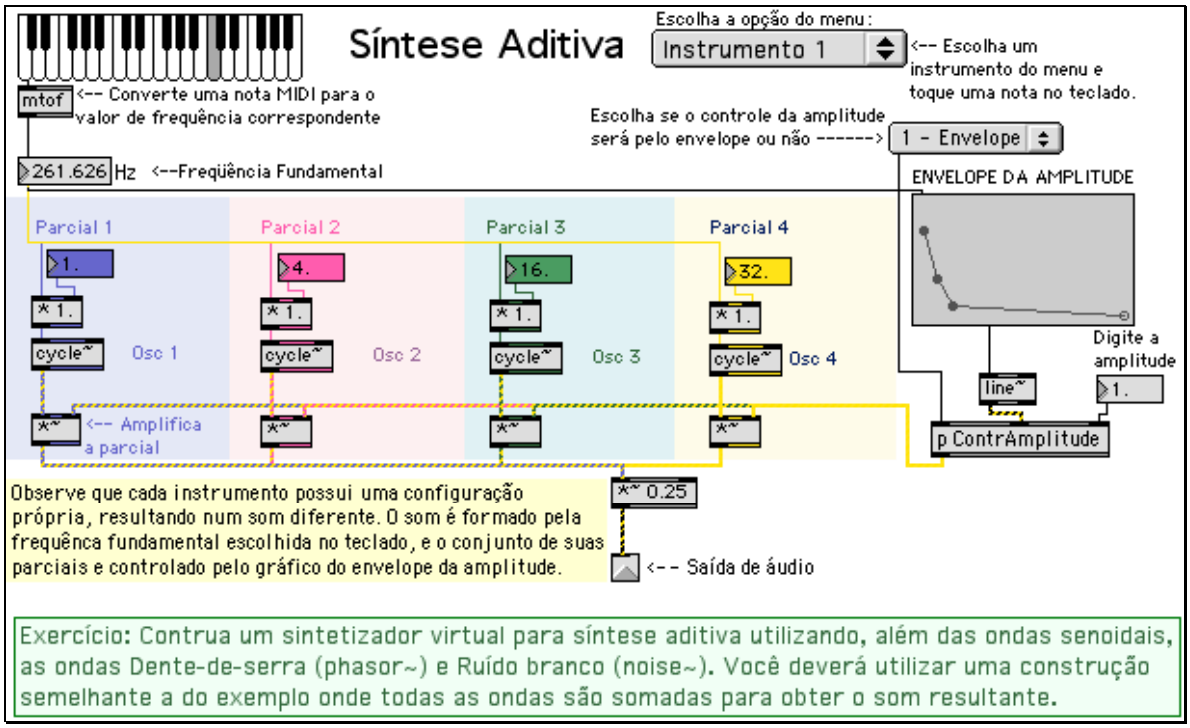


Figura 189 – Criando sons com Síntese Aditiva

Na resposta do exercício (figura 190) foram usados os mesmos princípios do exemplo da figura 181, onde quatro parciais geradas por objetos (agora diferentes) são controlados por envelopes de amplitude.

A parcial 1 difere das parciais 2 e 3 já que estas utilizam o objeto **phasor~,** enquanto que a primeira utiliza um objeto **cycle~**. A parcial 4 foi implementada utilizando um objeto **noise~**. Este objeto é a fonte sonora para a produção do ruído branco.

Neste exercício revisamos a utilização do objeto **preset**, que salva todas as configurações modificáveis pelo usuário e que são visíveis na tela. Por exemplo, todos os valores das caixas numéricas e as configurações dos envelopes são guardados em um dos "quadrinhos" que o objeto possui. Para carregar uma configuração, basta clicar no "quadrinho" que ela foi armazenada.

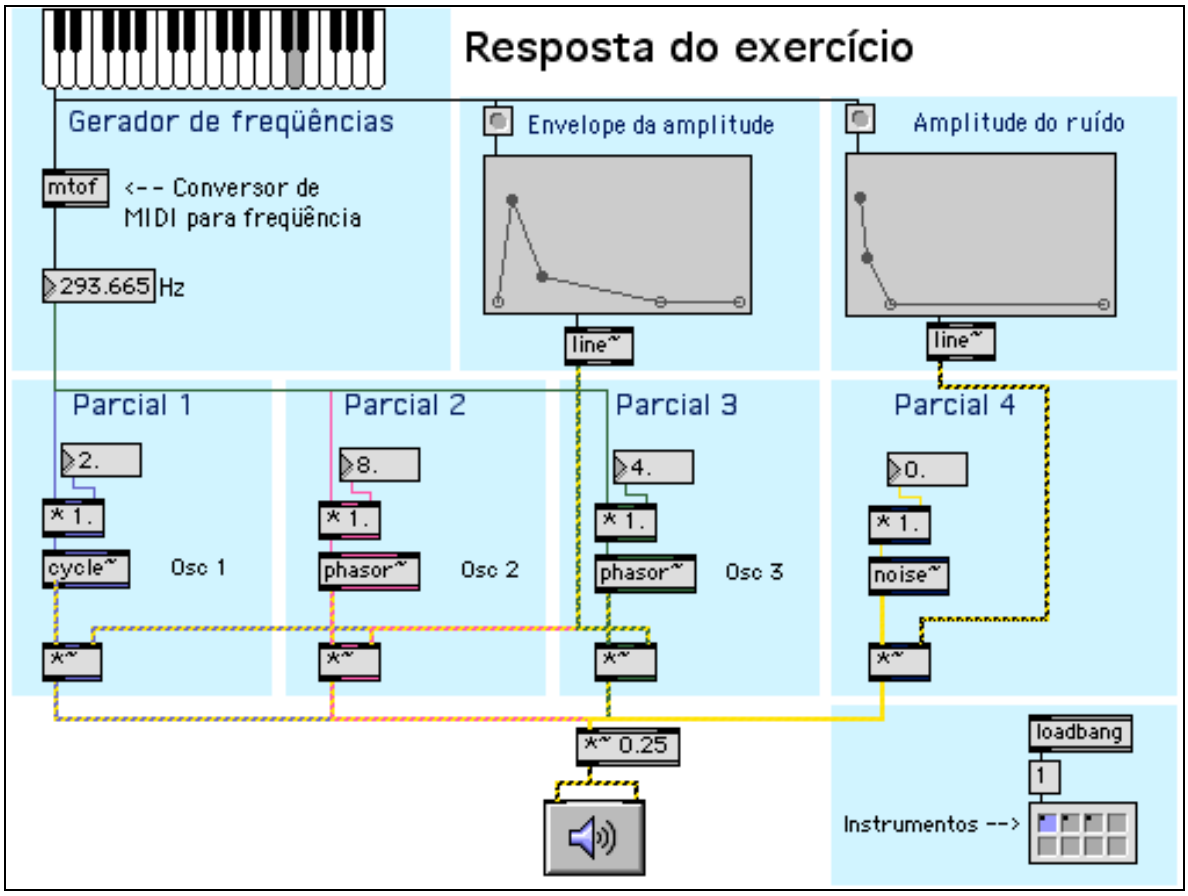


Figura 190 – Resposta do exercício de Síntese Aditiva

7.3.3.2 Ring Modulation

A multiplicação das amplitudes de cada amostra de uma onda pela outra cria um efeito conhecido como ring modulation. A ring modulation é um tipo de modulação de amplitude. Ao clicar no pach Ring Modulation da figura 191, um exemplo detalhado aparece.

O exemplo da figura 192 apresenta dois objetos cycle~ com seus sinais de saída multiplicados por um *~. Este sinal é enviado a um amplificador onde é amplificado de acordo com as configurações definidas pelo envelope de amplitude no topo da figura 192.

Procure utilizar o patch de Síntese Ring Modulation, estude como foi realizada a programação e após resolva o exercício proposto.

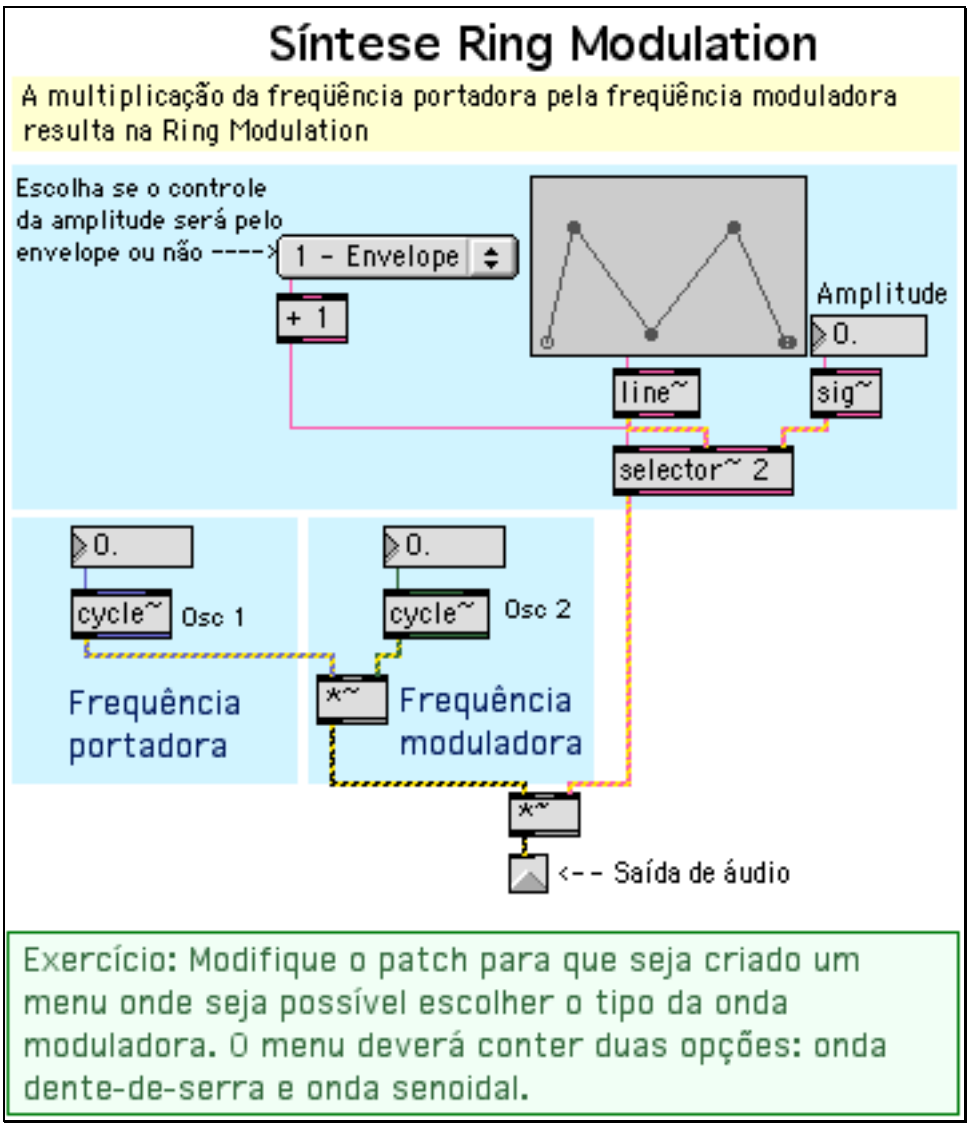


Figura 191 – Ring Modulation

A solução para o exercício pode ser vista na figura 192. Ao unirmos o patch da figura 180 com o da figura 191 poderemos obter outros resultados sonoros. A síntese Ring Modulation é obtida ao realizarmos a multiplicação dos sinais portadores pelo sinal modulador.

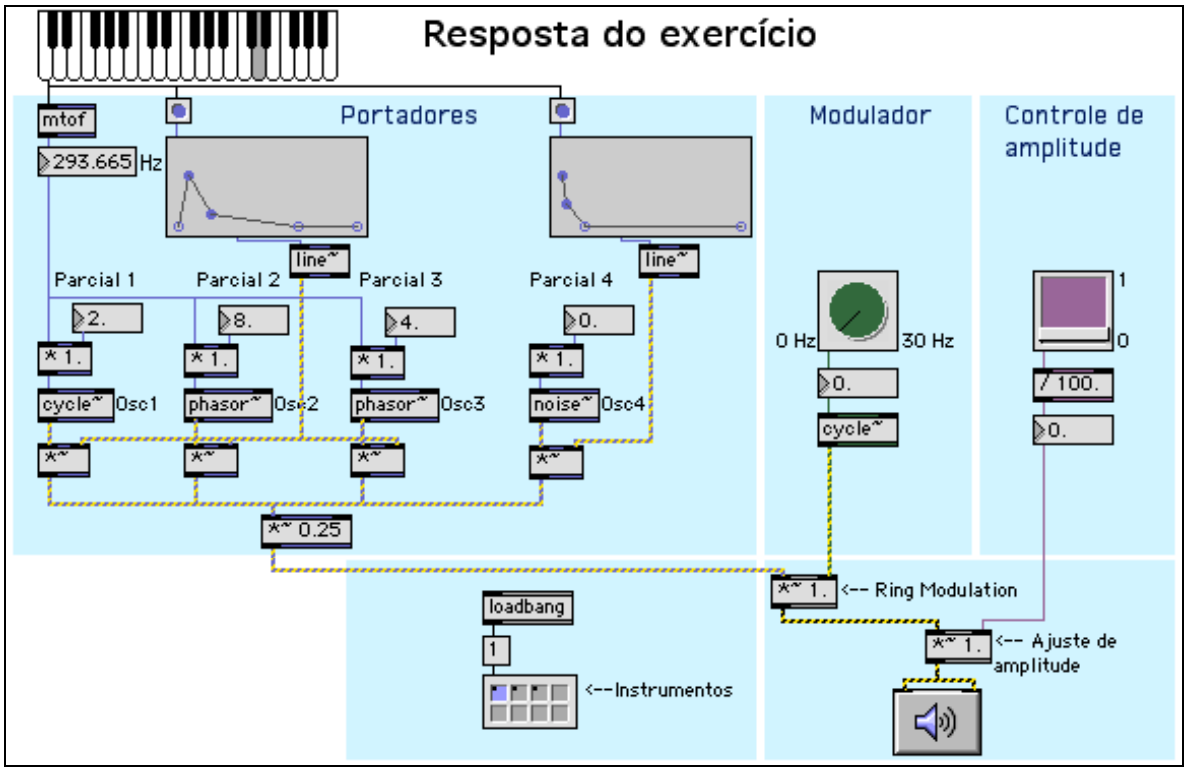


Figura 192 – Resposta do exercício de Ring Modulation

7.3.3.3 Modulação de Amplitude

Na síntese AM, a amplitude de uma onda portadora (*carrier*) é alterada pela amplitude de uma onda moduladora (*modulator*). A multiplicação de um sinal de áudio por um sinal de sub-áudio resulta em flutuações regulares da amplitude conhecida como trêmolo. A multiplicação de sinais cria *sidebands* que são freqüências que não estão presentes nas ondas originais. Isso pode criar ondas complexas com as freqüências harmonicamente relacionadas. Quando dois sinais são multiplicados, a amplitude da saída é determinada pelo produto das amplitudes da onda portadora e da onda moduladora. A figura 193 demonstra este conceito. Realize experimentos com o path Sintese AM para obter diferentes sonoridades e estude a programação, após procure solucionar o exercício proposto.

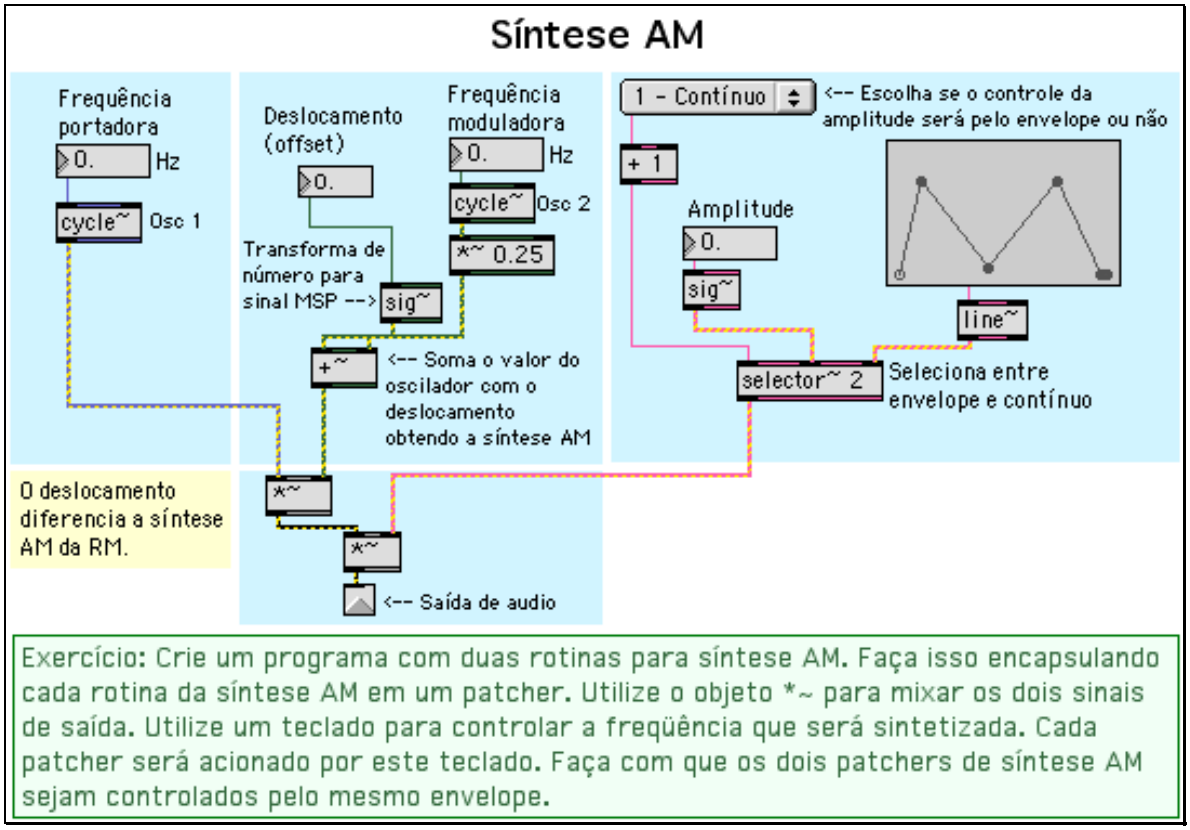


Figura 193 – Criando sons com Síntese AM

A resposta para o exercício de síntese AM possui, na tela principal (figura 194), um objeto **kslider~** gerando notas MIDI e um objeto **mtof~** transformando estes valores MIDI em sinais de áudio.

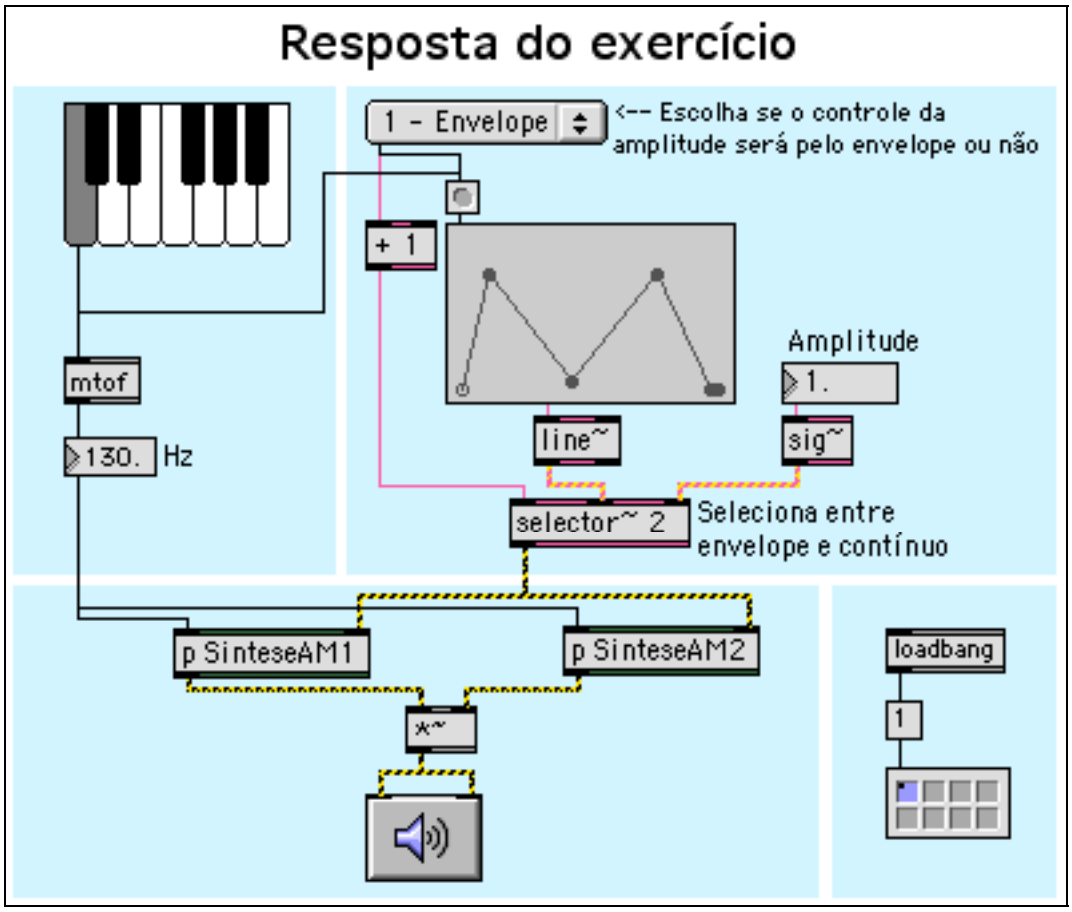


Figura 194 – Resposta do exercício de Síntese AM

Além disso podemos ver um conjunto de objetos compostos por um envelope de amplitude e dois objetos **patch** que são, na verdade, dois sub-patches iguais (figura 195). Estes sub-patches possuem dois geradores de ondas – objetos **cycle~** - sendo que um deles sofre uma adição, o *offset* ou *deslocamento*. Os sinais que saem destes sub-patches são multiplicados e geram o efeito criado por síntese AM.

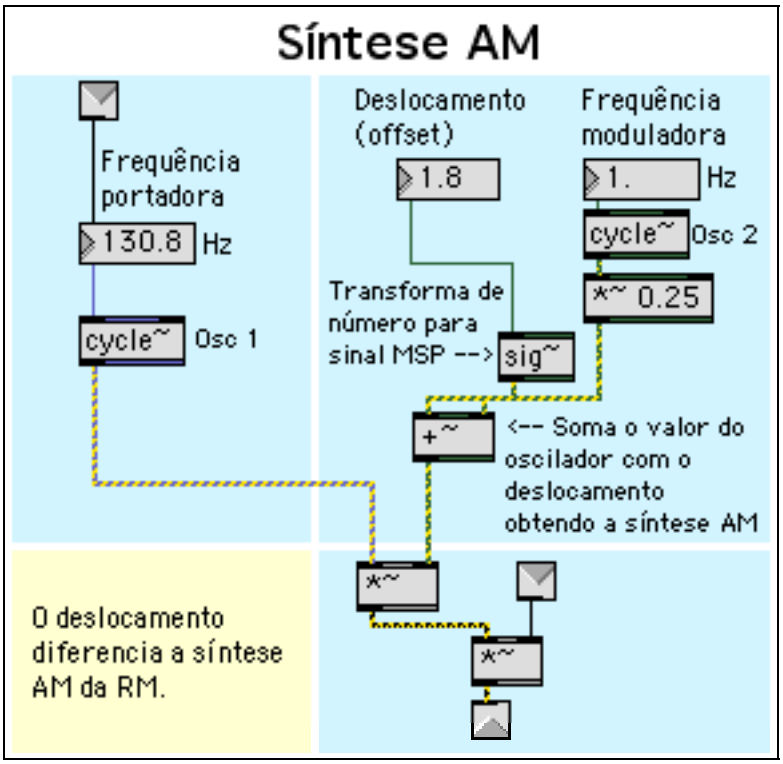


Figura 195 - Subpatch da resposta do exercício de Síntese AM

7.3.3.4 Síntese FM

No início da década de 80, os dois métodos mais comumente utilizados para gerar timbres complexos eram a síntese aditiva, que consistia na adição de ondas senoidais, e a síntese subtrativa – utilização de filtros para "esculpir" sons ricos em harmônicos. Entretanto, ambos os métodos existentes apresentavam problemas - a síntese aditiva necessitava de muitos parâmetros e a síntese subtrativa produzia uma quantidade limitada de timbres. A síntese por freqüência modulada, conquistou uma grande fatia dos músicos quando se tornou disponível, pois necessitava apenas de alguns parâmetros para sintetizar.

A síntese FM consiste em modificar continuamente a freqüência de um oscilador pela freqüência de outro. Quando isto ocorre, nós dizemos que a freqüência de um oscilador foi modulada pela freqüência de outro – toda síntese FM é baseada nessa relação de freqüência entre dois osciladores. De acordo com a terminologia da síntese FM, osciladores são chamados de operadores; o operador que sofre a modulação é o portador e o operador que causa a modulação é o modulador. O parâmetro que representa a quantidade de modulação é chamado de índice de modulação e é representado na forma de proporção (modulador de amplitude / modulador de freqüência).

As variações na freqüência de uma forma de onda básica, criadas pela modulação de freqüência, nos permitem gerar um espectro harmônico muito rico com um número muito pequeno de osciladores senoidais – a criação de um espectro tão complexo com síntese aditiva necessitaria de um vasto número de osciladores, já que cada parcial necessitaria de seu próprio oscilador. Adicionando mais e mais operadores para a configuração básica de dois osciladores, também se aumenta o número de possíveis conexões entre os operadores. Diferentes métodos de conexão são chamados de algoritmos na linguagem de síntese FM. Como cada operador pode ter seu próprio

envelope e índice de modulação, um número aparentemente ilimitado de possibilidades se abre para os sintetistas.

Ao clicar no patch "Síntese FM" da figura 187, um exemplo detalhado aparece. Na figura 196 é exemplificada a forma mais simples de síntese FM com apenas um operador constituído por uma onda portadora e uma onda moduladora. Procure ouvir os sons produzidos e compare com os outros tipos de síntese já vistos.

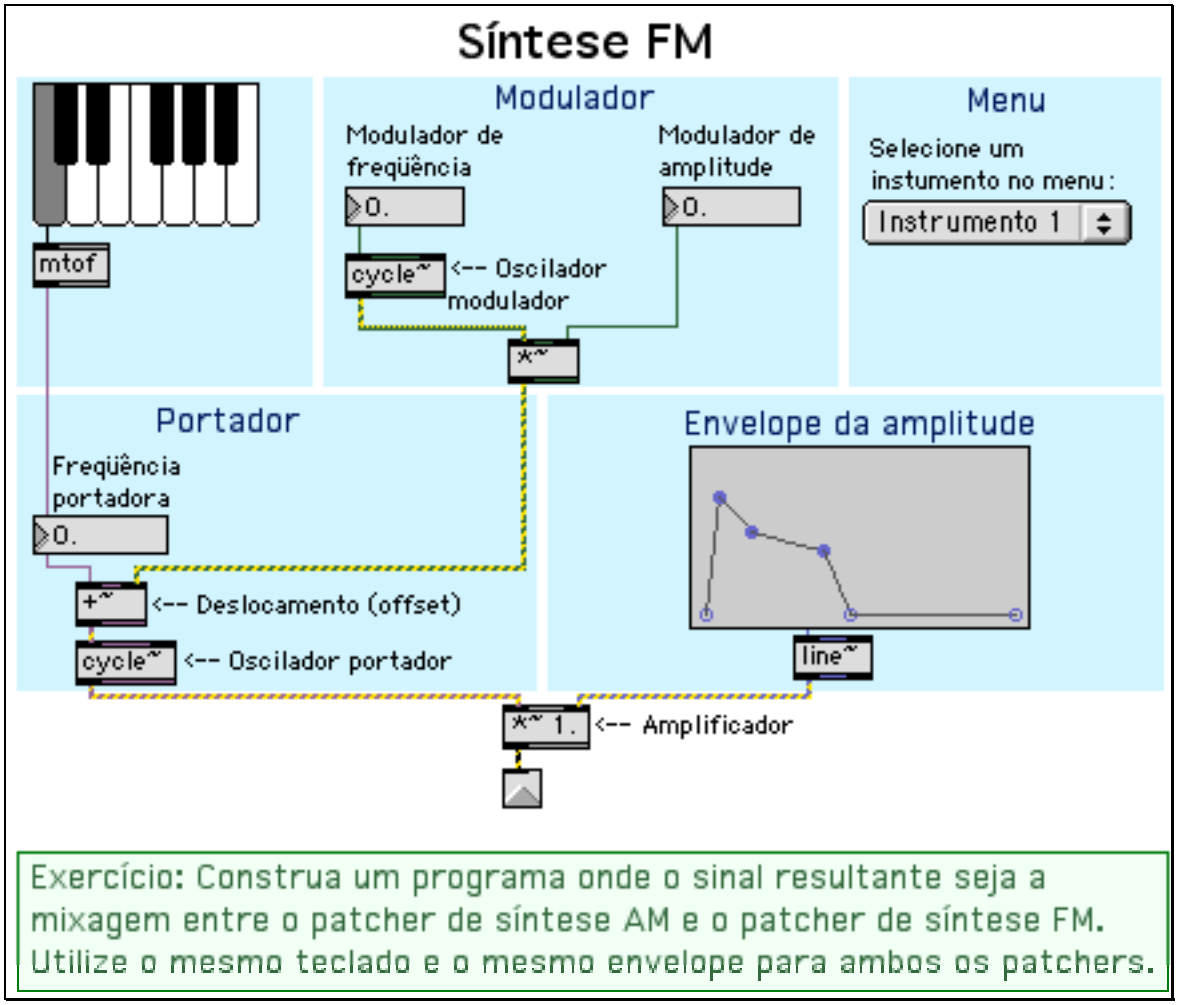


Figura 196 – Criando sons com Síntese FM

A figura 197 apresenta apenas a interface visual, facilitando o controle pelo usuário e deixando a parte funcional escondida em sub-patches. Note que a subpatch “p Síntese AM” é a mesma subpatch implementada na resposta do exercício anterior.

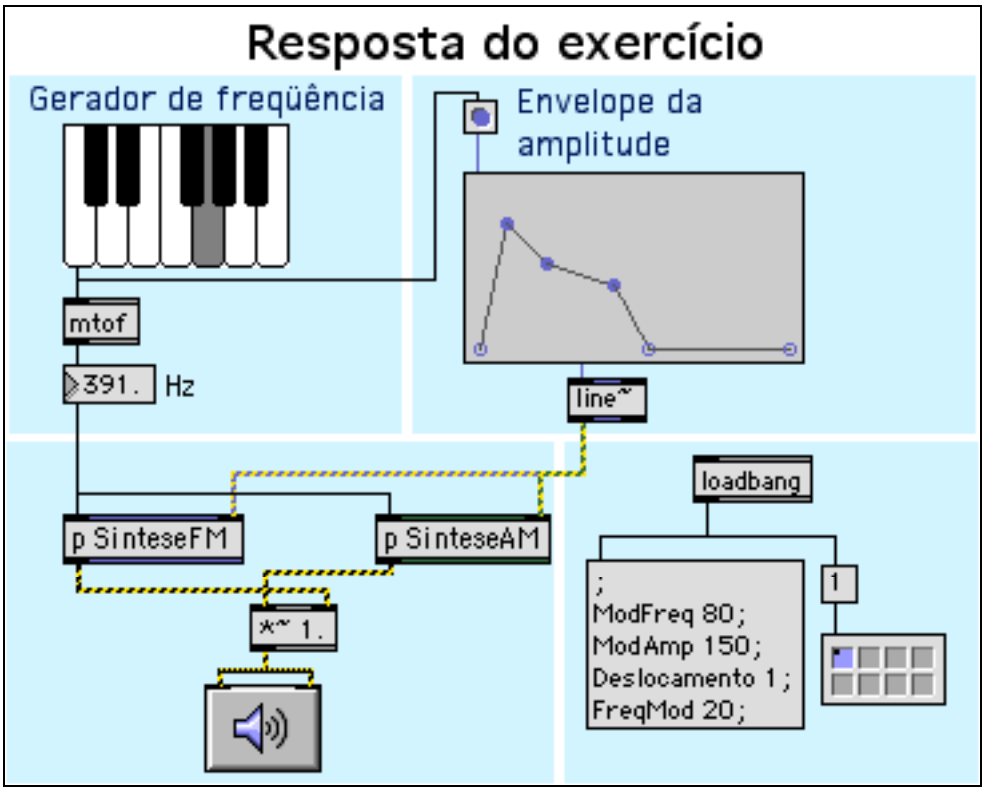


Figura 197 – Resposta do exercício de Síntese FM

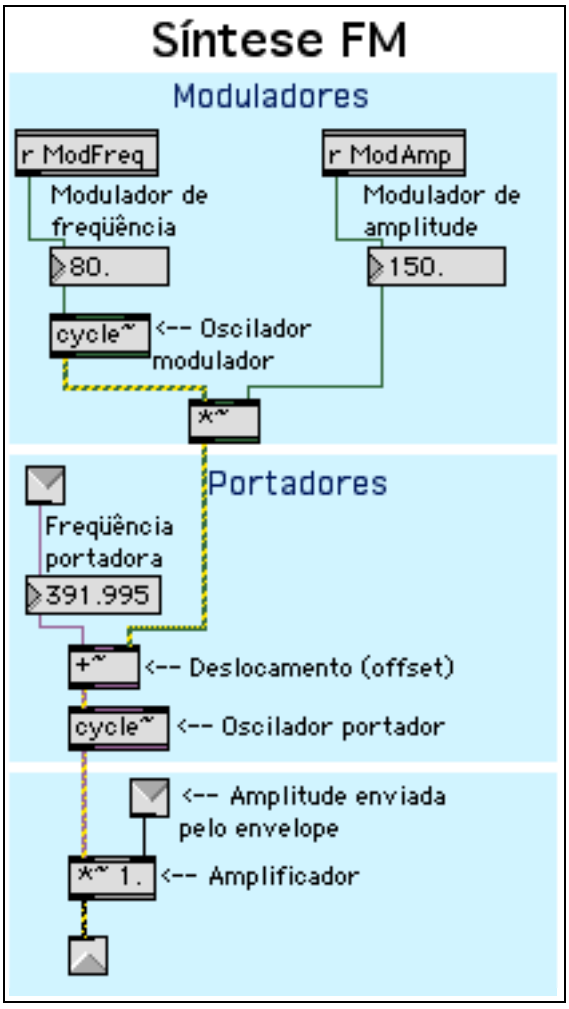


Figura 198 – Subpatch da resposta do exercício de Síntese FM

O subpatch da figura 198 é um objeto no patch principal. Ele possui *inlets* e *outlets* como qualquer outro objeto. Cabe aqui relembrar que sinais são recebidos em seus *inlets* no patch principal através de um objeto chamado justamente de **inlet,** e o envio de um sinal à sua saída, ao seu *outlet,* ocorre através de um objeto **outlet**. Ainda, são recebidos dois valores (neste caso numéricos) pelos objetos **receive**, abreviados por **r**. Os valores são enviados através da mensagem no canto inferior direito do patch principal. Isto é feito simplesmente pela menção de um nome ou de uma variável seguida de seu valor – por exemplo, "ModFreq 80".

As operações realizadas no subpatch são as mesmas efetuadas na figura 196 e baseadas nos princípios de Síntese FM. Um sinal de áudio gerado por um oscilador - modulador de freqüência - é multiplicado por um número que representa o modulador de amplitude, este sinal é somado à freqüência portadora e o resultado é enviado a um oscilador que gera o som final.

7.3.4 AULA 4 – Sintetizador FM

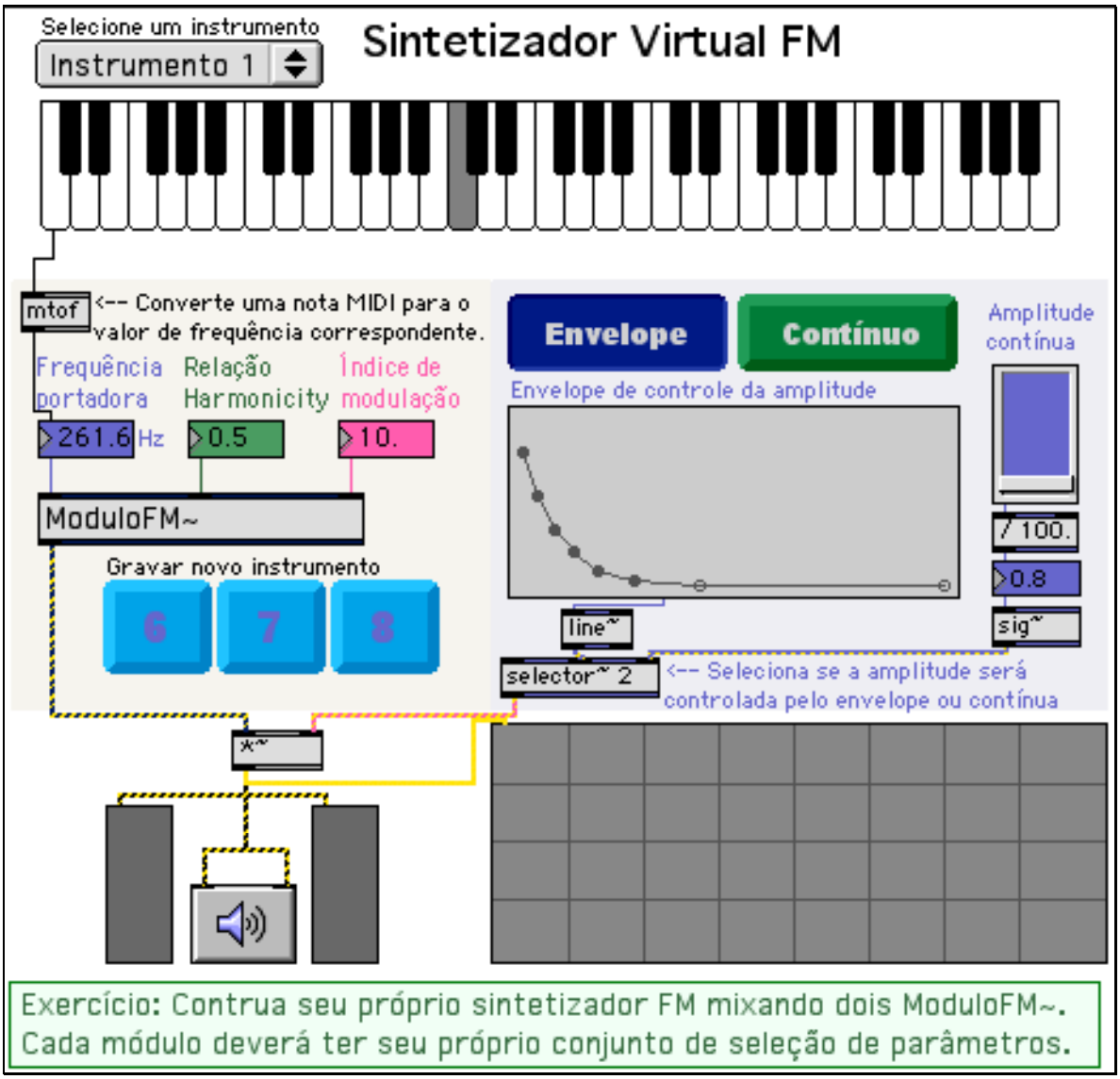


Figura 199 – Sintetizador Virtual FM

O Sintetizador FM Virtual é apresentado na figura 199. O ModuloFM~ é o "núcleo" do sintetizador: Os valores dos moduladores de amplitude (A_m) e de freqüência (F_m) são calculados basendo-se nos argumentos de entrada (freqüência portadora (F_c), relação harmonicity e índice de modulação).

A relação harmonicty, que é a razão entre o modulador de freqüência e a freqüencia portadora, é multiplicada, no **ModuloFM~** (figura 200), pela freqüência portadora e com isso obtemos o modulador de freqüência. Observe abaixo a representação matemática deste processo:

$$\frac{F_m}{F_c} \times F_c = F_m$$

De forma semelhante, o índice de modulação, que é a razão entre o modulador de amplitude e o modulador de freqüência, é multiplicado pelo modulador de freqüência

obtido no passo anterior, e desta forma obtemos o modulador de amplitude. Observe abaixo a representação matemática deste processo:

$$\frac{F_a}{F_m} \times F_m = F_a$$

Na figura 199, a função dos botões 'Envelope' e 'Contínuo' é permitir que seja possível escolher de que maneira a amplitude irá se comportar num transcurso do tempo: se segundo o gráfico do envelope, ou se permanecerá contínua indefinidamente. Neste segundo caso, é possível controlar o nível da amplitude utilizando o slider para controle da 'Amplitude contínua'.

Ao utilizar o Sintetizador FM Virtual para alterar os parâmetros do **ModuloFM~** e do envelope, você poderá obter timbres de seu agrado. Para armazená-los na memória utilize os botões numerados 6, 7, e 8. Basta clicar em um deste para gravar as configurações do timbre que você acabou de sintetizar. Para utlizar este som, posteriormente, bastará utilizar o menu de seleção de instrumento.

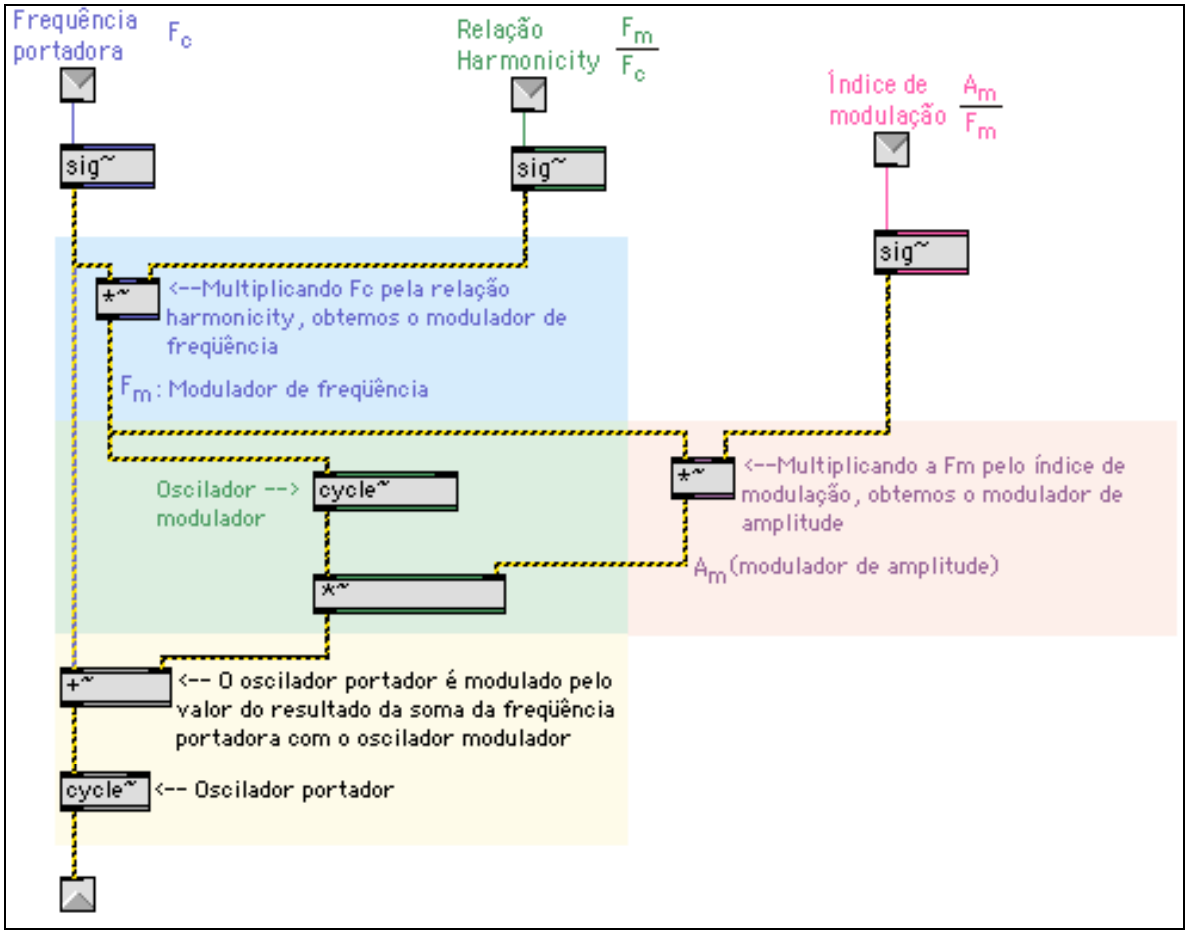


Figura 200 – Módulo FM

O esquema básico para resolver o exercício proposto (figura 201) é mixar dois blocos de síntese FM.

Em cada bloco, teremos um **ModuloFM~** e seus respectivos parâmetros. Além disso, adicionamos um envelope para controle da amplitude e também um objeto ***~** para ajustar amplitude do sinal do ModuloFM de acordo com o envelope.

Adicionalmente, devemos dividir a saída do *~ pelo número de blocos de síntese FM que tivermos no programa (nesse caso, por 2) para evitarmos distorções na saída sonora.

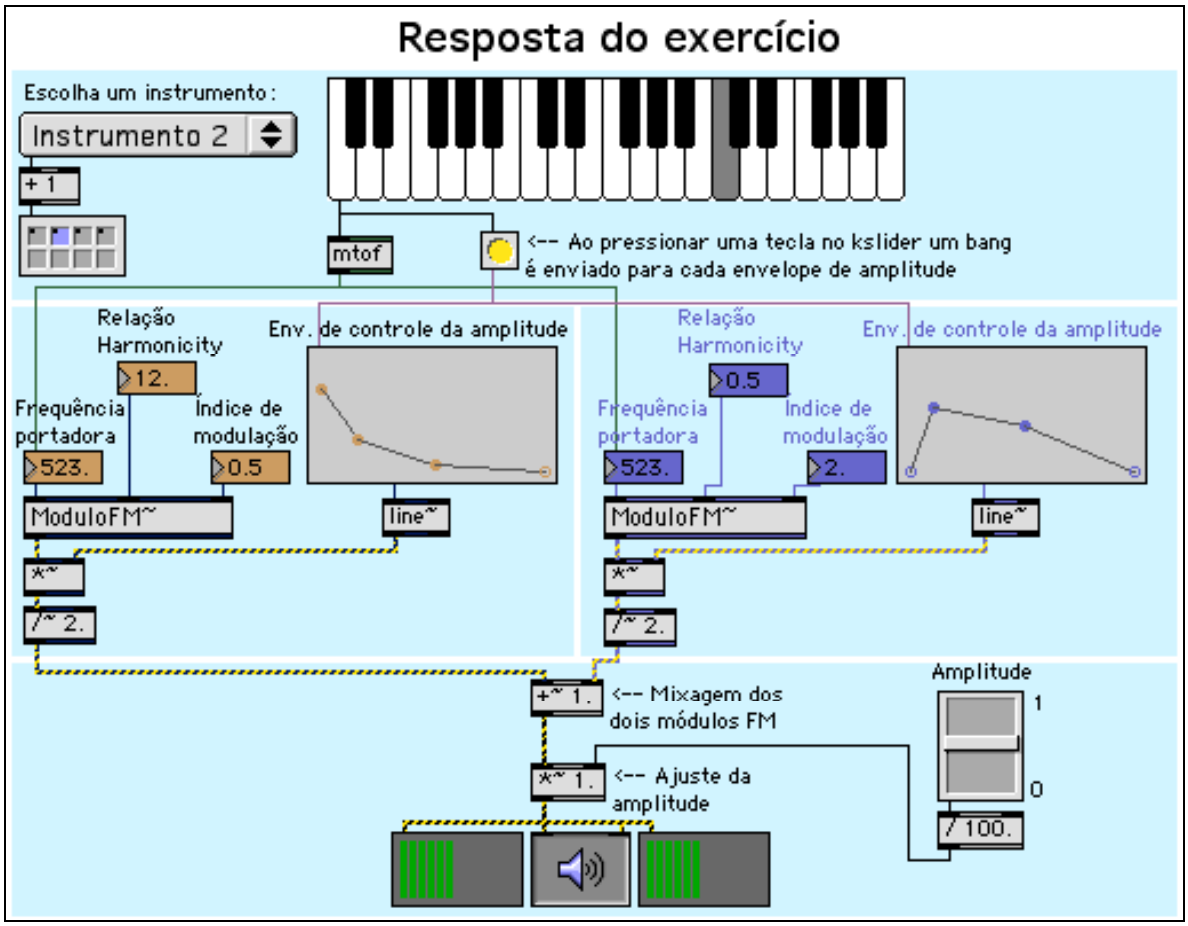


Figura 201 – Resposta do exercício com Sintetizador FM

7.3.5 AULA 5 - Amostragem Digital

A gravação de sons externos pode ser realizada conectando um microfone à placa de som do computador. O som pode ser capturado, armazenado, modificado e tocado pelo computador

Na figura 202 é apresentado um exemplo de sampler virtual. O som enviado ao computador entra no MSP através do objeto **adc~**. O objeto **record~** salva este som – ou qualquer outro sinal – em um objeto **buffer~**. É possível gravar no **buffer~** inteiro, ou em uma determinada porção deste especificando uma posição inicial e outra final nos *inlets* mais à direita do objeto **record~**. Para ouvir o conteúdo de um buffer são utilizados os objetos **count~** e **index~** que tocam o conteúdo sonoro na atual velocidade de amostragem. Pode-se, também, utilizar os objetos **line~** e **play~** para alterar a velocidade em que a música será tocada, e/ou ler o conteúdo do **buffer~** em ambas as direções. Observe bem o exemplo da figura 202, tente utilizá-lo como está indicado e resolva o exercício proposto.

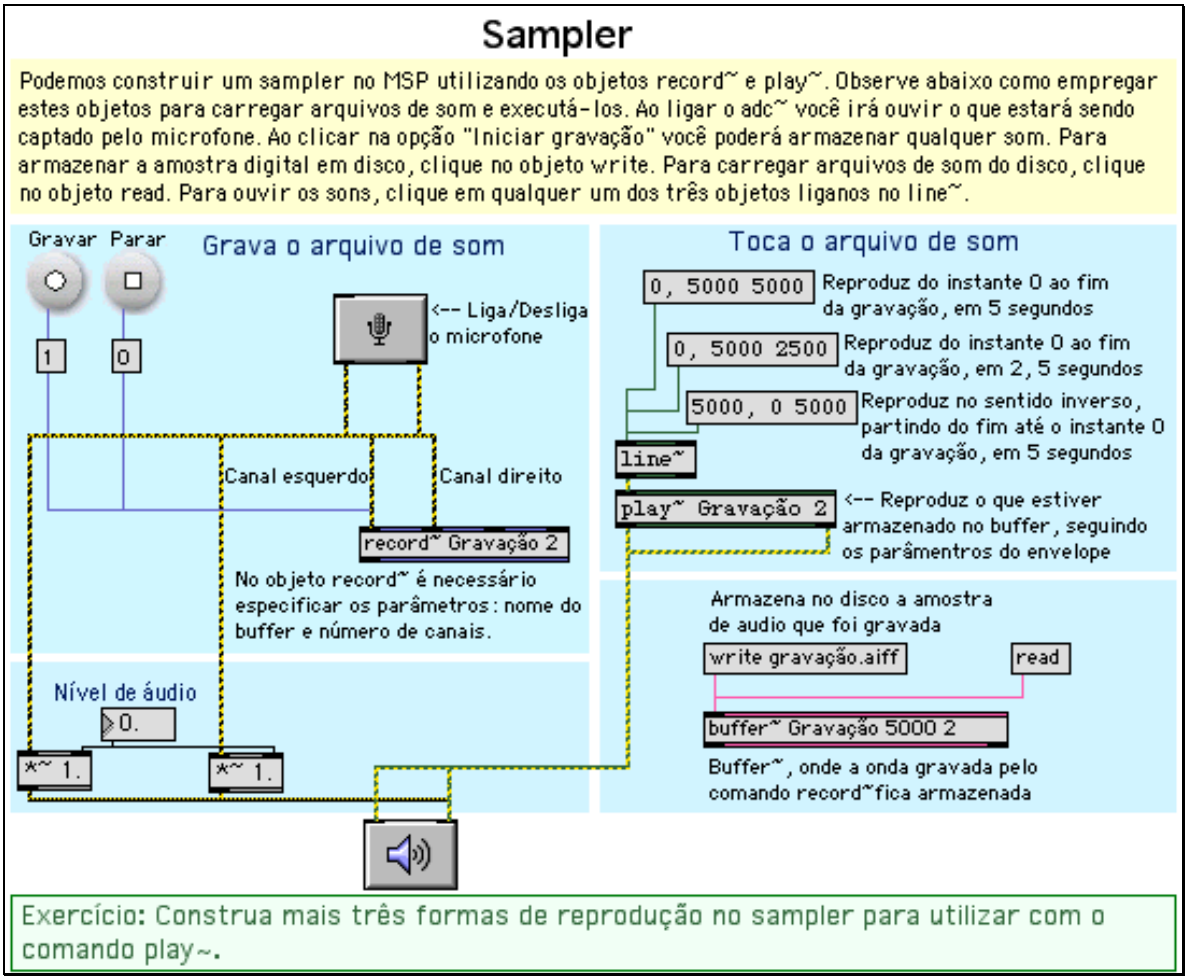


Figura 202 – Amostragem Digital: Sampler

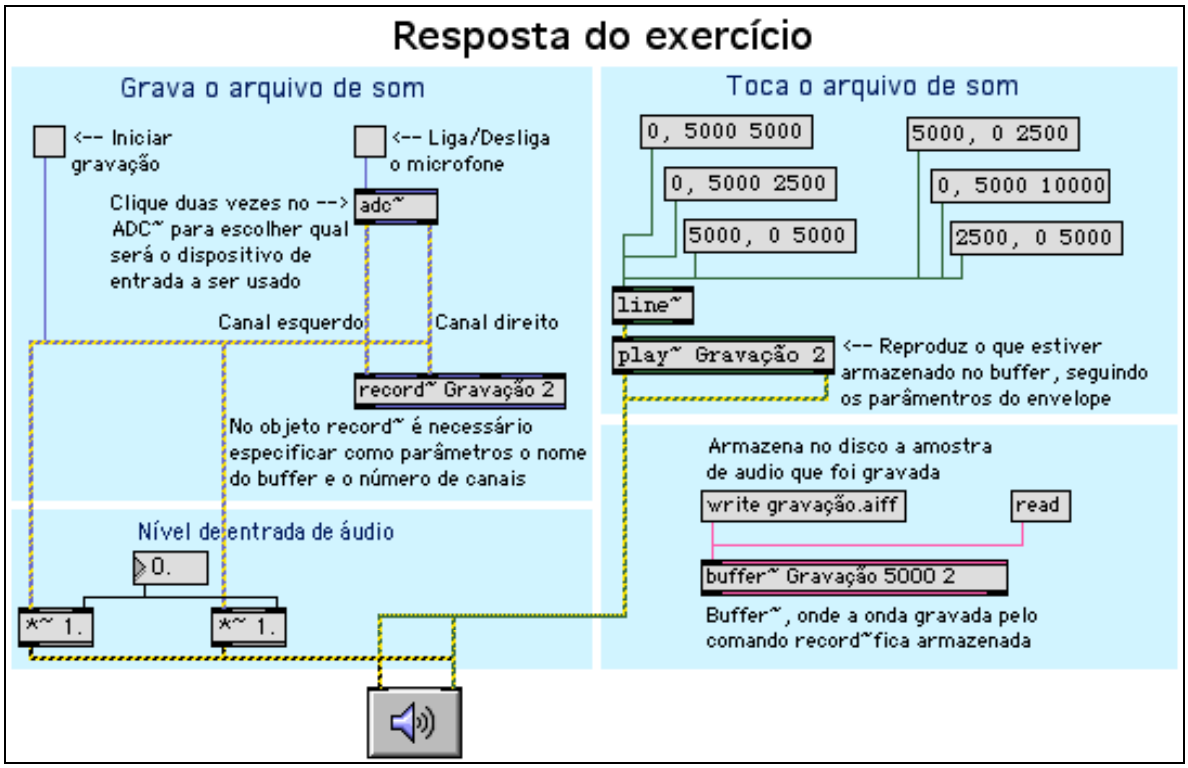


Figura 203 – Resposta do exercício do sampler virtual

7.3.6 AULA 6 - Processamento Digital de Efeitos

Atualmente os efeitos de áudio são processados digitalmente, usando-se equipamentos dotados de circuitos DSP (digital signal processor), que digitalizam o som original, manipulam-no por meio de recursos computacionais e convertendo o resultado novamente em som. A seguir apresentamos alguns dos efeitos mais utilizados na música.

7.3.6.1 O Efeito de Reverb

A reverberação é produzida pelas múltiplas reflexões do som em diversas direções, com diversos tempos de atraso. Em ambientes acústicos naturais, a reverberação se dá graças à reflexão do som em diversos pontos das diversas superfícies. Como as distâncias percorridas pelo som entre as superfícies são diferentes, a percepção do sinal refletido é difusa, não inteligível como no caso do eco, por exemplo.

A reverberação natural tem como característica a atenuação gradual das reflexões no decorrer do tempo (chamado de reverberation time). As reflexões mais próximas (rápidas) são chamadas de early reflections. Os processadores de efeitos criam reverberação somando ao sinal de áudio original (seco) diversas cópias dele com atrasos e intensidades diferentes (veja figura 204). A qualidade tonal da reverberação em um ambiente depende do tipo de material usado em suas superfícies, e em muitos processadores é possível ajustar esta qualidade ou coloração, filtrando freqüências na reverberação.

A reverberação é usada com o objetivo de criar ambiência ou profundidade ao som, produzindo uma sensação mais natural.

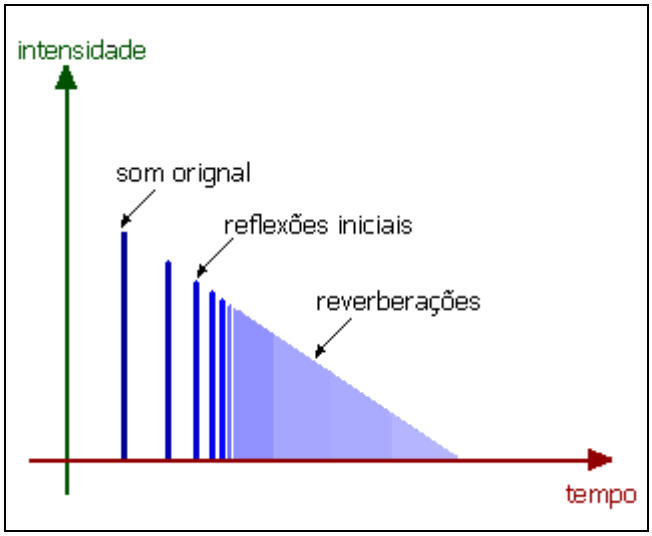


Figura 204 – Reverberação

7.3.6.2 Efeito de Delay

O delay (eco) é um efeito obtido pela soma do sinal de áudio original com uma cópia sua, atrasada, mas enquanto a reverberação é o resultado de diversas cópias, com diversos atrasos diferentes (que simulam as inúmeras reflexões), o eco caracteriza-se por uma ou mais reflexões, que ocorrem com atrasos determinados, que permitem ao ouvinte distinguir o som atrasado e percebê-lo claramente como um eco.

O tempo entre a ocorrência do som original (seco) e a primeira repetição é chamado de delay time e, assim como a reverberação, a repetição ou repetições do sinal ocorrem com amplitudes (intensidade) reduzindo-se gradualmente. Para criar as várias repetições ou ecos, os processadores (inclusive o MSP) usam um recurso em que o próprio eco é realimentado à entrada do processador, produzindo ecos do eco. Dessa forma, o número de repetições pode ser controlada pela quantidade de realimentação (feedback).

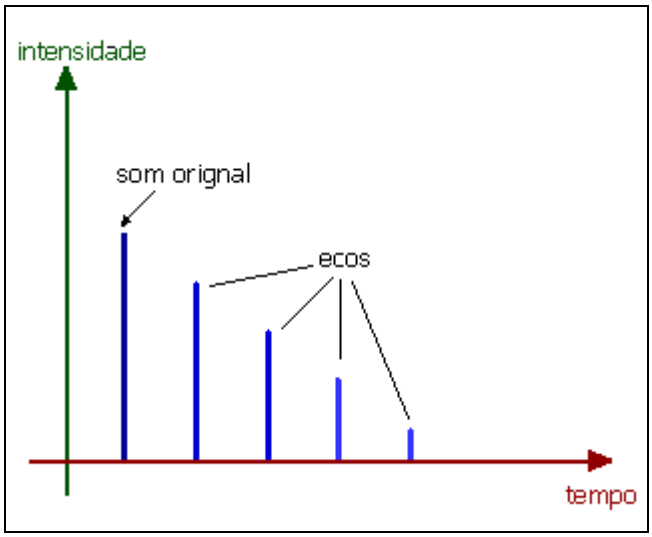


Figura 205 – Representação gráfica do efeito de delay

A figura 206 mostra um exemplo de programa para gerar um efeito de atraso utilizando o Msp. Observe o emprego dos objetos tapin~ e tapout~ e execute o programa utilizando o microfone como entrada do sinal de áudio. Após estudar o programa e compreender seu funcionamento, resolva o exercício proposto.

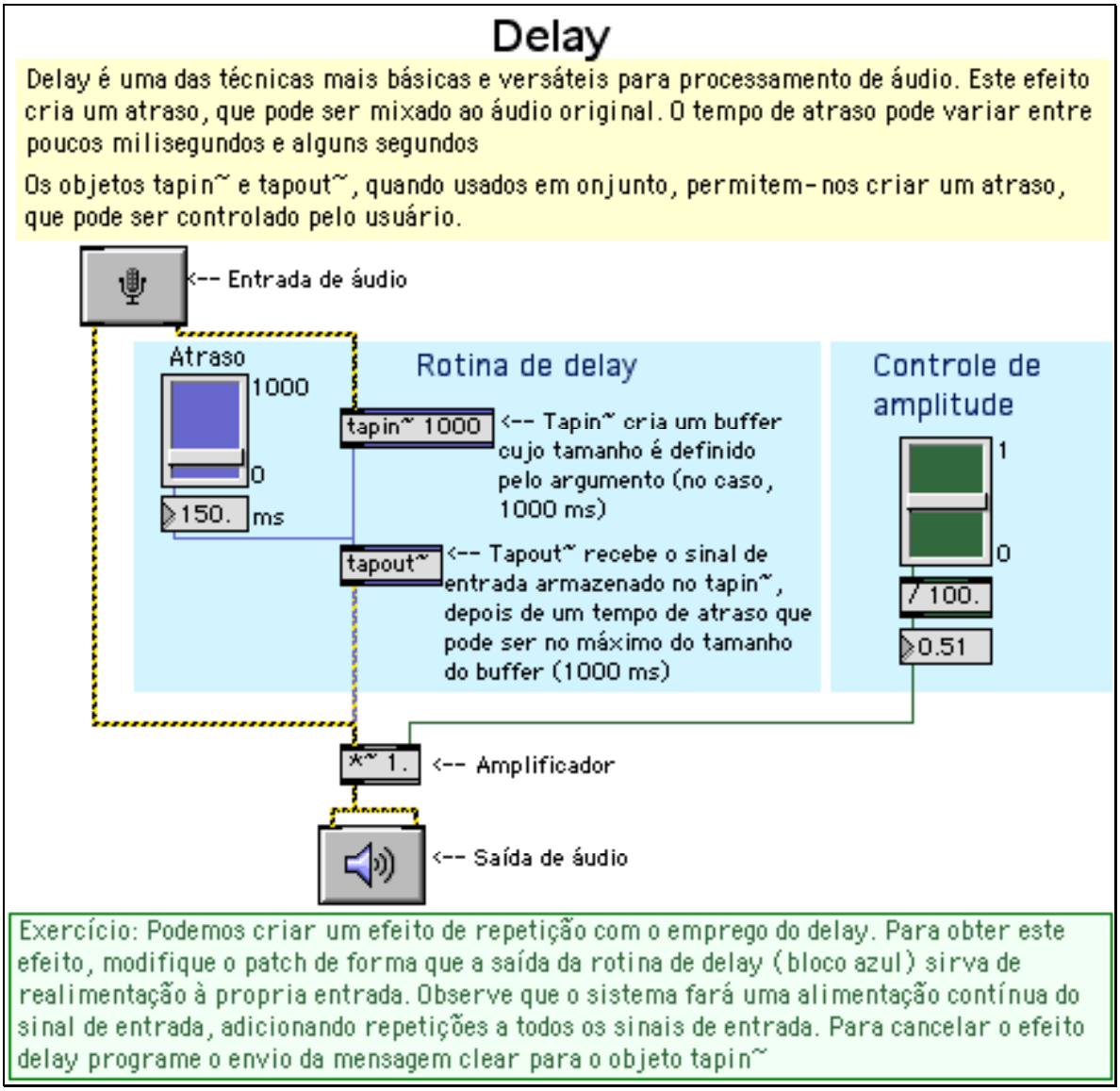


Figura 206 – Implementaçao do efeito de delay

A figura 207 mostra a solução do exercício sobre delay. Observe a resposta e compare com a sua.

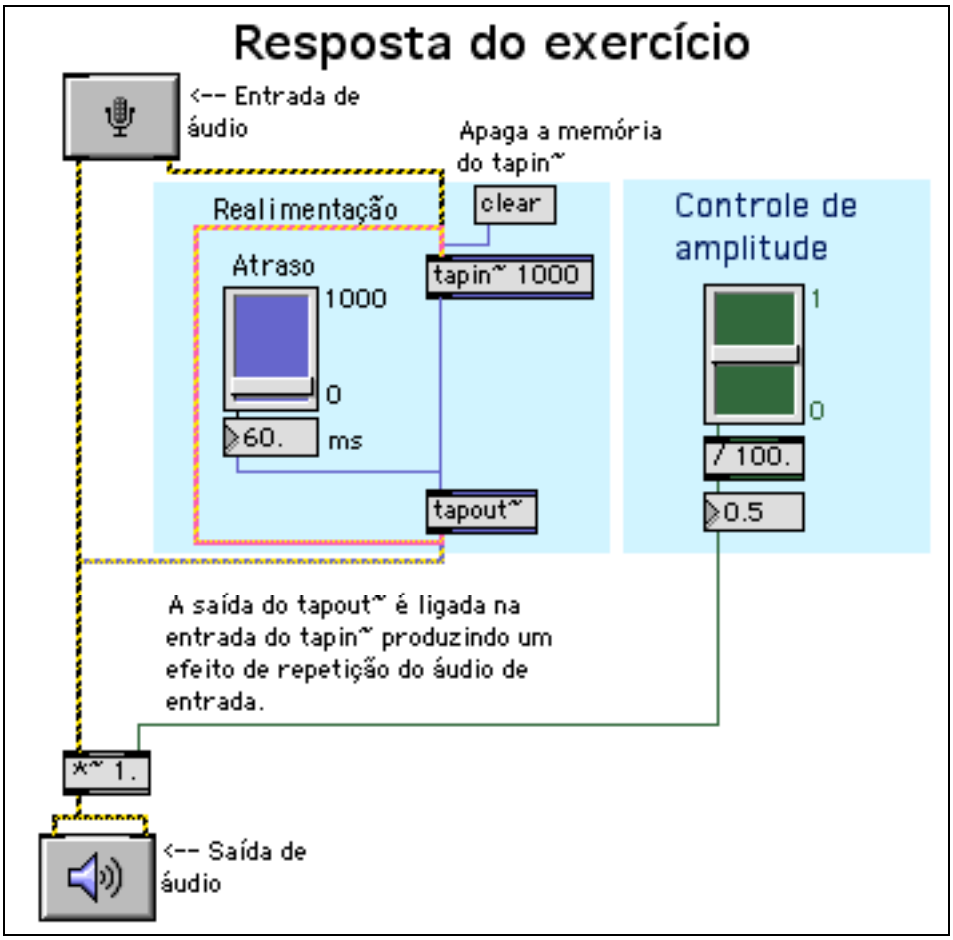


Figura 207 – Resposta do exercício para a programação de delay com repetições

7.3.6.3 Chorus

O Chorus é obtido somando ao som original uma cópia sua ligeiramente defasada, sendo que essa defasagem varia ciclicamente no tempo. Na maioria dos processadores, pode-se controlar a velocidade da variação cíclica da defasagem, por meio do parâmetro modulation speed, e também a magnitude da defasagem, por meio do parâmetro modulation depth.

A figura 208 apresenta um programa para a criação de um efeito de chorus. Utilize o microfone para entrada do áudio que será processado gerando o efeito de chorus. Após estudar o exemplo tente resolver o exercício proposto.

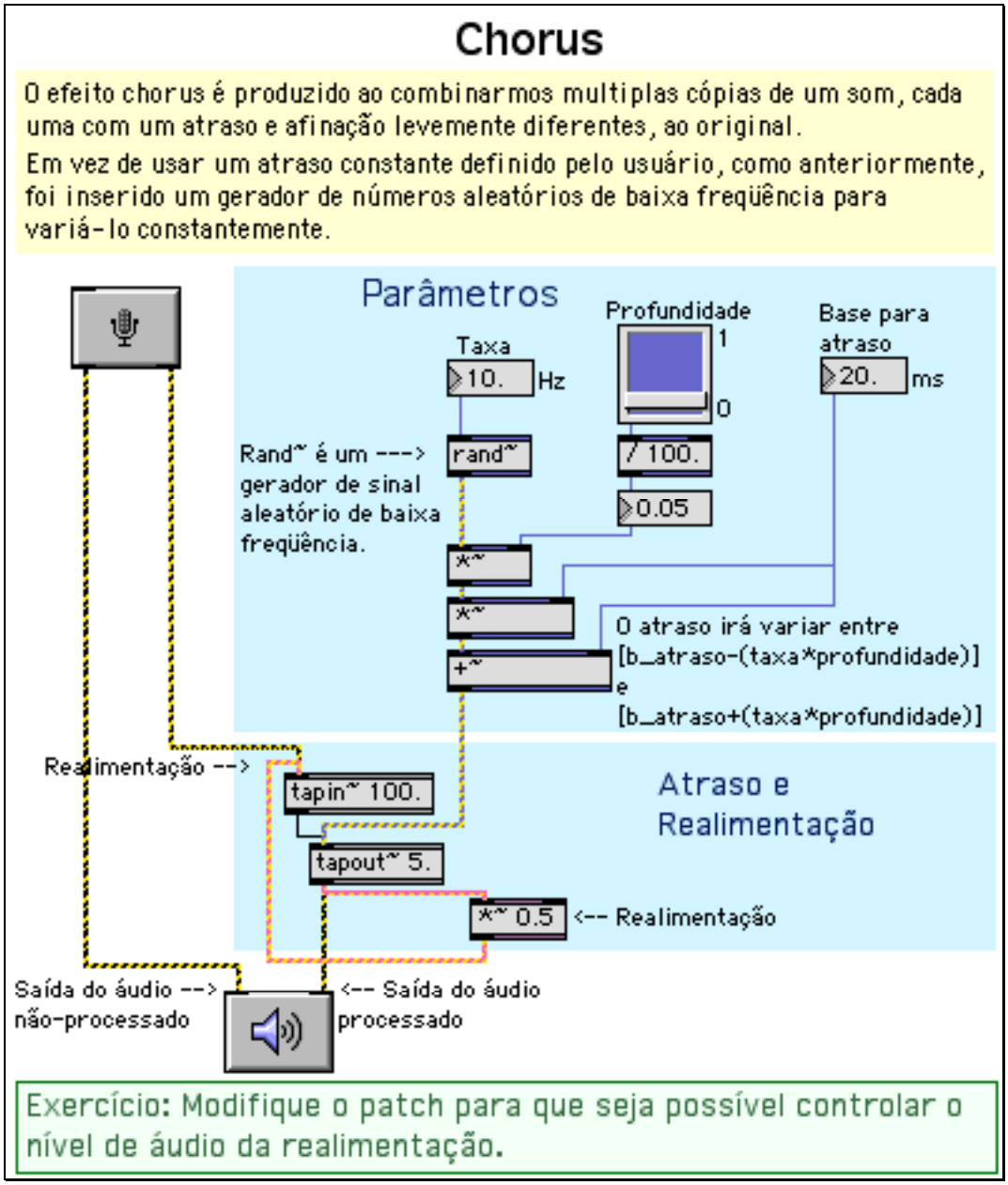


Figura 208 – Implementação do efeito de chorus

A figura 209 mostra como pode ser feito o controle do nível de áudio da realimentação do chorus. Compare sua resposta com a solução da figura 209.

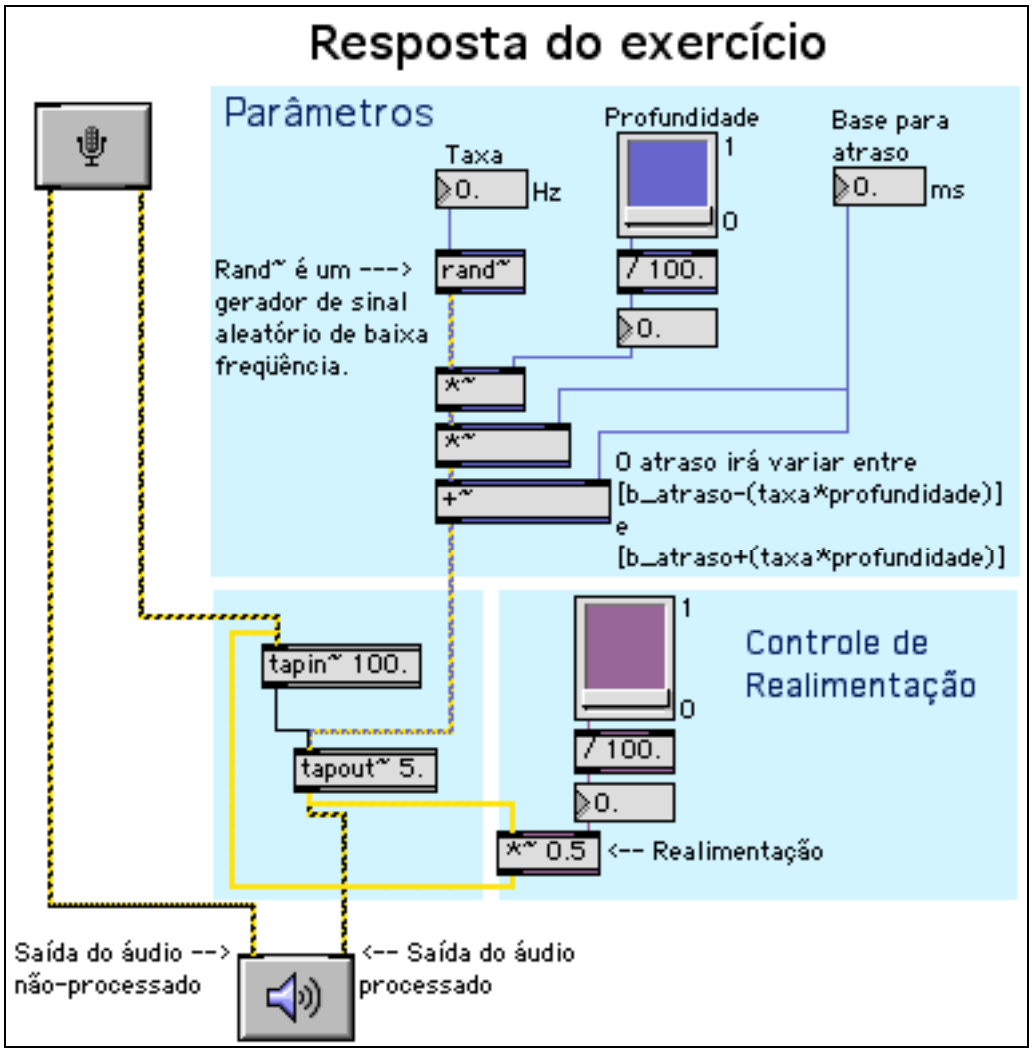


Figura 209 – Resposta do exercício para controle do nível de áudio de realimentação do chorus

7.3.6.4 Flanger

O flanger é um efeito produzido por características similares às do chorus, usando, no entanto, defasagens menores. Por isso, ao invés de dar uma impressão de dobra, o flanger produz na realidade uma alteração cíclica de composição harmônica (coloração), que às vezes dá ao ouvinte a sensação de semelhança ao som de um avião a jato passando.

Assim como no chorus, no flanger também muitas vezes é possível ajustar a velocidade da variação de defasagem e sua intensidade, e o flanger em geral é aplicado em instrumentos com uma coloração rica em harmônicos, onde ele sobressai mais, como guitarras, cordas e pratos de bateria.

Para entender como é possível programar um efeito de flanger, estude o programa da figura 210. Utilize o microfone para entrar com o áudio. Escolha um preset para experimentar vairações no efeito de flanger atuando sobre o sinal de entrada. Depois resolva o exercício proposto.

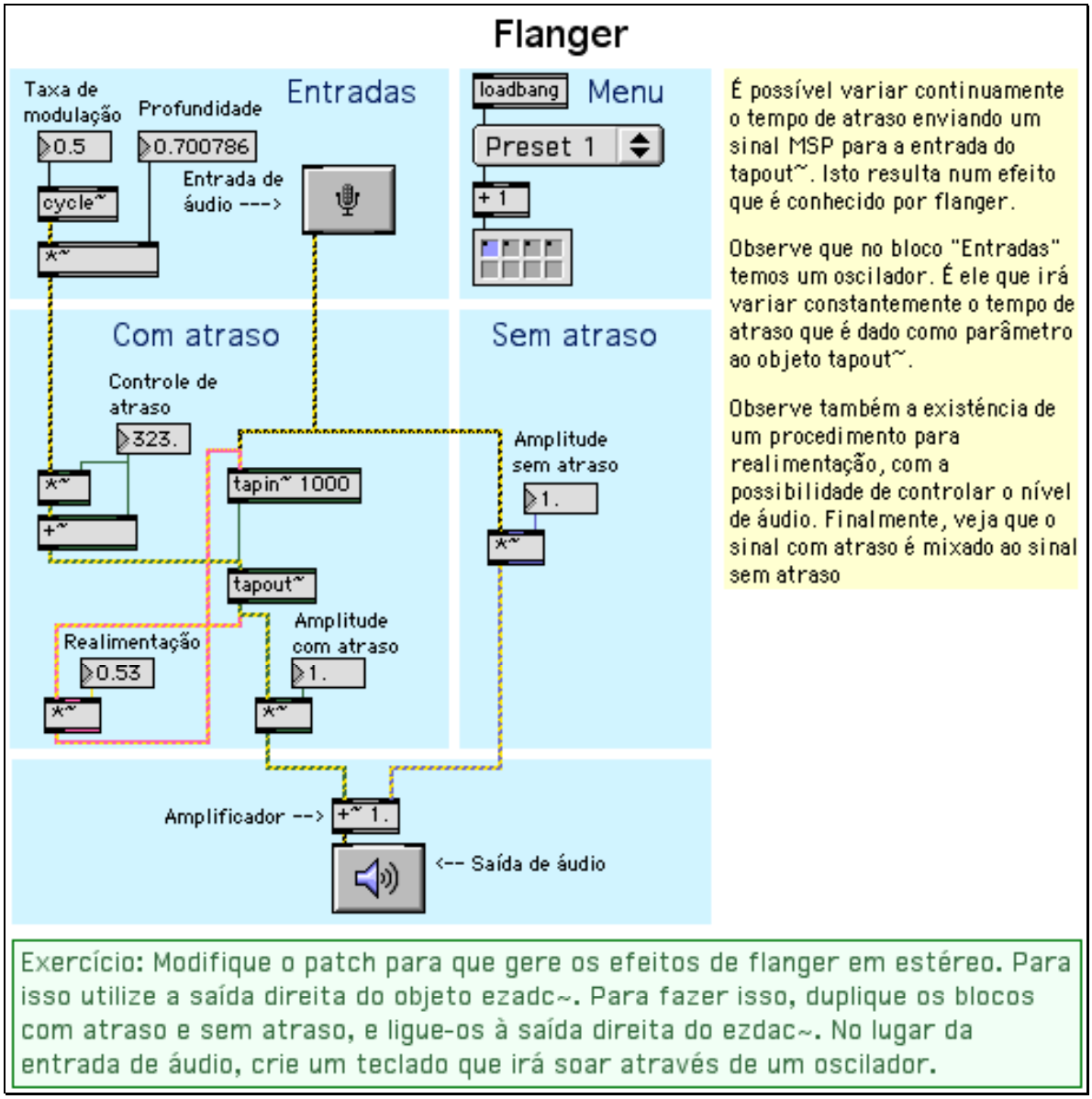


Figura 210 – Implementação do efeito de Flanger

A figura 211 mostra a construção de um efeito de flanger em estéreo. Compare sua solução com a da figura 211.

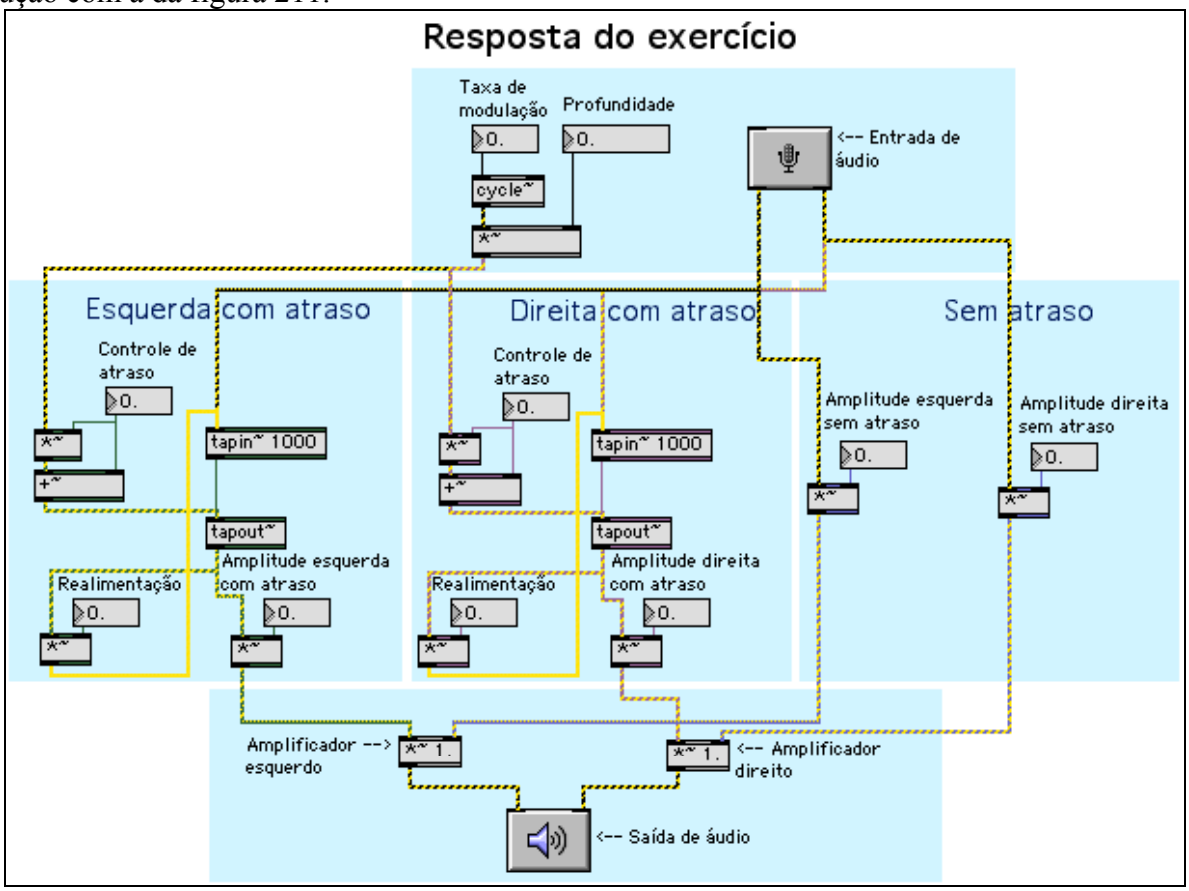


Figura 211 – Resposta do exercício para a programação do efeito de flanger em estéreo

7.3.6.5 Distorções

As distorções são obtidas pela saturação do sinal de áudio, que introduz e realça harmônicos antes pouco perceptíveis, alterando assim substancialmente a coloração do som.

8 NÍVEL DE PROGRAMAÇÃO APLICADA

8.1 MÓDULO DE PROGRAMAÇÃO DE APLICAÇÕES NA ÁREA DE COMPOSIÇÃO MUSICAL

O Módulo de Composição apresenta técnicas básicas para composição auxiliada por computador. A ênfase desse tópico está em ensinar o compositor a programar o computador, tendo como objetivo, a geração de material musical para composições eletroacústicas.

Os exemplos e exercícios práticos, que constituem o Módulo de Programação de Aplicações na Área de Composição Musical, foram implementados com base nas obras de [WIN 89] e [MIR 2001].

8.1.1 AULA 1 – Histórico e Conceitos da Composição Auxiliada por Computador

O MEPSOM – Método de Ensino de Programação Sônica para Músicos, apresenta um roteiro para o estudante observar exemplos e realizar exercícios propostos de programação. Antes de iniciar os exemplos práticos e os exercícios de programação em composição, o aluno necessita de uma fundamentação na área de composição auxiliada por computador. A seguir serão apresentados conceitos e o histórico da composição por computador. Somente após entender os princípios dessa área, o aluno deverá ler com muito cuidado as explicações e apontamentos nos programas exemplo para um perfeito entendimento da utilização dos objetos em conjunto.

8.1.1.1 A Composição por Computador

A composição encabeça a lista de Áreas de Pesquisa na Música, porque a modelagem, simulação e projeto de estruturas musicais provê soluções para problemas encontrados nas demais áreas da música. Normalmente a pesquisa nesta área requer a formulação de estratégias ou paradigmas antes de haver algum trabalho para o computador executar. Algumas das categorias de pesquisa são:

I Composição Algorítmica Interativa: aplicação de procedimentos de transformação de dados para estruturas sonoras.
II Composição Algorítmica Automática: estudo de sistemas de composição auto-controlados baseados em algoritmos que favorecem um estilo musical particular
III Sistemas para Música Experimental: Música estocástica e fractal.
IV Estudos na Estrutura Interna do Som Musical: Pesquisa de timbres através do auxílio de técnicas de síntese.

8.1.1.2 As Origens da Composição Algorítmica

Por trás dos esforços modernos em composição algorítmica, há uma longa tradição de observar a música proceduralmente. A seguir apresentamos uma compilação

desse assunto realizada a partir da obra de [ROA 96] englobando os processos formais (software) e as máquinas (hardware) empregadas na composição algorítmica.

8.1.1.3 Histórico dos Processos Formais na Música

Os procedimentos para composição já existem existem a muito tempo. Por volta de 1026, Guido d'Arezzo desenvolveu uma técnica formal para composição de uma melodia para acompanhamento de um texto [KIR 68]. O seu esquema assinalou um tom para cada vogal. Assim, a melodia variava de acordo com a vogal contida no texto. (Veja [LOY 89] para saber mais sobre este método.)

Existe uma grande quantidade de procedimentos para composição baseados em razões numéricas. Um dos primeiros expoentes neste tema foi Guillaume Dufay (1400-1474). Dufay também aplicou procedimentos sistemáticos como a *inversão* (transformar todos os intervalos positivos em negativos, e vice-versa) e a *reversão* (reverter a ordem) para seqüências de tons, séculos antes deles serem utilizados por compositores de música serial [SAN 81].

A utilização de seqüências rítmicas repetitivas como uma técnica formal apareceu nos motetos *isorítmicos* compostos por G. Machaut e outros no período de 1300 a 1450. A técnica isorítmica insere um modelo rítmico de repetição em diferentes camadas melódicas de uma composição. . (Veja [KOE 96] para saber mais sobre motetos.)

Os cânones e variações da música tradicional são exemplos de processos musicais formalizáveis. Talvez, o exemplo histórico mais famoso de composição algorítmica seja o *Musicalisches Würfelspiel* de W. A. Mozart - um jogo de dados para montagem de minuetos através de um conjunto de medidas musicais pré-escritas. A seqüência de medidas foi determinada por um conjunto de dados jogados. Portanto, este programa incorporou um elemento de probabilidade - um aspecto de muitos programas algorítmicos dos dias de hoje.

Em 1822, foi colocado em jornais de Boston anúncio para venda do sistema Kaleidoacústico - um baralho de cartas com instruções indicando como até 214 milhões de valsas poderiam ser compostas com as cartas [SCH 75]. Este pode ter sido o primeiro software comercial para música.

No século XX, o pensamento científico se expandiu, permitindo a introdução de procedimentos matemáticos para composição. Entre eles, estão as estratégias seriais e estocásticas, nas quais elementos musicais são manipulados em acordo com um conjunto de operações teóricas ou processos probabilísticos [HIL 59]; [BAB 60]; [XEN 92]. Qualquer fórmula pode ser transformada em um gerador de dados musicais [SCH 46], [CLA 78].

A proliferação atual das composições algorítmicas está relacionada diretamente à expansão dos computadores eletrônicos.

8.1.1.4 Máquinas de Composição Automatizada Anteriores ao Computador

A automação implica a existência de um conjunto de máquinas. Máquinas para execução automatizada de músicas, como por exemplo os carrilhões, existem há séculos [BUC 78]. Foram feitas menos máquinas para composição do que para execução, porém elas existiram. As máquinas para composição refletem a tecnologia disponível em sua época. Em 1660, Athanasius Kirchner projetou sua Arca Musirítmica, uma espécie de jogo de composição numa caixa, para ser engendrado com as mãos [PRI 75]. Um outro engenho para composição era impulsionado por um mecanismo com um fole manual, com alavancas e engrenagens. O Componium, do tamanho de uma sala, de Dietrich

Winkel's, um mecanismo mediano acabado em 1821, produziu variações sobre um tema [BUC 78].

Com o advento da eletrônica no século XX, surgiram novas possibilidades para a automação musical. H. Olson e H. Belar, inventores do sintetizador RCA e pioneiros do seqüenciador analógico, também montaram uma máquina eletromecânica de composição por volta de 1951 [OLS 61]; [HIL 70]. A primeira inovação da máquina de Olson e Belar foi a automatização de um sistema probabilístico de composição. A máquina deles utilizava um par de multivibradores estáveis e assíncronos (geradores de ondas quadradas) para gerar dígitos randômicos para um gerador probabilístico de tons e ritmos.

O aparecimento de computadores chamados de "cérebros eletrônicos" foi nos anos cinqüenta. A promessa de uma máquina para composição foi o argumento de venda do GENIAC [SOW 56]. Os anúncios definiam a caixa como "Uma genuína máquina cerebral - não um brinquedo".

8.1.1.5 Controle Seqüencial de Composições

A possibilidade de reproduzir gravações automatizadas de seqüências musicais contribuiu para a composição algorítmica. Um aparelho controlado seqüencialmente lê uma seqüência de entrada de dados que informa quando e o que tocar (tons, amplitudes, etc.). A especificação MIDI (Musical Instrument Digital Interface) é um protocolo para controle seqüencial muito utilizado atualmente em composição. A combinação de implementação lógica em hardware para composição com a execução controlada seqüencialmente permite a geração de vários tipos de sistemas. Alguns desses sistemas foram desenvolvidos após a invenção do computador, porém na época em que eles ainda eram muito grandes, caríssimos e de difícil ligação com outros equipamentos.

Os aparelhos eletrônicos musicais controlados seqüencialmente começaram a aparecer depois que o seu princípio foi demonstrado em 1929 através do protótipo de sintetizador Couplex-Givelet [RHE 72]. Outros sintetizadores que apareceram, foram o RCA Mark I e Mark II, utilizados por Milton Babbitt [OLS 55]; [OLS 61]; [BAB 64].

O computador de composição Barr e Stroud Solidac instalado em uma mesa adaptada na Universidade de Glasgow, 1959, serviu tanto como um aparelho de composição algorítmica como um gerador digital de som (utilizando um circuito divisor de freqüências). Operando a uma taxa de clock de 30 Khz (os computadores atuais operam a uma taxa 1000 vezes maior, ou até mais), o leitor de cartões do Solidac rodou um "Programa Mestre de Dados Musicais" que, de acordo com os seus desenvolvedores, eram capazes de gerar quase um bilhão de trios Haydn [ROA 96].

O Electronium (no início dos anos 60, em Los Angeles), um aparelho totalmente diferente, era uma máquina de composição em larga escala, programada por botões e interruptores. (Não confundir com o Elektronium, instrumento de teclado inventado pela empresa Hohner em 1950 e usado por compositores como Stockhausen.) Um compositor solicitava ao Electronium para sugerir um tema, o qual era então reproduzido em um alto-falante. Quando o compositor ouvia um tema que se adequasse, ele movimentava um interruptor que acionava os retransmissores elétricos e a memória de baterias do Electronium. A música gerada pelo sistema podia ser modificada pelo compositor por botões e interruptores, porém não de forma direta, uma vez que a resposta do Electronium a estes controles de entrada não era totalmente previsível. De acordo com o seu inventor, o compositor Raymond Scott, "O Electronium não é tocado; ele é orientado" [RHE 84].

Diferentemente da adaptação feita no Electronium, Coordinated Electronic Music Studio - CEMS, no Estado de Nova Iorque, em Albany, idealizado pelo compositor[ROA 96], foi montado a partir de um seqüenciador padrão e componentes de sintetização feitos pela empresa R. A. Moog. Enquanto o sistema CEMS foi montado como um estúdio, o Sal-Mar Construction, desenvolvido pelo compositor Salvatore Martirano e pelo engenheiro Sergio Franco, foi projetado para ser tocado ao ar livre. [MAR 71]. Apesar de que a lógica de composição do Sal-Mar era digital, a máquina não possuia um computador para propósitos gerais. À semelhança do Electronium, o Sal-Mar Constrution era orientado para improvisação

Lejaren Hiller contribuiu muito para as bases da música computacional moderna. Desestimulado com a vida de um químico industrial - na qual ele recebeu o seu doutorado e rapidamente se estabeleceu - Hiller voltou sua mente para a música, não mais retornando ao seu passado. Muitos "primeiros" são relacionados com o seu nome: ele foi um dos primeiros a aplicar computadores na composição algorítmica [HIL 59], na notação impressa de músicas [HIL 65] e na síntese de modelos físicos [ROA 96], entre outros feitos. Com acuidade sobre a significância cultural de sua era, ele contribuiu como o primeiro historiador da composição musical, com a pesquisa publicada em 1970 "Composição Musical por Computadores" ("Music Composed with Computers").

O centro das atenções de Hiller permaneceu na composição algorítmica. Nos meados de 1950, o computador digital recém desenvolvido proporcionou o veículo ideal para as suas visões composicionais [HIL 70]; [HIL 79]; [HIL 81]. Havia poucos computadores no tempo em que ele começou os seus experimentos. Era usual uma universidade de ponta projetar o hardware de seus computadores. Assim sendo, nesses tempos utilizou-se de um computador Illiac construído por engenheiros na Universidade de Illinois, Urbana-Champaign [HIL 1959]. Codificando em linguagem binária de máquina (seqüência de zeros e uns aceitos diretamente pelo computador), Hiller e seu colaborador Isaacson criaram, em 1956, a primeira composição por computador: *The Illiac Suite for String Quartet* - um marco na história da música. Vieram então a *Computer Cantata* [HIL 64] e *HPSCHD*, em colaboração a John Cage [ROA 96]. Os conceitos em torno da "música experimental" de Hiller expandiram-se do campus de Illinois para o mundo da música [HIL 64].

Rapidamente, alguns bravos compositores seguiram de perto os passos de Hiller. Entre eles, Herbert Brün e John Myhill (Urbana-Champaign), James Tenney (Murray Hill), Pierre Barbaud, Michel Phillipot, Iannis Xenakis (Paris) e G. M. Koenig (Utrecht). Suas aplicações computacionais vieram pouco após dos trabalhos de Hiller. Em alguns casos, porém, eles já haviam composto seguindo procedimentos formais. Por exemplo, uma peça composta por fórmulas estocásticas, trabalhadas manualmente, a *Metástase (Metastasis)* para orquestra de Xenakis, teve sua première em 1955 - no mesmo ano que Hiller começou seus experimentos computacionais [ROA 96].

Los Angeles tinha uma orientação mais comercial: uma música estilo pop, Bertha Empurra o Botão (Push-Button Bertha), de 1956, com letra do mito musical de Hollywood, M. Klein, e melodia gerada de um computador Burroughs Datatron (programado por D. Bolitho), não conseguiu chegar às paradas musicais. Esta foi a única incursão do trio nas artes computacionais. Rudolf Zaripov, alguns anos depois, em Moscou, analisou e recompôs música folk utilizando-se de computadores URAL da União Soviética [ZAP 60]; [HIL 70].

Bastante cedo, também, foram realizados na Grã-Bretanha experimentos de composição com computadores fora de instituições musicais estabelecidas, por D. Champernowne e S. Gill, um economista e um matemático, respectivamente [HIL 70].

Os programas escritos na época primordial da composição algorítmica confrontavam várias limitações. Por exemplo, o computador Illiac, que ocupava o espaço de um quarto, no qual eram rodados os programas de Lejaren Hiller, tinha uma memória total de 1024 palavras (A história se repetiu nos anos 1970 quando os primeiros microcomputadores surgiram com limitações de memória similares)[ROA 96].

No final dos anos sessenta, os primeiros terminais gráficos de computador abriram as possibilidades de unir algoritmos à interação gráfica. Cientistas como Max Mathews nos Laboratórios da Bell Telephone exploraram novas possibilidades radicais oferecidas pelo controle gráfico de composição e síntese. Estes experimentos incluíam desenhos de curvas em uma tela de um computador que controlavam o grau de interpolação de ritmo e tom entre duas músicas [MAT 69]. Matematicamente falando, a interpolação é um processo de preenchimento (ligação por um traço) entre dois pontos . Este termo também se aplica adequadamente para denotar uma fusão ou medição entre duas funções,

8.1.1.6 Motivações Estéticas por trás da Música Algorítmica

Seria um descuido descrever a composição algorítmica puramente do ponto de vista técnico. A estética tende a emergir do mundo musical, inspirado nas primeiras pesquisas em composição algorítmica, bem como desenvolver rapidamente possibilidades técnicas.

O advento da 2.ª Guerra Mundial permitiu uma aceleração na tendência de formalizar e sistematizar métodos de composição no início dos anos cinqüenta. O método básico de composição - promulgado 30 anos antes por Arnold Schoenberg - tem sido generalizado, do tom, aos demais parâmetros musicais (duração da nota, marcação da dinâmica, etc.) por Anton Webern, Olivier Messiaen e outros.

Tentando romper com o determinismo extremo da composição serial, compositores europeus e americanos como Karlheinz Stockhausen, Pierre Boulez, John Cage e Earle Brown realizaram experimentos com métodos de composição *aleatórios*. A aleatoriedade, toma o sentido de que alguns detalhes da peça ficam ao critério do executante ou que as composições são realizadas com técnicas randômicas como, por exemplo, o lançamento de dados. Tomando um caminho mais sistemático, Iannis Xenakis, em meados dos anos cinqüenta, compunha músicas com fórmulas *estocásticas*, que formalizam pesos em um intervalo de probabilidades [ROA 96].

Os celebrados experimentos de Hiller com a composição automatizada provou que o computador pode modelar qualquer procedimento formal: de cânones da harmonia tradicional à doutrina da técnica serial; ambos os métodos determinístico e estocástico podem ser codificados [HIL 59]. A programação acelerou o longo trabalho associado à composição sistemática. Conseqüentemente, o software surgiu como uma extensão lógica da estética da composição formalizada [CHA 97].

8.1.1.7 Processos Estocásticos x Determinísticos

Os programas de Hiller e de outros pioneiros da composição algorítmica seguiam duas soluções contrastantes entre si: a de procedimentos *determinísticos* versus a de procedimentos *estocásticos* (ou *probabilísticos*). Os procedimentos determinísticos geram notas musicais carregando uma tarefa fixa, porém possivelmente complexa que não envolve seleção randômica. As variáveis que suprem um procedimento

determinístico são chamadas de dados-semente. Podem ser um conjunto de tons, uma frase musical, ou algumas diretivas que o procedimento deve cumprir.

Pode-se citar como exemplo de um procedimento determinístico um programa para harmonizar uma melodia de coral no estilo de J. S. Bach. Neste caso, a semente é a melodia. As regras de harmonização e voz líder, oriundas de um livro-texto, asseguram que somente algumas seqüências de acordes são permitidas.

Os procedimentos estocásticos, por outro lado, integram escolha randômica no processo de decisão. Eles geram eventos musicais de acordo com tabelas que elegem certos eventos em detrimento de outros. Estas tabelas asseguram uma tendência geral, mas os ornamentos de eventos locais permanecem imprevisíveis. Um gerador estocástico básico produz um número ramdômico e compara-o com valores armazenados em uma tabela de probabilidade. Se o número randômico cai em um certo intervalo de valores em uma tabela de probabilidades, o algoritmo gera um evento associado a esse intervalo.

Alguns algoritmos exibem um mecanismo comportamental de maneira que podemos identificá-los quando os escutamos. Desconsiderando estes casos simples, não é possível assegurar, simplesmente escutando, quando certo fragmento musical foi gerado por um processo estocástico ou determinístico. Portanto, a escolha do algoritmo é uma questão estética e de filosofia composicional. Muitos tipos de algoritmo podem ser intercalados em um sistema e aplicados em várias dimensões do processo de composição.

[ROA 96], de Karlheinz Stockhausen é um exemplo de música aleatória. O compositor instrui o executante a olhar randômicamente na partitura e iniciar com qualquer grupo, em qualquer tempo, nível de dinâmica e tipo de nota de ataque. Ao terminar um grupo, observa o tempo, a dinâmica e as marcações de ataque. Então, escolhe de novo outro grupo randomicamente, executando-o com as marcações escritas ao final no grupo anterior [ROA 96].

8.1.1.8 Os Primeiros Programas de Composição

Já foram escritos muitos programas de composição algorítmica. Poucos, porém, têm sido usados por mais de um compositor. Neste texto apresentamos brevemente três dos mais conhecidos programas que foram utilizados por outros além de seu próprio autor: o (Stochastic Music Program - SMP) de Iannis Xenakis; o Project 1 de G. M. Koenig e o programa POD de Barry Truax. Esses programas representam a primeira geração dos programas de composição automatizada - todos eles escritos por programadores-compositores. O SMP e o Projeto 1 foram escritos nos anos sessenta e a primeira versão de POD no início dos anos setenta [CHA 97].

O SMP e o Projeto 1 geram, na saída, uma lista de notas (representadas alfanumericamente). Inicialmente, as listas de notas destinavam-se à transcrição em notação musical comum e tocadas em um instrumento tradicional. No entanto, ambos os programas têm sido conectados em sistemas de síntese de som digital em vários estúdios. Os programas POD, por outro lado, foram originalmente projetados para esses sistemas de síntese de som digital. Como conseqüência, os resultados com os POD podem ser escutados quase que imediatamente [ROA 96].

O autor do SMP, o compositor estabelecido em Paris, Iannis Xenakis, é um dos pioneiros da composição automatizada. Na verdade, o seu trabalho neste campo é apenas uma entre várias de suas inovações apresentadas. Uma versão primária do programa SMP foi publicada em seu livro [XEN 92] como o programa Música Estotástica Livre (Free Stochastic Music). As fórmulas estocásticas no SMP foram

desenvolvidas originalmente por cientistas para descrever o comportamento de partículas em gases. Dessa maneira, na visão de Xenakis, a composição era representada como uma seqüência de *nuvens de sons*, nas quais as partículas correspondiam a notas individuais. (Veja [XEN 60] e [XEN 92] para um exame detalhado das concepções estéticas do programa.). Os programas SMP auxiliaram a criação de alguns trabalhos importantes de Xenakis, incluindo *Eonta* (1964, Edições Salabert), para piano e metais, que teve Yuji Takahashi no piano em sua première (Vanguard Records). Posteriormente, o programa foi aprimorado pelo matemático e compositor John Myhill [ROA 96] e, desde então, gravado para uso em computadores pessoais.

Gottfried Michael Koenig criou o programa Projeto 1 no Instituto de Sonologia (Institute of Sonology, Utrecht) em 1970. O Projeto 1 compõe aplicando sete *princípios de seleção* em um banco de dados com cinco *parâmetros* de eventos musicais: o instrumento, o ritmo, a harmonia, a marcação e a dinâmica [ROA 96]. No Projeto 1, os princípios de seleção variam da completa aleatoriedade ao completo determinismo. Enquanto os programas SMP e Projeto 1 geram partituras para instrumentos tradicionais, os programas POD (Distribuição de Poisson) de Barry Truax foram projetados diretamente para síntese digital de som [TRU 75]; [TRU 77]; [TRU 85]. Os sistemas POD substituem o tradicional *conceito de alturas,* utilizado no SMP e no Projeto 1, por um conceito mais geral de *objetos digitais sonoros*. Truax começou a desenvolver o primeiro sistema POD no Instituto de Sonologia de Utrecht, em 1972. Durante vários anos ele desenvolveu novas versões para outros sistemas computacionais [CHA 97].

Segundo [ROA 96], os programas do tipo do SMP e do Projeto 1 foram projetados para gerar listas de notas alfanuméricas. Esses relatórios eram então transcritos pelos compositores em partituras para instrumentos tradicionais. Hoje em dia, os dados gerados por programas de composição podem ser transmitidos a equipamentos de síntese de sons, mostrados na tela do computador ou impressos. Os relatórios podem ser numéricos (listas de valores de parâmetros), em notação tradicional ou em uma partitura gráfica.

Um dos pioneiros da integração de programas musicais foi Donald Byrd (atualmente conhecido por seu programa de notação musical, o Nightingale). Trabalhando na Universidade de Indiana, Byrd comparou o SMP com o MUSC, programa de composição de sua autoria, e concluiu, em 1977, seu sistema de impressão musical, o SMUT. Assim, as partituras geradas pelos programas podiam ser imediatamente fornecidas a músicos para serem executadas [ROA 96].

O programa Project 1 não tem opção para notação musical; ele gera um relatório alfanumérico. Koenig sempre considerou o processo de transcrição da listagem em uma partitura musical uma tarefa interpretativa importante para o compositor. Por exemplo, o mesmo relatório podia ser usado para escrever uma partitura para piano ou arranjo para música de câmara. É permitida uma liberdade considerável ao compositor na tarefa de orquestrar a listagem [KOE 79]. Em programas com a capacidade de síntese do POD, vários esquemas de orquestração podem ser tentados diretamente.

8.1.1.9 Automatização Total x Composição Interativa

Um dos primeiros receios com relação à composição automatizada, é que ela poderia substituir seres humanos na composição, do mesmo modo que as gravações substituíram os músicos em muitos locais. Quatro décadas após ampla publicação dos experimentos de Hiller, isto não ocorreu. Mesmo assim, permanecem controvérsias sobre o uso de programas para composição. Para um compositor que não é um

programador, programas de composição totalmente automatizados demandam pouca criatividade. A interação do compositor é limitada a fornecer uma pequena quantidade de dados-semente antes de executar o programa. A estratégia de composição é fixada no programa, e o usuário, simplesmente, colhe a safra das notas quando finalizado. Em sua forma mais extrema, a composição automatizada remonta uma forma de "achado artístico"; o compositor seleciona a saída; então assina, informa um meio de execução e coloca um título no trabalho.

Uma maneira de ultrapassar a estratégia fixa é modificar a lógica do programa. Neste caso, um compositor, que também é um programador assume total responsabilidade.

Outra saída para a estratégia fixa é revisar as saídas geradas pelo programa. Alguns proponentes da composição automatizada, notavelmente Hiller e Barbaud, aderiram à doutrina de que a partitura gerada de um programa de composição não deviam ser editadas à mão; ao invés disso, a lógica do programa deveria ser modificada e rodada novamente. Esta doutrina se origina de uma estética que observa formalmente a música no aspecto da consistência. Outros compositores não sentem ser um sacrilégio modificar a partitura gerada por um programa de composição. Xenakis, particularmente, rearranjou e refinou os dados emitidos pelo programa SMP. O seu acervo contém numerosos exemplos de seleção e rearranjo de saídas originadas dos programas [ROA 96].

8.1.1.10 Ambientes de Programação para a Música

Um ambiente de composição algorítmico é um kit de ferramentas: uma coleção de "mecanismos" montado em combinações para serem escolhidas pelo usuário. A flexibilidade do kit de ferramentas permite que o compositor projete estratégias personalizadas para sua composição.

De forma geral, não há muita distinção entre os "ambientes" e as "linguagens de composição procedural". Alguns dos ambientes disponíveis atualmente, como o Max, são orientados à gráficos interativos. Outros ambientes de composição são embutidos em linguagens de programação, como é caso do Csound.

Geralmente, os ambientes são extensíveis. Quer dizer, é fácil adicionar novas funções, ao contrário dos programas aplicativos fechados ou "enlatados". Interagimos com aplicativos fechados através de diferentes facetas ou editores. Pode-se selecionar itens de menu ou utilizar modelos, mas não se pode alterar a lógica do programa ou extendê-lo significativamente. Em contraste, muitos ambientes não fazem nada até o usuário montá-los em um sistema personalizado [ROA 96].

O primeiro ambiente de software musical talvez tenha sido a biblioteca de subrotinas de linguagem assembly MUSICOMP, desenvolvida por Robert Baker e Lejaren Hiller, na Universidade de Illinois [BAK 63]; [HIL 69]. A biblioteca MUSICOMP incluia ferramentas para a seleção de itens de uma lista, de acordo com uma distribuição de probabilidades, o embaralhamento randômico de itens de uma lista, a manipulação de listas de tons, a inserção de regras melódicas e a coordenação de perfis de ritmo.

As saídas das rotinas do MUSICOMP podiam ser impressas ou formatadas para entrada de dados em programas de síntese de sons. Hiller utilizou as rotinas do MUSICOMP para criar trabalhos tanto para instrumentos tradicionais como para sons gerados por computador, entre eles *Algorithms I* (1969).

Programadores-compositores têm desenvolvido muitos outros ambientes e linguagens de composição recentemente como é o caso de [POP 86], [LEN, 84],[BOY et al. 1986];[BLE, JEN e GLA 88] e [COP 87] [COP 90].

Esses ambientes trilham objetivos estéticos diferentes, mas compartilham a meta de prover um kit de ferramentas que pode ser montado em configurações personalizadas por compositores. Muitos ambientes geram partituras, editores de som, geradores algorítmicos e características de execução em tempo real. Os ambientes mais flexíveis são modulares, ou seja, os compositores necessitam utilizar somente as funções que lhes interessem, combinando-as de maneira tal que venham ao encontro de suas necessidades. [ROA 96].

Os programas de composição conseguem manipular mais detalhes em um espaço de tempo mais curto do que seria possível para um ser humano trabalhando sozinho. Eles permitem que o compositor coloque sua atenção em minúsculos detalhes da composição (os quais são manipulados pelo programa, conforme instruções especificadas ou, pelo menos, endossadas pelo compositor), ou níveis maiores de abstração. Neste nível, o compositor gerencia a criação da peça considerando seus movimentos, a arquitetura formal e o modelo de processos.

Segundo [ROA 96] , um dos perigos no uso de programas de composição, é que eles servem como um substituto da criatividade de seu usuário. Os primeiros programas de composição por lote eram particularmente suscetíveis a este tipo de abuso. Os programas interativos e os sistemas de performance exigem mais do compositor, uma vez que eles equilibram a automação pré-programada com decisões espontâneas (intervenções).

O talento dos compositores que utilizam métodos algorítmicos de composição deve refletir na sua habilidade em gerenciar os resultados obtidos através do emprego do software musical .

8.1.2 AULA 2 - Métodos de Composição por Computador

O computador pode ser utilizado de várias maneiras no processo de composição musical. Através de um software seqüenciador o computador pode realizar o papel de um estúdio MIDI gravando todas as ações do músico sobre um instrumento eletrônico. Com a utilização de um software de gravação digital de áudio, o computador pode ser encarado como um estúdio digital, desde que possua capacidade para processar e armazenar o áudio digitalizado. Mas estas aplicações são utilizadas para armazenamento e edição de informação musical, portanto são pouco utilizadas na geração automática de material musical para composição musical auxiliada por computador.

Neste texto iremos abordar, especificamente, métodos para a composição musical por computador utilizando o Max.

8.1.2.1 Objetos Composicionais em MAX

Objetos Composicionais criam música diretamente ou indiretamente pela resposta de uma execução musical. Objetos Composicionais são algoritmos que englobam o processo de raciocínio, experiência, e técnica de um compositor. A lógica envolvida em criar um algoritmo não implica no talento do compositor (isto é, na sua criatividade, imaginação e no seu refinamento artístico). Estes aspectos são secundários à lógica dos algoritmos [WIN 98].

Objetos composicionais podem receber dados de uma variedade de fontes: de dados MIDI em tempo real, do teclado do computador ou do mouse. Objetos de armazenamento podem dispor de dados pré-determinados ou material musical que é carregado dentro de um programa antes da performance. Os dados pré-determinados podem ser qualquer informação de uma partitura musical completa ou poucos intervalos usados para criar novas melodias ou acordes. Os objetos *seq*, *table*, ***coll*** e *detonate* são usados, na maioria dos casos, com este propósito. Eles podem reter uma grande quantidade de informações em tempo real de forma organizada.

A geração de dados é produzida por algoritmos que criam dados a partir de um pequeno subconjunto armazenado ou, então, dados em tempo real. O computador pode utilizar uma fórmula matemática para, internamente, gerar números usados para a entrada musical. Um exemplo disso é o objeto *random*, que gera números aleatórios numa determinada faixa de valores. Observe a figura 212 e utilize o patch em Max para produzir seqüencias musicais automáticas e aleatórias. Armazene o resultado no seqüenciador e produza uma pequena composição utilizando o material musical obtido.

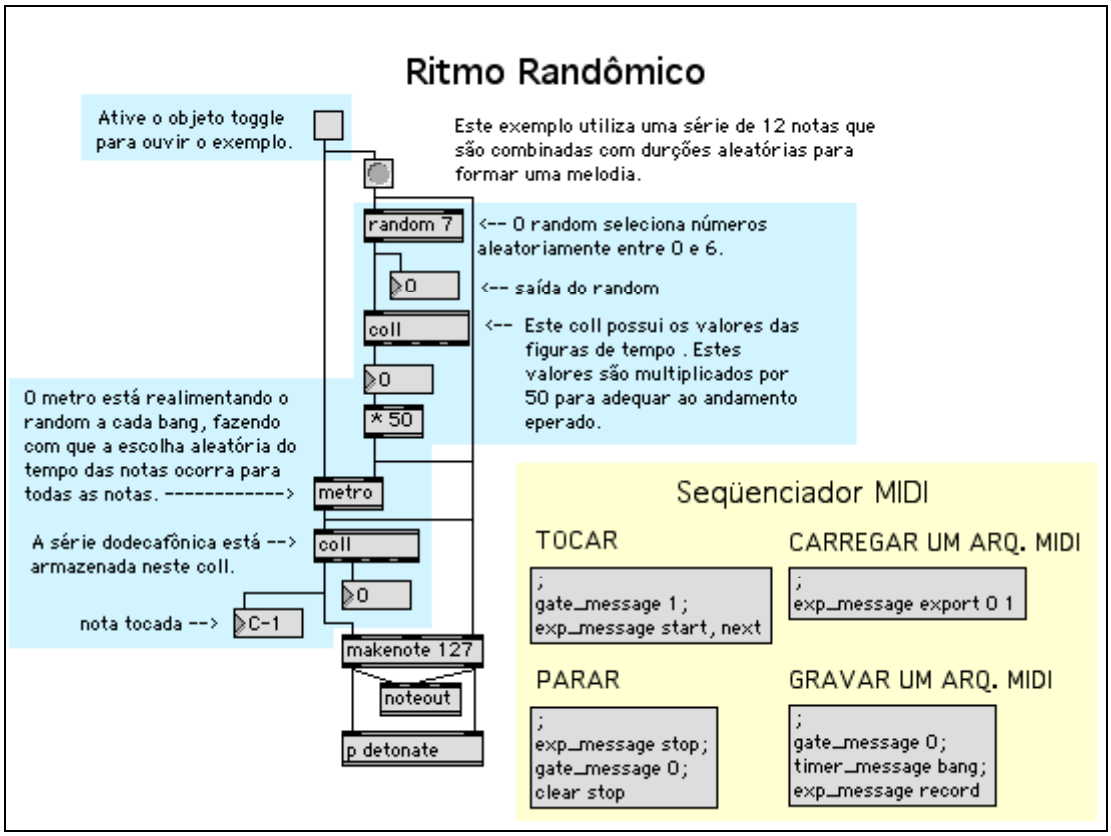


Figura 212 – Ritmo Randômico

Os dados processados são o resultado do algoritmo. Grande parte das aplicações desenvolvidas para MAX utilizam vários algoritmos onde os dados são passados de um objeto para outro, até resultarem na saída desejada.

8.1.2.2 Composição Interativa

A composição interativa é uma operação limite entre a composição e improvisação. É uma espécie de processo híbrido de criação. Esse processo híbrido combina muitos conceitos de composição e improvisação. A improvisação geralmente ocorre quando as decisões composicionais são feitas na hora, ou seja, em tempo real. [YAV 92].

A Composição interativa não é uma novidade, muitos músicos, que trabalhavam com computador, já utilizavam sistemas de composição elaborados por eles próprios para auxiliar na criação musical. O que existe de novo atualmente é a disponibilidade comercial de ferramentas interativas para a composição como é o caso do MAX.

Composição interativa não pode ser confundida com música gerada por computador, composicão algorítmica e nem composição automática, apesar de possuírem ancestrais em comum. Com o surgimento do conceito de Composição Interativa, os termos “música gerada por computador" tem evidenciado a idéia de um tipo específico de algoritmo que, uma vez em funcionamento acha seu próprio caminho e executa suas próprias tarefas com uma pequena intervenção humana na definição dos parâmetros iniciais do sistema. Contrastando com isso, a composição interativa adiciona interatividade em tempo real no ato da composição. Por isso, a composição interativa assemelha-se à improvisação. Através dessa fusão de atividades de composição, improvisação e também performance ao vivo, a composição interativa tem em muito

estimulado as pesquisas na área. Isto pode ser atribuído a um número de fatores interdependentes que são inerentes ao processo [YAV 92].

O primeiro fator é a redefinição da relação entre o homem e a máquina. A antiga relação mestre/escravo passa a ser a relação sócio/colaborador.

O segundo fator revela que as atividades computacionais não estão mais restritas à competência de um mestre em computação.

Uma das formas mais comuns de geração de música através do computador é a utilização das coordenadas de movimentação do cursor na tela. Na figura 213 é apresentado um programa para gerar música através da movimentação do mouse. Procure utilizar o programa para gerar seqüências musicais variando o movimento do mouse quanto à velocidade e a posição na tela do computador.

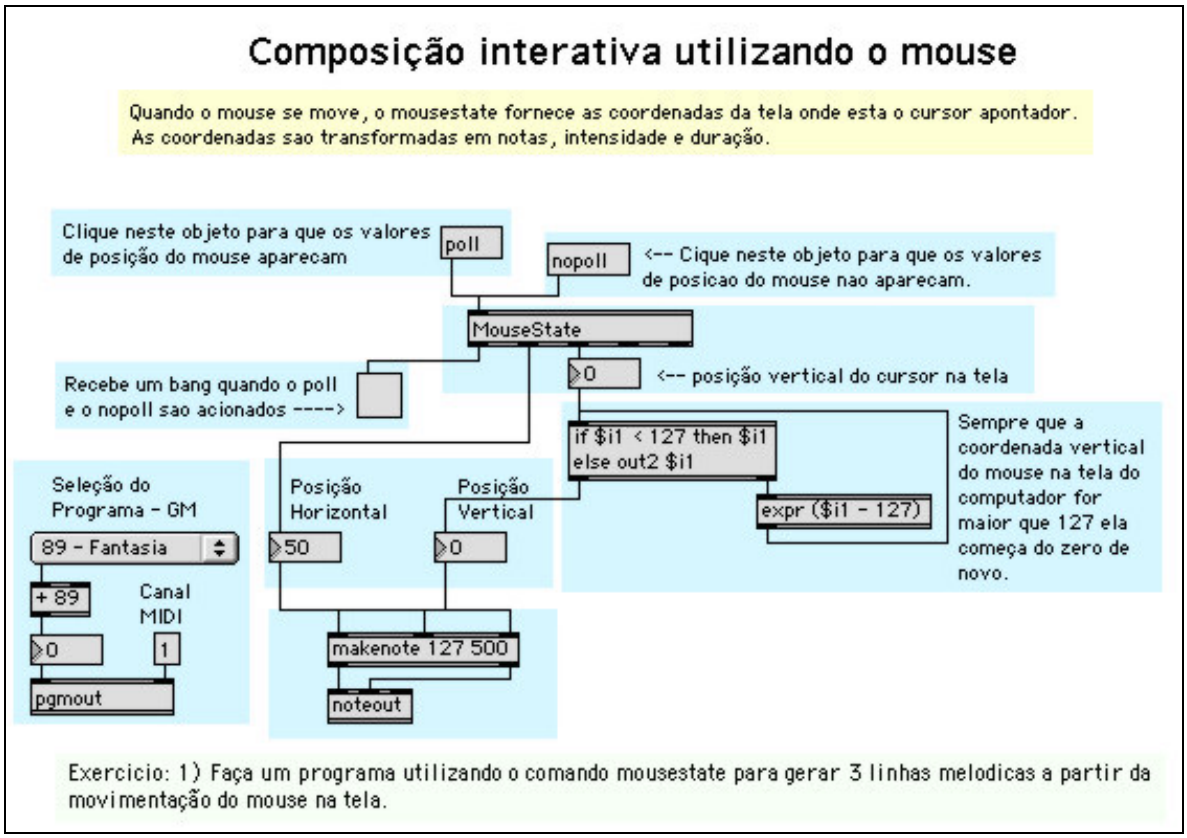


Figura 213 – Geração de alturas através do movimento do mouse

8.1.2.3 Instrumentos Inteligentes

Um instrumento eletrônico é um aparelho que produz sons em resposta a um controle de execução. Por sua vez, um instrumento inteligente, compartilha o controle da música com o compositor/performista. Um instrumento pode ser considerado inteligente se, ao receber as entradas provenientes do músico, produz mais resultados do que poderia ser obtido se feito por um instrumento tradicional [YAV 92]. A figura 214 apresentada um protótipo de instrumento inteligente construído para exemplificar essa idéia. O programa possui escalas e arpejos armazenados na memória que são disparados ao primeiro toque no teclado. A partir daí o executante controla o sistema através da intensidade do toque e das notas escolhidas.

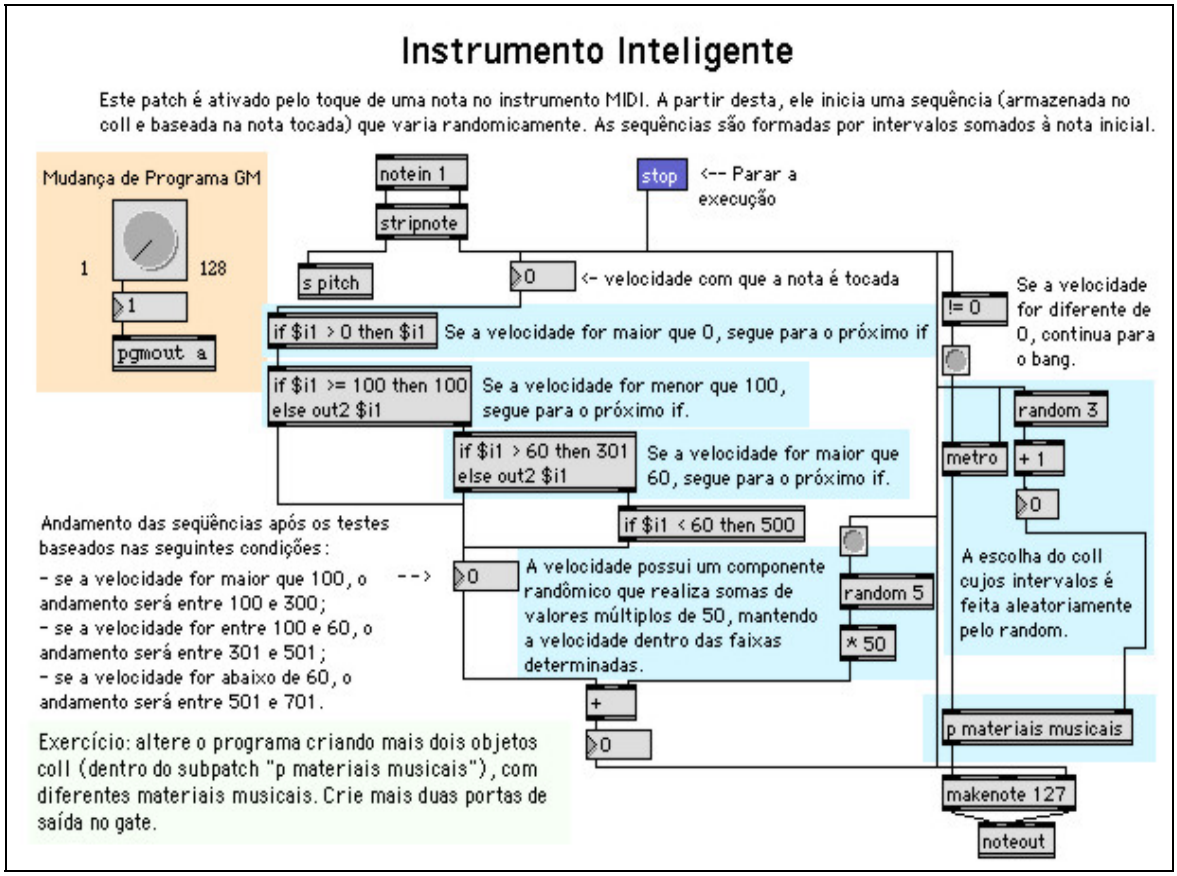


Figura 214 - Instrumento inteligente

O objeto ***random*** presente no canto superior direito do *patch* é responsável pela seleção de uma das seqüências contidas no *subpatch* "materiais musicais". Ele escolhe aleatoriamente um número dentro da faixa especificada pelo seu argumento, o qual é somado um (isso porque o objeto "gate" no *subpatch* não reconhece o zero). O mesmo objeto ***random*** influencia o ritmo com que as seqüências automáticas serão tocadas, um número entre 0 e 5 é sorteado e este é multiplicado por 50 para manter o valor dentro das faixas determinadas. O resultado desta multiplicação é somado com um dos valores determinados pelos objetos "if", e este valor é enviado ao inlet de duração do makenote.

O objeto "s pitch" no *patch* acima está enviando a nota tocada para dentro do *subpatch* "materiais musicais", atuando na soma que faz com que uma nota cause a execução de seqüências, como visto na figura 215.

O material musical utilizado no instrumento esta armazenado em objetos ***coll*** localizados no *subpatch* "materiais musicais". Ao clicar duas vezes sobre este *subpatch*

podemos visualizar como ele foi construído. Observe a figura 215 para entender melhor como o material musical armazenado é utilizado pelo programa.

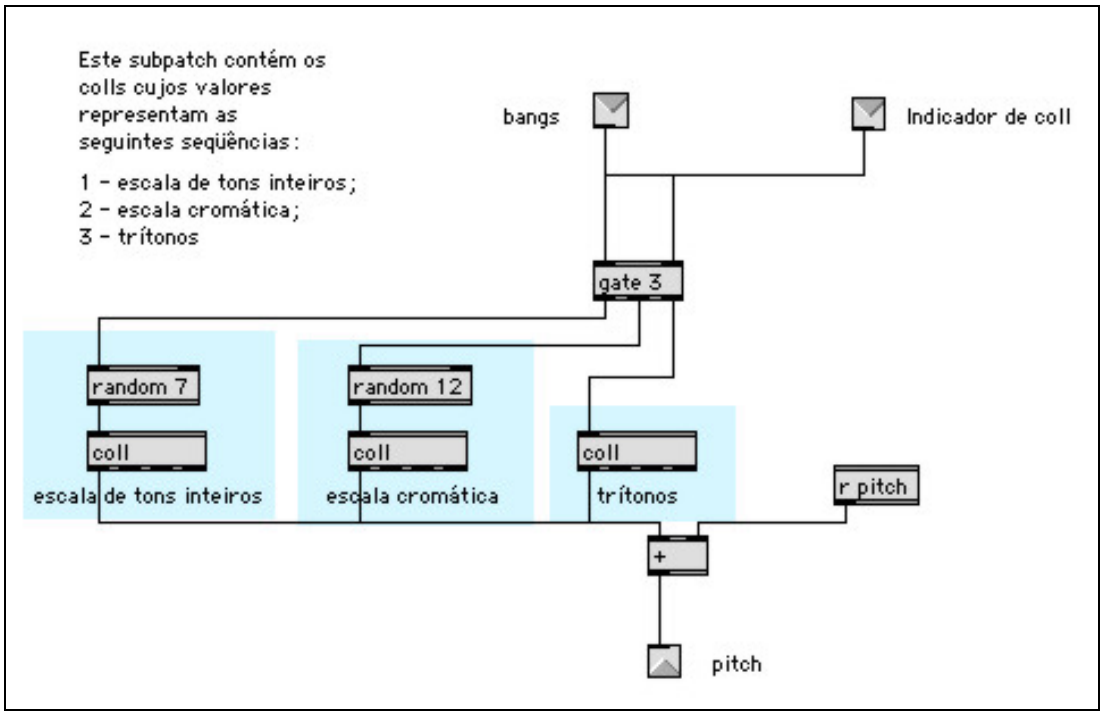


Figura 215 – Subpatch "materiais musicais", do patch Instrumeto Inteligente

Utilize o programa para realizar uma performance auxiliada por computador. Para entender melhor como foi programado o instrumento inteligente, procure fazer o exercício indicado.

Uma possível solução para o problema pode ser visualizada na figura 215. Nessa solução, o *patch* principal não é alterado. Porém, conforme mostra a figura 216, o *subpatch* (subprograma) possui alterações em relação ao original.

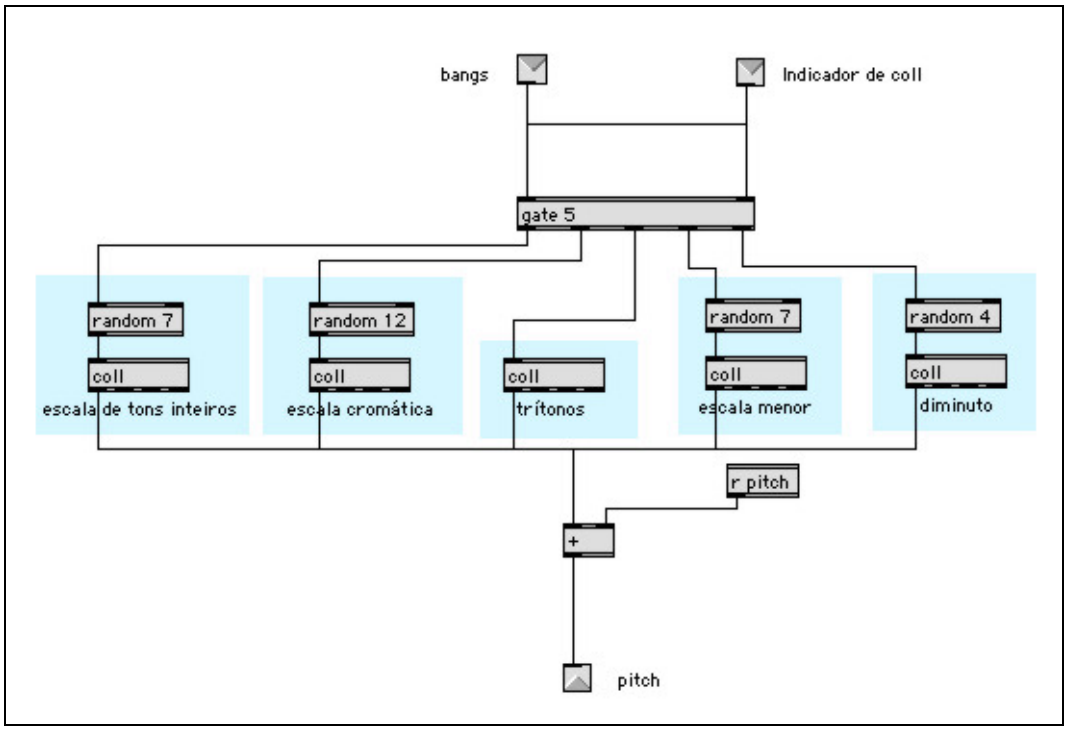


Figura 216 – *Subpatch* "materiais musicais" da resposta do *patch* Instrumento Inteligente

Composição Automática

A composição automática é o processo de geração musical feito pelo computador em tempo real. Ao contrário da composição interativa, que permite a interferência do usuário durante o processo de criação, a composição automática realiza toda a geração da música a partir de entradas fornecidas pelo usuário, não permitindo interferência humana ao longo do processo de geração musical [YAV 92].

Na figura 217 é apresentado um programa que gera uma composição automática por computador. Para ativar o programa, basta o usuário clicar em *inicia*. Utilizamos, neste exemplo, o objeto ***clocker***, que funciona de forma semelhante ao ***metro***, porém ao invés de enviar *bangs* para a saída em intervalos regulares de tempo ele envia o tempo transcorrido desde o instante de sua ativação. Com esta informação você pode fazer com que certos valores sejam modificados de alguma forma relacionada com a passagem do tempo. Utilize o patch para produzir frases musicais automaticamente e estude as ligações entre os objetos. Depois de estudar o patch leia o enunciado do exercício e tente resolver.

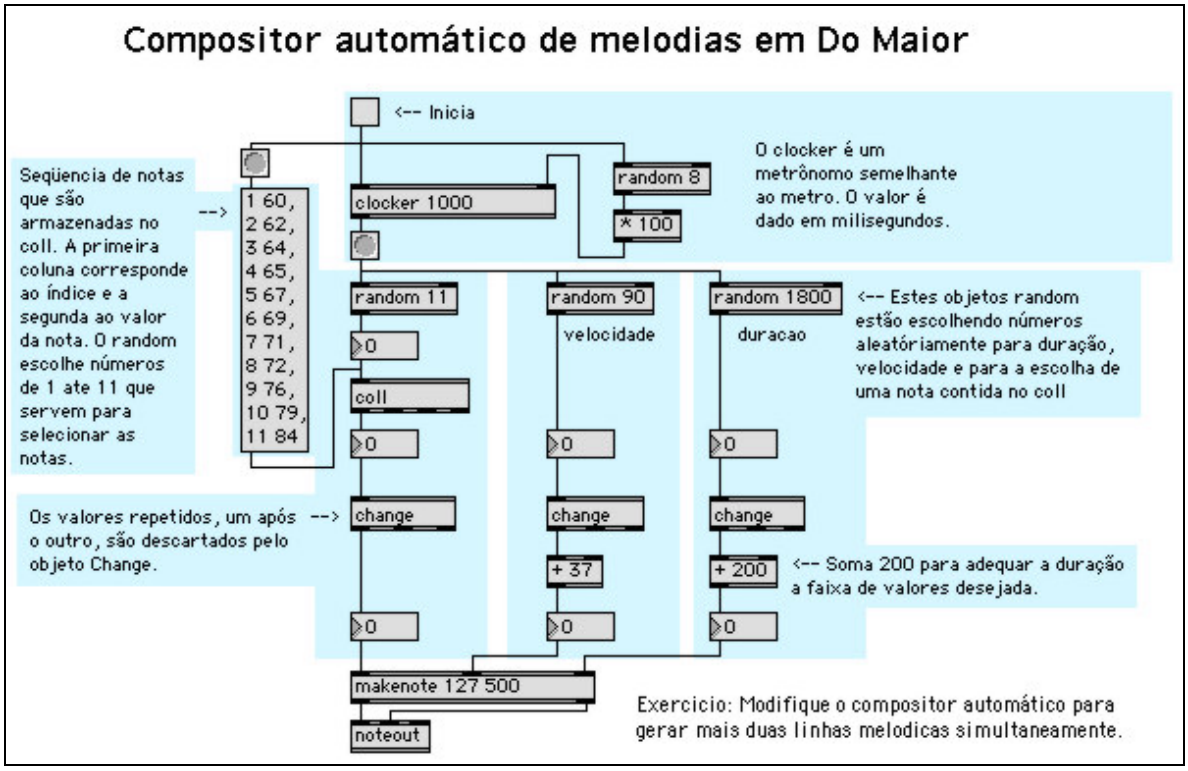


Figura 217 – Geração automática de alturas

Uma possível solução para o exercício é apresentada na figura 218. Observe que nesta solução, foram utilizados exatamente os mesmos dados nos objetos **coll**. Portanto, o que varia é a escolha dos índices das tabelas de notas. Além disso, os pulsos do metrônomo (clocker) são enviados para três patches que selecionam as vozes que serão enviadas. Nesta solução também é possível controlar a intensidade das notas através do teclado MIDI externo. Nesta solução, parte do programa é encapsulado em um *subpatch*. Com isso, é possível modularizar o programa e criar subrotinas para a execução de tarefas específicas. Além disso, o programa torna-se mais inteligível pois o espaço da tela é melhor aproveitado. Nesta solução os três *subpatches* são iguais, logo, é apresentado, na figura 219, apenas um *subpatch*. Nele pode ser visualizada a rotina para a seleção das alturas.

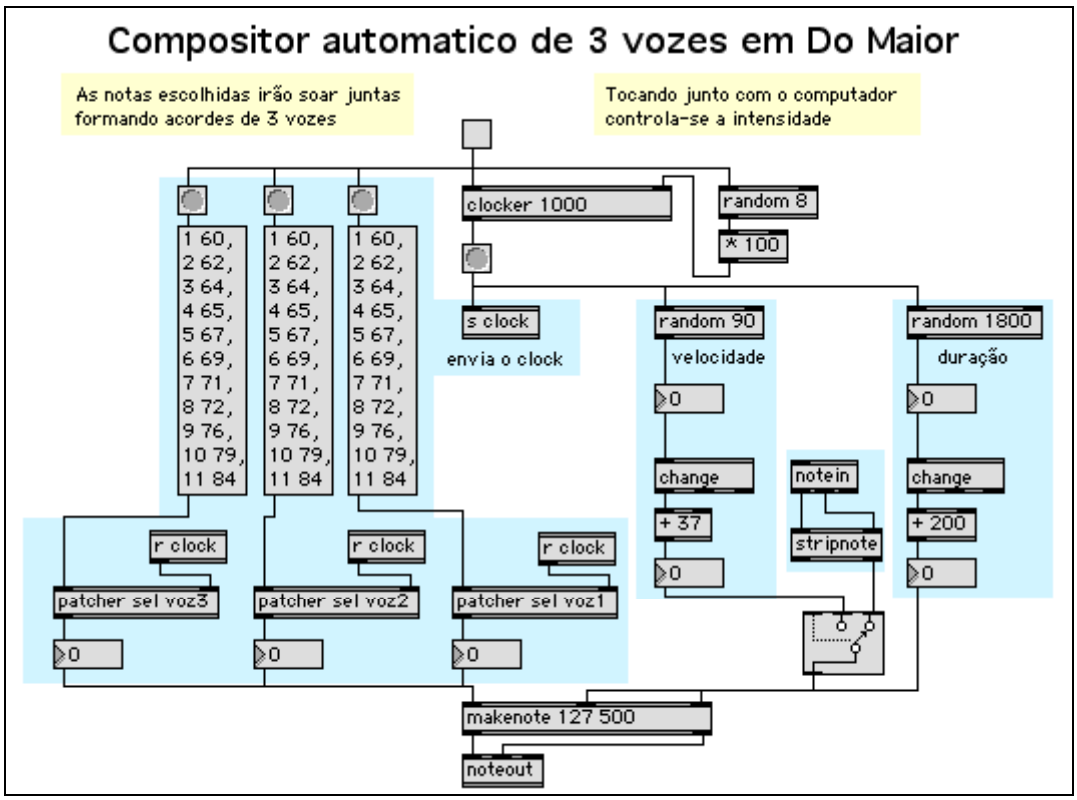


Figura 218 - Compositor automático de 3 vozes

Figura 219 – Subpatch para a seleção das alturas

8.1.3 AULA 3 - Composição Algorítmica

Programas para a composição algorítmica destinam-se à redução de componentes musicais para níveis elementares e procedurais que servem como construtores de módulos para descrever qualquer estrutura. Algoritmos podem ser aplicados em qualquer estágio da hierarquia musical de uma composição.

O processo de composição pode ser descrito como: a arte de fazer escolhas musicais. Se é possível descrever os princípios fundamentais dessas escolhas em um conjunto de regras, então o computador é uma excelente ferramenta para manipular este tipo de conhecimento em regras de uma maneira poderosa.

Composição Algorítmica consiste na criação de uma rede de regras e procedimentos que permitam que o computador faça escolhas na produção musical [YAV 92].

Novos algoritmos estão sendo constantemente incorporados a sistemas musicais, muitos deles provindos do mundo da ciência. A lista de algoritmos pode estender-se indefinidamente com o passar do tempo. Alguns dos principais autores e seus métodos algoritmos para composição podem ser encontrados em [XEN 92], [WIN 87], [WIN 90], [WIN 91]. Um exemplo clássico de algoritmo para composição musical é o Random Walk. Observe no exemplo da figura 220 como pode ser feita a programação visual desse algoritmo.

A cada execução do programa Random Walk obteremos resultados diferentes. Assim, ao dispararmos o funcionamento do programa, um intervalo qualquer é sorteado pelo objeto *random 12* e é armazenado no objeto *int*. O objeto *int* funciona como uma memória (basta introduzir o valor no inlet da esquerda para que ele seja armazenado), ao mesmo tempo que esse valor é enviado e tocado pelo *makenote*. Na continuação do programa, um novo valor para o intervalo é calculado, enviado pelo *s(end) intervalo*, recebido pelo *r(eceive) intervalo* e gravado no *int*. Após 500 milisegundos, o *metro* dispara o *int* que por sua vez libera no outlet o valor anteriormente calculado para que ele seja tocado pelo *makenote*, e assim sucessivamente.

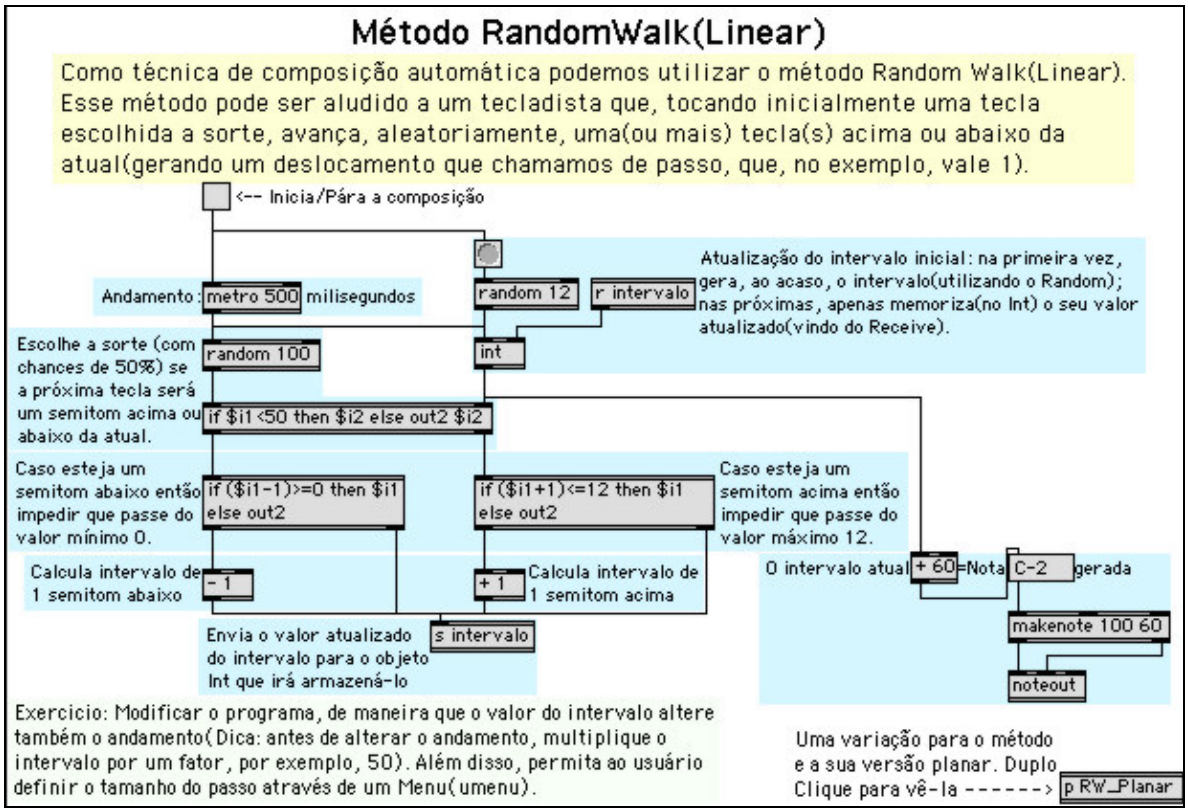


Figura 220– Random Walk Linear

A figura 221 apresenta uma amostra do resultado obtido quando o patch é acionado.

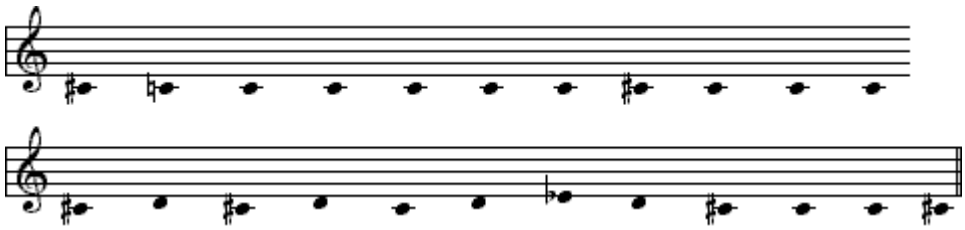


Figura 221 – Resultado obtido a partir da execução do programa Random Walk Linear

Resolva o exercício proposto no exemplo do Random Walk. Observe que o valor do intervalo deverá alterar o tempo das notas. Portanto, o programa irá utilizar o algoritmo de Random Walk para gerar diferentes durações.

Observe a solução para este exercício na figura 222 e compare com a sua resposta.

Na solução do exercício utilizou-se o objeto *umenu* para apresentar um menu com as opções de passo. Cada uma das opções do menu deve ser digitada (separadas por vírgulas umas das outras) no campo "Menu Text" das propriedades do objeto (o "Get Info..."), de maneira que durante a execução do programa, a primeira opção do menu tem a ela associado o índice 1, a segunda o índice 2, a terceira o índice 3, e assim por diante. Logo, a opção "Semitom" corresponde ao valor 1, "Tom" corresponde a 2, "Tom e meio" a 3, sucessivamente, sendo que essa associação representa exatamente o valor do intervalo musical de cada opção. Quando o programa é executado, o valor do intervalo (ou seja, o passo) escolhido no menu é transmitido e armazenado no interior dos objetos *+(add)* e *–(subtract)*, e o programa segue para o cálculo do intervalo a ser tocado. Após decidir se o intervalo aumenta ou diminui de um passo na próxima nota tocada, o programa calcula esse novo intervalo, somando ou subtraindo o valor do passo do valor numérico do intervalo, e armazenando esse valor em *int*, como já foi explicado.

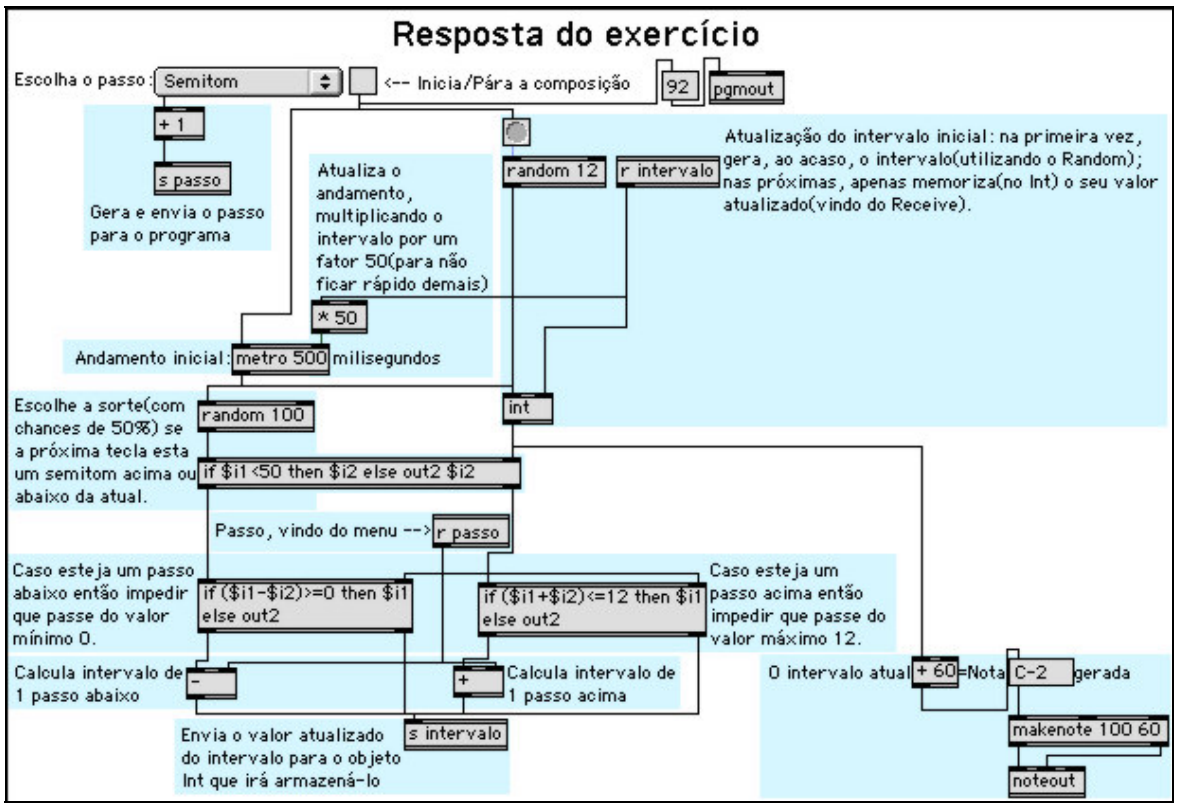


Figura 222 – Geração de alturas e durações através do Random Walk

Um fragmento musical do programa Random Walk da figura 222 é apresentado na figura 223.

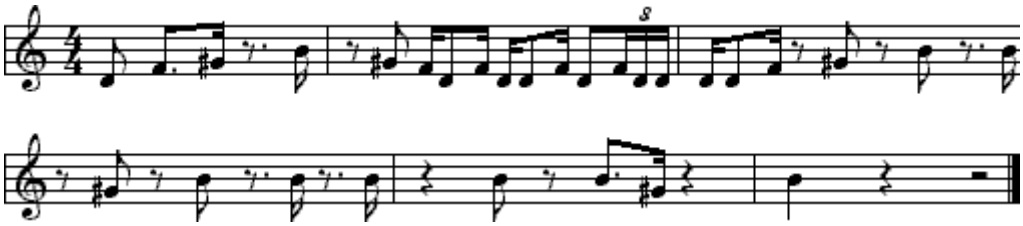


Figura 223 – Resultado obtido a partir da execução do programa Random Walk para a geração de alturas e durações

A utilização de vetores e matrizes em Max deve ser implementada utilizando-se o objeto ***coll.*** Conforme já visto anteriormente, o objeto **coll** é uma tabela onde podem ser armazenados o índice e o valor associado. Os valores da tabela são recuperados através do acesso ao índice da tabela. Para construirmos uma matriz, é necessário utilizar mais de um ***coll***. Isso porque cada ***coll*** possui apenas uma dimensão (vetor) e a matrizes pode ter duas ou mais dimensões.

O objeto *drunk* do Max, utilizado no exemplo da figura 220, realiza a função do Random Walk Linear já estudado anteriormente. No entanto, para outras extensões do algoritmo Random Walk, como o Random Walk Planar da figura 224, é necessário programação.

O Algoritmo do Random Walk Planar pode ser utilizado como um bom exemplo para mostrar o uso de matrizes em Max. Observe o funcionamento do exemplo 14. O objeto *drunk* da esquerda varre as linhas da matriz e o objeto *drunk* da direita varre as colunas da matriz. Utilizando 5 objetos **coll**, no qual cada um deles contém 6 notas, teremos o dimensionamento da matriz em 5x6 (5 colunas x 6 linhas), onde foi armazenado o conjunto das notas 60 (Dó da 3ª) até 89 (Fá da 5 ª). Lembrando que o Random Walk Planar do exemplo "caminha" em passos de 1 grau musical tanto na horizontal e vertical quanto nas diagonais, uma transição pode gerar intervalos melódicos de 1 grau se o passo acontecer em uma linha (pois entre cada nota da linha o intervalo vale 1), 6 graus se o passo se der entre duas linhas (como cada linha possui 6 notas, os elementos de cada coluna diferem de 6 graus), 7 graus se o passo ocorrer numa diagonal principal e 5 graus numa diagonal secundária. Observe a figura abaixo para um exemplo. Suponhamos que a última nota tocada seja 74 (Ré da 4 ª). Se o passo ocorrer para 73 (Dó# da 4 ª), teremos um intervalo de 1 grau, e se a próxima nota for 68 (Sol# da 3 ª), teremos 6 graus. Caso a próxima nota seja 67 (Sol da 3 ª), teremos 7 graus, e se for 69 (Lá da 3 ª) teremos 5 graus(veja a figura **MAT** abaixo).

	1	2	3	4	5	6
1	60	61	62	63	64	65
2	66	67	68	69	70	71
3	72	73	74	75	76	77
4	78	79	80	81	82	83
5	84	85	86	87	88	89

Agora, leia atentamente as explicações do patch e tente entender como ele foi elaborado. Para resolver o exercício proposto você deve alterar as dimensões da matriz e o conteúdo dos objetos *coll* armazenando suas próprias séries. Para alterar o conteúdo de cada *coll* clique duas vezes com o mouse sobre o objeto e escreva o índice e o valor das notas MIDI.

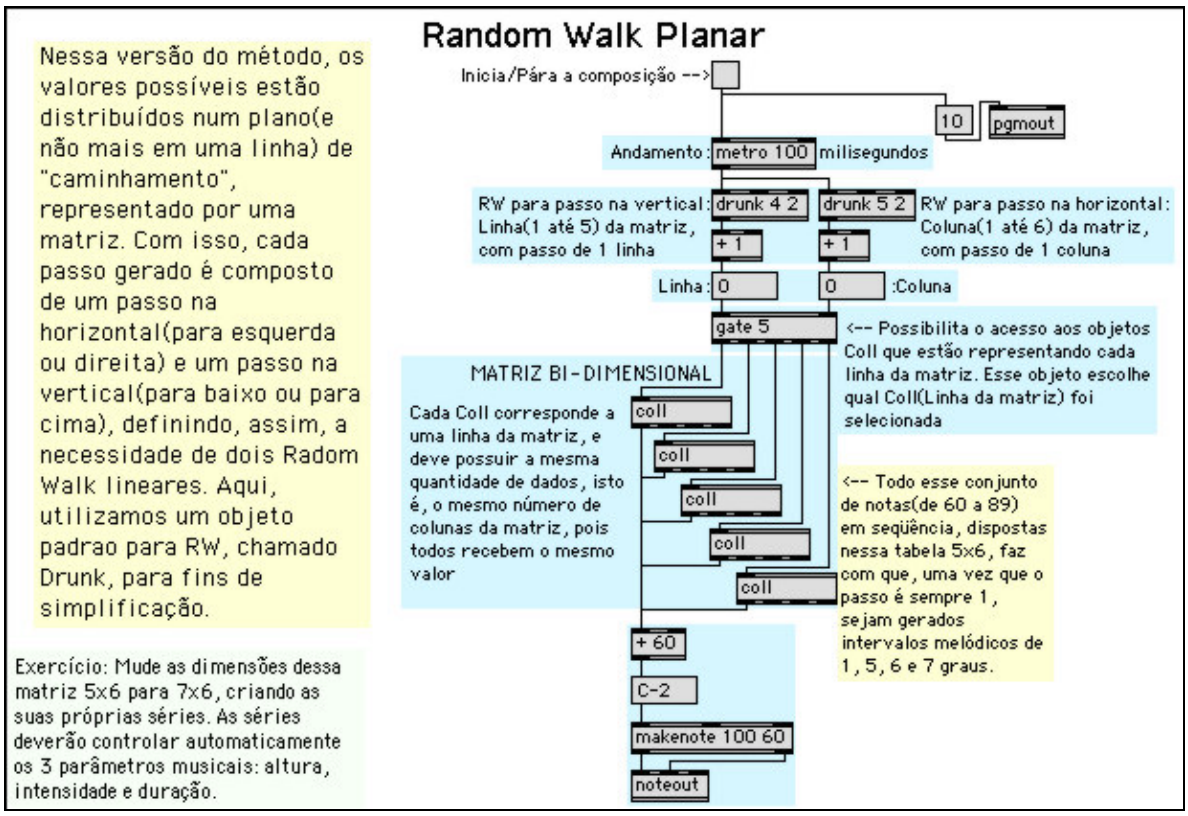


Figura 224 – Random Walk Planar implementado através de matrizes com a utilização do objeto ***coll***

Compare sua solução com a da figura 225. Observe que na figura 225 a Matriz tem as dimensões de 7x6 porque foram estipulados 7 objetos **colls**, sendo que cada um deles com 6 notas armazenadas. Entretanto, é importante lembrar que os valores das notas armazenados na matriz são números que podem ser utilizados em quaisquer cálculos. Assim, cada valor gerado pelo Random Walk Planar sofre um cálculo(quando necessário) a fim de se enquadrar as valores dos parâmetros que ele vai controlar. No exemplo, utilizamos um único Random Walk Planar para controlar ao mesmo tempo os parâmetros de altura, intensidade e duração de notas. Para o caso da altura e da duração de cada nota, não é possível apenas transpor os valores da matriz para a oitava desejada, como fazemos para o objeto *makenote*, transpondo os valores para a oitava central(somando 60 ao valor do intervalo). Dessa forma, a solução proposta ajusta o valor recebido para a intensidade somando o valor 86 e, para a duração, multiplicando o valor 10, pois os resultados gerados por estes cálculos ficam limitados aos valores aceitáveis dos parâmetros (ou seja, mínimo de 0 para a intensidade e duração e o máximo de 127 para a intensidade).

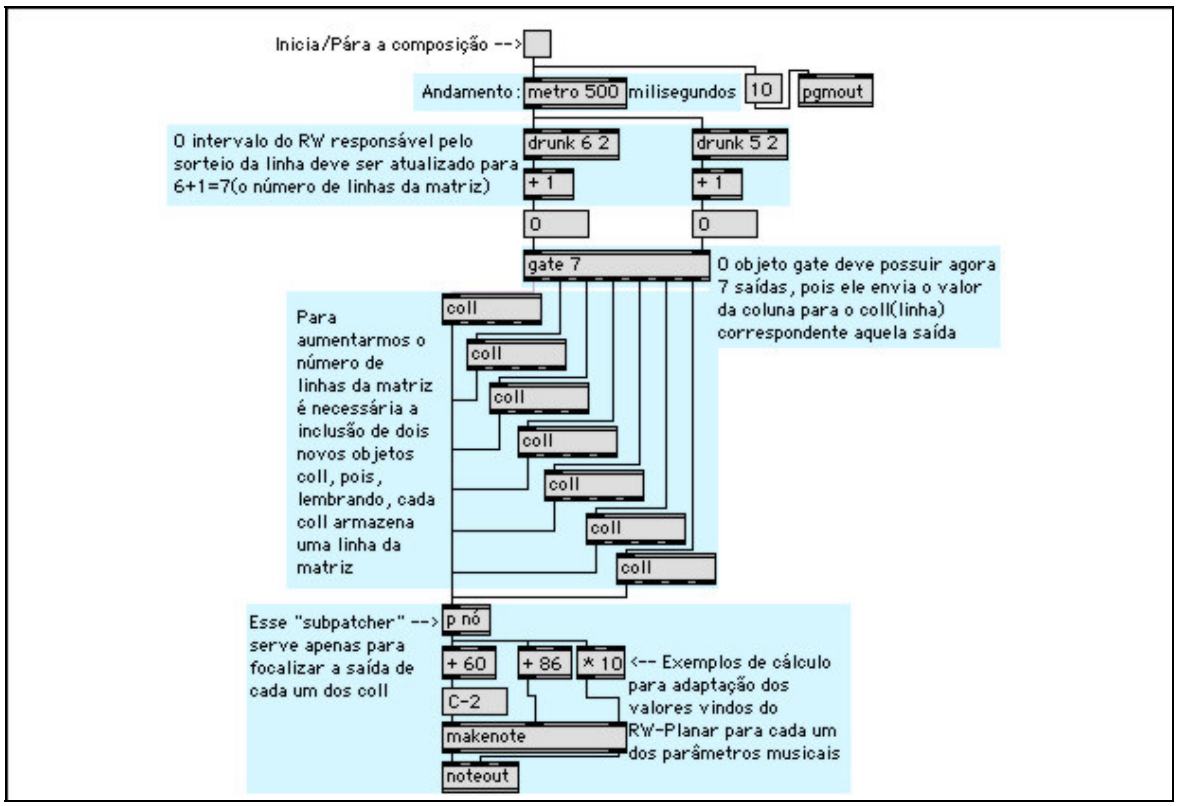
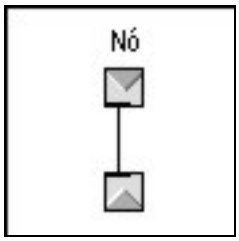


Figura 225 – Random Walk Planar com dimensões 7x6 para controlar altura, intensidade e duração



O “subptach nó” evita que futuras alterações no *subpatch* (tais como a utilização do Random Walk Planar para outras finalidades do que a geração de notas) venham a desfazer as conexões que cada ***coll*** deve ter com os elementos a eles conectados.

8.1.4 AULA 4 - Composição Determinística Usando Motivos

Um motivo musical é um pequeno fragmento melódico/rítmico de um tema musical. Pode ser formado somente por três ou quatro notas que podem ser independentemente variadas e manipuladas. Willi Apel escreveu que na música de Bach e Beethoven, "motivos são os blocos bem construídos ou células germinativas da composição musical" [ROA 96].

Para assegurar variedade suficiente na música construída através de motivos, estes são sujeitos a um conjunto de técnicas de variação que nada mais são do que algoritmos passíveis de serem utilizados em sistemas computacionais. Algumas das variações mais comuns de motivos incluem repetição, transposição, e alteração do contorno melódico e características rítmicas. A figura 226 mostra um motivo rítmico/melódico.

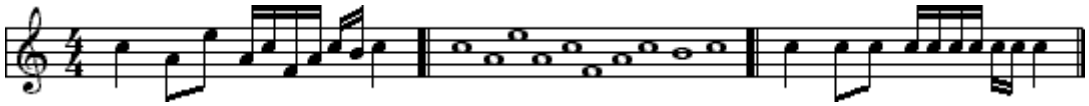


Figura 226 - a) Motivo musical b) Desenho Melódico c) Padrão Rítmico

A figura 227 mostra o efeito de transposição das notas de um motivo. Para realizar isso no computador, deve-se utilizar um algoritmo de transposição. Alguns seqüenciadores também dispões destes recursos automáticos.

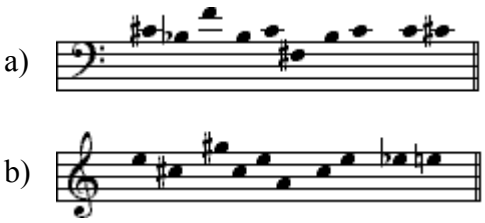


Figura 227 - a) Transposição das alturas do motivo uma Terça Maior acima. b) Transposição de um semi-tom acima

A figura 228 apresenta um patch em Max que realiza a operação de transposição de um motivo musical. No menu do programa é possível escolher a quantidade de semitons que será utilizada no processo de transposição. O usuário deverá tocar um motivo composto por apenas dez notas que serão armazenadas. Se forem tocadas mais de 10 notas, o processo de contagem é reiniciado. O número de notas tocadas pode ser observado no contador logo abaixo do objeto *kslider*. A opção clear irá apagar as notas armazenadas para que uma nova seqüência de 10 notas seja tocada. Logo abaixo do menu que escolhe o tipo da transposição, existe uma opção para regular o andamento da execução do motivo. Utilize essa opção para selecionar o andamento desejado antes de ouvir o resultado da transposição. Estude o patch e depois realize o exercício indicado.

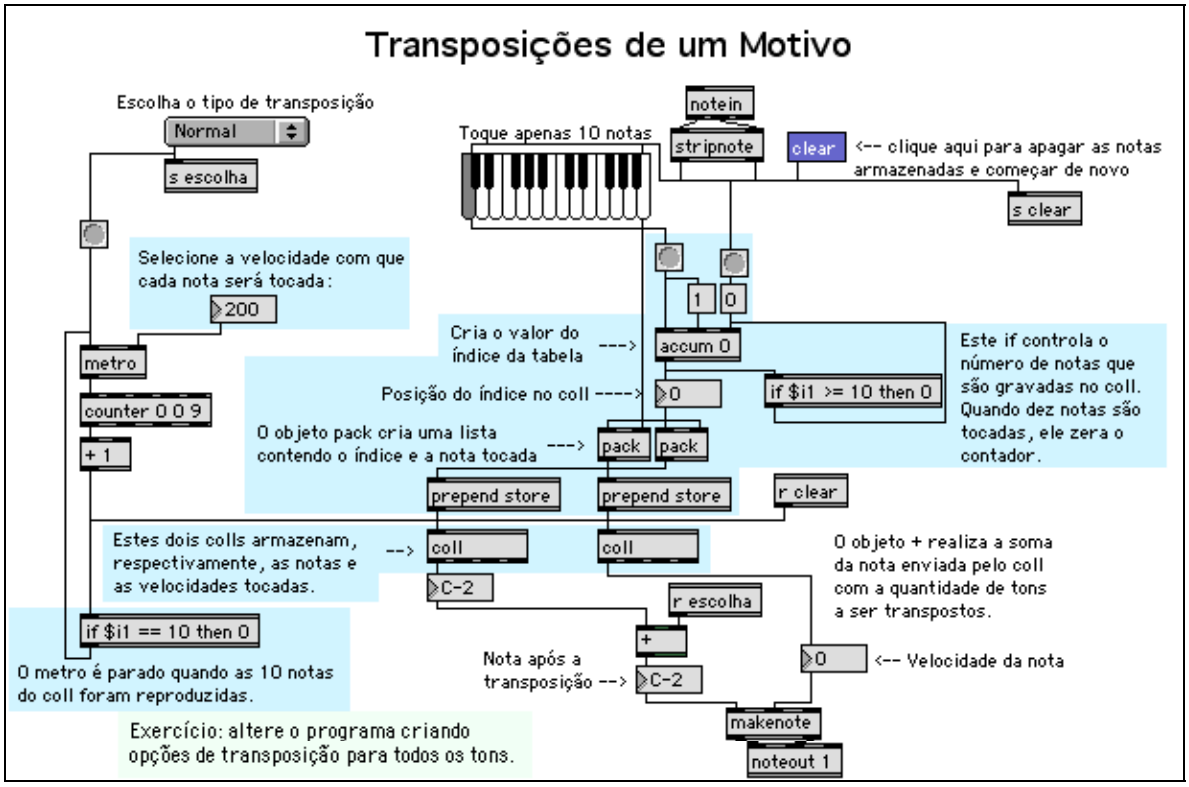


Figura 228 – Transposições de um Motivo

O exercício proposto no patch acima é de fácil solução. Como o tipo de transposição é selecionada pelo objeto **umenu**, deve-se adicionar mais quantas opções forem necessárias para que o programa seja capaz de transpor para todos os tons. Como o número enviado por este objeto no momento da escolha representa o seu próprio intervalo (em MIDI), nada mais é necessário.

O computador também pode ser programado para realizar operações como deslocamento de registros conforme a figura 229.

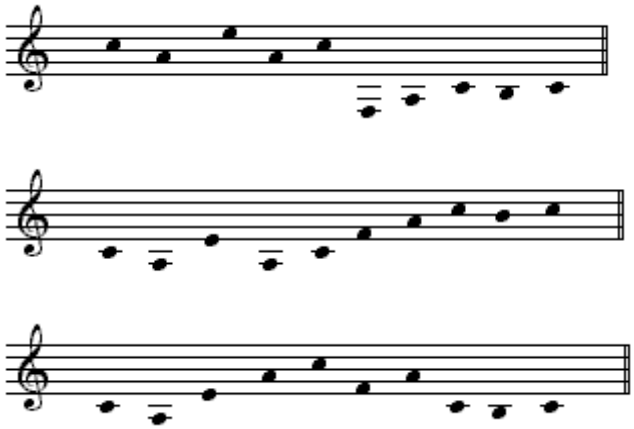


Figura 229 - Três deslocamentos dos registros de um motivo

A figura 230 apresenta um exemplo que implementa o deslocamento dos registros de um motivo musical. O programa possui uma estrutura semelhante aos já vistos para facilitar o entendimento. Para utilizá-lo, o usuário deve tocar apenas dez notas para serem armazenadas e transformadas através do patch. No menu é possível escolher a opção de deslocamento. Utilize o patch para tocar um motivo composto por dez notas e realizar operações de deslocamento de registros.

O objeto **coll** está sendo utilizado para armazenar as notas (e suas respectivas velocidades) tocadas no objeto **kslider** ou no instrumento MIDI externo. Para que isto ocorra, devemos fornecer a nota/velocidade junto com o índice apropriado. Este índice é gerado pelo objeto **acumm**, que é iniciado com o valor 0 ao qual é somado 1 a cada nota tocada. A saída do objeto **accum** é enviada ao **inlet** esquerdo dos objetos **pack**. Este objeto forma listas a partir de suas entradas, ao receber o índice do **accum** e a nota/velocidade de uma das fontes, uma lista do tipo "índice nota/velocidade" é enviado ao objeto **prepend**. Por sua vez, o **prepend**, cuja função é adicionar caracteres ao início de qualquer mensagem que receba, adiciona a palavra *store* à lista. Tornando-a "*store índice nota/velocidade"*, uma mensagem que fará o objeto **coll** salvar a nota/velocidade com seu índice. Perceba que há dois objetos **coll**, um para as notas e outro para as velocidades, ambos recebem o mesmo índice para que toquem a nota com sua velocidade correspondente. Estude o programa buscando um entendimento do mesmo.

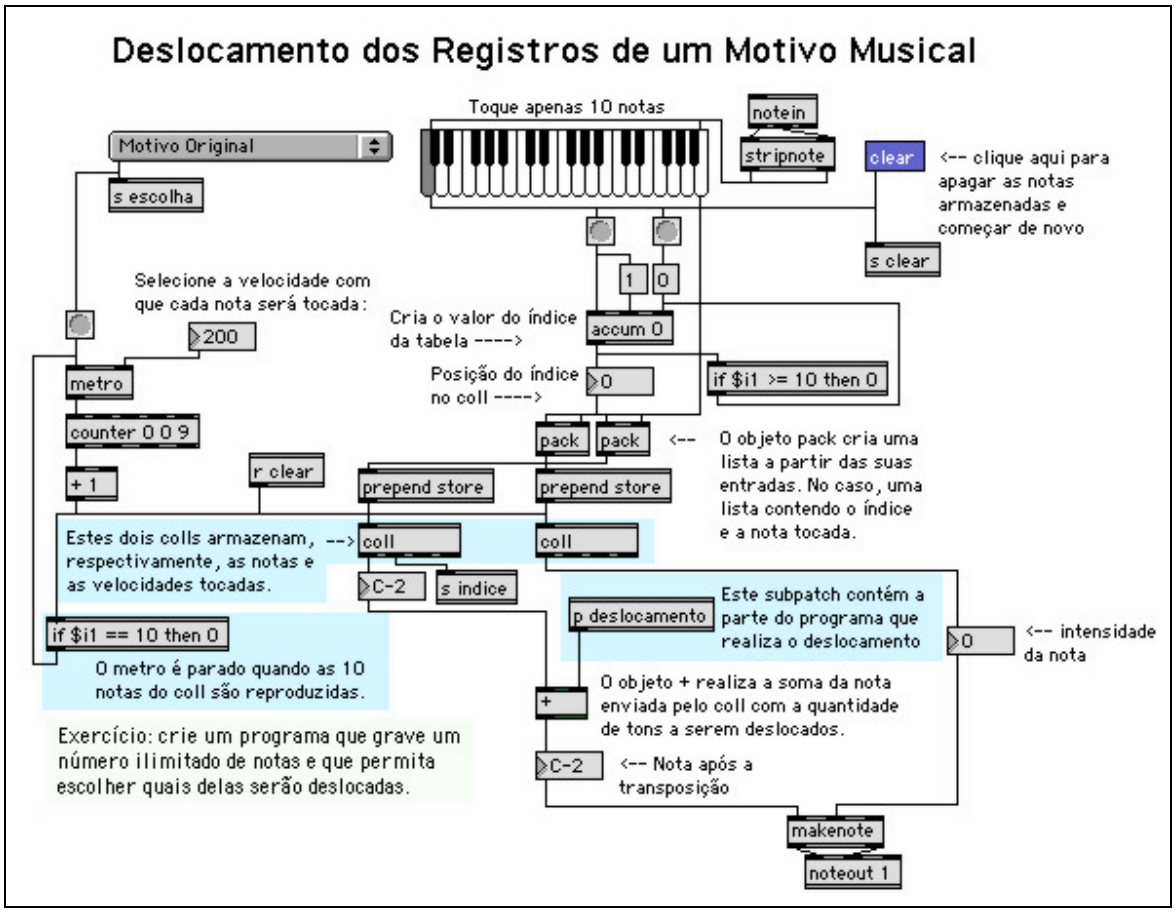


Figura 230 – Programa que produz o deslocamento dos Registros de um Motivo Musical

Para estudar como foi feita a programação desse patch é necessário entender o funcionamento do subpatch. Para isso clique duas vezes sobre o subpatch "deslocamento". A figura 231 mostra as ligações entre os objetos **gate** e **if** utilizados para seleção de parâmetros.

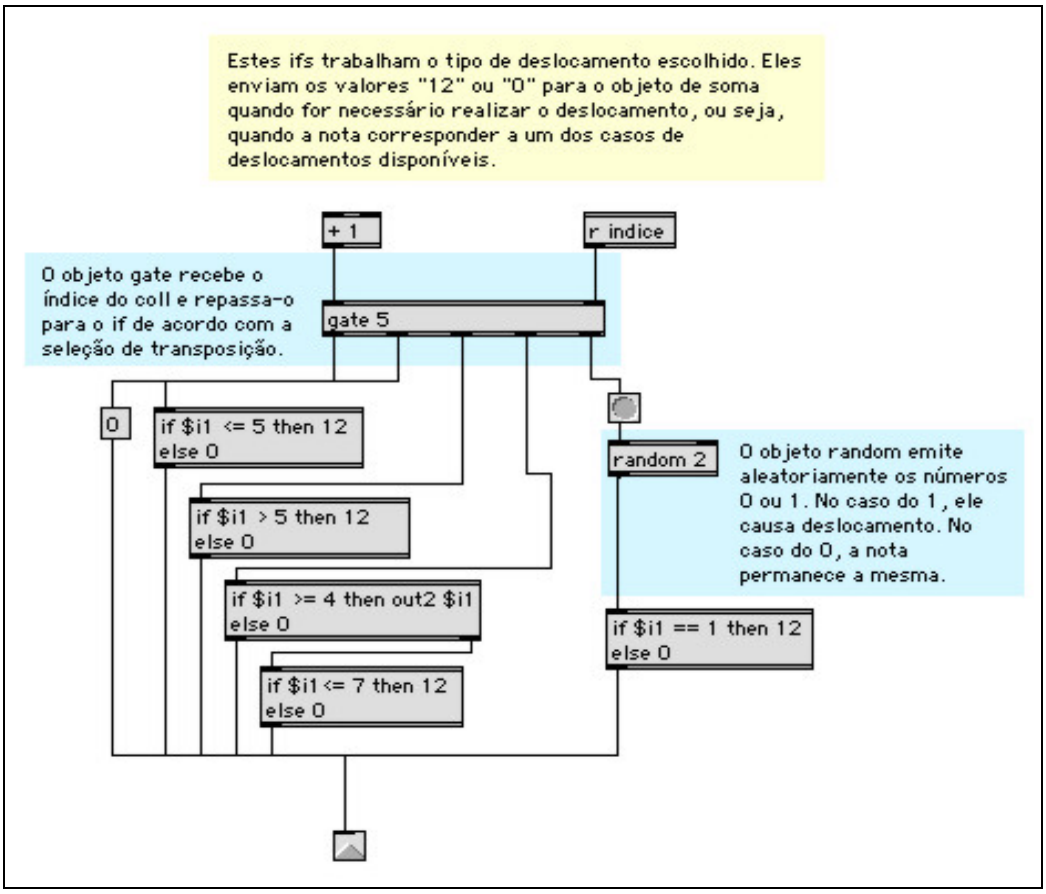


Figura 231 – Subpatch “deslocamento” do patch de Deslocamento dos Registros de um Motivo Musical

Depois de compreender o programa realize o exercício criando seu próprio programa para solucionar o problema. Um exemplo de como isso pode ser feito pode ser observado na figura 232.

Nesta solução o usuário precisa marcar no quadro de notas aquela que será deslocada (oitavada). Procure utilizar o programa e comparar os resultados e a programação com a sua solução.

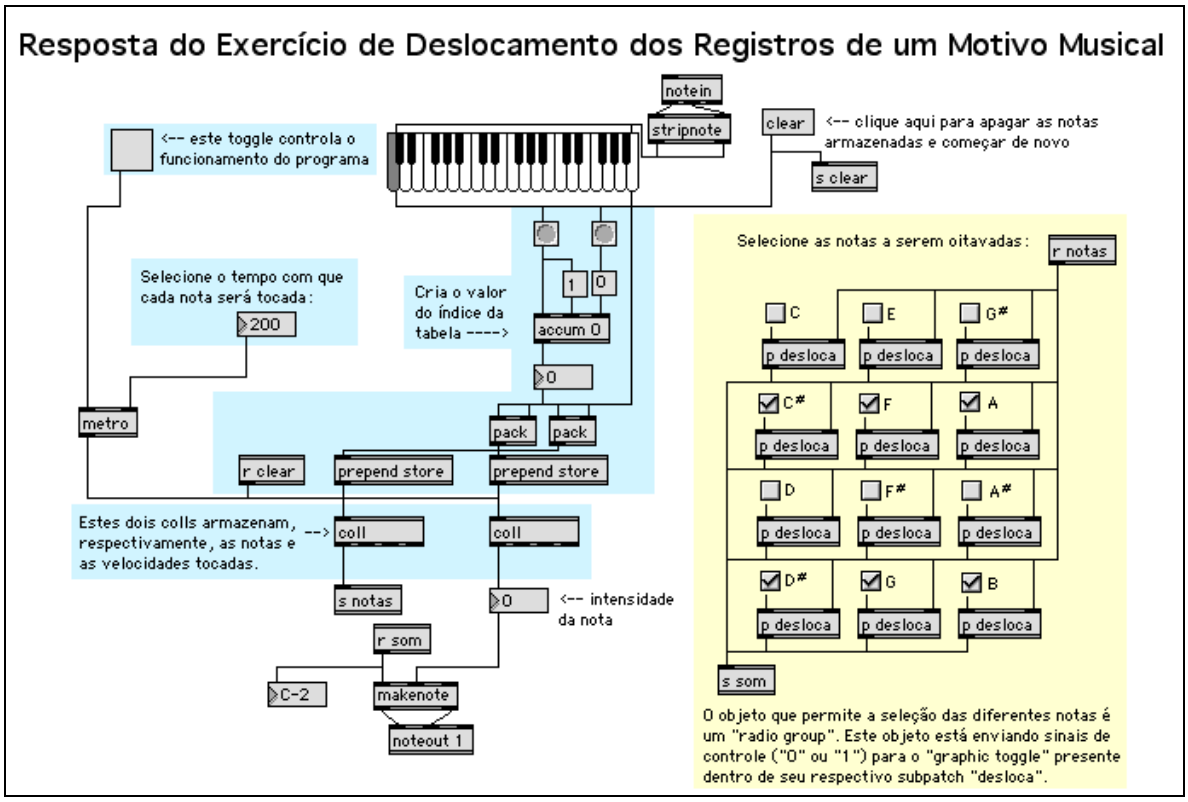


Figura 232 – Resposta do exercício do patch Deslocamento dos Registros de um Motivo Musical

Outra operação realizada com muita freqüência sobre um motivo musical é a inversão das alturas. A figura 233 ilustra essa operação.

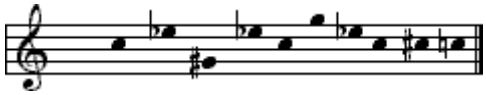


Figura 233 - Inversão das alturas de um motivo

Um programa de computador para a inversão das alturas de um motivo pode ser exemplificado através do exemplo da figura 234. O usuário toca as notas no *kslider* que são, por sua vez, armazenadas em um ***coll***. Para efetuar esse armazenamento, existe um *accum* (que serve para somar e armazenar números) responsável por incrementar, de um, o seu valor, toda vez que uma nota é tocada. Esse valor é então utilizado para indexar a posição do ***coll*** no qual o valor numérico da nota é armazenado, através do envio de uma mensagem com tal posição e o valor que deve ser armazenado nela. Assim, para criar essa mensagem, utilizamos o objeto *zl join* a fim de agregar o valor de posição vindo do *accum* com o valor da nota vindo do *kslider*, criando uma única mensagem para o ***coll***. Quando a parte do programa responsável pela inversão do motivo é acionada, o ***coll*** utilizado para armazenar as notas tocadas é copiado para outros ***coll*** axiliares, utilizando as mensagens "write" e "read" do objeto, que efetuam as operações de escrita do ***coll*** principal para um arquivo e, a seguir, a leitura desse

arquivo com os dados para o ***coll*** auxiliar. Essas mensagens do tipo operação, que instruem o ***coll*** principal a gravar os seus dados em arquivo, são enviadas a ele através de um *s(end)* intitulado "operação", a fim de evitar a poluição visual do programa com mais ligações. É importante ressaltar que, algumas vezes, essas operações de escrita e leitura podem ser um tanto demoradas, que é o caso do exemplo. Por isso é utilizado o objeto *pipe*, pois ele é capaz de atrasar a propagação de todos os dados que passam por ele por um determinado período de tempo, que, no caso, é de 95 milisegundos. Graças a esse atraso ser suficiente, podemos garantir que os dados utilizados no programa estão corretos, pois não há perigo de serem utilizados valores desatualizados. Feito isso, o programa calcula o inverso de cada nota utilizando os ***coll*** auxiliares que simplificam o programa, pois cada um armazena um parâmetro diferente para o cálculo do inverso. O cálculo segue a definição de inversão musical, isto é, colocar de trás para frente os intervalos entre as notas do motivo. Para tanto, é necessário utilizarmos o valor da nota que está sendo invertida, o valor da nota anterior a ela no motivo, e o valor da última nota do motivo invertido. Subtraindo o valor numérico da nota atual do valor da nota anterior do motivo, temos o intervalo musical entre elas. Ao subtrairmos esse valor da última nota invertida, estamos gerando o valor inverso da nota atual. Após a inversão da nota, ela é armazenada na mesma posição que ela ocupava no ***coll*** original antes de ser invertida. Esse procedimento é efetuado utilizando-se o objeto *zl join* para criar uma mensagem de armazenamento para o ***coll***, indicando a posição da nota invertida juntamente com o seu valor. Leia atentamente as explicações do patch observando as ligações entre os vários objetos.

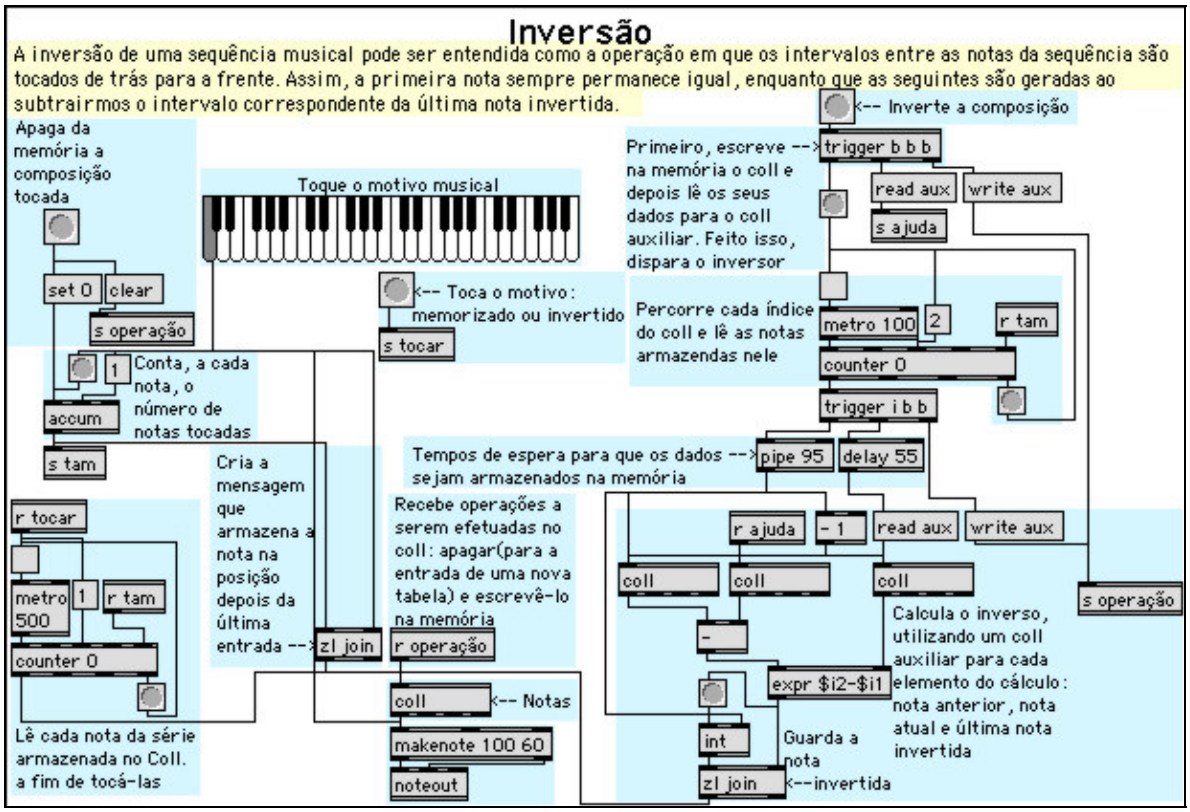


Figura 234 – Programa exemplo para a inversão das alturas de um motivo

A operação de Retrógrado de um motivo pode ser exemplificada na figura 235. O retrógrado é o inverso da ordem das notas do motivo.

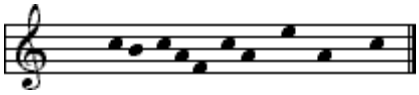


Figura 235 - Retrógado do motivo

O exemplo da figura 236 mostra como pode ser programada a função de Retrógrado de um Motivo através da inversão da ordem das alturas. O ***coll*** responsável por armazenar as notas do motivo(intitulado " Notas") tocado no *kslider* recebe uma ligação do objeto *r(eceive) operação*. Essa ligação é responsável por simplificar e centralizar as mensagem de operação destinadas ao ***coll***. Como no exemplo da inversão, cada nova nota tocada na composição incrementa de 1 o valor do contador *accum*, para que seja possível saber de quantas notas é composto o motivo. Assim, toda vez que uma nota é tocada, o valor armazenado no *accum* é repassado para o objeto *counter* e para a expressão *expr $i2+1-$i1*, através da conexão efetuada pelo *s(end) tam* e *r(eceive) tam*. Esse valor indica o tamanho, em notas, do motivo. Quando a parte do programa responsável pela criação do retrógrado do motivo é ativada, um **coll** auxiliar é criado, a fim de simplificar o cálculo do retrógrado. Isso é feito através de mensagens de operações, onde "write" faz com que o motivo seja primeiramente armazenado em um arquivo temporário, e, após isso, "read" efetua a leitura desse arquivo, carregando o motivo para o ***coll*** auxiliar. Feito isso, o *counter* percorre cada uma das posições(índices) do ***coll*** auxiliar a fim de calcular o valor da posição retrógrada de cada nota. Esse cálculo é feito invertendo a posição do ***coll*** através da subtração da posição do tamanho do motivo, ou seja, em um motivo de 10 notas (com tamanho 11, pois o ***coll*** pode endereçar a partir da posição 0), a nota que antes ficava na posição 1, passa a ficar

na 10(11-1=10), a nota da posição 2, na posição 9(11-2=9), e assim por diante(tamanho+1-posição). Por fim, o objeto *zl join* unifica a nova posição com o valor da nota daquela posição, gerando uma mensagem para que o ***coll*** principal armazene a nota na tal nova posição.

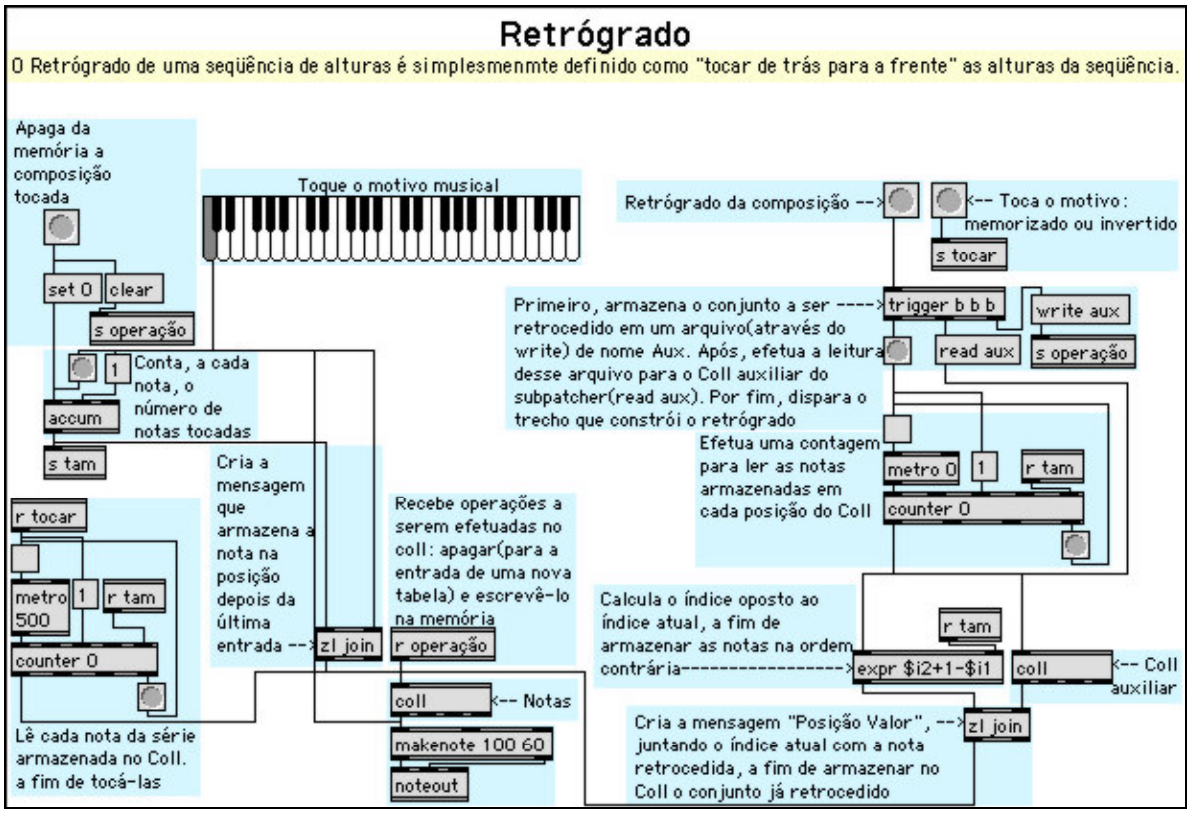


Figura 236 – Programa exemplo que realiza a operação de Retrógrado de um motivo

Outras operações padrões que podem ser realizadas em um motivo é a expansão e a contração. Essas operações são exemplificadas na figura 237. Observe que na expansão os intervalos são expandidos em relação ao motivo e na contração os intervalos são contraídos.

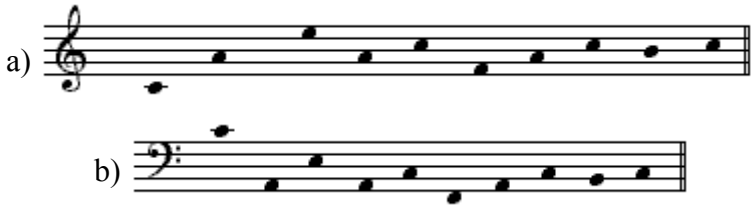


Figura 237 - a) Expansão b) Contração de intervalos

A figura 238 apresenta um patch que realiza a expansão e a contração de um motivo musical. Observe que a estrutura do programa é igual a dos programas anteriores. Este patch realiza a expansão ou contração da altura de um motivo tocado pelo objeto *kslider* ou pelo instrumento MIDI externo, que foi gravado nos objetos ***coll***. Estes dois objetos ***coll*** armazenam as alturas e as intensidades, respectivamente. A operação sobre a altura original se dá pela soma ou subtração do valor 12 ao valor MIDI da nota, correspondendo a um tom acima ou abaixo da nota inicial. Note que a primeira nota da seqüência é mantida no tom original. Execute o programa, estude como foi feito e depois realize o exercício.

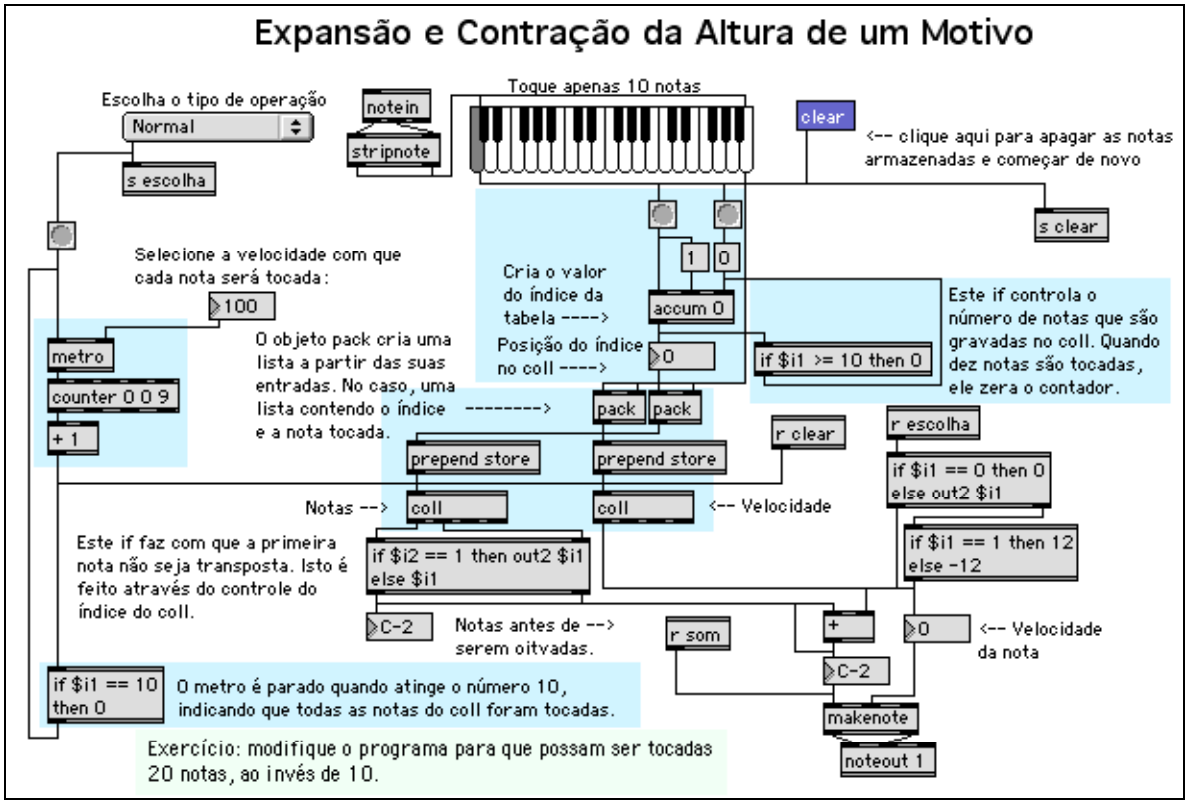


Figura 238 – Programa para Expansão e Contração da Altura de um Motivo

A solução para o exercício pode ser vista na figura 239. Observe que nesta versão modificada do programa foi utilizado um contador (objeto *counter*) com valores diferentes daqueles do programa anterior. Procure comparar a sua solução com a apresentada na figura 224.

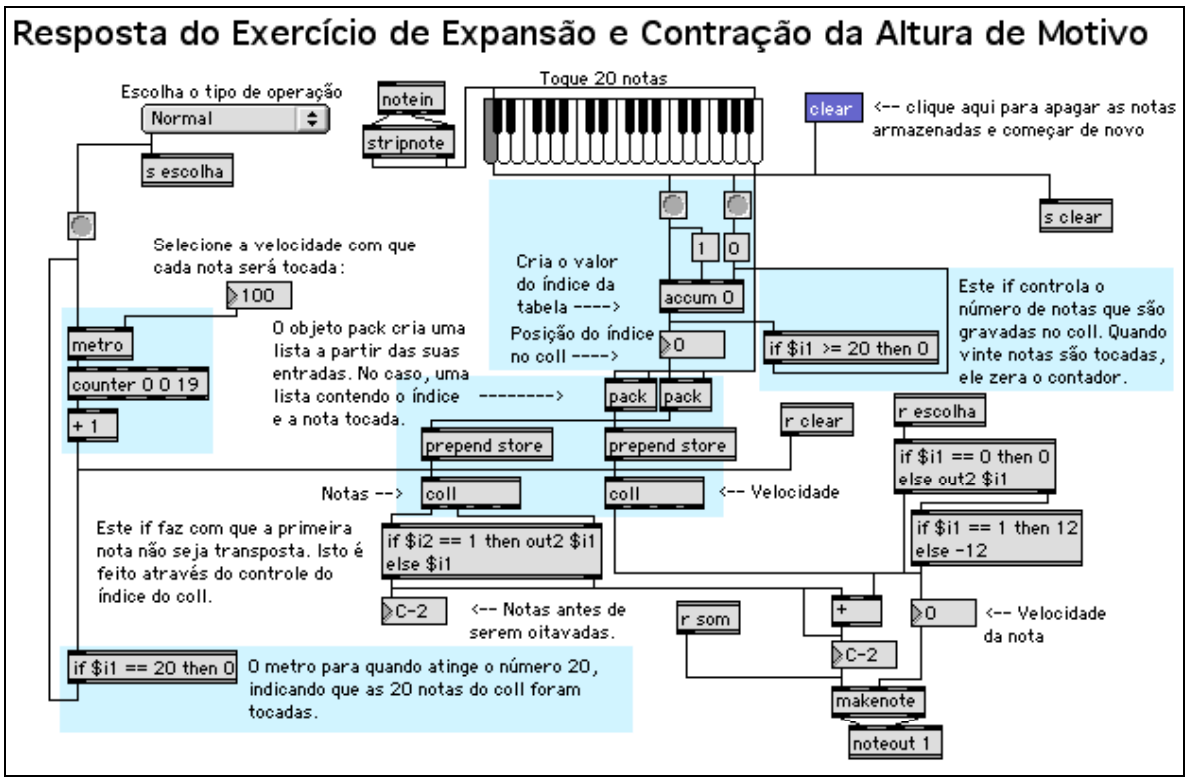


Figura 239 – Resposta do exercício do patch Expansão e Contração da Altura de um Motivo

8.1.5 AULA 5 - Música Serial com Computadores

O computador também pode auxiliar nas operações de material musical obtido a partir de séries. A música dodecafônica pode ser vista como a mais sistemática abordagem da composição com motivos, porque na música dodecafônica, todo o material de alturas é derivado de uma ordenação de 12 notas da escala cromática. O Exemplo 240 mostra as quatro formas básicas de variação de uma série dodecafônica (transposição, inversão, retrógrado e retrógrado-inverso). Estes procedimentos são facilmente transcritos em algoritmos para computador [WIN 87].

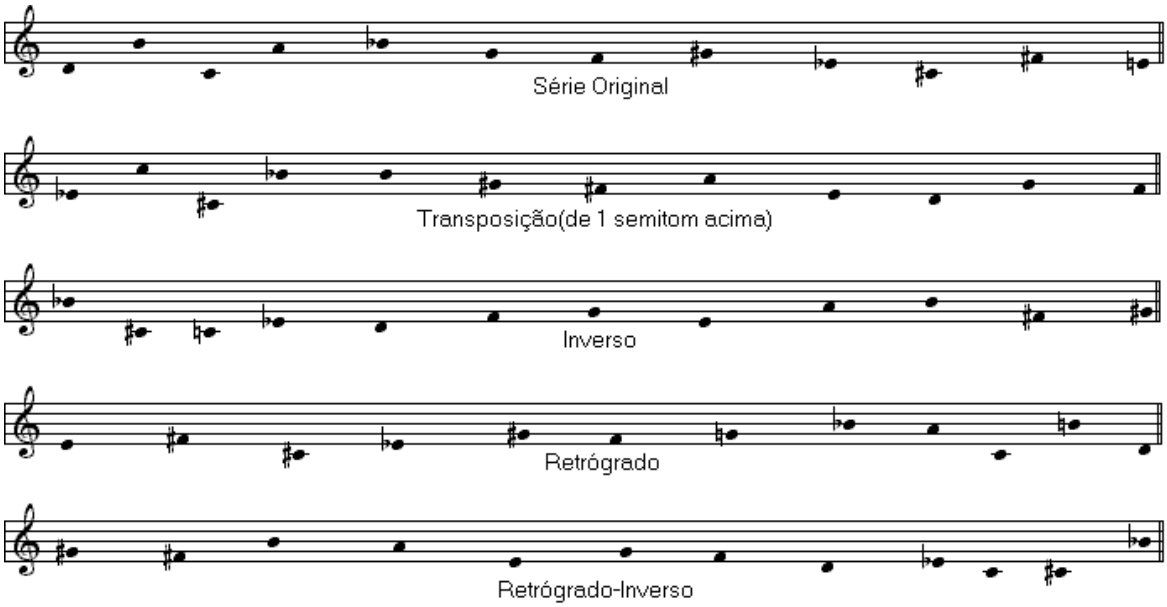


Figura 240 – Operações básicas sobre uma série dodecafônica

Desde a Segunda Guerra Mundial, compositores tem aplicado princípios de serialização para outras dimensões da música. Ordenamento serial de um número de parâmetros, incluindo ritmo, dinâmica, registros, timbre, e articulações tem sido implementado em alguns casos.

O método mais simples para operações seriais com ritmos, é criar uma série de durações análogas às séries de pitch classes. Uma vez estabelecida a série de durações, ela pode ser sujeita às mesmas permutações ocorridas na série de notas. Muitos compositores europeus incluindo Olivier Messiaen, Pierre Boulez, and Luigi Nono, empregaram serialização do ritmo em seus trabalhos usando métodos similares aos demonstrados aqui. Na prática atual, não há restrições contra o uso de diferentes séries para cada parâmetro musical separado. Portanto é possível utilizar uma série de 7 elementos no lugar de 12.

Observe no exemplo 241 como é possível programar um gerador de séries dodecafônicas. Tente utilizar o patch para gerar suas séries e para realizar operações de Retrógrado e Inverso sobre elas. Além disso, procure analisar como foram programadas as operações de Retrógrado e Inverso sobre a série. Em seguida realize o exercício indicado e programe a operação de Retrógrado-Inverso.

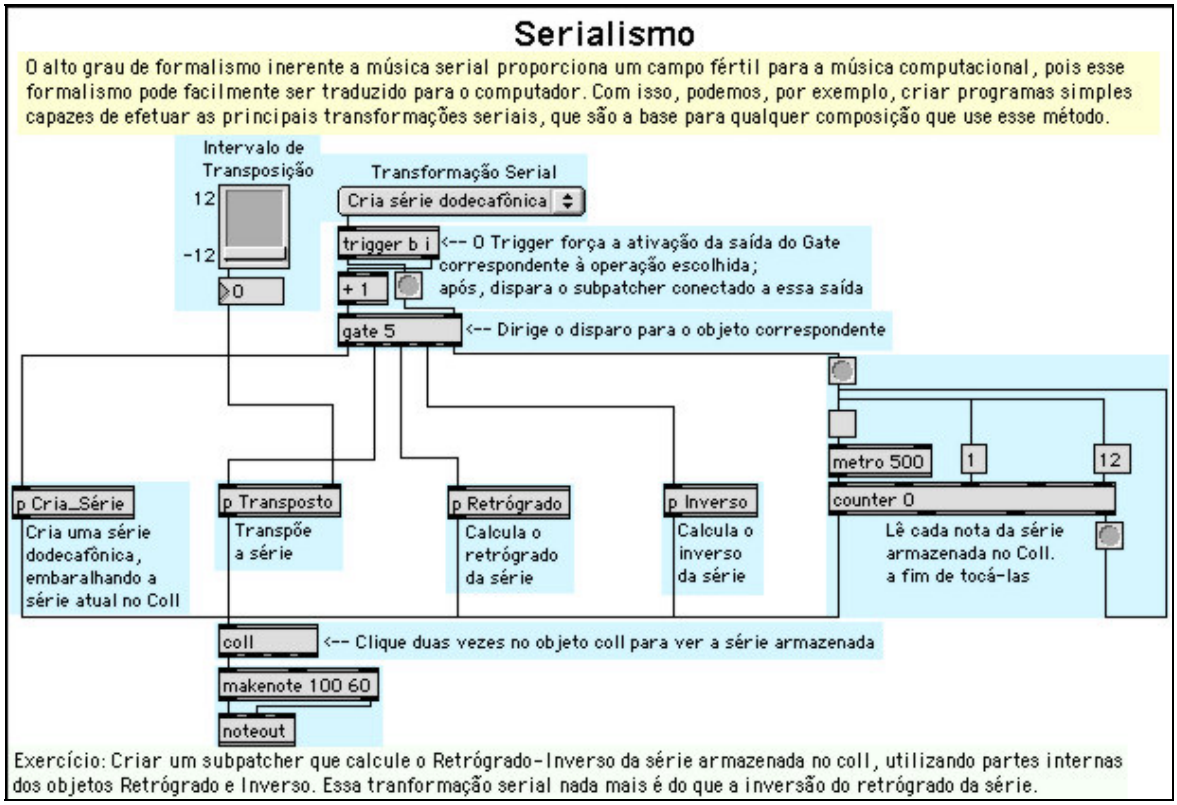


Figura 241 – Programa para serialismo de alturas

Para entender o programa de serialismo por completo é necessário estudar cada um dos *subpatches* do programa principal. O subpatch Cria_série é apresentado na figura 242. Observe que este algoritmo realiza o "embaralhamento" dos valores para produzir a série original que será transformada posteriormente pelas demais operações.

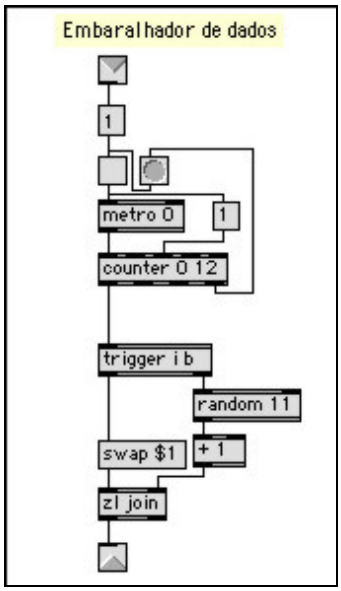


Figura 242 – Patch que cria a série dodecafônica original

O subpatch que realiza a operação de transposição da série pode ser visualizado na figura 242. Antes de mais nada, ele faz uma cópia do ***coll*** responsável por armazenar as notas da série para um ***coll*** auxiliar, utilizando as mensagens de operação sobre arquivos desse objeto. Primeiramente, o programa solicita ao ***coll*** da série a escrita dos dados nele armazenados para um arquivo temporário(chamado " aux") utilizando a mensagem "write aux". Depois, envia uma mensagem para o ***coll*** auxiliar para que ele leia esse arquivo da memória(fazendo assim uma cópia do original), com o envio da mensagem "read aux". Finalizada a criação do ***coll*** auxiliar, o programa passa a transpor cada uma das notas da série, percorrendo/contando cada posição do ***coll*** com o uso do *counter*, lendo o valor dessa posição, e somando o valor a ser transposto ao valor numérico(valor MIDI) da nota(cada vez que *metro* gera um "bang", o valor na outlet do *counter* é incrementado de 1; isso ocorre instantaneamente, pois o valor 0 no objeto indica que ele deve esperar 0 milisegundos entre cada "bang"). Por fim, cria e envia uma mensagem de armazenamento do valor transposto na mesma posição que ele ocupava anteriormente, através do objeto *zl join*.

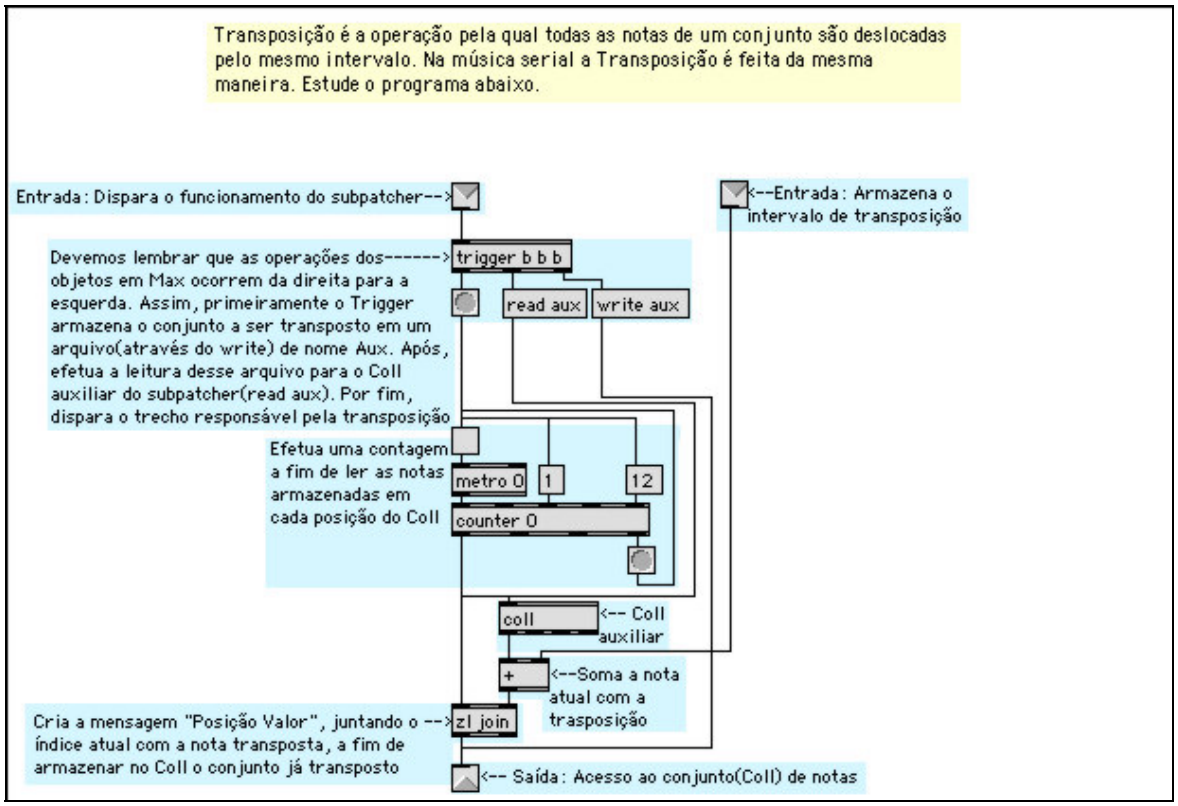


Figura 243 – Subpatch para transposição de uma série

O *subpatch* que realiza a operação de Retrógrado do motivo pode ser vista na figura 244. Ele copia o ***coll*** da série para um ***coll*** auxiliar para simplificar o programa, utilizando as mensagens de operação "write"(para escrever o ***coll*** da série em um arquivo temporário) e "read"(para ler esse arquivo da memória para o ***coll*** auxiliar). Depois, percorre/conta cada uma das 12 posições do ***coll*** auxiliar, com o uso do objeto *counter*. A cada posição percorrida, a expressão matemática *expr 13-$i1* calcula a posição retrógrada correspondente: a nota da posição 1 deve agora ficar na posição 12(13-1=12), a nota na 2 deve agora ficar na 11(13-2=11), e assim por diante. Finalizando a criação do ***coll*** retrocedido, o objeto *zl join* junta o valor da nova posição da nota com o valor numérico da nota, gerando uma mensagem de armazenamento daquela nota naquela posição para o ***coll***.

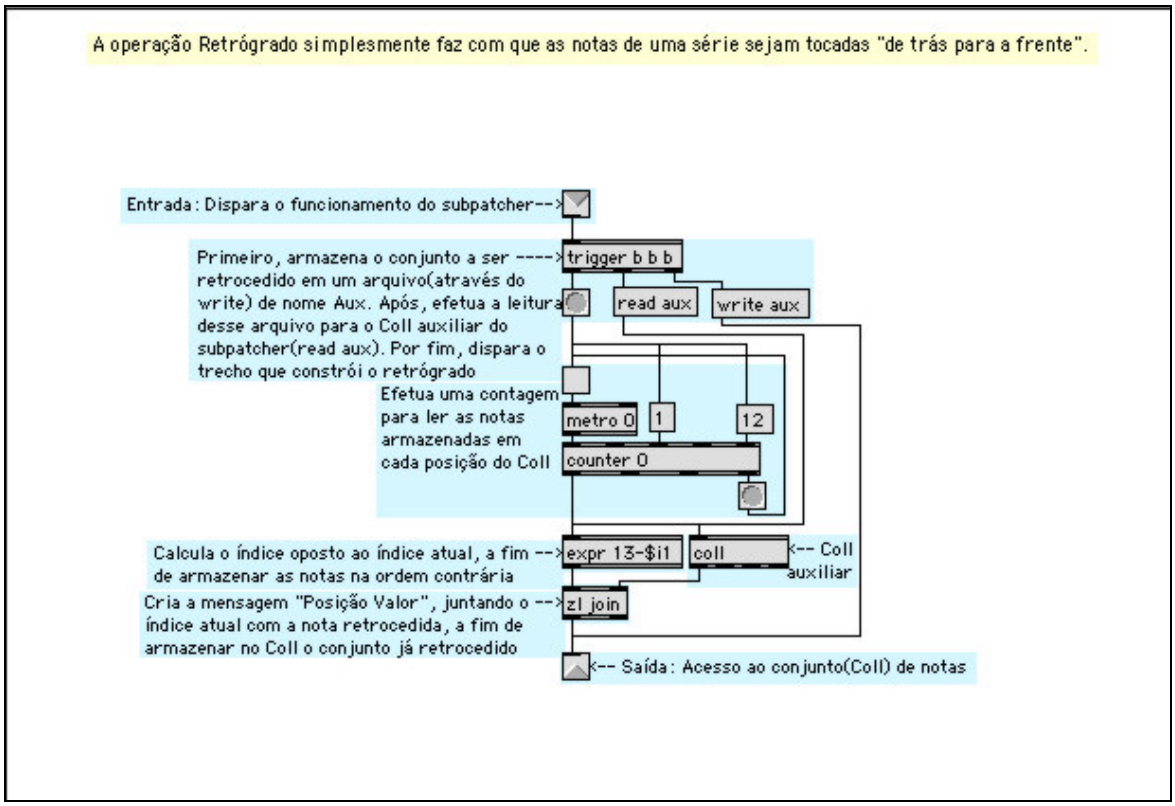


Figura 244 – Subpatch Retrógrado do Motivo

O *subpatch* que realiza a operação de Inversão do motivo pode ser visto na figura 245. Ele calcula o inverso de cada nota da série segundo a definição musical de inversão serial, que diz: a nota inversa corresponde ao intervalo complementar de tal nota dentro oitava(ou seja, até o Dó da próxima oitava). Como a escala cromática possui 12 graus em cada oitava, o complemento de cada nota(e, logo, o inverso) pode ser calculado pela simples subtração do intervalo da nota na oitava dos 12 graus da oitava. Entretanto, antes de efetuarmos essa operação, é necessário tomarmos o valor do intervalo da nota dentro da oitava, o que pode ser calculado tomando-se o resto de sua divisão inteira por 12, pois um intervalo será sempre do tipo 12_graus × o_número_da_oitava + o_intervalo_da_nota, e a divisão é a operação inversa da multiplicação. O Max não possui operação para o resto de uma divisão, então é necessária a criação do nosso próprio objeto, que nada mais é do que a transcrição da fórmula da matemática que usa a divisão real para calcular a valor do resto da divisão inteira: Resto=Dividendo-((Dividendo÷Divisor)×Divisor). Assim, o programa calcula o intervalo musical de cada nota dentro da oitava utilizando o *subpatcher* da figura 246, intitulado " resto". Então, inverte a nota com o uso de uma subtração de 12. Após essa inversão, o valor do intervalo pode estar dentro de outra oitava, pois a subtração nada mais fez do que transpor a nota. Para ajustarmos isso, é necessário tomarmos novamente o resto da divisão desse valor por 12, como já foi explicado. Por fim, o intervalo invertido é transposto novamente para a oitava original, somando-se a ele o valor numérico(MIDI) da oitava(esse valor pode ser calculado simplesmente, pois ele é apenas o valor do intervalo da nota subtraído do valor MIDI dela).

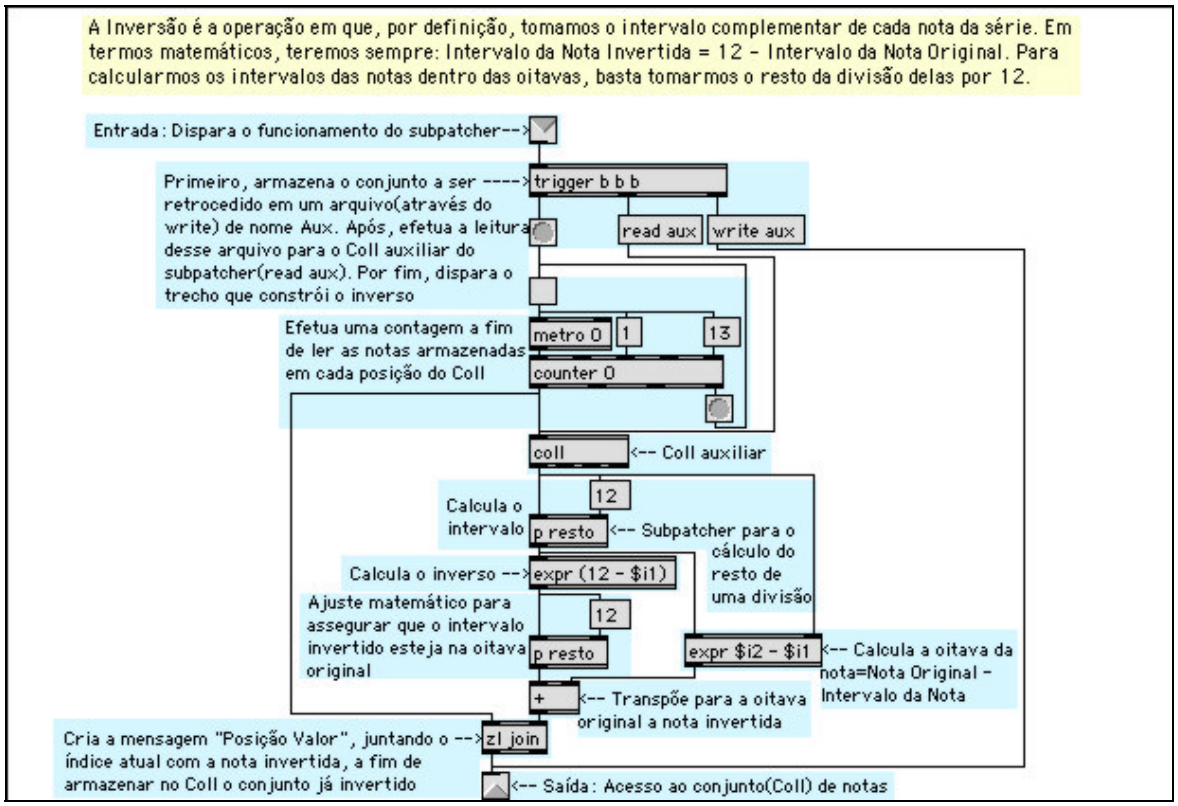


Figura 245– Inversão de um Motivo

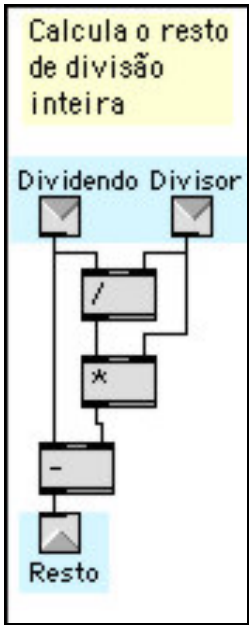


Figura 246 – Subpatch para o cálculo do resto da divisão inteira

Após estudar os *subpatchs* que implementam as operações com séries, resolva o exercício programando um patch que realize a operação de Retrógrado-Inverso.

A solução pode ser observada na figura 247. Esta só apresenta uma interface inicial para a realização da operação de Retrógrado-Inverso. Todo o algoritmo está programado no subpacth Retrógrado-Inverso.

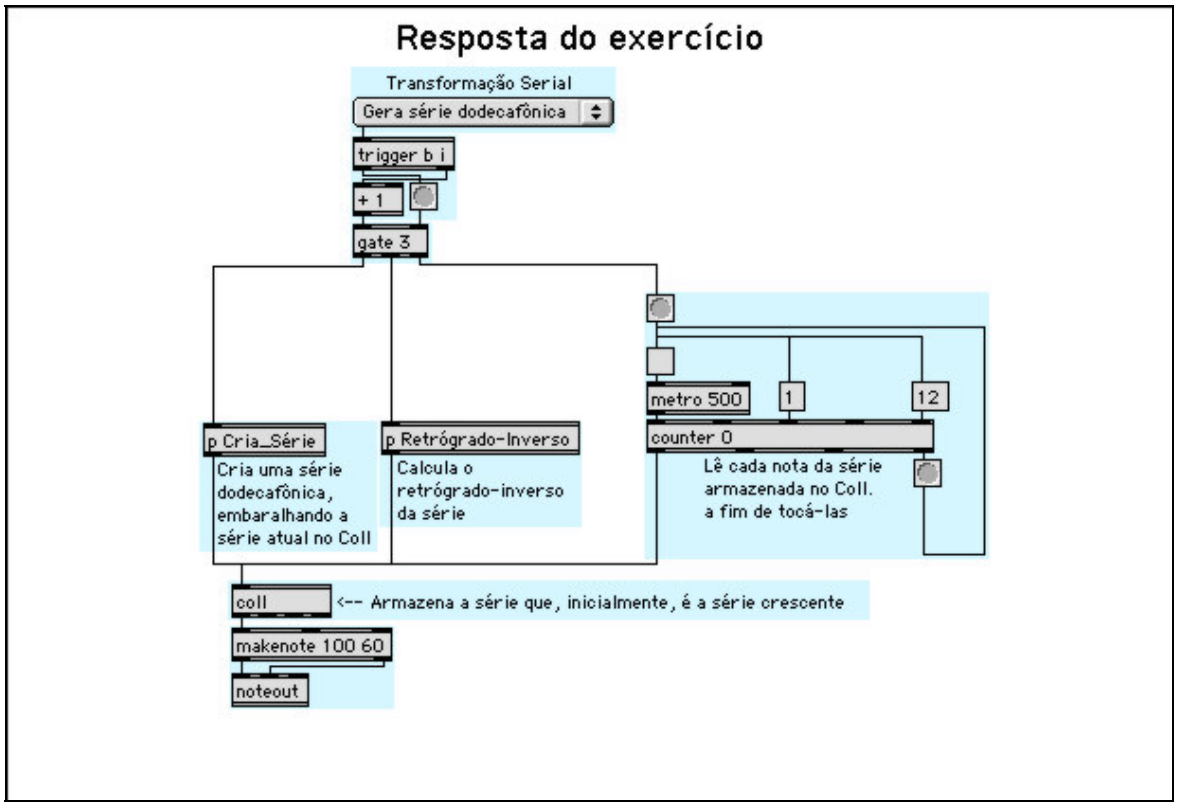


Figura 247 – Patch que realiza a operação de Retrógrado-Inverso

Com um duplo clique no *subpatch* Retrógrado-Inverso é possível visualizar como foi feita a programação (figura 248). Como a definição de Retrógrado-Inverso não pode ser entendida nem calculada sem as definições de Retrógrado e Inverso, simplesmente utilizou-se esses dois conceitos para efetuar essa operação. Assim, como o Retrógrado-Inverso é nada mais do que a inversão do retrógrado da série, o programa, a cada nota lida de uma posição contada pelo *counter*, calcula a posição retrógrada de tal nota(pela expressão *expr 13-$i1*, cujo funcionamento já foi explicado) e, em seguida, inverte o valor da nota, como já foi explicado. Ou seja, essa resposta é simplesmente composta pela justaposição do *subpatcher* "retrógrado", seguido pelo *subpatcher* "inverso".

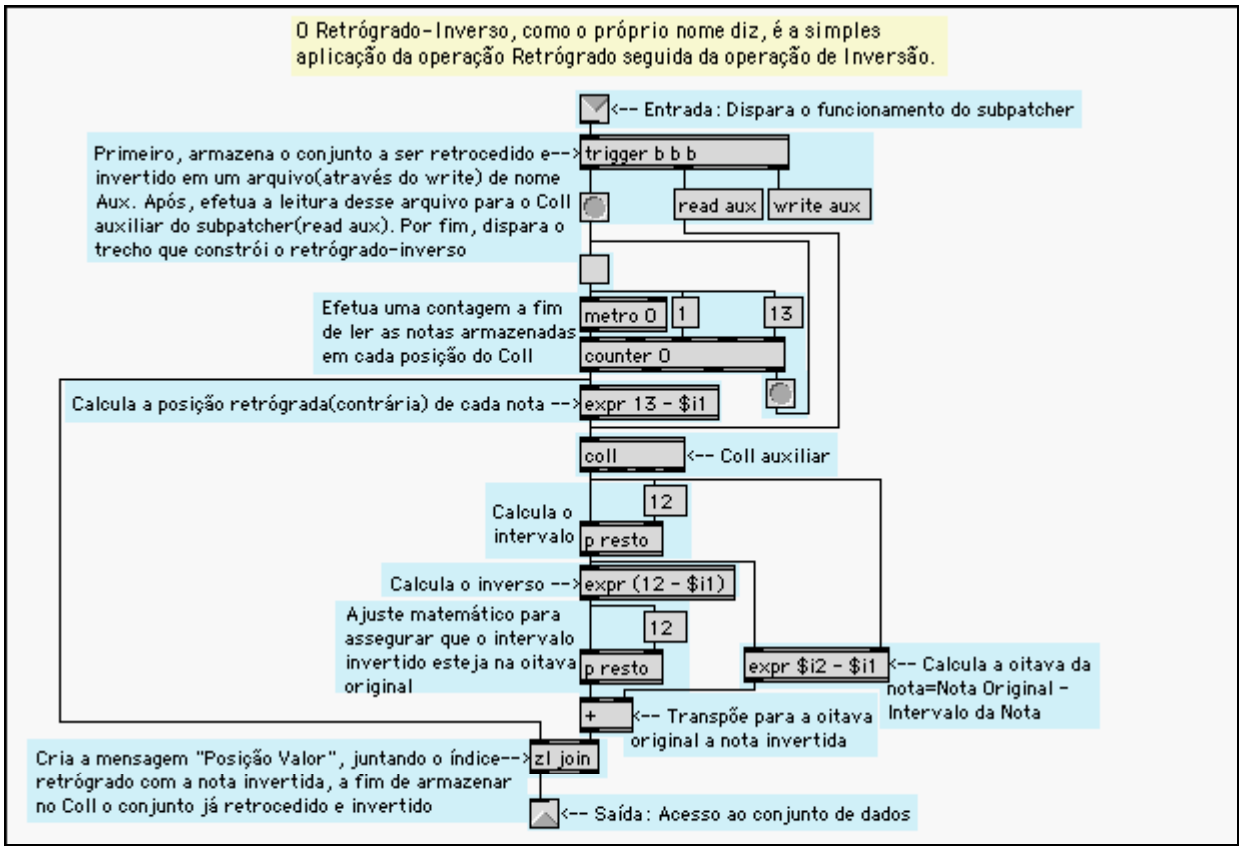


Figura 248 – Subpatch para cálculo do Retrógrado-Inverso

Algoritmos para operações com música dodecafônica, escritos em outras linguagens, podem ser obtidos em [WIN 87] e [MIR 2001].

8.1.6 AULA 6 - Composição Aleatória por Computador

Muitos compositores do passado utilizaram processos randômicos para gerar suas criações. O emprego da aleatoriedade por Cage e seus discípulos, resultou na utilização de processos randômicos em vários estágios de uma composição. Pierre Boulez, um defensor da idéia do compositor possuir total controle sobre os parâmetros de uma composição musical, compôs sua Terceira Sonata Para Piano na qual é possível tocar as seções escolhendo-as dentro de um conjunto pré-determinado de opções. Cada futura escolha de seções deve ser realizada dependendo das escolhas previamente feitas [ROA 96].

Existem duas classes gerais de processos randômicos que tem sido usados para gerar material para composição musical:

1. **Processos randômicos com observações independentes**.

Quando observações independentes são feitas num processo randômico e o resultado não depende de nenhum resultado prévio. Os valores dos resultados são distribuídos em um conjunto ou uma faixa com um padrão característico de um processo particular. Por exemplo, lançar uma moeda representa um processo onde o conjunto de possibilidades resultante tem dois resultados onde o padrão de distribuição fornece uma chance igual para ambos[DOD 97].

2. **Processos nos quais os resultados prévios influenciam no resultado corrente de alguma forma**:

Uma maneira de expressar a influência dos resultados passados de um processo randômico nos resultados atuais, é através do uso explícito de probabilidades condicionais. Neste caso, a probabilidade da ocorrência de um evento particular é baseada no resultado de um ou mais eventos prévios [DOD 97].

Quando um processo randômico é usado em composição, raramente o resultado obtido é definitivo. Na maioria das vezes existe a necessidade de aprimorar o material musical para, posteriormente, utilizá-lo na composição.

8.1.6.1 Probabilidade e Processos Randômicos

A chance de que um evento particular possa ocorrer pode ser expressa em probabilidade. Ao jogar dados com seis lados teremos um conjunto com seis elementos {1,2,3,4,5,6}. Se o dado estiver bem balanceado teremos a possibilidades iguais para cada um dos seis números. A chance de dar o número seis é de 1/6 podendo ser escrito como:

$$P(X=6) = 1/6$$

X é conhecida como variável randômica.

As propriedades da probabilidade são:

1. O valor numérico da probabilidade está sempre entre 0 e 1.
2. A probabilidade que um evento não ocorra é obtida com o valor de 1 subtraído do valor da possibilidade que o evento ocorra. Ex: $P(X <> 6) = 1 – P(X=6) = 5/6$, pois $P(X=6) = 1/6$.
3. A probabilidade de que qualquer um dos eventos ocorra é igual a soma da probabilidade dos eventos individuais.
4. A soma das probabilidades de todos os eventos possíveis de um processo deve ser igual a 1.

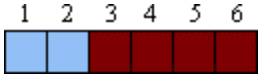
As probabilidades de um fenômeno randômico contínuo são normalmente expressas por uma função de densidade probabilística f(x).

$F(x) = 1$ para $0 < x < 1$, denotando que todos os resultados deverão estar entre 0 e 1.

A Distribuição Uniforme é a mais comumente encontrada onde existe uma chance igual de obter qualquer um dos resultados como no caso do jogo de dados.

8.1.6.2 Tabelas de Probabilidades

Podemos também descrever a probabilidade de um fenômeno utilizando uma tabela. Normalmente, essa tabela possui 100 posições (a fim de criarmos um paralelo com porcentagem), de maneira que cada valor possível de ser sorteado deve ser guardado na tabela o número de vezes igual à sua porcentagem. Com isso, se escolhermos, ao acaso, qualquer uma das 100 posições da tabela para sortearmos o valor que lá está armazenado, teremos mais ou menos chance de tomarmos esse valor se ele ocupar mais ou menos posições na tabela. Observe a figura abaixo:



Imagine agora que vamos sortear uma figura de tempo entre semínima e colcheia, em uma posição da tabela, tomando aquela posição correspondente ao índice de valor gerado por um dado. A cor azul corresponde à semínima e a cor vermelha à colcheia. Ao lançarmos o dado, qualquer um dos valores 1 até 6 pode ser sorteado, indistintamente. Entretanto, se verificarmos que 4 posições da tabela estão ocupadas

pela colcheia (vermelho) e apenas 2 estão ocupadas pela semínima (azul), podemos perceber facilmente que a colcheia será sorteada mais vezes do que a semínima, mesmo embora qualquer uma das posições da tabela possa ser sorteada com igual chance. Valendo-se disso, podemos descrever qualquer fenômeno probabilístico utilizando uma tabela e um gerador uniforme de números (isto é, uma distribuição uniforme).

A figura 249 abaixo apresenta um programa que permite definir uma tabela e, posteriormente, sortear amostras de notas para a geração de uma composição aleatória. É importante ressaltar que o papel do gerador uniforme de números mencionado acima é desempenhado pelo objeto ***random***, enquanto que o ***coll*** é a tabela. Acompanhe o funcionamento completo do programa através das explicações da figura e resolva o exercício proposto.

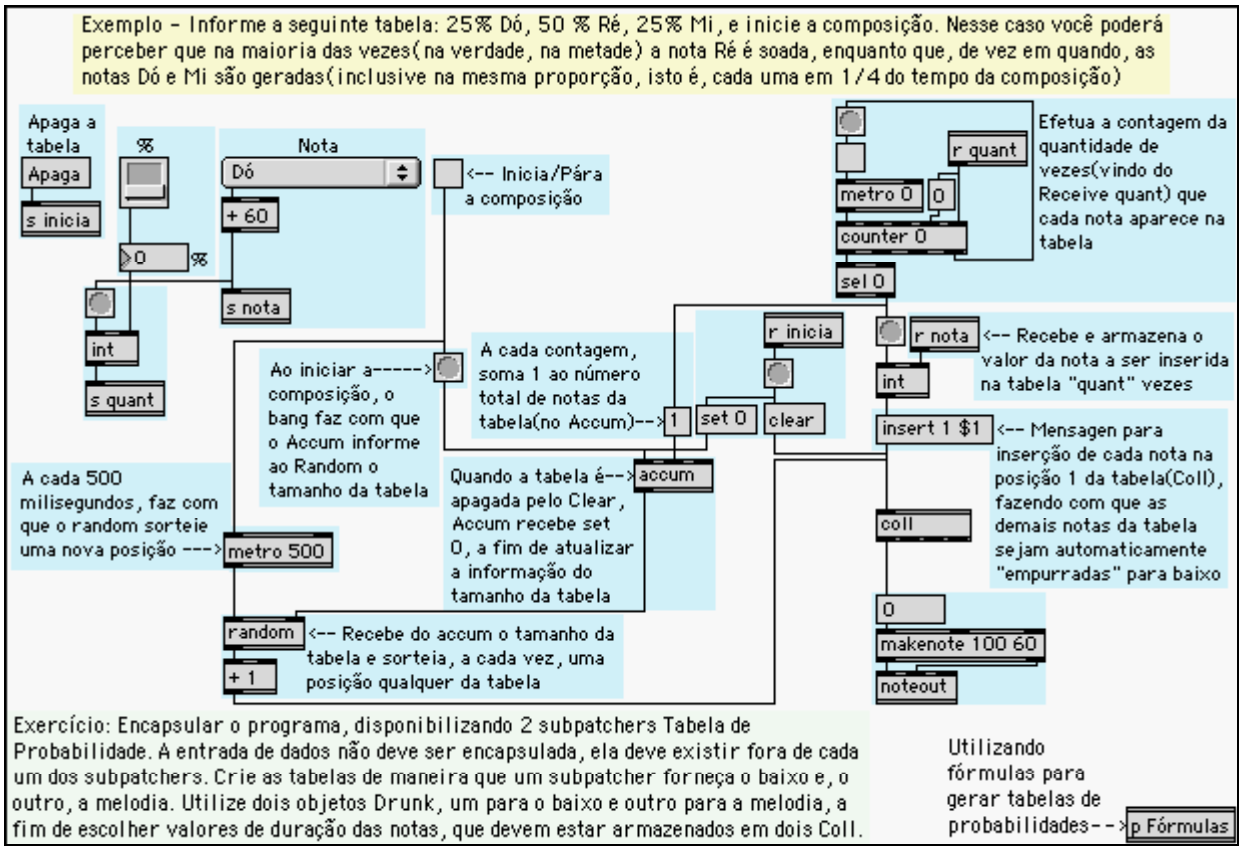


Figura 249 – Gerador de tabelas de probabilidades

Uma solução para o exercício sugerido pode ser vista na figura 250 abaixo.

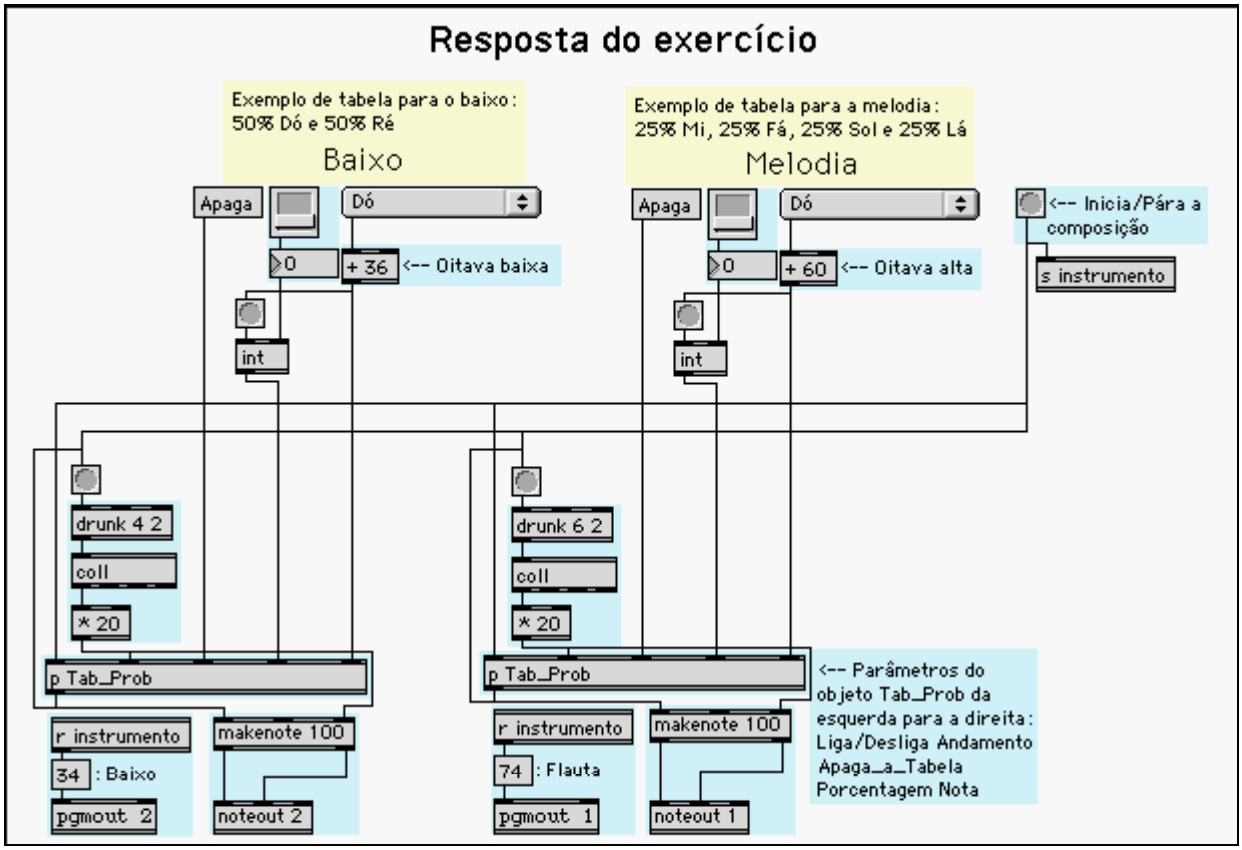


Figura 250 – Resposta do exercício sobre tabelas de probabilidades

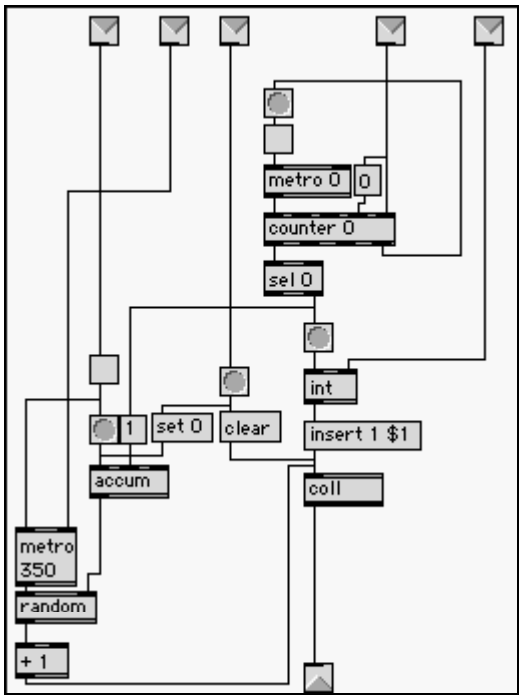


Figura 251 – Subpatcher para criação e utilização de tabelas de probabilidades

8.1.7 AULA 7- Fórmulas Probabilísticas

A distribuição Linear apresenta um comportamento diferente da distribuição uniforme vista na aula 6. Na distribuição Linear a coleção de probabilidades de todos os possíveis resultados de um processo são expressas por um gráfico, tal como mostrado a figura 252.

$F(x)= 2(1-x) \quad 0 \leq x \leq 1$
0 para qualquer outro valor

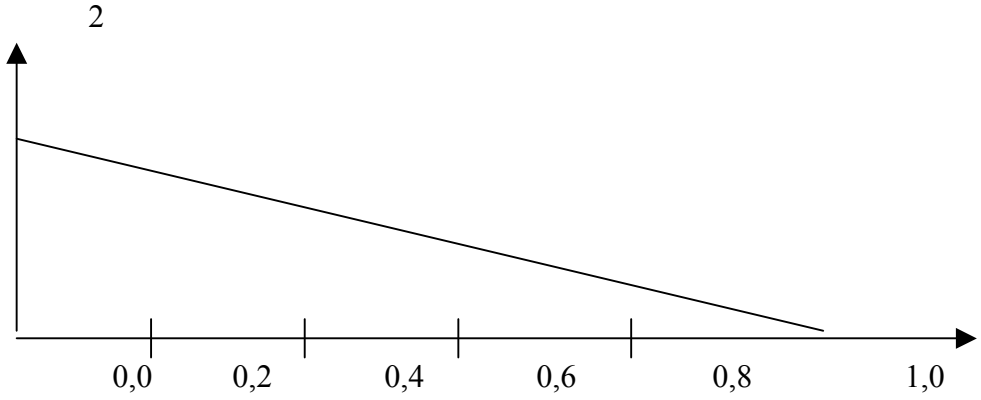


Figura 252 – Probabilidade da distribuição variável randômica linear

Segundo [DOD 97], além da distribuição uniforme e linear, várias outras funções probabilísticas podem ser utilizadas na composição musical tais como:

Distribuição Triangular
Distribuição Exponencial
Distribuição Exponencial Bilateral
Distribuição de Gauss
Distribuição de Cauchy
Distribuição Beta
Distribuição Weibull
Distribuição de Poison

Em programação visual podemos disponibilizar várias fórmulas matemáticas probabilísticas dessas distribuições sob a forma de objetos. Exemplos da programação dessas fórmulas para geração automática de material musical serão abordados a seguir.

Na figura 253 é apresentado um menu com várias opções de geração probabilística de notas. Cada ítem do menu é um *subpatch* que apresenta um tipo de fórmula probabilística programada em Max. Estude cuidadosamente cada ítem do menu e execute os programas observando as peculiaridades de cada resultado gráfico e sonoro obtido.

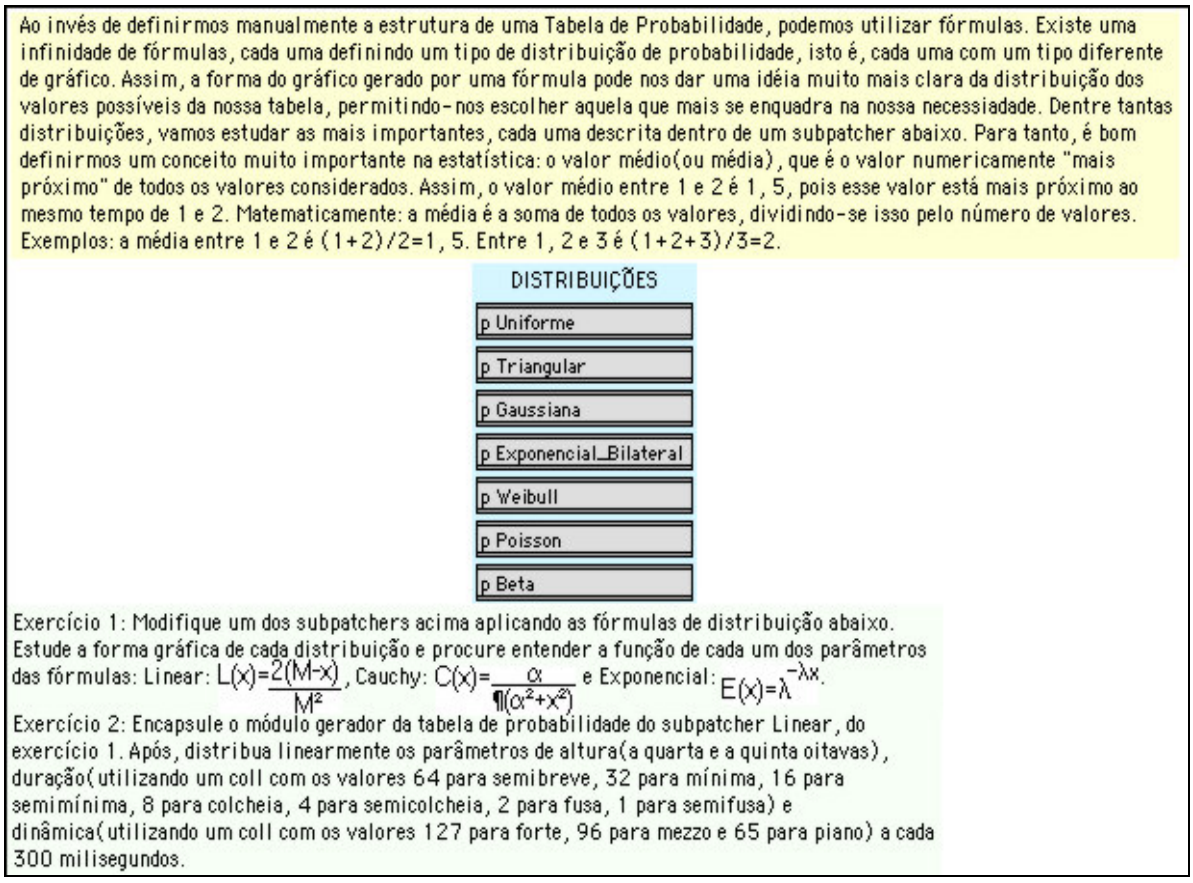


Figura 253 – Tela principal da geração de material musical através da programação de funções estocásticas

Nestes exemplos não iremos nos ater às informações sobre a construção do gráfico(figura 256). A rotina de plotagem foi feita apenas para mostrar o comportamento gráfico da função probabilística que está sendo utilizada. Estes exemplos, sobre geração de alturas através de probabilidades, contém três saídas de dados. Além do gráfico, existe a emissão sonora das alturas e a representação individual delas na pauta musical. Nos exemplos, a pauta sempre aparecerá no canto superior direito da tela e mostrará as notas que estão sendo produzidas pela função probabilística.

No exemplo da figura 254, a distribuição uniforme é obtida pela utilização do objeto ***random*** que, por sua vez, é capaz de produzir valores com a mesma chance de ocorrência. Utilize o programa e observe os resultados gráficos e sonoros resultantes.

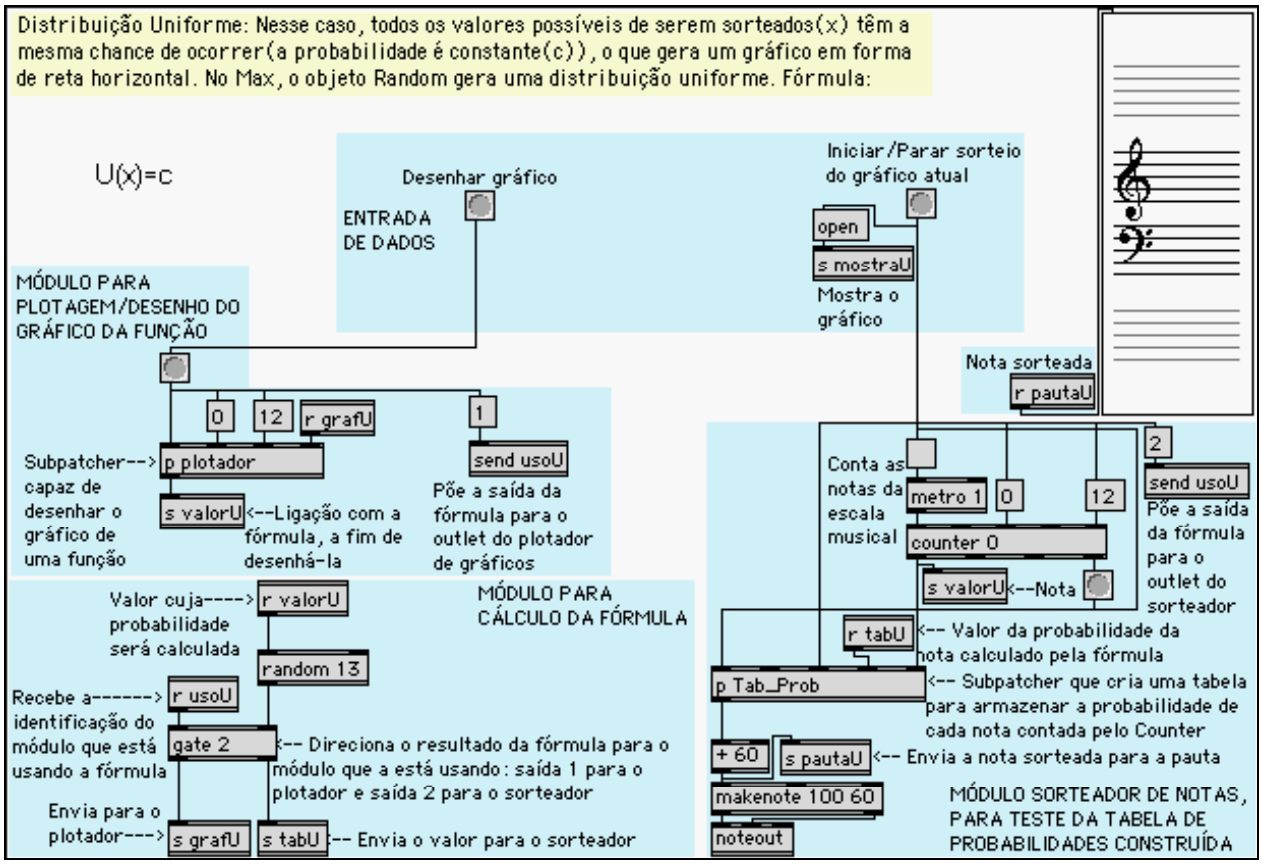


Figura 254 – Programação da Distribuição Uniforme

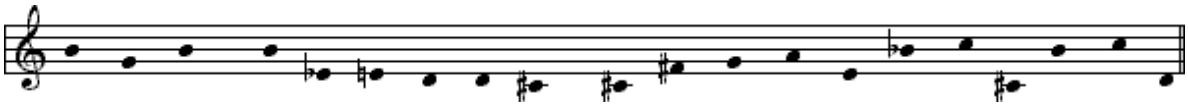


Figura 255 – Resultado obtido a partir da execução do programa de distribuição uniforme

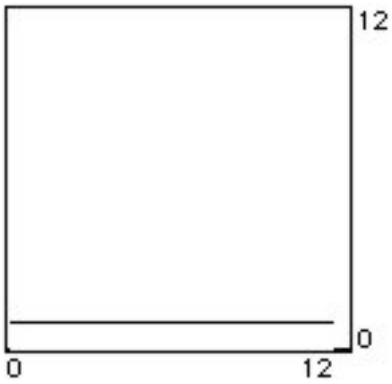


Figura 256 - Gráfico da distribuição uniforme

Na figura 257 é apresentado um exemplo sobre Distribuição Triangular. Neste exemplo, as escolhas dos parâmetros iniciais devem ser realizadas pelo músico através da movimentação de três sliders. Após escolher os três valores, clique no botão Iniciar.

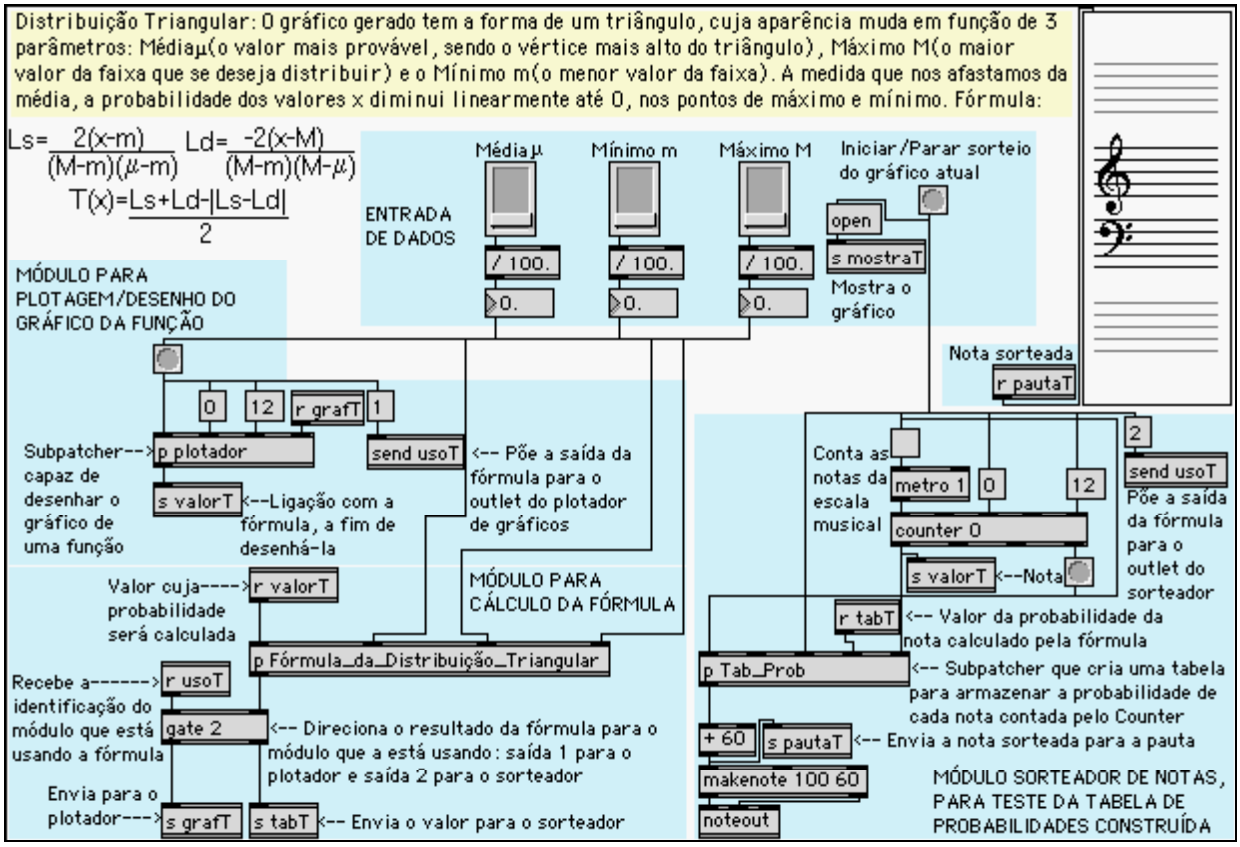


Figura 257 – Programação da Distribuição Triangular

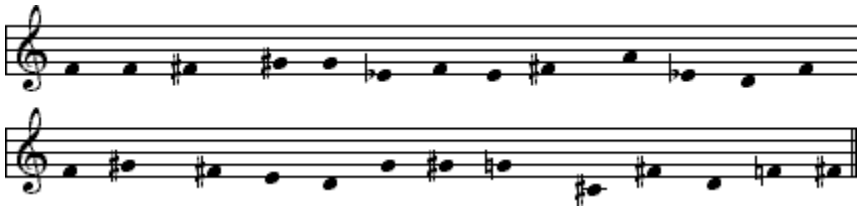


Figura 258 – Resultado obtido a partir da execução do programa de distribuição triangular, com Mínimo 0, Máximo 12 e Média 6

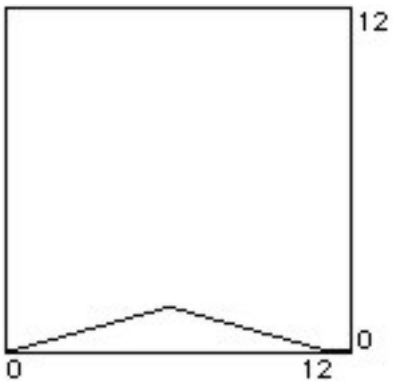


Figura 259 – Gráfico da distribuição triangular(Mínimo 0, Máximo 12 e Média 6)

A fórmula matemática da Distribuição Triangular foi convertida numa expressão computacional contida no *subpatch* **Fórmula_da_Distribuição_Triangular**. Este *subpatch* é apresentado na figura 260 onde podem ser observadas quatro inlets para o recebimento de valores provindos do patch. Um valores é enviado do *subpatch* plotador e os outros três são enviados dos sliders.

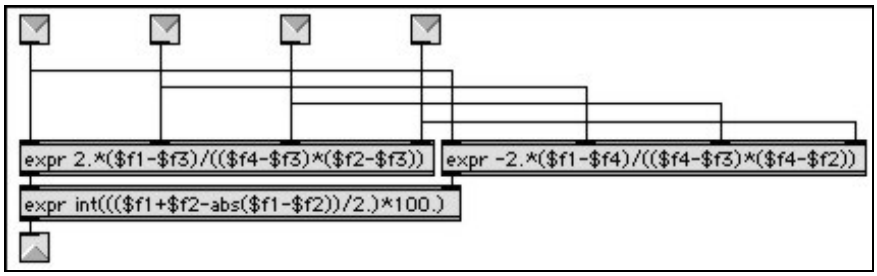


Figura 260 – Subpatch com a expressão da Distribuição Triangular

As fórmulas escritas no objeto *expr* devem seguir o padrão utilizado em programação. A seguir explicamos como as três fórmulas utilizadas foram convertidas para esta nova notação.

Fórmula da Linha de Subida(Ls):

$$Ls = \frac{2(x-m)}{(M-m)(\mu-m)} \longrightarrow 2.*(\$f1-\$f3)/((\$f4-\$f3)*(\$f2-\$f3))$$

Constantes numéricas da fórmula: são os números utilizados na fórmula, que podem ser inteiros ou reais. Para os números reais utilizamos ponto "." ao invés de vírgula "," para separarmos a sua parte fracionária da parte inteira(nos casos onde a parte fracionária vale zero é suficiente colocarmos apenas o ".").

Ex.:

Número na fórmula matemática	Número na notação computacional
2,0	2.
-1,3	-1.3
4	4
0,5	0.5

Variáveis da fórmula: são representadas por $ seguido de um identificador do tipo, sendo **i** para inteiro ou **f** para real, juntamente com o índice correspondente a entrada do objeto, da esquerda para a direita.

Ex.:

Variável da fórmula matemática	Variável na notação computacional
x	$f1
m	$f3
M	$f4
μ	$f2

Operadores da fórmula: toda fórmula matemática em notação computacional deve ser escrita em uma linha apenas. Assim, levando-se isso em consideração, fica fácil lembrarmos intuitivamente a correspondência entre elas, como por exemplo, no caso da divisão, que é representada pela barra "/". Veja a tabela abaixo.

Operador da fórmula matemática	Operador na notação computacional
Multiplicação	*
Divisão	/

Parênteses: alteram a ordem de precedência de cálculo dos operadores da fórmula.

Ex.: ($f1-$f3)/(($f4-$f3)*($f2-$f3)), onde os parênteses evitam que a divisão seja calculada primeiramente entre $f3 e $f4.

Com isso, podemos escrever as fórmulas da distribuição triangular da seguinte forma:

Fórmula da Linha de Descida(Ld):

$$Ld=\frac{-2(x-M)}{(M-m)(M-\mu)} \longrightarrow \text{-2.*($f1-$f4)/(($f4-$f3)*($f4-$f2))}$$

Fórmula do Triângulo(T):

$$T(x)=\frac{Ls+Ld-|Ls-Ld|}{2} \longrightarrow \text{int(((\$f1+\$f2)-abs(\$f1-\$f2))/2.)*100.)}$$

Funções matemáticas da fórmula: são os cálculos "mais complicados " que utilizamos, tais como o módulo/valor absoluto de um número. Em computação, uma função é representada por um nome seguido pelos seus parâmetros separados por "\," e entre parênteses.

Ex.:

Função da fórmula matemática	Função na notação computacional(*expr*)
Módulo: |x|	abs(x)
Truncagem	int(x)
Exponenciação: x^y	pow(x\,y)
Raiz quadrada: $\sqrt{x}$	sqrt(x)
Exponenciação(em **e**): e^x	exp(x)

A saída do *subpatch* **Fórmula_da_Distribuição_Triangular** é enviada para uma tabela que armazena a probabilidade de cada nota e, por sua vez, envia o valor para ser tocado.

No exemplo da figura 261, assim como os demais programas probabilísticos que o sucedem, são empregadas as mesmas estruturas de programação já utilizadas em exemplos anteriores (figuras 257 e 258). Observe que as mudanças são realizadas no número de entradas e na programação da fórmula no objeto *expr* para o cálculo da probabilidade. Portanto, a maior tarefa está em saber programar a expressão estocástica. Para exemplificar essa tarefa, tomamos a fórmula da distribuição Gaussiana (Normal) transformando-a numa expressão computacional passível de ser utilizada no objeto *expr*.

Devemos abordar o problema de adaptação da fórmula por partes. Inicialmente adaptamos o numerador. Este possui operações de exponenciação que também devem ser resolvidas por partes.

$e^{\frac{-(x-\mu)^2}{2\sigma^2}}$ ⟶ exp(-(pow(($f1-$f2)\,2.)/(2.*pow($f3\,2.))))

Após transformar o numerador em expressão faça o mesmo para o denominador. Observe que no denominador temos uma operação de radiciação.

$\sqrt{2\P\sigma}$ ⟶ (sqrt(2.*3.1415)*$f3)

Após transformar o numerador e o denominador em expressões basta dividir o numerador pelo denominador.

$G(x)=\frac{e^{\frac{-(x-\mu)^2}{2\sigma^2}}}{\sqrt{2\P\sigma}}$ ⟶ int((exp(-(pow(($f1-$f2)\,2.)/(2.*pow($f3\,2.))))/
(sqrt(2.*3.1415)*$f3))*100.)

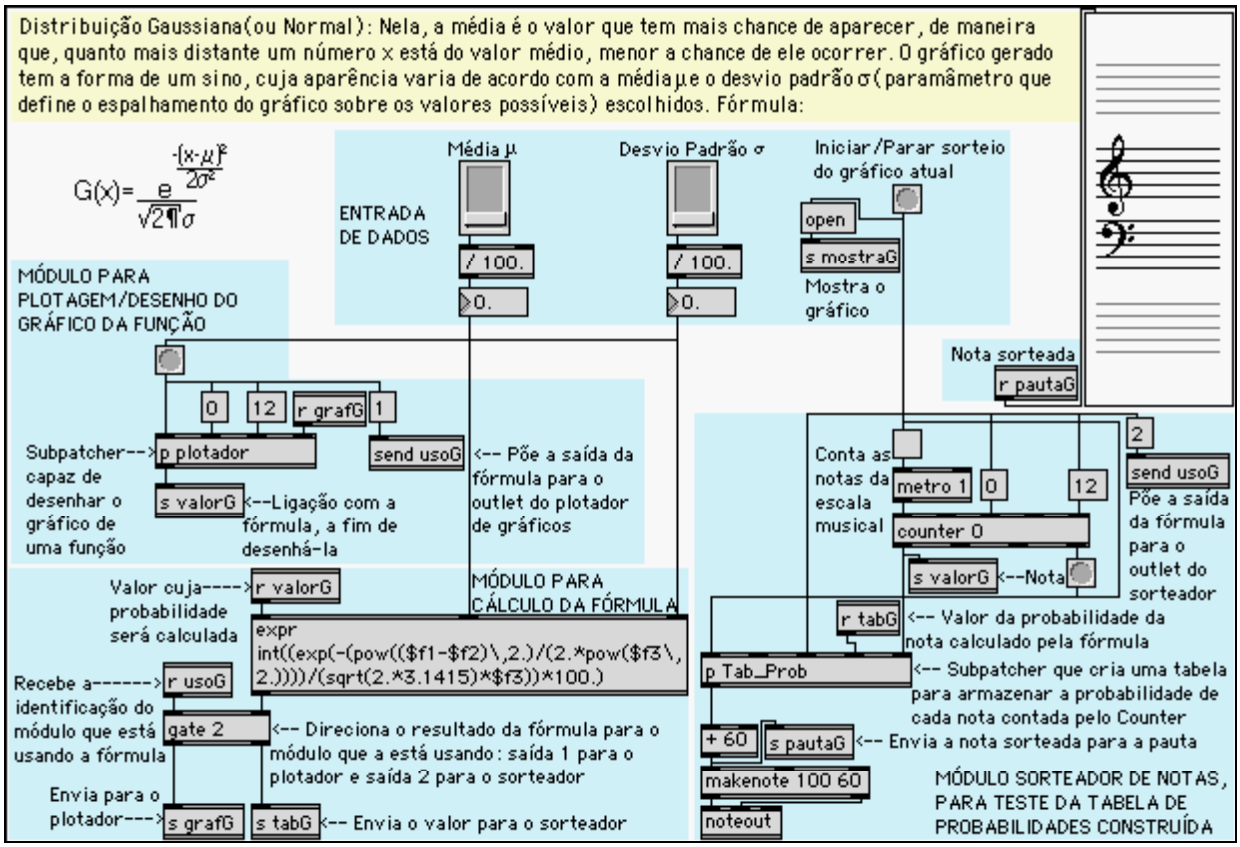


Figura 261 – Programação da Distribuição Normal

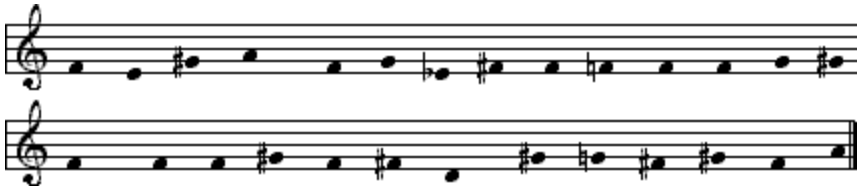


Figura 262 – Resultado obtido após a execução do programa de Distribuição Normal, com Média 6 e Desvio padrão 2

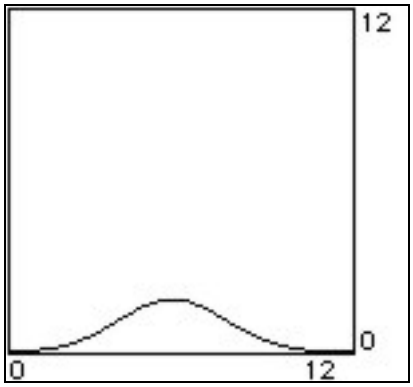


Figura 263 - Gráfico da distribuição Gaussiana(Média 6 e Desvio padrão 2)

Novamente, na programação da Distribuição Exponencial Bilateral da figura 264, o que muda, em relação aos exemplos anteriores, é a entrada de dados e o conteúdo do objeto *expr* .

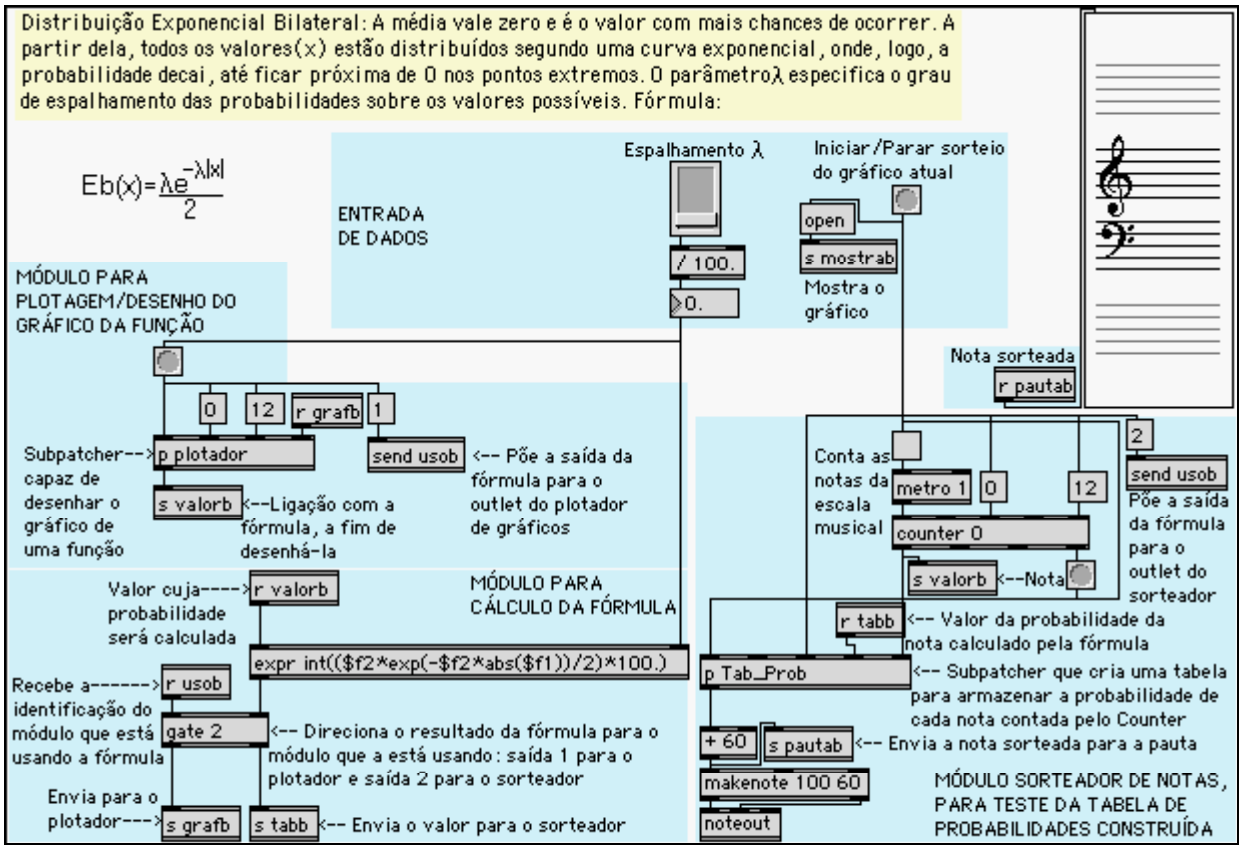


Figura 264 – Programação da Distribuição Exponencial Bilateral

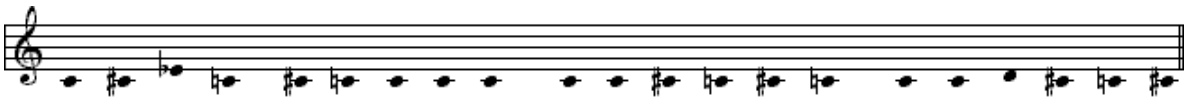


Figura 265 – Resultado obtido após a execução do programa de Distribuição Exponencial Bilateral, com Espalhamento de 0,8

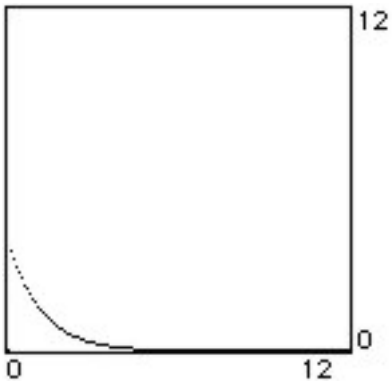


Figura 266 - Gráfico da distribuição exponencial bilateral(Espalhamento de 0,8)

Na figura 267 é apresentado o exemplo de como pode ser programado a Distribuição de Weibull.

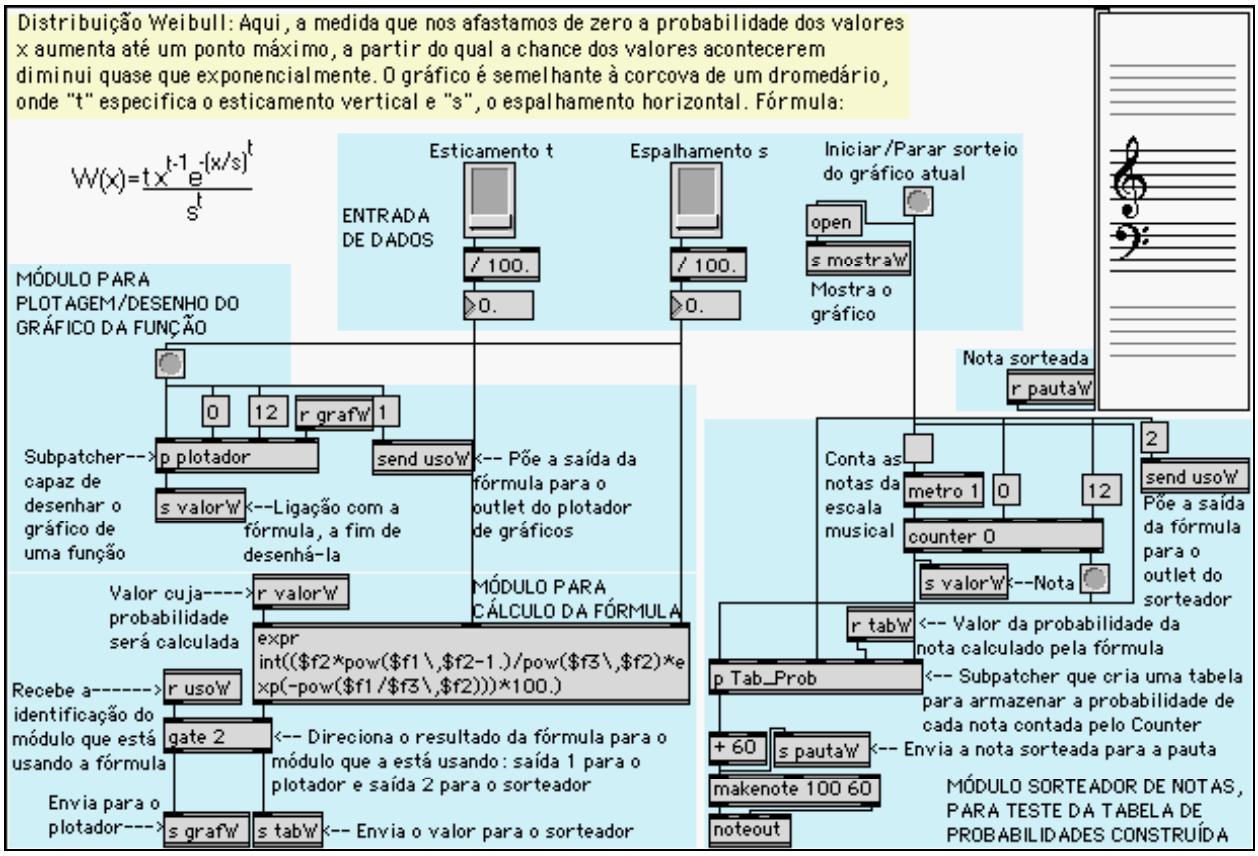


Figura 267 – Programação da Distribuição de Weibull

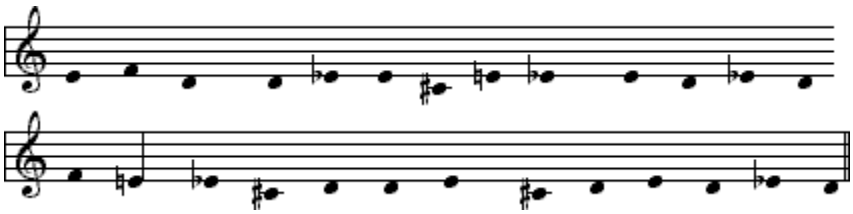


Figura 268 – Resultado obtido após a execução do programa de Distribuição de Weibull, com Esticamento 2 e Espalhamento de 3,5

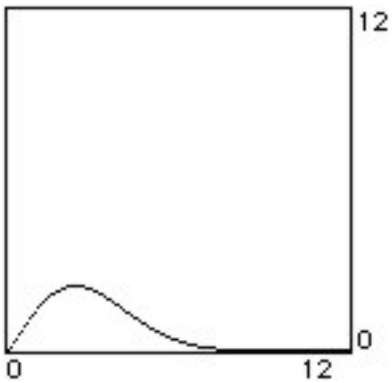


Figura 269 - Gráfico da distribuição Weibull(Esticamento 2 e Espalhamento de 3,5)

Na figura 270 é apresentado o exemplo de como pode ser programado a Distribuição de Poisson.

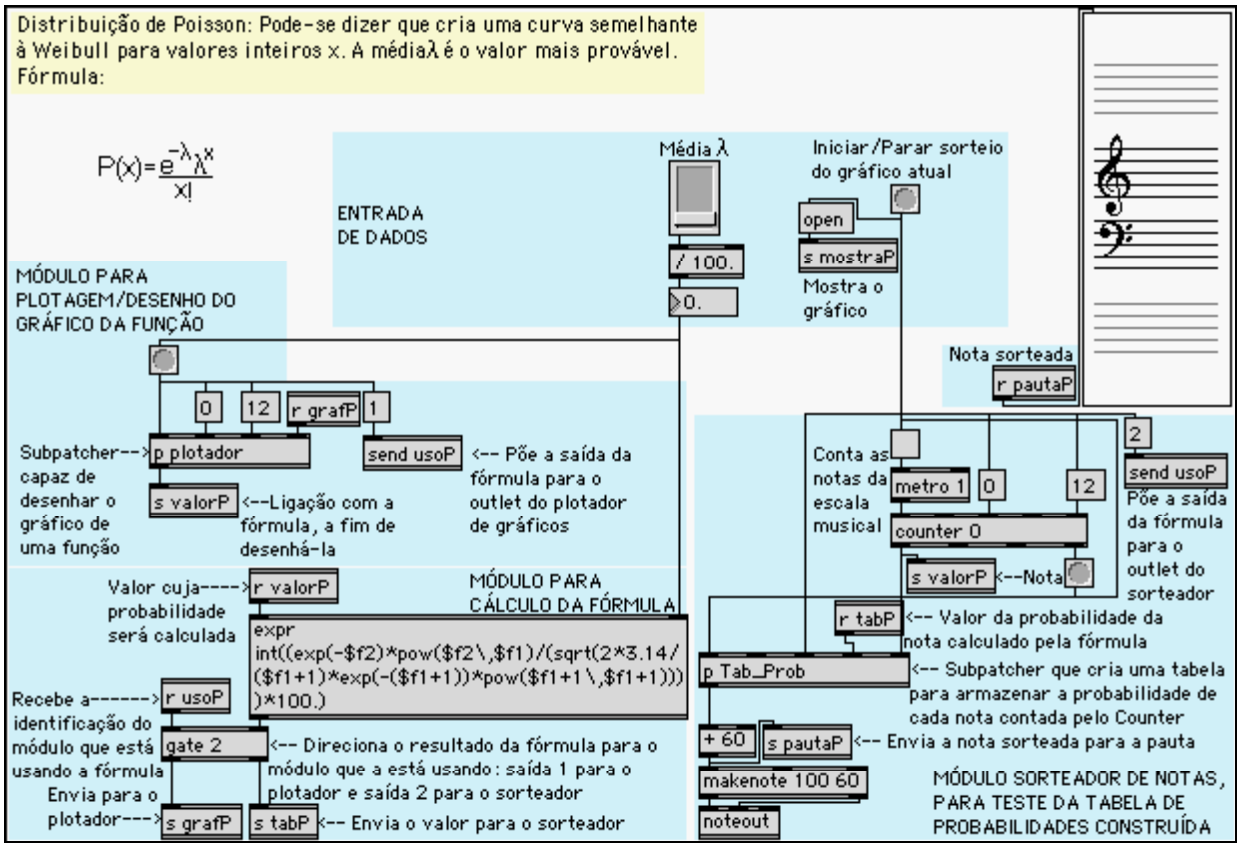


Figura 270 – Programação da Distribuição de Poisson

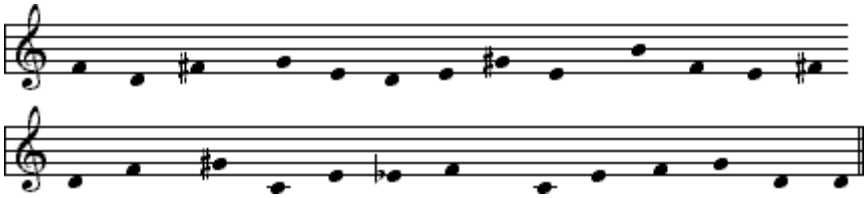


Figura 271 – Resultado obtido após a execução do programa de Distribuição de Poisson, com Média 1,9

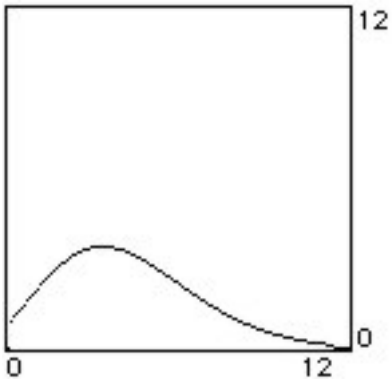


Figura 272 - Gráfico da distribuição de Poisson(Média 1,9)

Na figura 273 é apresentado o exemplo de como pode ser programado a Distribuição Beta. Devido à complexidade/impossibilidade em representar a fórmula em termos computacionais através do objeto expr foi preciso criar um *subpatch* chamado **Fórmula_da_Distribuição_Beta**.

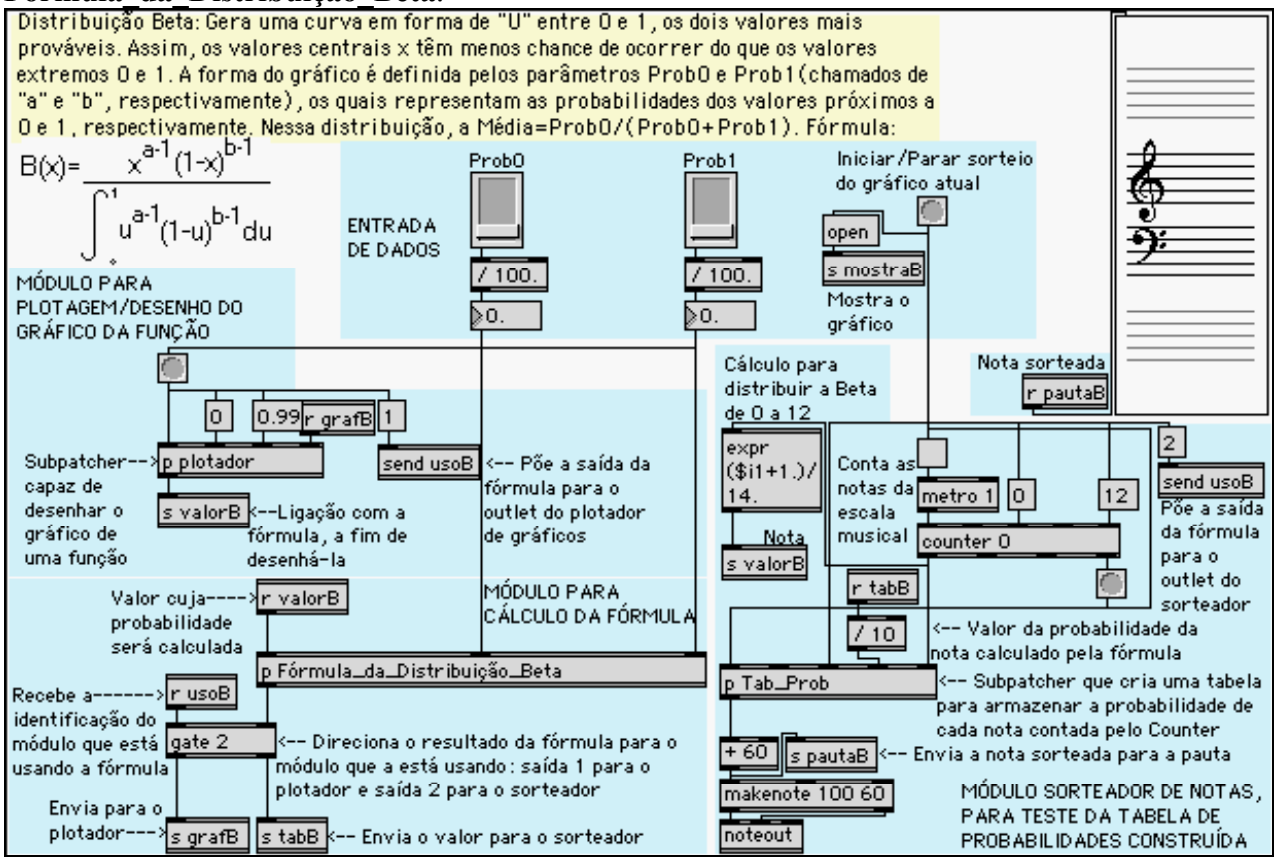


Figura 273 – Programação da Distribuição Beta

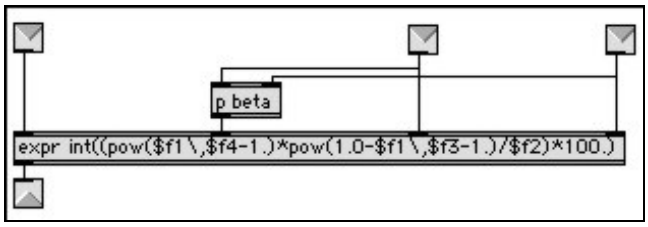


Figura 274 - Subpatch com a expressão da Distribuição Beta

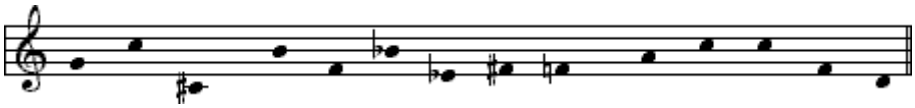


Figura 275 – Resultado obtido após a execução do programa de Distribuição Beta, com Prob0 e Prob1 iguais a 0,5

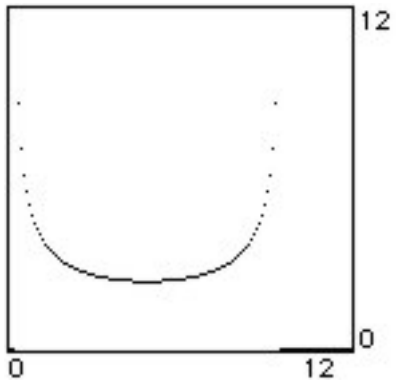


Figura 276 - Gráfico da distribuição beta(Prob0 e Prob1 iguais a 0,5)

Apresentaremos fórmulas menos complexas que a Distribuição Beta para possibilitar a construção da expressão a partir da fórmula matemática. Para exercitar a programação de fórmulas probabilísticas para geração de material musical resolva os problemas propostos na figura 253. A solução do exercício 1 da figura 253 é apresentado na figura 277. Observe como a fórmula da Distribuição Linear é escrita no objeto expr. A entrada de valores do patch é composta apenas pelo valor Máximo que nada mais é do que o ponto de início da reta decrescente no plano X,Y.

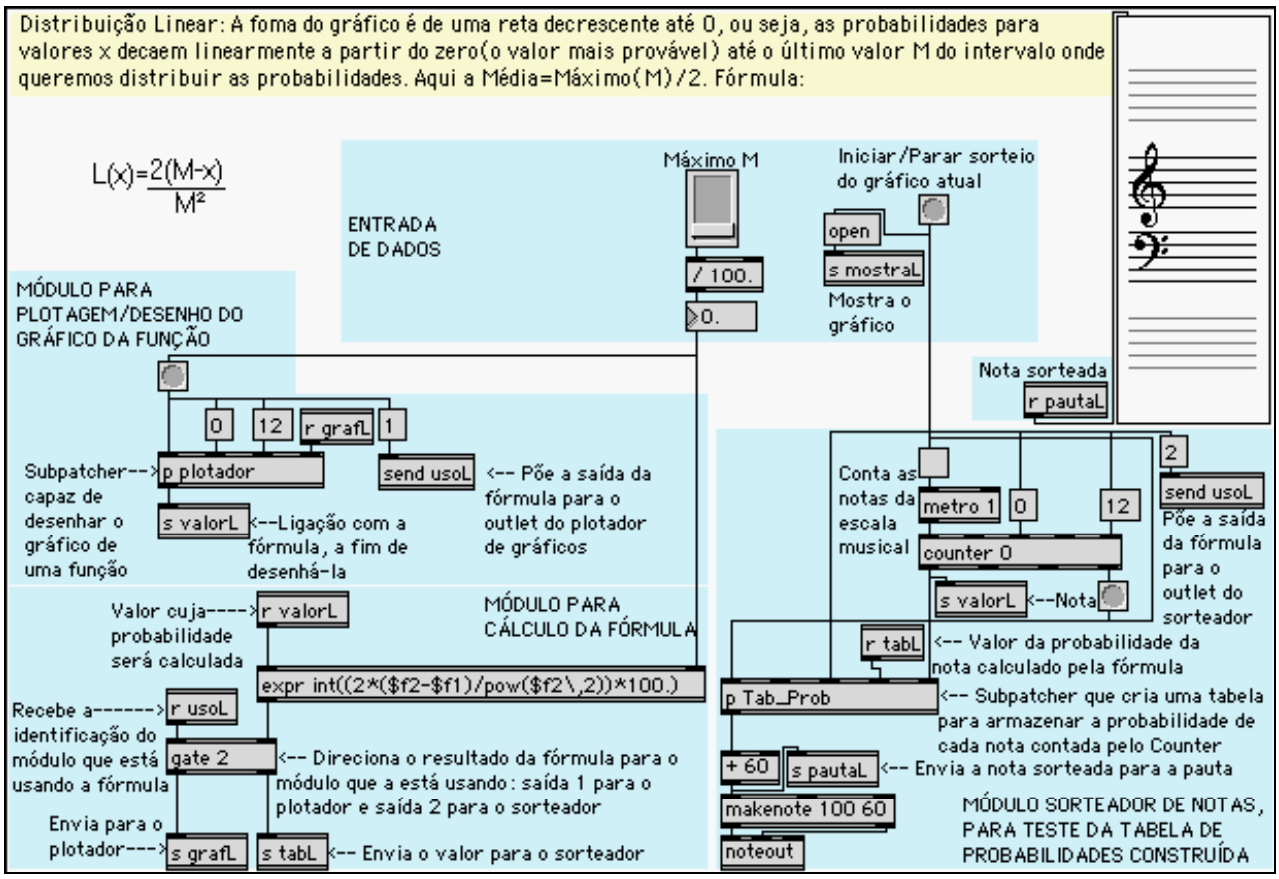


Figura 277 – Solução do exercício para a progranação da Distribuição Linear

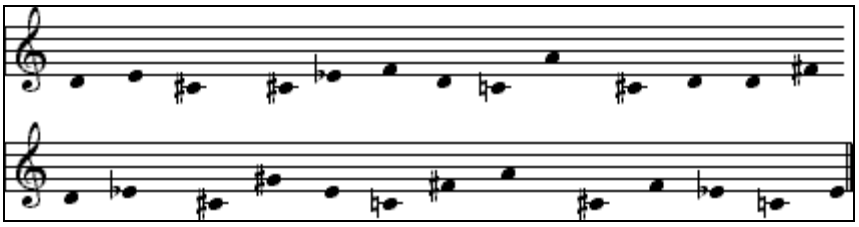


Figura 278 – Resultado obtido a partir da distribuição linear com Máximo igual a 12

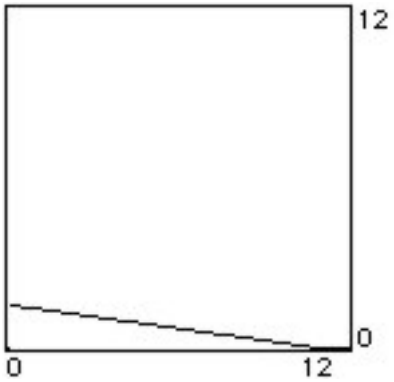


Figura 279 - Gráfico da distribuição linear(Máximo igual a 12)

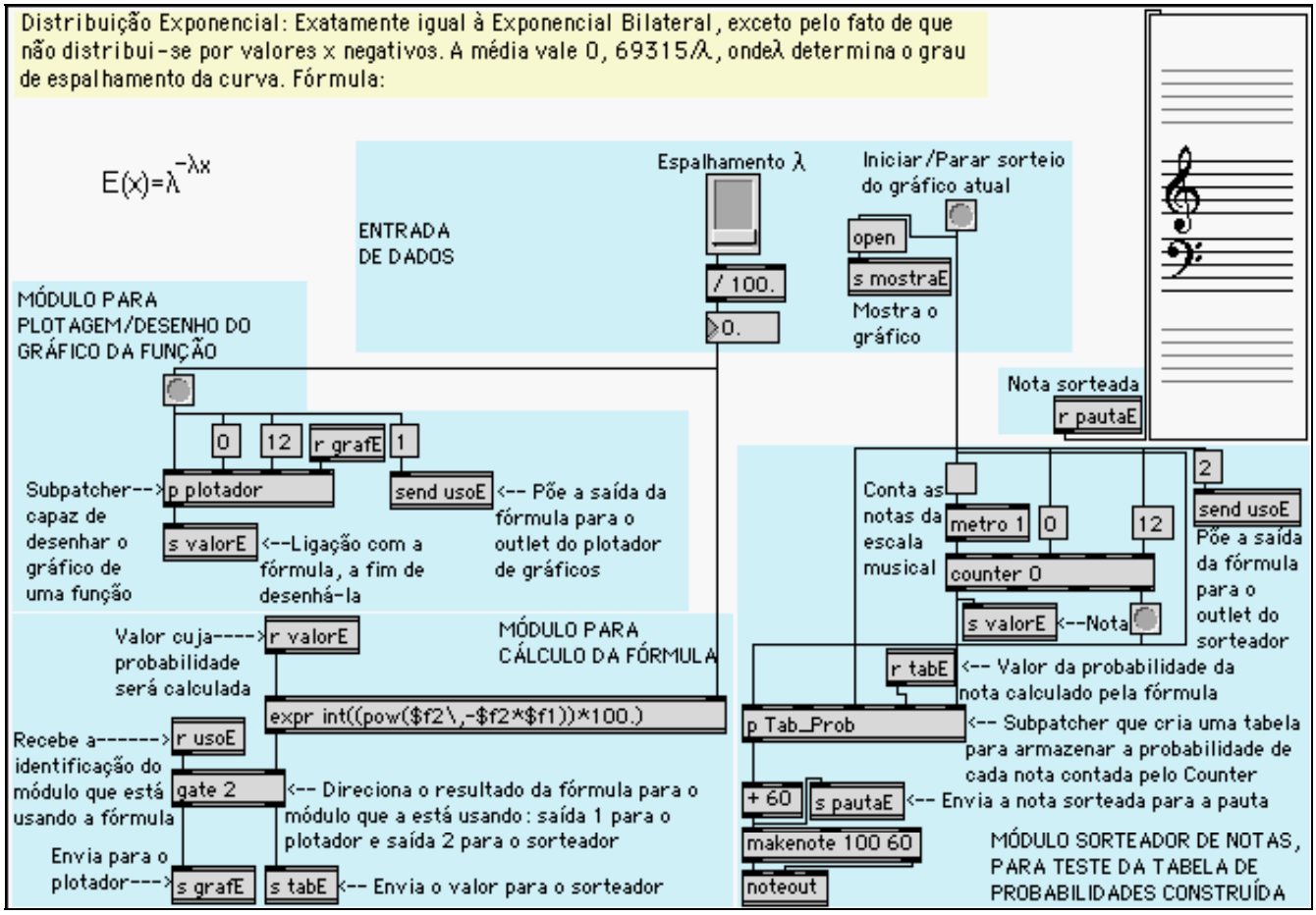


Figura 280 – Programação da Distribuição Exponencial

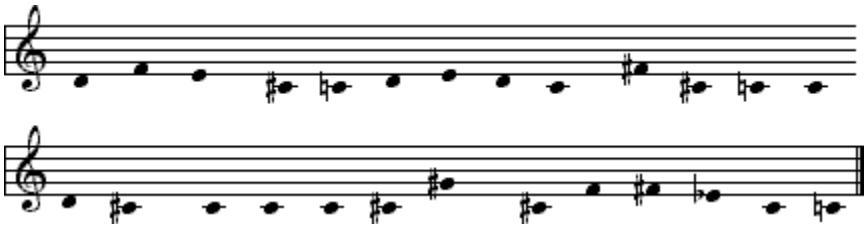


Figura 281 – Resultado obtido após a execução do programa de Distribuição Exponencial, com Espalhamento de 1,3

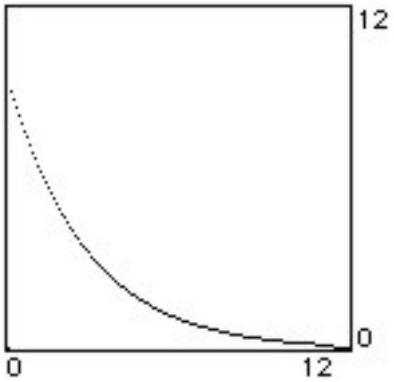


Figura 282 - Gráfico da distribuição exponencial(Espalhamento 1,3)

Outra solução do exercício 1 da figura 253 é apresentado na figura 283. Observe como a fórmula da Distribuição de Cauchy é escrita no objeto *expr*.

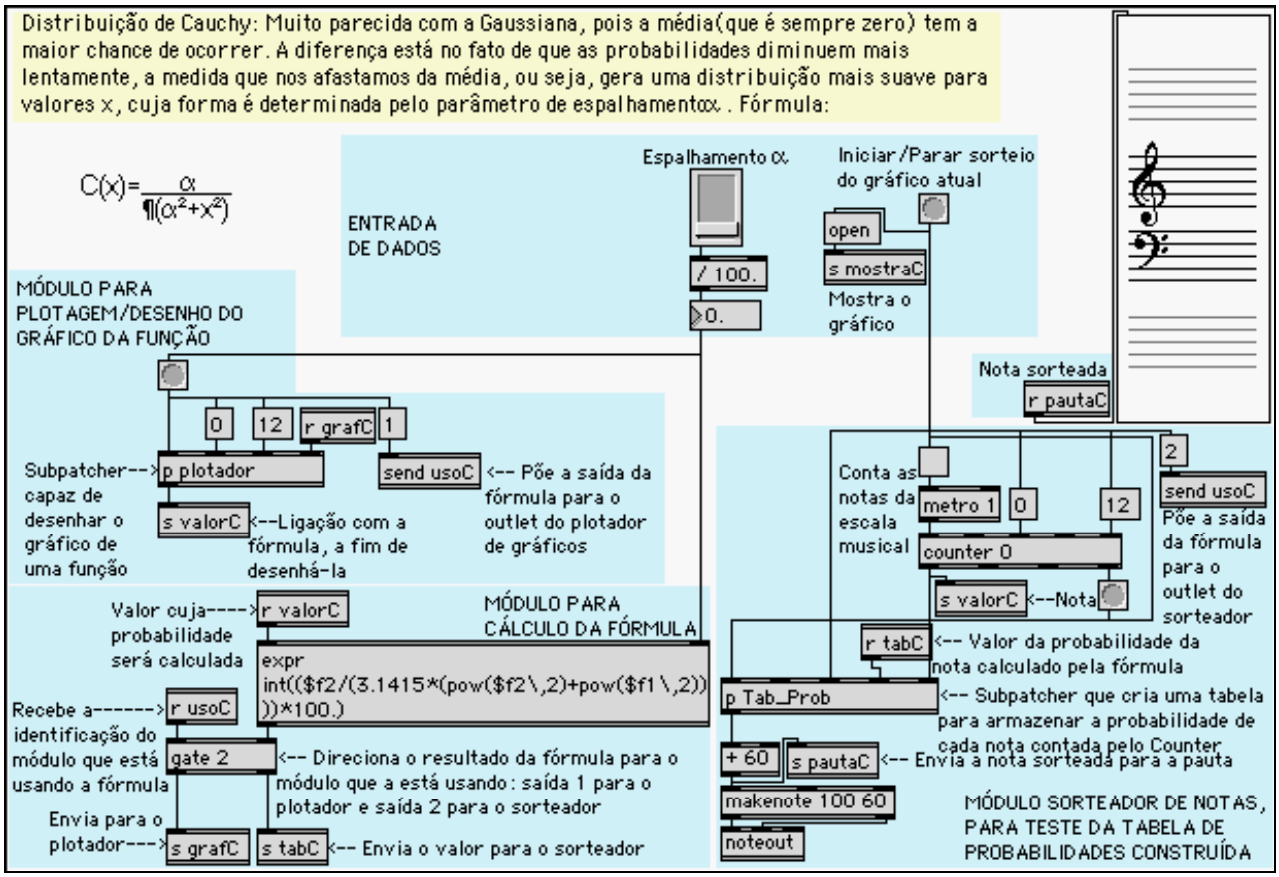


Figura 283 – Solução do exercício para a progranação da Distribuição de Cauchy

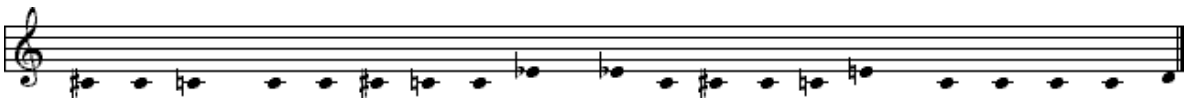


Figura 284 – Resultado obtido após a execução do programa de Distribuição de Cauchy com Espalhamento 1

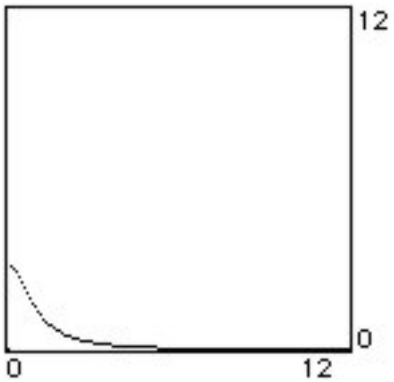


Figura 285 - Gráfico da distribuição de Cauchy(Espalhamento 1)

Para exemplos, em Basic, de programas probabilísticos para a composição ver [WIN 87]. Para exemplos na linguagem C ver [DOD 97].

As possibilidades para o uso das distribuições randômicas na composição musical são ilimitadas. A música composta por programas de probabilidade pode exibir padrões e sucessões imprevisíveis que poderão enriquecer a composição. Estes padrões e sucessões não são idéias próprias do compositor, mas sim sugestões oferecidas pelos programas probabilísticos e escolhidas pelo compositor. Com um método probabilístico, o compositor também pode planejar a forma total da peça enquanto o computador encarrega-se de preencher, em detalhes, utilizando-se dos algoritmos fornecidos pelo próprio compositor. Lejaren Hiller, Iannis Xenakis e outros elaboraram um uso composicional dos padrões de distribuição randômica [HIL 70], [XEN 60].

8.1.8 AULA 8 - Probabilidades Condicionais

É a categoria de probabilidades cujo a ocorrência de um evento particular é baseada no resultado de um ou mais eventos anteriores.

8.1.8.1 Cadeias de Markov

Segundo [ROA 96], a Cadeia de Markov é uma das primeiras e mais populares estratégias para composição algorítmica por computador [HIL e ISS 59]; [XEN 60]. Formulada pela primeira vez pelo matemático Russo A . A . Markov (1856-1922), a cadeia de Markov é um sistema de probabilidade no qual a probabilidade de um futuro evento é determinada pelo estado de um ou mais eventos num passado imediato.

As probabilidades das cadeias de Markov podem ser colocadas em uma matriz de transição de estados. A matriz de transição de estados na figura 286 é uma cadeia de Markov para a composição de uma melodia simples, usando as notas da escala pentatônica: G, A, C, D, E. As células mostram a probabilidade de uma próxima nota, dado uma nota atual mostrada à esquerda da linha. Por exemplo, se a nota de início é G, nós procuramos pela primeira linha da matriz para ver as probabilidades para a próxima nota. Nós vemos que existe 50% de chance que outro G seja tocado, e 50% de probabilidade que um C seja tocado. Note que 0% de probabilidade para A,D e E significa que somente outro G ou um C irão seguir o G.

	G	**A**	**C**	**D**	**E**
G	0,5	0	0,5	0	0
A	0	0,25	0,25	0,25	0,25
C	00,25	0,25	0,25	0,25	0,25
D	0,5	0	0,5	0	0
E	0,33	0	0	0,33	0,33

Figura 286 – Na matriz, as linhas representam a probabilidade da próxima nota, de acordo com a nota corrente

A figura 287 mostra uma seqüência possível de notas geradas por esta matriz. Em uma escala local, isto é, na sucessão de duas notas, o caráter da cadeia é claro. Em uma escala global maior, contudo, este tipo de cadeia de Markov pode gerar um tipo de "vagar sem alvo"; não existe sentido de estrutura na frase – de início, meio e fim – nas melodias que ela gera. Parte desse comportamento vago é que, para gerar a próxima nota, a cadeia olha atrás somente um estado, para a próxima nota.

G	C	D	C	D	G	C	G	C	C	A	E	D	C	G

Figura 287 – Vetor de notas gerado pela Matriz de transição de estados de Markov

O exemplo da figura 288 é um patch que ensina como programar Cadeias de Markov no computador utilizando o Max. Existe um objeto destinado à sua criação: o *prob*. Para programá-lo, primeiramente é necessário especificarmos cada uma das transições da tabela. Isso é feito através do envio de mensagens do tipo "estado_atual", "próximo_estado", "chance_de_ele_acontecer", as quais significam que um certo estado pode passar ao outro com uma certa probabilidade. Após especificarmos as transições, basta disparar o objeto através de um "bang", fazendo com que ele passe a sortear novos estados, sempre baseando-se nos dados informados. Experimente ler as explicações, entrar com os valores na tabela e ouvir os resultados. Depois de estudar como foi construído esse patch e resolva o exercício proposto.

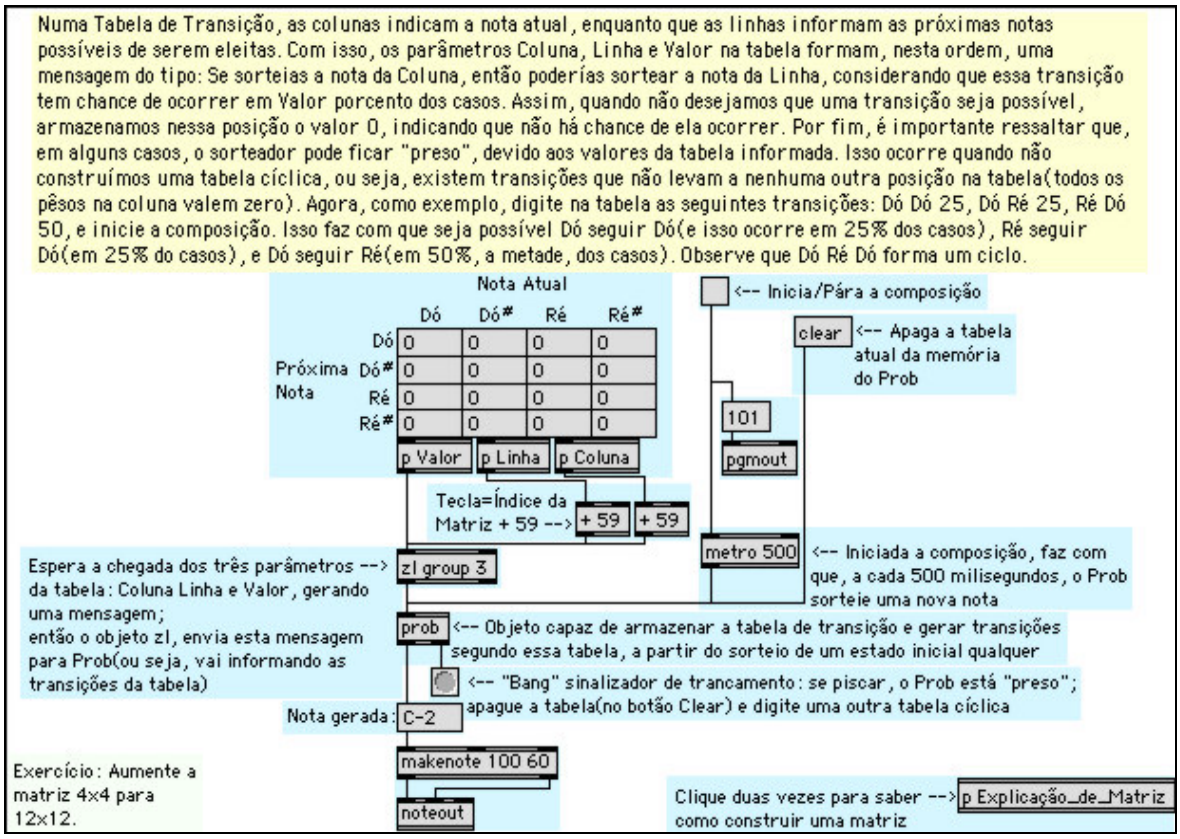


Figura 288 – Programação da Cadeia de Markov

Para resolver o exercício proposto é necessário compreender como é possível criar graficamente uma matriz para a entrada dos dados na Cadeia de Markov. Na figura 289 é apresentado o método para a programação gráfica da matriz. Os *subpatchers* Valor, Linha e Coluna servem simplesmente para unificar todas as ligações de saídas de valores(vindos de cada caixa de números), linhas(ligações das mensagens que indicam à que linha pertence cada uma das caixas de números acionadas) e colunas(mensagens que indicam as colunas), respectivamente. Isso é feito porque nunca é possível saber de antemão qual das ligações de valor, linha e coluna serão acionadas na tabela. Assim, concetando-se todas ao mesmo ponto, permite buscar qualquer uma destas informações na mesma posição, simplificando o programa. O exercício da figura 288 propõe que seja criada uma matriz 12x12 para todas as notas da escala cromática.

Observe com cuidado as indicações da figura 289 e resolva o exercício proposto baseado nela.

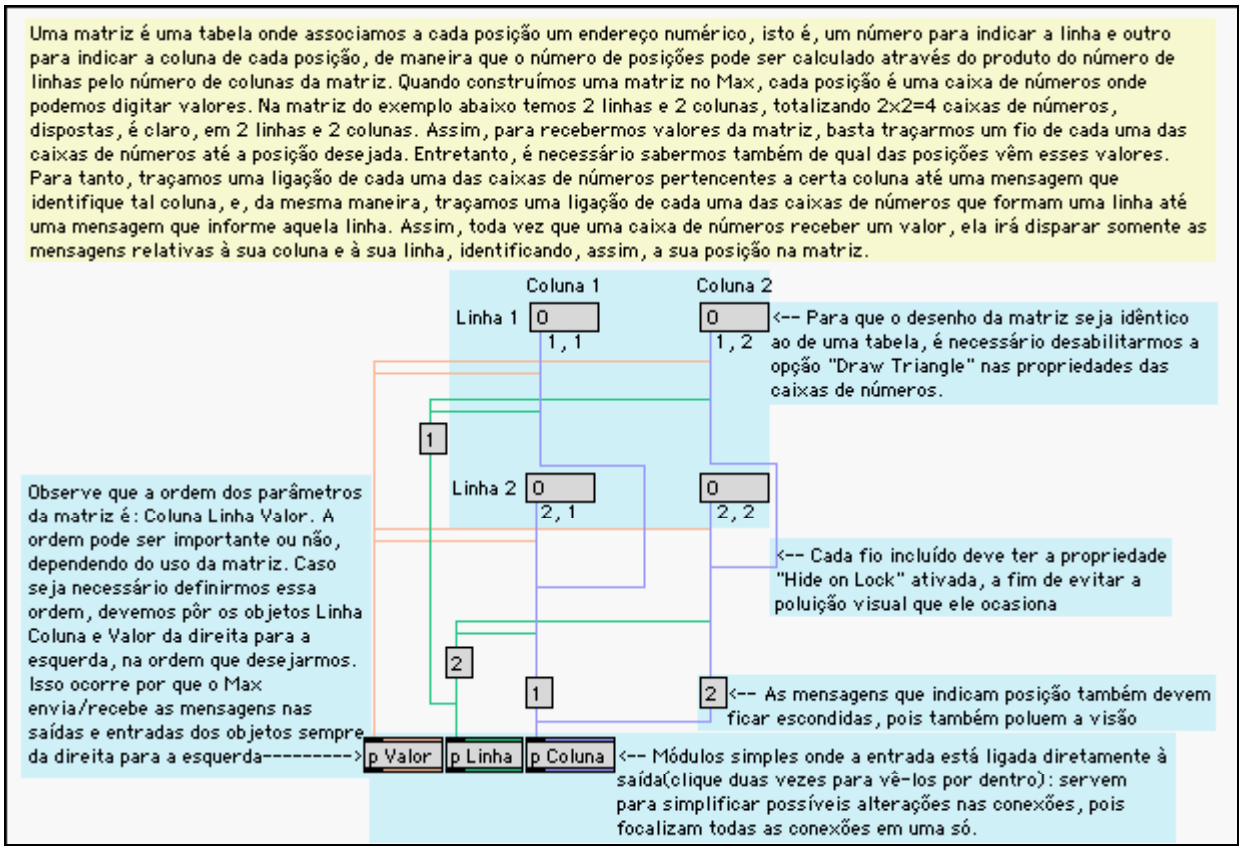


Figura 289 – Programação da interface gráfica da Cadeia de Markov

A solução para o exercício é apresentada na figura 290. Compare com o seu resultado.

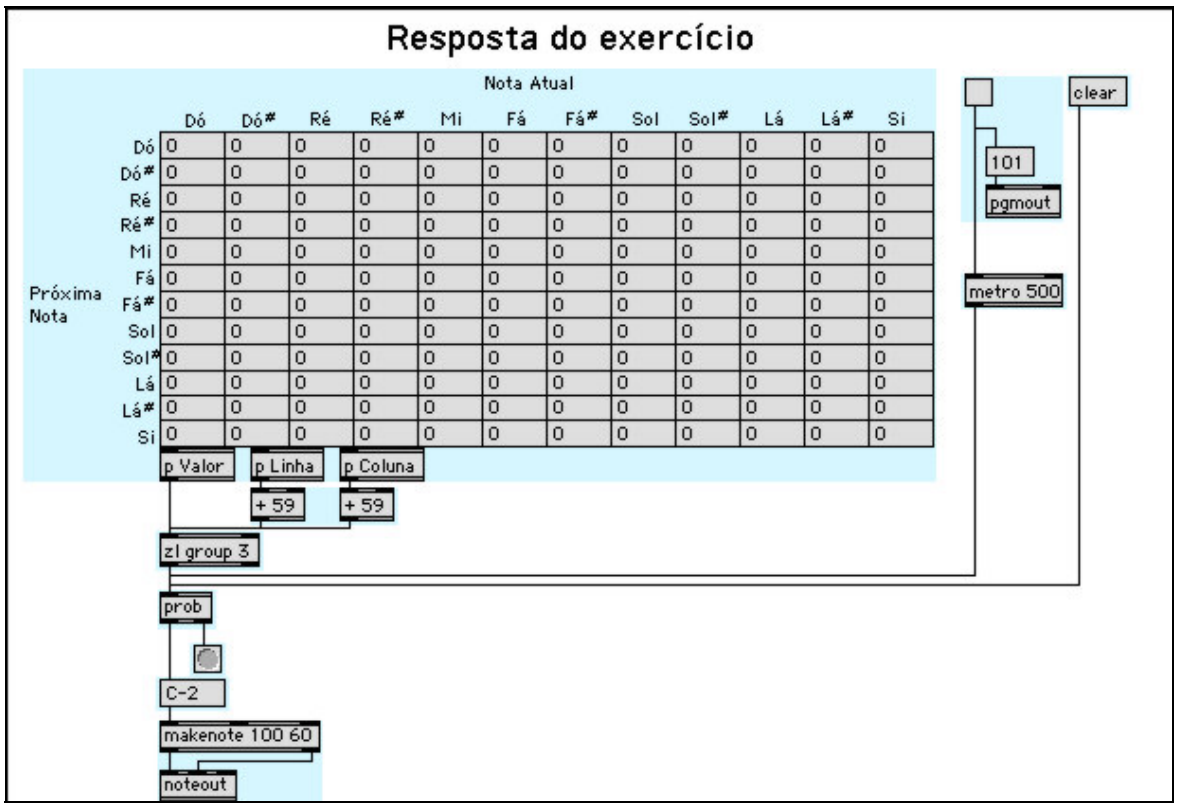


Figura 290 – Markov para 12 notas

Pode-se aumentar a sofisticação desse método pelo incremento da ordem da cadeia. A ordem da cadeia de Markov indica o número de estados anteriores que são levados em conta. Eventos em cadeia de ordem zero (equivalente à uma tabela de probabilidade regular sem procura para trás no estado anterior) são independentes um dos outros. Eventos numa cadeia de segunda ordem procuram por dois estados anteriores, e assim por diante. O exemplo da figura 291 apresenta a programação para a criação de uma Cadeia de Markov de Segunda Ordem. Este exemplo permite a entrada de poucas notas devido a complexidade e a quantidade de trabalho despendido para um conjunto maior de notas. No entanto, ilustra como outras ordens da Cadeia de Markov podem ser programadas.

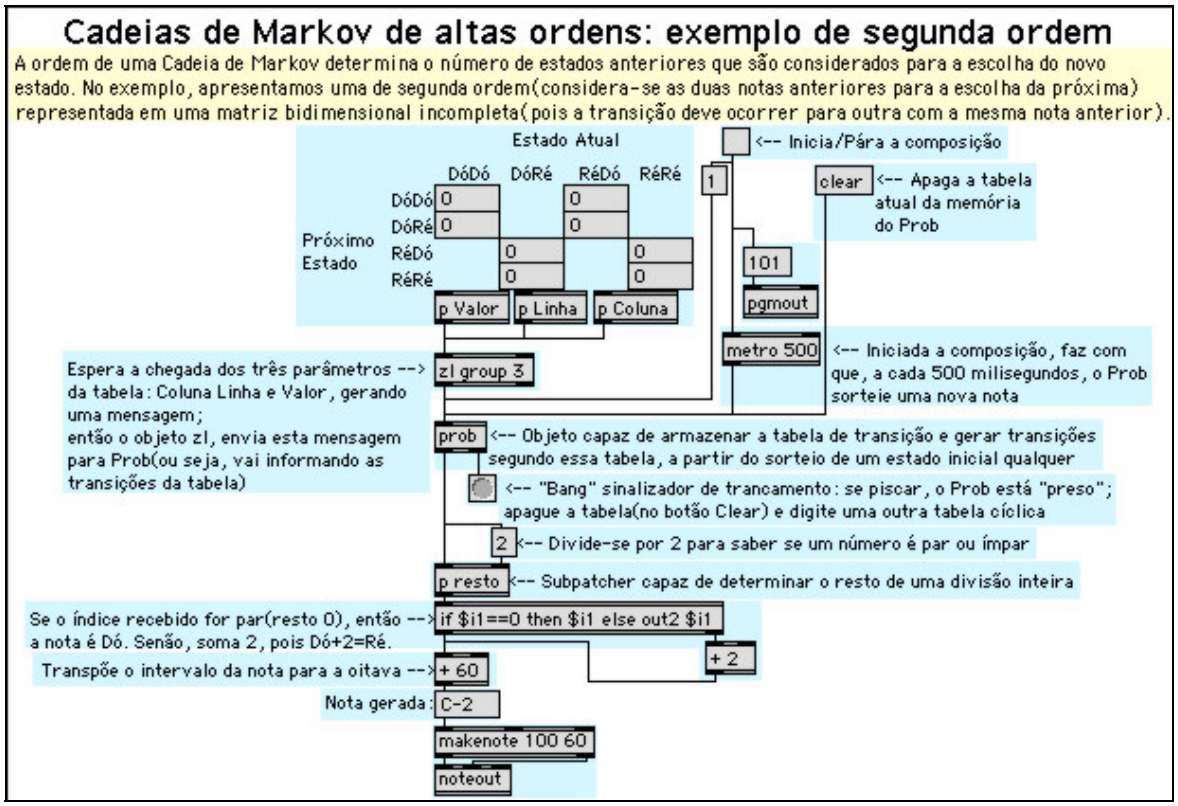


Figura 291 – Cadeia de Markov de Segunda Ordem

Moorer verificou que cadeias de ordem oito produzem seqüências de frases justapostas [ROA 96]. Estudos de composições de hinos mostraram que cadeias com baixa ordem geraram melodias randômicas, enquanto cadeias de alta ordem consistem em partes de hinos originais agrupados, como se a primeira metade de um hino fosse colocada sobre a segunda metade de outro hino [ROA 96]. Resultados tais como esse provam que uma música possui vários níveis de estrutura, e que um sistema de composição que somente refere-se a um nível (tal como produzir frases coerentes localmente) irá apresentar deficiências em outros níveis, tal como: transições entre frases e alto nível de organização local.

8.1.8.2 Técnicas Avançadas de Markov para a Música

Várias técnicas existem para persuadir o comportamento complexo musical de uma cadeia de Markov [JON 89]. É possível estender o conceito de estado de uma única unidade, como uma nota, para unidades de múltiplos valores, como um conjunto de notas: (nota, duração, amplitude, instrumento). Para considerar todas as variações possíveis de notas é necessário uma matriz de transição extensa.

Outra maneira de extender o conceito de estado é tratar um agrupamento como acorde, frases, ou compassos em um estado único. Então ao invés de compor seqüências de notas individuais, a cadeia irá gerar seqüências de acordes, frases ou compassos [ROA 96].

8.1.8.3 Cadeias de Markov Hierárquicas

Segundo [ROA 96], para manipular diferentes níveis de organização musical, é possível organizar cadeias de Markov em hierarquias. Uma cadeia que gera estruturas de alto nível pode selecionar uma entre diferentes tipos de seções. Por exemplo, um

movimento rápido, um movimento lento ou um movimento de uma linha particular de notas. Em uma seção de nível intermediário as cadeias podem selecionar a seqüência de frases contidas na seção. Os detalhes de cada tipo de frase podem ser preenchidos pela cadeia de baixo nível. A linguagem musical HMSL (Hierarchical Music Specification Language) suporta este tipo de construção [POL, ROS, and BUR 87]. Para mais informações sobre cadeias de Markov em música veja [AME 89].

8.1.8.4 Improvisação com Computadores

A improvisação pode ser realizada com um processo de improvisação através do computador, simulando o diálogo entre dois ou mais músicos.

O computador pode estar num "estado" de improvisação musical esperando qualquer entrada do instrumentista. A partir dessas entradas, um algoritmo que contenha variáveis randômicas pode produzir variações contínuas ou outros resultados imprevisíveis. Um bom trabalho de improvisação pode requerer várias seções, onde cada seção terá um comportamento estático, a fim de desenvolver uma resposta contínua às entradas do instrumentista. O caráter de cada seção pode ser diferente mesmo que esta produza uma grande quantidade de variações sobre o material musical fornecido pelas entradas do instrumentista. As seções de improvisação envolvem também a mudança gradual de parâmetros ao longo do tempo [ROA 96].

8.1.9 AULA 9 - Fractais

Assim como as cadeias de Markov, os Fractais representam outra classe de sistemas probabilísticos que também mantém um contexto [ROA 96].

A mais intrigante classe de ruídos de fractais é a pura $1/f^1$. No ruído puro 1/f , a probabilidade de uma dada freqüência ocorrer diminui com o aumento de freqüência [ROA 96].

Seqüências geradas por um algoritmo de ruído puro 1/f possui interessantes propriedades matemáticas. Por exemplo, ele relaciona-se logaritmicamente com o passado. Assim, a atividade média dos últimos dez eventos tem muita influência no seu valor corrente. Assim, o processo 1/f tem uma relativa memória a longo prazo constituindo-se num processo atrativo para processos musicais que referem-se a eventos no passado, tais como melodias tonais que vagam por outras tonalidades e retornam ao contexto tonal inicial. Cientistas, que estudam fractais, tem observado que os padrões de altura e intensidade das músicas tradicionais as vezes tem a mesma forma do padrão 1/f. [CLA e VOS 78].

Além da seleção de alturas, algoritmos fractais podem ser aplicados a qualquer outro parâmetro musical. Por exemplo, [McN 86, 89] usa algoritmos fractais para gerar efeitos de vibrato, e [WAS e KUR 89] aplicaram fractais para o controle de timbre na síntese granular.

8.1.9.1 Auto-similaridade

Uma importante propriedade de ruído 1/f é a auto-similaridade. Numa seqüência de auto-similaridade, o padrão de pequenos detalhes espelham o padrão de formas maiores, mas apenas numa escala diferente. Esse conceito também é encontrado na abordagem da "célula germinativa", na qual uma grande escala musical formada é gerada pela elaboração de um minúsculo fragmento, tal como uma linha de notas ou um conjunto de relações. Uma maneira de aplicar o conceito de auto-similaridade na música polifônica é gerar um motivo com poucos intervalos e pouca duração. A isso é adicionado uma repetição rápida do mesmo motivo que inicia em cada uma das notas da primeira linha [DOD 88], [THO 91].

Um exemplo de algoritmo fractal para a geração de alturas é apresentado na figura 292.

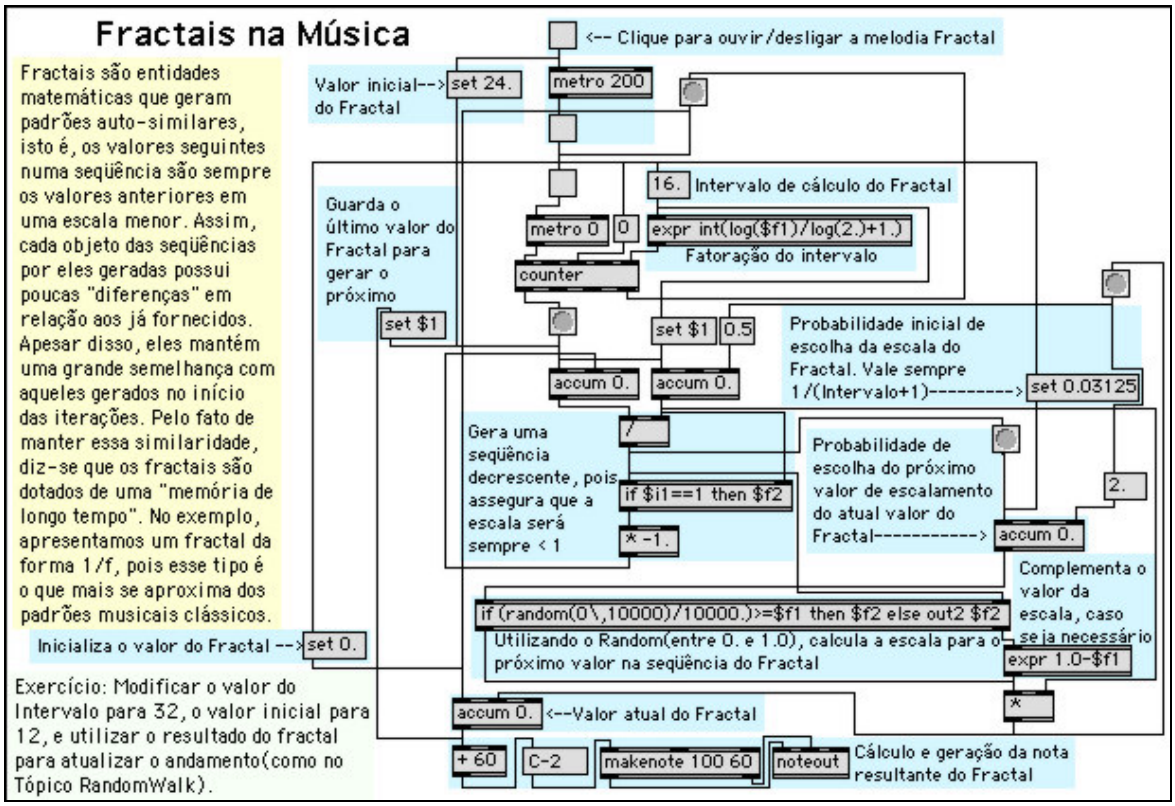


Figura 292 – Exemplo de programação de um fractal para geração musical

Uma solução para o exercício proposto é apresentada na figura 293. Nele mostramos as alterações nos parâmetros de cálculo afim de observarmos sonoramente a função de cada um deles no programa. Inicialmente, forçamos o programa a iniciar com o valor 12, fazendo com que a composição se inicie numa oitava mais baixa. Definimos, também, um novo limite para a geração de notas por parte do fractal igual a 32. Isso faz com que a composição apresente um maior número de notas, com alturas mais agudas. É bom lembrar, que a estrutura matemática do fractal nos obriga a definir valores dentro dos resultados possíveis da função de probabilidade 1/(Intervalo+1). Assim, definimos um novo valor para a probabilidade de escolha do escalamento dos padrões do fractal, igual a 1/(32+1)=0,015625, menor do que a anterior. Isto faz com que o fractal gere padrões que se mantém durante mais tempo na sequência de frases gerada, diminuindo a randomicidade da composição. Além disso, utilizamos os valores numéricos gerados pelo fractal não só para a geração de notas, mas também para o cálculo de um novo andamento a cada nota tocada. Como utilizamos um fator de adaptação multiplicativo de valor 5 (isto é, multiplicamos os valores retornados pelo fractal por 5), podemos concluir (e comprovar sonoramente também) que, quanto mais altura tiver a nota gerada, maior será a pausa adicionada à composição.

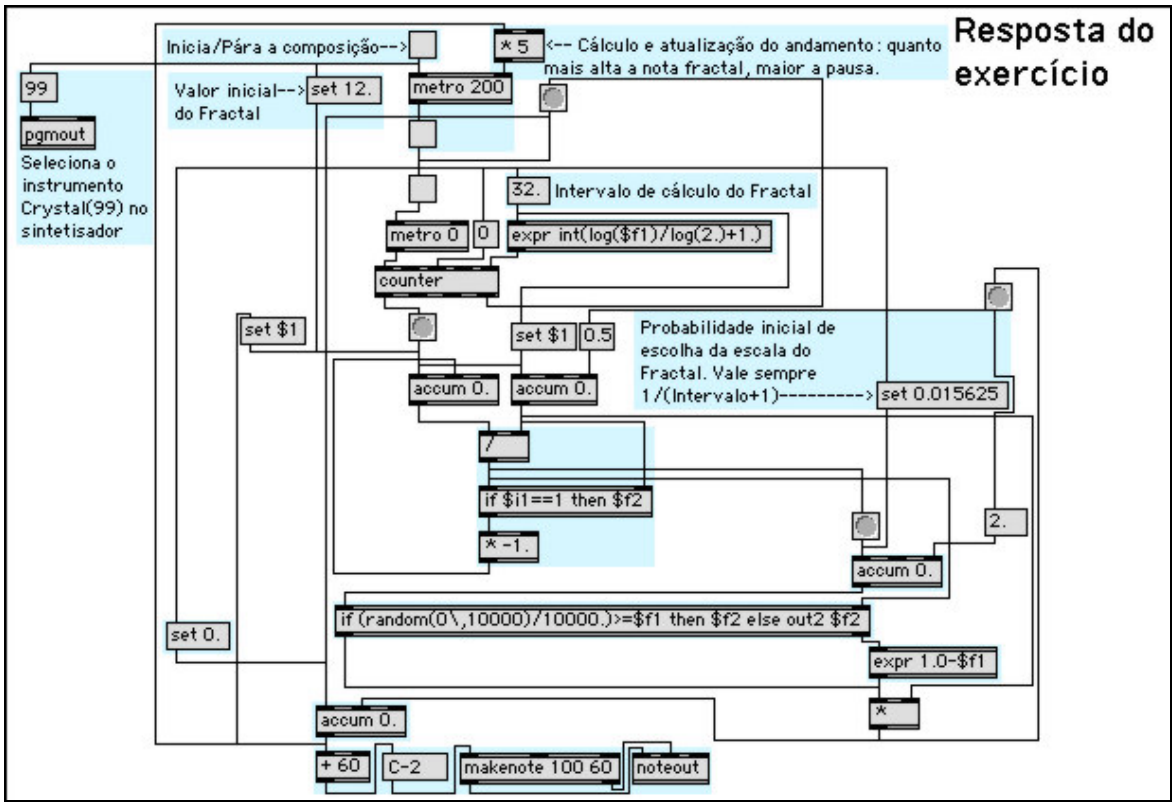


Figura 293 – Solução para o exercício com fractais

8.2 MÓDULO DE PROGRAMAÇÃO PARA EDUCAÇÃO MUSICAL

Os exemplos e exercícios práticos, que constituem o Módulo de Programação para Educação Musical, foram implementados com base nas obras de [GOO 90] e [RED 88].

Antes de iniciar os exemplos práticos e os exercícios de programação em educação musical, o aluno necessita de uma fundamentação na área de software musical para educação. Após entender os princípios dessa área, o aluno deverá ler com muito cuidado as explicações e apontamentos nos programas exemplo para um perfeito entendimento da utilização dos objetos em conjunto.

O músico deverá utilizar programas de educação musical antes de iniciar o estudo através desse módulo. Além disso, o músico necessita ter conhecimento prévio dos métodos de ensino de música que, posteriormente, poderão ser adaptados para o computador. A importância da prática pedagógica e da experiência com alunos de música será fundamental para a criação dos programas educacionais. Quanto mais profundo for o conhecimento do programador em Educação Musical, maior serão as possibilidades de abstrair o conhecimento e representá-lo sob a forma de software.

As linguagens *"amigáveis"* permitem aos programadores, "não profissionais", a chance de criar suas próprias aplicações musicais. A programação é de fácil aprendizado e permite a implementação de software através de um ambiente interativo e de uso intuitivo.

A linguagem interpretada apresenta vários benefícios para o ensino de programação de computadores para a música. Os programas escritos numa linguagem interpretada são convertidos em linguagem de máquina ao mesmo tempo em que suas instruções estão sendo executadas. Portanto, enquanto o programa é executado, o computador pode informar os erros cometidos pelo programador.

O HyperCard é uma linguagem *"amigável"* interpretada possuindo uma pilha (biblioteca) com uma linguagem musical embutida chamada HyperMIDI. Para o aprendizado de programação para aplicações na área da educação musical, é necessário utilizar linguagens que disponibilizam comandos para a geração de som por áudio e MIDI. Inicialmente estudaremos a programação básica utilizando a linguagem HyperCard. A filosofia de funcionamento do ambiente HyperCard é de fácil entendimento pois assemelha-se muito à navegação utilizada pelos browsers na internet. Este ambiente e a linguagem de programação serão utilizados para o desenvolvimento de aplicações musicais. Posteriormente, será introduzida a linguagem HyperMIDI para controle da comunicação entre instrumentos musicais e computadores.

8.2.1 Ambiente de Programação HyperCard

O HyperCard é um software de autoria para Macintosh. Através da criação de uma pilha é possível sobrepor vários cartões (ou páginas) para formar o programa. Cada cartão é ligado à outro por um endereço que é o nome do cartão.

A seguir apresentamos as principais telas de comandos e menus do HyperCard.

8.2.1.1 Conhecendo o ambiente HyperCard

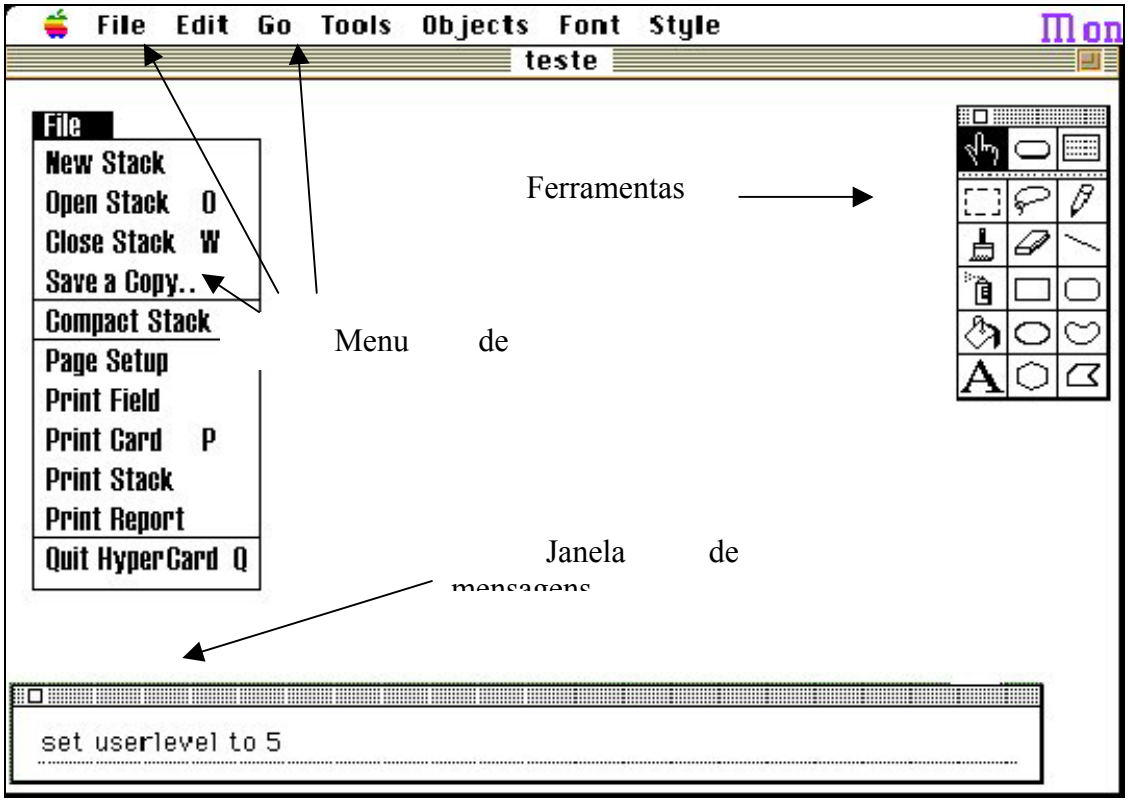


Figura 294 – Ambiente *Hypercard*

8.2.1.2 Resumo dos principais Menus e Opções

Stack

Primeiramente é importante introduzir o conceito de pilha (*Stack)* em HyperCard, que nada mais é do que uma coleção homogênea de informações. Cada pilha será um arquivo separado no Macintosh podendo ser vista como um ícone de um documento no Sistema Operacional.

Backgrounds

Toda a pilha HyperCard tem pelo menos um fundo (*background*), mesmo que seja branco. Ele permite que se ilustre o fundo de tal maneira que facilite ao usuário identificar seus programas visualmente.

Cards

Um cartão (*card*) pode conter informações textuais, gráficas ou ambas. Vários cartões podem ser empilhados constituindo uma pilha.

Field

São campos que podem ser selecionados na caixa de ferramentas e servem, além de outras coisas, para a inserção de textos nos cartões.

Menu **Edit** é utilizado para realizar operações de edição de cartões, além de inserção de estilo de texto, *background* e escolha de ícones.

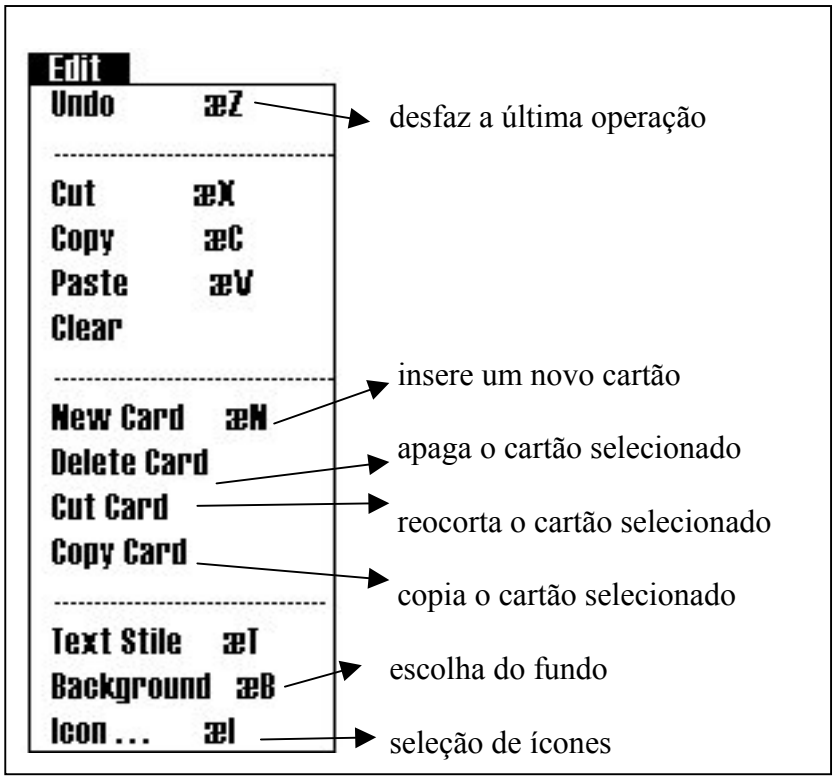


Figura 295 - Menu Edit

O Menu **Go** disponibiliza a navegação pelos cartões da pilha. Possibilita também enviar mensagens e comandos ao HyperCard.

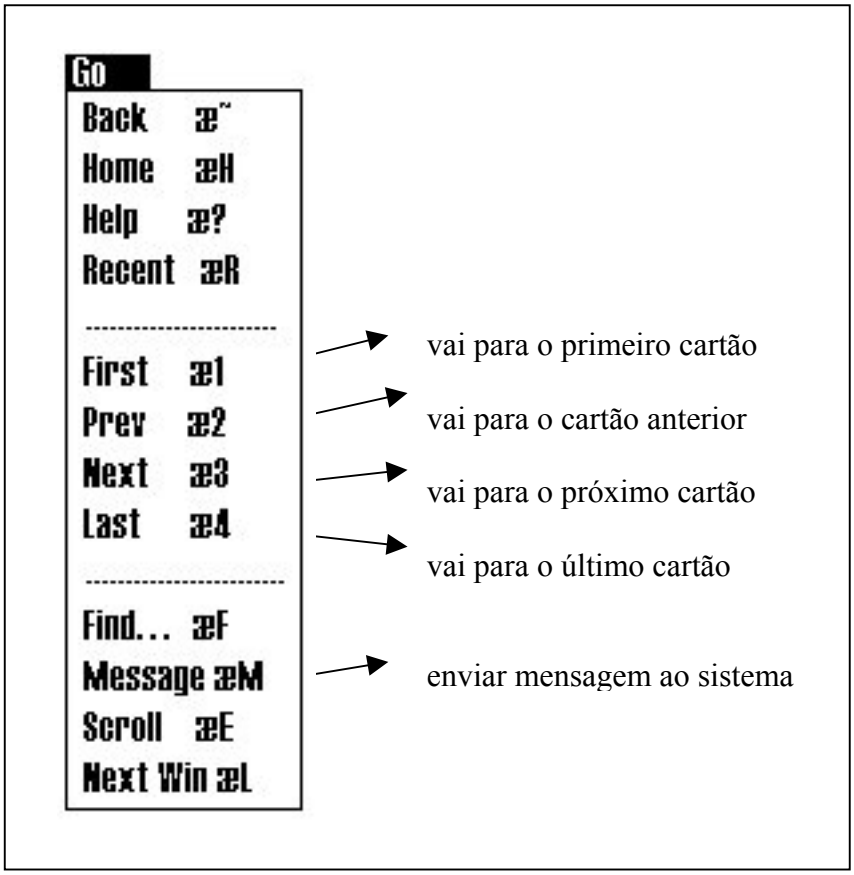


Figura 296 – Menu Go

O menu **Object** permite operações de criar novos botões, campos e *background* além de obter informações a respeito dos mesmos.

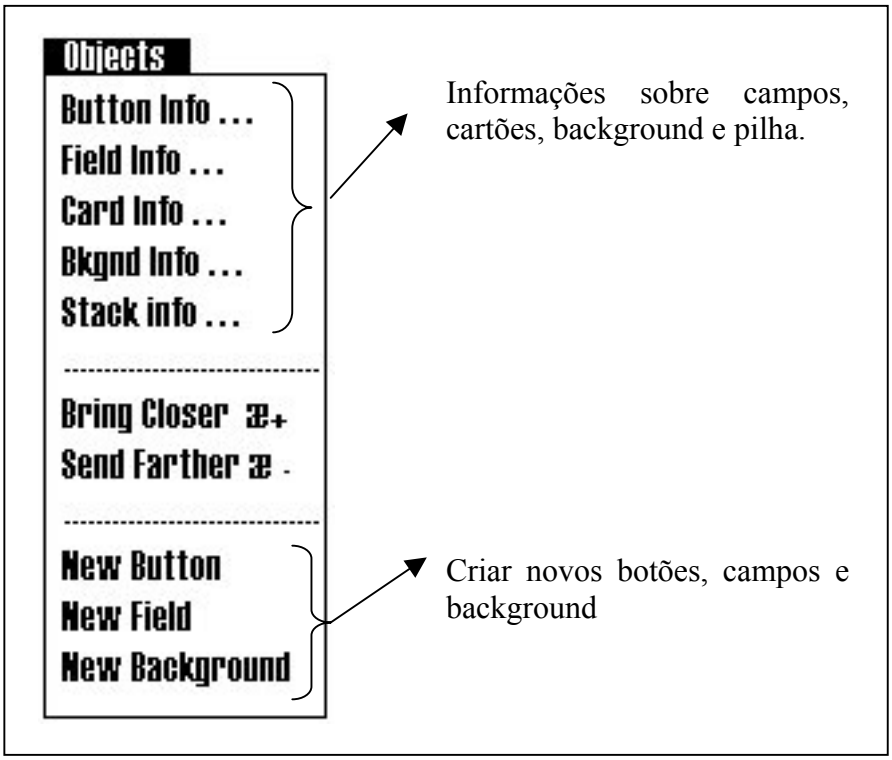


Figura 297 – Menu *Object*

8.2.2 AULA 1 - Criando Cartões e Botões

Após carregar o programa do HyperCard, abra a primeira aula para iniciar o aprendizado sobre a programação elementar do HyperCard.

Para editar uma pilha, ou alterar seu conteúdo é necessário estar no nível de usuário 5. Para tanto, vá até o menu GO e selecione a opção MESSAGE. Após uma caixa de texto aparecer na tela digite o comando SET USER LEVEL TO 5.

O primeiro exemplo fornece uma pilha inicial e permite você criar outros cartões com botões que executam ações.

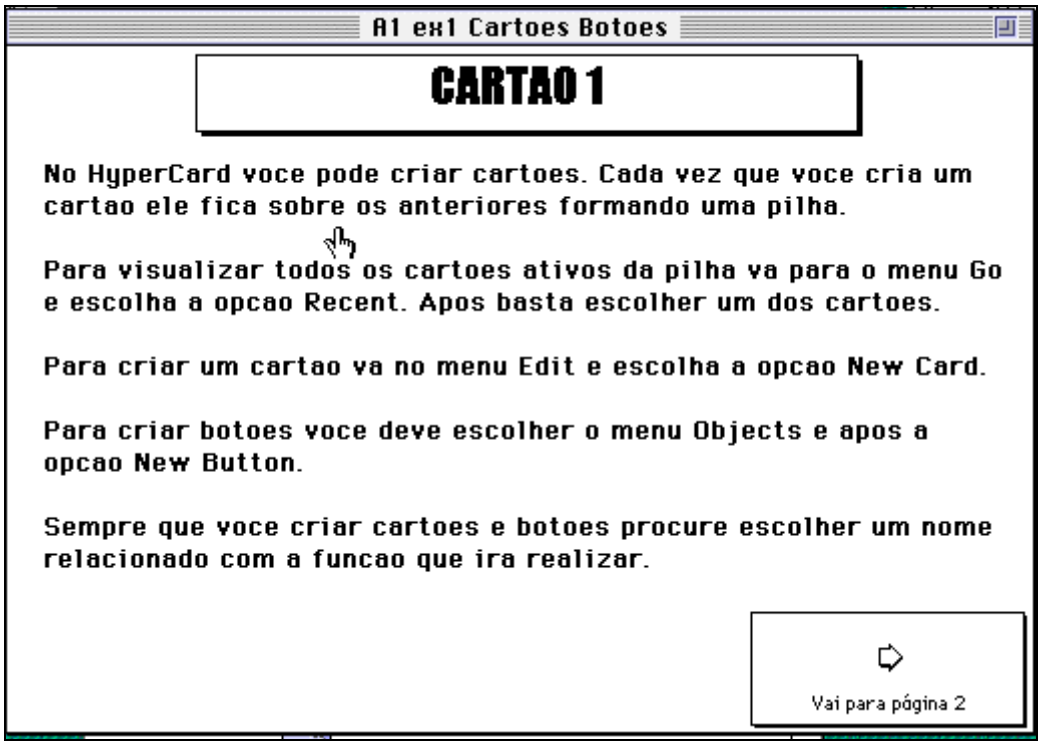


Figura 298 – Criando o primeiro cartão da pilha HyperCard com botões de navegação

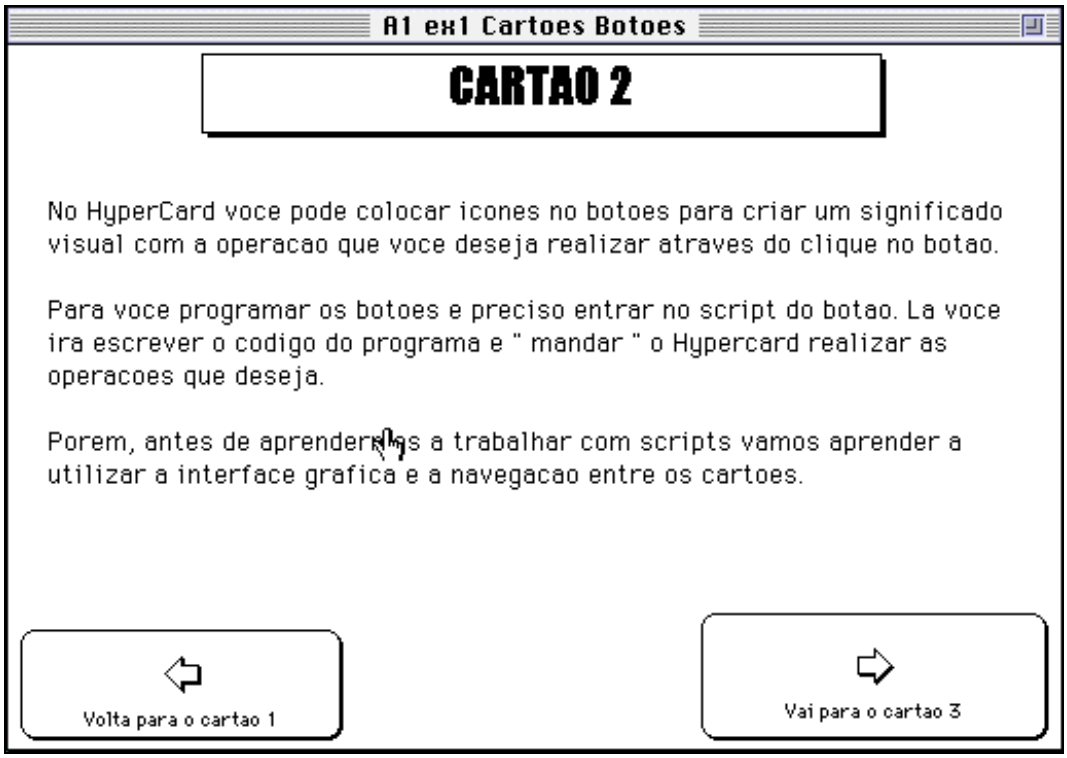


Figura 299 – Criando o segundo cartão da pilha HyperCard com botões de navegação

Para ir de um cartão para outro é necessário que você programe a ligação entre os cartões. Para isso crie o botão de ligação, abra o script do botão e escreva o comando go to "nome do cartão". Depois crie o cartão com o mesmo nome que colocou no script do botão.

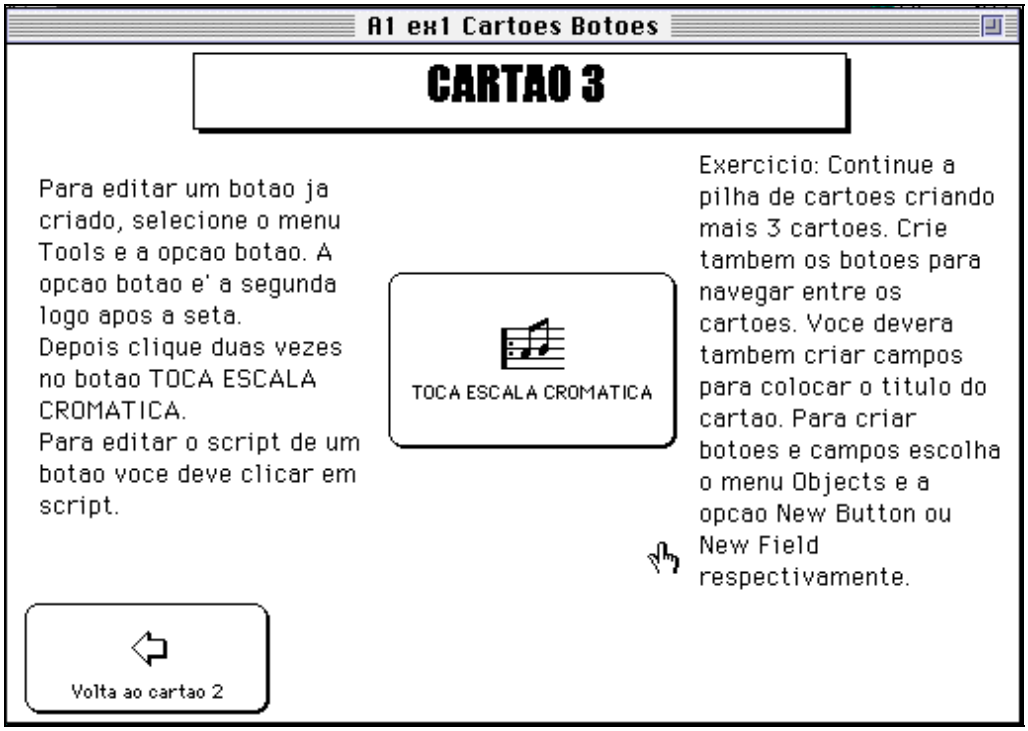


Figura 300 – Criando o terceiro cartão da pilha HyperCard com botões de navegação

Você deve ir criando nos cartões os botões que levam de um cartão ao outro. Desse modo terá um programa com várias telas. Cada tela correspondendo a um cartão.

Você pode adicionar ícones(figuras) nos botões para representar graficamente o que deseja realizar, tornando a navegação mais interativa. Escolha as setas para indicar a direção para onde deseja navegar.

Script TOCA ESCALA CROMÁTICA.

-- os dois hifens significam que o que estiver escrito após, está em forma de comentário e não será executado ao clicar o botão.

```
-- ao clicar com o louse faça
on mouseUp
    -- toque a escala cromatica em uma oitava
    play "harpsichord" tempo 120 "cq c# d d# e f f# g g# a a# c c5"
    -- fim do script
end mouseUp
```

8.2.2.1 Programando um Teclado

Nesse exercício você deve completar as teclas do piano virtual com sons do Macintosh. Observe na figura 301 o conjunto das teclas que devem ser criadas e realize a programação criando os botões e programando os scripts conforme as indicações do exercício

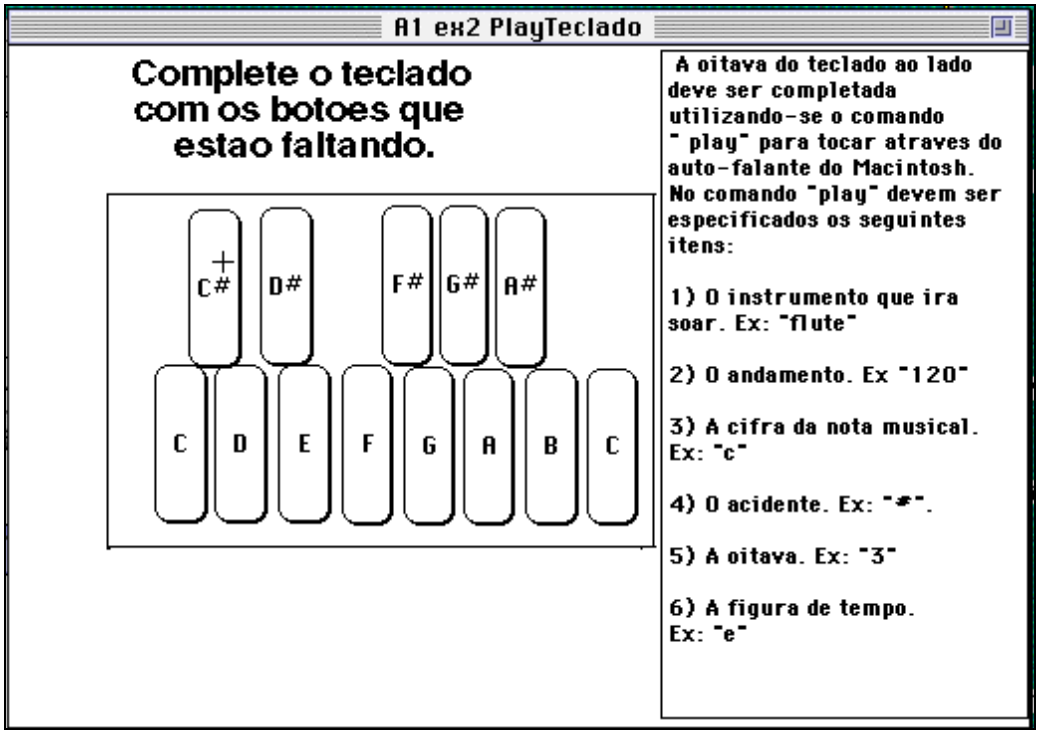


Figura 301 – Utilização do comando play

Os scripts dos botões podem ser escritos tomando por base o script do botao C do teclado.

```
on mouseUp
  play "flute" tempo 120 "c4e"
end mouseUp
```

8.2.2.2 Programando Melodias

HyperCard possui um comando de geração de música monofônico.

O comando *play* toca um som previamente gravado de maneira a permitir ser executado através do alto-falante do Macintosh ou pela porta de áudio.

Para se gravar sons reais é necessário um dispositivo externo chamado *sampler*. Este dispositivo tem um microfone e um conector de entrada. A unidade grava pequenos segmentos de qualquer som, incluindo vozes reais como sinais eletrônicos, e assemelha-se a maneira de trabalhar do audio digital no Compact Disc (CD). No caso do usuário desejar utilizar essas fontes sonoras, as mesmas devem ser convertidas no Macintosh e adicionadas a uma pilha HyperCard.

Além disso, para reproduzir diretamente o som como foi gravado, HyperCard pode tocar qualquer som com ampla variação de velocidade e tom.

HyperCard vêm equipado com os seguintes sons pré-instalados:

-Harpsichord
-Boing
-Flute

Parâmetros:

O comando *play* é seguido de um conjunto de instruções especificadas pelo músico e que são chamadas de parâmentros musicais.

O nome da voz (som digitalizado) é o único parâmetro obrigatório e deve estar entre aspas e iniciar a escrita do nome do recurso contendo a forma da onda.

O parâmetro de tempo é opcional. Caso não seja especificado, HyperCard usa o último valor configurado (ou um valor médio de 200).

Deve-se especificar as notas musicais que serão tocadas pelo comando *play* utilizando-se cifras que as respresentam (c, d, e, f , g, a, b). Os acidentes podem ser representados pelos símbolos # (sustenido), ou *b* (bemol). Se não for especificada nenhuma oitava para a primeira nota em um comando Play, HyperCard toca na oitava central. Uma vez ajustado o valor da oitava no comando, ele permanece ativo para notas subseqüentes na mesma linha de comando ou até que seja especificada sua troca.

A duração da nota pode variar de uma semibreve a uma fusa. A notação para o parametro de duração é o seguinte:

w	semibreve – (whole note)
h	mínima – (half note)
q	semínima – (quarter note)
e	colcheia – (eighth note)
s	semicolcheia – (sixteenth note)
t	fusa – (thirty-second note)

Pode-se estender a duração da nota pela metade do seu valor digitando o ponto de aumento depois do valor da nota.

Para tocar uma melodia musical contínua, coloque todas as notas da melodia em seqüência depois de um comando play conforme figura 302.

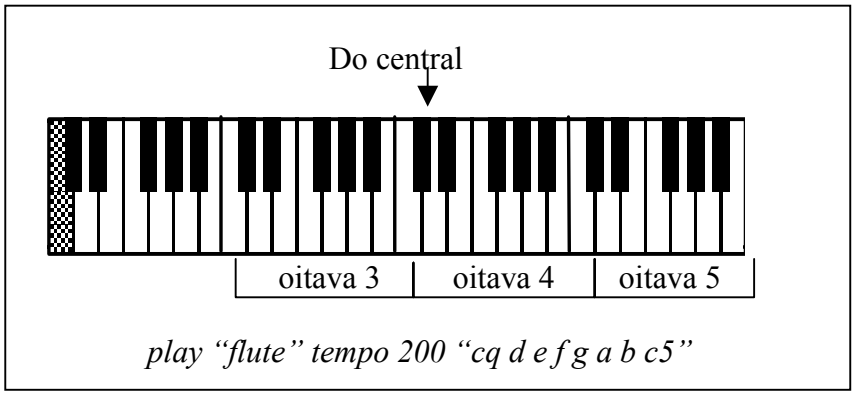


Figura 302 – Identificando as oitavas do teclado em HyperCard

No exemplo 3 você pode programar melodias para serem tocadas pelo alto-falante do Macintosh através do comando Play.

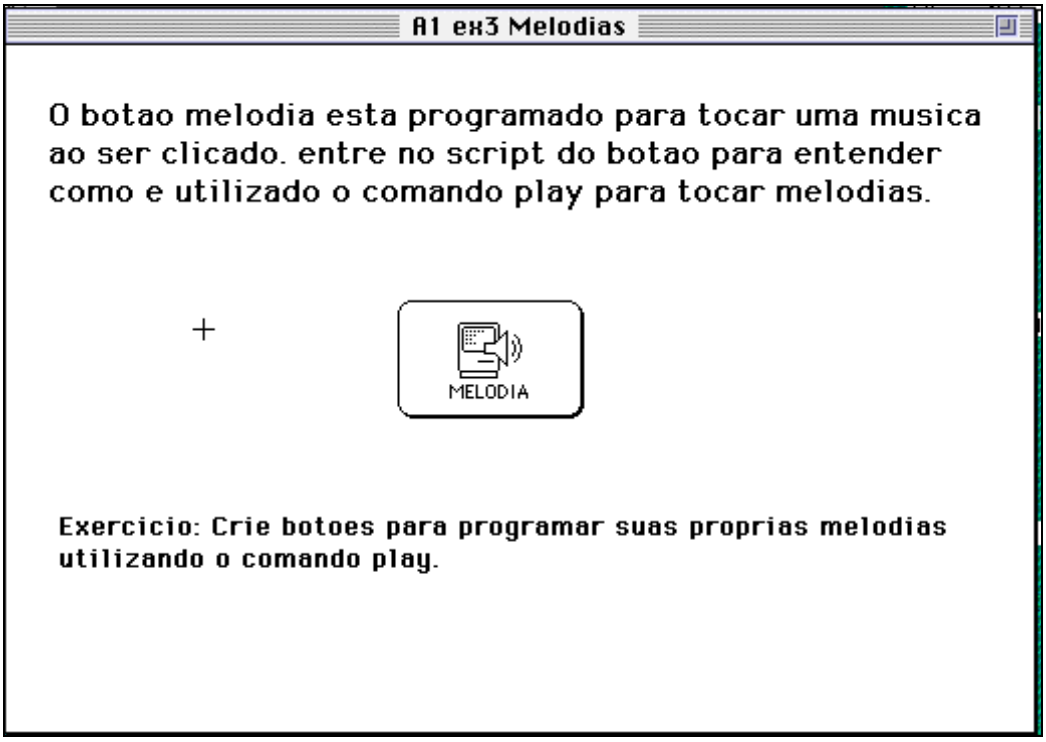


Figura 303 - Utilização do comando *play* para tocar melodias

Script do botao Melodia

```
on mouseUp
  -- As notas sao colocadas em forma de lista separadas por espacos.
  play "flute" tempo 120 "gq re de gq re de ge de ge be d5q rq cq
                          re a4e c5q re a4e c5e a4e f#e a4e dq rq"
end mouseUp
```

8.2.3 AULA 2 – Comandos Básicos de Programação no HyperCard

Nesta aula serão apresentados conceitos de variáveis e comandos do HyperCard que serão empregados nos próximos exercícios.

Variáveis são espaços de memória do computador que são alocados para armazenar informações.

Variável Global (***global)*** é aquela cujo o valor será utilizado por todo o programa, ou seja, em qualquer parte do código(toda a pilha) poderemos fazer referência a ela para sabermos o seu conteúdo. Para a variável ser global, deve ser declarada em todos os scripts de cartões, conforme o exemplo abaixo.

Exemplo de declaração de variável global:
Global var1

A variável local é aquela cujo o valor só será válido dentro do mesmo botão ou campo do cartão e não deve ser declarada como global.

O ***if*** é um comando que serve para direcionar o fluxo dos dados nos scripts, através de testes e condições. Essa estrutura poderá evoluir caso o comando seja verdadeiro, o que o fará executar o teste. Em caso de falso, o teste será ignorado. *If...(*condição) *then*....(faça x) *else*.... (faça y)

O comando ***answer*** é utilizado para imprimirmos algo na tela, como alguma mensagem de retorno ao usuário (ex: "você acertou..")

O comando ***put*** é um comando de atribuição, que utilizamos para atribuir um valor a uma determinada variável.

Exemplo:
Put var1 into var2
Put (var1+2) into var2
Put 3 into var1

8.2.3.1 Criando um programa de percepção musical

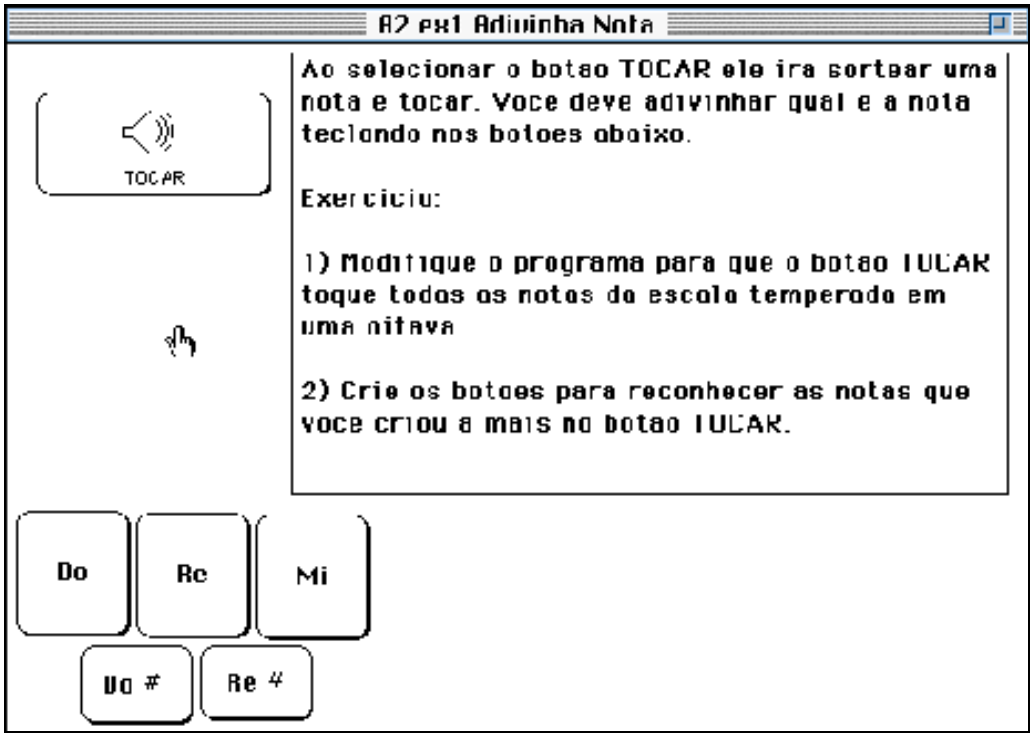


Figura 304 – Programa de percepção musical em HyperCard

Script do Botao Tocar

```
on mouseUp
  global nota
  put the random of 13 into nota
  if nota = 1 then
    play "flute" tempo 100 "c4e"
  end if
  if nota = 2 then
    play "flute" tempo 100 "c#4e"
  end if
  if nota = 3 then
    play "flute" tempo 100 "d4e"
  end if
  if nota = 4 then
    play "flute" tempo 100 "d#4e"
  end if
  if nota = 5 then
    play "flute" tempo 100 "e4e"
  end if
  if nota = 6 then
    play "flute" tempo 100 "f4e"
  end if
  if nota = 7 then
    play "flute" tempo 100 "f#4e"
  end if
```

```
  if nota = 8 then
    play "flute" tempo 100 "g4e"
  end if
  if nota = 9 then
    play "flute" tempo 100 "g#4e"
  end if
  if nota = 10 then
    play "flute" tempo 100 "a4e"
  end if
  if nota = 11 then
    play "flute" tempo 100 "a#4e"
  end if
  if nota = 12 then
    play "flute" tempo 100 "b4e"
  end if
  if nota = 13 then
    play "flute" tempo 100 "c5e"
  end if
end mouseUp
```

Script do botao Do

```
on mouseUp
  global nota
  if nota = 1 then
    answer "acerto"
  else
    answer "tente novamente"
  end if
end mouseUp
```

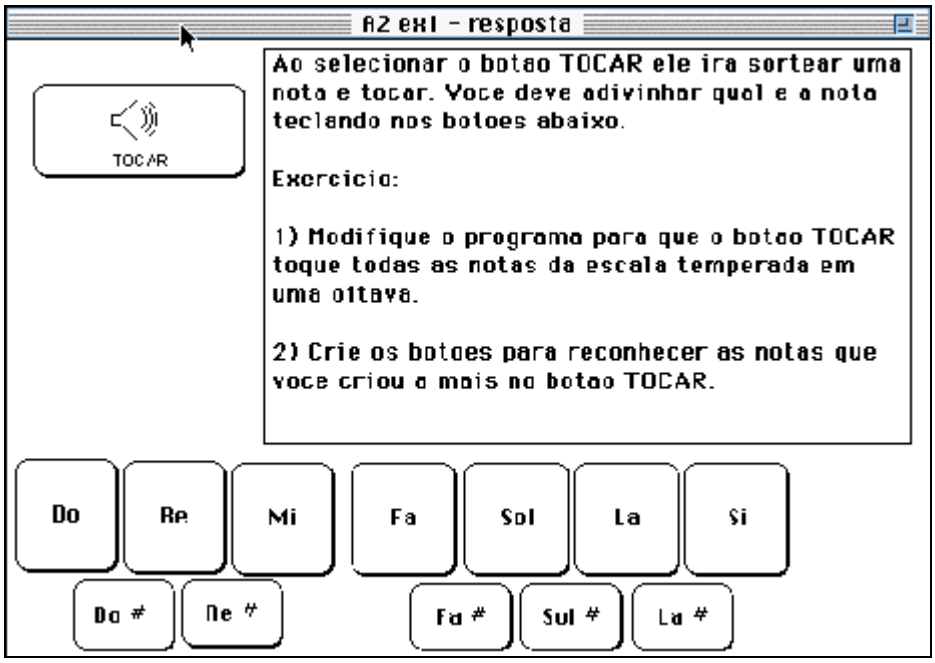


Figura 305 – Resposta do Exercício Toca Notas

8.2.3.2 Modificando andamentos de melodias

Nesse exemplo serão apresentados os comandos *show* e *hide*, que são usados para "apresentar" e "ocultar", respectivamente, botões e caixas de texto na tela, quando executamos algum evento como o clicar do mouse, por exemplo.

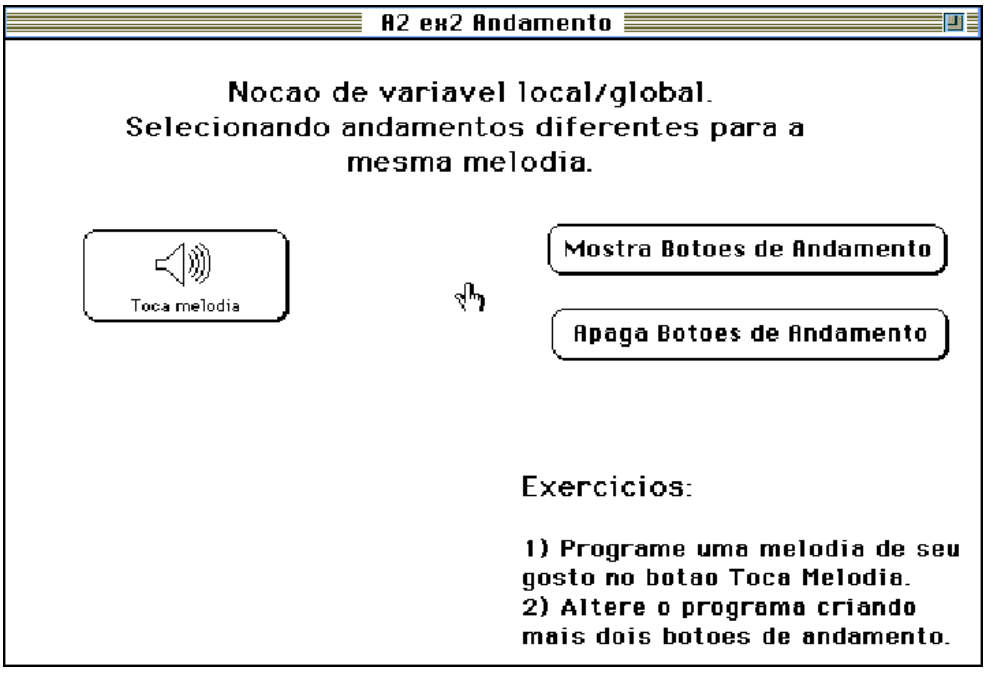


Figura 306 – Tabalhando com variáveis

Script TOCA MELODIA
on mouseUp

-- Define a variavel como global
-- Ou seja, o valor sera passado para todos os outros cartoes

```
  -- botoes da mesma pilha.
  global andamento

  -- Toca o comando play utilizando o valor da variavel andamento
  -- Provindo da selecao dos botoes do cartao
  play "flute" tempo andamento "a4 b c d"
end mouseUp
```

Script MOSTRA BOTOES DE ANDAMENTO

```
on mouseUp

  -- mostra os botoes
  show button "Andamento Lento"
  show button "Andamento Medio"
  show button "Andamento Rapido"
end mouseUp
```

Script APAGA BOTOES DE ANDAMENTO

```
on mouseUp

  -- apaga os cartoes da tela
  Hide Button "Andamento Lento"
  Hide Button "Andamento Medio"
  Hide Button "Andamento Rapido"
end mouseUp
```

O comando **repeat while** é utilizado para executar uma instrução ou um conjunto de instruções repetidamente dependendo de uma condição determinada pelo programador. Podem ser chamadas de laço ou loop. (faça enquanto a condição for verdadeira...)

O comando **put into** é um comando de atribuição que serve para atribuirmos um valor a uma variável. Ex. put 100 into a (estamos atribuindo o valor "100" a variável "a")

Repeat forma o que chamamos de estrutura de repetição, podendo ser acompanhado de outras "palavras reservadas" que executarão, cada uma, uma forma diferente de repetição. **Repeat,** simplesmente, faz um laço (ou loop) infinito até que uma condição if ... then ... seja encontrada. Ex repeat <comando> <if ...algo diferente then exit repeat> end repeat.

Já o **repeat for** repete o comando programado um número fixo de vezes. Ex repeat for <numero de vezes> <comando que ser quer> end repeat.

Repeat until executa os comandos programados até que uma expressão retorne um valor verdadeiro. É comumente usado quando se espera pelo clic do mouse, por exemplo. Ex repeat util <clic do mouse> <toque várias notas> end repeat.

O comando **repeat while** é essencialmente o oposto do comando repeat until**,** ou seja, executa os comandos enquanto a expressão for verdadeira.

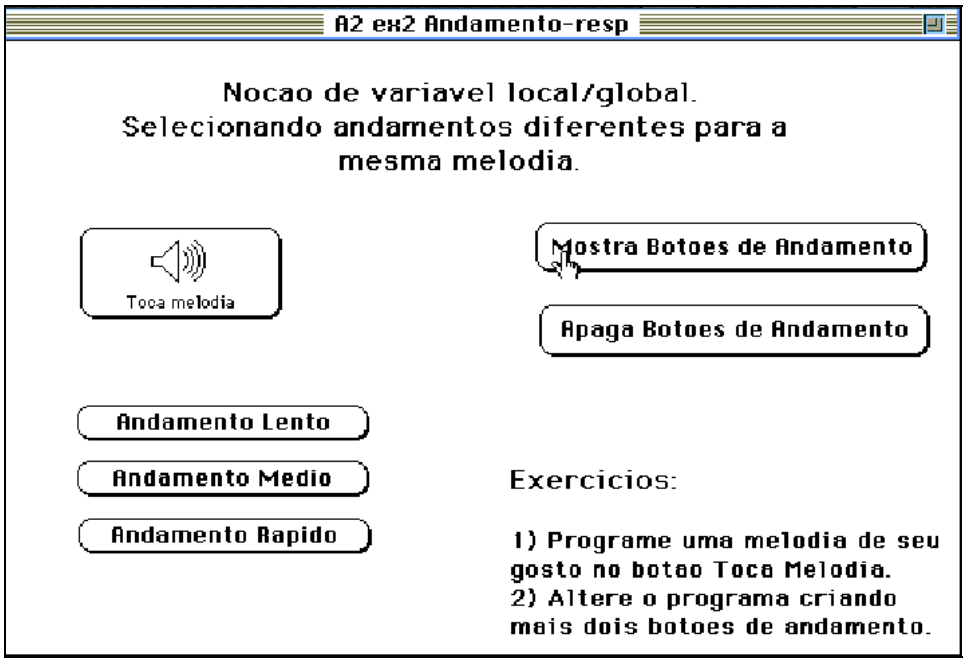


Figura 307 – Resposta do exercício sobre andamento

Script ANDAMENTO LENTO

```
on mouseUp

  -- define a variavel andamento como global
  global andamento

  -- armazena 50 na variavel andamento
  put 50 into andamento

  -- toca a nota Do
  play "boing" tempo 200 "c5"

end mouseUp
```

Script ANDAMENTO MEDIO

```
on mouseUp
  global andamento
  put 150 into andamento
  play "boing" tempo 200 "c6"
end mouseUp
```

Script ANDAMENTO RAPIDO

```
on mouseUp
  global andamento
  put 300 into andamento
  play "boing" tempo 200 "c7"
end mouseUp
```

8.2.4 AULA 3 – Utilizando MIDI para programar software educacional

O HyperMIDI é a pilha MIDI feita pela EarLevel Engeneering e distribuída para ser usada em combinação com HyperCard para que esse possa processar e enviar mensagens MIDI. Para apresentar exemplos de utilização de programas educacionais que podem ser utilizados em conjunto com instrumentos musicais conectados ao computador utilizaremos o HyperMIDI.

8.2.4.1 HYPERMIDI

O HyperMIDI apresenta comandos que devem ser utilizados nos scripts dos botões dos programas MIDI. Iniciamos esse estudo apresentando convenções de sintaxe para a o emprego correto dos comandos.

8.2.4.2 Convenções de sintaxe

De um modo geral, não importa se você usa letras maiúsculas ou minúsculas para os nomes de comandos e funções ou seus parâmetros; tais esquemas de diferenciação são usados apenas por razões de clareza. Por exemplo, hmReadMIDI e hmreadmidi são equivalentes. A tabela 1 apresenta algumas convenções usadas ao longo dos capítulos em relação à sintaxe dos comandos e das funções.

8.2.4.3 ABRINDO E FECHANDO O AMBIENTE MIDI

O ambiente MIDI pode ser inciado e encerrado através dos comandos:

- hmOpenMIDI
- hmCloseMIDI

Estes comandos são essenciais. Use hmOpenMIDI para abrir e inicializar o HyperMIDI e hmCloseMIDI para encerrar a sessão de trabalho.

hmOpenMIDI Comando

Finalidade Inicializar o HyperMIDI, criar portas e fluxos

Descrição O primeiro formato de hmOpenMIDI mostrado acima instala o controlador do HyperMIDI, cria as portas e fluxos de entrada e saída e inicializa todos os modificadores de fluxo. Use este comando antes de chamar quaisquer outras rotinas de HyperMIDI.

Erros Error: MIDI Manager not available! (*MIDI Manager não disponível!)*
Error: bad input spec *(especificação de entrada inválida)*
Error: bad output spec *(especificação de saída inválida)*
Error: not enough memory *(sem memória suficiente)*

Exemplos Inicializa o HyperMIDI com:
1 fluxo de entrada (única porta de entrada) de 4000 bytes.
1 fluxo de saída (única porta de saída) de 1000 bytes

```
hmOpenMIDI 4000,1000
```

hmCloseMIDI Comando

Finalidade Fechar o HyperMIDI
Descrição Este comando "remove" o HyperMIDI do *MIDI Manager* e libera toda a memória usada pelo HyperMIDI. Já que o HyperMIDI ocupa espaço na partição de memória do HyperCard (sob o MultiFinder), deve-se executar este comando antes de fechar o HyperCard.
Erros nenhum
Exemplo HmCloseMIDI

8.2.4.4 LENDO E ESCREVENDO EM MIDI

Para ler e escrever informações MIDI são utilizandos os comandos:

- hmReadMIDI
- hmWriteMIDI

Estas rotinas lidam com entrada e saída de dados MIDI com controle de duração.

HmReadMIDI Função

Finalidade Ler dados MIDI a partir de um fluxo de entrada
Descrição Use esta função para ler mensagens MIDI a partir do *buffer* de dados de um fluxo de entrada.
Erros Error: HyperMIDI driver is not installed *(o* driver *do HyperMIDI não está instalado)*

Exemplos Lê todos os bytes MIDI disponíveis a partir do fluxo 1 e atribui à um campo. put hmReadMIDI () into card field "mostra"

HmWriteMIDI Comando

Finalidade Envia dados MIDI com os tempos de execução para um fluxo de saída
Descrição Use este comando para a saída de dados MIDI.
Erros Error: HyperMIDI driver is not installed *(o* driver *do HyperMIDI não está instalado)*
Error: bad output stream *(fluxo de saída inválido)*
Error: bad data byte count *(problemas na contagem de bytes)*
Error: user abort *(o usuário provocou o encerramento da operação)*
Warning: bad SysEx termination (EOX added) *(terminação da Mensagem Exclusiva inválida (EOX adicionado))*
Exemplo Envia nota-ativada, seguida por nota-desativada 1000ms depois hmWriteMIDI "144 60 64 1000:144 60 0"

8.2.4.5 Programando MIDI no HyperCard

Antes de chamar quaisquer outras rotinas de HyperMIDI, você deve inicializá-lo chamando hmOpenMIDI. Ponha hmOpenMIDI no script da pilha após o comando *openStack*, para que ele seja executado sempre que sua pilha for aberta. Escolha a opção *Stack Info* do menu *Objects* e pressione o botão *Script* para editar o *script* da pilha.

Figura 308 – Usando comandos MIDI

Script TOCAR

```
on mouseUp

  -- Se o botao " Liga MIDI"  estiver ligado entao
  If the hilite of button "Liga MIDI" is true then

    -- Envia um comando para a interface MIDI
    -- e toca a nota Do no canal MIDI 1 com a velocidade de 127
    -- e a duracao de 500 milisegundos.
    -- Depois desliga a nota Do (velocidade = 0)
    hmWriteMIDI "144 60 127 500:144 60 0"

    -- Senao
  else

    -- Toca a nota Do atraves do auto-falante do Macintosh
    -- com o andamento de 120 utilizando o instrumento Cravo
    play "harpsichord" tempo 120 "cq5"

    -- Fim do if
  end if
```

```
  -- Fim do Programa
end mouseUp
```

Script LIGA MIDI

```
on mouseUp

  -- Liga o botao - verdadeiro
  set the hilite of card button "Liga MIDI" to true

  -- Desliga o botao - falso
  set the hilite of card button "Desliga MIDI" to false

end mouseUp
```

Script DESLIGA MIDI

```
on mouseUp

  -- Comando que desliga o botao - falso
  set the hilite of button "Liga MIDI" to false

  -- Comando que liga o botao - verdadeiro
  set the hilite of button "Desliga MIDI" to true

end mouseUp
```

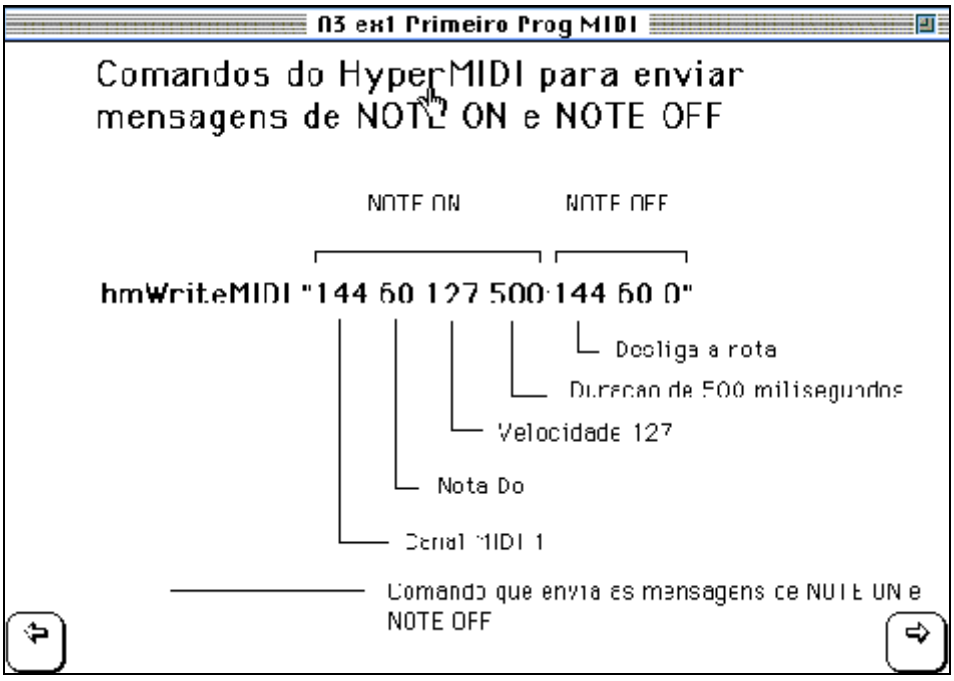


Figura 309 – Liga/desliga notas

Figura 310 - Criando percussão

Script BUMBO

On mouse up

HmWriteMIDI " 153 36 127 100:153 36 0"

End mouse up

8.2.4.6 Programando Geração aletória de alturas

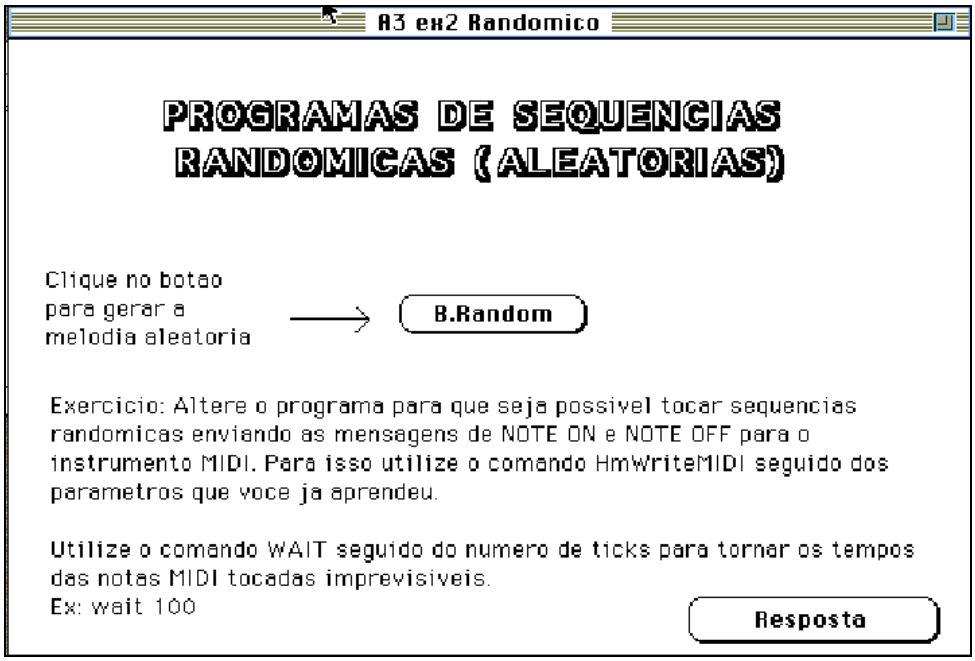


Figura 311 – Usando o comando Random

Script RANDOM

on mouseUp

-- Repita 10 vezes - Ira tocar 10 notas numa sequencia

```
repeat 10 times

  -- Armazene o valor 100 na variavel andamento
  put the random of 7 into andamento

  -- multiplica o valor de andamento por 10 e armazena
  -- o resultado da multiplicacao em andamento
  -- a variavel andamento sera usada no comando play
  multiply andamento by 10

  -- seleciona as notas que serao tocadas
  -- escolhe um numero de 1 a 12 e coloca ele na variavel note
  put the random of 12 into note

  -- Toca as notas atraves do auto-falante do Macintosh
  -- com o andamento de fornecido pelo valor da variavel andamento
  -- obtido do numero randomico multiplicado por 10
  -- utilizando o instrumento Flauta

  -- se o numero da variavel note for 1 faca:
  if note = 1 then
    play "flute" tempo andamento "c5e"
  end if
  if note = 2 then
    play "flute" tempo andamento "c#5e"
  end if
  if note = 3 then
    play "flute" tempo andamento "d5e"
  end if
  if note = 4 then
    play "flute" tempo andamento "d#5e"
  end if
  if note = 5 then
    play "flute" tempo andamento "e5e"
  end if
  if note = 6 then
    play "flute" tempo andamento "f5e"
  end if
  if note = 7 then
    play "flute" tempo andamento "f#5e"
  end if
  if note = 8 then
    play "flute" tempo andamento "g5e"
  end if
  if note = 9 then
    play "flute" tempo andamento "g#5e"
  end if
  if note = 10 then
    play "flute" tempo andamento "a5e"
  end if
```

```
   if note = 11 then
    play "flute" tempo andamento "a#5e"
   end if
   if note = 12 then
    play "flute" tempo andamento "b5e"
   end if

  end repeat
end mouseUp
```

Script RESPOSTA

```
on mouseUp
  go to card "Resposta"
end mouseUp
```

Programando repetições

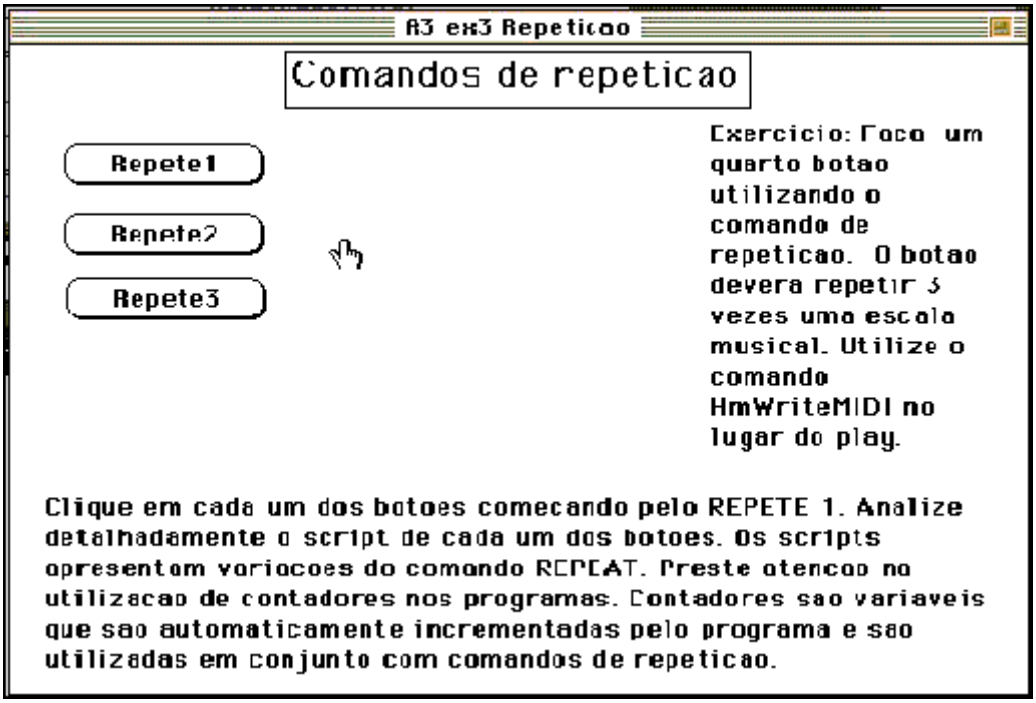


Figura 312 – Comando de repetição em HyperMIDI

Script REPETE 1

```
on mouseUp
  -- Define a variavel cont como global
  global cont

  -- Armazena 100 na variavel cont
  put 100 into cont
  -- repete os comandos ate que cont seja menor que 300
  repeat while cont < 300

    -- Toca a nota Do com o andamento dado por cont
```

```
    play "flute" tempo cont "c4e"

    -- Incrementa o cont de 5 em 5 cada vez que acontece
    -- uma repeticao
    put cont + 5 into cont

  end repeat
end mouseUp
```

Script REPETE 2

```
on mouseUp

  -- Escreve a pergunta na tela do computador e aguarda que
  -- o usuario responda com um numero
  ask "Quantas vezes voce quer repetir?"

  -- Armazena a resposta na variavel cont
  put it into cont

  -- Repete os comandos o numero de vezes escolhido pelo usuario
   repeat for it

    -- multiplica o valor da variavel cont por 2
    -- soma 20 ao resultado e armazena em cont
    put (cont * 2) + 20 into cont

    -- Toca 3 notas seguidas com o andamento dado pela variavel cont
    play "flute" tempo cont "c4e"
    play "flute" tempo cont "e4e"
    play "flute" tempo cont "g4e"
  end repeat
end mouseUp
```

Script REPETE 3

```
on mouseUp

  -- Mostra o campo Texto na tela
  show card field "Texto"

  -- Armazena 100 na variavel cont
  put 100 into cont

  -- Repete os comandos ate que seja pressionada uma tecla
  repeat until the mouseclick

    -- Incrementa a variavel cont de 10 em 10
    put cont +10 into cont
```

```
  -- Toca tres notas consecutivas no andamento dado pela variavel cont
  play "flute" tempo cont "c4e"
  play "flute" tempo cont "e4e"
  play "flute" tempo cont "g4e"
end repeat
  -- Apaga o campo Texto
hide card field "Texto"
end mouseUp
```

8.2.5 AULA 4 – Programando um sequenciador

Neste exercício aprederemos como armazenar informações MIDI que são tocadas no teclado externo. Para isso siga corretamente as instruções logo a seguir da figura 313 e tente escrever o scripts dos botões para que funcionem conforme indicado.

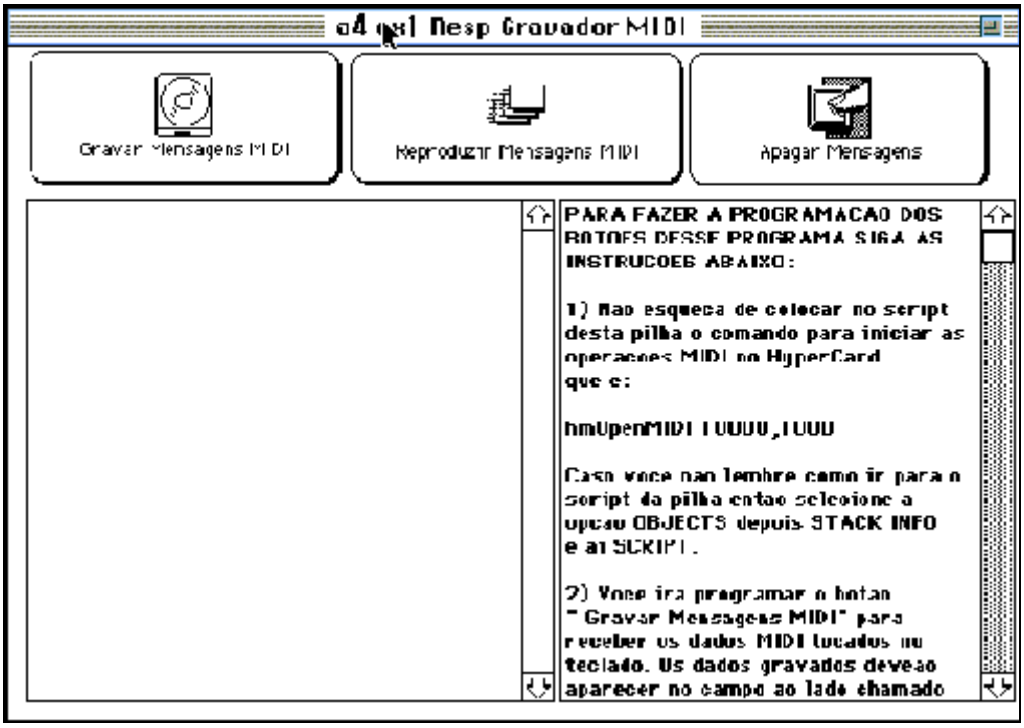


Figura 313 – Gravando mensagens MIDI

8.2.5.1 Gravando Mensagens MIDI

Toque algumas notas em seu teclado MIDI. Quaisquer notas, alavancas *pitch bend*, mudanças de *patch, aftertouch* e tudo o mais serão automaticamente armazenadas no *buffer* do fluxo de entrada. Não provoque uma sobrecarga de dados por enquanto; foi criado um pequeno *buffer* de entrada, suficiente para apenas 50 notas, aproximadamente. Se informações demais forem entradas sem terem sido lidas, você sobrecarregará o *buffer* de entrada e perderá alguns dados. Nenhum dano será feito, mas você estará perdendo algumas notas que você tocou.

Bem, se tudo correu bem, os dados MIDI estão acomodando-se em seu *buffer* de entrada. Agora vamos colocá-los onde possamos olhar para eles: crie um novo campo do cartão, escolhendo a opção *New Field* no menu *Objects*. Dê um duplo clique no contorno do campo (ou escolha a opção *Field Info* no menu *Objects*) para abrir a caixa de diálogo *field*. Dê um nome ao campo *MostraMIDI* e ajuste o estilo do campo para *"scrolling"*, pressionando em seguida OK para confirmar as modificações. Redimensione o campo a fim de torná-lo maior, pois ele será usado tanto para armazenar como para exibir seus dados MIDI.

Escolha agora a opção *New Button* no menu *Objects* e abra a caixa de diálogo *button* dando um duplo clique sobre o botão. Nomeie-o como "Captura", selecione a caixa de verificação "*Auto hilite*", edite o *script* do botão clicando em *Script* na caixa de

diálogo e entre com o seguinte *script* (o HyperCard já "digitou" a primeira e última linhas para você):

```
on mouseUp
        put hmReadMIDI () into card field "MostraMIDI"
end mouseUp
```

Escolha OK para confirmar as mudanças no *script* e então clique na ferramenta *Browse* do menu *Tools*. Clique no seu botão **uma só vez** para executar o *script*. Você deverá ver uma seqüência de números aparecer no seu mostrador: esta é a série de dados MIDI com etiquetas de tempo que você tocou há algum tempo atrás, que estiveram aguardando pacientemente enquanto você esteve trabalhando em sua pilha. Ao usar hmReadMIDI sem parâmetros você está solicitando todos os dados atualmente alocados no *buffer* do fluxo de saída 1. Esta declaração lê os dados do *buffer* e os aloca (em base decimal) no campo do cartão. O *buffer* de entrada deverá estar agora vazio; à medida em que você lê dados de um fluxo de entrada, você os remove do *buffer*.

8.2.5.2 Reproduzindo Mensagens MIDI

Crie outro botão, semelhante ao Capture, mas nomeie-o como "Play". Escreva o seguinte no *script* do botão:

```
on mouseUp
        hmWriteMIDI card field "MostraMIDI"
end mouseUp
```

Escolha a ferramenta *Browse* e clique em seu novo botão. Você deverá ouvir as mesmas notas, na mesma ordem e andamento que você as tocou anteriormente.

Adicionemos agora o comando hmCloseMIDI para remover o HyperMIDI quando a pilha for fechada. Este comando é o mais simples de todos, já que ele não tem parâmetros. Basta colocá-lo em um manipulador closeStack no *script* da pilha:

```
on closeStack
        hmCloseMIDI
end closeStack
```

→ *Importante:* É essencial que você chame hmCloseMIDI antes de sair do HyperCard. O *MIDI Manager* ficará surpreso ao não encontrar ninguém em casa, se você não disser adeus antes.

Script APAGA MENSAGENS

```
On mouseUp
        Put empty into card field "MostraMIDI"
End mouseup
```

O HyperMIDI disponibiliza vários comandos para a manipulação de eventos MIDI. Abaixo temos um exemplo de programa que grava e transpõe eventos MIDI.

8.2.5.3 Programando a função de transposição no sequenciador

Neste último exercício do nosso curso de Programação de Computadores para a música apresentamos um exemplo de como modificar dados MIDI através do HyperMIDI. O programa é muito parecido com o GravadorMIDI, no entanto, o exercício 2 solicita que você faça uma função de transposição utilizando o comando HmSetTranspose. Depois de escrever o programa, para que possa conferir os comandos, listamos os scripts dos botões que podem ser encontrados logo a seguir, após a figura 314.

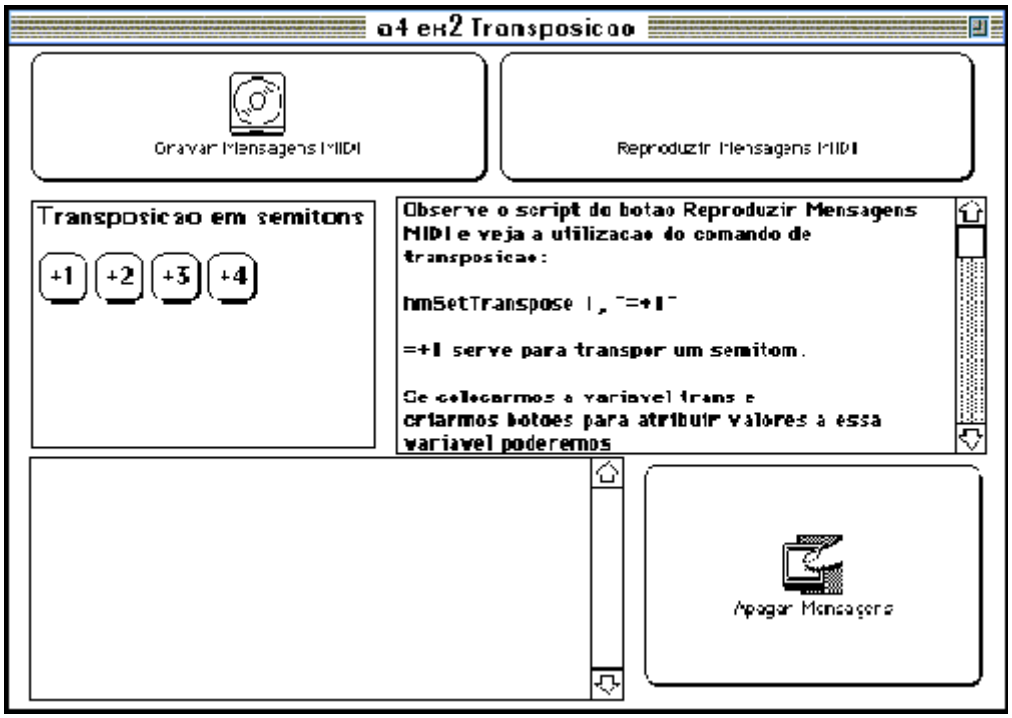


Figura 314 – Comando de transposição de tom

Script GRAVAR MSG MIDI

```
on mouseUp
  global trans
  put 0 into trans
  put HmReadMIDI () into card field "mostra"
  hmSetFilter 1, "block all, pass noteOn, pass noteOff"

end mouseUp
```

Script REPRODUZIR MSG MIDI

```
on mouseUp

  global trans

  if trans > 0 then

    -- Se a transposicao for maior que zero e porque um dos botoes
```

```
    -- de transposicao foi selecionado

    hmSetTranspose 1,"=+" && trans
    hmWriteMIDI card field "mostra"

  else

    -- Caso os botoes de transposicao nao tenham sido selecionados
    -- apenas toca a sequencia

    hmWriteMIDI card field "mostra"

  end if

end mouseUp
```

Script TRANSPOSICAO DE SEMITONS

Script BOTÃO +1

```
on mouseUp
  global trans
  put 1 into trans
end mouseUp
```

Script BOTÃO +2

```
on mouseUp
  global trans
  put 2 into trans
end mouseUp
```

Script BOTÃO +3

```
on mouseUp
  global trans
  put 3 into trans
end mouseUp
```

Este curso apresentou uma introdução à programação de computadores para a música utilizando o método MEPSOM – Método de Ensino de Programação de Computadores para Músicos. O conjunto de informações e comandos ensinados neste curso foram escolhidos criteriosamente para que servissem de alicerce para o aprendizado de outros novos comandos das linguagens MAX e HyperCard/HyperMIDI. Logo, estes são apenas os primeiros passos de uma grande jornada que você, músico programador, esta iniciando. A quantidade e variedade de programas de computador que você, músico criativo, irá realizar, dependerá apenas de você. Sua vontade e necessidade de continuar a programar em MAX e HyperMIDI serão os principais fatores de desenvolvimento dos conhecimentos obtidos neste curso.

De nada adianta aprender linguagens de programação para a música, se estas, depois, não forem utilizadas para criar programas de acordo com seus interesses musicais. Não esqueça, o computador é um laboratório de experiências musicais. Portanto, não deixe de pesquisar e utilizar sua criatividade musical através dele.

8.3 Conclusões e Trabalhos Futuros

Nesta Tese de Doutorado apresentamos o MEPSOM e os primeiros resultados da sua aplicação em cursos de Programação de Computadores para a Música.

O MEPSOM possui diferenças em relação às obras existentes sobre o assunto. Em [WIP 89] e [MES 98] os autores ensinam programação de computadores para músicos através de linguagens declarativas enquanto o MEPSOM utiliza linguagens visuais. As obras de [ZIC 88], [DOB 98], [RED 88] e [GOO 90] não dispõem, individualmente, de um método de ensino através de exemplos com a abrangência de conteúdos que o MEPSOM disponibiliza. As obras de [MES 98] e [CON 88] são dirigidas à usuários com experiência em programação de computadores enquanto que o MEPSOM foi concebido para ensinar músicos sem experiência em programação. Em [WIN 98] é empregado um método similar ao MEPSOM, porém o autor não abrange o aspecto de síntese sonora limitando-se a exemplos com programação MIDI.

Esta Tese demonstra que é possível melhorar o ensino de computação musical no Brasil. As obras de [DOD 97], de [CON 88], de [MES 98] e de [WIP 89] apresentam algoritmos musicais programados em linguagens de difícil entendimento pelo músico, conforme demonstrado no item *3.1.4, Linguagens de Programação Visual*. As experiências relatadas no item *4.4.4.2, Questionários Abertos,* levam-nos a crer que o emprego de linguagens visuais e sonoras contribui para que o músico entenda com maior facilidade o processo de programação sônica de computadores. As obras de [MES 98], de [DOB 98], [DOD 97] e [COM 88] procuram ensinar apenas parte do conteúdo proposto no MEPSOM, não disponibilizam um método de ensino baseado em exemplos para iniciantes e são voltadas para alunos norte-americanos, pois empregam exclusivamente expressões técnicas e musicais em inglês. Portanto, não são totalmente adequadas às situações de ensino de programação de computadores para músicos no Brasil. O emprego do MEPSOM, conforme apresentado no item *4.4.4.1, Questionários Fechados*, apresentou resultados positivos encorajando a sua utilização em cursos de computação musical. As avaliações realizadas demonstraram a necessidade do emprego do método de ensino de programação para introduzir o músico na programação e servir de suporte para a continuidade de seu aprendizado. Se, por um lado, o emprego de exemplos simples, linguagens visuais interpretadas e textos explicativos em português ainda deram margem à ocorrência de resistências à prática de programação, por outro lado, o emprego de linguagens baseadas em linha de comando, compiladas, com explicações técnicas e musicais na língua inglesa, tendem a ser inadequadas para iniciantes. Contudo, os resultados obtidos não desqualificam as obras citadas. Eles apenas apontam para a necessidade da utilização de um método mais adequado às necessidades de ensino nas Universidades brasileiras.

De acordo com a proposta inicial do MEPSOM, processos complexos de geração musical por computador não foram empregados, por criarem situações de difícil solução computacional para o músico. Fazem parte dessa categoria de algoritmos complexos: fractais, redes neurais artificiais, autômatos celulares e gramáticas[17]. Se por um lado a ausência desses procedimentos resulta numa limitação do MEPSOM, por outro lado resulta num método acessível, claro e realizável dentro de um período estimado em dois meses. Outro fator que colabora para a não-utilização de algoritmos complexos nos exemplos do MEPSOM é a necessidade de um embasamento sólido em

[17] Várias técnicas para composição automática por computador são detalhadas nas obras de [ROA 96], [DOD 97] e [MIR 01].

matemática. Ao imaginar uma situação em que o músico necessita aprender integração e derivação matemática para após traduzir esse conhecimento em programação, fez-nos refletir sobre algumas dificuldades desnecessárias para o processo de aprendizagem de programação sônica de computadores. Logo, a utilização de algoritmos que exijam muito conhecimento matemático e muita experiência em programação foge ao escopo desse trabalho. O MEPSOM simplifica a tarefa de programação através de exemplos e exercícios que possam ser compreendidos pelo músico. Portanto, o MEPSOM evita, sempre que possível, algoritmos complexos que possam ser exemplos desaconselháveis para o aprendizado de programação sônica de computadores.

Outro fator limitante desta pesquisa está no fato de que a grande maioria dos algoritmos presentes no método são destinados à geração automática e interativa de material musical para a composição. O método não aborda a problemática da construção da forma musical através do auxílio do computador. Esse problema poderia ser resolvido em trabalhos futuros em que novos algoritmos, visando à construção de formas musicais, possibilitassem a organização e seqüenciamento do material musical produzido. Sendo assim, cabe ao professor e ao aluno adequarem o material musical, de acordo com suas necessidades.

Do ponto de vista da área de educação musical, o MESPCM contribui com exemplos que irão abrir possibilidades para a confecção de programas nas áreas de literatura musical, teoria e percepção, permitindo que o educador desenvolva programas para seus próprios alunos. Principalmente a criatividade e a experiência do educador farão com que ele transforme suas idéias em programas musicais. Os exemplos do método fornecidos em HyperCard/HyperMIDI envolvem a solução de problemas específicos e básicos da música. No entanto, outros trabalhos desenvolvidos por esse autor, como o SETMUS [FRI 95], o STI [FRI 96] [FLO 2000a], o SETMUS e o STR [FRI 98] podem ser utilizados como exemplos de soluções para problemas mais aprofundados na área da teoria e percepção musical[18].

Uma das contribuições deste trabalho consiste no emprego de linguagens visuais para a música. Algoritmos e instruções, antes disponíveis em linguagens procedurais, foram apresentadas aos músicos sob a forma visual, facilitando seu entendimento. Logo, a contribuição desta pesquisa está na disponibilização de um método de ensino que enfatiza a interface sonora e visual, possibilitando que o compositor aprenda a realizar novos processos criativos de composição e performance através da programação sônica de computadores.

A implementação do MEPSOM foi possível de ser realizada utilizando-se duas linguagens de programação para a música. A Linguagem *HyperCard/HyperMIDI* mostrou dispor de mais ferramentas para a construção de programas do Módulo de Programação de Aplicações na Área de Educação Musical. Por outro lado, a Linguagem MaxMsp mostrou ser adequada para o desenvolvimento do Módulo de Programação de Aplicações na área de Composição Musical.

O emprego da tecnologia de comunicação entre computadores e sintetizadores, nesta etapa da pesquisa, proporcionou a utilização de equipamentos simples e com pouca capacidade de memória e processamento. Essa tecnologia pode ser utilizada com maior facilidade por escolas de música que dispõem de poucos recursos por apresentar baixo custo e dispensar computadores sofisticados, o que não acontece com a tecnologia de audiodigital, cujo o custo se eleva sobremaneira, em função dos requisitos de *hardware* e *software*. Os programas que utilizam DSP estão sendo feitos apenas em

[18] Os programas SETMUS, STI e STR possuem o código aberto encorajando um estudo mais aprofundado.

MaxMsp devido às restrições que o *HyperCard* possui com relação às rotinas para manipulação de áudio digital.

A utilização do método proporcionou resultados significativos ao ser avaliado pelos estudantes. O emprego do MEPSOM em situações reais de ensino de programação para músicos resultou em um refinamento dos programas, obtido através do *feedback* das aulas, fornecendo elementos para o aprimoramento dos exemplos e atividades que constituem o método.

No futuro este método poderá ser adaptado para aplicações via Web através de novas tecnologias que visam à utilização de música na Internet [FLO 2000]; [HEL 96]. As linguagens jMax e Java Sound poderão ser utilizadas para implementar o método, com o objetivo de possibilitar a independência de plataforma e utilização em educação a distância. Com o aumento na taxa de transmissão da internet, será possível um músico utilizar o MEPSOM remotamente. A transmissão de mensagens MIDI é rápida, porém a barreira da síntese em tempo real pela internet utilizando transmissão de audiodigital ainda precisa ser ultrapassada. Isso possibilitará o emprego do método em cursos que têm por objetivo o ensino a distância. Possivelmente, apenas parte dos exercícios poderão ser implementados para a Web num primeiro momento. Porém, esses primeiros programas nos mostrarão o caminho para a utilização de recursos alternativos na internet. Apesar de o método não ter sido desenvolvido para a internet, devido às inúmeras restrições na transmissão de grandes quantidades de som, ele pode ser adaptado, e seus exemplos e exercícios reprogramados para esse fim.

Quando iniciamos os primeiros testes com protótipos do MEPSOM, utilizávamos computadores e programas do início da década de noventa. No entanto, a evolução tecnológica é muito veloz. Os programas mais recentes desenvolvidos para o método foram feitos em computadores de última geração. Deve ser levado em conta que a diferença na performance computacional pode influir no processo de aprendizagem do aluno. Portanto, sempre haverá adaptações a serem feitas, dependendo da configuração e dos recursos dos equipamentos utilizados na criação e utilização dos programas do método.

Apesar de esta Tese de Doutorado enfocar o aspecto tecnológico-computacional, não podemos nos esquecer de que o aluno é a peça fundamental para o aprendizado eficiente, sendo ele, no papel de usuário, quem necessita de recursos compatíveis com seus desejos e necessidades. A composição de música por computador não deve ser um privilégio de poucos, mas, sim, de toda uma comunidade sintonizada com os progressos nesta área. Acreditamos que o ensino através de exemplos possa ser um ótimo recurso quando auxiliado pelas facilidades do uso do computador. O papel social desta pesquisa está em disponibilizar um conteúdo que atualmente ainda é de difícil acesso, até mesmo para cursos de música no Brasil. Além de disponibilizar o conteúdo, também propiciar a prática do mesmo através de um estudo auxiliado pelo computador. Atualmente são necessárias máquinas onerosas para a execução do método, devido, principalmente, ao Módulo Programação com Audiodigital. Todavia, em alguns anos, esse equipamento será mais acessível e poderá ser adquirido por escolas de música que não disponham de grandes recursos financeiros. Com algumas adaptações, é possível utilizar o MEPSOM em partes[19].

Cada vez mais existe uma tendência na utilização de instrumentos e equipamentos eletrônicos, tais como, sintetizadores, gravadores digitais e estações de

[19] Essa experiência já foi realizada durante os primeiros cursos ministrados para avaliar o Nível Preparatório do método no Programa de Extensão em Música Eletrônica da UFRGS.

trabalho musicais. Essa categoria de instrumentos está sendo, aos poucos, convertida em software. Sintetizadores, gravadores e teclados virtuais já são uma realidade neste novo século [MIR 99]. Portanto, o uso do computador na área da produção e composição musical será cada vez maior à medida que o músico atual for tomando parte desse mundo tecnológico. Como foi apresentada nesta Tese de Doutorado, a flexibilização na utilização do computador passa por um processo de programação. O MEPSOM poderá contribuir para que os músicos descubram como tornar o computador mais flexível no processo de criação e produção musical. Também poderá contribuir para que o músico entenda que pode utilizar sua criatividade, não só com os pacotes prontos provindo das empresas que dominam o milionário mundo da tecnologia musical, mas também com a criação de seus próprios programas para suas próprias necessidades.

O MEPSOM não visa a substituir as funções do professor de computação musical, ao contrário, tem a finalidade de auxiliá-lo em aulas práticas com turmas numerosas, servindo como recurso didático.

Por fim, o MEPSOM irá contribuir para o desenvolvimento das atividades criativas e construtivas dos músicos, possibilitando que aprendam através de exemplos o conteúdo que antes era de difícil acesso e repleto de barreiras técnicas.

9 REFERÊNCIAS BIBLIOGRÁFICAS

[AME 89] AMES, C. **The Markov Process as a Composicional Model: A Survey and Tutorial.** Leonardo 22(2): 175-188, 1989.

[BAB 64] BABBITT, M. **An Introduction to the RCA Synthesizer.** Journal of Music Theory 8(2): 251, 1964.

[BAB 62] BABBIT, M. **Twelve-tone invariants as composicional determinants.** Musical Quaterly 46: 246-259. Reprinted in P.Lang. Problems in Modern Music. New York: Norton pp. 108-121, 1962.

[BAK 63] BAKER, R. **MUSICOMP: Music Simulator-Interpreter for COMpositional Procedures for the IBM 7090 Electronic Digital Computer.** Technical Report 9. Urbana: University of Illinois Experimental Music Studio, 1963.

[BEC 95] BECKENKAMP, Fábio Ghignatti. **A Proposal of a Harmonic Classification Artificial Intelligence Model.** In: International Joint Conference on Artificial Intelligence, IJCAI, 14., 95, Montreal. **Workshop notes**. Montreal: AAAI, 1995.

[BEC 93] BECKER, Fernando. **A Epistemologia do Professor**: o Cotidiano da Escola. 6ª ed. Petrópolis: Vozes, 1993.

[BEV 95] BEVAN, N. Usability is Quality of Use. In: **Symbiosis of Human and Artifact**. Elsevier, 1995. p.349-354.

[BLE 88] BLEVIS, E.; JENKIS, M. e GLASGOW, J. **Motivations, Sources and Initial Design Ideas for CALM: A Composition Analysis/Generation Language for Music.** In Workshop on Artificial Inteligence and Music. AAAI-88 Conference. Menlo Park, California: American Association for Artificial Inteligence, 1988.

[BOY 86] BOYNTON, L.; LAVOIE, P.; ORLAREY, Y.; RUEDA, C.; e WESSEL, D. **MIDI-Lisp – A Lisp Based Music Programming Enviromment for the Macintosh.** In P. Berg, ed. Proceedings of the 1986 International Computer Music Conference. San Francisco: International Computer Music Association. pp . 183-186, 1986.

[BUC 78] BUCHNER, A. **Mechanical Musical Instruments.** Westport: Greenwood Press, 1978.

[CAR 92] CARY, Tristam. **Dictionary of Musical Technology**. New York: Greewood, c1992. 542p. Il.

[CAS 92] CASEY, Donald E. Descriptive Research: Techniques and Procedures. In: Colwell, R. (ed). **Handbook of Research on Music Teaching and Learning**. New York: Schirmer, 1992. p.115-123.

[CAM 97] CAMP, Victoria. **Making Music on Your PC**. Grand Rapids: Abacus, 1997. 348p. Il.

[CHA 97] CHADABE, Joel. **Electric sound – The past and the promise of electronic music.** Ed. Prentice-Hall, 1997.

[CLA 78] CLARKE, J. and R.VOSS. **1/f noise in music: music 1/f noise.** Journal of the Acoustical Society of America. 63 (1): pp. 258-263, 1978.

[COH 85] COHEN, Louis; MANION, Lawrence. **Research Methodos in Education.** 2nd ed. London: Croom Helm, 1985. p.120–148.

[CON 88] CONGER, Jim. **C Programming for MIDI**. California: M & T Books, 1988.

[COP 87] COPE, David. **An Expert System for Computer-Assisted Composition**, Computer Music Journal, vol. 11 no. 4, MIT press, Massachussetts, 1987.

[CYP 93] CYPHER, Allen. **Watch What I Do**: Programming by Demonstration. London: Editora, 1993.

[DEA 97] DEAL, John J.; TAYLOR, Jack. Technology Standards for College Music Degrees. **Music Educators Journal.** p.17-23. July 1997.

[DOB 98] DOBRIAN, Christopher. **MSP - The Documentation.** Cycling '74, San Fancisco, CA, 1998.

[DOD 88] DODGE, C. **Profile: a musical fractal.** Computer Music Journal 12(3): 10-14, 1988.

[DOD 97] DODGE, Charles; JERSE, Thomas A. **Computer music:** synthesis, composition, and performance**.** New York, Schirmer Book, 1997.

[FER 86] FERREIRA, Aurélio Buarque de Holanda. **Novo Dicionário da Língua Portuguesa.** Ed. Nova Fronteira, 1986.

[FLO 2000a] FLORES, Luciano Vargas. **STI - Sistema para Treinamento de Intervalos para Plataforma Windows 95/98.** 2000. Projeto de Diplomação (Bacharelado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grand do Sul, Porto Alegre.

[FLO 2000b] FLORES, Luciano Vargas. **Música e Internet**: uma Revisão Bibliográfica Visando à Aplicação em Educação Musical via Web. 2000. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grand do Sul, Porto Alegre.

[FLO 2000] FLORES, L. V.; VICCARI, R. M.; PIMENTA, M. S. **Extending the Musical Capabilities of a Multimedia Authoring Environment.** In: BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC, 7., 2000, Curitiba. **CD-ROM dos Anais do XX Congresso Nacional da Sociedade Brasileira de Computação.** Curitiba: Champagnat, 2000.

[FLO 01] FLORES, L. V.; VICARI, R. M.; PIMENTA, M. S. **Some Heuristics for the Development of Music Education Software: First Steps Towards a Methodology.** In: BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC, 8., 2001, Fortaleza. **CD-ROM dos Anais do XXI Congresso da Sociedade Brasileira de Computação.** Niterói: Instituto Doris Ferraz de Aragon, 2001.

[FRI 92] FRITSCH, Eloi Fernando**. Um Estudo Sobre Música & IA e a Implementação de um Sistema Especialista Teórico Musical.** Trabalho individual, CPGCC- Porto Alegre: UFRGS, 1992.

[FRI 94] FRITSCH, E. F.; VICARI, R. **CAMM - Compositor Automático de Melodias Musicais.** (Dissertação de Mestrado). Porto Alegre: CPGCC da UFRGS, 1994.

[FRI 95] FRITSCH, E. F.; VICCARI, R. M. **SETMUS: Uma Ferramenta Computacional para o Ensino da Música.** In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO E MÚSICA, 2., 1995, Canela - RS. **Proceedings.** Instituto de Informática / UFRGS, 1995. p.267-273.

[FRI 95] FRITSCH, E. F. **Música Computacional: A Construção de Sistemas de Computação para Música.** Logos**,** Canoas, v.7, n.2, p.76-88, out. 1995.

[FRI 96] FRITSCH, E. F. STI - **Sistema para Treinamento de Intervalos.** In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO E MÚSICA, 3., 1996, Recife. **Anais...** Recife: Departamento de Música / UFPE, 1996. p.45-55.

[FRI 98] FRITSCH, E. F. et al. **Desenvolvimento de Software Educacional para a Música: STR - Sistema de Treinamento Rítmico.** In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO E MÚSICA, 5., 1998, Belo Horizonte. **Anais do XVIII Congresso Nacional da Sociedade Brasileira de Computação, v.3.** Belo Horizonte: Escola de Música / UFMG, 1998. p.209-218.

[FRI 99] FRITSCH, E. F. **Pesquisa em Hardware e Software para Elaboração de Cursos de Computação e Música.** Acta Scientiae, Canoas, v. 1, nº 1, p.23-31, jan./jun. 1999.

[GOO 90] GOODMAN, Danny. **The Complete HyperCard 2.0 Handbook.** New York, Bantam Doubleday Dell Publishing Group, 1990.

[GOL 85] GOLD, J.; LEWIS, C – Designing for Usability: key Principles and What designers think. **Comunication of the ACM**. New York, v.2, n. , p300-311, Mar. 1985.

[HEL 96] HELMSTETTER, Anthony. **Web Developer's Guide to Sound & Music**. Scottsdale: Coriolis Group Books, c1996. 317p. Il.

[HEN 96] HENTSCHKE, Liane. **Um Estudo Longitudinal aplicando a Teoria Espiral de Desenvolvimento Musical de Swanwick com Crianças Brasileiras da Faixa Etária de 6 a 10 Anos de Idade: Pólo Porto Alegre - 1994.** In: MÚSICA: PESQUISA E CONHECIMENTO 2. Porto Alegre: CPG em Música - Mestrado e Doutorado / UFRGS - NEA, 1996a. p.9-34.

[HEN 96a] HENTSCHKE, Liane. **A Teoria Espiral de Swanwick com Fundamentação para uma Proposta Curricular.** In: V Encontro Anual da ABEM (1996: Londrina). Anais. Londrina, ABEM, 1996b. p.171-185.

[HEY 96] HEYWOOD, Brian. **PC Music Handbook: windows 95 compatible**. 2.ed. Kent: PC, 1996. 216p. Il.

[HIL 59] HILLER L. and L. ISSACSON. **Experimental Music.** New York: MacGraw-Hill, 1959.

[HIL 64] HILLER, L. and R.BAKER. **Computer Cantata: A Study in Compositional Method.** Perspectives of New Music. 3:62-90, 1964.

[HIL 65] HILLER, L. and R.BAKER. **Automated Music Printing.** Journal of Music Theory 9: 129-150, 1965.

[HIL 69] HILLER, L. **Some Compositional Techniques Involving the Use of Computers.** In H. Von Foerster and J. Beauchamp, eds. Music by Computers. New York: John Wiley and Sons. Pp 71-83, 1969.

[HIL 70] HILLER L. **Music Composed with Computer – a Historical Survey.** In H. Lincoln, ed. The Computer and Music. Ithaca: Cornell University Press. Pp. 42-96, 1970.

[HIL 71] HILLER, L and P. RUIZ. **Sinthesizing sounds by solving the wave equation for vibrating objects.** Journal of the Audio Engineering Society 19: 463-470, 542-551, 1971.

[HIL 79] HILLER L. **Phrase Structure in Computer Music.** In C. Roads, ed. Proceedings of the 1978 International Computer Music Conference. Evanston: Northwestern University Press. pp. 192-213, 1979.

[HIL 81] HILLER L. **Composing with Computer: a Progress Report.** Computer Music Journal 5(4): 7-21. Reprinted in C. Roads ed. 1989. The Music Machine. Cambridge, Massachusetts: The Mit Press. Pp 75-89, 1981.

[HUN 97] HUNT, Andy; KIRK, Ross. 1997. Technology and Music: Incompatible Subjects? Britsh Journal of Music Education Vol. 14, Number 2, Cambridge University Press, UK.

[KIR 68] KIRCHMEYER, h. **On the Historical Constitution of a Rationalistic Music.** Die Reihe 8. English Edition. Bryn Mawr: Theodore Presser Company, 1968, pp. 11-24.

[KOE 96] KOELLREUTTER, H. J. **Contraponto Modal do Século XVI** : Palestrina, Brasília, DF: Musimed Editora, 1996.

[KOE 78] KOENING, G.M. **Description of the Project 1 Programme.** Utrecht: Institute of Sonology, 1978 .

[KOE 79] KOENING, G.M. **Protocol.** Sonological Reports Number 4. Utrecht: Institute of Sonology, 1979.

[KRÜ 99] KRÜGER, S. E.; FRITSCH, E. F.; FLORES, L. V.; GRANDI, R. H.; SANTOS, T. R.; HENTSCHKE, L.; VICCARI, R. M. **Developing a Software for Music Education: An Interdisciplinary Project.** In: SIMPÓSIO BRASILEIRO DE COMPUTAÇÃO E MÚSICA, 6., 1999, Rio de Janeiro. **Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação, v.3.** Rio de Janeiro: EntreLugar, 1999. p.251-264.

[KRÜ 2000] KRÜGER, Susana Ester. **Desenvolvimento, Testagem e Proposta de um Roteiro para Avaliação de Software para Educação Musical.** Dissertação de Mestrado. Porto Alegre: Programa de Pós-Graduação em Música / UFRGS, 2000.

[LAW 85] LAWLER, B., **Computer Experience and Cognitive Development,** New York: Ellis Horwood and Sons, 1985.

[LEH 93] LEHRMAN, Paul D. et al. MIDI for the Professional, Ed Amsco Publication, 1997.

[LOY 89] LOY, D. G. **Composing with Computers – a survey of some compositional formalisms and music programming languages.** In M.Mathews and J. R. Pierce, eds. Current Directions in Computer Music Research. Cambridge. Massachusetts: The MIT Press, 1989, pp. 292-396.

[MAR 71] MARTIRANO, S. **An Electronic Music Instrument Wich Combines The Composing Processes With Performance in Real Time.** Progress Report I. Department of Music. Urbana: University of Illinois, 1971.

[MAT 69] MATHEWS, M. e ROSLER, L. **Graphical Language for the Scores of Computer Generated Sounds.** In H. von Foerster and J. Beauchamp, eds. Music by Computers. New York: John Wiley and Sons. Pp. 84-114, 1969.

[McN 86] McNABB, M. **Computer Music: some aesthetic considerations.** In S. Emmerson. The Language of Eletroacoustic Music. New York: Harwood Academic. Pp. 141-154, 1986.

[McN 89] McNABB, M. **Dreamsong: the composition**. The Music Machine. Cambridge, Massachusets: The Mit Press, 1989.

[MES 98] MESSIACK, Paul. **Maximun MIDI**: music applications in C ++. Manning Pub, 1998.

[MIR 98] MIRANDA, Eduardo Reck. **Computer Sound Synthesis for the Electronic Musician**. Oxford, Focal Press, 1998.

[MIR 99] MIRANDA, Eduardo Reck. **Músicas Y Nuevas Tecnologias – Perspectivas para el Siglo XXI.** Ed. Associacion de Cultura Contemparania L´angelot – Barcelona, Espanha, 1999.

[MIR 2001] MIRANDA, Eduardo Reck. **Composing Music with Computers**. Oxford, Focal Press, 2001.

[NEW 98] NEWMAN, Isadore; BENZ, Carolyn R. **Qualitative – Quantitative Research Methodology**: Exploring the Interactive Continuum. Carbondale and Edwardsville: Southern Illinois University Press, 1998.

[NIE 93] NIELSEN, E. et al. **Usability Engeneering.** Boston: AP Professional, 1993.

[NOR 86] NORMAN, D. A.; Draper, S. W (Ed.), **User Centered System Design**: New Perspectives on Human-Computer Interaction. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986.

[OET 69] OETTINGER, A; MARKS, S. **Run, Computer, Run.** Cambridge, MA, Harvard University Press, 1969.

[OLI 96] OLIVEIRA, Alda. **Um Estudo Longitudinal Aplicando a Teoria Espiral de Desenvolvimento musical de Swanwick com crianças brasileiras da faixa etária de 6 a 10 anos de Idade.** *Pólo Salvador – 1994*. In: Música: Pesquisa e Conhecimento. Vol. 2, CPG-Música – Mestrado e Doutorado/UFRGS – NEA. Porto Alegre: 1996, p. 35-67.

[OLS 55] OLSON, H. and H. BELAR. **Electronic Music Synthesizer.** Journal of the Acoustical Society of America 27(5): pp. 595, 1955.

[OLS 61] OLSON, H. and H. BELAR. **Aid to Music Composition System Employing a Random Probability System.** The Journal of Acoustic Society of America. 33: pp 1163-1170, 1961.

[PEN 95] PENFOLD, R.A. **Advanced MIDI User's Guide**. 2.ed. Kent: PC, 1995. 184p. Il.

[POL 87] POLANSKY, L. D; ROSENBOOM; BURK, P. **HMSL: Overview (Version 3.1) and notes on intelligent instrument design.** In J. Beauchamp;

Proceedings of the International Music Conference. San Francisco: International Computer Music Association. pp 220-227, 1987.

[POP 86] POPE, S. **The Development of na Inteligent Composers's Assistant.** In P. Berg, ed. 1986. Procedings of the 1986 International Computer Music Conference. San Francisco: International Computer Music Association. Pp 131-144, 1986.

[PRI 75] PRIEBERG. **Musica Ex Machina.** Italian Edition. Turin: Giulio Einaudi Editore, 1975.

[PUC 1990] PUCKETTE, Miller et al. – **An Interative Graphic Programming Enviromment** , IRCAM, 1990.

[RAT 92] RATTON, Miguel. **Guia básico de Referência,** Ed Campus Ltda, 1992.

[RAT 97] RATTON, Miguel. **Computador & Música**. Informus, Rio de Janeiro, CD-ROM, 1997.

[RED 88] REDMON, Nigel. **HyperMIDI 1.1.** Torrance, CA: EarLevel Engineering, 1988

[RHE 72] RHEA, T. **The Evolution of Electronic Musical Instruments in the United States** PhD. Diss. Nashville: George Peaboly College for Teachers, 1972.

[RIC 96] RICHEY, Rita C.; NELSON, Wayne A**.** Developmental Research. In: JONASSEN, David H. (ed.). **Handbook of Research for Educational Communications and Technology**. New York: Macmilan, 1996. p.1213-1246.

[ROA 96] ROADS, Curtis. **The Computer Music Tutorial.** MIT, 1996.

[SAN 81] SANDRESKY, M. **The Golden Section in three Byzantine Motets of Dufay.** Journal of Music Theory, 1981, pp. 25(2).

[SCH 46] SCHILLINGER, J. **The Schillinger System of Musical Composition.** New York : Carl Fischer, Reprinted 1978. New Yrok: Da Capo Press, 1946

[SCH 75] SCHOLES, P. The Oxford Companion to Music. London: Oxford University Press, 1975.

[SEL 97] SELFRIEDGE-FIELD, Eleanor. **Beyond MIDI – The Handbook of Musical Codes.** Center for Computer Assisted Research in the Humanities, 1997.

[SOL 96] SOLOWAY, E. et al. **Learning Theory in Practice: Case Studies of Learner-Centered Design**. In: *Proceedings of CHI96*. New York: ACM Press, 1996. pp. 189-196.

[SOW 56] SOWA, J. **A Machine to Compose Music.** Instruction Manual for Geniac. New Haven: Oliver Garfield Co., 1956.

[SQU 96] SQUIRES, David; PREECE, Jenny. **Usability and Learning: Evaluating the Potential of Educational Software**. In: Computers & Education, vol. 27, n.º 1, 1996. p.15-22.

[SWA 79] SWANWICK, Keith. **A Basis for Music Education**. London: Routledge, 1979.

[SWA 88] SWANWICK, Keith. **Music, Mind and Education**. London: Routledge, 1988.

[THO 91] THOMAS, J. **Fractals in Algorithmic Composition with Computers.** In Computers and Music Research Conference Handbook. Belfast: The Queen's University, 1991.

[THO 97] THOMPSON, R. **PC MUSIC – A Tutorial Introduction,** in the CD-ROM of the book Computer Sound Synthesis, Atlanta, 1997.

[TRI 87] TRIVIÑOS, Augusto N. S**. Introdução à Pesquisa em Ciências Sociais**: a Pesquisa Qualitativa em Educação. São Paulo: Atlas, 1987.

[TRU 85] TRUAX, B. **The PODX System: Interactive Composicional Software for the DMX-1000.** Computer Music Journal 9(1): 29-38, 1985.

[VAL 2000] VALIATI, E.R.A.; LEVACOV, M.; LIMA, J.V.; PIMENTA, M.S. **Guia-GEPESE: User Interface Guidelines for Educational Software.** In: Simposio Internacional de Informática Educativa - SIIE'2000, II. Puertollano (Ciudad Real), Espanha: Novembro 2000. **Anais...** (Em mídia eletrônica).

[WAS 89] WASCHKA, R. e KUREPA, A. **Using Fractals in Timbre Construction: A exploratory study.** In T. Wells and D. Butler, eds. Porceedings of the 1989 International Computer Music Conference. San Francisco: International Computer Music Association. pp 332-335, 1989.

[WIN 2000] WINCKLER, M. A. A.; NEMETZ, F.; LIMA, J. V. **Interação entre Aprendiz e Computador: Métodos para Desenvolvimento e Avaliação de Interfaces.** In: Versão preliminar do livro TECNOLOGIA DIGITAL NA EDUCAÇÃO (apresentado no IV Workshop Informática na Educação, Porto Alegre, Set. 2000), Cap.1. Porto Alegre: Gráfica UFRGS, 2000.

[WIN 98] WINKLER, Todd. **Composing Interactive Music – Techniques and Ideas Using MAX,** MIT, 1998.

[WIP 89] WINSOR, Phil. **Automated Music Composition**, USA, 1989.

[WIN 87] WINSOR, P. **Computer-Assisted Music Composition.** Princeton: Petrocelli Books, 1987.

[WIN 90] WINSOR, P. **Computer Composer's Tookbox.** Blue Ridge Summit: Tab Books, 1990.

[WIN 91] WINSOR, P. **Computer Music in C.** Blue Ridge Summit: Tab Books, 1991.

[WUL 01] WULFHORST, R. D.; FLORES, L. V.; NAKAYAMA, L.; FLORES, C. D.; ALVARES, L. O. C.; VICARI, R. M. **An Open Architecture for a Musical Multi-Agent System.** In: BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC, 8., 2001, Fortaleza. **CD-ROM dos Anais do XXI Congresso da Sociedade Brasileira de Computação.** Niterói: Instituto Doris Ferraz de Aragon, 2001a.

[WUL 01] WULFHORST, R. D.; FRITSCH, E.; VICARI, R. M. **SeVEM - Separador de Vozes em Execuções Musicais Polifônicas.** In: BRAZILIAN SYMPOSIUM ON COMPUTER MUSIC, 8., 2001, Fortaleza. **CD-ROM dos Anais do XXI Congresso da Sociedade Brasileira de Computação.** Niterói: Instituto Doris Ferraz de Aragon, 2001b.

[XEN 60] XENAKIS, I. **Elements of Stochastic Music.** Gravesaner Bläter 18:84-105, 1960.

[XEN 92] XENAKIS, I. **Formalized Music.** Revised Edition. New York: Pendragon Press, 1992.

[YAV 91] YAVELOW, C. **Music & Sound Biblle**, San Mateo, California: IDG Books WordWide, Inc, 1992.

[YOU 96] YOUNG, Robert. **The MIDI Files**. London: Prentice Hall, 1996. 318p. Il.

[ZAP 60] ZARIPOV, R. **A n Algorithmic Description of the Music Composing Process.** Doklady Academiia Nauk SSSR 132: 1283. English Translation in Automation Express 3:17, 1960.

[ZAR 69] ZARIPOV, R. **Cybernetics and Music.** Perspectives of New Music 7(2): 115-154. Translation by Russel of Kibernetika i Muzyka (1963), 1969.

[ZIC 88] ZICARELLI, David; PUCKETTE, Miller. **MAX Development Package.** France: IRCAM, 1988.