Desenvolvimento de áudio para jogos com Unity e FMOD

Luis Henrique Marinho Alves* Jucimar Maia Silva Junior Cristina Souza de Araújo

Universidade do Estado do Amazonas, Escola Superior de Tecnologia, Brasil

RESUMO

Para trabalhar como músico na área de jogos digitais não basta apenas saber tocar um instrumento e escrever melodias, é um trabalho que requer um conhecimento amplo de música e lógica de programação. Com o intuito de despertar o interesse das pessoas à área de desenvolvimento de músicas/áudio para jogos, serão demonstradas técnicas e ferramentas utilizadas para criação de trilhas e efeitos sonoros para jogos, explorando as funcionalidades do FMOD em conjunto com o Unity.

Palavras-chave: *Game audio*, fmod, unity.

1 HISTÓRIA DOS SONS EM JOGOS

Antigamente, para se ter músicas e efeitos sonoros em um jogo era preciso ter um compositor sentado por horas com algum programador, cada som ou nota tinha de ser estrategicamente colocado para economizar o processamento do console, e dependendo das limitações do chip de áudio de cada um, deveria se ter o cuidado com o quanto de áudio poderia ser executado simultaneamente.

Pegando como exemplo um Nintendo Game Boy [10], suas limitações permitiam apenas um total de 4 sons tocados ao mesmo tempo, sendo uma real dificuldade até mesmo para os compositores mais experientes. [9]

2 COMPOR PARA FILMES X COMPOR PARA JOGOS

Quando se compõe para um filme, animação ou programa de televisão, o compositor recebe a cena que irá adicionar a sonoplastia e então pode ver e rever a cena inúmeras vezes, podendo adicionar detalhes, criar toda emoção e acompanhar o desenvolvimento da cena até chegar a perfeição. Uma vez finalizado, a cena será reproduzida da mesma forma independente da pessoa que esteja assistindo o filme, logo o áudio estará sempre em perfeita sincronia com as imagens.

Ao se trabalhar com jogos, a forma de pensar precisa ser outra, pois a mesma pessoa pode jogar a mesma parte de um determinado jogo inúmeras vezes, e a execução dele pode ser diferente em toda elas, obrigando a música se adaptar a cada ação feita pelo jogador.

3 TIPOS DE SONS EM JOGOS

Ao desenvolver sons para jogos, o *Sound Designer* (Projetista de som) vai precisar trabalhar com vários tipos de sons, não apenas músicas, pois um jogo é composto de grupos de sons, contendo ambiências, músicas, falas, efeitos sonoros, botões, entre muitos outros, mas é possível separá-los em categorias

*e-mail: lhma.eng@uea.edu.br

3.1 Músicas

A música de um jogo é tudo o que está contido em sua trilha sonora, desde músicas de menus, temas de locais e personagens até sutis músicas de fundo, podendo ter horas de duração ou apenas alguns segundos.

3.2 Efeitos Sonoros (SFX)

Um jogo é repleto de efeitos sonoros. Esses efeitos são o que fazem o jogador sentir-se dentro do jogo, criando o real ambiente onde se passa. Pode-se citar efeitos sonoros como o fundo do ambiente, como sons de florestas, de cidades, até outro planeta, como os sons de tiros e armas, o som da passada do personagem ao caminhar ou correr, o som de quando o personagem é atingido ou atingindo um inimigo. A quantidade de efeitos sonoros de um jogo depende da necessidade do cliente. Efeitos sonoros podem ser feitos de várias maneiras, podendo ser gravados ou produzidos.

3.2.1 Gravados

Uma forma de confecção de SFX que ficou bastante conhecido por conta dos cinemas e rádios, mas que hoje também é utilizado nos jogos, é o *Foley*. Esse tipo de efeito sonoro é utilizado para criar sons do cotidiano no ambiente presente no jogo, como os passos, o respirar, o som de uma cadeira sendo arrastada etc. São tantos que vão até o limite da criatividade do artista, assim tentando cobrir basicamente todos os sons que existem (sons orgânicos).

Outra forma de confecção de efeitos sonoros é a utilização de efeitos em sons já gravados aplicando efeitos, assim podendo diminuir a velocidade de execução do som, ou acelerar ele, entre inúmeros efeitos, como *Chorus*, *Flanger*, *Reverse*, *Noise Gate*, distorção, para assim adaptar ao que está procurando, como utilizar uma voz gravada e então diminuir a sua velocidade e seu tom e então adicionando um pouco de distorção, para assim criar uma voz de um monstro.

Para gravar efeitos sonoros por conta própria é necessário conhecimento sobre microfones, acústica, reverberação entre outros fatores, tudo para poder obter o som da melhor forma. Um fator muito importante para uma boa gravação é saber qual microfone usar, pois é necessária saber qual polaridade de captura o microfone possui, assim podendo focar o som sobre um ponto pequeno ou gravando uma vasta área.

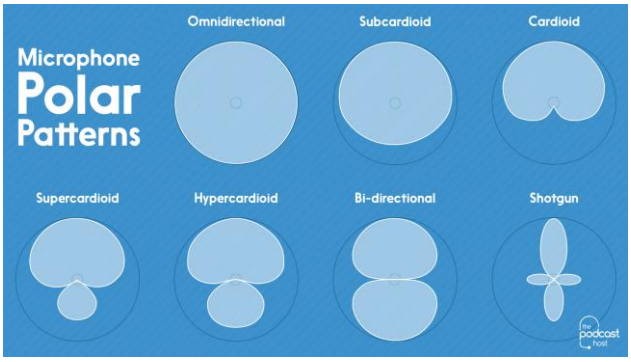

Figura 1: Padrões de polaridade de microfones [13]

A figura 1 demonstra cinco padrões de polaridades de microfones, o tipo omnidirecional, que possui uma captação para todos os lados como o nome diz, sendo a melhor escolha para captura de gravações de ambientes, seguindo o cardióide, com a sua captura em formato de coração, assim eliminando ruídos da parte de trás do microfone bom para a captação de vozes em um ambiente controlado. O microfone cardióide possui duas variações, o supercardióide e o hipercardióide, onde o super possui uma captação mais fechada que o cardióide simples, eliminando assim mais ruídos que o simples poderia captar em sua parte de trás. Já o hipercardióide é o tipo de microfone mais fechado, também conhecido como *shotgun*, esse microfone é muito aconselhado para trabalhos externos, onde há bastante barulho indesejado e queira focar em um único som. O último padrão é o bidirecional, com um formato de captura que rejeita unicamente os sons laterais, capturando o que acontece na frente e atrás do microfone, ideal para uma captura de conversação entre duas pessoas frente a frente.

3.2.2 Sintetizadores

Além dos sons orgânicos, também existem o caminho dos sons digitais, sons que são criados com o auxílio, muitas vezes, de sintetizadores, que é um instrumento musical que manipula correntes elétricas, assim sendo de forma analógica, ou simula esse feito de forma digital. Esse tipo de efeito é bastante utilizado na criação de sons para gêneros futuristas, como o som de um disparo de um *Laser* em um jogo de ficção.

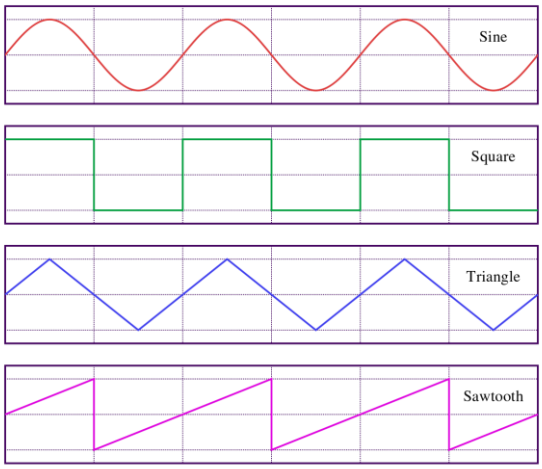

Figura 2: Formatos de ondas [18]

Sintetizadores podem modificar a corrente elétrica para gerar formatos de ondas diferentes, como apresentados na figura 2, existem as ondas senoidais, quadradas, triangulares e serrilhadas, além de variações das mesmas e uniões delas, cada uma com suas peculiaridades e timbres.

3.2.3 Bibliotecas/Bancos de Áudio

Muitas vezes um *Sound Designer* não consegue gravar ou produzir todos os sons que lhe é necessário, mas não porque ele não quis gravar, mas sim pois não possuía o que era preciso para obter o som que procurava, como por exemplo, para um jogo de aventura, seria solicitado a equipe de sonoplastia um som de um trem em movimento, mas no local onde eles moram não há nenhum trem por perto, ou então lhe pedem um som específico de um animal que mora no meio de uma floresta, e a equipe precisaria se deslocar até o meio da floresta para obter apenas um som, um gasto que poderia ser evitado.

Para trabalhos muito complexos ou de um grau de dificuldade de obtenção alto, existem pessoas que o trabalho é apenas de gravar esses sons, e assim criar uma biblioteca de áudios com um material podendo ser vasto ou bem específico, e então vendem o acesso a esse material.

Empresas como Sound Ideas [16] fornecem vastos estilos de bibliotecas, de pequenas à enormes, podendo chegar até a mais de dois *terabytes* de arquivos de áudio, e os preços variam de acordo com a quantidade, qualidade e até grau de dificuldade de obtenção dos áudios.

Algumas produtoras de jogos grandes possuem bancos de áudios próprios, isso facilitando o trabalho de *Sound Designers*, podendo trabalhar em projetos sem perder as características que a empresa deseja. [1]

3.3 Voice-Over (VO)

O *Voice Over*, ou em português, a "voz sobreposta" pode ser chamada de dublagem, é uma parte que possui grande importância em alguns jogos, porém, em outros, nem está presente.

O trabalho de dublagem em um jogo envolve uma parte contratual elaborada por conter o trabalho de outras pessoas por fora da equipe de *Sound Designer*, os dubladores

3.4 User Interface (UI)

O som de UI é o tipo de som presente nas partes que o jogador interage com a interface do jogo, podendo não intervir com a jogabilidade do jogo, isso inclui sons como os de menus e notificações.

3.4.1 Diegético

O termo diegético vem de diegese, um conceito utilizado em artes como cinemas para dizer respeito da dimensão que a narrativa toma em questão a ficção. Algo diegético vem dizer que está presente no ambiente onde o jogo é tratado.

É possível observar em jogos de franquias como Dead Space [5] e Metro [12] a utilização de menus diegéticos, como no caso da barra de vida do personagem principal em Dead Space, para não utilizar uma barra externa ao jogo, como em um menu de HUD (*Heads-Up Display* , em português, tela de alerta), foi utilizada a armadura do personagem principal como uma barra de vida, da mesma forma que quando se acessa o menu é como se o personagem principal estivesse olhando para o menu junto do jogador, dando maior imersão.

Figura 3: User Interface no jogo Metro 2033 [8]

A figura 3 apresenta um relógio do jogo Metro 2033 [12], da empresa 4A Games, esse jogo apresenta menus inteiramente diegéticos, onde o jogador se sente mais imerso no jogo, o relógio funciona para mostrar a quantidade de tempo que o personagem aguenta ainda antes que a sua máscara de gás pare de funcionar.

3.1.2 Não-diegético

Já que o diegético é o que está imerso no jogo, o não-diegético é justamente o oposto, assim menus que se sobressaem na tela jogo são o exemplo perfeito, como nos jogos da franquia The Legend of Zelda [21] ou Resident Evil [15], observa-se que ao acessar o inventário é como se o jogo parasse e o menu estivesse por fora dele.

Essa forma de trabalho com áudios diegéticos e não-diegéticos se diferem na parte de sonoplastia pois ao trabalhar com um menu que está sobressaído do jogo esse mesmo pode ter sons fora do contexto presente no jogo, pois não está habitado no mesmo local que o personagem está, assim usar sons simples de botões como cliques ou *beeps*. Já no diegético os sons precisam ser tratados como parte da jogabilidade, onde o personagem está, logo utilizasse sons presentes no menu ou no ambiente do jogo são ideais para esse trabalho.

4 BASE TEÓRICA MUSICAL

Para se trabalhar na área de *Sound Designer* é muito questionada a necessidade de ser músico no ramo, ou se é preciso saber compor para trabalhar com áudio em jogos digitais. Ela não é composta apenas por músicos, ela apresenta ramos de dublagem, engenharia de áudio, composição, masterização, artistas de *foley* entre outros. Logo para se trabalhar como *Sound Design* não é uma obrigação o conhecimento profundo de composição, porém, ele ajuda a abranger os ramos possíveis do trabalho.

Para se compor para um jogo, dependendo do jogo, o trabalho musical pode ser bem simples, composto apenas por pequenas obras simples, para isso o conhecimento musical pode ser mais superficial.

A música tradicional, ou melhor definindo, a música ocidental, apresenta um total de 12 notas musicais, sendo essas, 7 notas concretas e 5 acidentes. Essas notas concretas são as notas que compõem a escala de Dó maior, sendo elas Dó, Ré, Mi, Fá, Sol, Lá e o Si, nomenclatura que também pode ser definida pelas primeiras 7 letras do alfabeto, sendo assim respectivamente, as notas C, D, E, F, G, A, B. Já os acidentes são notas presentes entre as notas concretas, que são chamadas de sustenido (#), que representa um semitom acima, e bemol (b), um semitom abaixo, um exemplo seria a nota Dó sustenido (C#), nota essa presente entre o Dó (C) e o Ré (D). Todas as notas concretas possuem acidentes entre si, com exceção entre as notas Mi (E) e Fá (F) e as notas Si (B) e Dó (C).

A distância entre duas notas é conhecida pelo nome de intervalo, esse intervalo podendo ser chamado de tom ou semitom, sendo assim, a distância de um tom igual a distância de dois semitons, logo o termo semitom podendo ser denominado também como meio tom.

Os componentes básicos de uma música são: Melodia, harmonia e ritmo, esses cada um tendo o seu papel específico e fundamental na confecção de uma música.

4.1 Melodia

Músicas, em sua grande maioria, possuem um tema principal, a qual fica na cabeça do ouvinte, tema o qual a pessoa pode cantar depois, e lembrar com facilidade. Esse tema pode ser chamar de melodia. A melodia, ou motivo, é a parte "solo" de uma composição, onde é feita por notas tocadas de forma a se sobressair da demais notas da música.

Temas presentes como no da música principal do Super Mario World [19] são exemplos que representam a importância da melodia. Nessa música existem mais sons além da melodia principal, porém as notas que se sobressaem são as que formam a melodia.

4.1.1 Escalas

Uma boa forma de começar a desenvolver uma melodia é pensar na formatação de escalas. Ao trabalhar com escalas fica mais fácil de começar a pensar nas intenções da sua composição, assim partindo para algo mais triste ou mais alegre.

Para enumerar todas as escalas existentes seria necessário um trabalho específico para este tópico, pois sua quantidade é enorme, porém é possível separar duas das escalas que se destacam, a escala maior e a escala menor.

C maior	C	D	E	F	G	A	B	C
G maior	G	A	B	C	D	E	F#	G
D maior	D	E	F#	G	A	B	C#	D
A maior	A	B	C#	D	E	F#	G#	A
E maior	E	F#	G#	A	B	C#	D#	E
B maior	B	C#	D#	E	F#	G#	A#	B
F# maior	F#	G#	A#	B	C#	D#	F	F#
C# maior	C#	D#	F	F#	G#	A#	C	C#
F maior	F	G	A	Bb	C	D	E	F
Bb maior	Bb	C	D	Eb	F	G	A	Bb
Eb maior	Eb	F	G	Ab	Bb	C	D	Eb
Ab maior	Ab	Bb	C	Db	Eb	F	G	Ab
Db maior	Db	Eb	F	Gb	Ab	Bb	C	Db
Gb maior	Gb	Ab	Bb	B	Db	Eb	F	Gb

Figura 4: Tabela de Escalas Maiores

As escalas são formadas por conjuntos de notas determinadas por uma sucessão de intervalos. Como a escala maior, que é

determinada pela ordem de intervalos Tom-Tom-Semitom-Tom-Tom-Tom-Semitom, assim podendo ser aplicada como um *template* sobre qualquer nota e então resultando na sua escala. Por exemplo, a escala de Ré maior (D), iniciando do Ré (D) e então seguindo o *template* se avança um tom, chegando a nota Mi (E). Ao continuar a avançar mais um tom, chegando a nota Fá sustenido (F#). Assim seguindo o *template*, somando um total de 7 notas no final, onde a nota Ré se repete, sendo a oitava nota. Por isso outra notação, a notação de Oitava, onde você tem uma mesma nota estando a uma distância de uma escala entre si, podendo ter um Ré mais grave e outro mais agudo. A escala resultante seria formada pelas notas D-E-F#-G-A-B-C#, e esse mesmo processo gera todas as escalas maiores, como apresentado na figura 4.

A escala menor é trabalhada da mesma forma, porém utilizando de outro *template* de intervalos, dessa vez sendo o formato Tom-Semitom-Tom-Tom-Semitom-Tom-Tom. Essa escala é conhecida por apresentar um ar mais triste e melancólico para a composição. Trabalhando dessa forma, para obter a escala de Dó menor (Cm) começaria pelo C e seguiria por todo o *template*, resultando na escala com as notas C-D-Eb-F-G-Ab-Bb.

I	II	III	IV	V	VI	VII
C	Dm	Em	F	G	Am	B^{ø}
G	Am	Bm	C	D	Em	F#^{ø}
D	Em	F#m	G	A	Bm	C#^{ø}
A	Bm	C#m	D	E	F#m	G#^{ø}
E	F#m	G#m	A	B	C#m	D#^{ø}
B	C#m	D#m	E	F#	G#m	A#^{ø}
F	Gm	Am	Bb	C	Dm	E^{ø}
F#	G#m	A#m	B	C#	D#m	F^{ø}
Bb	Cm	Dm	Eb	F	Gm	A^{ø}
Eb	Fm	Gm	Ab	Bb	Cm	D^{ø}
Ab	Bbm	Cm	Db	Eb	Fm	G^{ø}
Db	Ebm	Fm	Gb	Ab	Bbm	C^{ø}

Figura 5: Tabela de Campos Harmônicos Maiores

4.2 Harmonia

Além da melodia, a música precisa de uma base, uma sustentação para a melodia, essa parte é conhecida como a harmonia, ela sendo formada por acordes. Esses acordes são utilizados para criar a fundação da música, onde a melodia poderá ser executada por cima sem nenhum conflito.

A harmonia musical tem como a sua principal teoria a utilização de Campos harmônicos, oriundos de uma escala musical. Pegando como exemplo a criação do campo harmônico de Dó maior, começa-se com a escala de Dó maior, seguindo o *template* de escala maior, resultará em C-D-E-F-G-A-B. Para criar um acorde é preciso ter, pelo menos, 3 notas, e para formar os acordes simples fundamentais de um campo harmônico essas notas são adquiridas selecionando uma delas na escala e então pulando uma nota e pegando a próxima, e repetindo esse processo para ter a terceira nota. Trabalhando com a escala de Dó maior terá então a primeira nota Dó (C), então pula-se a segunda Ré (D) e usa-se a terceira, que é a nota Mi (E), repetindo esse processo a partir da nota Mi (E) terá a nota que falta para o acorde, então, pula-se a nota Fá (F), chegando a nota Sol (G). Unindo as 3 notas, Dó (C), Mi (E) e Sol (G), o resultado será então o acorde de Dó maior. Esse mesmo processo pode ser repetido para gerar todos os acordes do campo harmônico dependendo de onde se parte, como na figura 5.

O campo harmônico então oferece ao compositor uma gama de acordes que lhe é garantido que poderá usar em sua música e tocar a escala que o originou sem ter muitos problemas de conflito.

4.3 Ritmo

O ritmo determina o andamento da música, sendo assim, a velocidade que a música possui, se ela é rápida ou lenta, normalmente determinado pela sigla *Bpm* (Batidas Por Minuto), assim colocando em sua composição a escrita "*Bpm*=80" ou por nomes em italiano que abrange uma região de *Bpms* onde o compositor sente que a música está confortável e pode sofrer leves alterações, como por exemplo, o termo *Andante Moderato*, a qual abrange a região dos *Bpms* de 90 a 100, dependendo do que o maestro reger.

5 Como se preparar para compor para jogos

Um compositor de jogos poucas vezes irá compor apenas para um único tipo de jogo, pois são inúmeros, assim como quem trabalha com filmes. Então é preciso conhecer um pouco de tudo para estar preparado completamente para trabalhar na área. É preciso ouvir as vezes mais do que o seu gosto lhe permite para ficar inteirado, pesquisar sobre instrumentos diferentes, conhecer os instrumentos para o qual vai compor, basicamente como um orquestrador trabalha.

Em uma orquestra, o compositor precisa conhecer perfeitamente cada instrumento para qual vai escrever, isso tendo em vista o timbre do instrumento, seu alcance de notas, e suas limitações, além de conhecer bastante teoria musical, para então construir o ambiente desejado, saber os acordes mais alegres, tristes, suspense, liberdade, claros, escuros, criar melodias, *leitmotifs* (frases curtas que se associam a personagens ou atributos que estão presentes na ideia da música), dentre muitas outras técnicas.

Para fazer um trabalho com jogos, o conhecimento não se diferencia do trabalho de um compositor para uma orquestra. Até porque, dependendo da demanda do cliente, um jogo pode

requerer uma trilha composta para uma orquestra com vários instrumentos. O que ocorre normalmente é que o compositor de jogos muitas vezes segue um estilo de composição, focando no que conhece mais.

Podendo ir de músicas eletrônicas a instrumentos reais, o campo que um compositor pode seguir varia de acordo com o que mais se identifica, mas isso podendo se adaptar ao estilo que o cliente deseja, pois, por exemplo, uma trilha de um jogo de terror pode ser feita tanto com instrumentos reais como por sintetizadores.

Porém, o grande fator que diferencia o compositor de jogos do resto é saber como funciona a música dentro de um jogo e o que é possível fazer com ela no decorrer do jogo.

6 INSERINDO ÁUDIOS EM JOGOS

O trabalho de adicionar as músicas a um jogo é feito, muitas vezes, através de linhas de código no Software utilizado para a confecção do jogo. Tratando-se do Unity [22], a programação pode ser feita em duas linguagens, C# e JavaScript. Essa parte de programação pode ser feita justamente por algum programador, não sendo necessariamente um trabalho para o compositor. [25]

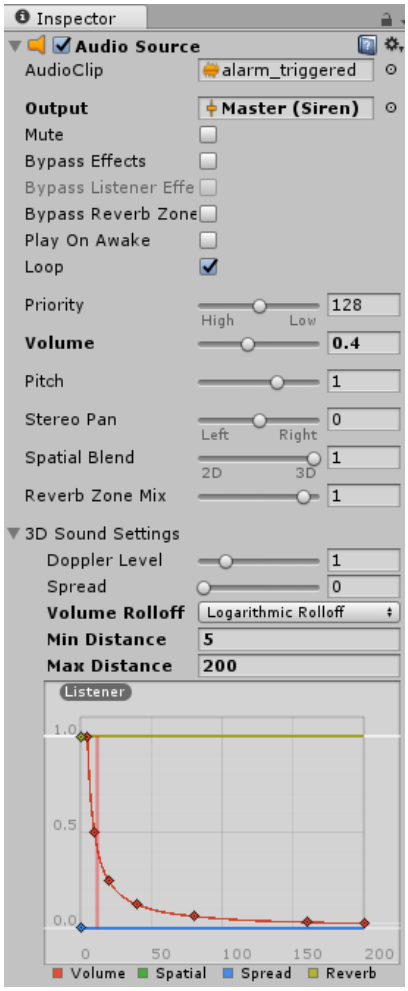

Figura 6: Barra Inspector de um *Audio Source*

O áudio no Unity possui duas ferramentas para serem utilizadas, o *Audio Source* (fonte de áudio) e o *Audio Listener* (ouvidor de áudio). Como pode-se observar na figura 6.

O *Audio Source* oferece várias funções para o seu áudio, desde funções simples como volume a mais elaboradas como trabalhar com áudio 3D. Essas funções são também variáveis que podem ser alteradas por *scripts*, assim podendo acionar quando quiser, por exemplo, a variável *Mute*, capaz de silenciar o Audio Source específico diretamente pelo código-fonte do jogo.

Outra forma de silenciar um *Audio Source* seria utilizar a variável Volume em vez de usar apenas a variável *Mute*, pois ao usar o *Mute* ou som só pode estar ou ligado ou desligado, devido ser uma variável booleana (aceita apenas valores binários), já ao utilizar o Volume é possível alterar o volume gradualmente, trabalhando com *Fades*, pois, ao invés de pular do valor 1 para o 0, essa variável pode ir recebendo atualizações e ir adquirindo valores entre os dois números, assim descendo gradualmente entre os valores 0.9, 0.8, até chegar a 0.

Ao utilizar a linguagem C# para recriar o exemplo citado no parágrafo anterior é o uso da função *Mathf.Lerp(a, b, t)* para atribuir o valor de volume para um *Audio Clip*, onde pode-se fazer uma transição gradual, onde *a* é o valor inicial, *b* o final, e o *t* o valor interpolar entre os outros dois valores. Um bom exemplo seria:

```
Music.volume = Mathf.Lerp(min, max, 0.5f *
        Time.deltaTime) [24]
```

Onde *min* e *max* são duas variáveis do tipo *float* e terá uma transição gradual de valores do *min* para o *max* baseado no valor de *0.5f * Time.deltaTime* (Relógio interno do computador).

6.1 Configurando Áudio 3D

Ao trabalhar com áudios em 3D é preciso ter cuidado com a curva que o volume (*Rolloff*) do Audio Source fará de acordo com o posicionamento do Audio Listener. O Unity já oferece para trabalho duas formas de *Rolloff*, o logarítmico, presente na figura 6, e o linear, como na figura 7.

O áudio no mundo real se dispersa no ambiente de forma logarítmica. Isso quer dizer que para recriar um som ambiente que simule o planeta Terra precisaria ser trabalhado com um *Rolloff* logarítmico.

O cuidado de utilizar o *Rolloff* linear é o de saber o tipo de som que está sendo executado no jogo. *Rolloff* lineares são utilizados principalmente em sons executados fora do ambiente, onde o personagem, ou o *Audio Listener*, se localiza, como músicas, notificações ou diálogos fora do ambiente (como conversas por rádio ou telefone).

O *Rolloff* apresenta duas variáveis de distância, o *Min Distance* e o *Max Distance*, essas variáveis funcionam para criar um campo que representa o alcance do som de um *Audio Source*, assim diminuindo o volume gradualmente, acompanhando o formato de *Rolloff* selecionado.

Ao se trabalhar com uma ferramenta de *middleware*, o único trabalho do programador será colocar o *Audio Listener* e o *Áudio Source* que o software possui no local onde se precisa dele, já toda a parte de lógica de execução do som ficará por parte do *Sound Designer*.

No caso do FMOD [7], no próprio site da empresa encontra-se para baixar um arquivo de integração para as engines Unity e Unreal. No caso do Unity, é um arquivo *.unitypackage* que se importa dentro do projeto no Unity e então apresentará uma nova aba na barra de ferramenta contendo configurações do FMOD então só é necessário selecionar qual o arquivo de FMOD que desejas fazer a integração ou seu projeto.

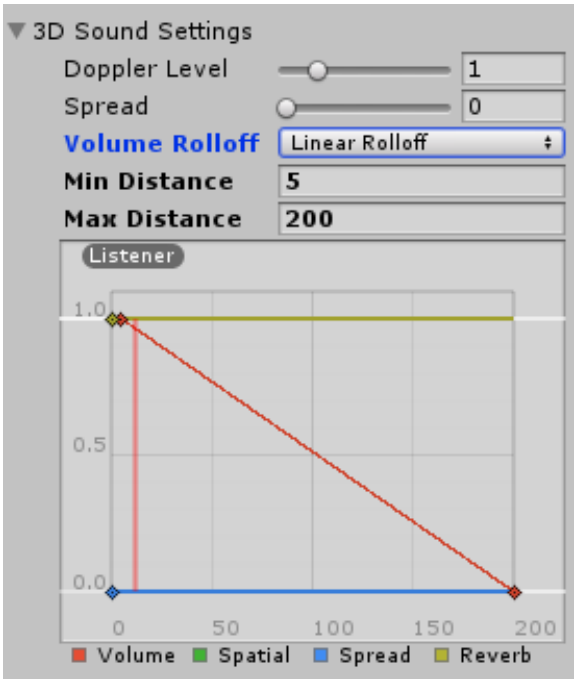

Figura 7: Exemplo de *Rolloff* linear)

Ao trabalhar no FMOD será necessário exportar o arquivo *GUID (função localizada na barra de tarefas, na opção File)* . Também, assim como se faz no Unity para um projeto, é preciso fazer a Build do seu trabalho para que ele possa ser inserido ou atualizado no projeto do Unity.

7 O QUE SÃO MIDDLEWARES E PARA O QUE SERVEM

Como que um *Sound Designer* poderia ter seus sons executados como ele gostaria se não é necessário para ele ficar junto de um programador? Tendo de ficar ajustando cada detalhe no código do jogo?

Então entra o papel dos *Middlewares* de áudio, onde o programador e o *Sound Designer* podem trabalhar separadamente e ter os seus sons tocados da forma planejada.

Figura 8: Interface do Middleware FMOD

O *Middleware* é uma ferramenta presente entre a programação e a composição de músicas para um jogo, onde o músico não precisaria se preocupar mais com linhas de códigos, deixando esse papel para o programador. O *Sound Designer* a partir de agora irá focar apenas na parte de organizar a lógica de execução dos seus sons.

Middlewares como FMOD (Figura 8), Wwise [25] e o Fabric [20] fornecem ao *Sound Designer* ferramentas que permitem toda a organização de como o áudio funcionará, construindo *loops*, transições, adicionar efeitos ajustáveis, propriedades do áudio, além de muitas outras formas de trabalhar com o som de um jogo.

8 UTILIZANDO PARÂMETROS NO FMOD

A funcionalidade que pode ser considerada uma das mais interessantes de trabalhar no FMOD pode ser a função de parâmetros, onde o *Sound Designer* pode criar automatizações para o seu projeto que podem ser controladas por variáveis dentro do jogo. Essa função que cria realmente a sincronia entre o programador e a equipe de áudio.

Quando a equipe está junta organizando o desenvolvimento do jogo, o *Sound Designer* pode entrar em conjunto com os programadores e criarem variáveis que podem pegar valores, em tempo real, do jogo e atribuir a parâmetros no FMOD.

As utilidades dos parâmetros podem criar automatizações em qualquer sentido, tanto para efeitos sonoros quanto para a própria trilha do jogo.

Um exemplo para efeitos sonoros seria a utilização de parâmetros para um jogo de corrida, onde o carro acelera e, consequentemente, precisa que o som do motor acompanha o desempenho do carro em tempo real, tanto com a velocidade do carro aumentando, quanto diminuindo. Para isso, é possível criar um parâmetro "Velocidade", e então automatizar qual som será tocado para cada velocidade e qualquer outra variável que possa ser de importância para a adaptação do efeito, como *pitch* ou volume.

Figura 9: Exemplo de parâmetro para som de passadas

A figura 9 apresenta um exemplo de parâmetro utilizado para criar os passos de um personagem de forma dinâmica, onde, dependendo da velocidade dele. Para trabalhar desta forma, primeiro será preciso criar um *loop* que funcionará infinitamente na sua *Timeline*, então, ao lado da aba escrito *Timeline* tem uma aba com um "+", essa aba vai adicionar um parâmetro, onde pode-se determinar um nome e seus valores mínimos e máximos, no caso apresentado, chamado de "*Speed* e os valores entre 0 e 1000. Após criar o parâmetro será preciso apertar com o botão direito do mouse sobre o *knob* de volume na trilha de áudio que deseja automatizar, e então apertar em "*Add Automation*". Agora está criada a sua trilha de automatização de volume, onde pode mudar o que será tocado de acordo com o valor atribuído ao parâmetro. No exemplo foram gravados 4 *takes* de *loops* passos, cada um respectivamente com o som de passos mais rápidos e fortes que o outro, assim, dependendo do valor, além de aumentar o volume, também altera qual *loop* de passos será tocado.

No que se refere a parte musical do jogo, os parâmetros também podem ser explorados, de inúmeras maneiras. Um exemplo pode ser um jogo que a música se adapte a quantidade de vida do personagem, supondo que a vida estivesse abaixo de 15%, a música sofresse uma alteração, enfatizando o estado de vida do personagem sem que o jogador tenha de ficar olhando para a barra de vida.

A inserção de um evento e todos os seus parâmetros é feita por linhas de códigos, neste caso em C#, primeiro declara o evento do FMOD para o Unity, então inicializando uma variável do tipo *string* usando um valor do caminho até o evento desejado.

```
[FMODUnity.EventRef]
 public string music = "event:/Music";
 FMOD.Studio.EventInstance musicEv;
 FMOD.Studio.ParameterInstance musicChangeParam;
```

Figure 10: Implementação de um evento do FMOD em C#

Nesse exemplo, visto na figura 10, é apresentado um evento que possui uma música e um parâmetro, uma variável possui o nome de "*music*", e essa variável recebe o caminho para chegar até o evento que deseja ser trabalhado, no exemplo "*event:/Music*", para especificar a pasta a qual ele está presente. Após a variável que determina o local onde o evento está presente é então instanciado o evento em si em uma variável, no exemplo com o nome de *musicEv* do tipo "FMOD.Studio.EventInstance". A variável de parâmetro, essa que vai atribuir valores ao evento em si, no caso, um parâmetro "*Change*", que foi implementado no código como a variável "*musicChamParam*" do tipo "*FMOD.Studio.ParameterInstance*".

Para iniciar o som é preciso atribuir esses valores de forma coerente para que possam funcionar. Assim trabalhando três variáveis: uma contendo o evento (*musicEv*), outra com o destino do evento (*music*) e outra com o parâmetro que será trabalhado (*musicChangeParam*). Então começa-se atribuindo o destino ao evento em si, nesse caso ficaria "*musicEv*", seguido de um símbolo de "=", para dizer que receberá o valor a seguir, e a função "*FMODUnity.RuntimeManager.CreateInstance(x)*", que irá criar a instância do destino do evento à variável desejada, nesse caso ficaria "*musicEv = FMODUnity.RuntimeManager.CreateInstance(music)*".

Para o parâmetro é utilizada a função "*getParameter(a, out b)*", onde *a* é o nome da *string* do parâmetro e *b* o nome do parâmetro instanciado, no caso a cima ficaria "*musicEv.getParameter(change, out musicChangeParameter)*", dessa forma o parâmetro Change, dentro do evento musicEv, seria atualizado para o valor que a variável *musicChangeParameter* possuir, esse valor sendo alterado pela função "*setValor*()" de forma simples, "*musicChangeParameter.setValor(1)*". Após tudo atribuído de forma correta, o evento pode ser inicializado utilizando a função "*start*()", ficando "*musicEv.start*()".

A quantidade de parâmetros que o *Sound Designer* irá utilizar vai de sua própria criatividade, pois não há um limite de parâmetros ou efeitos. É possível que mais de um parâmetro possa fazer alterações em um mesmo áudio de forma diferente. Por exemplo, um jogo que o jogador precise fazer um transporte de caminhão uma carga pesada de um local para o outro, pode-se então utilizar dois parâmetros, um "Peso" e um "Velocidade", para, ao se ter mais peso, deixar o som do motor mais robusto e com a sensação de mais esforço, e usar a variável velocidade para alterar o *pitch* do som, deixando ele mais agudo. Dessa forma, os dois parâmetros poderiam trabalhar juntos para criar a sensação de peso e esforço, pois um caminhão sem peso faz um som mais agudo em grande velocidade, e ao acrescentar peso, o som do motor iria baixar, e então uma variável compensaria o outro, tornando o som mais realista.

9 TÉCNICAS DE COMPOSIÇÕES PARA JOGOS

Um dos grandes problemas enfrentados quando se trabalha com sons para jogos é a fatiga sonora do jogador, e para evitar isso é feito um trabalho conhecido como "Música adaptável" ou "Música interativa".

Como dito por Aaron Makrs no livro The Complete Guide to Game Audio, "*Jogadores não são previsíveis. Nós não sabemos quando eles vão andar, correr, esconder, entrar em um novo quarto, encontrar o vilão, sacar suas armas, ou fazer qualquer outra centena de possíveis ações que podem ocorrer durante o jogo. Assegurando que a música possa transitar naturalmente é o que faz isso funcionar.*". [1]

Quando se compõe de forma adaptável é possível criar ambientes que se desenvolvam de acordo com as ações do jogador, isso quer dizer, a música pode se adaptar a tudo o que está acontecendo no jogo, caso tenha um acontecimento mais sentimental a música poder intensificar seu arranjo para acompanhar o clímax do jogo.

Com o auxílio dessa forma de composição, as trilhas de jogos podem muito bem serem comparadas a trilhas de filmes, que acontecem de acordo com cada ação presente na cena que está acontecendo.

Existem várias técnicas e variações das mesmas para se compor de forma adaptável, mas é possível destacar duas dessas para serem enfatizadas, a composição horizontal (*Cross-*) e a vertical.

9.1 Composição Horizontal

A composição horizontal, ou *Cross-fade*, é uma técnica de composição que consiste em criar dois *loops* de que possam ser intercambiáveis entre si, possibilitando que a qualquer momento possa ser feito a passagem de um *loop* para outro.

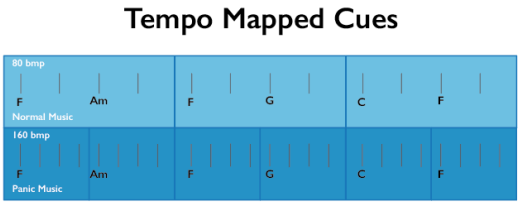

Figura 11: Representação de *loops* para *Cross-fade* [4]

Na figura 11 pode-se observar um exemplo representado que possui trilhas com *bpms* diferentes mas que ainda assim são intercambiáveis por fecharem seus *loops* juntos. Uma boa forma de pensar é criar duas músicas, uma com o dobro do *bpm* da outra, para assim terem a mesma duração, mas uma mais "agitada" do que a outra. Essa forma de trabalhar é boa para quando você tem um jogo com ambientes diferentes sem telas de carregar entre si, mas que possuam trilhas diferentes para cada ambiente.

Figura 12: *Timeline* em um *Cross-fade*

Ela funciona com os dois (ou mais) *loops*. No caso da figura 12, há apenas duas trilhas de *loops* sendo tocados ao mesmo tempo, mas apenas um tendo seu volume ativo. Ao ser acionado por um evento seria então feita a passagem para a outra trilha de forma suave, como uma mixagem de um DJ, aumentando o volume de uma e baixando o volume da outra, por isso o nome "*Cross-fade*" (*fade* significa uma transição gradual).

A figura 13 demonstra um tipo de configuração para o parâmetro *Change* trabalhar como em uma composição horizontal, gerando assim o *Cross-Fade*.

Figura 13: Parâmetro *Change* em um *Cross-fade*

9.2 Composição Vertical

A outra forma de composição é conhecida por composição vertical. Funcionando de uma forma diferente da horizontal essa técnica é muito usada para a criação de emoções ou para aumentar o clímax de um momento.

Quando se trabalha com a composição vertical o compositor precisa pensar na sua música como se fossem camadas (*Layers*). Essas camadas sendo utilizadas para acrescentar informação ou sensações à música, somando o que cada uma contém com as outras.

Supondo que esteja compondo uma trilha para um jogo de ação ou algum jogo que tenha uma batalha, quando o personagem estiver sozinho vagando pelo mapa a música de fundo não deve ser igual a música que estará tocando quando ele estiver enfrentando 50 inimigos, pois isso iria cortar toda a sensação de perigo e ação do jogador.

Então para desenvolver a música e ela acompanhar o que acontece no jogo, funcionando assim como uma composição de um filme, o *Sound Designer* cria uma série de camadas, uma para quando o personagem estiver só, outra que será somada a primeira, para quando o jogador precisar ter mais atenção, outra para quando estiver em perigo, e assim sucessivamente até chegar ao ápice da sua composição e, por exemplo, o personagem estiver em uma batalha muito intensa em situação de vida ou morte.

A figura 14 apresenta 5 camadas de intensidade e logo a cima pode ver 4 *knobs*, cada um controlando o volume de cada camada, assim sendo acionado em sequência, criando um ambiente mais versátil, acompanhando cada momento do jogo.

Outra vantagem de se compor em camadas, como na composição vertical, é, além de intensificar o clima do jogo de forma coerente e suave, poder diminuir o número de camadas e assim diminuir o que o jogador está sentindo.

No exemplo de batalha citado anteriormente foi suposto que a trilha que estaria tocando primeiramente seria feita por uma única camada, e assim acrescentando novas camadas uma a uma e aumentando a tensão do momento, mas outro exemplo poderia ser trabalhar com a trilha principal ser composta por umas três camadas, e dependendo do que acontecesse ir diminuindo essas camadas ou aumentando, tornando a trilha o mais versátil possível.

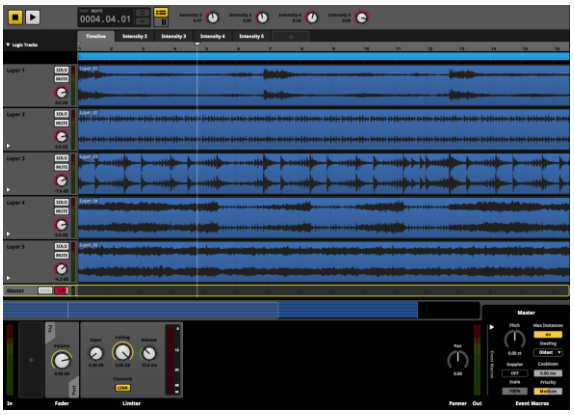

Figura 14: Composição vertical no FMOD

Pensando em um jogo de exploração, como The Legend Of Zelda [21] ou Elder Scrolls V: Skyrim [16], o jogador investiga bastante o mapa com bastante informação e ambientes diferenciados, além de interagir com outras personagens e também participar de confrontos.

Para trabalhar em um jogo como esse, a sonoplastia precisa ser bastante versátil, podendo se adaptar a tudo o que esteja presente na tela, dessa forma pode-se usar até mesmo combinações mais complexas dessas técnicas, como ter trilhas com camadas de músicas e que façam *Cross-fade* entre si, o que seria a união, respectivamente, das técnicas de composição vertical com a horizontal.

Um bom exemplo para se observar seria a exploração do jogo, onde cada cidade que o jogador passa possui uma própria trilha, sendo essa sendo dividida em 3 camadas. O número de camadas que serão executadas juntas varia de acordo com a proximidade do personagem com relação ao centro da cidade, quanto mais distante do centro, menos camadas serão tocadas, deixando o ar de vazio na trilha.

Além dessa aplicação de composição vertical, ao se distanciar da cidade ao ponto de não ter mais o tema da cidade tocando, a música faz um *fade* de volume deixando o som ambiente mais alto, e então, dependendo do que estiver próximo do personagem, acontece um *fade* que inicia uma música gerada aleatoriamente, esses *fades* funcionando de forma horizontal, nunca deixando um silêncio entre uma música ou ambiente.

10 FMOD APLICADO PARA EFEITOS SONOROS

Outra funcionalidade que torna *middlewares*, como FMOD, ferramentas muito poderosas é a sua utilização para trabalhar com efeitos sonoros. O mesmo problema que pode se ter com a

música de um jogo, a de gerar a fadiga auditiva por executar muitas repetições, pode ser ainda mais frequente na integração de efeitos sonoros.

A melhor forma de representar esse problema é pensar no som dos passos de algum personagem. O personagem de um jogo pode caminhar sobre vários terrenos, com vários calçados, aplicando forças diferentes com cada pé, mas se tiver apenas um único som de passo gravado, podendo usar apenas ele para todos esses locais e todas as variáveis possíveis, isso sério completamente desgastante para os ouvidos do jogador.

Para resolver isso pode-se trabalhar de várias formas: uma opção seria, em vez de trabalhar com um som de passo, é trabalhar com um conjunto de passos gravados em sequência, eles sincronizados com a animação do personagem. Outra forma de trabalhar seria gravar vários áudios de passos, que podem ser chamados aleatoriamente e gerar mais "naturalidade" ao som por não seguir um padrão.

Para obter melhores resultados ainda, pode-se trabalhar então com algumas funções do FMOD para ter ainda mais sons diferentes executados no nosso jogo.

10.1 Multi Sound

O FMOD possui duas opções de trilhas de áudio muito úteis para efeitos sonoros: o *Multi Sound* e o *Scatterer Sound*. Com o *Multi Sound* é possível se trabalhar com um conjunto de sons em uma mesma trilha de áudio, assim podendo chamar eles individualmente para serem executados. Uma boa utilidade para o *Multi Sound* seria para trabalhar com sons de tiros por exemplo, quando se atira com uma arma, o som do tiro nunca será igual ao outro, assim como o som de recarregar a arma, então é possível colocar duas trilhas de *Multi Sound* uma após a outra, uma contendo sons de tiros e a outra contendo sons da arma sendo carregada.

Indo para a matemática, quando se tem duas trilhas de *Multi Sound*, cada uma contendo 3 sons, é possível se ter um total de 9 variações de sons. Essa função toma proporções enormes quando feita com mais trilhas e mais sons em cada trilha.

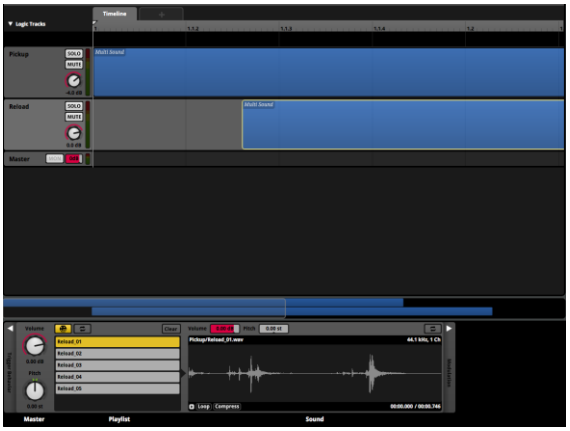

Figura 15: Componentes do Multi Sound

No exemplo da figura 15, foram utilizadas duas trilhas de *Multi Sound* para simular o efeito de um personagem pegando munição para recarregar sua arma, a primeira trilha contém 3 sons de confirmação criados com um sintetizador e a segunda trilha contém 6 sons de armas sendo recarregadas, gerando um total de 12 efeitos sonoros para a mesma ocasião, porém com suas diferenças para assim evitar a fadiga auditiva.

10.2 Scaterrer Sound

O *Scatterer Sound* que é parecido com o citado anteriormente, porém é possível se ter mais de um áudio sendo executado ao mesmo tempo. O *Scatterer Sound* possui outras variáveis novas para serem exploradas, entre elas: *Interval Between Sounds*, *Min & Max Scatter Distance*, *Polyphony*, *Total Sounds*, *Vol Rnd* (*Volume Randomizer*) e o *Pitch Rnd* (*Pitch Randomizer*).

O fator que mais diferencia o funcionamento do *Scatterer* é você poder trabalhar com valores mais amplos, podendo gerar margens para a execução de suas variáveis e explorar mais caminhos. Quando se trabalha com o *Interval Between Sounds*, como o nome diz, é o intervalo entre os sons executados, diferente do *Multi Sound*, onde o próximo áudio seria tocado exatamente após o término do áudio anterior, com o *Scatterer* é possível alterar o tempo, colocando um valor diferente da duração do áudio, e além disso, poder gerar uma margem de execução para o som, por exemplo, querer que o som seja executado aleatoriamente entre 3 e 7 segundos, fugindo mais ainda do problema de gerar padrões que fatigam o ouvinte.

Ao utilizar o *Min & Max Scatter Distance,* o *Sound Designer* pode criar uma melhor ideia de espaço e ambiente a introdução do efeito sonoro. Essa variável é capaz de alterar o volume e o *pan* dos sons tocados de forma aleatória, criando um ar de profundidade aos sons selecionados. Muito utilizado para a criação de ambientes, é possível se colocar vários sons em uma única trilha de *Scatterer* com as configurações de um ambiente.

A variável *Polyphony* representa a quantidades de sons que podem ser executados simultaneamente na trilha, isso quer dizer, quando se trabalha com o *Scatterer Sound*, é possível executar mais de um som por vez, assim colocando inúmeros sons tocando um sobre o outro, porém é preciso ser usado com cuidado, pois essa quantidade de sons precisa ser coerente com a necessidade do jogo. Um jogo habitado em uma floresta precisará de uma quantidade grande de sons, sons de animais, árvores balançando, vento, entre outros dependendo do tipo de floresta, mas todos esses sons possuem uma quantidade de sons a serem executados no máximo por vez, por exemplo, não se terá uma quantidade grande de onças rugindo ao mesmo tempo, pois onças atacam sozinhas, por conta disso que é preciso estudar o que vai ser apresentado no ambiente, para parecer o mais próximo da realidade.

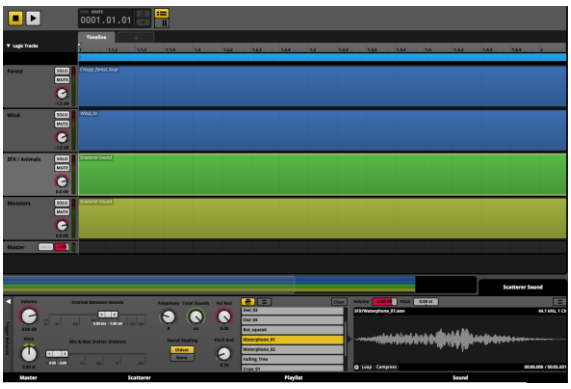

Figura 16: Interface do *Scatterer Sound*

A figura 16 apresenta a construção de uma ambiência de floresta mal-assombrada, contendo 4 trilhas, duas delas sendo *Scatterer Sound*, uma contendo efeitos sonoros e sons de animais, já a outra apenas com sons de monstros. Ambas foram feitas em trilhas separadas pois foi solicitado o som dos animais mais distantes que os sons dos monstros, então foram utilizados valores diferentes para cada trilha.

Ao observar a interface mostrada do *Scatterer Sound* apresentam-se dois botões (*knobs*), um de volume e um de *pitch*, respectivamente controlam o volume e o outro o tom do som executado. Mas o grande diferencial é poder criar uma margem de atuação para cada um deles, no Windows é *Alt*+clicar+puxar do mouse e no Mac *Option*+clicar+puxar, com essa função é possível trabalhar com valores gerados aleatoriamente dependendo da margem de erro escolhido pelo usuário.

11 LIVE UPDATE

Ao trabalhar com FMOD em um jogo, é necessário que testar todas as alterações feitas dentro do FMOD dentro do jogo de verdade. Para facilitar esse trabalho em projetos, o FMOD possui uma função que permite atualizar os projetos em outra plataforma, como o Unity, ao mesmo tempo que se é editado no próprio FMOD. A função de *Live Update* funciona como um *link* entre as duas plataformas, para configurá-la basta apertar no canto inferior direito do FMOD no botão *Live Update*, e então colocar, como o próprio programa sugere, colocar *"localhost*" ou o endereço "127.0.0.1" para conectar os dois softwares.

Depois de configurado, todas as alterações feitas no FMOD, serão atualizadas direto no Unity sem que seja preciso fazer *Build* no seu projeto, ele já estará pronto para testes ao mesmo tempo que é feito as alterações no FMOD.

12 CUIDADOS A SEREM TOMADOS COMO SOUND DESIGNER

12.1 Limitações Técnicas

O trabalho de um *Sound Designer* pode ser algo que liberte a imaginação de equipes de áudio ao seu ápice, porém esse trabalho não depende apenas do que a equipe de áudio quer, pois o jogo não é feito só de músicas.

Tendo consciência de um trabalho conjunto, a equipe de áudio precisa tomar como um fator de grande importância, mesmo nos dias atuais, as limitações que a empresa fundadora do jogo exige.

A sonoplastia pode ser uma das partes mais pesadas para um jogo dependendo do seu porte. Jogos feito para celulares não devem utilizar todo o seu processamento para trabalhar o áudio, ou ter o seu peso sobrecarregado de sons, tornando o jogo pesado demais. Esses cuidados podem ser tomados tendo um pouco de conhecimento sobre *bit rate* e conhecimento dos formatos de áudio.

O termo *bit rate* significa a quantidade ou taxa de bits que são processados por unidade de tempo, medidos em *bps* (*bits* por segundo).

Quanto maior o *bit rate* mais qualidade pode ser armazenada no áudio, uma ligação telefônica possui uma quantidade de 8kbps, enquanto um disco de Blu-ray a 1080p trabalha com uma quantidade de 40Mbps, essa sendo 5 mil vezes maior que a de uma ligação telefônica, logo tendo uma qualidade bem superior de áudio.

Formatos de áudio como MP3 trabalham com uma quantidade de *bit rate* entre 32kbps, valor aceitável apenas para reproduzir vozes, até 320kbps, a melhor qualidade disponível para se trabalhar com arquivos de áudio em formato MP3.

12.2 Compor o possível de ser executado

Uma das grandes vantagens de se compor com o computador é poder utilizar o MIDI (*Musical Instrument Digital Interface*, em português Interface Digital para Instrumentos Musicais) que pode obter informações como qual nota está sendo tocada e sua duração, e então adaptar essas variáveis a qualquer VSTI (*Virtual Studio Technology Instrument*), assim simulando uma imensa variedades de instrumentos musicais, isso sem o compositor saber tocar o instrumento, apenas selecionando o que quer que o instrumento toque.

Essa ferramenta traz uma liberdade muito grande a compositores e equipes de desenvolvimento que não possuem equipamento, ou às vezes, verba para contratar músicos que toquem instrumentos específicos que precisam para um jogo. Porém é necessário ter um conhecimento grande dos instrumentos que está tentando simular, para assim não escrever algo que seja impossível de se reproduzir por pessoas, se tiver essa intenção.

Compor trilhas para orquestras pode ser algo simples de ser feito se comparado a dificuldade de conseguir uma orquestra de verdade para executar sua obra e então gravar para colocar em um jogo, mas quando apenas se compõe o compositor precisa ter em mente o que cada instrumento é capaz de tocar. Ao usar MIDI, o compositor não tem limites, o piano pode ter mais de 10 notas tocada ao mesmo tempo ou um trompetista pode segurar uma mesma nota por mais de um minuto, algo impossível para humanos cujo tem apenas 10 dedos e pulmões que precisam encher de ar.

Figura 17: Interface do Cubase utilizando MIDI [14]

Outra variável que pode ser trabalhada com MIDI para criar alguma mais "humanizado" está apresentado na parte inferior da figura 17, conhecido como expressão, essa variável pode criar maior variação de força para as notas, evitando que pareça um robô executando a música. Pensando por exemplo em um pianista, para tornas as notas mais sutis ele as toca com mais leveza, e para dar mais impacto ao que toca exige mais força do músico, assim criando mais expressão.

Esses detalhes são apenas alguns para quem quer criar músicas que realmente simulam instrumentos reais. Para recriar isso é precisa conhecer cada um dos instrumentos, fatores como alcance ou registro deles podem ser de grande importância. Instrumentos como violino e viola podem ser similares de aparência, porém os dois trabalham com registros diferentes de notas. Violinos tem registro entre as notas o Sol 3 (G3) e o Lá 7 (A7). Já a viola é mais grave, com o registro entre as notas Dó 3 (C3) e o Mi 6 (E6). Logo um violino não pode tocar uma nota abaixo de G3 e a viola uma nota acima de E6, mas o inverso sim.

12.3 Volume saudável para o ouvinte

Um cuidado que é de extrema importância para o desenvolvimento de áudio para qualquer plataforma ou mídia é o volume que se trabalha.

Os ouvidos humanos são extremamente sensíveis, assim podendo captar sons dos mais sutis como uma agulha caindo e o

bater das asas de um mosquito, logo quando se trata de volumes muito altos o ouvido é preciso ter cuidado.

Para cuidar dos volumes altos existem medidas de volume que facilitam o controle dele durante o processo de masterização de áudio. LU, ou LUFS, ou LKFS, é conhecido como "*Loudness Units*" que em português significa "Unidade de altura de volume do som", que foi decretado pela ATSC (Advanced Television Systems Committee, Inc.) [2] como tendo o valor ideal para os humanos como -24 LUFS/LKFS e tendo os valores em decibéis (dB) entre -6dB e 0dB. [11]

Softwares de DAW (*Digital Audio Workstation*) como Cubase [14], fornecem essas medidas para quem está usando, assim facilitando a sua masterização.

13 Conclusão

O trabalho de *Sound Designer* ganha novos caminhos com softwares como FMOD. Fornecendo uma grande quantidade de ferramentas e utilidades, *middlewares* criam um meio mais prático de trabalhar com áudio, facilitando o que antes só poderia ser feito com um programador ao lado do compositor.

Desde músicas, até efeitos sonoros, as aplicações do FMOD vão até o limite da imaginação do *Sound Designer*, com seus efeitos que modificam o som, interface gráfica que facilita a visualização do que está sendo feito e integração total com softwares de criação de jogos como Unity.

Fundamental para um trabalho mais profissional, *middlewares* são um novo mundo para quem deseja trabalhar na área de sonoplastia para jogos, mostrando que a confecção da mesma vai muito além de apenas inserir músicas e alguns sons.

Referências

[1] A. Marks. The Complete Guide to Game Audio. Focal Press, Burlington, MA, 2nd edition, 2009.

[2] ATSC
https://www.atsc.org/

[3] Bit rate - Wikipedia
https://en.wikipedia.org/wiki/Bit_rate

[4] Curso Introduction of Game Audio.
https://online.berklee.edu/courses/introduction-to-game-audio
Berklee College of Music, Boston, MA, 2017.

[5] Dead Space Games - EA
http://www2.ea.com/deadspace

[6] Defining Adaptive Music, Andrew Clark.
https://www.gamasutra.com/view/feature/129990/defining_adaptive_music

[7] [7] FMOD
https://www.fmod.com/

[8] Gamasutra:Anthony's Blog - User interface design in video games
https://www.gamasutra.com/blogs/AnthonyStonehouse/20140227/211823/User_interface_design_in_video_games

[9] Game boy sound hardware - GbdevWiki
http://gbdev.gg8.se/wiki/articles/Gameboy_sound_hardware

[10] Game Boy | A Empresa | Nintendo
https://www.nintendo.pt/A-empresa/Historia-da-Nintendo/Game-Boy/Game-Boy-627031.html

[11] Loudness Explained - Loudness | TC Eletronic
http://www.tcelectronic.com/loudness/loudness-explained/

[12] Metro Exodus | UK
http://www.metrothegame.com/en-gb/

[13] Midi Editing - www.steinberg.net
https://www.steinberg.net/forums/viewtopic.php?f=226&t=94146

[14] Resident Evil
http://game.capcom.com/campaign/biohd/lang.html

[15] [16] Site Oficial The Elder Scrolls | Skyrim
https://elderscrolls.bethesda.net/pt/skyrim

[16] [17] Sound Effects Library Categories | Sound Ideas
https://www.sound-ideas.com/

[17] Square Wave - Wikipedia
https://en.wikipedia.org/wiki/Square_wave

[18] Super Mario World for Nintendo 3DS – Nintendo Game Details
http://www.nintendo.com/games/detail/super-mario-world-vc-snes-3ds

[19] Tazman-Audio | Fabric
http://www.tazman-audio.co.uk/fabric

[20] The Legend of Zelda series for Nintendo Systems
http://www.zelda.com/

[21] Unity – Game Engine
https://unity3d.com/pt

[22] Unity - Manual: Audio Source
https://docs.unity3d.com/Manual/class-AudioSource.html

[23] Unity - Scripting API: Mathf.Lerp
https://docs.unity3d.com/ScriptReference/Mathf.Lerp.html

[24] Unity - Sound Effects & Scripting
https://unity3d.com/pt/learn/tutorials/topics/audio/sound-effects-scripting

[25] What Are Microphone Polar Patterns?
https://www.thepodcasthost.com/equipment/microphone-polar-patterns/

[26] Wwise | Audiokinetic
https://www.audiokinetic.com/products/wwise/