Universidade Federal da Paraíba

Centro de Informática

Programa de Pós-Graduação em Computação, Comunicação e Artes

RAMON PAULINO DE LIMA

Soluções viáveis para utilização do *Pure Data* no desenvolvimento de *audiogames mobile* para pessoas com deficiência visual na plataforma *Unity*

João Pessoa

Dezembro / 2021

Ramon Paulino de Lima

Soluções viáveis para utilização do *Pure Data* no desenvolvimento de *audiogames mobile* para pessoas com deficiência visual na plataforma *Unity*

Dissertação apresentada ao Programa de Pós-Graduação em Computação, Comunicação e Artes (PPGCCA) da Universidade Federal da Paraíba, como requisito parcial para a obtenção do título de Mestre em Computação, Comunicação e Artes, na linha de pesquisa Arte Computacional.

Orientador: Prof. Dr. Carlos Eduardo Coelho Freire Batista

João Pessoa

Dezembro / 2021

Universidade Federal da Paraíba

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO, COMUNICAÇÃO E ARTES

ATA Nº 1

Aos dezesseis dias do mês de dezembro do ano de dois mil e vinte e um, às 16h00min, através de videoconferência, instalou-se a banca examinadora de dissertação de Mestrado do aluno RAMON PAULINO DE LIMA. A banca examinadora foi composta pelos professores Dr. FRANCISCO DE PAULA BARRETTO, UFBA, examinador externo à instituição, Dra. LILIANE DOS SANTOS MACHADO, UFPB, examinador interno, Dr. CARLOS EDUARDO COELHO FREIRE BATISTA, UFPB, presidente. Deu-se início a abertura dos trabalhos, por parte do professor Dr. CARLOS EDUARDO COELHO FREIRE BATISTA, coordenador do Programa, que, após apresentar os membros da banca examinadora e esclarecer a tramitação da defesa, passou a presidir os trabalhos da banca, solicitando de imediato ao candidato que iniciasse a apresentação da dissertação, intitulada Uso do Pure Data e Unity no desenvolvimento de audiogames mobile para pessoas com deficiência visual. Concluída a exposição, o professor Dr. CARLOS EDUARDO COELHO FREIRE BATISTA, presidente, passou a palavra ao professor Dr. FRANCISCO DE PAULA BARRETTO, para arguir o candidato, e, em seguida, à professora Dra. LILIANE DOS SANTOS MACHADO; após o que fez suas considerações sobre o trabalho em julgamento, tendo sido APROVADO o candidato, conforme as normas vigentes na Universidade Federal da Paraíba.

A versão final da dissertação deverá ser entregue ao programa, no prazo de 60 dias; contendo as modificações sugeridas pela banca examinadora e constante na folha de correção enviada ao aluno. O candidato não terá o título se não cumprir as exigências acima.

Dr. FRANCISCO DE PAULA BARRETTO, UFBA

Examinador Externo à Instituição

Dra. LILIANE DOS SANTOS MACHADO, UFPB

Examinador Interno

Dr. CARLOS EDUARDO COELHO FREIRE BATISTA, UFPB

Presidente

RAMON PAULINO DE LIMA

Mestrando

Catalogação na publicação
Seção de Catalogação e Classificação

L732s Lima, Ramon Paulino de.
Soluções viáveis para utilização do Pure Data no desenvolvimento de audiogames mobile para pessoas com deficiência visual na plataforma Unity / Ramon Paulino de Lima. - João Pessoa, 2021.
100 f. : il.

Orientação: Carlos Eduardo Coelho Freire Batista.
Dissertação (Mestrado) - UFPB/CCHLA.

1. Linguagem de programação visual. 2. Pure Data. 3. Jogos baseados em áudio. 4. Acessibilidade. I. Batista, Carlos Eduardo Coelho Freire. II. Título.

UFPB/BC CDU 004.43(043)

Elaborado por MAGNOLIA FELIX DE ARAUJO - CRB-15/883

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Carlos Eduardo Batista Freire, pelo apoio e sabedoria nesta trajetória.

A todos os professores, técnicos e colegas de turma que contribuíram no meu crescimento durante este período.

Gostaria de deixar registrado também, minha gratidão e reconhecimento à minha mãe e minha companheira Elizabeth, pois sem o seu apoio e suporte eu jamais estaria aqui.

Enfim, a todos os que de alguma forma contribuíram para o meu crescimento.

RESUMO

O segmento de jogos para dispositivos móveis representa uma parte relevante na indústria, movimentando bilhões de dólares todos os anos, devido sua popularidade e fácil acesso no dia-a-dia. Infelizmente uma grande parcela de usuários fica fora desse tipo de mídia, os IDV (indivíduos com deficiência visual), e para isto os jogos baseados em áudio (ou *audiogames*) surgem para suprir essa necessidade. O objetivo deste estudo é investigar o uso do *Pure Data* (*Pd*) no desenvolvimento de *audiogames mobile* na plataforma *Unity*. Para isso, desenvolvemos o *Binaural Flight*, um *endless scrolling game* vertical explorando a integração de Pure Data e a plataforma *Unity*. No decorrer deste trabalho verificamos que o *Pd* é viável e versátil como ferramenta de manipulação e geração de áudio em tempo real no jogo. O uso de *libpd* permite a implementação do *Pd* em diversas plataformas. A partir deste estudo, propomos um conjunto de soluções para auxiliar no uso e implementação do *Pure Data* no desenvolvimento de *audiogames mobile* na plataforma *Unity*, assim tornando uma opção viável para desenvolvedores independentes e assim fomentar novas produções de *audiogames* para indivíduos com deficiência visual.

Palavras-chave: *audiogames* móveis, acessibilidade, *pure data*.

ABSTRACT

The mobile games segment represents a relevant part of the industry, moving billions of dollars every year, due to its popularity and easy access in everyday life. Unfortunately a large portion of users are out of this type of media, the VI (visually impaired), and for this audio-based games (or audiogames) emerge to meet this need. The aim of this study is to investigate the use of Pure Data (Pd) in the development of mobile audiogames on the Unity platform. To do this, we developed Binaural Flight, an endless vertical scrolling game exploring the integration of Pure Data and the Unity platform. In the course of this work we verified that Pd is viable and versatile as a real-time audio manipulation and generation tool in the game. The use of libpd allows the implementation of Pd on several platforms. From this study, we propose a set of solutions to assist in the use and implementation of Pure Data in the development of mobile audio games on the Unity platform, thus making it a viable option for independent developers and thus fostering new audio game productions for visually impaired individuals.

Keywords: *mobile audiogames*, *accessibility, pure data.*

LISTA DE FIGURAS

LISTA DE TABELAS

SIGLAS E ABREVIATURAS

IDV	Indivíduo com deficiência visual
OMS	Organização Mundial da Saúde
PPGCCA	Programa de Pós-Graduação em Computação, Comunicação e Artes
Pd	*Pure Data*
AAA	Jogo triple A
WACAG	*Web Content Accessibility Guidelines*
HRTF	*Head-related transfer function*
RPG	*Role Playing Game*
CI	Critérios de Inclusão
CE	Critérios de Exclusão
TTS	*Text-To-Speech*
LAVID	Laboratório de Aplicações de Vídeo Digital

SUMÁRIO

1. INTRODUÇÃO

Os jogos eletrônicos oferecem diversos benefícios para seus usuários, seja no desenvolvimento de habilidades psicomotoras e de raciocínio lógico quanto com viés de entretenimento e lazer. Os jogos de ação e aventura, por exemplo, induzem o jogador a tomar decisões rápidas e que o permitam resolver problemas, assim avançando no jogo. Desta maneira, o jogador aprimora suas capacidades ao mesmo tempo em que se envolve no enredo do jogo. Atualmente, os dispositivos móveis (*smartphones* e *tablets*) são responsáveis por movimentar grande parte da indústria dos jogos ao redor do mundo. Segundo pesquisa realizada pela *Gamingscan*[1], dos 152,1 bilhões de dólares arrecadados no ano de 2019, destes 68,5 bilhões de dólares vieram apenas dos dispositivos móveis.

Nos *audiogames*, ou jogos baseados apenas em áudio, toda a imersão e mecânica do jogo é totalmente baseada em áudio, sem o uso dos elementos visuais. O nicho de jogadores é composto por pessoas com deficiência visual (parcial ou completa), desta maneira, o som é o elemento principal que compõe o jogo. Segundo a OMS (Organização Mundial da Saúde, 2019), mais de 2.2 bilhões de pessoas possuem algum grau de deficiência visual ao redor do mundo.

Ballan et al. (2015) diz que os *audiogames* constroem uma experiência de usuário inteira baseada em uma interface auditiva, e que as pessoas com a deficiência visual devem perceber os *audiogames* como um brinquedo, um método fácil e interativo de aprender e se divertir ao mesmo tempo. Devemos destacar também que existem *audiogames* que não são necessariamente voltados para o público com deficiência visual, existem jogos narrativos, imersivos e musicais com o intuito de aprendizagem ou treinamento de percepção sonora, os quais utilizam *engines* genéricas e não se preocupam com o aspecto de acessibilidade para os indivíduos com deficiência visual, ou IDV.

Em 1997, Puckette desenvolveu o *Pure Data,* que é uma linguagem de programação que utiliza recursos visuais e intuitivos para manipulação de dados puros. Entre as diversas utilizações do *Pure Data (Pd)* está a síntese e processamento de sinais de áudio, característica que fez com que a comunidade (organizada em grupos de usuários em fóruns especializados na internet) aprimorasse ainda mais a linguagem, adicionando novas funcionalidades e usos.

[1] <https://www.gamingscan.com/gaming-statistics/>. Acessado em: 17/08/2021.

Devido ao fato de ser uma linguagem simples e de código aberto, o *Pd* pode facilmente ser integrado em outros sistemas, permitindo diversas utilidades e aplicações.

Com isto, buscamos auxiliar na experiência de navegação sonora e geração de elementos sonoros em *audiogames mobile* para pessoas com deficiência visual, completa ou parcial. As soluções levantadas no decorrer deste estudo devem proporcionar aos desenvolvedores um conjunto de passos e parâmetros para a utilização de ferramentas que possibilitem a fácil implementação do *Pure Data* na criação de *audiogames* para dispositivos móveis na plataforma *Unity*. O modelo deve ser acessível e replicável, de maneira que possa ser utilizado por desenvolvedores independentes ou entusiastas, assim fomentando novas produções de *audiogames* e aplicações, assim gerando mais inclusão dos IDV nesse cenário.

Durante as pesquisas iniciais deste estudo, observamos que os termos utilizados no meio acadêmico para se referir aos jogos acessíveis para pessoas com deficiência visual são diversos, tanto em inglês quanto em português não existe um consenso ou um termo universal, porém todas se referem a mesma temática. Observamos o uso de termos em inglês como *audio-only games*, *audio-based games, video-less games, blind-accessible games, audiogames.* Já na língua portuguesa verificamos o uso de termos como jogos acessíveis, jogos inclusivos, áudio jogos e *audiogames*. Devido a grande diversidade de termos e também para melhor entendimento desta pesquisa, vamos utilizar o termo *audiogame* para nos referir aos áudio jogos para IDV, devido ao fato que podemos encontrar esse mesmo termo em publicações nas duas línguas utilizadas nesta pesquisa.

A interdisciplinaridade explorada no Programa de Pós Graduação em Computação, Comunicação e Artes (PPGCCA) abrange de forma elementar este estudo, visto que, o eixo de arte computacional está diretamente ligado com a parte de desenvolvimento e programação do *audiogame*, e também no viés artístico, no qual podemos associar diretamente com a utilização e manipulação do áudio digital, que serão elementos fundamentais no *audiogame*.

Desta forma, foi notável o suporte do PPGCCA para a pesquisa, visto que, o estudo está diretamente ligado à interdisciplinaridade característica do programa. Durante o período de estudo nas disciplinas ofertadas pelo curso, tivemos contato com diversas pessoas de áreas distintas e que de certa forma contribuíram para o amadurecimento desta pesquisa. A ideia e motivação inicial já eram de tratar sobre *audiogames*, mas durante as disciplinas de de Computação Aplicada à Comunicação e à Arte e a disciplina de Computação para Criação

Sonora e Musical fui apresentado a diversas ferramentas e linguagens de programação (como *Pure Data* e o *Processing*) que me deram o suporte necessário para o aprimoramento da desta pesquisa.

Podemos enquadrar este estudo no projeto desafios da produção musical mediada por computador, devido ao trabalho desenvolvido pelo Pr. Dr. Carlos Eduardo com tecnologias para acessibilidade e sua experiência com música e áudio digital, além de seu trabalho no LAVID[2], o qual tem impacto nacional e internacional, onde podemos destacar o Vlibras[3], um conjunto de ferramentas computacionais de código aberto, que traduz conteúdos digitais (texto, áudio e vídeo) para Língua Brasileira de Sinais - LIBRAS.

1.1 Objetivo Geral

O objetivo geral desta pesquisa é propor soluções viáveis para a utilização e implementação de *patches Pure Data* para o desenvolvimento de *audiogames mobile* usando a na plataforma *Unity*. Tais soluções devem auxiliar o desenvolvimento da experiência de navegação sonora para indivíduos com deficiência visual (IDV).

1.2 Objetivos Específicos

- Pesquisar e verificar o uso do *Pure Data* no desenvolvimento de *audiogames* para dispositivos móveis;
- Verificar e implementar o *Pure Data* na plataforma *Unity*;
- Levantar e desenvolver as ferramentas de áudio (*patches*) no *Pure Data;*
- Desenvolver um *audiogame* protótipo para teste das ferramentas propostas neste estudo;
- Propor soluções para uso e integração do *Pure Data* no desenvolvimento de *audiogames mobile* para IDV na plataforma *Unity*, bem como soluções de espacialização sonora;

[2] <http://lavid.ufpb.br/> Acessado em: 17/08/2020.
[3] <https://www.vlibras.gov.br/> Acessado em: 17/08/2020.

1.3 Justificativa

A temática foi escolhida como uma proposta de discussão sobre a inclusão de usuários com deficiência visual no grande mercado dos jogos para dispositivos móveis. Visto que, apesar das poucas produções de *audiogames* existentes, elas são desenvolvidas na maior parte pelo próprio público IDV. Grande parte são pequenas produções independentes ou com teor apenas educacional, o que limita o acesso desse público às grandes produções (também conhecidas como jogos AAA, ou *triple A*).

A inspiração para a pesquisa foi instigada durante a análise do *Audio Game Hub* (DE LIMA, 2018), onde foi observado o emprego de técnicas de sound design e do áudio binaural 3D na construção do sistema de navegação nos menus do jogo. Durante a pesquisa, notamos a escassez de material científico e produções sobre essa temática, nos servindo como motivação.

Escolhemos a linguagem de programação *Pure Data* devido a sua grande versatilidade, a qual proporciona uma interface visual de programação, o que é importante para usuários iniciantes e que possuem pouca afinidade com códigos e programação. O *Pd* já está bem estruturado e possui diversas funções que julgamos facilitadores no decorrer deste estudo, além do fato que esta linguagem é *open source* e gratuita, assim está alinhada com a proposta do modelo proposto neste trabalho.

1.4 Estrutura do trabalho

Para melhor entendimento desta pesquisa, organizamos o estudo em seções, nas quais estão divididas em nove grandes grupos: introdução, revisão teórica, trabalhos relacionados, materiais e métodos, *audiogame Binaural Flight,* resultados, conclusão, referências e anexos. Na seção 1 de introdução, subdividimos em objetivo geral, específicos, justificativa e estrutura do trabalho. Na seção 2, tratamos da revisão teórica, na qual está dividida em três subseções: *audiogames* móveis, *engines* de áudio, *Pure Data* e plataforma *Unity*. Os trabalhos relacionados podem ser encontrados na seção 3 deste estudo. A seção 4 contém os materiais e métodos utilizados na pesquisa, nesta seção também pode ser encontrada a RSL (Revisão Sistemática Literal), que por sua vez está subdividida em: questão de pesquisa, protocolo, critérios de inclusão e exclusão e os resultados. Na seção 5, apresentamos o

desenvolvimento do *audiogame Binaural Flight*, a qual subdividimos em: desenvolvimento do *audiogame*, utilizando *patches Pure Data*, implementação do *Libpd*, comunicação entre *Libpd* e a plataforma *Unity* e *LibPdIntegration* no *Binaural Flight*.

Em seguida, na seção 6 apresentamos os resultados da pesquisa, o *Pure Data* no desenvolvimento de *audiogames* e as soluções para a integração do *Pdi na plataforma Unity.* Na seção 7 temos a conclusão do estudo, subdividida em: limitações e perspectivas futuras, contribuições e considerações finais.

No final da pesquisa podem ser encontradas as duas últimas seções, as referências e os anexos, neste último disponibilizamos *scripts*, códigos fonte*, patches Pd,* equipamentos e materiais que são mencionados no decorrer deste estudo.

2. REVISÃO TEÓRICA

2.1 *Audiogames* móveis

Os *audiogames* (ou também chamados de áudio jogos, e no inglês de *audio-based games, audio only games* e *video-less games*) são jogos que não possuem elementos visuais, ou elementos gráficos, toda a experiência do jogador, tanto história como mecânica de jogo está diretamente apresentada ao jogador através de elementos sonoros. Os *audiogames* podem ser encontrados com diversos propósitos, seja com fim de acessibilidade e inclusão, ferramenta educacional e também como entretenimento e lazer. Este tipo de mídia é na maior parte das vezes associada aos IDV (indivíduos com deficiência visual), um nicho específico e que passa despercebido pelo grande mercado de games. Por isso, grande parte dos *audiogames* são desenvolvidos e projetados pelos próprios IDV, pesquisadores acadêmicos e também por desenvolvedores independentes. Devemos também destacar que nem todo *audiogame* é acessível, alguns são desenvolvidos para oferecer apenas uma experiência sensorial diferente, baseada inteiramente no som.

Para Araújo et al. (2015), os *audiogames* voltados para o usuário com deficiência visual são jogos desenvolvidos com propósito de acessibilidade e que reforçam as soluções em áudio para garantir a interação. A principal característica dos *audiogames* acessíveis é o tratamento das rotinas e narrativas do jogo através de elementos sonoros, como áudios ou uso de técnicas TTS (Text-To-Speech) sobre as informações textuais.

Balan et al. (2015) diz que os *audiogames* possuem uma grande quantidade de conteúdo sonoro e desenvolvem uma experiência de usuário inteira baseada apenas em uma interface auditiva ou em uma interface auditiva e gráfica. Os jogos de navegação baseados em áudio promovem o desenvolvimento de habilidades cognitivas espaciais e proporcionam aos IDV acessar e manipular informações sobre o ambiente, assim estimulando o aprendizado contextual. Com isso, podemos verificar a importância dos videogames no desenvolvimento de habilidades cognitivas, principalmente para os IDV.

> Os jogos de computador desempenham um papel importante no desenvolvimento de estruturas mentais, pois aprimoram o aprendizado, melhoram a criatividade e as habilidades de resolução de problemas, estimulam a motivação e resolvem os problemas de comunicação. Infelizmente, devido às suas óbvias limitações físicas e cognitivas e como os videogames dependem principalmente de informações gráficas, as pessoas com deficiência visual são incapazes de jogar a maioria dos

> jogos de computador disponíveis no mercado atualmente. (BALAN et al., 2015, p.2, tradução nossa).[4]

Os *audiogames* possuem um papel importante também no treinamento de habilidades de orientação e mobilidade, visto que possuem grande capacidade de aprimorar a percepção auditiva do jogador. Assim, podemos entender os *audiogames* como uma ferramenta poderosa para o ensino, aprimoramento e entretenimento de usuários com deficiência visual. "Os jogos de áudio, no entanto, são uma modalidade de treinamento confiável que melhora a orientação e a mobilidade por meio de uma abordagem acessível, agradável e divertida, adequada aos requisitos dos deficientes visuais."[5] (BALAN et al., 2015, p.3, tradução nossa).

> Em jogos baseados em áudio, as informações sonoras são percebidas continuamente e orientadas contextualmente dentro de uma interface atraente e dinâmica, cujo objetivo não é apenas fornecer uma experiência fascinante ao usuário, mas também construir mapas cognitivos espaciais, desenvolver capacidades sensoriais alternativas e melhorar a orientação , habilidades de mobilidade e navegação. (BALAN et al., 2015, p.3, tradução nossa).

Balan et al. (2015) diz que os IDV devem perceber os *audiogames* como brinquedos, um método fácil e interativo de aprendizagem e também uma fonte de entretenimento. Para ser acessível e atraente para os IDV os jogos devem atender às seguintes especificações:

- Um alto nível de imersão e atratividade para motivar o usuário a avançar no jogo;
- Diversidade em pistas de áudio;
- Uma pequena quantidade ou uma completa falta de situações de "game over" que desencorajam o jogador a continuar jogando;
- Uma maneira única e clara de resolver os problemas levantados pelo cenário;
- Uma estratégia de aprendizado intuitiva, integrada ao desenvolvimento do jogo, para que o IDV entenda os requisitos sem o uso de instruções humanas ou de texto-fala;
- Interatividade, engajamento e diversão, para que os IDV se beneficiem de uma experiência agradável e divertida;

[4] Computer games play an important role in the development of mental structures as they enhance learning, improve creativity and problem-solving skills, stimulate motivation and resolve communication issues. Unfortunately, due to their obvious physical and cognitive limitations and because video games rely primarily on graphical information, visually impaired people are unable to play most of the computer games available on the market today

[5] Audio games, however, are a reliable training modality that improves orientation and mobility through an accessible, pleasant and entertaining approach that is suitable for the requirements of the visually impaired.

Um dos grandes desafios pros desenvolvedores é a falta de conhecimento dessas especificidades que os IDV necessitam, o que tornam as produções ainda mais escassas, para isso, o autor aborda também o WACAG 2.0 (*Web Content Accessibility Guidelines*[6], que atualmente está na versão 2.2), no qual estabelece bases para o desenvolvimento de *audiogames* acessíveis para pessoas cegas, que são resumidamente:

- **Percepção:** A informação deve ser apresentada de maneira acessível e compreensível;
- **Operabilidade:** Os usuários devem poder interagir e se comunicar com a interface;
- **Compreensão do conteúdo:** A operação e manipulação da interface deve ser facilmente acessível;
- **Robustez:** O jogo deve permanecer acessível mesmo com o avanço da tecnologia;

Balan et al. (2014) fala que o processo de sonificação em *audiogames* deve se basear em um correlação bem definida entre o som e a informação, assim aumentando a inteligibilidade e a facilidade de acesso do reprodutor. O autor também separa as pistas de áudio em 2 grupos, o primeiro associado a elementos de interface e o segundo contendo a representação do jogo.

> Nos jogos de áudio, o processo de sonificação deve se basear em uma correlação bem definida entre som e informação. Para aumentar a inteligibilidade e a facilidade de acesso do reprodutor, os dados de áudio da correspondência devem incluir metáforas e outras associações compreensíveis. As dicas de áudio devem ser separadas em 2 grupos: o primeiro, informações auditivas relacionadas aos elementos gráficos e de design da interface - fornecem dicas sobre requisitos, tarefas e resultados e, por outro lado, fluxos de áudio contínuos que fazem parte do modelo de representação do jogo.[7] (BALAN et al., 2014, p.689, tradução nossa).

Outra classificação levantada pelo autor divide os dados de som em 3 categorias: ícones auditivos, *earcons* ou pedaços de música e informações sonoras auditivas.

[6] Diretrizes de Acessibilidade de Conteúdo para Web. (Atualmente está na versão 2.2).
[7] In audio games, the process of sonification should be based upon a well-defined correlation between sound and information. To increase the intelligibility and ease of access for the player, the correspondence audio-data should include metaphors and other comprehensible associations. The audio cues have to be separated into 2 groups: the first, auditory information related to the graphical and design elements of the interface- they give clues regarding requirements, tasks and results and, on the other hand, continuous audio streams that are part of the model of representation of the game.

> [..]outra classificação divide os dados sonoros em 3 categorias: ícones auditivos, ícones sonoros (*earcons)*- trechos curtos de música, ambas com o objetivo de notificar o jogador com relação à sua evolução no jogo e na terceira categoria, informações auditivas de sonificação que codificam dados abstratos contextuais para som sem fala.[8] (BALAN et al., 2014, p.689, tradução nossa).

Outra característica essencial nos *audiogames* é a sensação espacial, como a distância e a direção de elementos sonoros que indicam a fonte sonora, assim definindo a percepção auditiva do espaço. Balan et al. (2014) também descreve que a interação entre o jogador e o jogo é realizada no meta-nível de percepção, através da alteração de características como a tonalidade do som, volume, timbre ou utilizando filtros HRTF (*Head-related transfer function*) para fornecer informações de distância e direção.

Os *audiogames* nas plataformas móveis apresentam a mesma variedade de gênero que os jogos baseados em vídeo, como jogos de aventura, ação, terror, corrida, RPG e etc. (ARAÚJO et al.,2015). Neles os desenvolvedores utilizam o áudio binaural, técnica que simula o efeito tridimensional, assim como ouvimos os elementos sonoros no mundo real, para posicionar e dispor os sons no espaço sonoro 3D. Os dispositivos móveis, diferente dos computadores, possuem diferentes maneiras de interação do jogador com o jogo, por exemplo, o *touchpad,* acelerômetro, GPS ,bússola e outros sensores.

Sobre as questões de acessibilidade em dispositivos móveis, Araújo et al (2015), faz um estudo das principais recomendações de acessibilidade aplicada em jogos, como resultado, foi desenvolvido um conjunto de recomendações para avaliação de acessibilidade de *audiogames* móveis, sendo estas:

- **Textos alternativos:** Devem oferecer alternativas textuais para todo o conteúdo não textual apresentado na interface móvel;
- **Adaptabilidade:** Devem fornecer conteúdos e interfaces que possam ser modificados (alterando resolução ou utilizando apenas áudio) sem perder informação essencial;
- **Ambientação:** O jogador deve se situar com facilidade no cenário a partir dos conteúdos, incluindo separar, através do som, o primeiro plano do plano de fundo na interface móveis;

[8] [...]another classification divides sound data into 3 categories: auditory icons, earcons- short pieces of music, both of which have the purpose to notify the player in respect with his evolution in the game and the third category, sonification-auditory information that encodes contextual abstract data to non-speech sound.

- **Operabilidade:** Os componentes da interface,interação e navegação do jogador devem ser operáveis, de fácil entendimento e alteração pelo jogador (opções de início rápido, para que não tenha que percorrer muitos níveis de informação);
- **Facilidade de configuração:** Permitir que o jogador ajuste, simplifique e salve os controles e configurações do jogo (idioma, resolução, volumes, leitor de tela e etc);
- **Assistência e tutorial:** Oferecer a documentação e modos tutoriais objetivos e de fácil acesso;

2.2 *Engines* de áudio

Engines de áudio (também chamadas de *game sound engines, game audio engines* ou *audio middlewares*) são ferramentas utilizadas para implementar os elementos sonoros em um jogo, de maneira a programar e dinamizar eventos sonoros em tempo real. Esses *middlewares* são programas de computador, ou *softwares* que oferecem serviços para outros programas e plataformas (como *game engines*[9]). Amengual et al. (2019) diz que as *game audio engines* precisam lidar com os efeitos de espacialização e gerenciamento de vários elementos sonoros em tempo real. Entre as principais funcionalidades ou efeitos oferecidos pelas *audio engines* podemos destacar: *panorama*, *chorus*, *delay*, *reverb*, *flanger*, *pitch shifter*, *phaser* entre outras. Devemos destacar também que a quantidade de computação disponível no jogo para essa tarefa geralmente é bastante pequena, portanto cabe a *engine* de áudio otimizar esse recurso.

> Os mecanismos de áudio do jogo precisam lidar com os efeitos de espacialização e propagação de várias fontes em tempo real, e a quantidade de computação disponível para isso geralmente é bastante pequena. Além disso, ao contrário das simulações acústicas arquitetônicas, no áudio do jogo as cenas são tipicamente dinâmicas, com fontes móveis, ouvinte e geometria.[10] (AMENGUAL et al., 2019, p.2, tradução nossa)

[9] As game engines, são plataformas de desenvolvimento de games que facilitam o uso de ferramentas e de linguagens de programação, das quais algumas possuem integração direta com as engines de áudio.

[10] Game audio engines need to handle the spatialization and propagation effects of multiple sources in real-time, and the amount of compute available for this is usually fairly small. In addition, as opposed to architectural acoustic simulations, in game audio the scenes are typically dynamic, with moving sources, listener and geometry.

Muitos desenvolvedores preferem programar seus próprios jogos do zero, sem auxílio das *games engines*, para isso existem também linguagens de programação que podem ser utilizadas na manipulação e criação de sons.

Entre ass principais *audio engines* disponíveis no mercado, podemos destacar: *FMOD Studio, Wwise, Elias e Miles Sound System*. *FMOD Studio* é um *middleware* que propõe soluções de áudio adaptativo para games, possui uma versão gratuita para desenvolvedores independentes (possui integração com *Unity* e *Unreal Engine*). *Wwise* é um conjunto de soluções de áudio interativo para games (possui integração com *Unity* e *Unreal Engine*). *Elias* é um *middleware* de criação e implementação de música adaptativa para games (possui integração com *Unity* e *Unreal Engine*). *Miles Sound System* é um *middleware* de integração de áudio digital para jogos 3D e 2D desenvolvido pela *RAD Game Tool* (disponível para todas as plataformas).

Entre as principais linguagens de programação que podem ser utilizadas na manipulação de sons no contexto dos *audiogames* são: *Max, SuperCollider* e *Pure Data*. *Max* é uma linguagem de programação visual para música e multimídia desenvolvida pela *Cycling 74'* (o *software é pago* e está disponível para *Windows XP* e *Mac OS X*). *SuperCollider* é uma plataforma para síntese de áudio e composição algorítmica, gratuito e de código aberto (disponível para *Windows, Mac e Linux*). *Pure Data* é uma linguagem de programação que utiliza recursos visuais para manipulação de dados puros. Entre as diversas utilizações do *Pure Data* está a síntese e processamento de sinais de áudio. (disponível para todas as plataformas).

2.3 *Pure Data*

Puckette (1997) desenvolve o *Pure Data*, uma linguagem de programação visual *open source* utilizada para manipulação de multimídias (ou "dados puros"). O *Pd* está disponível para *Windows, MAC OS X, Linux* e smartphones (via *libpd, DroidParty e PdParty*), ele permite que músicos, artistas visuais, artistas, pesquisadores e desenvolvedores criem *softwares* graficamente sem precisar escrever linhas de código. O *Pd* permite realizar integrações de maneira fácil em redes locais e remotas, sistemas de motores, iluminação e outros equipamentos.

Entre as diversas maneiras de utilizar o *Pure Data* podemos destacar a síntese e processamento de sinais de áudio em tempo real. O *Pd* oferece uma interface visual intuitiva e de fácil manuseio dos dados, no nosso caso os sinais de áudio, o que torna a programação uma etapa da produção mais simples e rápida.

> A lógica por trás do design do *Pd* é facilitar o empilhamento de grandes coleções de dados, alguns dos quais podem ser obtidos através da análise de sinais de áudio (e alguns dos quais podem ser sinais de áudio), que podem ser considerados uma partitura de computador para ser "tocado" por um patch semelhante ao Max que o atravessa.[11] (PUCKETTE, 1997, p.3, tradução nossa).

O *Pd* utiliza o conceito de estruturas hierárquicas de dados, permitindo que o usuário possa definir e estruturar uma grande quantidade de dados de maneira lógica e organizada para o processamento (figura 1). Devido a linguagem simples e de código aberto, o *Pd* pode facilmente ser integrado em outros sistemas e plataformas, permitindo diversas utilidades e aplicações.

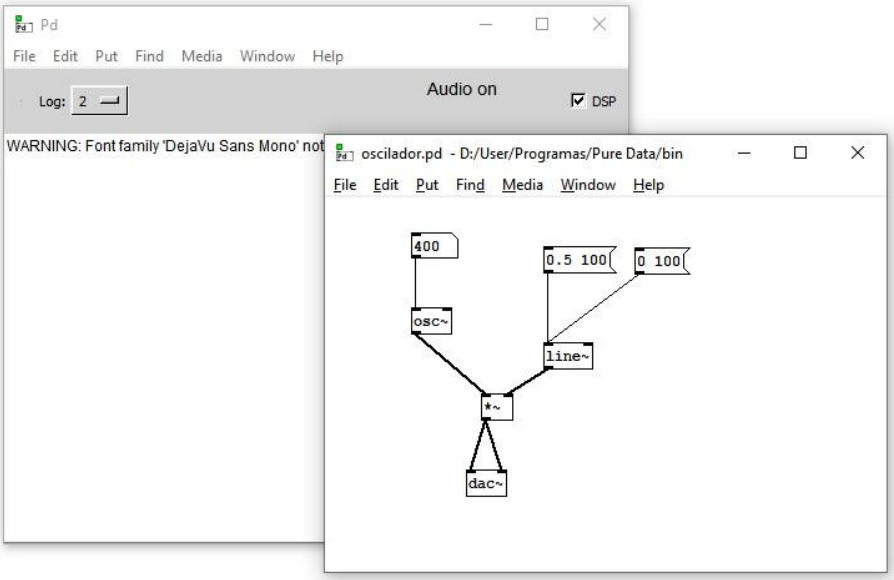


Figura 1 - Interface visual de programação do *Pure Data*. Fonte: o autor.

[11] The rationale behind Pd's design is to make it easy to pile up large collections of data, some of which might be obtained through analysis of audio signals (and some of which could be audio signals), which could be regarded as a computer music score to be "played" by a Max-like patch that traverses it.

Brinkmann et al. (2011) desenvolve o *libpd*, que transforma o *Pure Data* em uma biblioteca de áudio incorporável, a qual vem com ligações para *Java*, *Processing*, *Objective-C* e *Python*, além de suporte para *Android* e *IOS*.

> A *libpd* é um exercício de desenvolvimento subtrativo: removendo os recursos de áudio, interface do usuário, tempo e threading do Pd, é possível separar as preocupações da ferramenta, deixando uma biblioteca mais flexível e incorporável. Essa biblioteca pode ser executada no contexto de outros aplicativos, como o fornecimento de um mecanismo de música interativo para um jogo ou ferramenta de desempenho, além de funcionar mais facilmente em plataformas móveis como iOS e Android. (BRINKMANN et al., 2011, p.1, tradução nossa).

O principal objetivo do *libpd* é extrair do *Pd* a funcionalidade de processamento de sinal e disponibilizá-la como um retorno do processamento de áudio. O segundo objetivo é permitir a troca direta de mensagens de controle e informações MIDI entre o *Pure Data* e o código do usuário. O *libpd* dá suporte à funções *bang, float e symbol messages* (exceto as *pointer messages*) do *Pd.* O designer de som pode ficar dentro dos limites da interface gráfica do usuário e fluxos de dados do *Pd*, sem precisar ter conhecimento prévio da linguagem de programação C++, da mesma maneira o programador não precisa do conhecimento específico do *Pd* para implementação em seu jogo.

> Construir um patch para *libpd* não é diferente de criar um patch para o próprio Pd. Para preparar o patch para implantação, o designer de som precisa atribuir apenas os símbolos de envio e recebimento apropriados aos elementos da GUI. No contexto da libpd, o código do cliente se comunicará com esses símbolos de envio e recebimento programaticamente, enviando mensagens de eventos e sensores da GUI para o *Pd* ou atualizando sua própria GUI em resposta às mensagens do Pd. Agora, o programador do aplicativo pode simplesmente carregar o patch e usá-lo como uma caixa preta.[12] (BRINKMANN et al., 2011, p.2, tradução nossa).

O *libpd* permite também a integração do *Pd* com os dispositivos móveis, como por exemplo, o sistema operacional do *Google*, o *Android*. Segundo Brinkmann et al. (2011) o desenvolvimento de áudio para *Android* apresenta uma série de desafios, sendo um deles a grande latência de entrada e saída de ida e volta (até centenas de milissegundos, dependendo

[12] Building a patch for libpd is no different from building a patch for Pd itself. In order to prepare the patch for deployment, the sound designer only has to assign the appropriate send and receive symbols to the GUI elements. In the context of libpd, client code will communicate with these send and receive symbols programmatically, sending messages from GUI events and sensors to Pd, or updating its own GUI in response to messages from Pd. Now the application programmer can simply load the patch and use it as a black box.

do hardware), além da grande variedade de dispositivos, para isso o *AudioWrapper.java* lida com esses problemas o máximo possível.

Paul (2003) utiliza o *Pure Data* para prototipar áudio para games, utilizando de forma semelhante a uma *audio engine*, onde cria *patches* para gerenciar os arquivos de áudio de um protótipo de jogo. Para Paul (2003) existem quatro áreas do áudio para jogos: música, falas, efeitos sonoros e sons interativos (que reagem em tempo real).

> Além da música, fala e efeitos sonoros, os jogos também usam uma quarta categoria de áudio. Essa categoria de áudio tenta simular sons altamente complexos, como o rugido de uma multidão inteira ou o som de um motor de carro de corrida. Esses sons são compostos de muitos parâmetros dinâmicos que modulam o conteúdo de áudio em tempo real.[13] (PAUL, 2003, p.3, tradução nossa)

Utilizando o *Pd* é possível criar uma série de *patches* e *subpatches* para o gerenciamento de áudio em um jogo, seja da música, falas, efeitos sonoros e também como ferramenta de música adaptativa.

Neste trabalho, nós utilizamos o "*LibPdIntegration*" disponibilizado por Niall Moody na plataforma *github*[14], no qual disponibiliza o código fonte que permite a integração do *LibPd* com a plataforma *Unity*. Também nos baseamos no trabalho desenvolvido por Yann Seznec, que disponibiliza em seu canal do *YouTube*[15] uma série de vídeos, desenvolvendo e ensinando como utilizar o *LibPdIntegration.*

2.4 Plataforma *Unity*

A *Unity* (também conhecida como *Unity3D* ou *UnityEngine)* é uma plataforma de desenvolvimento 3D em tempo real, ou *game engine* (motor de jogo). Foi desenvolvida pela *Unity Technologies*, lançada em 2005 e está disponível para *Windows*, *Mac OS X* e *Linux*. Ela consiste em um motor de renderização e motor físico, e possui uma interface gráfica de usuário chamada *Editor Unity* (figura 2).

---

[13] Besides music, speech and sound effects, games also use a fourth category of audio. This category of audio attempts to simulate highly complex sounds, such as the roar of an entire crowd or the sound of a racing car engine. These sounds are made up of many dynamic parameters which modulate the audio content in realtime.
[14] <https://github.com/LibPdIntegration/LibPdIntegration> Acessado em: 19/07/2021.
[15] <https://www.youtube.com/user/amazingrolo/featured> Acessado em: 19/07/2021.

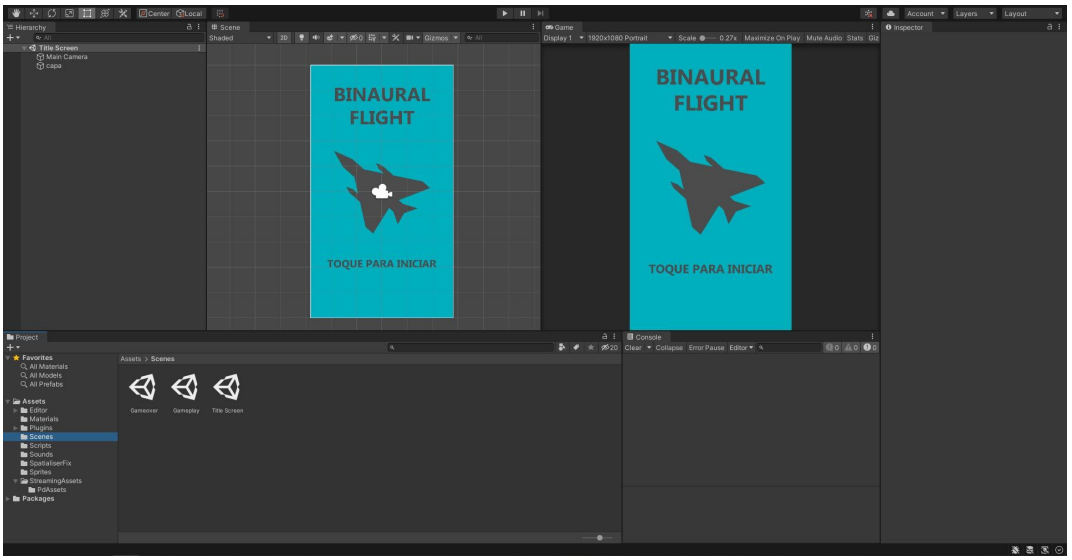


Figura 2 - Interface da plataforma *Unity*. Fonte: O autor.

A *Unity* é utilizada tanto para o desenvolvimento de jogos em 3D quanto em 2D, e possibilita a portabilidade dos jogos para múltiplas plataformas dentro de um projeto, como web browsers, desktops, consoles e para dispositivos móveis.

Por ser uma ferramenta popular, o acesso a documentação e tutoriais de uso é ampla na internet, este é um diferencial importante na escolha da plataforma de desenvolvimento para o nosso *audiogame*. A plataforma *Unity* utiliza *scripts* de programação na linguagem *C#* (*C Sharp,* figura 3).

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class scroll : MonoBehaviour{
6
7       public float speed = 0.5f;
8
9       void Start()
10      {
11
12      }
13
14
15      void Update()
16      {
17          Vector2 offset = new Vector2 (0, Time.time * speed);
18
19          GetComponent<Renderer>().material.mainTextureOffset = offset;
20      }
21  }
```

Figura 3 - Parte do código (*script C#)* utilizado no *Binaural Flight*. Fonte: O autor.

3. TRABALHOS RELACIONADOS

Como resultado da revisão sistemática literal (seção 3.1 deste estudo) realizada neste estudo, montamos um panorama geral das publicações e produções sobre a temática desta pesquisa. Podemos observar que os estudos mais recentes vem de diversas áreas diferentes, porém, grande parcela deles partem da necessidade de inclusão e acessibilidade do do IDV no mundo dos videogames. Nesta seção vamos discutir sobre os trabalhos semelhantes, nos quais existem diversos estudos sobre os *audiogames* em dispositivos móveis, porém a utilização do *Pd* como *engine* de áudio em *audiogames* móveis ainda é escassa.

Escarrone, Cheiran e Moura (2016), Da Conceição et al. (2016), Hansen e Hiraga (2017) e Avila et al. (2020) desenvolvem *audiogames* para dispositivos móveis, respectivamente: um *audiogame* de Batalha Naval, *Ortomonstro*, *Music Puzzle* (o qual utiliza *Pure Data*) e o *Soundmaze.*

Escarrone, Cheiran e Moura (2016) trazem um estudo no qual através do desenvolvimento de um áudio jogo tentam identificar as características da acessibilidade web mobile. Os autores destacam a grande procura por jogos para dispositivos móveis, visto a popularidade da plataforma, porém muitos jogadores acabam ficando de fora por terem alguma deficiência. Eles propõem o desenvolvimento do jogo de tabuleiro Batalha Naval (figura 4) no formato de áudio jogo para dispositivos móveis, o qual rodará diretamente nos navegadores.

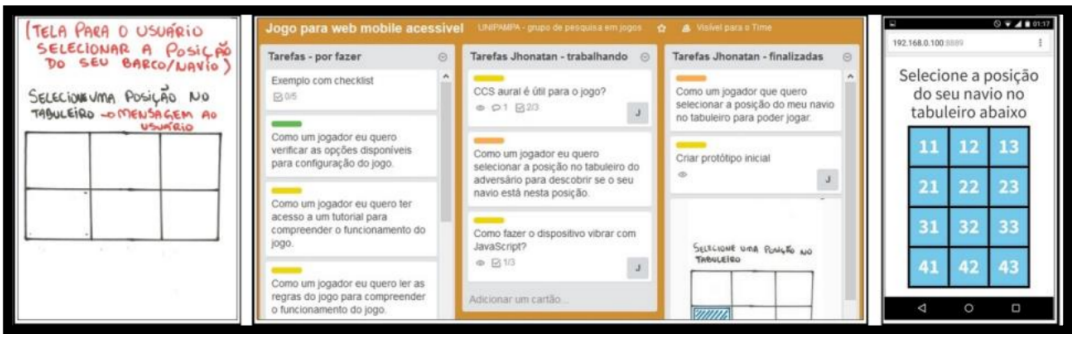


Figura 4 - Protótipo inicial em papel (à esquerda), demonstração da organização das tarefas no Kanban (ao centro) e versão preliminar do jogo (à direita). Fonte: (ESCARRONE, CHEIRAN E MOURA, 2016).

Da Conceição et al. (2016) desenvolve o jogo *Ortomonstro*, que consiste em um *audiogame* customizável para práticas ortográficas de português por meio de Braille para dispositivos móveis. O autor desenvolve o áudio jogo móvel acessível para pessoas com

deficiência visual voltado para educação, além do aplicativo móvel *Android* também é oferecido uma interface *Web* para customização e geração de conteúdos do *audiogame.*

Hansen e Hiraga (2017) conduzem um experimento usando um *audiogame* chamado *Music Puzzle* (figura 5) com estudantes universitários japoneses com diferentes níveis de percepção auditiva e experiência com música. O game foi desenvolvido para dispositivos móveis *(Android)* e utiliza uma biblioteca do *Pure Data* como *engine* de áudio em tempo real. O jogo é do tipo quebra-cabeça, e tem 3 modos, modo fala, modo música e um terceiro que é um mix dos dois anteriores. No primeiro modo o jogador escuta o quebra-cabeça inteiro uma vez, depois reordena as peças e altera a tonalidade e a EQ de forma apropriada. No modo música é necessário memorizar e entender não apenas a melodia e o ritmo, mas também o timbre e outras características. No terceiro modo mistura os dois conceitos anteriores, aumentando a dificuldade.

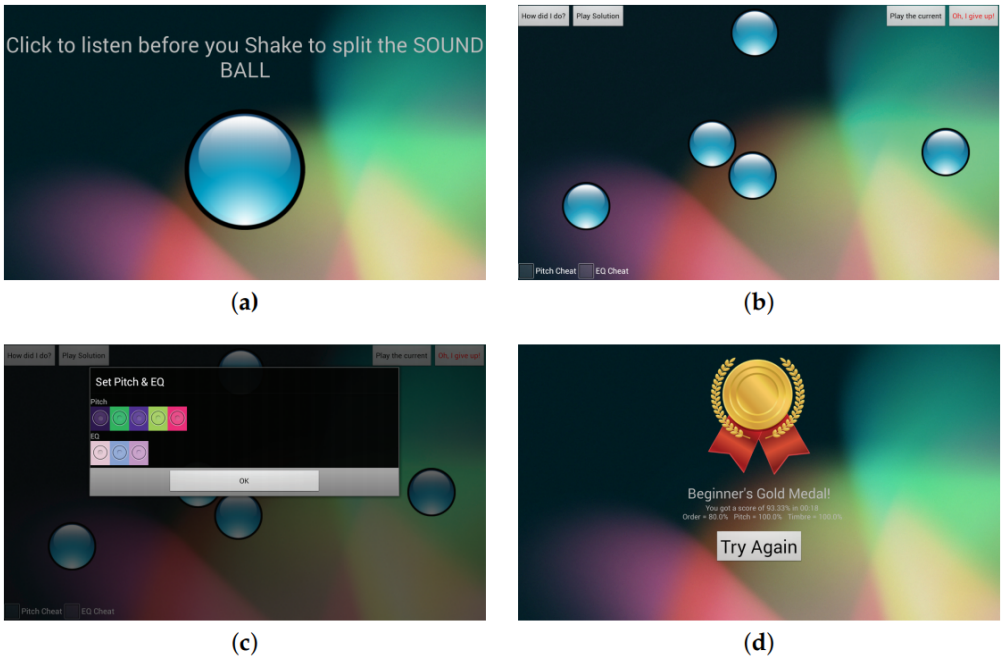


Figura 5 - A interface de jogo do Music Puzzle como vista em um tablet. (a) Inicie uma sessão, ouça a peça musical alvo e agite o tablet; (b) Ouvir e ordenar objetos sonoros pelo toque do dedo. Existem quatro botões de ação: Como eu me saí? (avalie a ordem atual), Jogue a Solução (repita a peça alvo), Jogue a atual (jogue a ordem como vista na tela), e Oh, eu desisto (saia do quebra-cabeça); (c) Ajuste o tom e a equalização (EQ; filtragem) para cada objeto. Os botões de opção são coloridos aleatoriamente e ordenados de forma a não dar nenhuma pista visual para a solução; (d) Quebra-cabeça preenchido com uma avaliação. Fonte: (HANSEN e HIRAGA, 2017).

Avila et al. (2020) desenvolvem o *Soundmaze* (figura 6), que tem como proposta ser um *audiogame* para dispositivos *Android* para indivíduos com deficiência visual. O jogo é

ambientado em um labirinto onde o jogador deve encontrar a saída de cada fase com o menor número de passos e sem gastar todas as suas vidas, cada nível possui sua chave correspondente escondida no labirinto. Durante o estudo, o autor faz uma apresentação de todas as fases do desenvolvimento do *audiogame* para dispositivos móveis com sistema operacional Android, o mesmo destaca e enfatiza que alguns cuidados devem ser tomados, como o excesso de detalhes apresentados ao usuário, quantidade e qualidade dos elementos sonoros a fim de não confundir o jogador durante a jogatina.

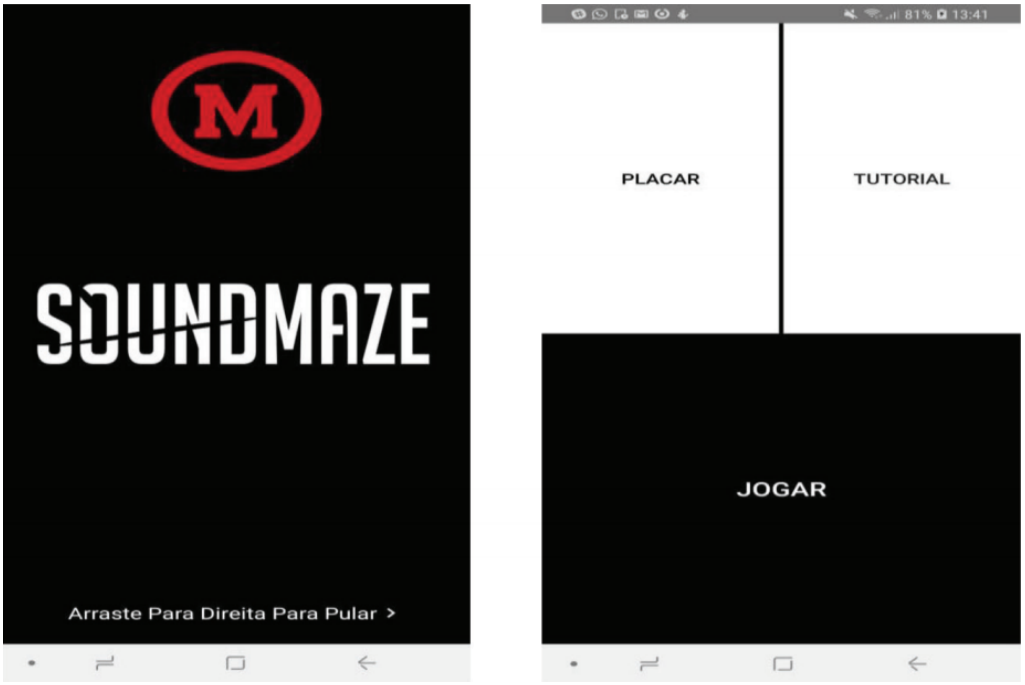


Figura 6 - Telas iniciais. (A) Splash screen apresentada ao usuário. (B) Menu apresentação ao usuário. Fonte: (AVILA ET AL., 2020).

De Lima (2019) e Monteiro et al. (2016) realizam uma análise de *audiogames* com proposta de acessibilidade para os IDV, ambos os jogos dos respectivos estudos possuem versões para dispositivos móveis. De Lima (2019) faz uma análise do áudio jogo *Audio Game Hub*, que tem como proposta ser uma experiência sensorial para pessoas com deficiência visual. O autor destaca o uso do som binaural como elemento de espacialização do som, permitindo ao jogador identificar o posicionamento dos elementos sonoros no espaço 3D e conclui enfatizando a importância da inclusão e da discussão sobre a desigualdade no mundo dos jogos eletrônicos.

Já Monteiro et al. (2016) faz uma análise das estruturas de áudio e efeitos sonoros do jogo *Blindside*, que consiste em um *audiogame* disponível para *IOS*, *Windows* e *MAC*.

Blindside é um jogo de terror com a proposta de não possuir elementos visuais, apenas o som como grande elemento de imersão e mecânica. O autor realizou um teste com 12 pessoas, sendo 8 homens e 4 mulheres com idades entre 18 e 13 anos, utilizando o jogo em dois dispositivos móveis com fones de ouvido, e questionário e uma entrevista após a aplicação do teste, com isso, o autor destaca após analisar os dados obtidos a importância do design de som em *audiogames*, tanto na construção da narrativa bem como na imersão do jogador, considerando um elemento vital na manutenção da imersão do jogador.

Kane, Koushik e Muehlbradt (2018) desenvolvem o Bonk, que é um *framework* para auxiliar pessoas com deficiência visual a programar *audiogames*. A proposta consiste em um conjunto de ferramentas acessíveis para iniciantes em programação que possuem deficiência visual, além disso desenvolvem um *framework* de programação de audiogame baseado em JavaScript que otimiza a criação de *audiogames* interativos.

Vendome et al. (2019) traz um estudo empírico sobre a acessibilidade em aplicativos *Android*, a partir do ponto de vista do desenvolvedor, no qual destaca os princípios universais de design, que tem como objetivo melhorar a acessibilidade considerando durante o design do produto todos os tipos de usuários, incluindo os que portam algum tipo de deficiência. Os autores também abordam durante sua pesquisa um estudo que propõe um manual de avaliação de *mobile audio games*. Como resultado da pesquisa, eles realizaram uma coleta de posts no *StackOverflow* onde desenvolvedores expõem suas dúvidas e compartilham experiências, a fim de identificar as principais dificuldades dos desenvolvedores mobile. Entre alguns dos resultados eles observaram que os serviços de acessibilidade são comumente usados para aspectos de não acessibilidade.

Da Silva (2019) faz um estudo buscando promover a empatia pela experiência do usuário com deficiência visual no ensino de game design. O autor dirige o estudo com um grupo de estudantes do curso de games, onde eles tinham o propósito de desenvolver *audiogames* com temática livre, para qualquer plataforma e dimensionalidade (2D/3D), tipos de interação, história e narrativa. O autor faz a análise de sete dos vinte jogos desenvolvidos, e destaca que é notável que os alunos que fizeram parte do experimento demonstraram, em sua maioria, empatia pelos jogadores que possuem deficiência visual enquanto pesquisavam e produziam os jogos baseados em suas principais necessidades. Destacando também a importância da inclusão desse público no mercado atual de games.

Wiermans (2016) apresenta um estudo sobre a viabilidade da música procedural, nele é desenvolvido um sistema de música procedural para um jogo chamado *Remnant* (figura 7), estilo arcade em um ambiente espacial desenvolvido para dispositivos móveis. Nele é utilizado o *Pure Data* por ser uma linguagem simples de entender mas muito poderosa, o autor utiliza também o *libpd* para realizar a implementação do *Pd* na *engine Unity.* Com este estudo podemos destacar a viabilidade de utilizar o *Pd* juntamente com o *libpd* no desenvolvimento de jogos para dispositivos móveis.

Figura 7 - Screenshots retiradas do jogo Remnant. Fonte: (WIERMANS, 2020).

Passos (2020) utiliza o *Pure Data* no desenvolvimento do *audiogame tonaudio* para dispositivos móveis, um jogo de habilidade de memorização e percepção sonora para o reconhecimento de sequências de intervalos musicais. O teve sua interface de usuário criada inteiramente no *Pd*, como um *patch* único, o jogo pode rodar em dispositivos móveis com a utilização do aplicativo gratuito *MobMuPlat*.

4. MATERIAIS E MÉTODOS

Em termos metodológicos, esse trabalho se trata de uma pesquisa aplicada, na qual pretendemos realizar um levantamento geral de conceitos e técnicas tanto do meio acadêmico quanto do mercado sobre o desenvolvimento de *audiogames* para dispositivos móveis. Prodanov (2013) diz que a pesquisa aplicada tem como objetivo gerar conhecimentos para o uso prático voltado para a solução de problemas específicos. Para alcançarmos o objetivo vamos utilizar o conceito de pesquisa exploratória também levantado por Prodanov (2013), que diz que este tipo de pesquisa tem como objetivo levantar informações sobre o tema a ser investigado, assim criando um panorama e assim facilitar a delimitação da pesquisa.

A pesquisa exploratória envolve diversas ações, como: levantamento bibliográfico, entrevistas, pesquisas e levantamento de material audiovisual e outros. Para criar o panorama sobre o tema, vamos realizar uma revisão sistemática literal, ou revisão de literatura, que tem como objetivo verificar o estado da arte sobre a temática, assim verificar os textos e publicações mais recentes e relevantes sobre o tema levantado.

> Nessa fase, devemos responder às seguintes questões: quem já escreveu e o que já foi publicado sobre o assunto, que aspectos já foram abordados, quais as lacunas existentes na literatura. Pode objetivar determinar o "estado da arte", ser uma revisão teórica, ser uma revisão empírica ou ainda ser uma revisão histórica. (PRODANOV, 2013, p.78).

Segundo Sampaio (2007) uma revisão sistemática assim como outras pesquisas de revisão utiliza a literatura sobre um assunto específico como uma fonte de dados. Com esse tipo de pesquisa é possível verificar e identificar temas que necessitam de mais evidência no meio acadêmico, assim esse tipo de pesquisa objetiva auxiliar futuras pesquisas sobre o tema.

Para a coleta dos dados é necessário o desenvolvimento de um protocolo de pesquisa, o qual contém parâmetros que ajudem a delimitar e afunilar o escopo da pesquisa, o qual deve ser empregado com rigor crítico na escolha dos estudos que vão compor os dados da pesquisa.

A *string* de busca são as palavras chaves utilizadas nos repositórios de publicações para encontrar os estudos sobre o tema, definimos também quais os idiomas que serão incluídos na revisão, o período temporal e também os repositórios que possuem as principais publicações da área. Após a definição do protocolo de busca vamos definir os critérios de

inclusão e exclusão dos dados na revisão sistemática, os quais servem para definir a relevância dos estudos obtidos.

Os dados que serão submetidos ao protocolo vão resultar em publicações relevantes e atuais sobre a temática da pesquisa, com isso teremos um panorama geral sobre o que tem se produzido na área. Para dar apoio a pesquisa bibliográfica vamos realizar também um levantamento das *engines* de áudio utilizadas no desenvolvimento de jogos.

Para a melhor visualização de determinados fluxos de trabalho deste estudo, buscamos inspiração nos modelos de arquitetura da informação propostos por Zachman (1987), no qual descreve uma série de indicações e formas para representar e descrever dados. O modelo de sistema de informação baseado na perspectiva do designer (figura 8), no qual faz um fluxo de "entidade-relacionamento-entidade" que no contexto do designer pode ser interpretado como "máquina-dados-máquina".

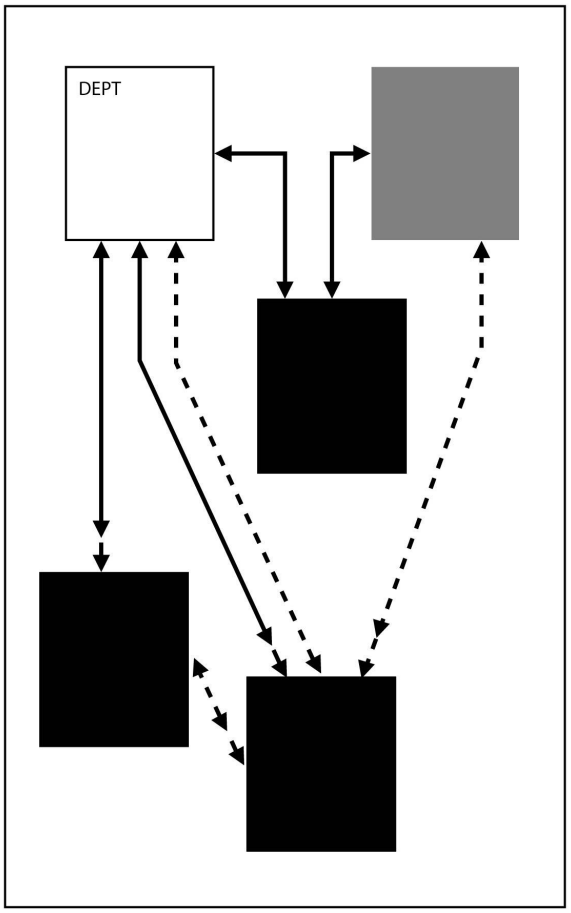


Figura 8 - Modelo conceitual de sistema de informação baseado na perspectiva do designer. Fonte: (ZACHMANN, 1987).

Para testar as soluções propostas por esta pesquisa, desenvolvemos um protótipo de *audiogame* chamado *Binaural Flight,* apenas para fins de teste das funcionalidades e implementação do *libpd.* Para isto, utilizamos a plataforma de desenvolvimento de jogos *Unity*, a qual permite o desenvolvimento de jogos 3D e 2D, e possibilita também direcionar o projeto para diversas plataformas, incluindo os dispositivos móveis. O *Unity* é uma dos motores de jogos (*game engine*) mais populares e utilizadas no mercado.

O percurso do trabalho se deu da seguinte maneira, iniciamos com a revisão sistemática, aplicando o protocolo de busca e criando um panorama geral sobre a temática (seção 4.1 deste estudo), onde observamos a utilização do *Pd* no desenvolvimento de *audiogames mobile*. Em seguida, fizemos o levantamento teórico a fim de guiar, embasar e fundamentar esta pesquisa, nele definimos os principais conceitos que utilizamos.

Parte do trabalho também consistiu em estudos e práticas de programação, tanto da linguagem visual *Pure Data* para o desenvolvimento dos *patches Pd*, quanto na linguagem de programação *C#*, esta segunda tem maior destaque por ser pré-requisito para o desenvolvimento do *audiogame mobile*. Estudamos também a plataforma de desenvolvimento *Unity,* que é responsável pelo mecanismo, ou *engine*, do jogo propriamente dito.

Após o período de estudo sobre desenvolvimento de games, iniciamos a produção dos assets do jogo, onde utilizamos o software de edição e criação de vetores *Adobe Illustrator*, nele criamos a interface visual e elementos guia no estilo *flat design*. Para a criação de alguns dos sons, utilizamos a *DAW Reaper*, onde é possível criar e manipular elementos sonoros, criar o desenho de som dos elementos. No *Pure Data* desenvolvemos os *patches Pd* a serem utilizados no *Binaural Flight*.

Na plataforma *Unity,* criamos os *scripts* para controlar e receber os *inputs touch screen,* movimentação, progressão de cenário, jogabilidade e outros elementos do jogo (os quais são detalhados na seção 5 deste estudo).

Reservamos uma seção inteira (seção 5 deste estudo) para a tratarmos do desenvolvimento do *audiogame Binaural Flight,* onde descrevemos o desenvolvimento do *audiogame* na plataforma *Unity*, *design* de áudio, utilização de *patches Pd,* implementação do *Libpd*, comunicação entre *Libpd* e *Unity* e o *LibPdIntegration* no jogo *Binaural Flight.*

No Anexo III descrevemos todos os materiais e equipamentos utilizados para o desenvolvimento do *audiogame Binaural Flight* e também no decorrer desta pesquisa.

Como mencionamos anteriormente, na subseção seguinte apresentamos a revisão sistemática literal, a qual nos permite visualizar um panorama geral sobre a temática principal desta pesquisa.

4.1 Revisão Sistemática da Literatura

Como descrito anteriormente, a revisão sistemática requer certo rigor crítico e uma sequência de passos coesos que possam ser replicados e assim qualquer pessoa possa obter os mesmo resultados obtidos pelo pesquisador. Para isso utilizamos um sequência de passos pré estabelecidos que pode ser verificado na figura 9:

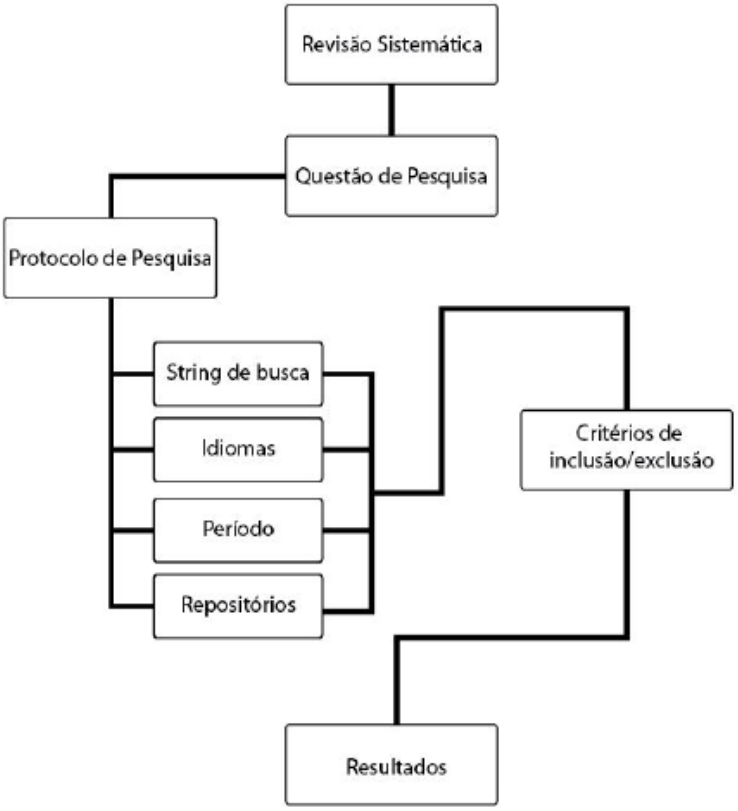


Figura 9 - Fluxograma da Revisão Sistemática Literal. Fonte: O autor.

4.1.1 Questão de pesquisa

O objetivo desta revisão sistemática é verificar a literatura mais recente sobre o desenvolvimento de *audiogames* para pessoas com deficiência visual nos dispositivos móveis, da mesma forma investigar o uso do *Pure Data* como *engine* de áudio na

implementação dos jogos. Com isso, estabelecer um panorama geral sobre a temática, verificar o que se tem falado e quais os avanços mais recentes no meio acadêmico. Para guiar esta revisão sistemática, vamo tomar como base as seguintes questões:

- Como se dá o desenvolvimento de *audiogames* para indivíduos com deficiência visual nos dispositivos móveis?
- Quais as ferramentas de áudio necessárias e mais utilizadas no desenvolvimento dos *audiogames*?
- Já se Utiliza o *Pure Data* como *engine* de áudio no desenvolvimento destes jogos?
- É possível viabilizar o uso do *Pure Data* como *engine* de áudio na criação de *audiogames* mobile?

Com essas questões podemos observar inicialmente o contexto no qual ela está inserida. Utilizando os motores de busca da internet é possível observar algumas produções acadêmicas. Após esta etapa inicial de contextualização foi possível identificar os termos, palavras e sinônimos utilizados que são comumente associados à temática nos artigos.

4.1.2. Protocolo de pesquisa

Na revisão sistemática é necessário a elaboração de um protocolo de pesquisa, o qual define parâmetros de busca que são aplicados durante a coleta dos artigos. Este processo tem como objetivo filtrar os estudos que farão parte da revisão, e assim tornar este processo replicável e cientificamente rigoroso.

Durante os primeiros testes de pesquisa é onde o escopo dos termos de busca são refinados, como resultado obtemos a *string* de busca, que são palavras chaves que permitem a busca dos artigos nos repositórios e indexadores de busca. Neste estudo utilizamos como base o período de 2016 a 2021, referente aos últimos 6 anos de publicações na área. Foram incluídos neste estudo apenas os trabalhos nos idiomas inglês e português, e devido aos termos que se referem a temática serem diferentes utilizamos duas *strings* distintas para cada idioma. Em seguida utilizamos as *strings* de busca nos repositórios e indexadores de busca (tabela 1 contendo os parâmetros de busca) para a coleta dos estudos.

Protocolo de Pesquisa	
***String* em português**	“Jogos acessíveis” OR “jogos inclusivos” OR "audiogames" OR "áudio jogos" AND “deficiência visual” OR “acessibilidade” AND "mobile" OR "dispositivos móveis"
***String* em inglês**	"blind-* games" OR "audio based games" OR "audio only games" OR "video-less games" OR "mobile audio games" "mobile devices" | "blind-* games" OR "audio based games" OR "audio only games" OR "video-less games" OR "mobile audio games" "mobile devices" AND “pure data”
Idiomas:	Inglês e Português;
Período:	Últimos 6 anos (2016 - 2021);
Repositórios:	IEEEXplore, ACM Digital Library, Springer, Periódicos Capes, Researchgate, Scopus, Scholar Google;

Tabela 1 - Parâmetros de busca.

4.1.3. Critérios de inclusão e exclusão

Após a elaboração do protocolo de pesquisa é necessário também definir critérios específicos de inclusão e exclusão de artigos na revisão. Vale ressaltar que eventualmente algum estudo que não possui relevância para a revisão pode ou vai entrar através do protocolo de pesquisa, cabe aos critérios de inclusão e exclusão definir e restringir a relevância dos dados obtidos. Os critérios de inclusão (CI) para este estudo foram:

CI.1 - Artigos que possuem acesso livre e gratuito;

CI.2 - Artigos que abordam o desenvolvimento, análise ou elaboram discussões sobre *audiogames* para dispositivos móveis;

CI.3 - Estudos primários;

CI.4 - Estudos de outras áreas do conhecimento mas que utilizem as ferramentas ou abordagens da temática;

CI.5 - Abordem especificamente *audiogames* para dispositivos móveis;

CI.6 - Artigos que façam uso do *Pure Data* como ferramenta de desenvolvimento de *audiogames* para dispositivos móveis.

Os critérios de exclusão tem o mesmo impacto que os de inclusão, os quais servem para filtrar e afunilar ainda mais o escopo de pesquisa, neste estudo foram utilizados os seguintes critérios de exclusão (CE):

CE.1 - Estudos secundários (guias, revisões);

CE.2 - Artigos pagos ou com acesso bloqueado;

CE.3 - Estudos que sejam versões anteriores de um mesmo mais completo sobre a pesquisa;

CE.4 - Artigos sem resumo;

CE.5 - Estudo que seja apenas uma descrição de ferramentas ou patente;

4.1.4 Resultados

Com a aplicação do protocolo de pesquisa anteriormente demonstrado, fizemos um levantamento inicial de 169 resultados, utilizando as *strings* de busca nos repositórios e buscadores previamente definidos. Em seguida aplicamos os filtros com as palavras chave no título e resumo, com o objetivo de filtrar os artigos que realmente contém os termos definidos pela pesquisa, após esse passo, chegamos ao resultado de 20 artigos que inicialmente podem abordar os temas que circundam a temática da pesquisa. Na etapa seguinte realizamos a filtragem desses resultados utilizando os critérios de inclusão e exclusão, desta vez com análise de título, resumo e introdução dos artigos, resultando nos 10 artigos definitivos presentes nesta revisão. Este processo pode ser melhor visualizado no fluxograma (figura 10) a seguir:

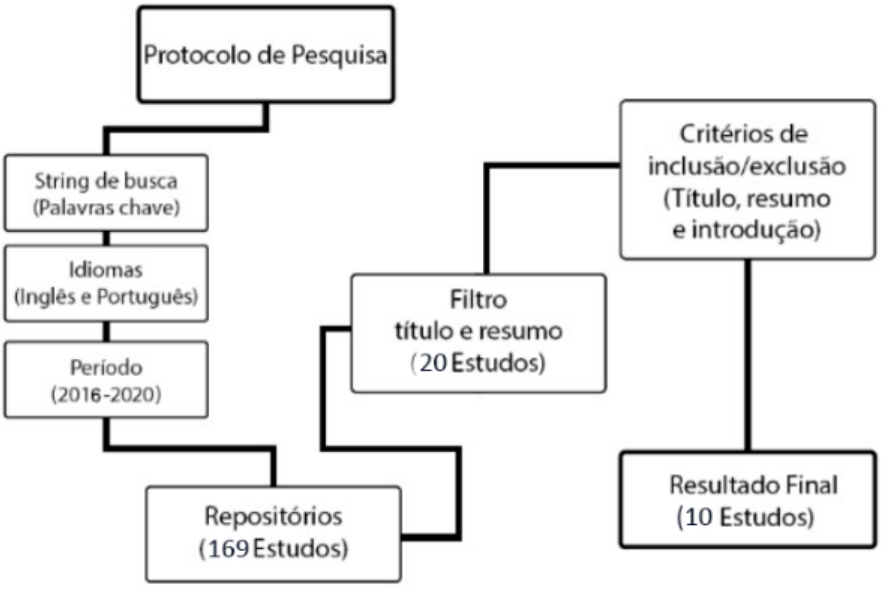


Figura 10 - Aplicação do protocolo de pesquisa. Fonte: O autor.

Os resultados obtidos com aplicação do protocolo de pesquisa foram analisados por completo, visto que atendem aos critérios definidos neste estudo. Os quais estão apresentados na tabela 2 a seguir:

RESULTADOS (Artigos)
Análise do Audio Game Hub: Uma Experiência Sensorial para Pessoas Com Deficiência Visual. (2019)
Bonk: accessible programming for accessible audio games. (2018)
Can everyone use my app? An Empirical Study on Accessibility in Android Apps. (2019)
Identificando características da acessibilidade Web Mobile: Desenvolvimento de um jogo acessível para dispositivos móveis. (2016)
Imersão e medo em jogos de terror: análise das estruturas de áudio e efeitos sonoros no jogo Blindside. (2016)
Ortomonstro: um audiogame móvel customizável para práticas ortográficas de português por meio do Braille. (2016)
Soundmaze: Desenvolvimento de um audiogame para deficientes visuais. (2020)
Pure Data for Pure Audio Games. (2020)

The effects of musical experience and hearing loss on solving an audio-based gaming task. (2017)
The Promotion of Empathy for the Experience of Users with Visual Impairment in the Game Design Education. (2019)

Tabela 2 - Resultados obtidos.

Ao analisarmos os dados obtidos pela revisão sistemática podemos elaborar um panorama geral do que se tem discutido sobre a temática de *audiogames* em dispositivos móveis. Podemos então destacar algumas características, na área de *audiogames* no geral tem levantado diversas discussões sobre a inclusão dos usuários com deficiência visual no "grande público" do mercado de games. Apesar dos avanços tecnológicos e do grande faturamento deste mercado, pouco tem sido produzido com o viés de acessibilidade pelas gigantes do mercado, muitas vezes os *audiogames* têm sido produzidos por desenvolvedores independentes e até mesmo pelos próprios IDV.

Na tabela 3 a seguir, podemos visualizar uma série de relações entre os resultados obtidos pela revisão sistemática, como por exemplo, se houve e quais as *engines* utilizadas no desenvolvimento do *audiogame*, se houve uma aplicação ou produção de *audiogame*, a área de estudo e também a abordagem utilizada no estudo.

Relações entre os resultados da RSL					
Artigo	Ano	*Engine*	Aplicação	Área	Abordagem
Análise do Áudio Game Hub: Uma Experiência Sensorial para Pessoas Com Deficiência Visual.	2019	Unity	Audio Game Hub	Comunicação	Análise
Bonk: accessible programming for accessible audio games.	2018	x	*Framework*	Computação/Programação	Desenvolvimento
Can everyone use my app? An Empirical Study on Accessibility in Android Apps.	2019	x	x	Computação/Programação	Estudo empírico (Análise)
Identificando características da acessibilidade Web Mobile:	2016	HTML/CSS	Batalha Naval	Computação/Programação	Desenvolvimento de *audiogame mobile*

Desenvolvimento de um jogo acessível para dispositivos móveis.					
Imersão e medo em jogos de terror: análise das estruturas de áudio e efeitos sonoros no jogo Blindside.	2016	x	*Blindside*	Ciências Sociais	Análise de *audiogame*
Ortomonstro: um audiogame móvel customizável para práticas ortográficas de português por meio do Braille.	2016	x	*Ortomonstro*	Computação/Programação	Desenvolvimento de *audiogame mobile*
Soundmaze: Desenvolvimento de um audiogame para deficientes visuais.	2020	Java	*Soundmaze*	Computação/Programação	Desenvolvimento de *audiogame mobile*
Pure Data for Pure Audio Games.	2020	*Pure Data*	*Tonaudio*	Música	Desenvolvimento de *audiogame mobile*
The effects of musical experience and hearing loss on solving an audio-based gaming task.	2017	*Pure Data*	*Music Puzzle*	Computação/Programação	Desenvolvimento de *audiogame mobile* utilizando *Pure Data*
The Promotion of Empathy for the Experience of Users with Visual Impairment in the Game Design Education.	2019	x	*IGenuys/ Memories Sound/ Turret Attack/ Balance/ In The Darkness/ Maneuver Simulator/ Explorer*	Computação/Programação	Estudo com alunos de *game design* no desenvolvimento de *audiogames* (*mobile*)

Tabela 3 - Relações entre os resultados da RSL. Fonte: o autor

A partir disto, podemos destacar que as plataformas *mobile* têm sido observadas como muito promissoras nesta área. Podemos destacar fatores como: a alta popularidade dos jogos

em dispositivos móveis, o acesso em massa da grande população e a possibilidade de acessar diversos conteúdos em qualquer lugar e hora. Do ponto de vista do desenvolvedor, podemos destacar também as diversas formas de interação do usuário com o dispositivo móvel, ou os *inputs* que o usuário pode utilizar, como gestos na tela, gestos com o dispositivo, bússola, microfone entre outras.

Outra abordagem que verificamos foi a do ponto de vista do desenvolvedor, alguns dos estudos discutem e destacam a empatia que deve ser estimulada no desenvolvedor, a fim de que possam fomentar a produção de novos *audiogames* para os IDV. Entre as aplicações e jogos desenvolvidos, vemos que os *audiogames* são bastante discutidos em áreas educacionais e em jogos sérios, como uma forma de aprendizagem e inclusão de IDV.

No que concerne aos jogos como entretenimento e lazer, vemos uma discrepância muito grande no mercado, são poucos jogos *AAA* que disponibilizam alguma forma de acessibilidade. Entre as formas de acessibilidade observadas, muitas delas são limitadas apenas a mudança de resolução, ajuste de volume de som, ativação de legendas e outras poucas. Dito isto, é pertinente e relevante a discussão sobre as produções de jogos voltadas para IDV com viés de entretenimento e lazer.

Observamos que o uso do *Pure Data* no desenvolvimento de *audiogames* é um campo ainda pouco explorado no meio acadêmico e está carente de produções. Observamos que, nesta RSL apenas dois dos resultados obtidos utilizou o *Pure Data* como ferramenta de áudio, o que destaca o objetivo desta pesquisa, tornar o uso dessa linguagem de programação *open source* e gratuita viável para a fomentação de novas produções de *audiogames* para dispositivos móveis.

5. *AUDIOGAME BINAURAL FLIGHT*

Para testar as ferramentas estudadas nesta pesquisa, desenvolvemos o protótipo de *audiogame* para dispositivos móveis chamado *Binaural Flight* (figura 11), um *endless scrolling game* [16] vertical inspirado no jogo *River Raid* (*Atari 2600)*. No jogo, o jogador controla uma aeronave que sobrevoa o oceano e deve desviar dos dirigíveis (obstáculos sonoros) pelo máximo de tempo possível. Para isso, o jogador deve movimentar a aeronave na horizontal (movimentos para a esquerda ou direita) para evitar os obstáculos que surgem aleatoriamente. Para isso, o jogador deve se basear na posição dos elementos sonoros no espaço e assim evitar que sua aeronave colida com os obstáculos.

Figura 11 - Tela inicial *Binaural Flight*. Fonte: O autor.

Para o propósito deste trabalho, criamos um escopo limitado de funcionalidades na jogabilidade no *audiogame*, visto que nosso objetivo é testar as ferramentas e funcionalidades propostas no decorrer desta pesquisa. Dito isto, o jogo não possui alguns aspectos de *game design* como: pontuação, progressão de dificuldade e outros modos de jogo, todavia, temos interesse na implementação desses elementos em possíveis atualizações futuras.

[16] Jogo infinito de rolagem vertical.

A jogabilidade do *audiogame Binaural Flight* é inteiramente baseada em elementos sonoros, e através do som binaural o jogador deve se situar no espaço sonoro 3D, e assim movimentar a aeronave no espaço e desviar dos obstáculos. Para fins de auxílio no desenvolvimento do *audiogame*, incluímos uma interface visual simples (com os *sprites* do jogo desenvolvidos em *flat design*). Essa interface pode ser desabilitada a qualquer momento durante o jogo através de um botão no canto superior direito da tela, assim proporcionando a experiência completa baseada apenas no som de um *audiogame*.

No game, a aeronave se movimenta entre três posições pré definidas na tela: esquerda, centro e direita. Da mesma forma os obstáculos também estão dispostos nessas mesmas posições, porém surgem no topo da tela e descem em direção a sua aeronave, sendo assim o jogador deve evitar estar na mesma posição que o obstáculo.

Por se tratar de um jogo para dispositivos móveis, os controles de movimentação da aeronave são realizados através do movimento *"swipe"*, que consiste em pressionar a tela e arrastar em um único movimento, da esquerda para a direita ou da direita para a esquerda, assim movimentando a aeronave para a direita e esquerda respectivamente.

A partir disso, podemos testar e verificar as soluções propostas por este estudo, por meio de sua aplicação no *audiogame Binaural Flight*. Nele utilizamos os *patches Pd* para processar e gerar elementos sonoros em tempo real no *audiogame*. Podemos também verificar a utilização do *libpd* na integração do *Pure Data* na plataforma *Unity*. E por último, investigar a comunicação entre os elementos do jogo e os *patches Pd* e vice-versa e as formas de utilizar a espacialização sonora com essas ferramentas.

5.1. Desenvolvimento do *audiogame* na plataforma *Unity*

Para o desenvolvimento do nosso *audiogame,* decidimos utilizar a plataforma de desenvolvimento de jogos *Unity*, também conhecida como *Unity3D* ou *UnityEngine*, desenvolvido pela *Unity Technologies.* A *Unity* oferece diversas funcionalidades para o desenvolvimento de jogos 3D e também 2D, decidimos utilizá-la por causa de sua grande popularidade e quantidade de material didático que pode ser encontrado na internet gratuitamente. Entre as funcionalidades do *Unity*, podemos destacar também a portabilidade dos jogos para diversas plataformas, dentro de um mesmo projeto, através disso, o

desenvolvedor pode de maneira simples criar "*ports"* do seu jogo para dispositivos móveis, *web browsers, desktops* e diversos *consoles.*

Para o propósito deste estudo, utilizamos o *Unity* para a criação de um *audiogame* 2D para dispositivos móveis. Para facilitar a parte de desenvolvimento e programação, optamos por criar uma interface visual simples. O *Binaural Flight* é inspirado no *River Raid*, por isso criamos uma interface gráfica (figura 12) onde a aeronave sobrevoa o oceano e deve evitar colidir com os dirigíveis (obstáculos). Os *sprites* do game foram desenhados no estilo *flat design* e exportados em formato *png*.

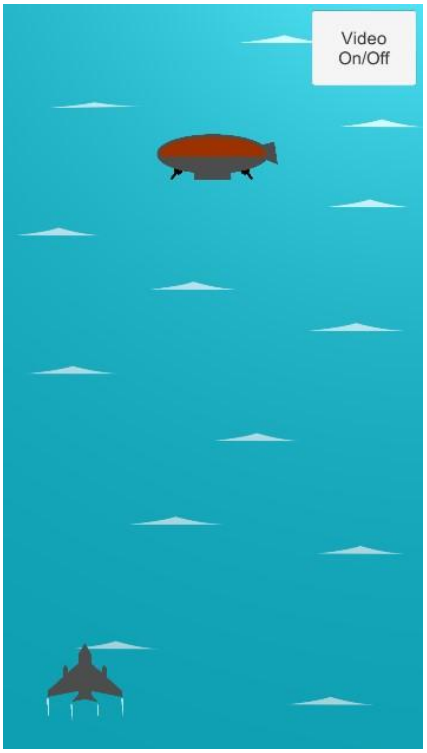


Figura 12 - Interface visual do *Binaural Flight*. Fonte: O autor.

Após criada a identidade visual do jogo, iniciamos o projeto do tipo *mobile 2D* na plataforma *Unity,* com resolução 1080x1920 (retrato) visando uma maior compatibilidade com a maioria dos dispositivos móveis. Dentro do projeto do game, criamos três cenas base que vão compor a *audiogame*, as quais são: *Title Screen, Gameplay e Gameover.*

A cena *Title Screen* contém a tela inicial (figura 11, apresentada anteriormente) do jogo, nela desenvolvemos um script chamado "inicio" (Anexo I) que tem como objetivo captar um *input touch* (entrada de toque) do jogador, quando acionado o *script* chama a cena *"Gameplay"*.

Na cena *"Gameplay"* (figura 13), estão os principais elementos do jogo: o cenário infinito, a aeronave controlada pelo jogador, os geradores de obstáculos (dirigíveis) e os *scripts* de jogabilidade. Primeiramente, criamos um *script* ("*scroll*" que pode ser encontrado em Anexo I) para movimentar o cenário automaticamente e infinitamente na vertical. O *sprite* do *background* se movimenta em uma velocidade de 0,5 frames por segundo na vertical, se repetindo infinitamente, resultando na sensação de velocidade e movimento.

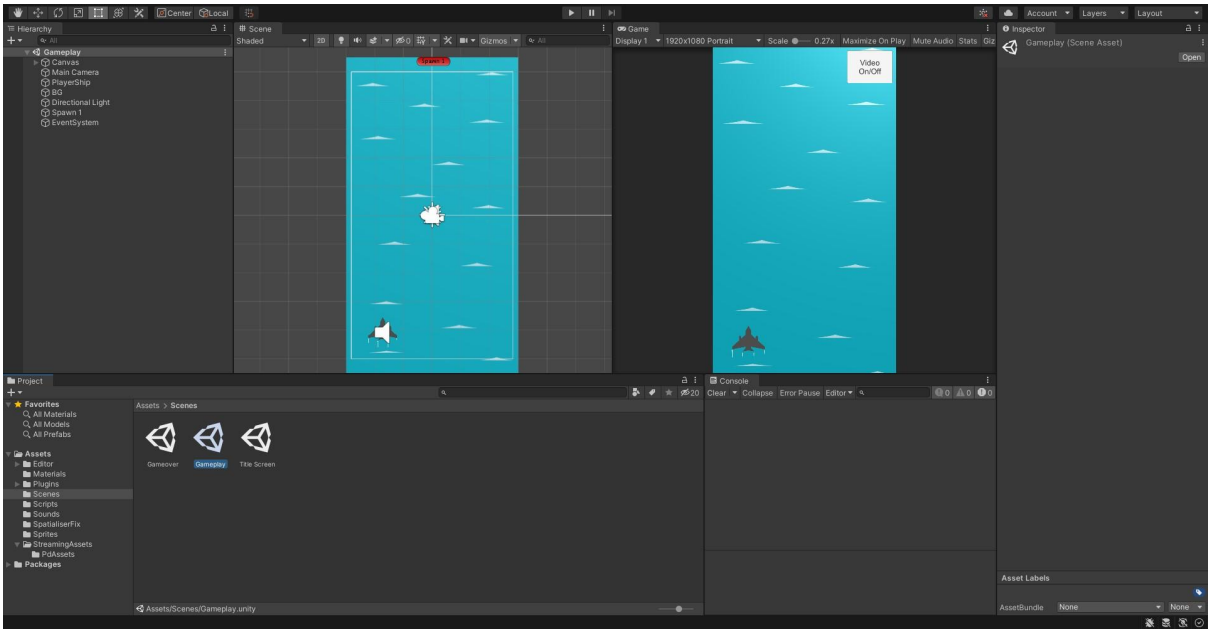


Figura 13 - Cena *"Gameplay"* do *Binaural Flight*. Fonte: O autor.

A aeronave controlada pelo jogador contém: o *sprite* da aeronave, *Box collider 2D* (delimita a área de colisão da aeronave), *Rigid body 2D* (que define a rigidez do elemento) e dois *scripts*, sendo uma instância do *libpd "LibPdInstance"* (a qual será detalhada na seção 5.3) e o *script "SwipeMove"*. O *script "SwipeMove"* (Anexo I) define as posições fixas de movimentação da aeronave no eixo x (esquerda, centro e direita), e também detecta as entradas por toque *"swipe"* realizadas pelo jogador e transforma em movimentação da aeronave. Nesse *script* também adicionamos uma linha de comando que faz com que caso a aeronave entre em colisão com o dirigível ele chama a cena *"Gameover"*. Assim, a parte de movimentação do jogador é completamente controlada por *inputs* realizados no *touch screen* do dispositivo móvel.

Os obstáculos (dirigíveis) contém os elementos: o *sprite* do dirigível, *Box collider 2D,* o *script "LibPdInstance"* e o *script "moveObject"*. O script "*moveObject*" tem a função de

movimentar o dirigível no eixo y, do topo da tela em direção a aeronave (movimento de cima para baixo). Para facilitar o desenvolvimento de vários obstáculos, utilizamos uma técnica de desenvolvimento de games no Unity chamada de *prefab*, que consiste em pré-fabricar elementos. Dito isto, criamos um *prefab* do dirigível (figura 14), assim podemos replicar e duplicar com facilidade o dirigível (contendo todos os elementos anteriormente citados).

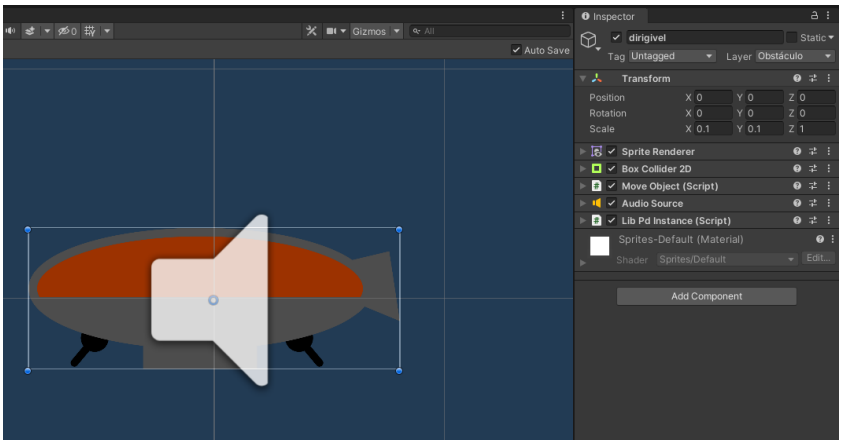


Figura 14 - *Prefab* do dirigível contendo todos os elementos internos. Fonte: O autor.

Com isso, criamos também um "*spawn*" (gerador) de obstáculos automático. Para isso, criamos um objeto nulo e adicionamos um *script* chamado *"spawnController"* (Anexo I). Nele, pré determinamos os três locais (no eixo x, coincidentes com a movimentação da aeronave) de surgimento dos obstáculos (*prefabs* do dirigível), para que assim possam colidir com o jogador. Após isso, adicionamos uma variável de frequência de surgimento e uma variável de randomização entre as três posições determinadas.

Na cena *"Gameplay"* também adicionamos um elemento *UI* (interface do usuário), um botão "Video On/Off" no canto superior direito (figura 15) contendo o *script "telapreta",* o qual desabilita a interface visual, assim permitindo que o jogador possa jogar se baseando inteiramente no som.

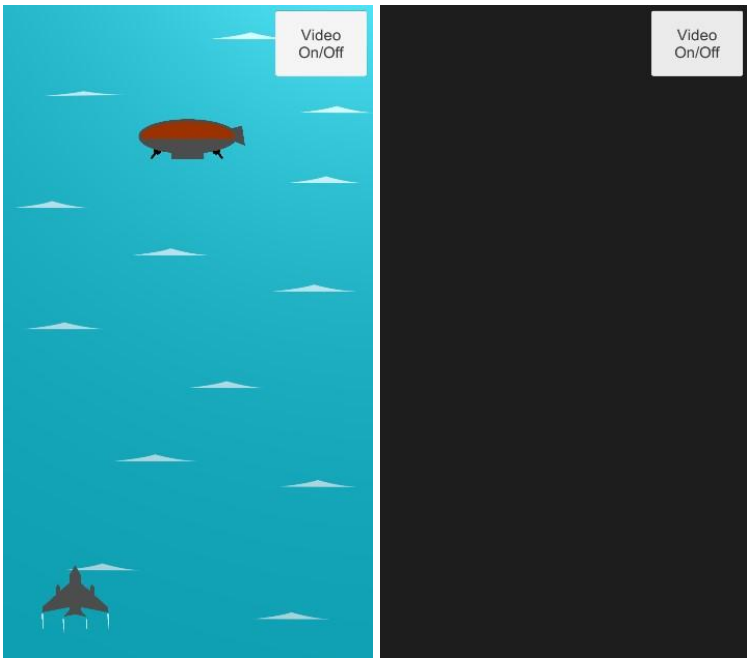


Figura 15 - Botão "Video On/Off". Fonte: O autor.

A cena *"Gameover"* contém a tela de fim de jogo (figura 16) e um *script* simples "fim" que tem como objetivo receber um *input* de toque, e em seguida chama a cena *"Gameplay"*, assim reiniciando o jogo.

Figura 16 - Tela de fim de jogo. Fonte: O autor.

Com isso, temos um ciclo completo de jogabilidade do *Binaural Flight*. Iniciamos o jogo na tela inicial, ao receber o *input* de toque (interação de toque na tela *touch*) chama a cena de jogabilidade. Na cena de jogabilidade, ao receber *inputs swipe* (interação de tocar e arrastar na tela *touch*) movimenta a aeronave. Simultaneamente, o gerador de obstáculos envia os dirigíveis automaticamente e aleatoriamente para a cena. Caso a aeronave entre em colisão com um dos obstáculos ele é direcionado para a tela de fim de jogo, a qual recebe *inputs* de toque para reiniciar o jogo. O jogador deve se basear no posicionamento dos elementos sonoros dos dirigíveis (som binaural) e assim movimentar sua aeronave para esquerda ou direita, desviando da colisão.

5.1.1. Design de áudio do Binaural Flight

O design de áudio do jogo foi pensado para gerar contraste entre os elementos sonoros, visto que a jogabilidade do *audiogame* é focada no áudio e na sensação espacial sonora. Para isso, utilizamos a *DAW* (*Digital Audio Workstation*, software de manipulação de áudio) *Reaper*, a qual oferece uma gama de ferramentas de edição de áudio. O *Reaper* é um *software* pago, porém oferece um período de teste gratuito por tempo indeterminado.

Com a proposta de evitar o uso excessivo de sons, e assim prejudicar a experiência do jogador, separamos os sons em categorias: interface, background, aeronave e obstáculos. A partir dessas categorias, utilizamos o *Reaper* para a criação de uma parte dos *assets* de som que compõem o *audiogame* (figura 17), a outra parte dos elementos sonoros foram desenvolvidos dentro do próprio *Pd* (como veremos na seção 5.2).

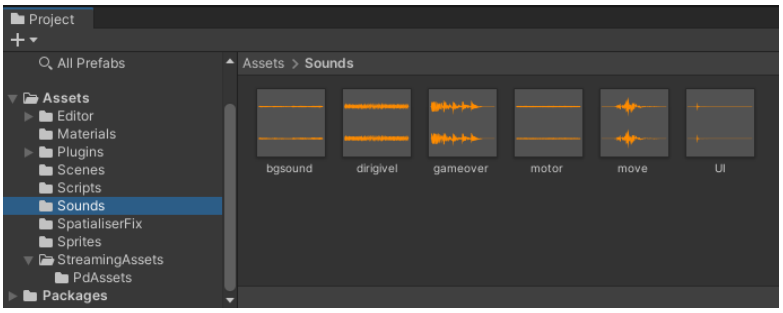


Figura 17 - *Assets* de som do jogo. Fonte: O autor.

Os sons de interface são os elementos que emitem o feedback de interação do jogador com os menus (botões) do jogo, por exemplo, ao habilitar ou desabilitar a interface visual do jogo o som de interface tocado informa ao jogador que a ação foi realizada. O *background* do *Binaural Flight* consiste no som do oceano, ondas e vento, tem como objetivo gerar imersão e construir uma identidade sonora para o jogo. O sons da aeronave indica ao jogador sua posição e movimentação no espaço sonoro, nele estão os sons do motor e o som de movimentação esquerda e direita, servem para dar feedback da jogabilidade ao jogador. Os obstáculos são os dirigíveis, que emitem o som binaural, para dar a sensação de espacialização.

5.2. Utilizando *Patches Pure Data*

Como descrevemos anteriormente, decidimos utilizar o *Pure Data* pela sua grande versatilidade no processamento de dados, principalmente na manipulação de áudio digital. Um fator importante e que deve ser destacado, é a interface visual, que facilita e torna mais intuitivo a programação propriamente dita, além do fato que o *Pd* é uma linguagem de programação visual *open source* e gratuita.

O *Pure Data* possui grande importância no desenvolvimento desta pesquisa, e é através dos *patches Pd* que vamos manipular e controlar vários elementos sonoros dentro do jogo em tempo real. Para isto, desenvolvemos *patches* específicos (anexo I) para atender as necessidades do jogo e também utilizamos *patches* desenvolvidos pela comunidade (distribuídos gratuitamente), com algumas adaptações para atender às nossas necessidades.

Durante o desenvolvimento dos *patches Pd* desta pesquisa, observamos duas maneiras de desenvolver e utilizar os *patches Pd* no contexto de desenvolvimento de games na plataforma *Unity.* A primeira consiste em desenvolver o *patch Pd* como um gerador de elementos sonoros em tempo real, vale destacar que o *Pd* já vem sendo utilizado como ferramenta de criação de música procedural e adaptativa em diversos contextos, incluindo os jogos. No contexto dos *audiogames* observamos que também podemos utilizar o *Pd* como um gerador de elementos sonoros. No *patch Pd* apresentado na figura 18, podemos observar o uso do *Pure Data* para a criação do som do dirigível, nele criamos um objeto *"osc~"*, que é um oscilador de onda senoidal, o qual gera uma onda na frequência 300 Hz.

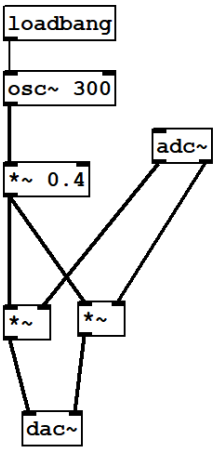


Figura 18 - *Patch Pd* utilizado para geração sonora do Dirigível. Fonte: O autor.

A segunda maneira que observamos para utilizar os *patches Pd* no contexto de *audiogames* é criar os *patches* para controlar e manipular os sons dentro dele mesmo. Na figura 19, criamos um *patch* simples que faz a leitura do arquivo de áudio.

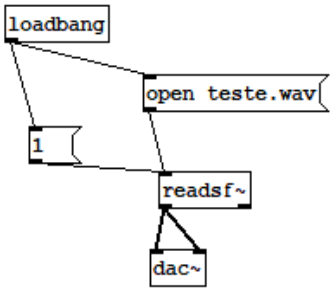


Figura 19 - *Patch Pd* para leitura de arquivo de áudio. Fonte: O autor.

Podemos também criar algo mais complexo, como por exemplo, um *patch* onde pré carregamos um arquivo de áudio (ou vários) em um *"array"* (que serve como um contêiner) e no próprio *patch* (ou em um *subpatch*) criar elementos de manipulação do som em tempo real. Na figura 20, por exemplo, criamos um *patch Pd* que faz a leitura e armazenamento de um arquivo de áudio (e também visualização da *waveform*), em seguida direciona o áudio para um elemento *"phasor~"* que nesse caso serve como um modulador de som em tempo real.

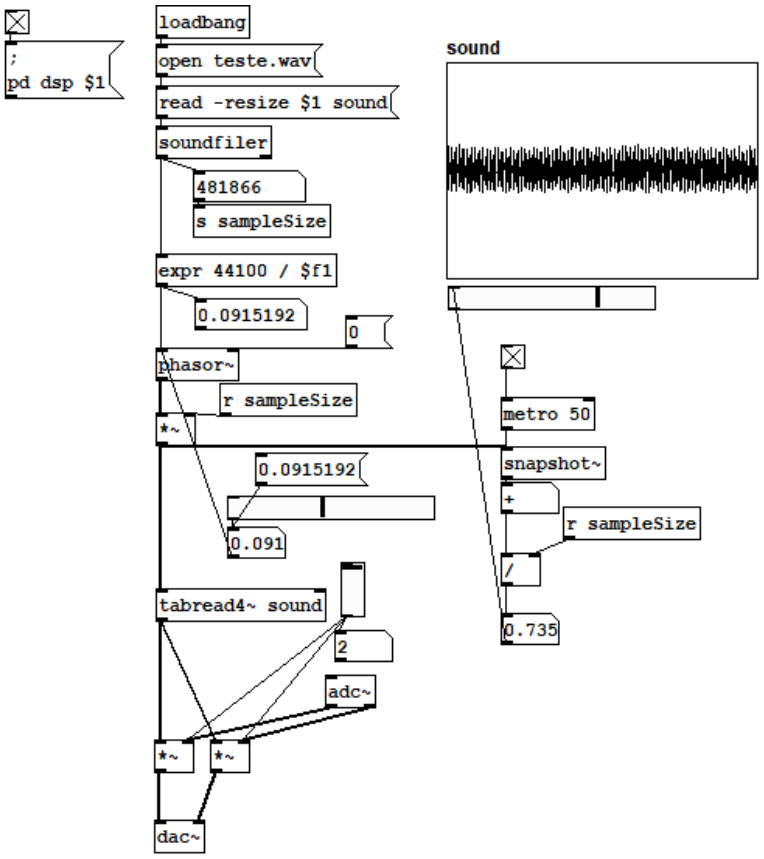


Figura 20 - *Patch Pd* para leitura, armazenamento e modulação de arquivo de áudio. Fonte: O autor.

O *Pure Data* possibilita diversas formas de manipulação e geração de sons através dos *patches Pd*, no contexto dos *audiogames*, temos diversas possibilidades e aplicações, entre elas, as demonstradas no decorrer deste estudo. Devemos ressaltar também a versatilidade e simplicidade do *Pd*, visto que, apesar das dificuldades enfrentadas no desenvolvimento do *audiogame*, se mostrou versátil e cumpriu com a proposta de sua utilização nesta pesquisa.

5.3. Implementação do *Libpd*

A implementação do *pure data* no projeto do jogo *Binaural Flight*, é realizada através do *LibPdIntegration* (desenvolvido na *Abertay University*). O *LibPdIntegration* é um repositório que permite incorporar *patches Pure Data* em um projeto na plataforma *Unity,* ele fornece os códigos e plugins necessários para implementação em diversas plataformas através de um único *script C#,* o "LibPdInstance.cs".

Para auxiliar nesse processo, utilizamos como base o projeto[17] desenvolvido por Yann Seznec, através de vídeo aulas disponibilizadas gratuitamente na internet. Yann Seznec demonstra em seu canal do *YouTube* o uso do *Pure Data* no desenvolvimento de jogos 3D na plataforma *Unity*, com isso, podemos observar o uso do *LibPdIntegration* em um contexto aproximado do qual estamos trabalhando nesta pesquisa.

Para implementar o *Pure Data* no projeto do jogo *Binaural Flight*, fizemos o download do *LibPdIntegration* (atualmente na versão 2.1.2) no repositório[18] disponibilizado no *GitHub*. O arquivo possui diversas pastas e arquivos, as quais devem ser inseridas na pasta raíz do projeto do jogo. Com isso, os arquivos necessários para realizarmos a integração do *libpd* e *Unity* estão prontos (este processo é devidamente explicado na seção 6 deste estudo).

Para associar um *patch Pd* a um *game object* na plataforma *Unity* é necessário inserir uma instância do *libpd* ao próprio *game object,* com isso ele também insere um objeto "*audio source*", para que o *Unity* possa processar a fonte de áudio do objeto, assim é possível inserir *patches Pd* dentro do jogo.

5.4. Comunicação entre *Libpd* e a plataforma *Unity*

Na documentação[19] (*wiki*) do *LibPdIntegration* apresentada por Niall Moody, ele descreve diversas formas de comunicação entre o *Unity* e o *libpd* e vice-versa. A comunicação do *Unity* para o *libpd* (ou *patch Pd*) é realizada de duas maneiras principais, a primeira é utilizando "*receive objects"* do *Pure Data* e a segunda é usando objetos *MIDI*. Com os "*receive objects"*, é possível enviar uma variedade de sinais diferentes para qualquer objeto de recepção nomeado no seu *patch Pd*, a comunicação pode ser realizada com as seguintes funções do *Pd* (tabela 4):

[17] <https://youtu.be/a4FBL6sARNI> Acessado em: 27/07/2021.
[18] <https://github.com/LibPdIntegration/LibPdIntegration#quickstart> Acessado em: 27/07/2021.
[19] <https://github.com/LibPdIntegration/LibPdIntegration/wiki/unity2libpd> Acessado em: 26/07/2021.

Funções de comunicação com *"receive objects*[20]*"*
void **SendBang** (string receiver)
void **SendFloat** (string receiver, float value)
void **SendSymbol** (string receiver, string symbol)
void **SendList** (string receiver, params object[] args)
void **SendMessage** (string receiver, string symbol, params object[] args)

Tabela 4 - Funções de comunicação com *"receive objects"*.

As funções apresentadas abaixo utilizam o protocolo MIDI para enviar sinais para diversos tipos de objetos, confira na tabela 5 a seguir:

Funções de comunicação com *"MIDI objects*[21]*"*
void **SendMidiNoteOn** (int channel, int pitch, int velocity)
void **SendMidiCc** (int channel, int controller, int value)
void **SendMidiProgramChange** (int channel, int value)
void **SendMidiPitchBend** (int channel, int value)
void **SendMidiAftertouch** (int channel, int value)
void **SendMidiPolyAftertouch** (int channel, int pitch, int value)
void **SendMidiByte** (int port, int value)
void **SendMidiSysex** (int port, int value)
void **SendMidiSysRealtime** (int port, int value)

Tabela 5 - Funções de comunicação com *"MIDI objects"*.

A comunicação inversa, isto é, do *libpd* para o *Unity*, acontece através de "*send objects*" ou qualquer um dos objetos de saída *MIDI* do *Pure Data*. Contudo, para que essa comunicação aconteça é necessário configurar um retorno para o seu próprio código usando um dos campos do *LibPdInstance, libpd -> Unity Events,* ele será chamado sempre que seu

[20] <https://github.com/LibPdIntegration/LibPdIntegration/wiki/unity2libpd> Acessado em: 26/07/2021.
[21] <https://github.com/LibPdIntegration/LibPdIntegration/wiki/unity2libpd> Acessado em: 26/07/2021.

patch mandar aquele sinal específico e também inicializar o *script Bind,* onde devemos vincular ao *send object*. As seguintes funções podem ser utilizadas (tabela 6):

Funções de comunicação do *libpd* para o *Unity*[22]
void **Bind** (string symbol)
void **UnBind** (string symbol)

Tabela 6 - Funções de comunicação do *libpd* para o *Unity.*

No contexto do jogo *Binaural Flight*, é possível observar os elementos *"receive objects"* (no nosso *patch* é o *"Loadbang",* pode ser observado na figura 18, 19, e 20), necessários para a comunicação do *Unity* com o nosso *patch Pd.*

Para ler e manipular os elementos *"arrays"* do *Pure Data* dentro de um projeto *Unity,* podemos utilizar as seguintes funções (tabela 7) dentro do *libpd:*

Funções de comunicação com *arrays* do *Pd*[23]
int **ArraySize** (string name)
void **ReadArray** (float[] dest, string src, int offset, int count)
void **WriteArray** (string dest, int offset, float[] src, int count)

Tabela 7 - Funções de comunicação com *arrays* do *Pd.*

Mesmo com isso, é possível utilizar arrays nos *patches* do *Pure Data* (como mostramos na figura 20), inclusive é uma das formas que utilizamos para o desenvolvimento do *Binaural Flight.*

Um dos problemas encontrados para trabalhar a espacialização com o *libpd* é que por padrão, o *Unity* não aplica espacialização a *patches Pd* através de uma instância do *libpd*, ele aplica apenas a arquivos de som reproduzidos por meio do elemento *"audio source"* dentro do próprio *Unity.* Uma forma de contornar esse problema, é utilizando um arquivo de áudio especial (fornecido pelo repositório do *libpdIntegration*) e adicionar um objeto *"adc~"* em

[22] <https://github.com/LibPdIntegration/LibPdIntegration/wiki/libpd2unity> Acessado em: 26/07/2021.
[23] <https://github.com/LibPdIntegration/LibPdIntegration/wiki/arrays> Acessado em: 26/07/2021.

nosso *patch Pd,* assim aplicar a espacialização da fonte de áudio à saída do nosso *patch Pd* (este processo é descrito detalhadamente na seção 6.2.4). Outra maneira de resolver esse problema, é utilizando um plugin de espacialização externo, como o *Steam Audio,* por exemplo (cujo processo detalhado pode ser encontrado na seção 6.2.4).

5.5. *LibPdIntregation* no *Binaural Flight*

Em nosso jogo, o *LibPdIntegration* fez a conexão entre os *patches Pd* e os *Game Objects* na plataforma *Unity*, mais especificamente, utilizamos para a criação dos obstáculos no jogo (os dirigíveis) e também para a espacialização sonora destes elementos.

Para que a integração seja realizada (como vimos anteriormente), adicionamos o *script* "LibPdInstance" (figura 21) no *prefab* "dirigivel" no game, desta maneira, o som do dirigível é gerado pelo *patch Pd* "dirigivel 2" (Anexo I).

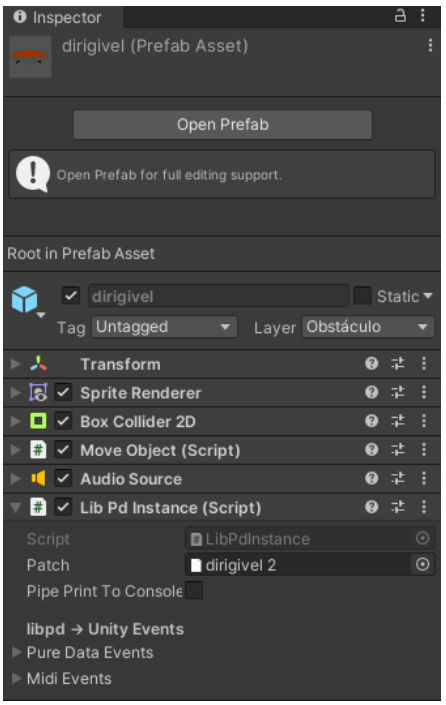


Figura 21 - *LibPdInstance* e *patch Pd* "dirigivel 2" carregados no prefab "dirigivel". Fonte: O autor.

Como descrevemos durante todo este estudo, o jogo *Binaural Flight* serve como um exemplo simples da aplicação do *Pure Data* no desenvolvimento de *audiogames*, e nessa

seção vimos como o *LibPdIntegration* foi utilizado na integração do *Pd* no projeto, e também no uso do *patch Pd* na criação de elementos sonoros em tempo real no jogo.

6. RESULTADOS

Com o desenvolvimento desta pesquisa, estudamos e também testamos o uso do *Pure Data* no desenvolvimento de *audiogames* para dispositivos móveis na plataforma *Unity*. A partir do desenvolvimento do *audiogame Binaural Flight*, apesar de um simples protótipo e com a finalidade de apenas testes, podemos levantar algumas questões e avaliações importantes sobre o seu desenvolvimento.

Inicialmente, devemos ressaltar mais uma vez a falta de experiência do autor deste estudo com a área de programação e desenvolvimento de jogos, o qual foi um fator limitante na pesquisa que realizamos. Todavia, ao mesmo tempo, vemos isso como um ponto positivo, visto que, uma pessoa com pouca experiência no desenvolvimento de games e também em programação consegue criar *audiogames* de maneira simples e descomplicada.

A plataforma *Unity* é uma das mais utilizadas no desenvolvimento de jogos, por isso, é relativamente fácil encontrar material didático disponibilizado gratuitamente na internet, tanto em português quanto em inglês. Devemos destacar também que apesar de popular, o *Unity* é uma ferramenta complexa, em alguns momentos pode ser difícil encontrar material didático sobre um tema específico, fator esse que limitou o desenvolvimento desta pesquisa. O *Unity* é utilizado por grandes empresas no desenvolvimento de grandes jogos 3D e 2D, oferece muitas funções e ferramentas, mas para o desenvolvimento de jogos simples podem existir outras opções mais acessíveis, entretanto ainda é a mais popular.

O *Pure Data* juntamente com o *libpd* na plataforma *Unity* abre diversas possibilidades na criação e manipulação de áudio, no contexto dos *audiogames*, serviram de maneira eficiente ao propósito que levantamos nesta pesquisa. Diferentemente de Passos (2021), que no seu estudo decidiu não utilizar o *LibPdIntegration* por falta de suporte a plataforma *android*, nesta pesquisa observamos que é possível a integração do *Pure Data* na plataforma *Unity* através desta ferramenta. A grande quantidade de *patches Pd* criados e disponibilizados pela comunidade é animadora, aumenta ainda mais as possibilidades para os desenvolvedores independentes na criação de *audiogames* para os IDV.

Como resultado desta pesquisa, levantamos um conjunto de soluções viáveis para auxiliar desenvolvedores no uso do *Pd* na plataforma *Unity*, desde a preparação do projeto *Unity*, implementação do *libpdIntegration* e uso dos *patches Pd* dentro do projeto de jogo.

6.1. *Pure Data* no desenvolvimento de *audiogames*

O uso do *Pure Data* no *Binaural Flight*, como mostramos na seção 5, foi eficiente dentro do escopo que delimitamos para esta pesquisa. Os *patches Pd* apesar de simples cumprem o seu propósito dentro do *audiogame* que desenvolvemos. Ao observarmos, quando utilizados na criação de elementos sonoros, o *Pure Data* oferece muitas possibilidades na manipulação de sinais de áudio, como por exemplo, os osciladores que utilizamos para gerar os sons dos dirigíveis. Podemos destacar também a manipulação de áudio em tempo real, servindo como um dinamizador de áudio.

O *Pure Data* já é utilizado há bastante tempo nos contextos de música procedural, trilha sonora adaptativa e diversos experimentos com instrumentos virtuais. Apesar de existir material didático e guias gratuitos na internet, são poucos disponíveis em língua portuguesa, e no contexto de *audiogames* especificamente, é ainda mais escasso. Podemos entender que isso pode ser um limitante no uso do *Pd* por desenvolvedores brasileiros, e também no uso em *audiogames*.

De maneira geral, o *Binaural Flight* foi uma maneira simples e eficiente de testar o uso do *Pure Data* juntamente com o *libpd* no desenvolvimento de *audiogames* na plataforma *Unity.* Avaliamos esse processo de maneira positiva, e a partir disso, podemos destacar algumas vantagens:

- Linguagem de fácil entendimento;
- Interface visual de programação;
- Roda em qualquer hardware e em diversas plataformas (windows, mac, linux, android, ios e etc);
- Open source gratuito;
- Comunidade de áudio ativa, tutoriais e patches Pd;
- Integração em outras plataformas através do Libpd;
- Manipulação de áudio em tempo real;

Da mesma maneira, podemos destacar também algumas desvantagens do *Pure Data* encontradas durante esta pesquisa:

- Pouco material didático em português;
- Apenas leitura de arquivos .wav (na versão vanilla);
- Sistema de espacialização de som (na plataforma Unity);

O *Pure Data* oferece diversas possibilidades de manipulação de áudio em tempo real, isso potencializa seu uso no contexto dos *audiogames*, visto que baseiam sua interface inteiramente no áudio. Podemos destacar também a possibilidade da criação de um *patch Pd* contendo outros *subpatches* com efeitos e funções que são oferecidas pelas *engines* de áudio, desta forma, é possível utilizar o *Pd* como uma *engine* de áudio.

Quando comparamos com as diretrizes de desenvolvimento de *audiogames* levantadas por Balan et al. (2015) na fundamentação teórica desta pesquisa (seção 2.1) podemos associar algumas possibilidades de utilização do *Pure Data*, como podemos observar na tabela 8 a seguir:

Especificações *audiogames*	*Pure Data*
Um alto nível de imersão e atratividade para motivar o usuário a avançar no jogo;	É possível criar diversas interações com elementos dentro do jogo com *patches Pd;*
Diversidade em pistas de áudio;	É possível utilizar diversas instâncias do *Pd* em um único projeto *Unity;*
Uma pequena quantidade ou uma completa falta de situações de "game over" que desencorajam o jogador a continuar jogando;	Esta funcionalidade não entra no escopo de utilização do *Pd*.
Uma maneira única e clara de resolver os problemas levantados pelo cenário;	Esta funcionalidade não entra no escopo de utilização do *Pd*.
Uma estratégia de aprendizado intuitiva, integrada ao desenvolvimento do jogo, para que o IDV entenda os requisitos sem o uso de instruções humanas ou de texto-fala;	Não está no propósito de utilização do *Pd*.
Interatividade, engajamento e diversão, para que os IDV se beneficiem de uma	É possível criar diversas interações com elementos dentro do jogo com *patches Pd;*

experiência agradável e divertida;	

Tabela 8 - Comparação de especificações de audiogames e *Pure Data.* Fonte: O autor.

Segundo as especificações levantadas por Balan et al. (2015), dependendo do propósito do desenvolvedor, o *Pure Data* consegue oferecer diversas soluções criativas e que suprem essas especificações. Já quanto as bases para desenvolvimento de *audiogames* acessíveis estabelecidas pela WACAG 2.0 (*Web Content Accessibility Guidelines*), que são: percepção, operabilidade, compreensão do conteúdo e robustez, podemos realizar o mesmo exercício de comparação com as funções oferecidas pelo *Pure Data* (tabela 9).

WACAG 2.0	*Pure Data*
Percepção: A informação deve ser apresentada de maneira acessível e compreensível;	Esta funcionalidade não entra no escopo de utilização do *Pd*.
Operabilidade: Os usuários devem poder interagir e se comunicar com a interface;	Através do Libpd a integração do *Pd* com elementos *Unity*;
Compreensão do conteúdo: A operação e manipulação da interface deve ser facilmente acessível;	Através do Libpd a integração do *Pd* com elementos *Unity*;
Robustez: O jogo deve permanecer acessível mesmo com o avanço da tecnologia;	*Pure Data* já possui muitos anos e continua sendo implementado em diversas plataformas novas, por sua simplicidade e portabilidade;

Tabela 9 - Comparação de especificações do WACAG 2.0 e *Pure Data.* Fonte: O autor.

Araújo et al. (2015) propõe um conjunto de recomendações para avaliação de acessibilidade de *audiogames* móveis, as quais também podemos traçar um paralelo com as possibilidades oferecidas pelo *Pure Data* (tabela 10).

Recomendações de acessibilidade	*Pure Data*
Textos alternativos: Devem oferecer alternativas textuais para todo o conteúdo não textual apresentado na interface móvel;	Esta funcionalidade não entra no escopo de utilização do *Pd*.

Adaptabilidade: Devem fornecer conteúdos e interfaces que possam ser modificados (alterando resolução ou utilizando apenas áudio) sem perder informação essencial;	É possível fornecer tais aspectos com o *Pure Data* na plataforma *Unity*;
Ambientação: O jogador deve se situar com facilidade no cenário a partir dos conteúdos, incluindo separar, através do som, o primeiro plano do plano de fundo na interface móveis;	O *Pure Data* pode ser utilizado como fonte sonora em elementos dentro de um projeto *Unity*;
Operabilidade: Os componentes da interface,interação e navegação do jogador devem ser operáveis, de fácil entendimento e alteração pelo jogador (opções de início rápido, para que não tenha que percorrer muitos níveis de informação);	Depende totalmente do propósito do desenvolvedor, mas os *patches Pd* podem contribuir nesse sentido;
Facilidade de configuração: Permitir que o jogador ajuste, simplifique e salve os controles e configurações do jogo (idioma, resolução, volumes, leitor de tela e etc);	Esta funcionalidade não entra no escopo de utilização do *Pd*.
Assistência e tutorial: Oferecer a documentação e modos tutoriais objetivos e de fácil acesso;	Esta funcionalidade não entra no escopo de utilização do *Pd*.

Tabela 10 - Comparação de recomendações de acessibilidade e *Pure Data.* Fonte: O autor.

6.2. Soluções para integração do *Pure Data* no *Unity*

A partir do desenvolvimento do *Binaural Flight*, propomos soluções viáveis de uso do *Pure Data* na plataforma *Unity* no contexto dos *audiogames* para dispositivos móveis. As soluções são resultado da experiência, solução de problemas e características que encontramos durante esta pesquisa, e tem como objetivo auxiliar desenvolvedores independentes na criação de jogos para os IDV.

As soluções são baseadas no material levantado para esta pesquisa, e inspiradas nos trabalhos disponibilizados por Niall Moody e Yann Seznec, seus trabalhos forneceram muito suporte para o desenvolvimento desta pesquisa. O resultado é uma mistura de nossa pequena experiência (amadora na programação) desenvolvendo o *audiogame Binaural Flight* e no uso do *Pure Data* para criação e manipulação de elementos sonoros em tempo real.

As soluções estão organizadas em uma sequência de passos, os quais o desenvolvedor pode realizá-los para usar e implementar o *Pure Data* no seu projeto *Unity*. Além disso, fornecemos dicas e indicações de uso para facilitar o entendimento do processo.

6.2.1. Localizando a pasta raíz do projeto *Unity*

Inicialmente devemos localizar a pasta raíz do seu projeto de jogo *Unity* no Disco Rígido do seu sistema, para que em seguida possamos iniciar o processo de integração adicionando os arquivos necessários, para isto, vamos seguir as indicações em sequência.

Primeiramente devemos inicializar o *software Unity HUB*, onde é possível verificar todos os projetos criados pelo desenvolvedor. Após localizar o seu projeto (no nosso caso o "Projeto teste", do tipo *"Mobile 2D"*), em seguida acessar o menu ao lado direito, clicando nos três pontos (como indicado na figura 22).

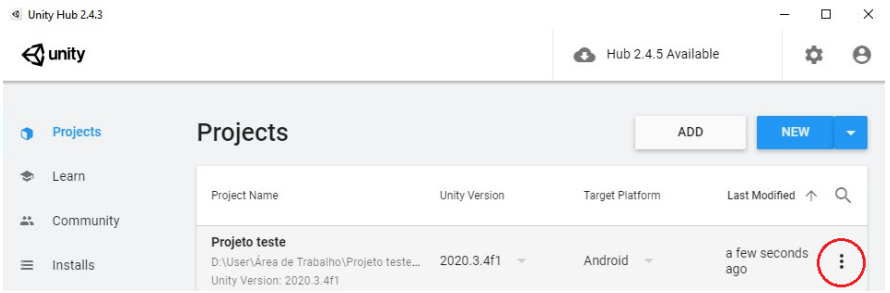


Figura 22 - Tela inicial *Unity HUB*. Fonte: O autor.

Em seguida, o próximo passo consiste em selecionar a opção "Show in Explorer" (indicado na figura 23), a qual irá abrir uma nova janela do explorador de arquivos do seu sistema operacional, indicando o "caminho" da pasta raíz do projeto, que pode ser observada na figura 24.

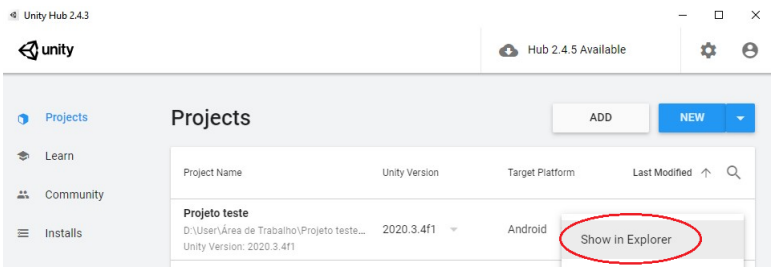


Figura 23 - Botão "Show in Explorer" no *Unity HUB*. Fonte: O autor.

Dentro da pasta do projeto “Projeto teste” (figura 24), ou também podemos chamá-la de “pasta raíz” do projeto, podemos observar os arquivos e pastas do seu projeto *Unity*, como *assets*, códigos, scripts, sons e outros arquivos do seu jogo (figura 24). Para prosseguirmos, devemos deixar essa pasta reservada, pois utilizaremos esse procedimento em diversas outras ocasiões.

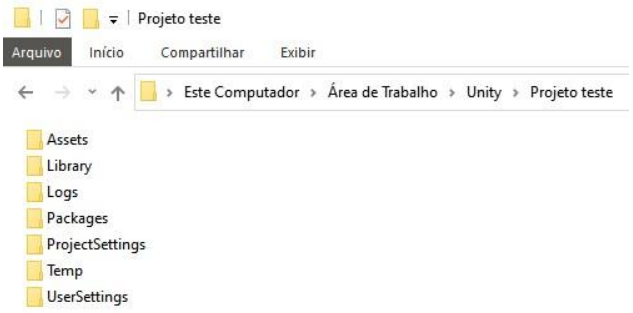


Figura 24 - Conteúdo da pasta “Projeto teste”. Fonte: O autor.

Para uma melhor visualização das etapas, apresentamos em formato de lista numerada os passos para aplicação das soluções, neste caso, para localizar a pasta raíz do projeto *Unity* devemos:

1. Inicializar o *Unity HUB*, localizar o seu projeto (no nosso caso o “Projeto teste”, do tipo *“Mobile 2D”*), em seguida clicar nos três pontos do lado direito*;*
2. Em seguida, devemos clicar em “Show in Explorer”;
3. Pasta do projeto “Projeto teste” é aberta, ou também podemos chamar de pasta raíz do projeto (no nosso caso a pasta “Projeto teste”);
4. Deixe essa pasta raíz reservada, voltaremos nela em breve;

6.2.2. Adicionando o *LibPdIntegration* no projeto *Unity*

Para adicionarmos e utilizarmos o *Pure Data* em nosso jogo, primeiramente devemos fazer o *download* dos arquivos necessários para a integração do *Pd* na plataforma *Unity*, que em nossa pesquisa é o *LibPdIntegration*, e em seguida implementá-los em nosso projeto, e para isto, vamos seguir os passos a seguir:

Primeiramente devemos acessar o site[24] repositório do *LibPdIntegration* (figura 25), onde é possível realizar o *download* dos arquivos do *LibPdIntegration* gratuitamente. Para isso, é necessário clicar no botão verde *"Code"* e em seguida *"Download ZIP"*, como demonstramos na figura 25 abaixo.

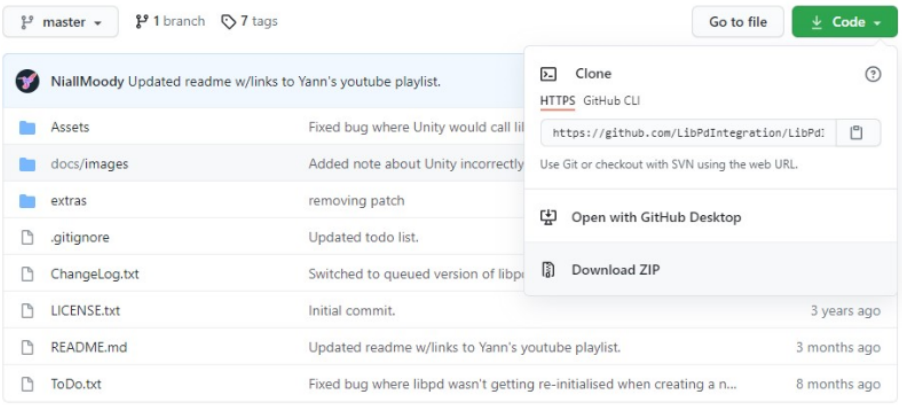


Figura 25 - Site repositório do *LibPdIntegration.* Fonte: O autor.

Após realizar o *download,* localize o arquivo *"LibPdIntegration-master.zip"* baixado em seu computador, e extraia o arquivo "*LibPdIntegration-master*" (figura 26) com o programa de sua preferência (como o *winrar*, *7-zip* ou outros).

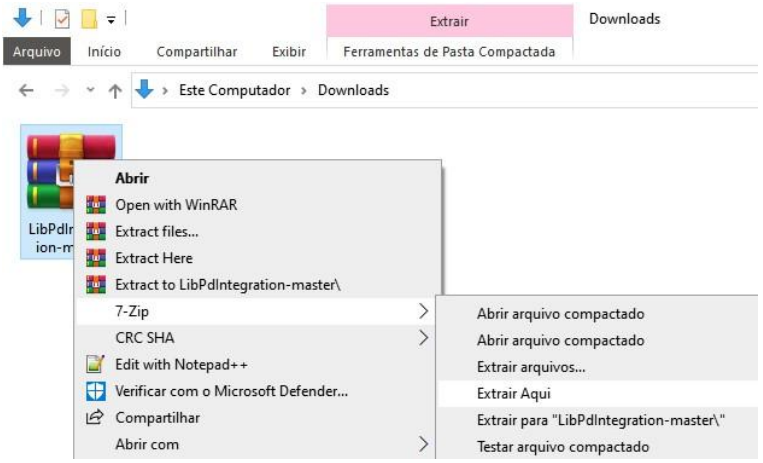


Figura 26 - Extraindo arquivo *LibPdIntegration.* Fonte: O autor.

Os arquivos "extraídos" do *"LibPdIntegration-master"* são os arquivos necessários para realizarmos integração do *"libpd"* na plataforma *Unity*. Para prosseguir, devemos

[24] Disponível em: <https://github.com/LibPdIntegration/LibPdIntegration> Acessado em: 28/07/2021.

selecionar todos os arquivos contidos nas pasta *"LibPdIntegration-master"* e copiá-los, conforme demonstramos abaixo na figura 27.

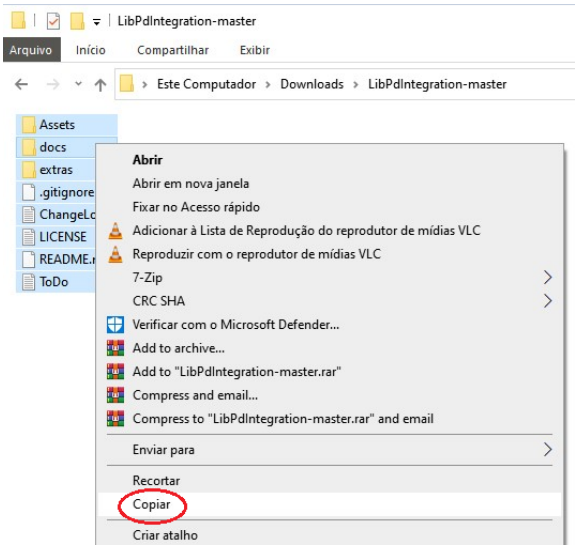


Figura 27 - Selecionando e copiando os arquivos da pasta "*LibPdIntegration-master"*. Fonte: O autor.

Em seguida, devemos "colar" os arquivos copiados na pasta raíz do projeto *Unity*, a qual encontramos e reservamos anteriormente (no nosso caso, a pasta "Projeto teste", figura 28). E pronto, os arquivos necessários para utilizarmos o *Pure Data* no desenvolvimento de um jogo no projeto *Unity* estão na pasta raíz do projeto;

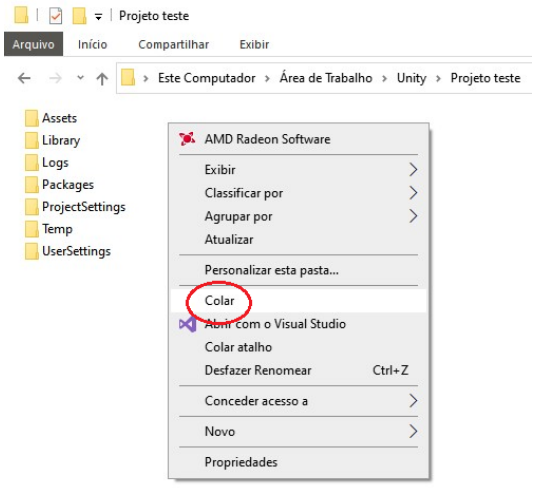


Figura 28 - Colando os arquivos da pasta "*LibPdIntegration-master"* na pasta raíz do projeto "Projeto teste". Fonte: O autor.

Para melhor visualizarmos os processos e etapas para adicionar o *LibPdIntegration* no projeto *Unity* devemos:

1. Acessar o site[25] repositório do *LibPdIntegration*;
2. Clique no botão verde *"Code"* e em seguida *"Download ZIP"*;
3. Localize o arquivo *"LibPdIntegration-master.zip"* baixado em seu computador;
4. Extraia o arquivo baixado "*LibPdIntegration-master*" com o programa de sua preferência (*winrar*, *7-zip* ou outros);
5. Vamos selecionar todos os arquivos contidos nas pasta *"LibPdIntegration-master"* e copiar;
6. Colar os arquivos copiados na pasta raíz do projeto *Unity*;
7. Pronto, os arquivos necessários para utilizar o *Pure Data* no projeto *Unity* estão na pasta raíz do projeto;

6.2.3. Utilizando um *patch Pd* dentro de um projeto *Unity*

Para utilizarmos um *patch Pd* dentro de um projeto *Unity*, vamos seguir as seguintes ações, em nosso caso, um exemplo criado no "Projeto teste" anteriormente mencionado. Em nosso projeto do tipo "*Mobile 2D*", criamos um *game object,* um círculo vermelho (figura 29), o qual vamos utilizar para receber um *patch Pd;*

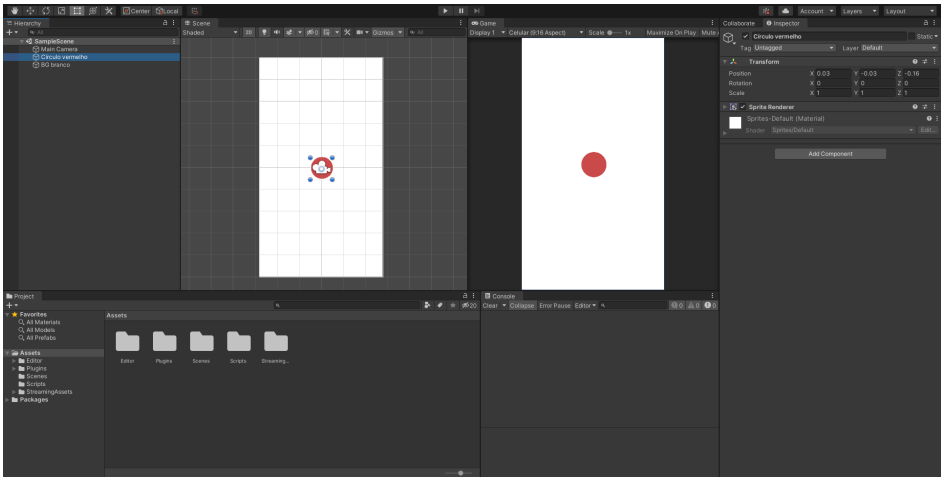


Figura 29 - "Projeto teste" contendo o *game object* "círculo vermelho". Fonte: O autor.

[25] Disponível em: <https://github.com/LibPdIntegration/LibPdIntegration> Acessado em: 28/07/2021.

Para esse teste, e também para melhor visualização das etapas guias, criamos um *patch Pd* chamado "teste" (figura 30). Nesse *patch*, adicionamos um gerador de uma onda senoidal em 300Hz, e também um elemento do tipo *"loadbang"*, que como explicamos na seção 5 deste estudo, é uma das formas de comunicação entre o *Pd* e *Unity*;

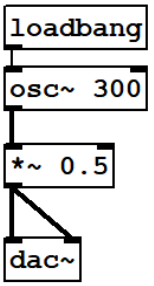


Figura 30 - *Patch Pd* "teste". Fonte: O autor.

Em seguida, vamos mover o *patch Pd* "teste" para a pasta *"Assets/StreamingAssets/PdAssets"* (figura 31) que fica dentro da pasta raiz do projeto (que reservamos na seção 6.2.1), é nessa pasta que você deve colocar todos os *patches Pd* e arquivos relacionados aos seus *patches* que utilizar no projeto do seu *audiogame* na plataforma *Unity*.

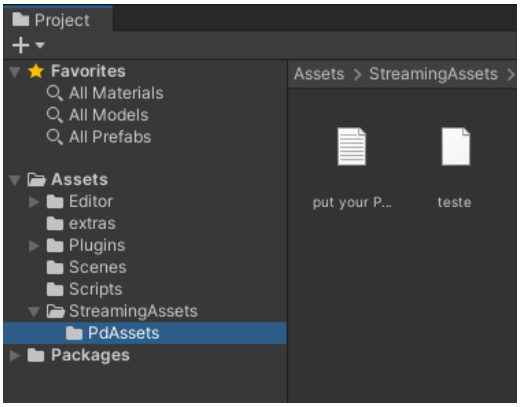


Figura 31 - Pasta do "Projeto teste" onde devem ser colocados os *patches Pd.* Fonte: O autor.

A próxima etapa consiste em adicionar o *script "Lib Pd Instance"* ao *game object*, em nosso caso o objeto "círculo vermelho" que criamos anteriormente. Após isso, note que foram adicionados os elementos *"Audio Source"* e o *script "Lib Pd Instance"* (como explicamos na seção 5.3 e pode ser visto mais detalhadamente na figura 32 abaixo);

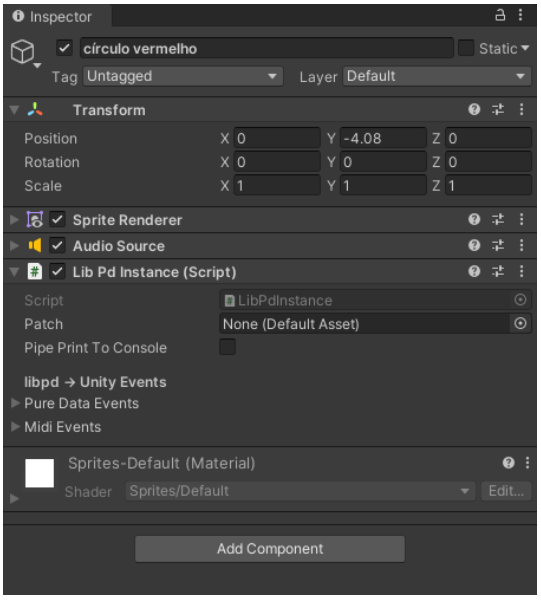


Figura 32 - Elementos *"Audio Source"* e *"Lib Pd Instance"* no *game object* "círculo vermelho". Fonte: O autor.

Para adicionarmos o *patch Pd* ao elemento *game object* precisamos "arrastar" o arquivo "teste" da pasta *"PdAssets"* para o campo *"Patch"* no *script "Lib Pd Instance"* (como na figura 33). Pronto, ao dar *play* no *preview* do jogo no projeto *Unity* o *patch Pd* é acionado e gera em tempo real a onda senoidal através do *game object* "círculo vermelho".

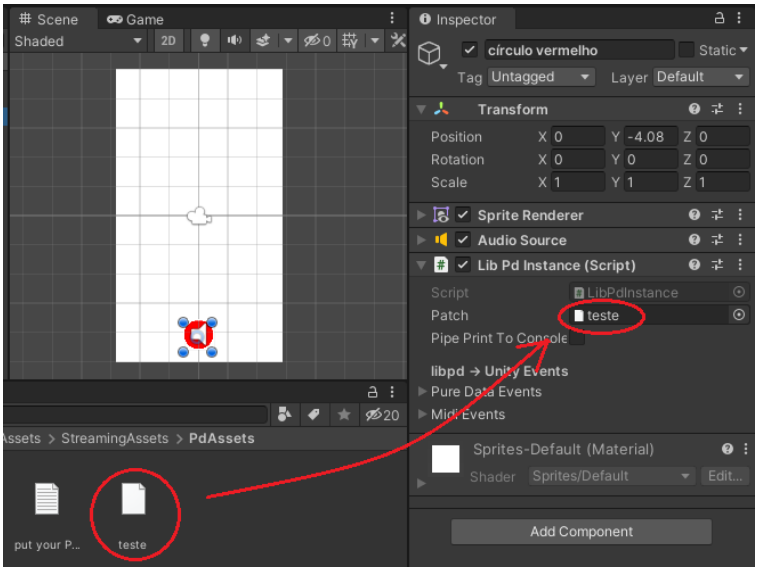


Figura 33 - Adicionando o *patch Pd* "teste" ao campo *Patch* no elemento *"Lib Pd Instance"* no *game object* "círculo vermelho". Fonte: O autor

Em seguida descrevemos os passos de forma numerada para melhor visualização das etapas apresentadas nesta seção:

1. Em nosso projeto do tipo "*Mobile 2D*", criamos um *game object* "círculo vermelho", o qual vamos utilizar para receber um *patch Pd;*
2. Criamos um *patch Pd* chamado "teste", contendo um gerador de uma onda senoidal em 300Hz, e um elemento *"loadbang"*;
3. Vamos mover o *patch Pd* "teste" para a pasta *"Assets/StreamingAssets/PdAssets"* que fica dentro da pasta raiz do projeto;
4. Vamos adicionar o *script "Lib Pd Instance"* ao *game object* "círculo vermelho";
5. Note que foram adicionados os elementos *"Audio Source"* e o *script "Lib Pd Instance"*;
6. Para adicionar o *patch Pd* ao elemento *game object* precisamos arrastar o arquivo "teste" da pasta *"PdAssets"* para o campo *"Patch"* no *script* "*Lib Pd Instance"*;
7. Pronto, ao dar *play* no projeto *Unity* o *patch Pd* é acionado e gera em tempo real a onda senoidal através *game object* "círculo vermelho";

6.2.4. Solucionando o problema de espacialização

Ao utilizarmos um *patch Pd* em um elemento móvel dentro do jogo, vamos notar que não existe espacialização sonora, ou seja, o som é estático independente da posição do elemento no jogo. Como falamos anteriormente na seção 5.4, o *Unity* não aplica espacialização a *patches Pd* através de uma instância do *libpd.* Dito isto, temos duas possibilidades funcionais para contornar esse problema.

A primeira possibilidade é utilizar um arquivo de áudio especial chamado *"SpatialiserFix"* disponibilizado na pasta *"extra"* (a qual movemos para a pasta raíz do nosso projeto, na seção 6.2.2). A segunda maneira, é utilizando um plugin externo de espacialização, como o *Steam Audio* e assim simular a sensação espacial sonora.

Seguindo as indicações presentes no item 6.2.4.1, vemos a primeira maneira de contornar esse problema de espacialização sonora de *patches Pd* na plataforma *Unity*.

6.2.4.1. Primeira solução

Para realizarmos a primeira solução de espacialização, primeiramente devemos fazer uma pequena modificação no nosso *patch Pd*. Devemos adicionar um elemento do tipo "*adc~"* e em seguida multiplicar os sinais de saída antes de enviar o sinal para a saída "*dac~"*, como demonstrado na figura 34 abaixo.

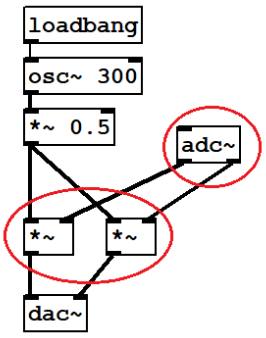


Figura 34 - *Patch Pd* "teste" com alteração. Fonte: O autor.

Em seguida, vamos localizar o arquivo "*SpatialiserFix.wav"* na pasta "*extra"* (que copiamos para a pasta raiz na seção 6.2.2), logo após vamos "arrastar" o arquivo *"SpatialiserFix.wav"* para o campo *"AudioClip"* no elemento "*Audio Source"* (que foi adicionado na seção 6.2.3, e que pode ser observado na figura 35);

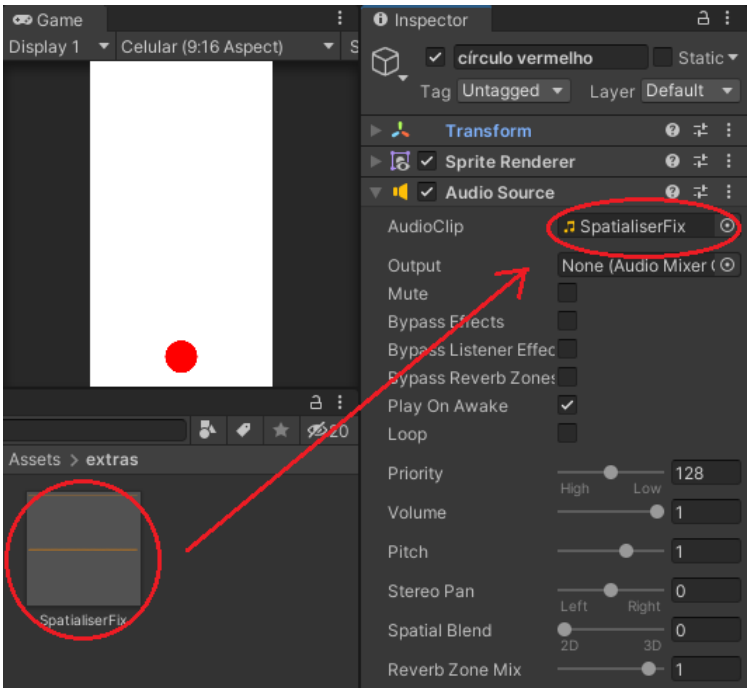


Figura 35 - Adicionando o arquivo *"SpatialiserFix"* ao campo AudioClip no elemento *"Audio Source"*. Fonte: O autor.

Feito isso, devemos nos certificar de marcar a opção *"Play On Awake"*, para que o arquivo *"SpatialiserFix.wav"* inicie assim que seu carregamento esteja pronto. Marcamos também a opção *"Loop"*, para que o áudio fique repetindo em loop infinitamente. E por último, devemos também colocar o *"Spacial Blend"* em 3D nas opções do elemento *"Audio Source"*, como demonstrado na figura 36 em seguida.

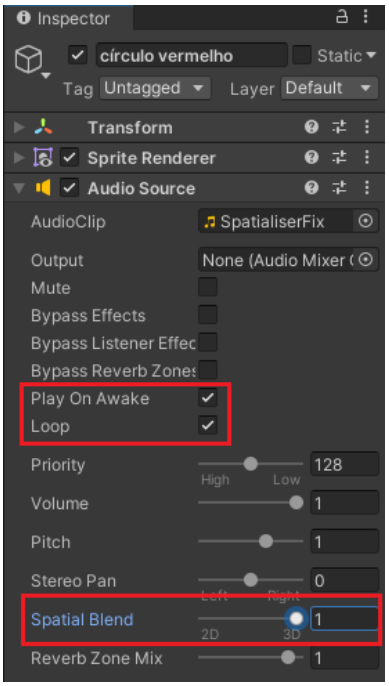


Figura 36 - Opções *"Play On Awake"*, *"Loop"* e *"Spacial Blend"* no elemento *"Audio Source"*. Fonte: O autor.

Pronto, feito isto o *Unity* vai processar a espacialização sonora através do elemento *"Audio Source"* e solucionar o problema anteriormente apresentado. Devemos destacar que esse procedimento deve ser realizado em cada instância do "*Lib Pd Integration"*. Para melhor entendimento das etapas realizadas nesta seção, podemos seguir:

1. Devemos fazer uma pequena modificação no nosso *patch Pd*, adicionando um elemento "*adc~"* e multiplicar os sinais de saída antes de enviar o sinal para a saída "*dac~"*;
2. Em seguida, vamos localizar o arquivo "*SpatialiserFix.wav"* na pasta "*extra"* (que copiamos para a pasta raiz;
3. Vamos arrastar o arquivo *"SpatialiserFix.wav"* para o campo *"AudioClip"* no elemento "*Audio Source"*;
4. Em seguida devemos nos certificar de marcar as opções *"Play On Awake"* e *"Loop"*, e também colocar o *"Spacial Blend"* em 3D;
5. Pronto, feito isto o *Unity* vai processar a espacialização através do elemento *"Audio Source"*;

6.2.4.2. Segunda solução

Para a segunda solução, vamos seguir as indicações a seguir para utilizar um plugin de espacialização externo, nesse caso, utilizamos o *Steam Audio*. Primeiro, devemos acessar o site do *Steam Audio*[26] e realizar o *download* do *"Unity Plugin (ZIP)"*. Assim como fizemos na seção 6.2.2, vamos extrair o conteúdo do arquivo *"steamaudio_unity.zip"*. Na tela principal do *Unity*, vamos na opção *Assets* > *Import Package* > *Custom Package,* como demonstrado na figura 37, e em seguida, vamos selecionar o arquivo *"SteamAudio"* que extraímos anteriormente.

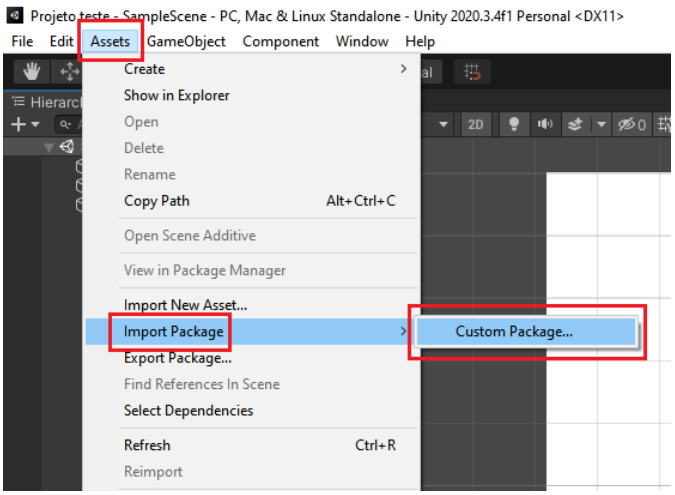


Figura 37 - Menu *Assets* na tela principal do *Unity.* Fonte: O autor.

Em seguida, na tela principal do *Unity*, vamos na opção *Edit* > *Project Settings*... e na aba *"Audio"* devemos alterar as opções *"Spatializer Plugin"* e *"Ambisonic Decoder Plugin"* para o plugin *"Steam Audio"* (como na figura 38), com isso o plugin está habilitado.

[26] *Steam Audio.* disponível em: <https://valvesoftware.github.io/steam-audio/downloads.html> Acessado em: 29/07/2021.

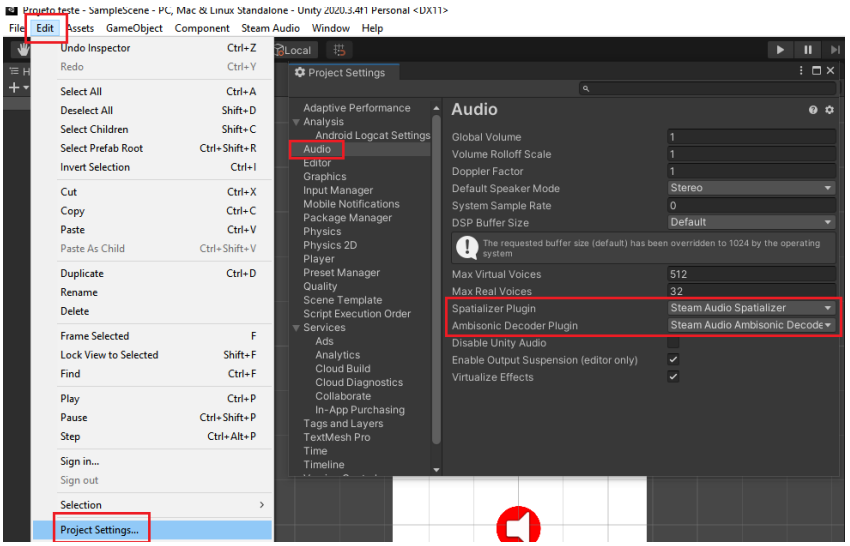


Figura 38 - Habilitando o plugin *"SteamAudio"* no *Unity*. Fonte: O autor.

Por último, no elemento *"Audio Source"* do *game object* "círculo vermelho", devemos nos certificar de assinalar as opções *"Spatialize"*, *"Spatialize Post Effects"* e *"Spacial Blend"* em 3D (figura 39), assim a espacialização do *plugin* externo estará habilitado no *game object*. Pronto, com isso a espacialização do *patch Pd* será processada pelo *plugin* externo do *"Steam Audio"*.

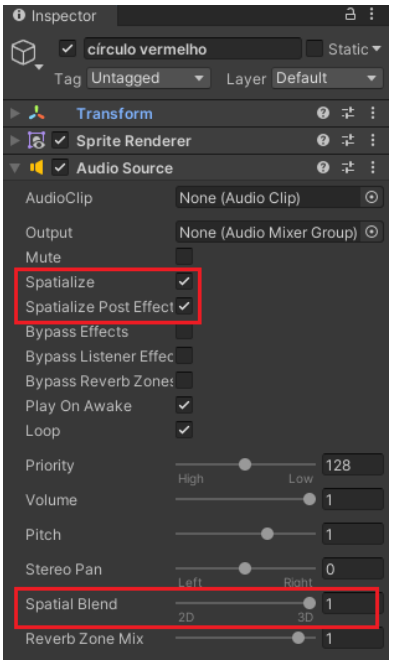


Figura 39 - Opções *"Spatialize"*, *"Spacialize Post Effects"* e *"Spacial Blend"* no elemento *"Audio Source"*. Fonte: O autor.

Dito isto, precisamos destacar que devemos utilizar apenas uma das soluções para a espacialização dos *patches Pd* na plataforma *Unity*. Para melhor entendimento das etapas, podemos observar a lista:

1. Primeiro vamos acessar o site do *Steam Audio*[27] e realizar o *download* do *"Unity Plugin (ZIP)"*;
2. Assim como fizemos na seção 6.2.2, vamos extrair o conteúdo do arquivo *"steamaudio_unity.zip"*;
3. Na tela principal do *Unity*, vamos na opção *Assets > Import Package > Custom Package*;
4. Em seguida, vamos selecionar o arquivo *"SteamAudio"* que extraímos na etapa 2;
5. O próximo passo é importar o plugin de espacialização do *Steam Audio*;
6. Em seguida, na tela principal do *Unity*, vamos na opção *Edit > Project Settings...* e na aba *"Audio"* vamos alterar as opções *"Spatializer Plugin"* e *"Ambisonic Decoder Plugin"* para o plugin *"Steam Audio"*;
7. Em seguida, no elemento *"Audio Source"* do *game object* "círculo vermelho", devemos nos certificar de marcar as opções *"Spatialize"*, *"Spatialize Post Effects"* e *"Spacial Blend"* em 3D;
8. Pronto, com isso a espacialização do *patch Pd* será processada pelo *plugin* externo do *"Steam Audio"*.

A partir do uso dessas duas soluções para o problema, observamos algumas características. Para o nosso projeto *Binaural Flight*, a primeira solução foi a mais eficiente, devido ao fato do jogo ser 2D, o som ficou melhor espacializado desta maneira. Em um projeto de jogo 3D, as duas soluções se mostram eficientes no processamento do áudio, e devem atender bem aos desenvolvedores.

[27] *Steam Audio.* disponível em: <https://valvesoftware.github.io/steam-audio/downloads.html> Acessado em: 29/07/2021.

7. CONCLUSÃO

No decorrer desta pesquisa, investigamos o uso da linguagem de programação *Pure Data* no desenvolvimento de *audiogames mobile* para pessoas com deficiência visual na plataforma *Unity*. A partir disto, podemos notar que é uma área com bastante espaço para novas pesquisas e desenvolvimentos, visto que, foram encontrados poucos estudos e propostas semelhantes nesse campo de estudo. Observamos que o *Pure Data* já vem sendo utilizado na criação de música procedural e também em jogos, e este é um fator que explica a grande produção e distribuição de *patches Pd* pela comunidade. Infelizmente não encontramos registro nenhum de *audiogames mobile* para pessoas com deficiência visual que utilizaram o *Pure Data*, fator relevante para esta pesquisa, visto que, nosso objetivo é fomentar novas produções e instigar os desenvolvedores independentes.

Observamos a importância dos *audiogames* para os IDV, seja no âmbito educacional, treinamento motor ou de lazer. Durante este estudo constatamos que são poucas produções, mas que existem diversos estudos que comprovam a importância deste tipo de mídia. Existem diversos *frameworks* para o desenvolvimento de *audiogames*, desde *serious games* até os de puro entretenimento. São utilizadas diversas plataformas de desenvolvimento de jogos (as *game engines),* e existem diversas *engines* de áudio, contudo, grande parte são *softwares* e *middlewares* pagos, o que ressalta ainda mais a importância e relevância do uso do *Pure Data* como uma linguagem de programação visual simples e *open source*.

A partir deste estudo, propomos e desenvolvemos o *Binaural Flight,* um *audiogame* simples para teste de implementação e funcionalidades do *Pd*. O *Binaural Flight* é um *endless scrolling* vertical, onde o jogador se baseia na espacialização sonora para desviar de obstáculos. Durante o desenvolvimento, conseguimos identificar diversas características positivas dessa linguagem de programação visual no desenvolvimento de um *audiogame* para dispositivos móveis na plataforma *Unity.*

Como resultado da criação do *audiogame Binaural Flight*, desenvolvemos e propomos soluções viáveis para a utilização de *patches Pure Data* no desenvolvimento de *audiogames mobile*, e sua implementação através do *LibPdIntegration* na plataforma *Unity*. Além das soluções, criamos listas de etapas para facilitar o entendimento das etapas de utilização das ferramentas, contendo informações e dicas valiosas sobre os possíveis

problemas que encontramos, suas soluções e também sobre funcionalidades ainda inexploradas.

Com esta pesquisa podemos concluir que é possível utilizar o *Pure Data* no desenvolvimento de *audiogames mobile* para pessoas com deficiência visual na plataforma *Unity*. Entretanto, devemos destacar também que ainda existem diversos cenários que podem ser explorados neste contexto. Além disso, encontramos grande valia no uso dos *patches Pd,* visto que seria possível até mesmo utilizar o *Pd* como uma *engine* de áudio, criando e manipulando sons em tempo real dentro de um *audiogame*. Dito isto, esperamos que este estudo possa ajudar de alguma maneira a fomentar novas produções acadêmicas e também instigue os desenvolvedores independentes a utilizar essas ferramentas gratuitas para criação de novos *audiogames mobile.*

7.1. Limitações e perspectivas futuras

Durante a pesquisa, houveram diversas limitações e dificuldades, como mencionamos antes, a falta de experiência do autor deste estudo com a área de programação foi um dos limitadores desta pesquisa, o investimento de tempo e recurso para criar e desenvolver soluções poderiam ser mais simples e rápidas. Da mesma forma, vemos essa limitação como um grande sucesso nesta pesquisa, visto que, nosso objetivo é fomentar novas produções independentes, e facilitar o desenvolvimento de *audiogames*, de certa forma o *Binaural Flight* é uma prova de que mesmo uma pessoa sem muito conhecimento de programação pode utilizar essas ferramentas com os objetivos levantados por esta pesquisa.

Outro fator importante de extrema relevância, é o fato de estamos vivendo uma pandemia global, infelizmente o COVID 19 tem matado milhões de pessoas, atrapalhado a vida de todo mundo, e com os pesquisadores também não é diferente, a limitação de recursos, a quarentena e também a nossa saúde mental com certeza foram limitantes em nosso dia-a-dia e em nossa vida. Por conta disso, infelizmente não conseguimos implementar algumas das funções importantes no *audiogame Binaural Flight* e também não foi possível testá-lo com o IDV, e assim o jogo ficou limitado a apenas um teste de implementação.

Sobre as perspectivas futuras, inicialmente podemos criar um vídeo tutorial ensinando como utilizar as soluções propostas nesta pesquisa, visto que não existe conteúdo sobre esse tema na língua portuguesa. Seria uma ótima contribuição para o cenário brasileiro de

desenvolvimento de *audiogames mobile.* Observamos muito potencial nesta pesquisa, principalmente na manipulação dos elementos sonoros e suas interações dentro de uma *audiogame* em tempo real. Para isto, podemos imaginar o uso do *Pure Data* como uma *engine* de áudio, utilizando *patches Pd* como modificadores de sinal sonoro em tempo real, como pequenos blocos de efeitos sonoros (*subpatches* do *Pd*), os quais seriam parte de um sistema sonoro. Entre as principais ferramentas das *engines* de áudio levantadas nesta pesquisa podemos destacar:

Patches Pure Data	
Panorama	*Patch* responsável pelo posicionamento espacial do som (L e R);
Chorus	Efeito sonoro *Chorus*;
Delay	Efeito de repetição sonora com atraso;
Reverb	Efeito que simula uma câmara de eco ou espaço físico;
Flanger	Efeito sonoro *Flanger*;
Pitch shifter	*Patch* responsável pela manipulação de tonalidade de um som;
Phaser	Efeito sonoro *Phaser*;

Tabela 11 - *Patches Pure Data.*

Para esse sistema sonoro, as ferramentas levantadas anteriormente iriam servir como base para os *patches Pd*, ou seja, buscar e/ou desenvolver os *patches* que sejam equivalentes às ferramentas das *engines* de áudio. Após realizar uma pesquisa nos bancos de *patches Pd* e em fóruns de áudio, fizemos um levantamento inicial dos *patches* (Tabela 11), os quais estão disponíveis detalhadamente no anexo II.

Podemos desde já imaginar a viabilidade do uso do *Pure Data* como uma *engine* de áudio, visto que, até o momento a maioria das funções oferecidas pelas *engines* de áudio do mercado podem de alguma forma ser desenvolvidas como *patches Pd*, obviamente existem diversas funções específicas oferecidas pelos *middlewares* que não seriam capazes de ser desenvolvidas, porém, esse sistema seria como uma *engine* de áudio gratuita para a criação de *audiogames mobile* para pessoas com deficiência visual.

7.2. Contribuições

Neste estudo, acreditamos ter contribuído de diversas maneiras diferentes para o cenário independente de *audiogames mobile*. A maior contribuição deste estudo são as soluções para utilização e implementação do *Pure Data* na plataforma *Unity*, no contexto de desenvolvimento de *audiogames mobile* para indivíduos com deficiência visual. Visto que não foram encontrados registros de materiais semelhantes a este em linguagem portuguesa. Acreditamos que disponibilizando essas soluções gratuitamente na internet podemos contribuir para a fomentação de novas produções, *audiogames* e tornar esse campo de estudo ainda mais relevante.

Podemos destacar também o levantamento bibliográfico e panorama geral criado através da revisão sistemática literal sobre a temática desta pesquisa, referente aos últimos anos de publicações, o qual observamos que existe muito espaço para novas produções neste campo de estudo.

Como resultado dos desdobramentos desta pesquisa, vamos disponibilizar o projeto *Unity* do jogo *Binaural Flight*, que pode servir de base e auxiliar pessoas que queiram iniciar no uso do *Pure Data* no desenvolvimento de *audiogames* para IDV na plataforma *Unity*. Também será disponibilizado todos os *patches Pd* e códigos fonte do *audiogame Binaural Flight* (Anexos), além de que, temos interesse em melhorar e implementar novas funções no jogo, a fim de torná-lo um produto completo e disponibilizá-lo gratuitamente nas plataformas mobile.

7.3. Considerações Finais

No final desta pesquisa, acreditamos ter alcançado os objetivos possíveis e propostos no início do trabalho apresentado ao programa de Pós-Graduação em Computação, Comunicação e Artes (PPGCCA). O desenvolvimento do *audiogame Binaural Flight* é uma prova clara de que é possível usar o *Pure Data* na criação de *audiogames mobile* para IDV. Observamos também como essas ferramentas são acessíveis para iniciantes, visto que, no início deste trabalho o autor não possuía experiência em programação e desenvolvimento de jogos.

As soluções propostas por esta pesquisa são suficientes para introduzir desenvolvedores iniciantes ou independentes no uso do *Pure Data* no desenvolvimento de

audiogames na plataforma *Unity.* O *Binaural Flight* mostra que é possível e que é viável utilizar essas ferramentas em conjunto.

Dito isto, vemos com bons olhos os resultados desta pesquisa, na esperança que possa servir de auxílio e inspiração para a área de desenvolvimento de *audiogames* para pessoas com deficiência visual, e que no futuro possamos ver ainda mais produções, *audiogames* e pesquisas nessa área.

8. REFERÊNCIAS

About Pure Data, Disponível em: <https://puredata.info/> Acessado em: 28/07/2020.

AMENGUAL GARÍ, Sebastià Vicenç et al. **Evaluation of real-time sound propagation engines in a virtual reality framework.** In: Audio Engineering Society Conference: 2019 AES International Conference on Immersive and Interactive Audio. Audio Engineering Society, 2019.

ARAÚJO, Maria CC et al. **Um estudo das recomendações de acessibilidade para audiogames móveis.** XIV Simpósio Brasileiro de Jogos e Entretenimento Digital-ISSN, p. 2179-2259, 2015.

Audio Games. Disponível em: <https://www.audiogames.net/> Acessado em: 26/07/2020.

AVILA, Gustavo et al. **Soundmaze: Desenvolvimento de um audiogame para deficientes visuais.** Revista Ibérica de Sistemas e Tecnologias de Informação, n. E26, p. 488-500, 2020.

BĂLAN, Oana et al. **Navigational 3D audio-based game-training towards rich auditory spatial representation of the environment.** In: 2014 18th International Conference on System Theory, Control and Computing (ICSTCC). IEEE, 2014. p. 682-687.

BĂLAN, Oana; MOLDOVEANU, Alin; MOLDOVEANU, Florica. **Navigational audio games: an effective approach toward improving spatial contextual learning for blind people.** International Journal on Disability and Human Development, v. 14, n. 2, p. 109-118, 2015.

BRINKMANN, Peter et al. **Embedding pure data with libpd.** In: Proceedings of the Pure Data Convention. Citeseer, 2011.

DA CONCEIÇÃO, Maria et al. **Ortomonstro: um audiogame móvel customizável para práticas ortográficas de português por meio do Braille.** In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE). 2016. p. 866.

DA SILVA, Isabel Cristina Siqueira. **The Promotion of Empathy for the Experience of Users with Visual Impairment in the Game Design Education.** In: International Conference on Human-Computer Interaction. Springer, Cham, 2019. p. 326-341.

DE LIMA, Ramon Paulino. **ANÁLISE DO AUDIO GAME HUB: UMA EXPERIÊNCIA SENSORIAL PARA PESSOAS COM DEFICIÊNCIA VISUAL.** Games e gamificação, p. 93-109, 2019.

Elias Adaptive Music. Disponível em: <https://www.eliassoftware.com/> Acessado em: 15/01/2020.

ESCARRONE, Anne Lize Lopes; CHEIRAN, Jean Felipe Patikowski; MOURA, Edison Jhonantan Regina. **Identificando características da acessibilidade Web Mobile:**

Desenvolvimento de um jogo acessível para dispositivos móveis. Anais do Salão Internacional de Ensino, Pesquisa e Extensão, v. 7, n. 2, 2016.

FMOD Studio. Disponível em: <https://www.fmod.com/studio> Acessado em: 15/01/2020.

GAL, Viviane et al. **Processes and tools for sound design in computer games.** In: Proceedings International Computer Music Conference. 2002.

Game Audio Tools. Disponível em: <https://instabug.com/blog/game-audio-tools/> Acessado em: 15/01/2020.

HANSEN, Kjetil Falkenberg; HIRAGA, Rumi. **The effects of musical experience and hearing loss on solving an audio-based gaming task.** Applied Sciences, v. 7, n. 12, p. 1278, 2017.

INTRODUCTION TO DIGITAL FILTERS WITH AUDIO APPLICATIONS. Disponível em: <https://ccrma.stanford.edu/~jos/filters/filters.html> Acessado em: 28/07/2020.

JULIANI, Arthur et al. **Unity: A general platform for intelligent agents.** arXiv preprint arXiv:1809.02627, 2018.

KANE, Shaun K.; KOUSHIK, Varsha; MUEHLBRADT, Annika. **Bonk: accessible programming for accessible audio games.** In: Proceedings of the 17th ACM Conference on Interaction Design and Children. 2018. p. 132-142.

MONTEIRO, Caio et al. **Imersão e medo em jogos de terror: análise das estruturas de áudio e efeitos sonoros no jogo Blindside.** Revista SBGames, p. 2179-2259, 2016.

MOODY, Niall. **LibPdIntegration.** Disponível em: < https://github.com/LibPdIntegration/LibPdIntegration> Acessado em: 19/07/2021.

PASSOS, Leonardo Porto; ARRUDA, Leonardo; FORNARI, José. **Pure Data for Pure Audio Games.** 2020.

PAUL, Leonard J. **Audio prototyping with pure data.** Gamasutra, May, v. 30, 2003.

pd-patches made for 'generative radio' Disponível em: **<http://www.martin-brinkmann.de/generativemusic.html>** Acessado em: 28/07/2020.

Phaser / Chorus / Flanger. Disponível em: <https://forum.pdpatchrepo.info/topic/9942/phaser-chorus-flanger> Acessado em: 28/07/2020.

PRODANOV, Cleber Cristiano; DE FREITAS, Ernani Cesar. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição.** Editora Feevale, 2013.

PUCKETTE, Miller S. et al. **Pure Data**. In: ICMC. 1997.

RF, SAMPAIO. **Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica,** 2007.

SEZNEC, Yann. **Yann Seznec Youtube Channel**. Disponível em: <https://www.youtube.com/user/amazingrolo/featured> Acessado em: 19/07/2021.

Smartphones lideram mercado de jogos com folga. Disponível em : <https://canaltech.com.br/mercado/smartphones-lideram-industria-de-jogos-160217/> Acessado em: 23/07/2020.

The Miles Sound System. Disponível em: <http://www.radgametools.com/miles.htm> Acessado em: 15/01/2020.

Unity Core Platform. Disponível em: <https://unity.com/pt/products/core-platform> Acessado em: 15/01/2020.

Unreal Engine. Disponível em: <https://www.unrealengine.com/en-US/features> Acessado em: 15/01/2020.

VENDOME, Christopher et al. **Can everyone use my app? An Empirical Study on Accessibility in Android Apps.** In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2019. p. 41-52.

Web Content Accessibility Guidelines (WCAG) 2.2 Disponível em: <https://www.w3.org/TR/WCAG22/#requirements-for-wcag-2-2> Acessado em: 14/07/2020.

WIERMANS, Mathijs. **PROCEDURAL MUSIC IN GAMES, A VIABLE ALTERNATIVE?,** 2016.

World health organization - **blindness and visual impairment**. Disponível em: <http://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment> Acessado em: 23/07/2020.

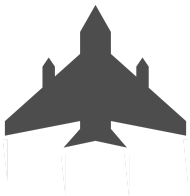
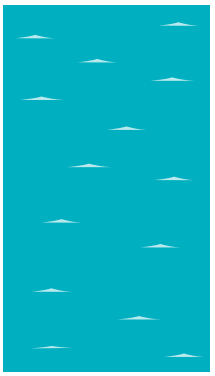
Wwise. Disponível em: <https://www.audiokinetic.com/products/wwise/> Acessado em: 15/01/2020.

ZACHMAN, John A. **A framework for information systems architecture.** IBM systems journal, v. 26, n. 3, p. 276-292, 1987.

9. ANEXOS

ANEXO I

Assets e Sprites Binaural Flight

Aeronave	Dirigível	Background	Tela inicial	Fim de jogo
				

Código Fonte *Binaural Flight*

Script - "inicio"

inicio.cs

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5
6   public class inicio : MonoBehaviour{
7       void Start()
8       {
9
10      }
11      void Update()
12      {
13          if (Input.touchCount > 0){
14              Touch touch = Input.GetTouch(0);
15              SceneManager.LoadScene("Gameplay");
16          }
17      }
18  }
```

Script - "scroll"

scroll.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class scroll : MonoBehaviour{

    public float speed = 0.5f;

    void Start()
    {

    }

    void Update()
    {
        Vector2 offset = new Vector2 (0, Time.time * speed);

        GetComponent<Renderer>().material.mainTextureOffset = offset;
    }
}
```

Script - "SwipeMove"

SwipeMove.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SwipeMove : MonoBehaviour {

    private Vector2 startTouchPosition, endTouchPosition;

    private void Update()
    {
        if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Began)
            startTouchPosition = Input.GetTouch(0).position;

        if (Input.touchCount > 0 && Input.GetTouch(0).phase == TouchPhase.Ended)
        {
            endTouchPosition = Input.GetTouch(0).position;

            if ((endTouchPosition.x < startTouchPosition.x) && transform.position.x > -1.75f)
                transform.position = new Vector2(transform.position.x - 1.75f, transform.position.y);

            if ((endTouchPosition.x > startTouchPosition.x) && transform.position.x < 1.75f)
                transform.position = new Vector2(transform.position.x + 1.75f, transform.position.y);
        }
    }

    void OnTriggerEnter2D(){

        SceneManager.LoadScene("Gameover");
    }
}
```

Script - "moveObject"

moveObject.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class moveObject : MonoBehaviour
{

    public float speed;
    private float y;

    void Start(){

    }

    void Update()
    {
        y = transform.position.y;
        y += speed * Time.deltaTime;

        transform.position = new Vector3(transform.position.x, y, transform.position.z);

        if(y<=-5.5f){
            Destroy(transform.gameObject);
        }

    }
}
```

Script - "spawnController"

spawnController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class spawnController : MonoBehaviour
{

    public GameObject navioPrefab;
    public float rateSpawn;
    private float currentTime;
    private float posicao;
    private float x;
    public float posA;
    public float posB;
    public float posC;

    void Start(){
        currentTime = 0;

    }

    void Update(){

        currentTime += Time.deltaTime;
        if(currentTime >=rateSpawn){
            currentTime = 0;
            posicao = Random.Range(1,3);
            if(posicao <= 1){
                x = posA;
            }
            else if(posicao > 1 && posicao <=2){
                x = posB;
            }
            else{
                x = posC;
            }
            GameObject tempPrefab = Instantiate(navioPrefab) as GameObject;
            tempPrefab.transform.position = new Vector3(x, transform.position.y, tempPrefab.
            transform.position.z);
        }
    }
}
```

Script - "telapreta"

telapreta.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class telapreta : MonoBehaviour
{
    public GameObject Tela;

    public void opentela()
    {
        if(Tela != null)
        {
            bool isActive = Tela.activeSelf;

            Tela.SetActive(!isActive);
        }
    }
}
```

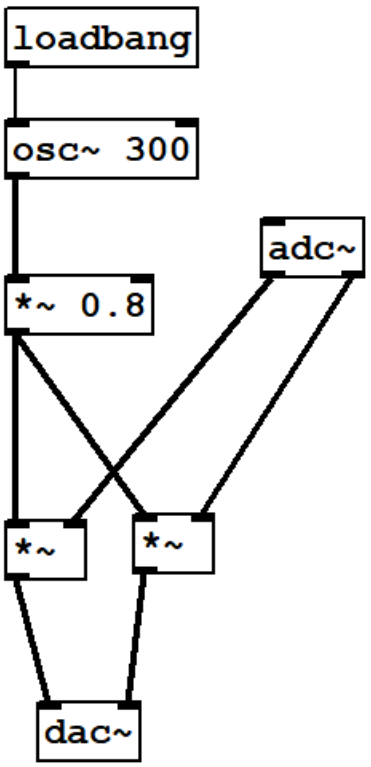
Script - “fim”

fim.cs

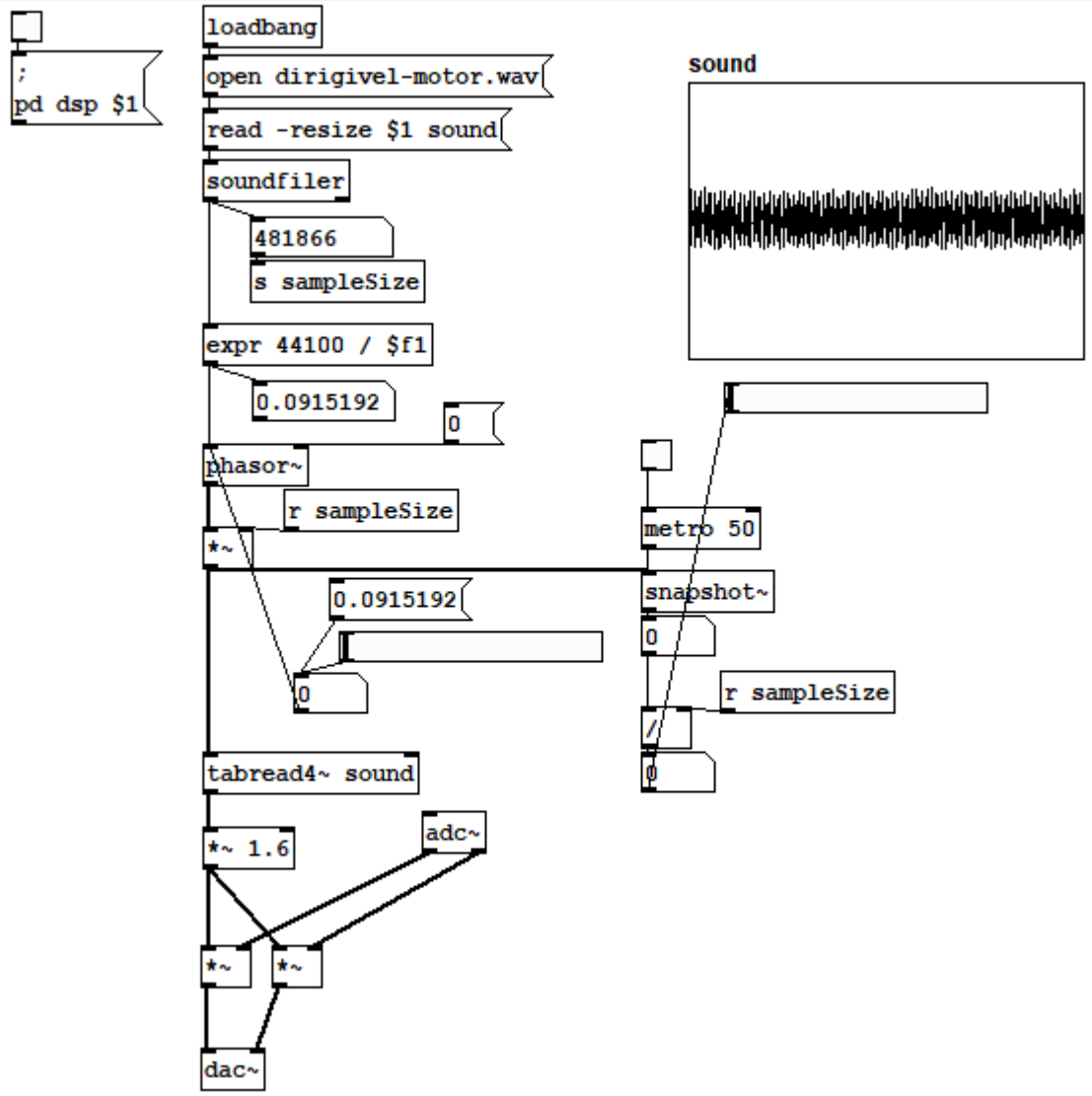
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class fim : MonoBehaviour{
7
8      void Start()
9      {
10
11     }
12
13     void Update()
14     {
15         if (Input.touchCount > 0){
16             Touch touch = Input.GetTouch(0);
17             SceneManager.LoadScene("Gameplay");
18         }
19     }
20 }
```

Patches Pd* utilizados no jogo *Binaural Flight

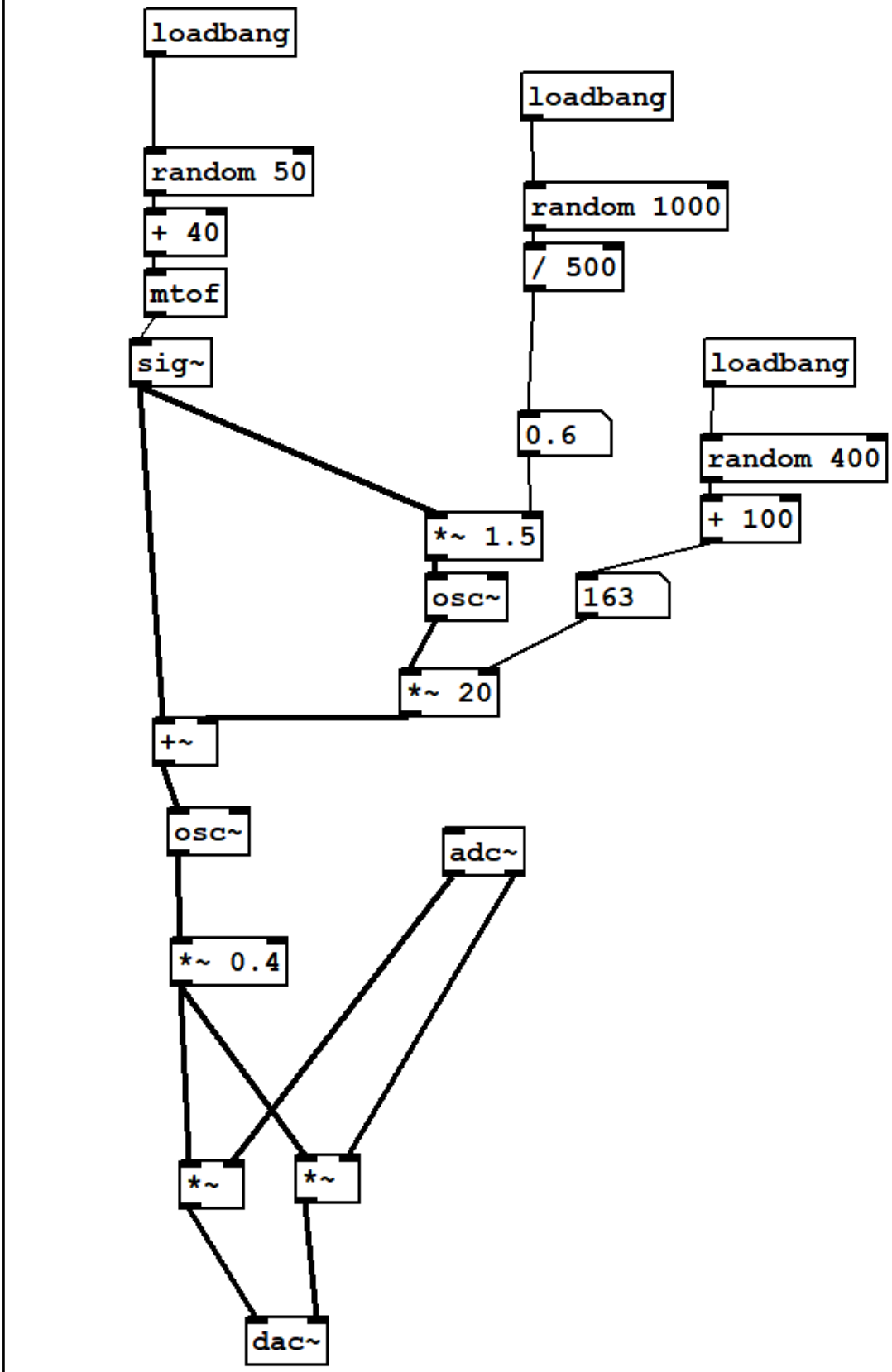
Patch Pd - dirigivel 1.Pd



Patch Pd - dirigivel 2.Pd



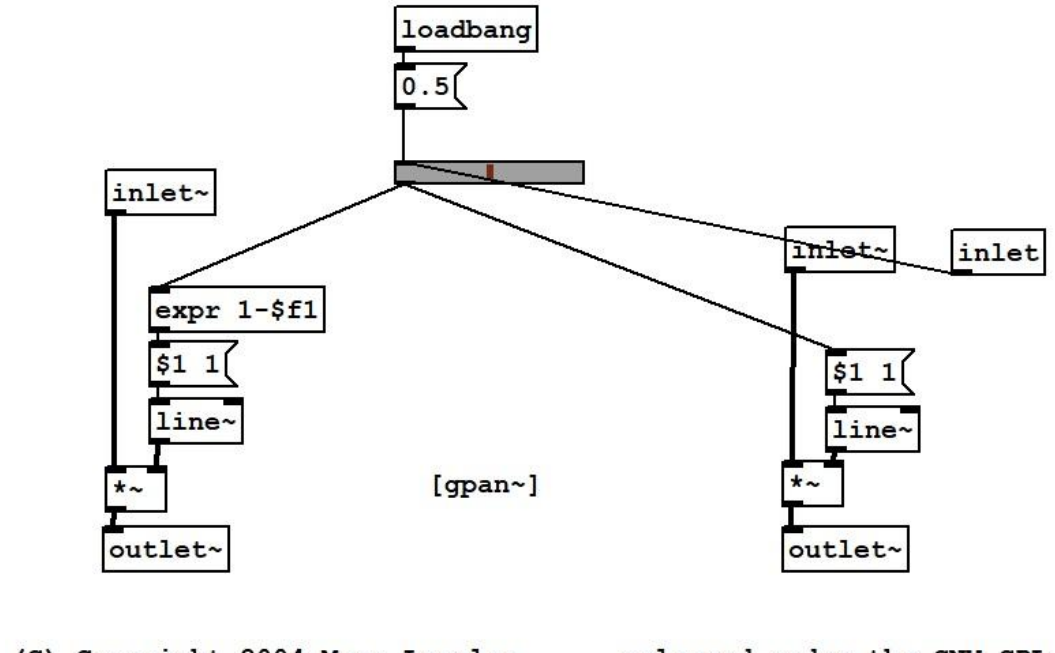
Patch Pd - dirigivel - random.pd



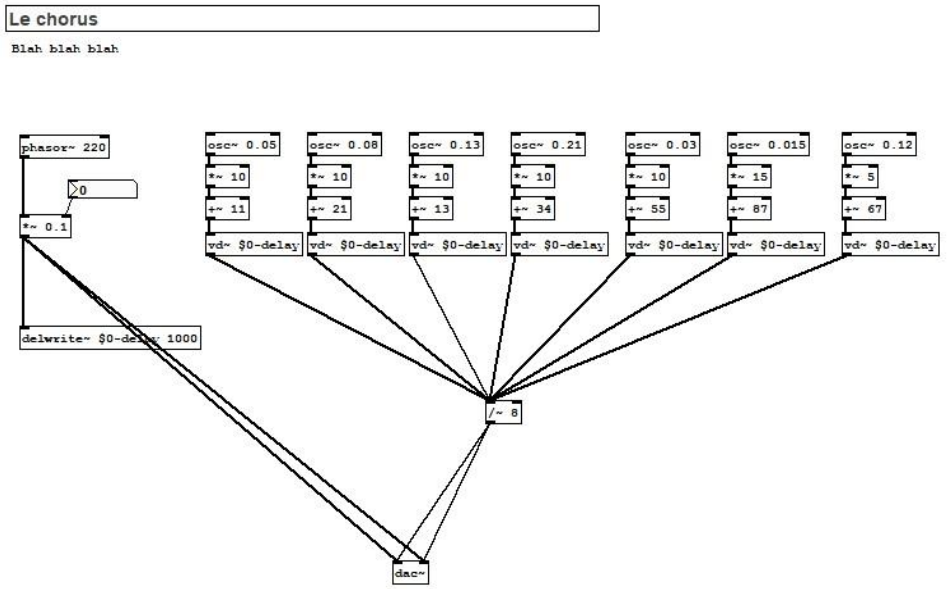
ANEXO II

Outros *Patches Pure Data*

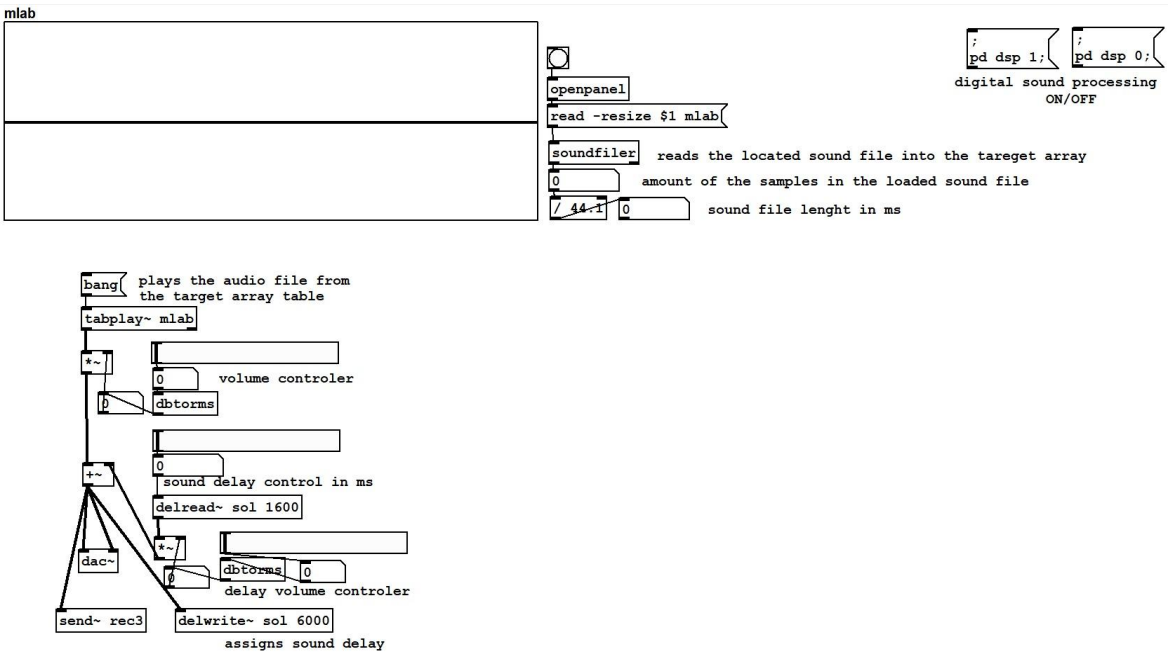
Patch Pd - Panorama



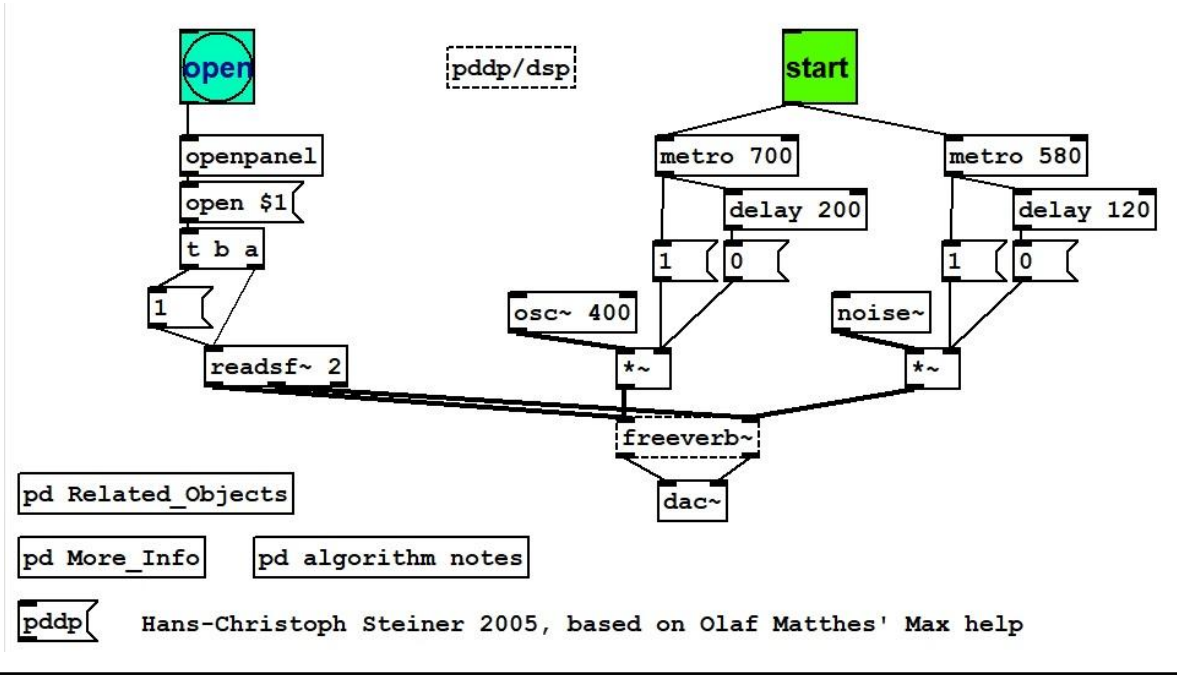
Patch Pd - Chorus



Patch Pd - Delay



Patch Pd - Reverb

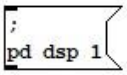


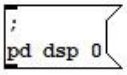
Patch Pd - Flanger

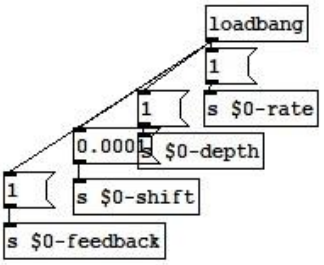
The flanger est un traitement audio où le signal original est mixé avec une version retardé de lui même et dont le temps de retard est contrôlé par un LFO (low frequency oscillator);
donnant l'effet de flanging.

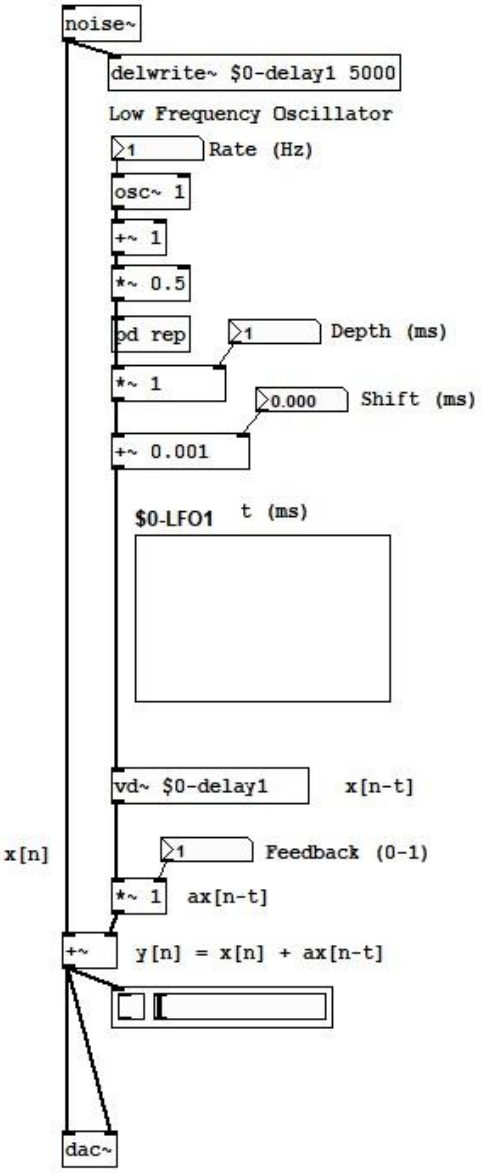
Cela produit un filtre en peigne (y[n] = x[n] + ax[n-t]) qui balaye les fréquences selon la fréquence (rate Hz), la profondeur (depth ms) et un offset (shift ms) du LFO et un coefficient a (feedback 0-1) de réinjection.

Une des premières utilisations de cet effet se trouve sur le morceau "Tomorrow Never Knows" de l'album « Revolver » des Beatles, où le flanging est utilisé pour artificiellement doubler la voix de John Lennon

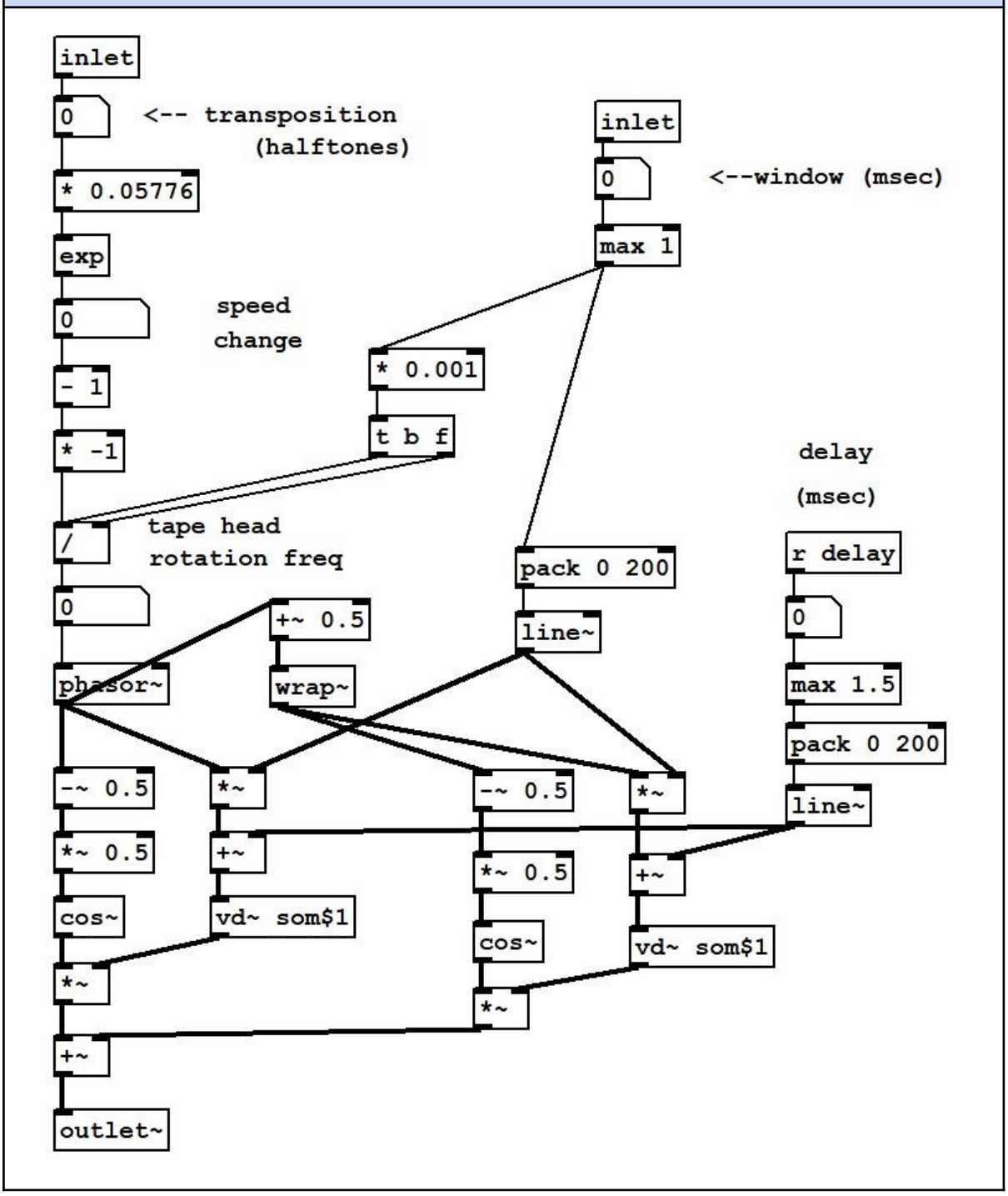








Patch Pd - Pitch shifter



Patch Pd - Phaser

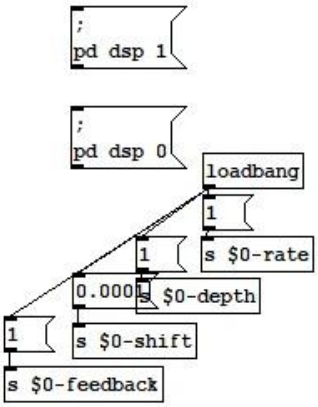
The phaser is an effect unit where the original signal is mixed with a version of itself filtered by a set of variable all-pass filters.

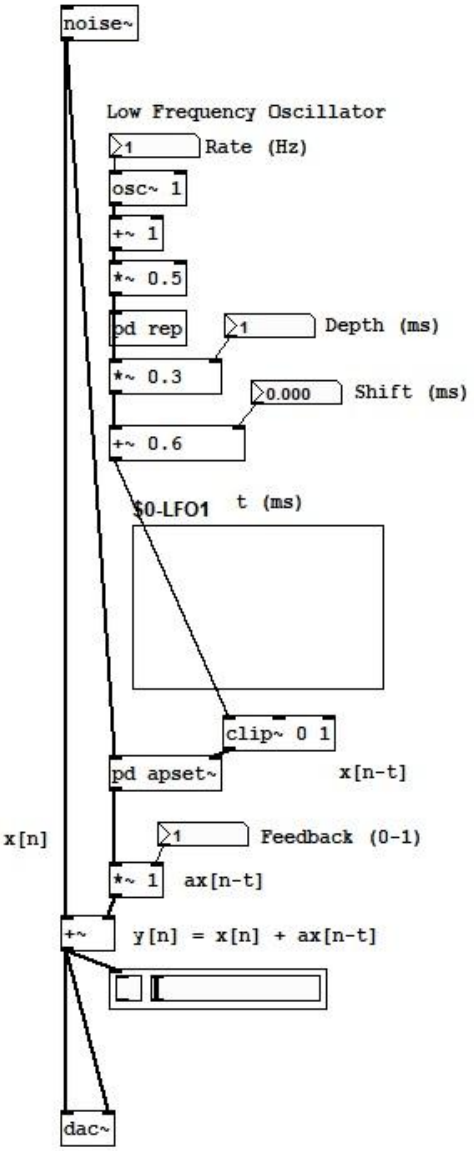
The all-pass filter can be created with an one zero filter (y[n] = -a[n] * x[n] + x[n-1]) and a pole filter (y[n] = x[n] + a[n] * y[n-1]) and the coefficient is driven by a LFO (low frequency oscillator).

The all-pass filters alter the phases of the different frequency components in the signal depending on the rate (Hz), the depth (ms) and the shift (ms) of the LFO and the feedback coefficient a (0-1).

The number of all-pass filters determines the number of notches affecting the sound. This example uses 16 all-pass filters resulting in 8 notches. Other implementations can use much more filters (or less) and the output can be fed back to the input.

The phaser is a famous guitar effect that can be found in several music like "Another Brick In The Wall (Part 3)" from the Pink Floyd or more recently in "Paranoid Android" from Radiohead.





ANEXO III

Equipamentos utilizados

- Processador Intel Core I5-7400 Kaby Lake LGA 1151 3.0Ghz 6MB Cache, BX80677I57400;
- Placa-Mãe Asus H110M-CS/BR Intel 1151 DDR4 Micro ATX;
- Memória Kingston DDR4 Hyperx Savage 8GB (1X8), 2133MHZ, HX421C13SB/8;
- Memória HyperX Fury, 8GB, 2400MHz, DDR4, CL15, Preto - HX424C15FB3/8;
- Placa de Vídeo Gigabyte AMD Radeon RX 590 Gaming 8G, GDDR5 - GV-RX590GAMING-8GD (rev. 2.0);
- HD Toshiba 1TB Sata III 3.5 7200RPM, HDWD110XZSTA;
- FONTE CORSAIR CX SERIES CX550 80 PLUS BRONZE 550W PFC ATIVO, CP-9020121;
- SSD Kingstone 240gb;
- Gabinete Gamer Sharkoon TG4 Red;
- Headphone M20X Audio-Technica;
- Smartphone Asus Zenfone 3 Max 5,5’;
- Cabo micro USB;
- Microfone condensador BM800;