

UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

YURI VINICIUS DE LIMA CHEN

**Implementação de interfaces gráficas para produção de cenas sonoras expandidas no sistema AUDIENCE de áudio espacial**

São Paulo

2021

YURI VINICIUS DE LIMA CHEN

**Implementação de interfaces gráficas para produção de cenas sonoras expandidas no sistema AUDIENCE de áudio espacial**

Monografia apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, como parte dos requisitos exigidos na disciplina ACH 2018 – Projeto Supervisionado ou de Graduação II, para obtenção do título de Bacharelado em Sistemas de Informação.

Modalidade: TCC longo (1 ano) – individual

Orientador: Prof. Dr. Regis Rossi Alves Faria

São Paulo

2021

**Agradecimentos**

Agradeço primeiramente aos meus pais por me darem todo suporte necessário ao meu desenvolvimento não somente acadêmico/profissional, mas também como pessoa.

Agradeço aos amigos e familiares por estarem sempre do meu lado.

Agradeço aos meus professores da EACH por todas as vezes que me ajudaram com as questões do curso.

Agradeço ao Prof. Regis, meu orientador, pela oportunidade de poder trabalhar neste projeto e pela orientação paciente e precisa.

**Resumo**

CHEN, Yuri Vinicius de Lima. **Implementação de interfaces gráficas para produção de cenas sonoras expandidas no sistema AUDIENCE de áudio espacial**. 2021. XX f. Monografia (Bacharelado em Sistemas de Informação) – Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, 2021.

A proposta do projeto é a implementação de uma nova interface gráfica de usuário para o sistema de áudio espacial AUDIENCE, uma biblioteca de objetos e *patches* para Pure Data (Pd). O AUDIENCE permite a criação, simulação, manipulação e reprodução de cenas espaciais sonoras. Já o Pure Data é um ambiente de programação visual voltado para áudio, vídeo e processamento gráficos, sendo utilizado para performances musicais ao vivo, efeitos sonoros/visuais, produção musical, análise e síntese sonora. O AUDIENCE é voltado para produtores musicais e do audiovisual, artistas, criadores de efeitos sonoros, etc. Basicamente, a construção da nova interface envolve solucionar dois problemas: integrar os níveis L1 e L4 do AUDIENCE, para permitir a edição de cena acústica e cena de alto-falantes,e implementar um mecanismo de autoconstrução da GUI usando-se programação dinâmica.

Palavras-chaves: Cena sonora espacial, Interface gráfica de cena, Auralização, Computação sonora e musical, Programação dinâmica

**Abstract**

CHEN, Yuri Vinicius de Lima. **Implementation of graphical interfaces for the production of expanded sound scenes in the AUDIENCE spatial audio system**. 2021 XX p. Monograph (Bachelor of Information Systems) – School of Arts, Sciences and Humanities, University of São Paulo, São Paulo, 2021.

The project proposal is the implementation of a new graphical user interface for the AUDIENCE spatial audio system, an object library and patches for Pure Data (Pd). AUDIENCE allows the creation, simulation, manipulation and reproduction of spatial sound scenes. The Pure Data is a visual programming environment focused on audio, video and graphics processing, being used for live musical performances, sound effects / visual production, music, analysis and sound synthesis. The AUDIENCE is aimed at music and audiovisual producers, artists, sound effects creators, etc. Basically, building the new interface involves solving two problems: integrating AUDIENCE`s L1 and L4 levels to allow editing of acoustic scene and loudspeaker scene, and implementing a GUI auto-build mechanism using dynamic programming.

Keywords: Spatial Sound Scene, Scene Graphic Interface, Auralization, Sound and Music Computation, Dynamic Programming

**Lista de figuras**

Figura 1 - As diversas aplicações do AUDIENCE…………………….....................................2

Figura 1 - Exemplo de Patch do Pure Data que imprime “Hello World”...................................2

Figura 2 - Exemplo de outro patch Pure Data mas que trabalha com sinal de áudio: geração um som puro de 440Hz com controle de volume………………………………………….......3

Figura 3 - Exemplo de uma cena sonora na interface do AUDIENCE……………………….4

Figura 4 - Ilustração alto-nível das camadas funcionais do AUDIENCE…………………….5

Figura 5 - Todos os objetos do AUDIENCE………………………………………………….6

Figura 6 - Exemplo de um patch Pd construído com a biblioteca AUDIENCE que faz a renderização acústica de uma cena sonora espacial…………………………………………..7

Figura 7 - Formato Ambisonics……………………………………………………………... 10

Figura 8 - Ilustração da interface já existente de cena L4, interface de cena L1 e de uma nova interface que integra em uma mesma cena camadas de objetos emissores de som de L1, fontes, e de L4, alto-falantes………………………………………………………………... 11

Figura 9 - Como instanciar uma interface L1 (oa\_l1\_gui)........................................................13

Figura 10 - Envio das posições dos elementos da cena……………………………………... 14

Figura 11 - O novo objeto AUDIENCE oa\_l1l4\_gui ………………………………………..22

Figura 12 - Instanciando oa\_l1l4\_gui com três ouvintes…………………………………...23

Figura 13 - Instanciação de uma interface L1 e L4 de 150x150, escala 0.1 (ou seja, 15x15m), 2 fontes, 1 alto-falante, 2 ouvintes e o formato da escala cartesiano.……………………...24

Figura 14 - Comparação entre interfaces de formato Ambisonics e cartesiano.……..…….24

Figura 15 - Demonstração passo-a-passo da funcionalidade de exibir/ocultar L1/L4………………………………………………………………………………………....24

Figura 16 - Os Módulos (partes) da interface reconfigurável………………………………...25

Figura 17 - Patch da matriz L1 aberto………………………………………………………...26

Figura 18 - Exemplo de patch que faz o uso de programação dinâmica para criar em tempo de execução o subpatch teste…………………………………………………….……………...26

Figura 18 - Exemplo de uso dos objetos “send” e “receive”....................................................26

Figura 20 - Módulo L1………………………………………………………..……………...28

Figura 21 - O subpatch build-interface à esquerda e o módulo da cena sonora à direita………………………………………………………………………………………...29

Figura 22: Subpatch build-L1-control aberto à esquerda e o subpatch dos controles de volume L1 aberto à direita.……………………………………………………………………….…..30

Figura 23: Patch build-L1-matrix aberto…………………………………………………….34

Figura 24: Patch build-L3 aberto…………………………………………………………….35

Figura 25: Patch da Matriz L3 aberto…………………………………………………...……37

Figura 26 - Módulo L4 aberto………………………………………………………………...38

**Lista de algoritmos**

Algoritmo 1 - Código do objeto “counter”. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

Algoritmo 2 - Espaço de dados (struct \_oa\_l1\_gui) e métodos do oa\_l1\_gui. . . . . . . . . . 14

Algoritmo 3 - Espaço de dados do objeto oa\_l1l4\_gui. . . . . . . . . . . . . . . . . . . . . . . . . . . 16

Algoritmo 4 - Método oa\_l1l4\_gui\_l4on. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

Algoritmo 5 - Método oa\_l1l4\_gui\_l4off. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

**Lista de abreviaturas e siglas**

AUDIENCE Audio Immersion Experience by Computer Emulation

Pd Pure Data

**Sumário**

[1 Introdução](#_heading=h.gjdgxs) **1**

[1.1 Pure Data](#_heading=h.d2e7olwor9eq) 4

[1.2 AUDIENCE](#_heading=h.oex4byesywpf) 6

[1.3 Outros projetos voltados para espacialização sonora](#_heading=h.7c7ubafms8tc) 9

[1.4 Objetivos](#_heading=h.d1cardxgtctl) 11

[1.5 Cronograma](#_heading=h.4w4tzbmdqfcl) 14

[2 Revisão bibliográfica](#_heading=h.30j0zll) **14**

[3 Estudo das interfaces L1 e L4](#_heading=h.2et92p0) **15**

[3.1 Argumentos, entrada e saída das interfaces L1 e L4](#_heading=h.w5a3i0n7kg2z) 15

[3.2 O algoritmo das interfaces](#_heading=h.8e3b2iz3mp17) 17

[4 Implementação da interface L1/L4 (oa\_l1l4\_gui)](#_heading=h.1t3h5sf) **19**

[4.1 O espaço de dados da nova interface](#_heading=h.4m9iqippo4ax) 19

[4.2 O construtor da nova interface](#_heading=h.fa4s3zbf62er) 22

[4.3 Os métodos da nova interface](#_heading=h.3h1bxxxysb45) 22

[5 Resultados da primeira fase](#_heading=h.h8v2vrhap5gv) **24**

[5.1 Integração L1 e L4:](#_heading=h.92uj2ta3prf5) 24

[5.2 Número de ouvintes configurável](#_heading=h.cbql9tygeejx) 25

[5.3 Formato da escala configurável (Ambisonics ou cartesiano)](#_heading=h.y21fwc3nydnq) 25

[5.4 Função de exibir/ocultar](#_heading=h.qya3p06p2oso) 26

[**6 Segunda fase: interface reconfigurável**](#_heading=h.gk4h99608foy) **27**

[6.1 Princípios básicos da programação dinâmica no Pure Data](#_heading=h.t7w3ezz7aeu8) 29

[6.2 Módulos da interface reconfigurável e seus mecanismos de auto-construção](#_heading=h.goosezaiwwvs) 31

[6.2.1 Módulo L1](#_heading=h.j83v296n3gn9) 31

[6.2.1.1 Subpatch build-interface](#_heading=h.v6qf6chkrwuv) 32

[6.2.1.2 Subpatch build-L1-control](#_heading=h.lzj8vw6efncc) 33

[6.2.1.3 Subpatch build-L1-matrix](#_heading=h.w1yg11lzkjfs) 34

[6.2.1.4 Subpatch build-L3](#_heading=h.rvddq511cvk7) 35

[6.2.2 Módulo L2](#_heading=h.dulptck8ex22) 36

[6.2.3 Módulo L3](#_heading=h.ku2l31xpxfs9) 36

[6.2.4 Módulo L4](#_heading=h.5popb8nuvgqr) 37

[**7 Conclusão**](#_heading=h.qvbahhqorub0) **38**

[Referências bibliográficas](#_heading=h.bngsmp8wzyx5) **39**

[ANEXO A – Como fazer um external para Pure Data](#_heading=h.26in1rg) **43**

[ANEXO B – Algoritmo de recebimento de mensagem de posição de fontes sonoras](#_heading=h.fkwd3l2pbgsd) **48**

[**ANEXO C – Algoritmos de exibir/ocultar L1/L4**](#_heading=h.v5af8770qd9o) **50**

# 1 Introdução

Imagine, por exemplo, ouvir um show gravado, mas como se realmente estivesse lá no lugar do show, com os instrumentos posicionados em diferentes pontos do ambiente e seus sons vindo de todos os lados e de cima. Essa é a promessa do áudio espacial ao oferecer o som distribuído em 360 graus onde a pessoa ouvinte é o centro do processo, tornando a experiência muito mais envolvente.

As técnicas de simulação espacial de áudio encontram diversas aplicações nas áreas de realidade virtual, games, produção musical, performances audiovisuais, teleconferências, etc. Alguns exemplos de aplicações:

* projeção de campos sonoros envolventes (surround) para uso em shows, teatros, auditórios, ambientes de apresentações multimídia, telecomunicações, etc.
* auralização; considere por exemplo um projeto em engenharia aeronáutica, onde projetistas navegam em torno de uma maquete de avião, interessados em inspecionar o fluxo de ar pela fuselagem. O ruído associado ao comportamento turbulento do ar e às diferentes intensidades sonoras nos diversos pontos da fuselagem poderiam ser adicionados à navegação visual, oferecendo um conjunto de informações acústicas relevantes ao projeto, trazendo a realidade simulada ao alcance do projetista. Em um projeto de isolamento e tratamento do ruído interno para um avião a navegação audiovisual dentro da maquete virtual permitiria por exemplo avaliar a qualidade e intensidade sonora em pontos distintos da cabine, auxiliando na identificação de problemas localizados, falhas, e numa melhor sintonia do projeto. Uma auralização fisicamente correta do campo sonoro seria necessária para tais atividades em concepção e projetos de engenharia.

A maioria de nós escuta música ou som em estéreo, o que só nos dá som vindo de uma dimensão: da frente da esquerda para a direita. O som surround nos trouxe uma segunda dimensão e com a introdução do som 3D ou imersivo, agora você pode ter uma sensação de fontes de som vindo de todos os lados. Com a realidade virtual como uma tecnologia emergente rapidamente, o áudio tridimensional está recebendo mais atenção do que nunca. A RV convincente precisa de áudio que coloque sons de forma convincente em um espaço tridimensional para que o usuário perceba o som como vindo de objetos físicos reais, em sua experiência de RV. Em outras palavras, tornando-o tão parecido com o que ouvimos na vida real.

Áudio espacial é qualquer áudio que lhe dá uma sensação de espaço além do estéreo convencional, permitindo ao usuário identificar de onde o som está vindo, seja acima, abaixo ou 360 graus ao seu redor. O estéreo permite que você ouça coisas na parte frontal esquerda e direita, mas você não pode ter uma sensação de surround, altura ou sons abaixo de você. Com a introdução da 3ª dimensão, você tem uma noção da localização exata das fontes de som ao redor.

Seguem os métodos e tecnologias mais recentes e populares de espacialização/auralização sonora:

* **Surround:** nos sistemas surround as caixas de som cercam o ouvinte, criando um campo sonoro de 360 graus. Além disso, o sistema divide o áudio entre as caixas, fazendo com que a emissão coincida com os microfones utilizados na gravação da cena, dando maior realidade.
* **Ambisonics:** Ambisonics é um método para gravar, mixar e reproduzir áudio tridimensional de 360 graus. Foi inventado na década de 1970, mas nunca foi adotado comercialmente até recentemente com o desenvolvimento da indústria de realidade virtual, que requer soluções de áudio 360 °. A abordagem básica do Ambisonics é tratar uma cena de áudio como uma esfera de som de 360 graus vinda de diferentes direções em torno de um ponto central. O ponto central é onde o microfone é colocado durante a gravação ou onde o "ponto ideal" do ouvinte está localizado durante a reprodução.

Os formatos surround tradicionais podem fornecer boa imagem quando estáticos; mas conforme o campo sonoro gira, o som tende a 'pular' de um alto-falante para outro. Por outro lado, o Ambisonics pode criar uma esfera de som suave, estável e contínua, mesmo quando a cena de áudio gira (como, por exemplo, quando um jogador usando um fone de ouvido VR move a cabeça). Isso ocorre porque o Ambisonics não é pré-limitado a qualquer conjunto de alto-falantes em particular.

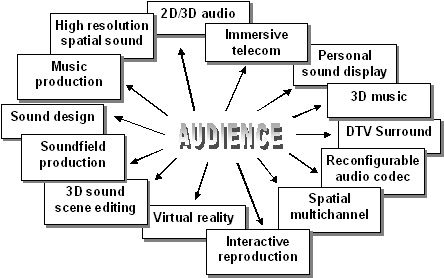
Os sistemas de alto-falantes surround tradicionais são geralmente "enviesados para a frente": as informações dos alto-falantes laterais ou traseiros não são tão focadas quanto o som da frente. Por outro lado, o Ambisonics foi projetado para espalhar o som uniformemente por toda a esfera tridimensional.

Por fim, enquanto os sistemas surround tradicionais têm várias dificuldades para representar o som além da dimensão horizontal, o Ambisonics foi projetado para fornecer uma esfera completa com elevação, onde os sons são facilmente representados como vindo de cima e de baixo, bem como na frente ou atrás do usuário. ( <http://www.ambisonic.net/> )

* **Wave Field Synthesis:** Wave Field Synthesis (WFS) é uma técnica de reprodução de som usando matrizes de alto-falantes que redefine os limites estabelecidos por técnicas convencionais (estéreo, 5.1 ...). Essas técnicas baseiam-se em princípios estereofônicos que permitem a criação de uma ilusão acústica (em oposição a uma ilusão de ótica) sobre uma área muito pequena no centro da configuração do alto-falante, geralmente referida como "ponto ideal". O WFS, por outro lado, visa reproduzir os verdadeiros atributos físicos de um determinado campo sonoro em uma área extensa da sala de audição. É baseado no princípio de Huyghens (1678), que afirma que a propagação de uma onda através de um meio pode ser formulada adicionando as contribuições de todas as fontes secundárias posicionadas ao longo de uma frente de onda.

Então o AUDIENCE, software que está sendo desenvolvido neste projeto, objetivou a pesquisa e o desenvolvimento de tecnologias para imersão sonora, produção e reprodução de áudio 2D/3D, com aplicações em multimídia, realidade virtual, "home-theaters" e sonorização "surround" que crescem cada vez mais em demanda e importância. O AUDIENCE é uma biblioteca de objetos e patches para Pure Data. Então antes de explicar o que é o AUDIENCE, é necessário primeiramente compreender o Pure Data.

Figura 1 - As diversas aplicações do AUDIENCE:



## *1.1 Pure Data*

Pure Data (ou abreviadamente Pd) é uma linguagem de programação visual baseada em fluxo de dados/sinal desenvolvida por Miller Puckette, professor no departamento de música da Universidade da Califórnia, na década de 1990 para processamento de áudio, vídeo e gráficos 2D/3D. Apesar de Miller Puckette ser o seu autor principal, o Pd é uma linguagem opensource com uma base de desenvolvedores bastante extensa que continuam a trabalhar em novas funcionalidades.

No Pure Data você consegue fazer manipulações e processamentos em sinais de áudio construindo um “patch”, que é uma composição de “objetos” conectados. Cada objeto pode receber um número/mensagem/sinal e realiza um processamento nessa entrada recebida e pode “jogar” o dado processado na entrada de outro objeto. Também ainda há vários tipos de botões e visualizadores.

Figura 1 - Exemplo de Patch do Pure Data que imprime “Hello World”. A janela à esquerda, é onde se faz a composição dos patches. A janela à direita, é o Console. Ao clicar na caixa de mensagem onde está escrito “Hello World”, a mensagem “Hello World” é enviada ao objeto “print” através da conexão entre o objeto de mensagem e o objeto print. O objeto “print” imprime no console do Pure Data a mensagem recebida. A parte do objeto que recebe o começo da conexão chama-se *inlet*. E a parte que recebe o fim da conexão chama-se *outlet*.

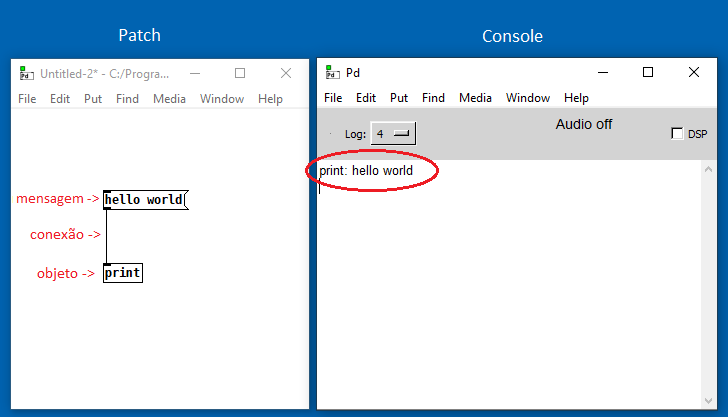
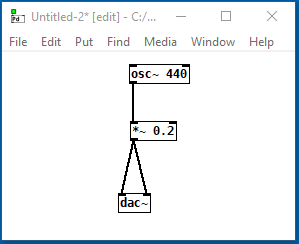


Figura 2 - Exemplo de outro patch Pure Data mas que trabalha com sinal de áudio: geração um som puro de 440Hz com controle de volume. O objeto “osc~” (todo objeto do Pd que trabalha com sinal de áudio tem “~” no final do nome) gera um som puro, uma onda senoidal O argumento de osc~ é “440” que é a frequência que essa onda senoidal terá em Hz, ou seja, está sendo gerada um som de uma onda senoidal de 440Hz. O sinal deste áudio é enviado para um objeto que multiplica esse som por número (no caso, 0.2), ou seja, faz o controle de volume. E finalmente, o sinal é enviado para o objeto “dac” (conversor digital-analógico) que reproduz o som.



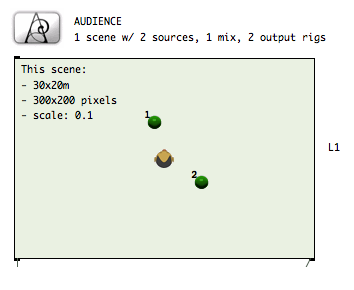
Os objetos que já vem no Pure Data (como o print, osc~ e dac~ vistos nos exemplos acima) se chamam *internals*. E também é possível você fazer os próprios objetos utilizando a linguagem C. Esses objetos que não vem como padrão no Pd, se chamam *externals* (logo os objetos do AUDIENCE são externals)*.*

## *1.2 AUDIENCE*

O software AUDIENCE assim permite criar, simular, manipular e reproduzir cenas sonoras espaciais para as mais diversas aplicações em diversas áreas do conhecimento. O projeto do AUDIENCE (Audio Immersion Experience by Computer Emulation) iniciou-se em 2003 no Laboratório de Sistemas Integráveis da USP ([www.lsi.usp.br/audience](http://www.lsi.usp.br/audience)) objetivando a pesquisa e o desenvolvimento de tecnologias para imersão sonora. O software consiste em uma biblioteca de objetos e patches para Pure Data.

O AUDIENCE possui duas versões: uma livre (Open AUDIENCE), para desenvolvedores de aplicações com áudio 3D e para disseminação da plataforma de auralização, e uma versão beta, restrita à equipe de desenvolvimento, base para a construção da distribuição livre, sistemas dedicados e distribuições embarcadas.

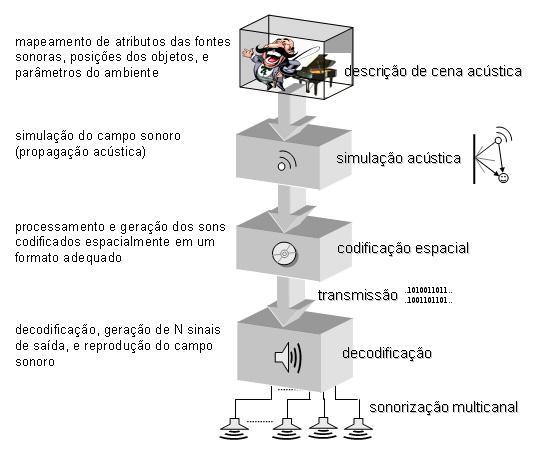
No AUDIENCE, o usuário constrói as cenas sonoras virtualmente, definindo os sons de cada fonte sonora, e suas posições no espaço que podem ser alteradas em tempo real, inclusive automatizadas. O campo sonoro real pode ser exportado e tocado em suporte hexa, octofônico, e outros formatos de saída (ex: modos 5.1 e estéreo).

Figura 3 - Exemplo de uma cena sonora na interface do AUDIENCE. O retângulo representa o ambiente da cena. O elemento amarelo no centro da cena é o ponto de audição. As esferas verdes representam as fontes sonoras.

O AUDIENCE é composto em 4 camadas:

* Descrição da cena sonora (L1): Nesta camada é onde se dá a composição da cena auditiva configurando-se os parâmetros do ambiente, posições e atributos das fontes sonoras.
* Simulação da cena acústica (L2): Nesta camada é onde ocorre o cálculo da propagação acústica dos sons da cena e obtém-se a representação auditiva do som em um determinado ponto de audição.
* Codificação sonora espacial (L3): camada onde ocorre o processamento e geração do áudio.
* Decodificação e reprodução (L4): camada onde ocorre a decodificação e a geração de N sinais de saída reproduzindo o campo sonoro.

Figura 4 - Ilustração alto-nível das camadas funcionais do AUDIENCE.



Fonte: http://www.lsi.usp.br/interativos/neac/audience/audience.html#0

Essa implementação em camadas apresenta diversas vantagens e características:

* Desacoplamento de maneira sistemática dos principais processos da cadeia permitindo que sejam realizados em módulos independentes.
* Padronização dos conjuntos de sinais, parâmetros e mensagens válidos que devem trafegar de uma camada para outra.
* Desenvolvimento de um sistema modular e escalável.

Figura 5 - Todos os objetos do AUDIENCE. Obs: essa figura não está atualizada. Na camada L4, está faltando o “audce\_L4\_gui” que instancia a interface de alto-falantes.

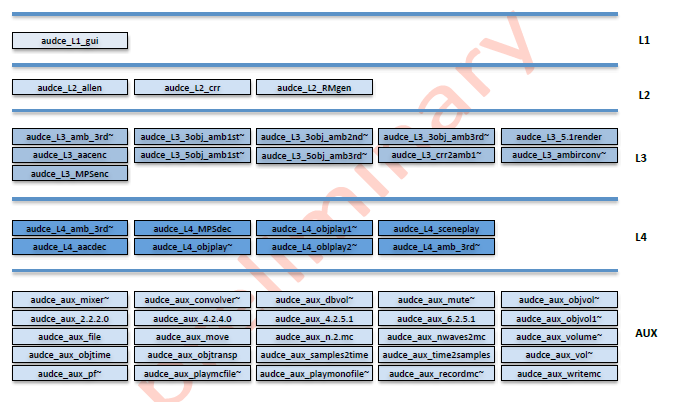
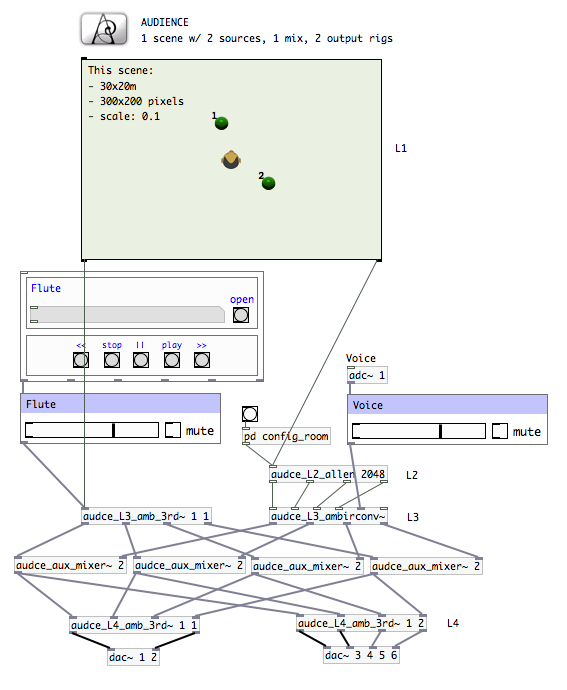


Figura 6 - Exemplo de um patch Pd construído com a biblioteca AUDIENCE que faz a renderização acústica de uma cena sonora espacial. Neste exemplo, 2 linhas de processamento paralelas são construídas, uma para cada fonte sonora em cena, produzindo no final 2 programas de áudio em modos distintos: um para audição em estéreo (2 canais) e um para audição quadrifônica (square, 4 canais). Pela interface (L1) vemos a configuração da cena: o ouvinte e duas fontes sonoras, uma flauta (esfera verde 1) e uma voz captada pelo microfone (esfera verde 2). No outlet esquerdo, a interface envia a posição da flauta na cena, e no outlet direito é enviada a posição da voz na cena. Os objetos da camada L3 codificam o áudio para o formato Ambisonics, passando para uma camada de objetos auxiliares mixers e depois passam para uma camada de objetos L4 que fazem a decodificação do sinal de áudio, e são reproduzidos pelos dois objetos “dac” (um em estéreo e o outro em quadrifônico)



## 

## *1.3 Outros projetos voltados para espacialização sonora*

Diversos fabricantes de sistemas de produção audiovisual e instituições de pesquisa tem produtos e projetos de pesquisa na área de espacialização sonora. Seguem alguns exemplos:

* **Spat:** O Spat permite que compositores, artistas sonoros e engenheiros de áudio criem e manipulem cenas sonoras espaciais. O Spat também é uma livraria de externals mas para Max/MSP (“rival” do Pure Data, <https://cycling74.com/products/max>). O software foi desenvolvido pelo IRCAM (Institut de Recherche et Coordination Acoustique/Musique; em português, Instituto de Pesquisa e Coordenação de Música e Acústica, é uma instituição francesa dedicada à pesquisa e à criação de música contemporânea) ( <https://forum.ircam.fr/projects/detail/spat/> ). Este software é bem parecido com o AUDIENCE.
* **Dolby Atmos:** O Dolby Atmos é uma tecnologia que aprimora o som surround, adicionando uma verticalização ao som espacial. O surround padrão simula o áudio 3D cercando o usuário horizontalmente, enquanto eletrônicos com a tecnologia Dolby Atmos prometem criar uma espécie de bolha virtual em volta do ouvinte, passando a ideia de que o som vem a partir de todos os lados. Em outras palavras, além de dar ao usuário uma noção de qual direção vem o som, o Dolby Atmos permite sentir se o mesmo vem de baixo ou de cima, criando uma perspectiva mais realista e aumentando a noção espacial. ( <https://www.dolby.com/technologies/dolby-atmos/> )
* **Facebook 360 Spatial Workstation:** O Facebook 360 Spatial Workstation é um pacote de software para projetar áudio espacial para vídeo 360 e VR cinematográfico. Inclui plugins para DAWs populares (Uma estação de trabalho de áudio digital, DAW, é um dispositivo de hardware ou software usado para compor, produzir, gravar, mixar e editar áudio, como por exemplo músicas, falas e efeitos sonoros. DAWs facilitam a mixagem de várias fontes de som em um grid baseado no tempo.), um reprodutor de vídeo 360 com sincronização de tempo e utilitários para ajudar a projetar e publicar áudio espacial em uma variedade de formatos (<https://facebook360.fb.com/spatial-workstation/> )
* **SoundScape Renderer:** O SoundScape Renderer (SSR) é uma ferramenta para reprodução de áudio espacial em tempo real que fornece uma variedade de algoritmos de renderização, por exemplo, Wave Field Synthesis, Ambisonics e técnicas binaurais. O SSR está atualmente disponível para GNU / Linux e Mac OS X e foi lançado como software de código aberto sob a GNU General Public License (GPL). É desenvolvido no Laboratório de Qualidade e Usabilidade / TU Berlin (<http://qu.tu-berlin.de/>) e no Institut für Nachrichtentechnik / Universität Rostock (<http://www.int.uni-rostock.de/>).
* **Adobe Premiere Pro:** Também é possível manipular áudio espacial no software de edição de vídeo Adobe Premiere Pro.
* **G’Audio Works:** é um plugin de áudio espacial para estações de trabalho de áudio digital (DAWs). O software é apenas para Mac que requer OS 10.10 e superior. Ele aproveita a mixagem baseada em objetos para a colocação detalhada do som em um ambiente 3D.

## *1.4 Objetivos*

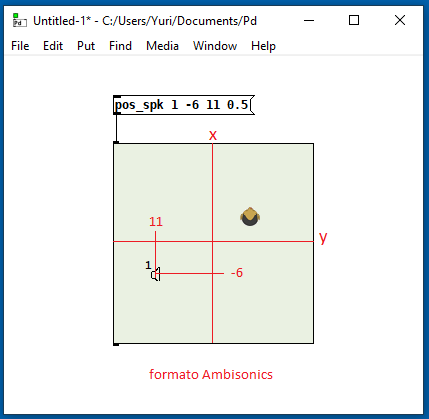
O AUDIENCE fornece duas interfaces para fazer a criação e manipulação das cenas sonoras. O objeto oa\_l1\_gui instancia uma interface onde é possível criar/manipular cenas espaciais sonoras com objetos que emitem sons (fonte sonoras). E o objeto oa\_l4\_gui instancia uma interface onde é possível criar/manipular cenas espaciais sonoras com auto-falantes. Porém ainda não havia uma interface onde fosse possível trabalhar tanto com fontes sonoras e auto-falantes na mesma cena.

Figura 8 - Ilustração da interface já existente de cena L4 (cena de alto-falantes. Figura da esquerda), interface de cena L1 (cena acústica com fontes sonoras, Figura da direita) e de uma nova interface que integra em uma mesma cena camadas de objetos emissores de som de L1, fontes, e de L4, alto-falantes (Figura do meio).



Além disso, em nenhuma das interfaces era possível aumentar o número de ouvintes, sempre há apenas um ouvinte. Seria interessante também poder configurar o número de ouvintes. Outra questão é que as interfaces atuais posicionam os elementos da cena segundo um plano no formato Ambisonics. No plano ambisonics, onde seria o eixo y é o eixo x, e onde seria o eixo x é o eixo y com sentido invertido. Isso pode confundir o usuário que não está acostumado com este formato ambisonics. Então também seria interessante fornecer para o usuário poder escolher posicionar os elementos em formato ambisonics ou cartesiano.

Figura 7 - Formato Ambisonics. A interface L4 recebe a mensagem *pos\_spk <nº do auto falante> <x> <y> <z>*, que em seguida configura o posição do objeto segundo esta mensagem passada. Note que, em relação ao plano cartesiano normal, os eixo estão trocados e o eixo y está com sentido invertido. Este é o formato Ambisonics.



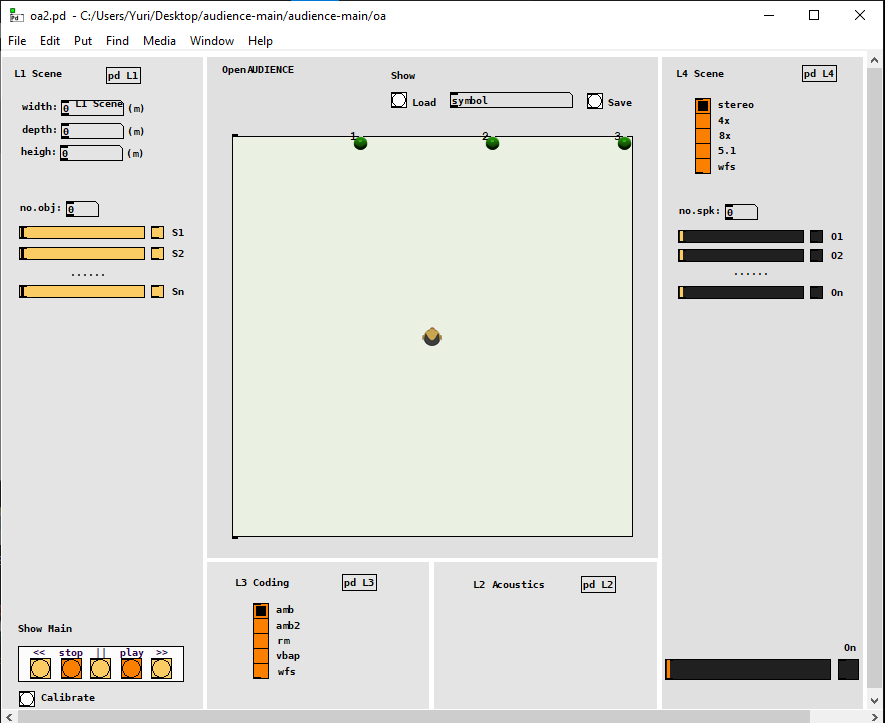
Então o objetivo do primeiro semestre de projeto foi implementar uma nova interface gráfica que forneça mais recursos e opções ao usuário. Esta interface deverá então:

* integrar os níveis L1 e L4 do AUDIENCE de maneira a possibilitar a produção virtual de uma cena sonora composta de alto-falantes e fontes sonoras
* Poder configurar o número de ouvintes da cena
* Poder escolher configurar o posicionamento dos elementos da cena segundo um plano ambisonics ou cartesiano
* Poder escolher exibir/ocultar as fontes/auto-falantes

Terminada esta interface que integra o tanto L1 como L4, o objetivo da segunda fase do projeto foi

* Tornar a interface reconfigurável, ou seja, ser possível reajustar as dimensões e o número de elementos da cena.
* Tornar a interface de modo que seja o Audience em sua totalidade: O usuário não precisará mais compor ele mesmo as conexões entre os objetos do Audience para poder criar uma cena sonora espacial, como se pode ver na figura 6. A interface fará tudo isso automaticamente, dinamicamente.

Figura 9: Rascunho no pure data da interface com parâmetros reconfiguráveis. Nesta interface, o usuário poderá configurar as dimensões da sala, número de ouvintes, número de fontes sonoras, número de auto-falantes, volume dos elementos, formato de codificação, etc. Então agora a interface agora deverá ser reconfigurável, e para isso será necessário implementar mecanismos de programação dinâmica no pure data.



No Pure Data, a programação dinâmica se dá quando há construção/reconstrução de objetos e conexões em tempo de execução

Figura 10: Exemplo de patch que faz o uso de programação dinâmica. Os objetos e as conexões são criadas automaticamente em tempo de execução.

Além disso, prevê-se a consolidação de uma distribuição revisada do pacote do software AUDIENCE.

## *1.5 Cronograma*

A primeira fase do projeto teve o seguinte cronograma:

1. Estudo geral sobre o AUDIENCE: o que é, história, funcionamento, arquitetura, etc.
2. Estudo do Pure Data
3. Estudo de como fazer um external Pure Data
4. Estudo dos algoritmos dos objetos e patches AUDIENCE, em especial, os objetos oa\_l1\_gui e oa\_l4\_gui, que geram as interfaces de fontes sonoras e auto-falantes respectivamente. (oa\_l1\_gui e oa\_l4\_gui, correspondem ao audce\_L1\_gui e audce\_L4\_gui (figura 4), só que na versão open source do AUDIENCE)
5. Elaboração e implementação da nova interface que integra L1 e L4

A segunda fase do projeto foi a construção dos mecanismos dinâmicos para tornar a interface reconfigurável.

# 2 Revisão bibliográfica

Primeiramente foi feito o estudo da literatura publicada relativa ao AUDIENCE e suas fases anteriores de desenvolvimento para obter uma compreensão mais geral sobre o software: o que é o AUDIENCE, quem criou, qual a sua história, como funciona, qual sua arquitetura, etc.

O passo seguinte foi estudar o Pure Data, ambiente onde é usado o AUDIENCE. A base de estudos foi a documentação do Pure Data, os patches de tutoriais que vem com o Pure Data e até mesmo as aulas de Computação Sônica desse semestre.

É também necessário estudar como se faz um external, afinal os objetos do AUDIENCE são *externals*, ou seja, objetos para Pure Data que não vem por padrão com o programa. Depois disso, foi feito um estudo do algoritmo dos objetos do AUDIENCE que instanciam as interfaces L1 e L4 (oa\_l1\_gui e oa\_l4\_gui) para saber como eles funcionam e aí finalmente poderá ser iniciado a elaboração do algoritmo da nova interface.

Não há muito material disponível sobre como escrever um external. As únicas fontes de estudo é o artigo “How to write an external for Pure Data” de de ZMOLNIG, Johannes M, uma série de videoaulas no YouTube de Rafael Hernandez (<https://m.youtube.com/user/cheetomoskeeto> ) e o fórum na internet dos desenvolvedores para Pure Data.

Antes de prosseguir para a próxima seção deste relatório, é fortemente recomendado a leitura do Anexo A - “Como fazer um external Pure Data”, trecho retirado do artigo “How to write an external for Pure Data” de ZMOLNIG, Johannes M, que mostra o passo-a-passo da construção de um external. Fica difícil a compreensão do trabalho sem que o leitor tenha uma noção básica da estrutura de um external, pois os objetos AUDIENCE são externals.

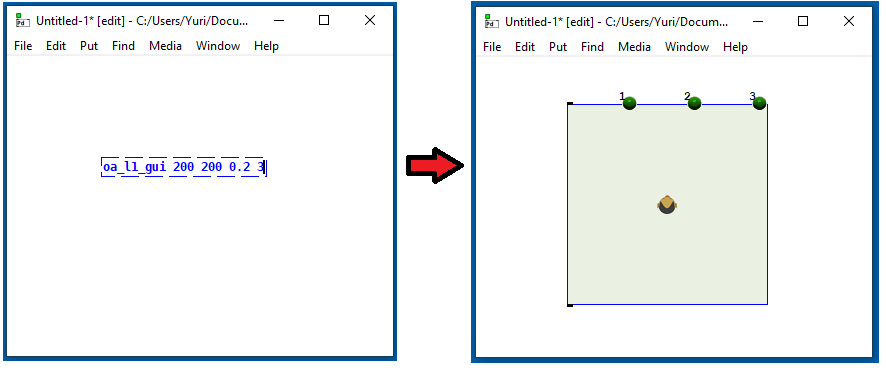
# 3 Estudo das interfaces L1 e L4

## *3.1 Argumentos, entrada e saída das interfaces L1 e L4*

Agora já possuindo uma noção de como se cria um external, o próximo passo foi entender o algoritmo dos objetos das interfaces L1 e L4 do AUDIENCE. A interface do AUDIENCE tem como finalidade permitir ao usuário configurar os atributos da cena sonora:

* comprimento da sala
* largura da sala
* escala
* número de fontes/auto-falantes
* posição das fontes/auto-falantes e ouvinte

O comprimento da sala, largura da sala, escala e número de fontes ou auto-falantes devem ser passados como argumento no momento de criação do objeto.

Figura 9- Como instanciar uma interface L1 (oa\_l1\_gui). Cria-se uma caixa de objeto. Coloca-se, nesta caixa de objeto, o nome do objeto de interface do AUDIENCE (oa\_l1\_gui) e na sequência os argumentos do objeto separados por espaço: comprimento da sala, largura da sala, escala e número de fontes. No exemplo dessa figura, foi instanciado uma cena sonora de dimensões 200x200 e escala 0.2. O número de fontes é de três.

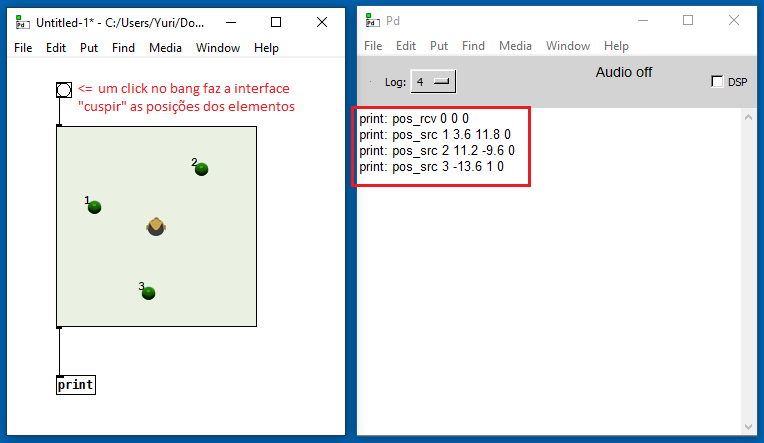
Na interface, o usuário pode definir as posições dos elementos da cena sonora clicando sobre eles e os arrastando para a posição desejada ou enviando uma mensagem pelo inlet da interface com a seguinte sintaxe:

* pos\_src <nº da fonte> <x> <y> <z>
* pos\_spk <nº do auto-falante> <x> <y> <z>
* pos\_rcv <x> <y> <z>

A interface “cospe” as posições dos elementos da cena pelo outlet quando recebe um bang.

Assim essas posições poderão ser enviadas para os outros objetos da livraria AUDIENCE para poderem espacializar a cena sonora criada na interface.

Figura 10 - Envio das posições dos elementos da cena



## *3.2 O algoritmo das interfaces*

| **Algoritmo 2** - espaço de dados (struct \_oa\_l1\_gui) e funções do oa\_l1\_gui. Vale ressaltar que os algoritmos do oa\_l1\_gui e oa\_l4\_gui são praticamente idênticos. Os desenhos da interface se dão pela linguagem Tcl. |
| --- |
| typedef struct \_oa\_l1\_gui  {  t\_object x\_obj;  t\_glist \*x\_glist;  t\_inlet \*\*x\_inlets;  t\_outlet \*\*x\_outlets;  t\_int x\_num\_src; //número de fontes sonoras  t\_int \*\*x\_src; //ponteiro p/ matriz que guarda as posições das fontes  t\_int x\_rcv[3]; //vetor que guarda a posição xyz do ouvinte  t\_int x\_type\_selected; //tipo do objeto selecionado  t\_int x\_selected; //guarda o estado selecionado  t\_int x\_nselected; //índice do item selecionado  t\_int x\_height; //altura da interface  t\_int x\_width; //largura da interface  t\_float x\_pixelsize; //guarda o valor de 1 pixel em metros  } t\_oa\_l1\_gui;  t\_widgetbehavior oa\_l1\_gui\_widgetbehavior;  static t\_class \*oa\_l1\_gui\_class;  t\_float meter[3];  //funções de conversão  static void oa\_l1\_gui\_convert\_pixel\_to\_meter(t\_oa\_l1\_gui \* x, t\_int \* pixel, t\_float \* meterr) {...}  static void oa\_l1\_gui\_convert\_meter\_to\_pixel(t\_oa\_l1\_gui \* x, t\_int \* pixel, t\_float \* meterr) {...}  //envia a posicao do ouvinte  static void oa\_l1\_gui\_send\_pos\_rcv(t\_oa\_l1\_gui \*x) {...}  //envia a posicao da fonte  static void oa\_l1\_gui\_send\_pos\_src(t\_oa\_l1\_gui \*x, int sel) {...}  //altera a posição do ouvinte  static void oa\_l1\_gui\_pos\_rcv(t\_oa\_l1\_gui \*x, t\_symbol \*s, int argc, t\_atom \*argv) {...}  //altera a posição da fonte  static void oa\_l1\_gui\_pos\_src(t\_oa\_l1\_gui \*x, t\_symbol \*s, int argc, t\_atom \*argv) {...}  //envia a posicao de todos elementos da cena quando recebe um bang  static void oa\_l1\_gui\_bang(t\_oa\_l1\_gui \*x) {...}  /\*----------- funções de desenho da interface e dos elementos da cena---------------------\*/  //As funções de desenho da interface usam a linguagem Tcl  //função que desenha a interface pela primeira vez  static void oa\_l1\_gui\_draw\_new(t\_oa\_l1\_gui \*x, t\_glist \*glist) {...}  //atualiza o desenho se houve alguma mudança na cena  static void oa\_l1\_gui\_draw\_update(t\_oa\_l1\_gui \* x, t\_glist \* glist) {...}  static void oa\_l1\_gui\_draw\_move(t\_oa\_l1\_gui \*x, t\_glist \*glist) {...}  static void oa\_l1\_gui\_draw\_erase(t\_oa\_l1\_gui\* x,t\_glist\* glist) {...}  static void oa\_l1\_gui\_draw\_select(t\_oa\_l1\_gui\* x,t\_glist\* glist) {...}  /\* ------funções que definem o comportamento gráfico da interface-----------------\*/  static void oa\_l1\_gui\_getrect(t\_gobj \*z, t\_glist \*owner, int \*xp1, int \*yp1, int \*xp2, int \*yp2) {...}  static void oa\_l1\_gui\_save(t\_gobj \*z, t\_binbuf \*b) {...}  static void oa\_l1\_gui\_properties(t\_gobj \*z, t\_glist \*owner) {...}  static void oa\_l1\_gui\_select(t\_gobj \*z, t\_glist \*glist, int selected) {...}  static void oa\_l1\_gui\_vis(t\_gobj \*z, t\_glist \*glist, int vis) {...}  static void oa\_l1\_gui\_dialog(t\_oa\_l1\_gui \*x, t\_symbol \*s, int argc, t\_atom \*argv) {...}  static void oa\_l1\_gui\_delete(t\_gobj \*z, t\_glist \*glist) {...}  static void oa\_l1\_gui\_displace(t\_gobj \*z, t\_glist \*glist, int dx, int dy) {...}  static void oa\_l1\_gui\_motion(t\_oa\_l1\_gui \*x, t\_floatarg dx, t\_floatarg dy) {...}  static int oa\_l1\_gui\_click(t\_gobj \*z, struct \_glist \*glist, int xpix, int ypix, int shift, int alt, int dbl, int doit) {...}  //construtor da interface  static t\_oa\_l1\_gui \*oa\_l1\_gui\_new(t\_symbol \*s, int argc, t\_atom \*argv) {...}  //destrutor da interface  void oa\_l1\_gui\_free(t\_oa\_l1\_gui \*x) {...}  void oa\_l1\_gui\_setupgui(void) {...}  //construtor da interface  void oa\_l1\_gui\_setup(void) {...} |

# 4 Implementação da interface L1/L4 (oa\_l1l4\_gui)

Agora sabendo mais a fundo como funciona a interface. Pode ser iniciado a elaboração da nova interface oa\_l1l4\_gui. Como o que será feito é uma nova interface que segue o mesmo modelo das já existentes L1 e L4 mas tendo um incremento de funcionalidades, o primeiro passo então foi criar um novo script mas com exatamente o mesmo algoritmo da interface L1 (Algoritmo 2. Poderia ser o algoritmo da interface L4, tanto faz, pois ambos são muito semelhantes)

Conforme já discutido na secção 1.2, a nova interface deverá permitir:

* criação de cenas sonoras tanto com fontes sonoras como com auto-falantes
* configurar quantos ouvintes terão na cenas
* configurar qual formato de escala será adotada: ambisonics ou cartesiano
* opção de exibir/ocultar as fontes/auto-falantes

No momento de criação do objeto da interface, além do número de fontes, também deverá ser passado como argumento o número de autofalantes, número de ouvintes e o formato da escala. A sintaxe para a criação do objeto escolhida foi essa:

* oa\_l1l4\_gui <altura> <largura> <n de fontes> <n de auto-falantes> <n de ouvintes> <formato da escala>

Foi decidido que o número de ouvintes e o formato da escala serão argumentos opcionais. Se não forem especificados, assumem-se valores padrões: um ouvinte e formato Ambisonics. Para o argumento da escala, Ambisonics ou cartesiano, será passado um caractere:

* “a” para escolher a escala Ambisonics
* “c” para escolher a escala cartesiana

## *4.1 O espaço de dados da nova interface*

Agora, o espaço de dados do objeto da nova interface oa\_l1l4\_gui, deverá acomodar:

* além do número de fontes, o número de auto-falantes e ouvintes: **x\_num\_spk, x\_num\_rcv**
* ponteiros para matrizes dinâmicas para também guardar, além das posições das fontes, as posições dos alto-falantes e ouvintes: **\*\*x\_spk, \*\*x\_rcv.**
* o formato da escala, Ambisonics ou cartesiano: **x\_format\_scale**
  + x\_format\_scale = “a” Ambisonics
  + x\_format\_scale = “c” Cartesiano
* o estado de exibição das fontes, habilitado ou não. A exibição das fontes será habilitada quando a interface receber a mensagem “l1on” e desabilitada quando receber a mensagem “l1off”: **x\_l1\_display**.
  + x\_l1\_display = 1 Exibição habilitada
  + x\_l1\_display = 0 Exibicao desabilitada
* o estado de exibição dos auto-falantes: habilitado ou não . A exibição dos alto-falantes será habilitada quando a interface receber a mensagem “l4on” e desabilitada quando receber a mensagem “l4off”: **x\_l4\_display**
  + x\_l4\_display = 1 Exibição habilitada
  + x\_l1\_display = 0 Exibicao desabilitada

| **Algoritmo 3** - Espaço de dados do objeto oa\_l1l4\_gui |
| --- |
| typedef struct \_oa\_l1l4\_gui  {  t\_object x\_obj;  t\_glist \*x\_glist;  t\_inlet \*\*x\_inlets;  t\_outlet \*\*x\_outlets;  t\_symbol \*x\_format\_scale; //formato da escala. ‘a’- Ambisonics, ‘c’ - cartesiano  t\_int x\_num\_src; //número de fontes  t\_int x\_num\_spk; //número de auto-falantes  t\_int x\_num\_rcv; //número de ouvintes  t\_int \*\*x\_src; //pont p/ matriz din. guarda posições fontes  t\_int \*\*x\_spk; //pont p/ matriz din. guarda posições auto-falantes  t\_int \*\*x\_rcv; //pont p/ matriz din. guarda posições ouvintes  t\_int x\_l1\_display; //estado das fontes: exibidas ou ocultados  t\_int x\_l4\_display; //estado dos auto-falantes: exibidos ou ocultados  t\_int x\_type\_selected;  t\_int x\_selected;  t\_int x\_nselected;  t\_int x\_height;  t\_int x\_width;  t\_float x\_pixelsize;  } t\_oa\_l1l4\_gui; |

## *4.2 O construtor da nova interface*

Depois de arrumar o espaço de dados com novas variáveis, e hora de implementar a integração L1 e L4. A primeira modificação foi no método oa\_l1\_gui\_new. Este método realiza:

* pega os valores passados como argumentos no momento de criação do objeto (altura, largura, escala, número de fontes), faz o tratamento de erros e atribui as variáveis correspondentes no espaço de dados
* cria a matriz dinâmica (\*\*x\_src) que armazenam as posições x y z das fontes
* cria o matriz dinâmica (\*x\_rcv) que armazenam a posição x y z do ouvinte
* atribui as posições das fontes e do ouvinte para suas estruturas de dados correspondentes para depois a funcao oa\_l1\_gui\_draw\_new recuperar essas posições e fazer o desenho da interface

O construtor oa\_l1\_gui\_new, agora chamado oa\_l1l4\_gui\_new (pois fizemos modificações e ajustes em cima do método oa\_l1\_gui\_new, que agora é um novo método para a nova interface L1/L4) sofreu modificações para poder ser feita a integração L1 e L4:

* Além da altura, largura, escala, numero de fontes, oa\_l1l4\_gui\_new tambem recebe o numero de auto-falantes, número de ouvintes e formato da escala
* Além da matriz que guarda as posições das fontes, também cria a matrizes que guardam as posições dos auto-falantes (\*\*x\_spk) e dos ouvintes (\*\*x\_rcv)

## *4.3 Os métodos da nova interface*

Mais métodos foram sendo modificados e criados para que a interface obtenha as funcionalidades desejadas.

* Modificações nos métodos que fazem o desenho da interface para que possam desenhar, além das fontes, os auto-falantes e os ouvintes
* Modificações nos métodos de comportamento gráfico (salvamento; seleção e clique nos objetos da cena)
* Modificação/criação de métodos de envio e recebimento de mensagens de posição de elementos. São nos métodos de recebimento, onde foi implementado a solução:
  + modificação no método oa\_l1\_gui\_send\_pos\_src() (agora oa\_l1l4\_gui\_send\_pos\_src): envia a posição de determinada fonte.
  + criado o método oa\_l1l4\_gui\_send\_pos\_spk(): envia a posição de determinado auto-falante
  + criado o método oa\_l1l4\_gui\_send\_pos\_rcv(): envia a posição de determinado ouvinte
  + modificação no método oa\_l1\_gui\_pos\_src() (agora oa\_l1l4\_gui\_pos\_src): método que pega uma mensagem recebida contendo a posição de uma fonte (pos\_src <n> <x> <y> <z>) e posiciona a fonte escolhida nesta posição passada segundo o formato da escala configurado, Ambisonics ou cartesiano. (Veja o algoritmo no Anexo B)
  + modificação no método oa\_l1\_gui\_pos\_rcv() (agora oa\_l1l4\_gui\_pos\_rcv): método que pega uma mensagem recebida contendo a posição de uma ouvinte (pos\_rcv <n> <x> <y> <z>) e posiciona o ouvinte escolhido nesta posição passada segundo o formato da escala configurado, Ambisonics ou cartesiano. (Veja o algoritmo no Anexo B)
  + criado o método oa\_l1l4\_gui\_pos\_spk(): método que pega uma mensagem recebida contendo a posição de um alto-falante (pos\_spk <n> <x> <y> <z>) e posiciona o alto-falante escolhido nesta posição passada segundo o formato da escala configurado, Ambisonics ou cartesiano. (Veja o algoritmo no Anexo B)

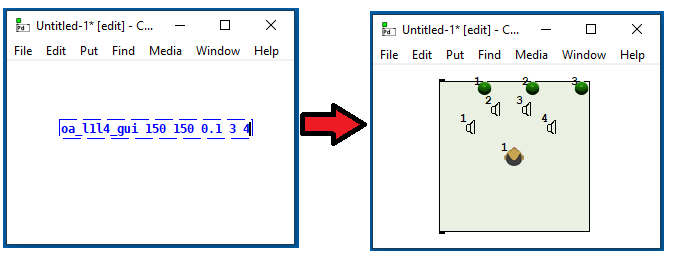
* Métodos para exibir/ocultar L1/L4 (veja os algoritmos desses métodos no Anexo C):
  + criado o método oa\_l1l4\_gui\_l1on(): ao receber a mensagem “l1on” e a exibição das fontes não estiver habilitada (ou seja, x->x\_l1\_display == 0), as imagens das fontes são recriadas
  + criado o método oa\_l1l4\_gui\_l1off(): ao receber a mensagem “l1off” e a exibição das fontes estiver habilitada (ou seja, x->x\_l1\_display == 1), as imagens das fontes são deletadas
  + criado o método oa\_l1l4\_gui\_l4on(): ao receber a mensagem “l4on” e a exibição dos auto-falantes não estiver habilitada (ou seja, x->x\_l4\_display == 0), as imagens dos auto-falantes são recriadas
  + criado o método oa\_l1l4\_gui\_l4off():ao receber a mensagem “l4off” e a exibição dos auto-falantes estiver habilitada (ou seja, x->x\_l4\_display == 1), as imagens dos auto-falantes são deletadas

# 5 Resultados da primeira fase

Com a implementação feita conforme explicado na secção 4, o novo objeto oa\_l1l4\_gui atingiu os objetivos almejados nesta primeira fase do projeto.

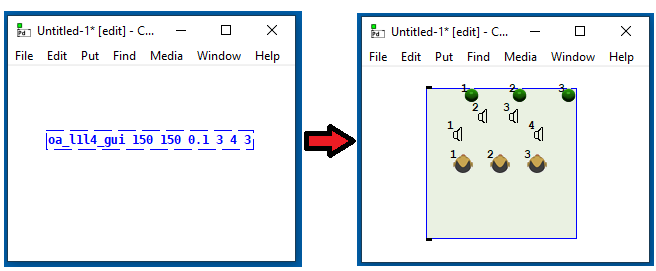
## *5.1 Integração L1 e L4:*

Figura 11 - O novo objeto AUDIENCE oa\_l1l4\_gui instancia uma nova interface onde é possível criar uma cena espacial sonora tanto com fontes sonoras como com auto-falantes. Neste exemplo é criado uma cena de dimensões 150x150 com escala 0.1, ou seja, representa uma sala quadrada com 15 x 15 metros, possuindo 3 fontes e 4 alto-falantes

**

## *5.2 Número de ouvintes configurável*

Agora também é possível haver mais de um ouvinte na cena. O número de ouvintes pode ser passado por argumento. É um argumento opcional. Se o número de ouvintes não for especificado, apenas um ouvinte é instanciado (como na Figura 9).

Figura 12 - Instanciando oa\_l1l4\_gui com três ouvintes

## *5.3 Formato da escala configurável (Ambisonics ou cartesiano)*

Agora é possível também configurar, por argumento, o tipo da escala, Ambisonics (“a”) ou cartesiano (“c”). É um argumento opcional. Se não for especificado na passagem de argumentos o formato da escala (como nas figuras 9 e 10), será adotado, por padrão, a escala Ambisonics.

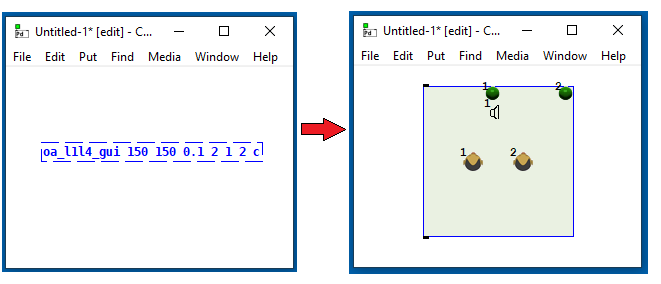
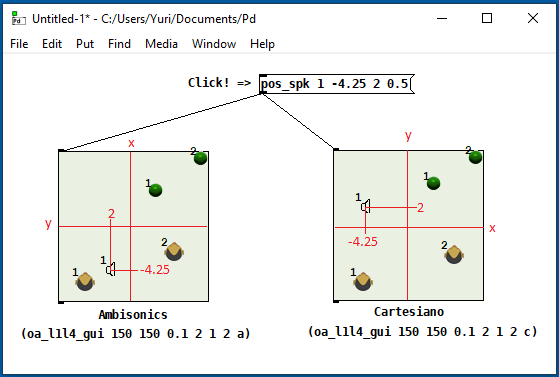
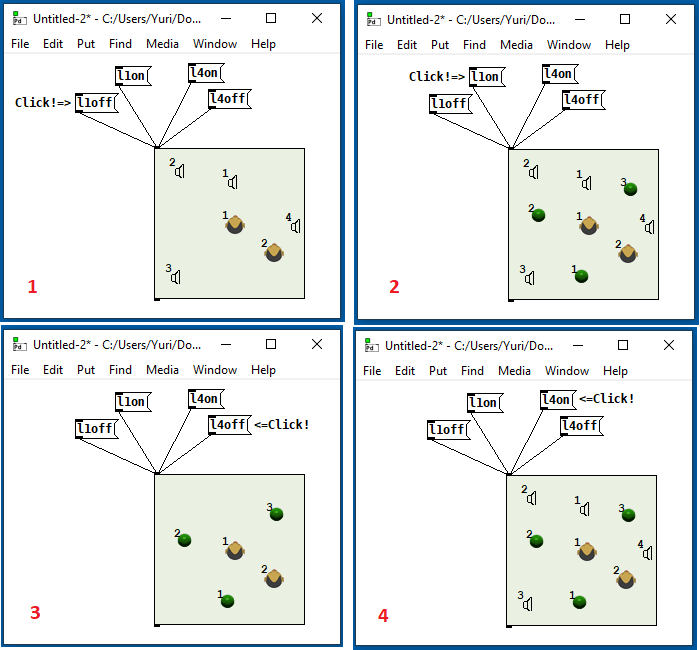
Figura 13 - Instanciação de uma interface L1 e L4 de 150x150, escala 0.1 (ou seja, 15x15m), 2 fontes, 1 auto-falante, 2 ouvintes e o formato da escala cartesiano.

Figura 14 - Comparação entre interfaces de formato Ambisonics e cartesiano. A interface à esquerda está no formato Ambisonics, enquanto que a interface da direita, está no formato cartesiano. Ambas recebem uma mensagem de posição para o alto-falante 1, que ambas possuem. Note a diferença de posicionamento:

## *5.4 Função de exibir/ocultar*

A exibição das fontes e auto-falantes podem ser habilitados e desabilitados passando-se as mensagens: “l1on” (exibir fontes), ‘l1off’ (ocultar fontes), “l4on” (exibir auto-falantes) e “l4off” (ocultar auto-falantes)

Figuras 15 - Demonstração passo-a-passo da funcionalidade de exibir/ocultar L1/L4. 

# 6 Segunda fase: interface reconfigurável

Agora com uma interface onde se pode compor cenas sonoras espaciais com L1 e L4 e também com mais configurações, o objetivo da segunda fase foi aprimorar a interface ainda mais tornando-a reconfigurável, ou seja, depois de instanciada, é possível alterar o número de fontes sonoras, número de alto-falantes, números de ouvintes, dimensões da sala (comprimento, largura e altura). E também outro objetivo foi que a interface possibilite ao usuário criar, modificar, sua cena sonora espacial sem ter que criar um patch com os objetos do Audience para isso (como mostrado na figura 6). Tudo isso dependerá de patches que façam o uso de programação dinâmica para poderem construir os objetos em tempo de execução. Todo o funcionamento do AUDIENCE será centralizado e automatizado por essa interface única, assim facilitando muito o uso do software.

A interface será dividida em 6 partes, na verdade subpatches (um subpatch é um patch que fica dentro de outro patch), mas vamos chamar de módulos. Cada módulo tem função na interface:

* Módulo da cena sonora: parte da interface onde se visualiza e edita as posições dos elementos da cena sonora (clicando e arrastando)
* Módulo L1: parte da interface onde se dá:
  + a configuração, em metros, das dimensões da cena sonora (comprimento, largura e altura);
  + opção de exibição das fontes sonoras;
  + controle de volumes das fontes sonoras (L1)
  + matriz L1 (figura 17). A matriz L1 é o subpatch onde se dá a construção dos players para cada fonte sonora. O player é o objeto onde se carrega o áudio de uma fonte sonora.
* Módulo L2: parte da interface que calcula a propagação e reflexões dos sons
* Módulo L3: parte da interface onde se configura a opção de codificação e onde se dá a construção da matriz L3 (figura 24). A matriz L3 é o subpatch que faz a codificação dos sons.
* Módulo L4: parte da interface onde se dá configuração do modo de reprodução, opção de exibição dos alto falantes, controle de volumes dos alto falantes

Figura 16: Os Módulos (partes) da interface reconfigurável

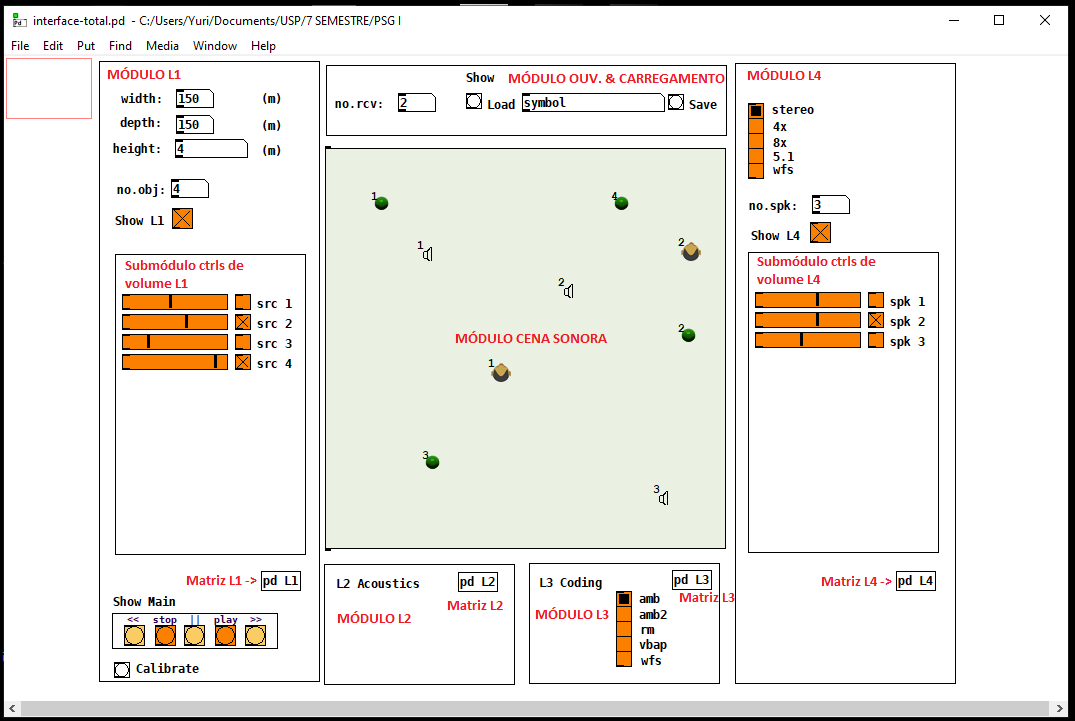
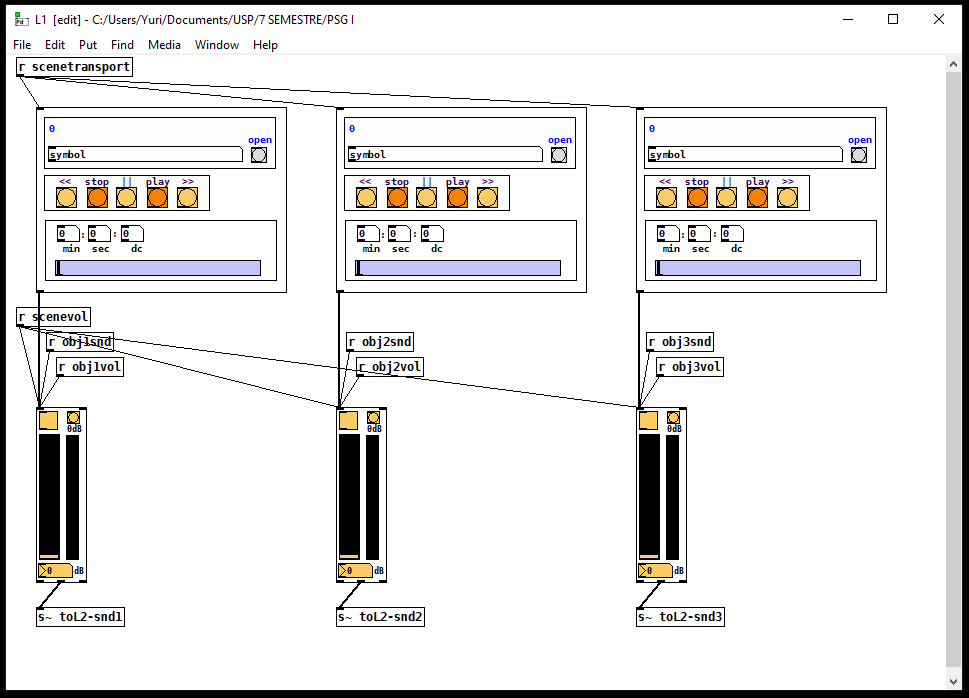


Figura 17: Patch da matriz L1 aberto. A matriz L1 é um subpatch encontrado no módulo L1 (está indicado na figura anterior). Neste subpatch, encontram-se os players de cada fonte sonora presente na cena. São nestes players que se carregam os sons das fontes sonoras. Cada player manda o sinal de áudio para um controle de volume, e esse controle de volume também recebe valores de volume dos controles de volume da interface e envia o sinal para a matriz L3 que faz a codificação sonora. Neste exemplo, é uma matriz L1 para uma cena de 3 fontes sonoras



## *6.1 Princípios básicos da programação dinâmica no Pure Data*

Antes de apresentar os mecanismos de programação dinâmica da interface desenvolvidos na segunda fase do projeto, é necessário compreender alguns princípios básicos de como funciona a programação dinâmica no Pure Data. Basicamente, a programação dinâmica no Pure Data se dá através de comandos dentro de um patch que quando acionados, constroem/deletam objetos e conexões em um outro patch. Existem basicamente três tipos de comandos:

* comando de construção de objetos. Este tipo de comando constrói os elementos do patch: objetos, sliders, botões, botões, mensagens, etc. Exemplo da sintaxe do comando que constrói objetos no patch:
  + pd-<nome-do-patch> obj <x> <y> <nome-do-objeto> <argumentos-do-objeto>
* comando de construção de conexão entre objetos. Este comando constrói as conexões entre os objetos do patch. E possui a seguinte sintaxe:
  + pd-<nome-do-patch> connect <índice-do-obj1> <índice-do-outlet-obj1> <índice-do-obj2> <índice-do-inlet-obj2>

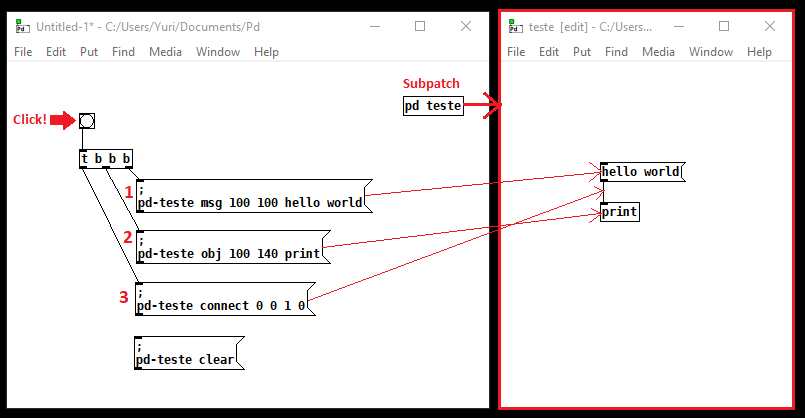
Apenas para relembrar: o outlet é a “saída” do objeto, ou seja, por onde o dado sai do objeto. E o “inlet” é a entrada do objeto, ou seja, por onde entra o dado no objeto.

Para poder conectar os objetos é necessário saber os seus índices. Cada objeto dentro de um patch recebe um número identificador, isso é o índice. O índice depende do momento quando o objeto foi criado e da posição do objeto dentro do patch.

* comando de limpeza de patch. Este comando deleta todos os objetos presentes no patch. Possui a seguinte sintaxe:
  + pd-<nome-do-patch> clear

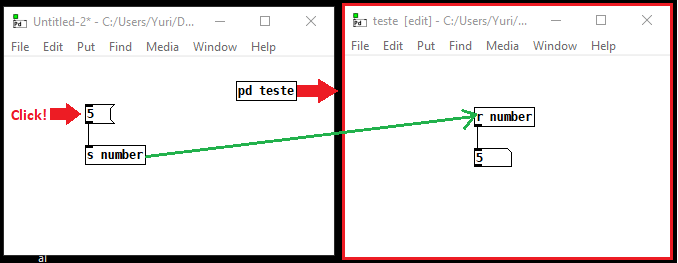
Não existe um comando onde se possa apenas deletar um objeto em específico. Então para poder reconstruir um patch, o único jeito é primeiramente deletar todos os objetos presentes no patch e reconstruí-lo novamente.

Figura 18: Exemplo de patch que faz o uso de programação dinâmica para criar em tempo de execução o subpatch teste. Ao clicar no botão indicado da figura, o objeto trigger é ativado, ativando, em ordem: o comando de construção do objeto de mensagem hello world (1), o comando de construção do objeto print (2) e o comando de construção da conexão entre a mensagem e o objeto print (3). Se o comando “pd-teste clear” for disparado clicando nele, todo conteúdo do subpatch “teste” será deletado.



Outra questão importante de se entender é a o envio e recebimento de dados entre patches. Isso pode se dar através dos objetos “send” e “receive”.

Figura 19: Exemplo de uso dos objetos “send” e “receive”. No patch à esquerda, ao clicar no objeto de mensagem com o valor 5, o valor é enviado ao objeto send “s number” que envia o valor ao outro patch que contém o objeto receive number” que recebe o valor



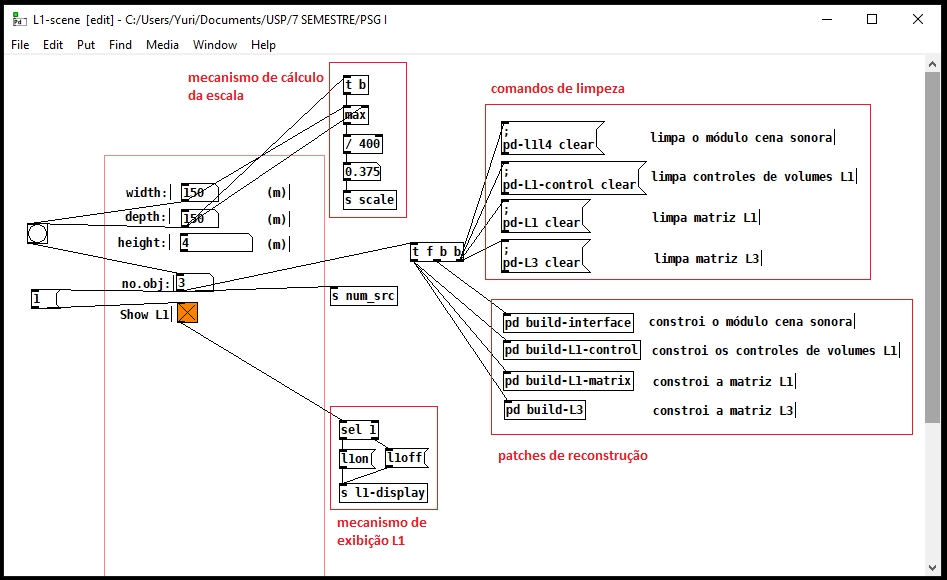
## *6.2 Módulos da interface reconfigurável e seus mecanismos de auto-construção*

### *6.2.1 Módulo L1*

Quando o usuário modifica o número de fontes sonoras, é deletado tudo que estiver no módulo da cena sonora, no submódulo de controle de volumes L1, na matriz L1 e na matriz L3. Depois disso, são acionados 4 sub-patches que fazem a reconstrução dessas partes para se ajustar ao novo número de fontes sonoras.

* build-interface: subpatch responsável pela reconstrução do módulo da cena sonora
* build-L1-control: subpatch responsável pela reconstrução dos controles de volumes. Também é acionado quando se altera as o número de alto falantes, ou o número de ouvintes
* build-L1-matrix: subpatch responsável pela reconstrução da matriz L1
* build-L3-matrix: subpatch responsável pela reconstrução da matriz L3

Figura 20: Módulo L1. O Módulo L1 tem as seguintes funções: calcular a escala da cena, quando a largura ou comprimento for alterado; reconstruir a cena, os controles de volume, a matriz L1 e L3, quando as dimensões da cena ou o número de objetos são alterados; e enviar a mensagem de exibição/ocultamento das fontes sonoras.

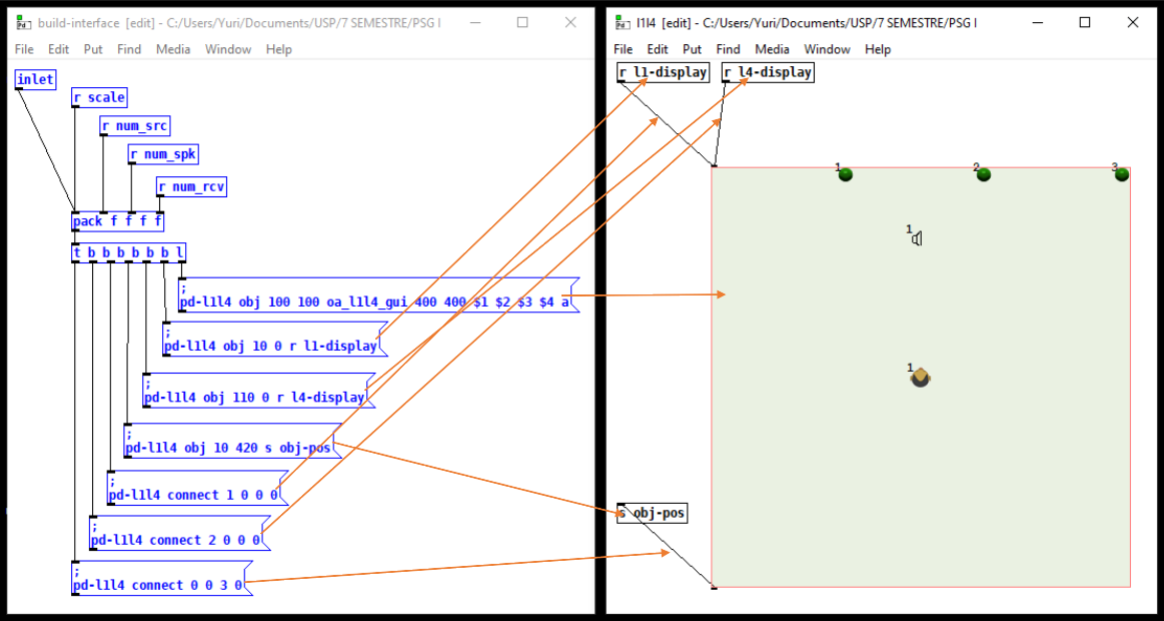


#### *6.2.1.1 Subpatch build-interface*

Este subpatch é responsável pela reconstrução do módulo da cena sonora. Veja a figura seguinte: ele coleta os valores da escala da cena, número de fontes, número de alto-falantes e número de ouvintes com o objeto “pack”. Quando acionado, ou seja, quando é alterado o número de fontes sonoras, o objeto pack ativa o trigger que executa em ordem:

* a construção do objeto oa\_l1l4\_gui, ou seja, a interface que foi construída na primeira fase do projeto
* a construção dos objetos que enviam os comandos de exibição de L1 e L4 (objetos “r l1-display e l4-display”, que podem serem vistos na figura seguinte)
* a construção do objeto que envia a posição de determinado objeto para a matriz L3 (objeto “s obj-pos”, que pode ser visto na figura seguinte)
* a construção das devidas conexões entre os objetos do módulo

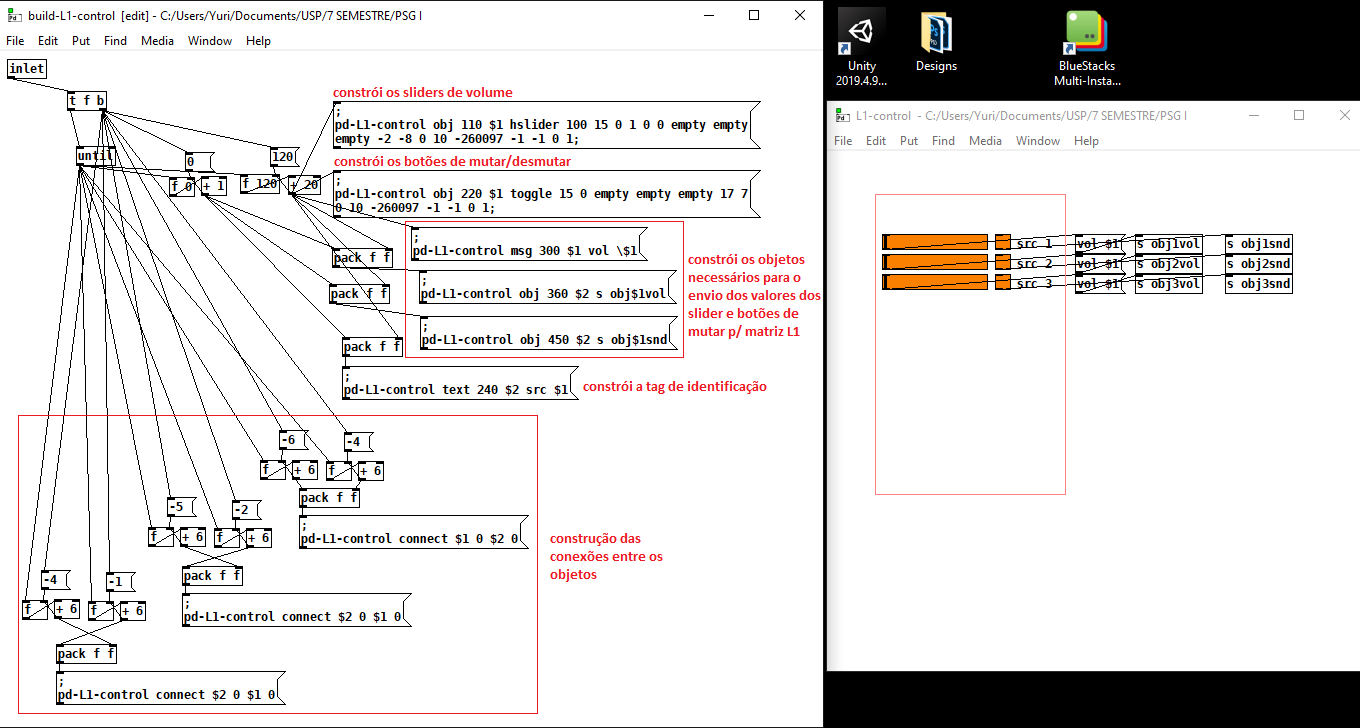
Figura 21: O subpatch build-interface à esquerda e o módulo da cena sonora à direita. Está indicado o que cada comando constrói dentro do módulo da cena sonora.



#### *6.2.1.2 Subpatch build-L1-control*

Este subpatch é responsável pela reconstrução dos controles de volumes do módulo L1. Quando acionado, ou seja, quando é alterado o número de fontes sonoras, ele reconstrói o submódulo onde se encontram os controles de volumes L1.

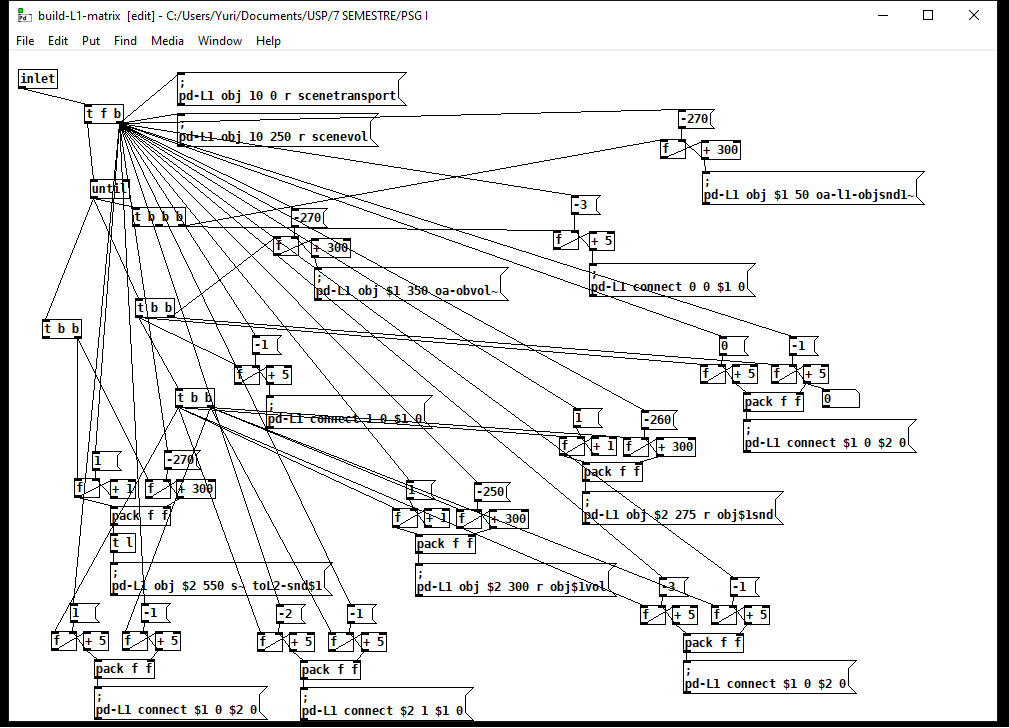
Figura 22: Subpatch build-L1-control aberto à esquerda e o subpatch dos controles de volume L1 aberto à direita. Este subpatch reconstrói, dentro do submódulo dos controles de volumes L1, os sliders de volumes, botões de mutar, tags e objetos e conexões para o envio dos dados para a matriz L1, pois dentro da matriz L1 (figura ) há outros controles de volumes que recebem o áudio dos players e mais o valor do slider e botão de mutar do submódulo de controles de volume L1 e assim repassando esse sinal de áudio para a matriz L3, onde ocorre a codificação sonora:



#### *6.2.1.3 Subpatch build-L1-matrix*

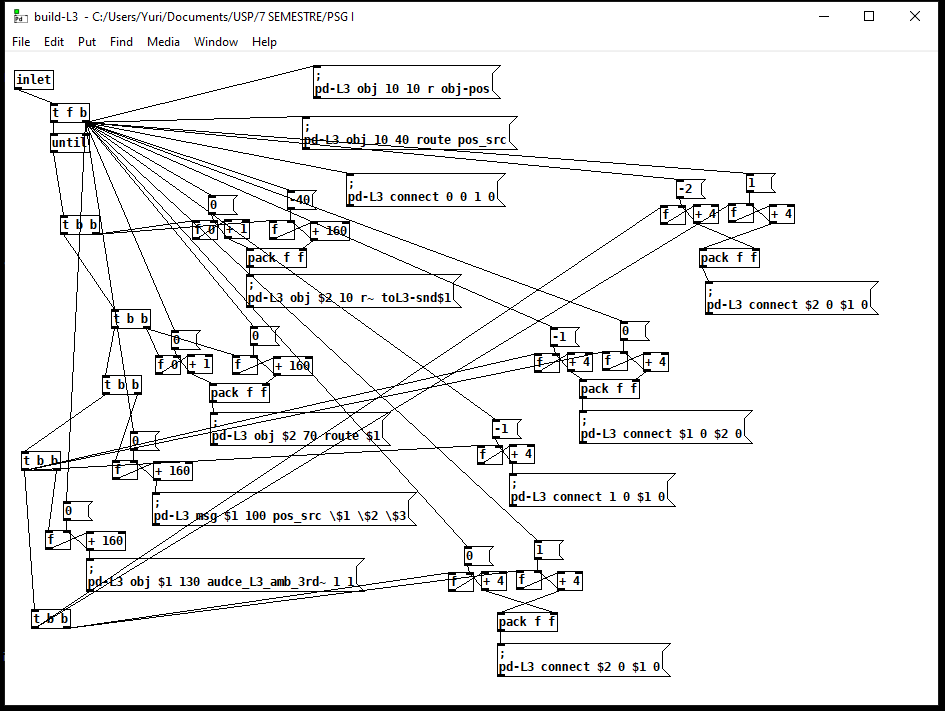
Quando este subpatch é acionado, ou seja, quando é alterado o número de fontes sonoras, ele reconstrói a matriz L1:

Figura 23: Patch build-L1-matrix aberto. Esse patch constrói os objetos presentes na matriz L1 (figura ): players, controles de volumes, objetos de recebimento de valores dos controles de volume da interface e conexões entre os objetos. O patch apesar de funcionar, está um pouco confuso. A ideia é posteriormente refazer esse patch de modo que fique mais organizado e compreensível



#### *6.2.1.4 Subpatch build-L3*

Quando este subpatch é acionado, ou seja, quando é alterado o número de fontes sonoras, ele reconstrói a matriz L3 (a matriz L3 é o subpatch responsável pela codificação sonora)

Figura 24: Patch build-L3 aberto. Este patch constrói os objetos da matriz L3 (figura 24)

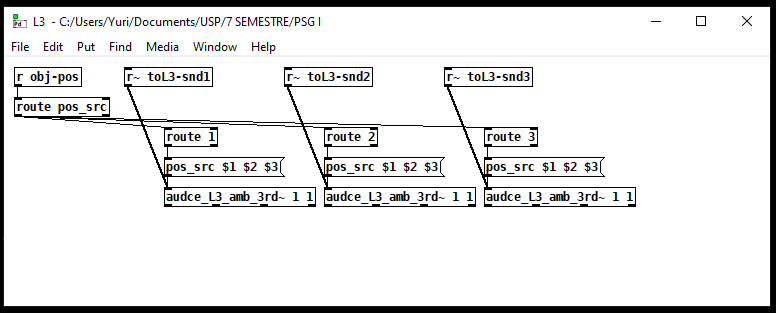
### *6.2.2 Módulo L2*

Ainda faltam alguns objetos L2 a serem construídos. Por isso, esse módulo não será implementado no momento

### *6.2.3 Módulo L3*

O módulo L3 é responsável pela codificação sonora. Este módulo contém a matriz L3.

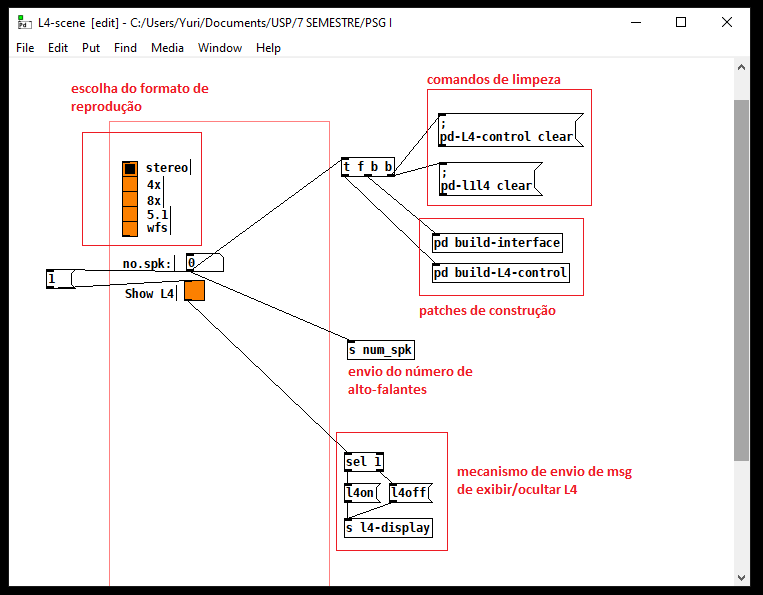
Figura 25: Patch da Matriz L3 aberto. A matriz L1 é um subpatch encontrado no módulo L3 (veja a figura 16). Este subpatch recebe (pelo objeto “r obj-pos”), do módulo de cena sonora, a posição de uma fonte que tem sua posição alterada na cena, e recebe também o sinal de áudio da matriz L1. A posição e áudio da fonte é enviada para o objeto audce\_L3\_amb\_3rd~ que faz a codificação. Neste exemplo, é uma matriz L3 para uma cena com 3 fontes sonoras



### *6.2.4 Módulo L4*

O módulo L4, na verdade, é bem parecido com o módulo L1. Assim como o módulo L1, também possui o subpatch build-interface que reconstrói o módulo da cena sonora. E também possui um submódulo que contém os controles de volume de cada alto falante que é reconstruído pelo subpatch build-L4-controls

Figura 26: Módulo L4 aberto. Como se pode ver, o módulo L4 é bem parecido com o módulo L1. Ao se alterar o número de alto-falantes, é ativado, em ordem, os comandos de limpeza do submódulo de controle de volumes L4 e do módulo da cena sonora. Depois é ativado os patches build-interface, que reconstrói o módulo da cena sonora (figura 20), e build-L4-control, que reconstrói o submódulo de controles de volume L4



# 7 Conclusão

O objetivo previsto para essa primeira fase de projetos foi concluído: a implementação do objeto oa\_l1l4\_gui (interface que integra fontes sonoras e alto-falantes). Mas ainda é necessário corrigir certos problemas: organizar melhor a documentação do código, correção de bugs (não está ocorrendo o salvamento das posições dos elementos da cena. Envolve o método de salvamento, oa\_l1l4\_gui\_save) e fazer mais tratamentos de erros que o usuário pode cometer ao fazer a passagem de argumentos: colocar mais (ou menos) argumentos que o previsto, colocar um argumento de tipo errado, etc.

Na segunda fase do projeto, a interface se tornou reconfigurável, sendo possível modificar as dimensões e o número de elementos da cena. Ainda falta implementar certas partes e detalhes:

* o módulo L2 e a matriz L4 da interface, que para serem implementados precisam de outros objetos que ainda não foram construídos
* alguns patches apesar de funcionais (como o build-L1-matrix e build-L3), precisam ser arrumados a modo de ficarem mais organizados e compreensíveis
* controle de volume geral, player geral, opções de codificação, opções de reprodução

A ideia é que o projeto continue sob a forma de pesquisa/bolsa para este ano de 2022. Assim, estas poucas partes faltantes e possíveis novas funcionalidades serão implementadas. Em relação ao início do projeto, a interface do AUDIENCE progrediu muito,indo de duas pequenas interfaces simples (oa\_l1\_gui e oa\_l4\_gui) até uma interface robusta, reconfigurável, com diversas funcionalidades e que centraliza todo o funcionamento do software.

# Referências bibliográficas

Faria, R.R.A.; Zuffo, M.K.; Zuffo, J.A. Improving spatial perception through sound field simulation in VR. VECIMS 2005 - IEEE International Conference on Virtual Environments, Human-Computer Interfaces, and Measurement Systems, Giardini Naxos, Italy, 18-20 July 2005, p.104-108.

FARIA, Regis Rossi Alves. Auralização em ambientes audiovisuais imersivos. 2005. Tese (Doutorado em Sistemas Eletrônicos) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2005. doi:10.11606/T.3.2005.tde-29092013-105249. Acesso em: 2021-08-13.

FARIA, R. R. A. AUDIENCE para Pd: uma biblioteca para áudio espacial orientada a cena sonora. In: Anais do 10o Congresso de Engenharia de Áudio da AES Brasil, São Paulo, 2012, p. 98-105. Acesso em <http://aesbrasil.org.br/congressos/anais/>

FARIA, R. R. A. Auralização em ambientes audiovisuais imersivos. 2005. Tese de Doutorado, Universidade de São Paulo. doi 10.11606/T.3.2005.tde-29092013-105249)

RUMSEY, F. Spatial audio. Oxford: Focal Press, 2003.

SILVA, Marcio José da. Modelagem de um sistema para auralização musical utilizando Wave Field Synthesis. 2014. Dissertação (Mestrado em Processos de Criação Musical) - Escola de Comunicações e Artes, University of São Paulo, São Paulo, 2014. doi:10.11606/D.27.2014.tde-18052015-163521. Acesso em: 2021-04-26.

VORLÄNDER, M. Auralization: fundamentals of acoustics, modelling, simulation, algorithms and acoustic virtual reality. Springer, 2011.

ZMOLNIG, J. M.. How to write an External for puredata. Disponível em: <http://mlab.taik.fi/~korayt/kuva/Pd-0.38.4-extended-RC7.app/Contents/Resources/doc/manuals/Externals-HOWTO/pd-externals-HOWTO.pdf>. Acesso em: 26 abr. 2021.

Ambisonics. Homepage, 2021. Disponível em: <<http://www.ambisonic.net/>>. Acesso em: 12 ago. 2021.

Cycling`74. What is Max?, 2021. <<https://cycling74.com/products/max>>. Acesso em: 12 ago. 2021.

Dolby. Homepage do Dolby Atmos, 2021. <<https://www.dolby.com/technologies/dolby-atmos/>>. Acesso em: 12 ago. 2021.

Facebook. Homepage do Facebook 360, 2021. <<https://facebook360.fb.com/spatial-workstation/>>. Acesso em: 12 ago. 2021.

IRCAM. Fórum IRCAM, 2021. <<https://forum.ircam.fr/projects/detail/spat/>>. Acesso em: 12 ago. 2021.

Núcleo de Engenharia de Áudio e Codificação Sonora. 2021. Projeto Audience. Disponível em: <<http://www.lsi.usp.br/audience>>. Acesso em: 26 abr. 2021.

Núcleo de Engenharia de Mídias. 2021. O Projeto Audience. Disponível em: <<http://www.lsi.usp.br/interativos/neac/audience/audience.html#0>>. Acesso em: 26 abr. 2021.

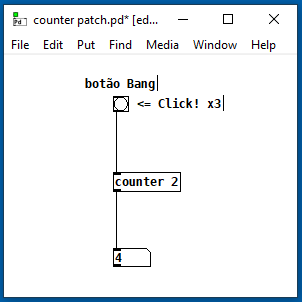
Pure Data. Homepage, 2021 Disponível em: <<https://puredata.info/>>. Acesso em: 26 abr. 2021.

Youtube. Canal “cheetomoskeeto”, 2021. <<https://m.youtube.com/user/cheetomoskeeto>>. Acesso em: 12 ago. 2021.

# ANEXO A – Como fazer um external para Pure Data

O Pd é escrito na linguagem de programação C. Devido à sua natureza gráfica, o Pd é um sistema orientado a objetos. Infelizmente C não suporta muito bem o uso de classes. Portanto, o código-fonte resultante não é tão elegante quanto o código C++ seria, por exemplo. Neste contexto de criação de externals, o termo *classe* refere-se aos dados e as funções que manipulam desses dados. Instâncias concretas de uma *classe* são chamadas de *objetos*.

Para dar uma breve explicação de como se cria um external, segue um tutorial que vem no artigo “How to write externals for Pure Data” de ZMOLNIG, Johannes M. sobre como fazer um external chamado “counter” que faz a contagem do número de vezes que recebeu uma mensagem “bang”, que é uma mensagem enviada quando é clicado um botão do Pure Data de mesmo nome. O botão Bang é muito utilizado no Pd, servindo de gatilho para ações de outros objetos que estão conectados a ele. Quando o counter recebe a mensagem “bang” enviada pelo botão Bang, o valor do contador aumenta em 1 e o valor do contador é enviado pelo outlet. O counter vai poder receber como argumento, um valor inteiro, que será o valor inicial do contador. Caso não seja especificado esse argumento, o valor inicial assumido será zero. Veja a figura 13 para melhor entendimento. Abaixo também está todo o algoritmo do *counter* e o passo-a-passo explicando o código.

Figura 13: O objeto counter. Neste exemplo o botão Bang é clicado três vezes, ou seja, é enviado em sequência três mensagens “bang” ao counter que está conectado a ele. O primeiro bang não é contabilizado, só serve para inicializar o counter. Apenas o segundo e o terceiro bangs são contabilizados, e como o valor inicial do counter foi definido como dois, então o valor da contagem é quatro. O valor da contagem é enviado ao objeto de número, assim exibindo o valor da contagem.

**Algoritmo 1** - código do objeto “counter”1: #include "m\_pd.h"

2: static t\_class \*counter\_class;

3: typedef struct \_counter {

4: t\_object x\_obj;

5: int i\_count;

6: } t\_counter;

7: void counter\_bang(t\_counter \*x)

8: {

9: t\_float f=x->i\_count;

10: x->i\_count++;

11: outlet\_float(x->x\_obj.ob\_outlet, f);

12: }

14: void \*counter\_new(t\_floatarg f)

15: {

16: t\_counter \*x = (t\_counter \*)pd\_new(counter\_class);

17: x->i\_count=f;

18: outlet\_new(&x->x\_obj, &s\_float);

19: return (void \*)x;

20: }

21: void counter\_setup(void)

22: {

23: counter\_class = class\_new(gensym("counter"),

24: (t\_newmethod)counter\_new,

25: 0, sizeof(t\_counter),

26: CLASS\_DEFAULT,

27: A\_DEFFLOAT, 0);

28: class\_addbang(counter\_class, counter\_bang);

29: }

O primeiro passo é incluir o arquivo “m\_pd.h” que é a interface para o Pd que provê as funções necessárias para se criar um external. (linha 1)

Primeiro, uma nova classe deve ser preparada e o espaço de dados para esta classe deve ser definido (linhas 3 a 6). O counter\_class será um ponteiro para a nova classe (linha 2). A estrutura t\_counter (do tipo \_counter) é o espaço de dados da classe. Um elemento absolutamente necessário do espaço de dados é uma variável do tipo t\_object, que é usada para armazenar propriedades de objetos internos, como a apresentação gráfica do objeto ou dados sobre entradas e saídas. t\_object deve ser a primeira entrada na estrutura. A variável i\_count será responsável por armazenar o número de “bangs” recebidos.

geração de uma nova classe

Para gerar uma nova classe, as informações do espaço de dados e do espaço de métodos dessa classe devem ser passadas para Pd quando uma biblioteca é carregada.Ao carregar uma nova biblioteca “my\_lib”, Pd tenta chamar a função “my\_lib\_setup ()”. Esta função (ou funções chamadas por ela) declara as novas classes e suas propriedades. É chamado apenas uma vez, quando a biblioteca é carregada. Se a chamada de função falhar (por exemplo, porque nenhuma função do nome especificado está presente), nenhum external da biblioteca será carregado (linhas 21 a 29).

A função class\_new (linha 23) cria uma nova classe e retorna um ponteiro para este protótipo. O primeiro argumento é o nome simbólico da classe. Os próximos dois argumentos definem o construtor e o destruidor da classe (linhas 24 e 25) . Sempre que um objeto de classe é criado em um patch Pd, o construtor de classe (t\_newmethod) counter\_new instancia o objeto e inicializa o espaço de dados.

Sempre que um objeto é destruído (fechando o patch que o contém ou excluindo o objeto do patch), o destruidor libera a memória reservada dinamicamente. A memória alocada para o espaço de dados estáticos é automaticamente reservada e liberada. Portanto, como não precisamos fornecer um destruidor neste exemplo o terceiro argumento é definido como “0”.

Para permitir que o Pd reserve e libere memória suficiente para o espaço de dados estáticos, o tamanho da estrutura de dados deve ser passado como o quarto argumento.

O quinto argumento tem influência na representação gráfica dos objetos da classe. O valor padrão é CLASS\_DEFAULT ou simplesmente “0”. Os argumentos restantes definem os argumentos de um objeto e seu tipo.

Até seis argumentos de objeto numéricos e simbólicos podem ser definidos por meio de A\_DEFFLOAT e A\_DEFSYMBOL. Se mais argumentos devem ser passados ​​para o objeto ou se a ordem dos tipos de átomos deve ser mais flexível, A\_GIMME pode ser usado para passar uma lista arbitrária de atoms.

É bastante útil para um contador, se um valor inicial puder ser definido pelo usuário. Portanto, esse valor inicial deve ser passado para o objeto no momento da criação. Portanto, temos um argumento adicional na função class\_new: A\_DEFFLOAT diz ao Pd que o objeto precisa de um argumento do tipo t\_floatarg. Se nenhum argumento for passado, o padrão será “0”.

A lista de argumentos-objeto é encerrada com “0”. Neste exemplo, não temos nenhum argumento de objeto para a classe.

Ainda temos que adicionar um espaço de método à classe. O class\_addbang adiciona um método para uma mensagem “bang” à classe que é definida no primeiro argumento. O método adicionado é definido no segundo argumento.

**Construtor: instanciação de um objeto**

Cada vez que um objeto é criado em um patch Pd, o construtor que é definido com o comando class\_new, gera uma nova instância da classe. O construtor deve ser do tipo void \*.

* ler e validar parâmetros de entrada
* criar entradas e saídas
* alocar memória

A função pd\_new reserva memória para o espaço de dados, inicializa as variáveis que são internas ao objeto e retorna um ponteiro para o espaço de dados. A conversão de tipo para o espaço de dados é necessária.

O método construtor tem um argumento do tipo t\_floatarg conforme declarado na rotina de instalação por class\_new. Este argumento é usado para inicializar o contador.

Um novo outlet é criado com a função outlet\_new. O primeiro argumento é um ponteiro para o interno do objeto para o qual a nova saída foi criada.

O segundo argumento é uma descrição simbólica do tipo de saída. Visto que o contador de saída deve gerar valores numéricos, ele é do tipo “flutuante”.

outlet\_new retorna um ponteiro para o novo outlet e salva esse mesmo ponteiro na variável t\_objeto x\_obj.ob\_outlet. Se apenas uma saída for usada, o ponteiro não precisa ser adicionalmente armazenado no espaço de dados. Se mais de uma saída for usada, os ponteiros devem ser armazenados no espaço de dados, porque a variável t\_object pode conter apenas um ponteiro de saída.

Os argumentos do método construtor dependem dos argumentos do objeto definidos com class\_new.

class\_new-argument constructor-argument

A\_DEFFLOAT t\_floatarg f

A\_DEFSYMBOL t\_symbol \* s

A\_GIMME t\_symbol \* s, int argc, t\_atom \* argv

Como não há argumentos de objeto para nossa classe “hello world”, o construtor também possui um anon.

A função pd\_new reserva memória para o espaço de dados, inicializa as variáveis que são internas ao objeto e retorna um ponteiro para o espaço de dados.

A conversão de tipo para o espaço de dados é necessária.

Normalmente, o construtor inicializaria as variáveis de objeto. No entanto, como não temos nenhum, isso não é necessário.

O construtor deve retornar um ponteiro para o espaço de dados instanciado. Se retornar NULL, Pd acha que o objeto não foi criado.

# ANEXO B – Algoritmo de recebimento de mensagem de posição de fontes sonoras

| **Algoritmo 4** - Método oa\_l1l4\_gui\_pos\_src. Este método recebe as mensagens de posição de uma determinada fonte da interface. Os métodos que fazem o recebimento das mensagens de posição de alto-falantes e ouvintes (oa\_l1l4\_gui\_pos\_spk e oa\_l1l4\_gui\_pos\_rcv) são praticamente idênticos ao oa\_l1l4\_gui\_pos\_src. |
| --- |
| static void oa\_l1l4\_gui\_pos\_src(t\_oa\_l1l4\_gui \*x, t\_symbol \*s, int argc, t\_atom \*argv) {  t\_int sel;  if ( argc != 4) {  post( "oa\_l1l4\_gui: pos\_src - número de argumentos errado!");  return;  }  if ( argv[0].a\_type != A\_FLOAT || argv[1].a\_type != A\_FLOAT || argv[2].a\_type != A\_FLOAT || argv[3].a\_type != A\_FLOAT) {  post( "oa\_l1l4\_gui: pos\_src - argumentos não numéricos" );  return;  }  sel = (int)atom\_getfloatarg(0, argc, argv) - 1;  if ( sel < 0 || sel >= x->x\_num\_src) {  post( "oa\_l1l4\_gui: pos\_src - Fonte inexistente!");  return;  }  //O vetor “meter” armazena as coordenadas x y z determinadas pela mensagem. Depois o método oa\_l1l4\_gui\_convert\_meter\_to\_pixel() pega esse vetor com essas coordenadas, transforma esses valores de metro para pixel e armazena na matriz que armazena as posições das fontes.  if ( strcmp(x->x\_format\_scale, DEFAULT\_FORMAT\_SCALE) == 0 )  {  meter[0] = atom\_getfloatarg(1, argc, argv);  meter[1] = atom\_getfloatarg(2, argc, argv);  meter[2] = atom\_getfloatarg(3, argc, argv);  }  if ( strcmp(x->x\_format\_scale, CARTESIAN\_FORMAT\_SCALE) == 0 )  {  meter[0] = atom\_getfloatarg(2, argc, argv);  meter[1] = -(int)atom\_getfloatarg(1, argc, argv);  meter[2] = atom\_getfloatarg(3, argc, argv);  }  oa\_l1l4\_gui\_convert\_meter\_to\_pixel(x, x->x\_src[sel], meter);  oa\_l1l4\_gui\_draw\_update(x, x->x\_glist);  oa\_l1l4\_gui\_send\_pos\_src(x, sel);  } |

# 

# ANEXO C – Algoritmos de exibir/ocultar L1/L4

| **Algoritmo 4** - Método oa\_l1l4\_gui\_l1on |
| --- |
| static void oa\_l1l4\_gui\_l1on(t\_oa\_l1l4\_gui \*x)  {  int ei;  t\_canvas \*canvas=glist\_getcanvas(x->x\_glist);  if (x->x\_l1\_display == L1\_OFF)  {  // Draw the sound objects (sources of L1)  for ( ei=0; ei<x->x\_num\_src; ei++ )  {  sys\_vgui("image create photo %xSOURCE%d -data %s -format gif -width %d -height %d\n",  x, ei, &src\_img, SOURCE\_WIDTH, SOURCE\_HEIGHT);  sys\_vgui(".x%lx.c create image %d %d -image %xSOURCE%d -tags %xISOURCE%d\n",  canvas,  text\_xpix(&x->x\_obj, x->x\_glist) + x->x\_src[ei][0],  text\_ypix(&x->x\_obj, x->x\_glist) + x->x\_src[ei][1],  x, ei, x, ei);  sys\_vgui(".x%lx.c create text %d %d -font -\*-courier-bold--normal--12-\* -text \"%d\" -tags %xSOURCENUM%d\n",  canvas,  text\_xpix(&x->x\_obj, x->x\_glist) + x->x\_src[ei][0] - SOURCE\_WIDTH/2,  text\_ypix(&x->x\_obj, x->x\_glist) + x->x\_src[ei][1] - SOURCE\_HEIGHT/2,  ei+1, x, ei );  }  x->x\_l1\_display = L1\_ON;  }  } |

| **Algoritmo 4** - Método oa\_l1l4\_gui\_l4on |
| --- |
| static void oa\_l1l4\_gui\_l4on(t\_oa\_l1l4\_gui \*x)  {  int ep;  t\_canvas \*canvas=glist\_getcanvas(x->x\_glist);  if (x->x\_l4\_display == L4\_OFF)  {  for ( ep=0; ep<x->x\_num\_spk; ep++ )  {  sys\_vgui("image create photo %xSPEAKER%d -data %s -format gif -width %d -height %d\n",  x, ep, &spk\_img, SPEAKER\_WIDTH, SPEAKER\_HEIGHT);  sys\_vgui(".x%lx.c create image %d %d -image %xSPEAKER%d -tags %xISPEAKER%d\n",  canvas,  text\_xpix(&x->x\_obj, x->x\_glist) + x->x\_spk[ep][0],  text\_ypix(&x->x\_obj, x->x\_glist) + x->x\_spk[ep][1],  x, ep, x, ep);  sys\_vgui(".x%lx.c create text %d %d -font -\*-courier-bold--normal--12-\* -text \"%d\" -tags %xSPEAKERNUM%d\n",  canvas,  text\_xpix(&x->x\_obj, x->x\_glist) + x->x\_spk[ep][0] - SPEAKER\_WIDTH/2,  text\_ypix(&x->x\_obj, x->x\_glist) + x->x\_spk[ep][1] - SPEAKER\_HEIGHT/2,  ep+1, x, ep );  }  x->x\_l4\_display = L4\_ON;  }  } |

# 

| **Algoritmo 5** - método oa\_l1l4\_gui\_l1off |
| --- |
| static void oa\_l1l4\_gui\_l1off(t\_oa\_l1l4\_gui \*x)  {  int ei;  t\_canvas \*canvas=glist\_getcanvas(x->x\_glist);  if (x->x\_l1\_display == L1\_ON)  {  for ( ei=0; ei<x->x\_num\_src; ei++ )  {  sys\_vgui(".x%lx.c delete %xISOURCE%d\n", canvas, x, ei );  sys\_vgui(".x%lx.c delete %xSOURCENUM%d\n", canvas, x, ei );  //sys\_vgui("image delete %xSOURCE%d\n", x, ei );  }  x->x\_l1\_display = L1\_OFF;  }  } |

# 

| **Algoritmo 5** - método oa\_l1l4\_gui\_l4off |
| --- |
| static void oa\_l1l4\_gui\_l4off(t\_oa\_l1l4\_gui \*x)  {  int ei;  t\_canvas \*canvas=glist\_getcanvas(x->x\_glist);  if (x->x\_l4\_display == L4\_ON)  {  for ( ei=0; ei<x->x\_num\_spk; ei++ )  {  sys\_vgui(".x%lx.c delete %xISPEAKER%d\n", canvas, x, ei );  sys\_vgui(".x%lx.c delete %xSPEAKERNUM%d\n", canvas, x, ei );  }  x->x\_l4\_display = L4\_OFF;  }  } |

# 