Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Nathalia Orlandi Borin

Jogos de Ritmo: geração automática de beatmaps e alternativas de jogabilidade

São Paulo
Janeiro de 2021

Jogos de Ritmo: geração automática de beatmaps e alternativas de jogabilidade

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Marcelo Queiroz

São Paulo
Janeiro de 2021

Resumo

Este trabalho tem o objetivo de explorar soluções para duas limitações presentes na maioria dos jogos de ritmo do mercado: o número restrito de músicas, devido ao caráter manual e demorado da criação de *beatmaps* que compõem suas fases, e a falta de originalidade na jogabilidade dos mesmos, que costumam exigir que o jogador execute ações em sincronia com a música, em instantes de tempo especificados por esse mesmo *beatmap*.

Foi desenvolvido um estudo na área de processamento de áudios musicais, especificamente no campo de detecção de *onsets* e *beats*, a fim de automatizar o processo de criação de *beatmaps*. Esses *beatmaps* foram gerados a partir de combinações de classificadores, que se baseiam em diferentes características do áudio (energéticas, espectrais, de fase e complexas) para determinar se um *onset* ocorre em um certo intervalo de tempo. Os *beatmaps* foram validados através da comparação entre 77 *beatmaps* do jogo referência de mercado e de código aberto *osu!*, sendo a combinação entre classificadores baseados em características energéticas e complexas responsável pelo melhor resultado de 60% de similaridade média entre *beatmaps* de uma mesma música.

Esses *beatmaps* foram usados no desenvolvimento de protótipos de jogos de ritmo: um dentro dos moldes tradicionais e outro com jogabilidade original. Os protótipos foram testados por 14 pessoas, que os avaliaram nos aspectos de usabilidade, diversão e percepção de sincronia com suas músicas. Numa escala de 1 a 5 de sincronia, os protótipos obtiveram nota média de 3,5, enquanto a média para o fator diversão foi de 3,85 e 2,92, respectivamente, consideradas avaliações satisfatórias para o escopo desse projeto.

Os resultados mostram que a automatização do processo de gerar *beatmaps* para jogos de ritmo tradicionais, assim como usos originais da música para compor a jogabilidade podem ser bem recebidos pelos jogadores, sendo estratégias viáveis para o desenvolvimento de jogos de ritmo. No entanto, os algoritmos e mecânicas desse trabalho ainda podem e devem ser aprimorados.

Palavras-chave: jogos de ritmo, *beatmap*, processamento de áudio, extração de informações musicais

Abstract

This project aims to explore alternatives for two common limitations in current commercial rhythm games: the restricted number of songs, caused by the difficulty in creating beatmaps for their levels, which is a manual and time consuming task, and the lack of original gameplay ideas, since these games usually involve pressing buttons in time with the music according to said beatmap.

In order to achieve these, a study was made in the field of musical audio processing, specifically in onset and beat detection. Those techniques were used for automating the process of beatmap generation, based on the combination of classifiers. Each classifier uses a different feature in the audio signal - energetic, spectral, phase and complex features - in order to determine if an onset is present in a given time interval. The beatmaps generated were validated through comparison with 77 osu! beatmaps, a well known open source game in the rhythm game genre. The combination which yielded the best results was the union between an energy-based and a complex-based classifier, producing an average of 60% similarity between beatmaps of the same song.

These beatmaps were used in the development of two rhythm game prototypes: one in the vein of traditional rhythm games and the other with original gameplay. These prototypes were tested by 14 people, which surveyed the usability, fun and perceived synchronization between the game and their music. In a five point scale, the prototypes obtained the average value of 3.5 for the synchronization factor and 3.85 and 2.92, respectively, for the fun factor. These values were considered acceptable for the scope of this project.

The results show that automating the process of beatmap generation for rhythm games in the traditional style, as well as original uses of music in gameplay can be well received by players, becoming viable strategies for game development in the genre. However, the algorithms and mechanics developed in this project can and should still be improved.

Keywords: rhythm games, beatmaps, audio processing, music information retrieval

Sumário

Capítulo 1

Introdução

Jogos de ritmo são jogos cujas principais interações baseiam-se em fatores musicais de caráter rítmico. Neste tipo de jogo, espera-se que o jogador realize suas ações dentro de janelas de *timing* precisas, geralmente seguindo a batida de uma música. Rebelo (2016) Alguns exemplos populares de jogos desse gênero são: *Guitar Hero, Rockband, osu!, Dance Dance Revolution* e *Beat Saber*.

Todos esses jogos possuem em comum os seguintes aspectos:

1. O jogador é forçado a realizar um conjunto de ações previamente definidas por um *beatmap* construído para cada música. Este *beatmap*, tido como *ground truth*, pode variar de acordo com a dificuldade do jogo. A pontuação do jogador é calculada considerando a distância entre as ações do jogador e as ações estipuladas no *beatmap*.

2. As fases do jogo são baseadas em músicas tocadas de forma contínua e linear, de forma que o jogador tenha a sensação de que ele mesmo está realizando alguma **performance** da música tocada Pichlmair e Kayali (2007).

3. Existe um *feedback* visual, de modo que o jogador não precisa guiar-se apenas por estímulos auditivos para ter um bom desempenho, fazendo com que o jogo possa ser considerado num desafio de coordenação motora.

4. Limitação na escolha das músicas que o jogador pode jogar, de forma que se um jogador quiser jogar uma fase com uma música de sua preferência, que não está disponível na biblioteca do jogo, precisará passar um tempo considerável configurando o jogo para tal. Para contornar essa restrição, observa-se também o desenvolvimento de ferramentas que auxiliam na geração de beatmaps para jogos específicos como Dance Dance Revolution Chris Donahue e McAuley (2017) e Beat Saber Society (2019), utilizando redes neurais.

1.1 Objetivos

O objetivo deste trabalho é desenvolver um jogo de ritmo que se diferencie dos jogos no mercado atualmente, resolvendo alguns problemas decorrentes principalmente dos aspectos 1 e 4 citados na seção anterior.

Em relação ao uso do *beatmap* como *ground truth* das ações do jogador, identifica-se aí uma restrição muito forte e possivelmente repetitiva em relação ao que se espera do jogador.

Em relação à limitação das músicas, identificam-se jogos que conseguem criar beatmaps de forma automatizada, dado um arquivo de áudio do jogador. Porém, observa-se que o

resultado acaba muitas vezes sendo insatisfatório: ou o *beatmap* gerado é muito simplificado, tornando a experiência pouco divertida, ou muito difícil, não levando em conta a complexidade dos *inputs* que se espera do jogador, tornando efetivamente impossível a obtenção de uma pontuação máxima.

Dessa forma, para o jogo que será desenvolvido, pretende-se utilizar técnicas de processamento de áudio para extrair, a partir de arquivos de música escolhidos pelo usuário, *features* que ajudarão na composição de um *beatmap*. Este *beatmap* será usado para influenciar aspectos do jogo, como quantidade e posicionamento de obstáculos, ou quantizando o movimento do jogador para se enquadrar nos *beats*, por exemplo.

Quanto aos itens 2 e 3, deseja-se manter o aspecto performativo dos jogos de ritmo, ainda que as ações do jogador não estejam diretamente relacionadas a música. Já a questão do *feedback* visual certamente contribui para a experiência do jogador, pretendendo-se utilizar o máximo possível da influência da música, como elementos que pulsam com os *beats* da música, variação de cores em função do *pitch*, entre outros.

Em resumo, os objetivos deste trabalho são:

1. Adquirir conhecimentos sobre técnicas de processamento de áudios musicais.
2. Aplicar esses conhecimentos no desenvolvimento de um software que produz *beatmaps* para uma música de entrada.
3. Desenvolver um protótipo de jogo de ritmo que usa esses *beatmaps* para gerar desafios interessantes para o jogador.

1.2 Metodologia

Para se atingir os objetivos citados anteriormente, a metodologia utilizada será:

1. Estudar jogos de ritmo, através do levantamento de literatura existente na área e do mapeamento dos jogos de ritmo existentes e de fatores comuns entre eles, assim como jogar alguns desses jogos, com o objetivo de obter maior familiaridade com as mecânicas.
2. Estudar algoritmos de processamento de áudios musicais, em especial algoritmos de detecção de *beats* e *onsets*. Implementar e testar os algoritmos estudados, realizando testes automatizados ou pequenos experimentos de validação, conforme o caso.
3. Desenvolver um ou mais protótipos de jogos que utilizem diferentes maneiras de atrelar a jogabilidade a eventos musicais presentes em uma música do usuário.
4. Colher *feedback* de jogadores sobre suas impressões desses protótipos em relação à sincronia com a música e originalidade da jogabilidade proposta.

Capítulo 2

Fundamentação Teórica

Este capítulo é um resumo dos aprendizados obtidos do livro Fundamentals of Music Processing, escrito por Müller (2015). Seu objetivo é introduzir alguns conceitos fundamentais a respeito de música, som e áudio, assim como apresentar algumas ferramentas que serão utilizadas para a extração de *features* relevantes para seções futuras.

2.1 Representações Musicais

O objetivo de construção de um jogo de ritmo traz uma pergunta inicial bastante natural: o que é ritmo? Para responder essa pergunta, é necessário um entendimento, ainda que superficial, sobre teoria musical.

Podemos entender música como um conjunto organizado, com base em escolhas estéticas e/ou subjetivas, de sons. A partir de uma **partitura**, um músico treinado consegue realizar a **perfomance** de uma música, que pode ser gravada, produzindo, então, um sinal de **áudio**.

Vamos agora investigar como essas duas maneiras distintas de se representar uma obra musical se relacionam entre si.

2.1.1 Partitura

Uma partitura possui todas as informações necessárias para que um músico consiga performar uma peça. Pode ser entendida como uma sequência de instruções a respeito de quais notas devem ser tocadas, por quais instrumentos, em quais instantes de tempo e com quais intensidades e durações.

A unidade mais simples dentro de uma peça musical é a **nota**. Uma nota possui uma **altura**, que percebemos dentro de uma escala que vai de grave a agudo, e uma duração. A altura de uma nota é determinada pela linha da partitura na qual ela está posicionada 2.1, enquanto sua duração é definida pelo desenho da nota 2.2.

Na música ocidental, a altura de uma nota costuma pertencer a um conjunto finito chamado de **escala musical**. Neste trabalho, será utilizada como referência a **escala diatônica**, compostas pelas classes de alturas C, C#, D, D#, E, F, F#, G, G#, A, A# e B. Avançar um passo nessa escala corresponde a um intervalo de **semitom**. Outro intervalo bastante importante é o de **oitava**, por motivos que serão explorados mais adiante.

À forma como as notas musicais estão organizadas no tempo se chama de **ritmo**. Na notação musical ocidental, organiza-se o tempo através de **compassos**, que por sua vez são definidos por uma **assinatura de tempo**, representado por um número fracionário. O denominador dessa assinatura determina qual duração corresponde a uma unidade de tempo

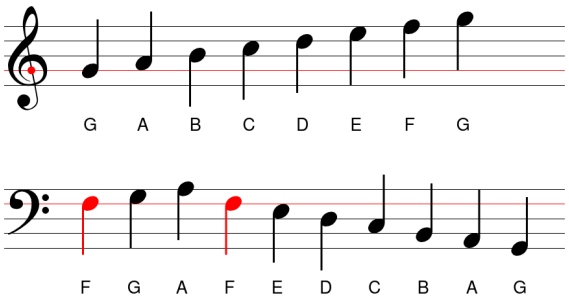

Figura 2.1: *Alturas musicais. Na esquerda, a clave de Sol indica que a segunda linha de baixo para cima corresponde à posição da altura Sol (G). Na direita, o mesmo se aplica para a clave de Fá (F)*

dentro do compasso, enquanto o numerador determina quantas notas dessa duração cabem nele musictheory.net (2020).

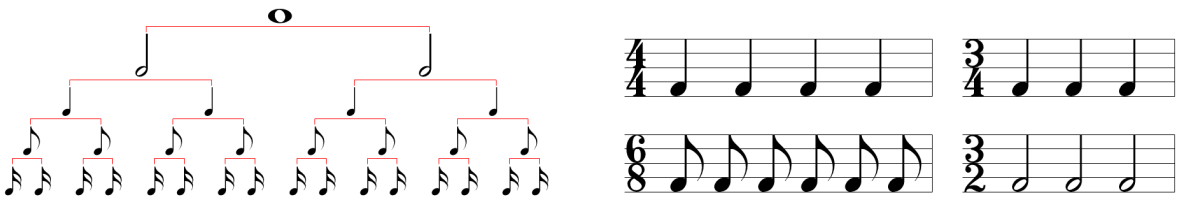

Figura 2.2: *Na esquerda: símbolos que representam a duração de uma nota. De cima para baixo: semibreve, mínima, semínima, colcheia e semicolcheia. Na direita: Exemplos de compassos completos com diferentes assinaturas de tempo*

Essas informações porém não são suficientes para a realização de uma peça: é preciso saber a quantos segundos corresponde uma unidade de tempo. Isso é medido pelo **tempo** de uma música, normalmente expresso em BPM (batidas por minuto).

Outra informação necessária para a realização da performance de uma música diz respeito à **intensidade** com o qual as notas devem ser tocadas. Na notação ocidental, essa intensidade está organizada numa escala que vai do *pianissimo* (mais fraco) ao *fortissimo* (mais forte).

2.1.2 Áudio

Quando uma partitura é performada por um músico, seu instrumento produz uma série de ondas acústicas, que são percebidas como **sons**. Um sinal de áudio é uma representação elétrica ou digital desses sons. Neste trabalho, o foco se concentrará nas representações digitais.

Som é um fenômeno físico gerado por um objeto que vibra, como as cordas vocais de um cantor ou as cordas de um piano ou violino. Essas vibrações fazem com que as moléculas de ar sejam deslocadas, gerando regiões de compressão e rarefação, que resultam em variações de pressão.

Costuma representar-se o fenômeno do som por um gráfico de variação de pressão (em relação à pressão atmosférica) ao longo do tempo. Tal gráfico é chamado de **forma de onda** de um som. Uma forma de onda bastante trabalhada em computação musical é a **senóide**. A senóide é o tipo de sinal periódico mais simples, podendo ser completamente especificada por sua **frequência**, **amplitude** e **fase** 2.3.

- **Frequência**: Quando um som produz um sinal periódico ou quase periódico, é possível depreender dele uma percepção de altura, numa escala de grave a agudo. Quanto maior

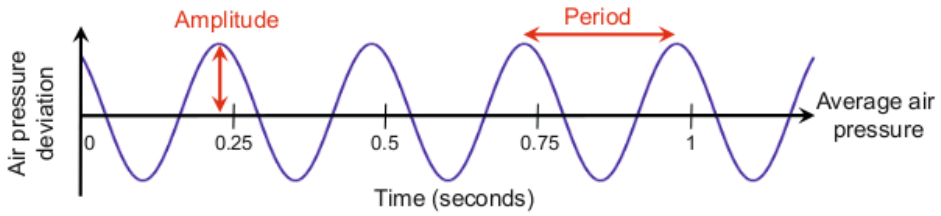

Figura 2.3: *Forma de onda de uma senóide de 4 Hz.*

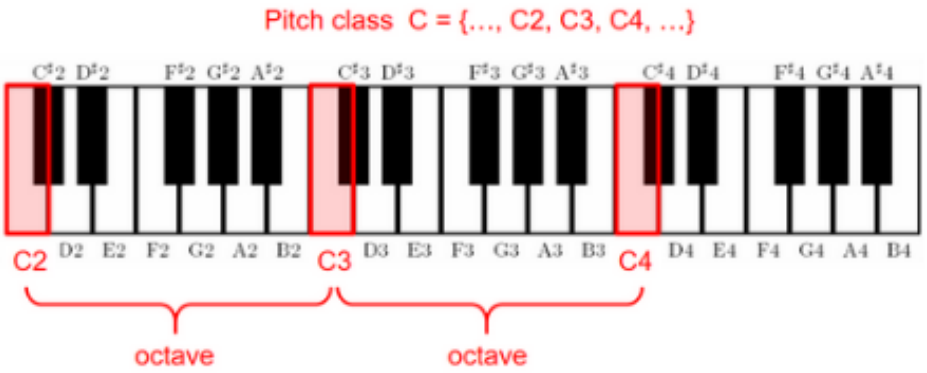

Figura 2.4: *Representação de um intervalo de oitava em um teclado.*

a frequência, mais **agudo** é o som percebido. Quanto menor a frequência, mais **grave** é o som percebido. A faixa de frequência audíveis para os seres humanos se situa entre 20 Hz e 20.000 Hz. Cada altura citada na seção anterior possui uma **frequência fundamental** relacionada. Quando uma nota tem o dobro da frequência de outra, dizemos que há entre elas o intervalo de **uma oitava**. Notas que diferem de uma oitava entre si possuem sons perceptivamente muito similares. Por isso, pertencem à mesma **classe de altura**. Na escala igualmente temperada, divide-se uma oitava em 12 semitons, no qual suas frequências estão divididas de forma igualmente espaçadas num eixo de frequência logarítmica.

- **Amplitude**: A amplitude de um som está relacionada com o **volume** com que percebemos esse som: sons com maior amplitude são percebidos como mais fortes, enquanto sons de menor amplitude são percebidos como mais fracos. Fisicamente, essa propriedade denota a energia de um som por unidade de área e é normalmente medida usando a escala de decibéis (dB), uma unidade logarítmica. Em áudio, porém, essa unidade é um valor adimensional normalmente situado no intervalo [0, 1] ou [-1, 1].

- **Fase**: A fase de um sinal corresponde ao ângulo, normalmente medido em radianos, no qual o sinal passa pelo zero. Não possui nenhum correspondente perceptivo, mas é um parâmetro importante ao se considerar a soma de sinais atrasados. Por exemplo, ao gravar um instrumento usando-se dois microfones, o sinal chegará ao microfone mais próximo em um certo instante de tempo, enquanto ao outro chegará com alguns milissegundos de atraso. Ao somar esses sinais, pode ocorrer um fenômeno chamado **cancelamento de fase**, fazendo com que algumas frequências se cancelem e sejam perdidas no resultado final.

Quando dois instrumentos diferentes tocam uma mesma nota (portanto, produzindo uma mesma frequência fundamental) e com mesma intensidade (portanto, com mesmo volume), o

que diferencia o som desses instrumentos é chamado de **timbre**. ois fatores que caracterizam o timbre de um instrumento são seu envelope e a proporção de seus parciais.

O envelope de um sinal pode ser descrito como o contorno de sua forma de onda. É normalmente descrito utilizando o modelo ADSR: primeiro, há uma fase de *Attack* (A), na qual normalmente ocorre um aumento repentino de energia; seguido pelo *Decaimento*, onde essa energia diminui; e depois pelo *Sustain* (S), quando a energia se mantém mais ou menos constante e, por fim, o *Release* (R), quando a energia vai a zero.

Quando um A4 (440 Hz) é tocado, por exemplo, o sinal resultante terá energia em 440 Hz, sua frequência fundamental, assim como nas regiões de 880 Hz, 1320 Hz, 1660 Hz, e outros múltiplos inteiros da frequência fundamental. À essas outras frequências, dá-se o nome de **parciais harmônicos** do som tocado. A proporção de energia presente nesses parciais dão uma coloração diferente para o som produzido, variando bastante de instrumento para instrumento.

2.1.2.1 Conversão Analógico para Digital

Para que um sinal analógico (infinito e contínuo) possa ser representado dentro de um computador, com capacidades de processamento e armazenamento finitas, algumas operações precisam ser realizadas:

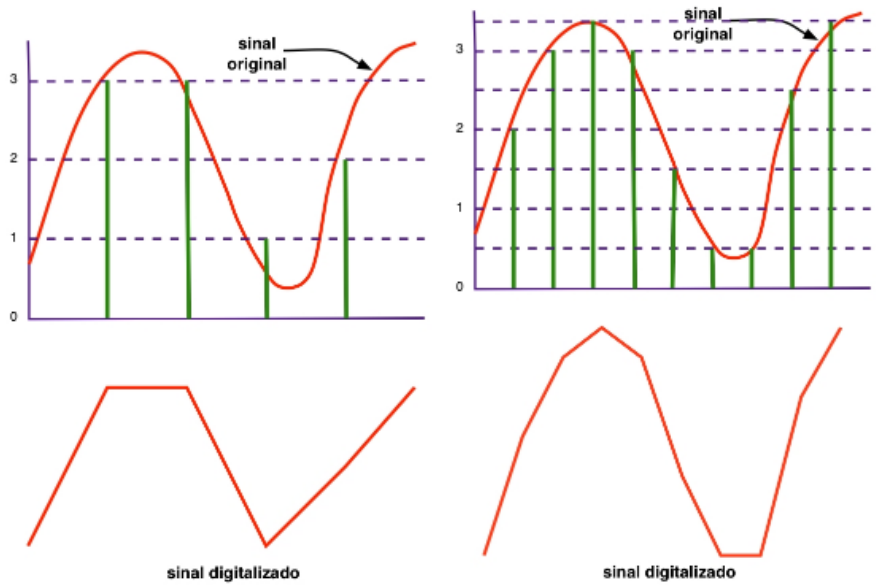

Figura 2.5: *Comparação entre sinais originais analógicos e suas respectivas digitalizações. Na esquerda, utilizou-se uma taxa de amostragem de 4 Hz e 2 bits para quantização. Na direita, a taxa de amostragem é de 9 Hz, e 3 bits foram usados na quantização.*

- **Amostragem** consiste em discretizar o sinal no domínio do tempo. Isto é, toma-se amostras do sinal com um período de tempo T entre amostras. Define-se como **taxa de amostragem** [1] o inverso desse período T. Quanto maior a taxa de amostragem, mais fiel ao sinal original será o resultado. A escolha desse parâmetro também determina um limite nas **frequências representáveis**. Segundo o Teorema de Nyquist, dado uma taxa de amostragem R, só é possível representar sinais de frequência até R/2. Tal frequência é chamada de **Frequência de Nyquist**. Para sinais com frequência

[1] Um valor de taxa de amostragem muito utilizado, popularizado pelos CDs é o de 44100 Hz.

maior ou igual a R/2, ocorre o fenômeno do **rebatimento**, que introduz componentes indesejadas e muitas vezes audíveis no espectro. Por esse motivo, um componente presente nos conversores analógico-digital (ADC) é um filtro do tipo **passa-baixa**, que corta frequências acima da frequência de Nyquist.

- **Quantização** consiste em discretizar o sinal no domínio da amplitude. Isto é, limitar a precisão de cada amostra tomada. O nível de quantização é normalmente representado pelo **número de bits** utilizado para representar cada amostra. [2] A escolha desse parâmetro determina a amplitude máxima do chamado **ruído de quantização**. Se o número de bits usado para representar cada amostra for muito baixo, esse ruído poderá ser audível, especialmente em sinais de pouca amplitude. Em áudio, manter uma boa **relação sinal-ruído** é uma tarefa fundamental Iazetta (2020).

2.2 Processamento de Sinais de Áudio

Uma pergunta bastante que faz bastante sentido dentro de um contexto de sinais musicais é: "que nota está sendo tocada agora?". Na seção anterior, foi exposto que a altura de uma nota está relacionada com sua **frequência fundamental**, a frequência do parcial mais baixo produzido por um instrumento tocando essa nota.

Sinais musicais são formados normalmente por uma composição dos sinais periódicos simples vistos na seção anterior. Uma operação muito utilizada em sinais desse tipo para decompô-los em sinais mais fáceis de compreender e interpretar é a **transformada de Fourier**.

2.2.1 Análise de Fourier

A transformada de Fourier converte um sinal do **domínio do tempo** para o **domínio da frequência**.

Intuitivamente, a transformada de Fourier realiza uma comparação entre o sinal original e várias senóides puras de frequências diferentes. Primeiro, é necessária uma amostragem do eixo das frequências, similar ao que ocorre com o tempo, dentro do intervalo $[0, \frac{\omega}{2}]$, sendo ω a Frequência de Nyquist. Para cada índice de frequência f dentro desse intervalo, a transformada retorna um coeficiente $\chi(f)$, que corresponde à similaridade entre o sinal sendo analisado e uma senóide pura de frequência k, assim como uma fase ϕ_k, que corresponde à fase desse sinal que maximiza essa similaridade. Essa comparação é feita através do produto interno entre os dois sinais.

A Transformada de Fourier no domínio discreto (DFT ou *Discrete Fourier Transform*) de um sinal $x[n]$ de tamanho N é definida como:

$$\chi(k) := \sum_{n=0}^{N-1} x(n) exp(\frac{-2\pi kn}{N})$$

Onde o argumentos k corresponde ao índice de frequência

A partir desses coeficientes, é possível reconstruir o sinal original através da **transformada inversa de Fourier**, que consiste numa soma de senóides de cada frequência de índice k deslocados de uma fase $\phi(k)$, ponderados pelo coeficiente de magnitude $\chi(k)$.

[2] Os CDs usavam 16 bits para representar cada amostra.

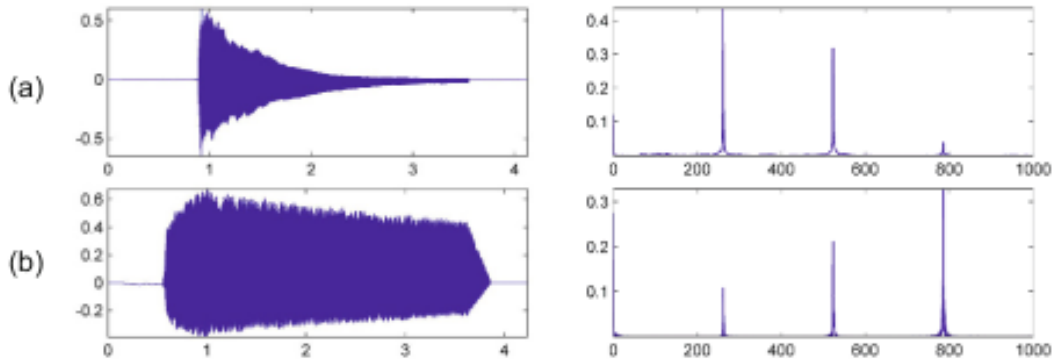

Figura 2.6: *Forma de onda e magnitude da Transformada de Fourier de um C4 tocado por piano (a) e trompete (b). É possível notar a diferença do timbre desses instrumentos tanto pelo formato de envelope quando pela proporção dos parciais harmônicos.*

2.2.1.1 Espectrogramas e Cromagramas

A DFT possui uma limitação: embora ela forneça informações sobre a frequência das senóides que compõem aquele sinal, não é possível localizá-las no **tempo**. A **transformada curta de Fourier** ou STFT (Short Time Fourier Transform) resolve esse problema. A STFT define uma função de janelamento w que é diferente de zero apenas para intervalos curtos de tempo. O sinal original é então multiplicado pela função de janela, gerando um sinal janelado. Essa janela vai sendo então deslocada ao longo do eixo do tempo para se obter a informação do espectro dentro de cada intervalo de tempo. Uma STFT possui então alguns parâmetros importantes:

- **Formato de janela:** O formato de janela introduz algumas diferenças no espectro resultante, decorrente do espectro da própria janela. Um formato bastante utilizado é o retangular, na qual $w(n) = 1$ para $n \in [0, N-1]$ sendo N o tamanho de janela e $w(n) = 0$ caso contrário.
- **Tamanho de janela:** Número N de amostras que serão usadas no cálculo da DFT para um único intervalo.
- **Tamanho de pulo:** Número H de amostras em que a janela é deslocada para a direita em cada iteração. Determina a resolução temporal do espectro obtido.

Define-se então a STFT de um sinal discreto $x[n]$ com função de janela w, tamanho de janela N e tamanho de pulo H como:

$$\chi(m,k) := \sum_{n=0}^{N-1} x(n+mH)w(n)exp(\frac{-2\pi kn}{N})$$

Onde os argumentos m e n correspondem, respectivamente, aos índices das amostras de tempo e frequência. É possível recuperar o tempo em segundos e a frequência em Hz usando a txa de amostragem R através das seguintes equações:

$$T(m) := mH/R$$
$$F(k) := kR/N$$

Uma DFT é uma operação que exige bastante processamento, sendo um algoritmo de complexidade O(n), o que tornaria inviável suas aplicações sucessivas na STFT. Porém,

utiliza-se hoje um algoritmo de **transformada rápida de Fourier** ou FFT (*Fast Fourier Transform*), com complexidade O(nlgn), baseado em técnicas recursivas.

A partir das informações obtidas de uma STFT, é possível construir um **espectrograma** de um sinal: um gráfico de frequência por tempo, no qual a intensidade das cores representa a energia da componente de uma certa frequência em um certo intervalo de tempo.

Uma variação do espectograma também muito utilizada é o **cromagrama**, no qual frequências pertencentes à mesma **classe de altura** (como C1, C2, ... Cn) são somadas.

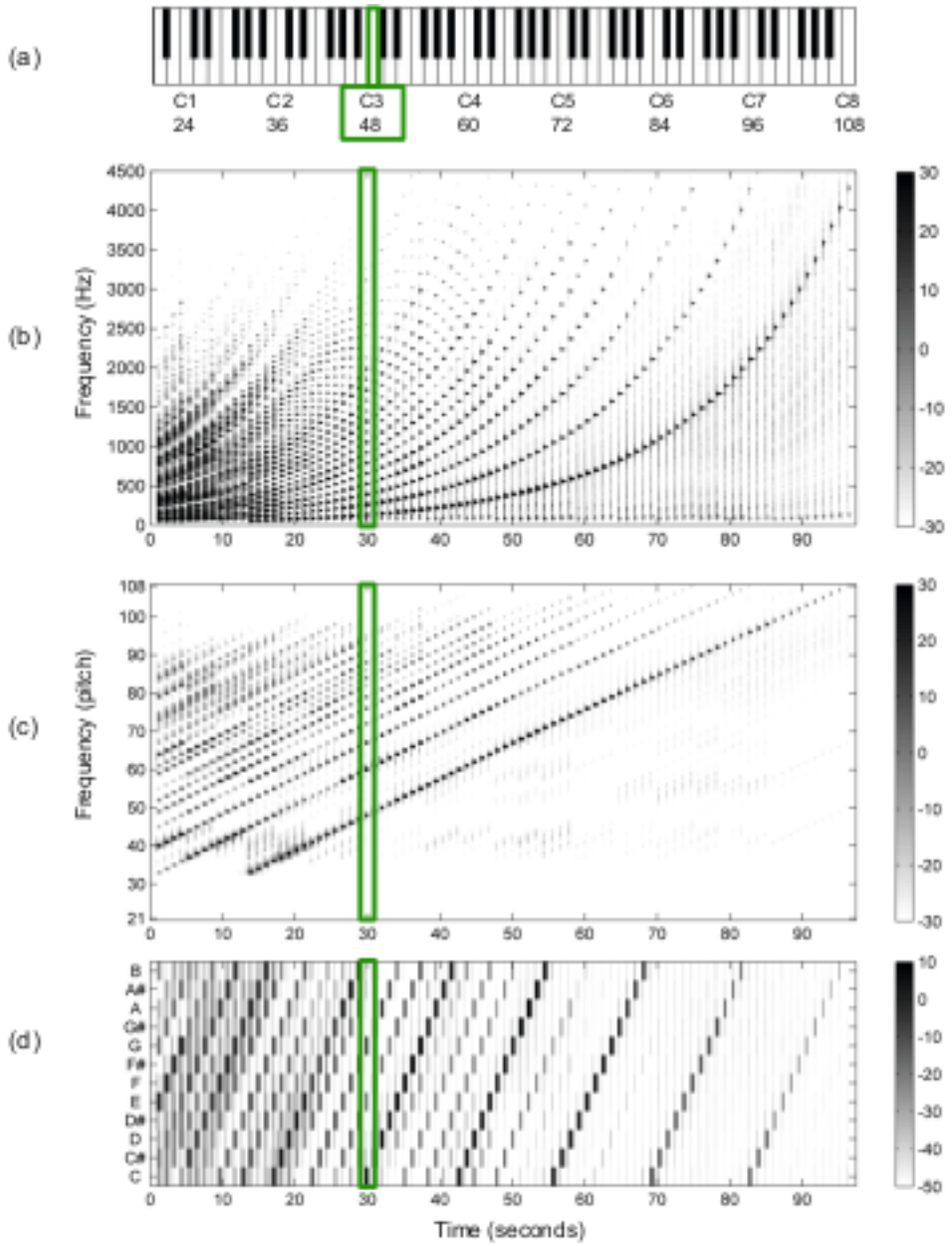

Figura 2.7: *Representações diferentes para o mesmo sinal de uma C3. Na esquerda, um espectrograma com o eixo das frequências em escala linear. Na direita, em (c), as frequências aparecem em escala logarítmica, enquanto (d) é um cromagrama.*

Capítulo 3

Desenvolvimento do Beatmapper

Em um jogo de ritmo, cada fase possui um *beatmap*, que dita as ações do jogador. Formalmente, podemos defini-lo como uma sequência de intervalos de tempo e ações que o jogador deve realizar dentro desses intervalos.

Nesse sentido o *beatmap* se aproxima bastante das representações simbólicas discutidas no capítulo 1, como uma partitura, pois também corresponde a uma sequência de instruções que dão origem a uma performance.

As ações realizadas em um jogo de ritmo não são aleatórias: é possível perceber, enquanto se joga, que elas correspondem à eventos da música sendo tocada, ou a complementa de alguma forma. Isso traz à tona a pergunta: a que eventos essas ações correspondem? Ou, mais genericamente: dado uma música, como é construído seu *beatmap*?

Ou, melhor dizendo, como é construído **algum** de seus *beatmaps*, pois a relação entre música e *beatmap* não é 1:1 - uma mesma música pode dar origem a infinitos *beatmaps*, pois jogos distintos possuem versões diferentes de *beatmaps* para uma mesma música. E mesmo dentro de um único jogo, *beatmaps* de uma mesma música podem variar de acordo com a dificuldade do jogo.

3.1 O Experimento

A observação anterior nos leva a crer que os *beats* de um *beatmap* correspondem a *features* musicais da obra a partir da qual ele foi construído.

Com o intuito de descobrir quais seriam essas *features*, foi projetado o seguinte experimento:

1. Encontrar exemplos de *beatmaps*
2. Extrair *features* a partir do sinal de áudio das músicas correspondentes
3. Comparar os *beatmaps* gerados a partir dessas *features* com o *beatmap* original
4. Encontrar a *feature* cujo *beatmap* gerado mais se assemelha ao original

3.1.1 Preparação

A maioria dos jogos de ritmo no mercado são de código fechado e não foi possível encontrar informações suficientes sobre seu desenvolvimento para concluir como seus *beatmaps* foram construídos.

Uma notável exceção é o jogo osu! [1], gratuito e open source, com milhões de *beatmaps*

[1] https://osu.ppy.sh

construídos pela comunidade e disponíveis de forma aberta na internet.

O formato de *beatmap* de osu! é, também, público, sendo um XML facilmente parseável usando qualquer linguagem de programação. Pela popularidade do jogo, até foi encontrado uma biblioteca já existente de NodeJS [2] que parseia o formato .osu e devolve um objeto de fácil manipulação.

As fases desse jogo podem ser baixadas num formato .osz, que consiste em um zip com o arquivo .osu [3] citado anteriormente, assim como um arquivo .mp3 da música correspondente.

A facilidade de acesso e manipulação desses beatmaps, assim como a disponibilidade dos áudios das fases torna o jogo osu! ideal para a realização do experimento descrito.

3.1.2 O Dataset

O dataset final consiste em 77 beatmaps escolhidos a mão dentre os disponíveis no site bloodcat [4]. Para a escolha dos beatmaps, foram tomadas as seguintes precauções:

1. Foram escolhidos apenas beatmaps bem avaliados pela comunidade osu!
2. Buscou-se contemplar também beatmaps de músicas instrumentais, compondo aproximadamente 25% da amostra.
3. Buscou-se a maior diversidade de gêneros musicais possível entre os beatmaps disponíveis.

3.1.3 Detecção de onsets

Define-se como *onset* o instante de tempo no qual uma nota começa a ser tocada por um instrumento. Mais precisamente, o *onset* marca o início da fase de Ataque dentro do modelo ADSR 3.1 discutido no capítulo anterior, uma fase caracterizada pelo aumento rápido da energia do sinal.

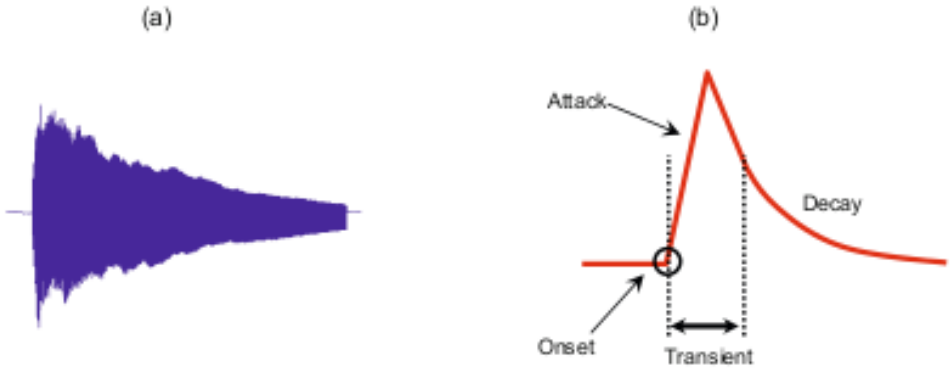

Figura 3.1: *(a) Sinal de áudio de uma nota de piano (b) Modelo ADSR*

Considerando-se que em jogos de ritmo, é muito comum a sensação de que o jogador está realizando uma **performance** Kähärä (2018) da música sendo tocada, isto é, que suas ações correspondem a notas da respectiva música, o conceito de *onset* parece ser um bom candidato para representar uma nota em um *beatmap*.

[2] https://www.npmjs.com/package/osu-parser
[3] https://osu.ppy.sh/help/wiki/osu!_File_Formats/Osu_(file_format)
[4] https://bloodcat.com/osu/

Os passos do algoritmo de detecção de onsets descritos por Müller (2015) são:

1. Extrair uma sequência de features de interesse
2. Criar uma função de novidade a partir de diferenças nessas features
3. Aplicar essa função de novidade no sinal de áudio
4. Aplicar um algoritmo de detecção de picos na saída da função utilizada

Os parâmetros passíveis de variação nesse experimento, são, portanto, a **função de novidade** e o **algoritmo de detecção de picos**.

3.1.4 Funções de Novidade

Müller (2015) descreve funções de novidade baseadas em features energéticas, espectrais, de fase e de domínio complexo. Neste experimento, foram utilizdas as implementações dos *notebooks* disponibilizados por Müller e Zalkow (2019).

3.1.4.1 Features Energéticas

A energia de um sinal pode ser representada como a soma dos quadrados dos valores de amplitude do mesmo. Para obtermos esses valores localmente, assim como na STFT, usamos uma função de janelamento w.

A energia local de um sinal $x(n)$ discreto, então, considerando uma função de janelamento w pode ser definida como:

$$E_w^x(n) := \sum_{m=-M}^{M} |x(m)w(m-n)^2|$$

O que nos interessa para a detecção de onsets, no entanto, não é os valores de energia em si, e sim sua **variação**. Intuitivamente, é fácil de entender porquê: espera-se que, no momento em que uma nota acaba de ser tocada, ocorra uma grande variação positiva na energia desse sinal.

A função de novidade energética é, então, a derivada da função de energia. No caso discreto, consideramos a diferença entre amostras subsequentes da seguinte forma:

$$\Delta_{Energia}(n) := |E_w^x(n+1) - E_w^x(n)|_{\geq 0}$$

3.1.4.2 Features Espectrais

A função de novidade energética possui algumas limitações. No caso de músicas polifônicas, o aumento de energia causado pelo *onset* de um instrumento pode ser mascarado pelo sinal de outros instrumentos na composição. Para evitar isso, costuma-se levar em conta também o conteúdo espectral de um sinal, isto é, a variação no domínio das **frequências**.

Novamente, essa é uma noção muito intuitiva: dado que diferentes notas musicais produzem sinais com diferentes **frequências fundamentais**, assim como frequências harmônicas. É de se esperar então que o momento de um onset corresponda a um momento de grande variação do conteúdo espectral de um sinal.

Seja χ uma STFT discreta: para cada índice de janela $n \in Z$ obtemos um vetor espectral $\chi(n)$ com valores $\chi(n,k) \in C$, para valores $k \in [0,K]$ de frequência. Podemos definir então o valor de energia espectral de um sinal em um certo intervalo de tempo como sendo a magnitude desse vetor.

Costuma-se aplicar também uma função de compressão logarítmica Γ a esses valores, de forma a ressaltar componentes espectrais que possuem baixa amplitude mas ainda são audíveis:

$$\Gamma(n,k) := log(1 + \gamma * |\chi(n,k)|)$$

Dessa forma, definimos a função de variação espectral de um sinal como:

$$\Delta_{Espectral}(n) := \sum_{k=0}^{K} |\Gamma(n+1,k) - \Gamma(n,k)|$$

3.1.4.3 Features de Fase

A função espectral já é considerada bem robusta para a tarefa de detectar *onsets*, mas existem ainda outras *features* que podem ser utilizadas. Uma delas é uma informação que a STFT já traz: a fase dos componentes.

A intuição por trás dessa função é um pouco menos trivial: em áreas estáveis de um sinal, espera-se que a fase varie de forma linear. Já em fases de ataque existe a produção de muitos **transientes**, cuja variação de fase é muito mais caótica e imprevisível.

Assim, dado um índice de janela n e um coeficiente de frequência k, obtém-se através da STFT a fase $\phi(n,k)$ que maximiza o coeficiente de magnitude $\chi(n,k)$. Define-se então as derivadas de primeira e segunda ordem da função de fase como:

$$\phi'(n,k) = \phi(n,k) - \phi(n-1,k)$$
$$\phi''(n,k) = \phi'(n,k) - \phi'(n-1,k)$$

Dessa forma, em regiões estáveis, $\phi''(n,k))$ irá se aproximar de zero quando a fase variar de forma linear. Define-se então a variação de fase de um sinal como essa variação somada para todos os valores de frequência:

$$\Delta_{Fase}(n) := \sum_{k=0}^{K} |\phi''(n,k))|$$

3.1.4.4 Features Complexas

A função anterior tem uma limitação: quando o coeficiente de magnitude $\chi(n,k)$ é muito pequeno, a fase $\phi(n,k)$ pode variar bastante, mesmo em regiões estáveis.

Para corrigir isso, utiliza-se uma função que pondera a informação de fase pelo coeficiente de magnitude.

3.1.5 Detecção de Picos

Existem diversos algoritmos de detecção de picos ou máximos locais em funções. Dentro do domínio de detecção de *onsets*, são muito citados o algoritmo de Shostakovich, de Boeck, de Roeder, do pacote MSAF (Music Structure Analysis Framework), entre outros.

Para fins desse experimento, adotou-se a implementação da biblioteca LibROSA Mcfee *et al.* (2015) da função *librosa.onset.onset_ detect()*, com a maioria dos parâmetros em seus valores padrão. A justificativa está no fato de que são valores adequadamente escolhidos a partir de treinamento sobre grandes datasets especializados para a tarefa de detectar onsets [5].

Para esse experimento, o tamanho de janela e comprimento de pulo foram fixado em valores *default* de 1024 e 512, respectivamente.

3.1.6 Avaliação

Para comparar os tempos de onsets encontrados, correspondente aos picos das funções de novidade, com o *ground truth*, utilizou-se a função *mir_ eval.evaluate()*, do pacote desenvolvido por C. Raffel e Ellis (2014). A função retorna 3 valores: Precision (Pr), Recall (Re), e F-Measure, cujos significados encontram-se resumidos na figura 3.2

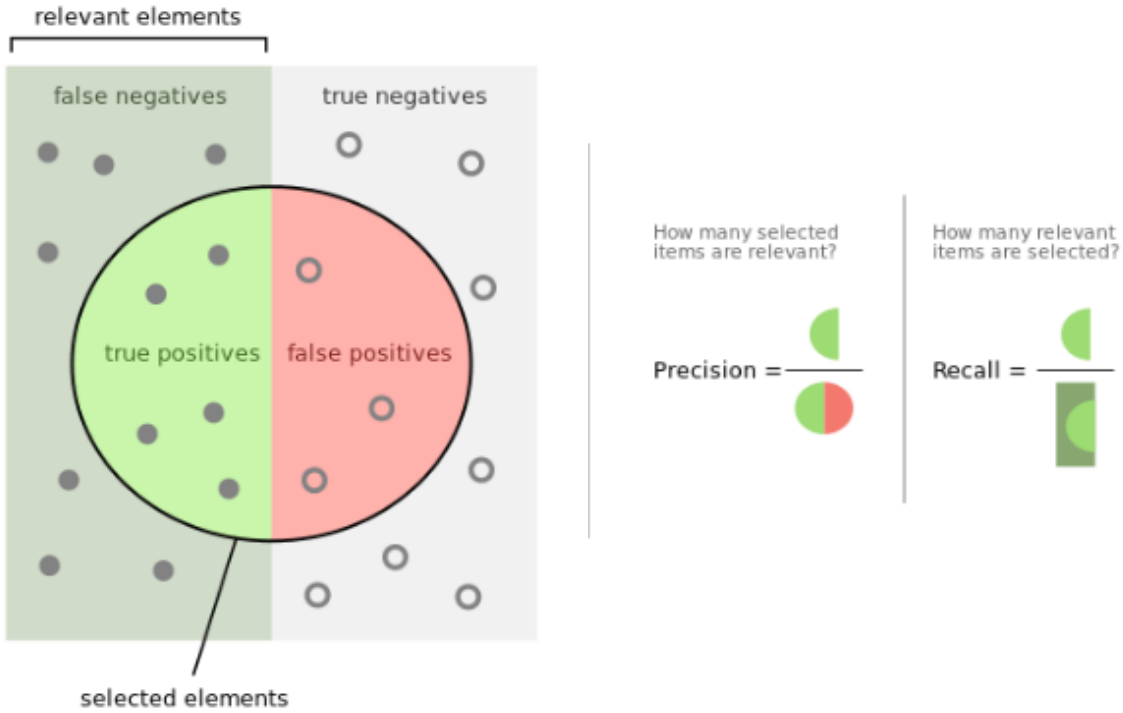

Figura 3.2: *Precision: razão entre verdadeiros positivos e todos os escolhidos e Recall: razão entre verdadeiros positivos e todos os positivos.*

3.1.7 Resultados Preliminares

Avaliando inicialmente os valores de F-Measure, obteve-se o melhor desempenho de 81,6% e a pior de 8,1%, mostrando que os valores variam bastante de música para música.

Posteriormente, analisou-se a distribuição dos valores de F-Measure para cada função de novidade através de boxplots 3.3. Concluiu-se então que as funções de novidade que tiveram melhor performance foram a baseada features energéticas (*label* energy512 na imagem) e a baseada em features complexas (*label* complex512 na imagem), com médias de 57,4% e 56,5% de F-Measure, respectivamente.

[5] Do link: https://librosa.org/doc/latest/generated/librosa.onset.onset_detect.html#librosa.onset.onset_detect

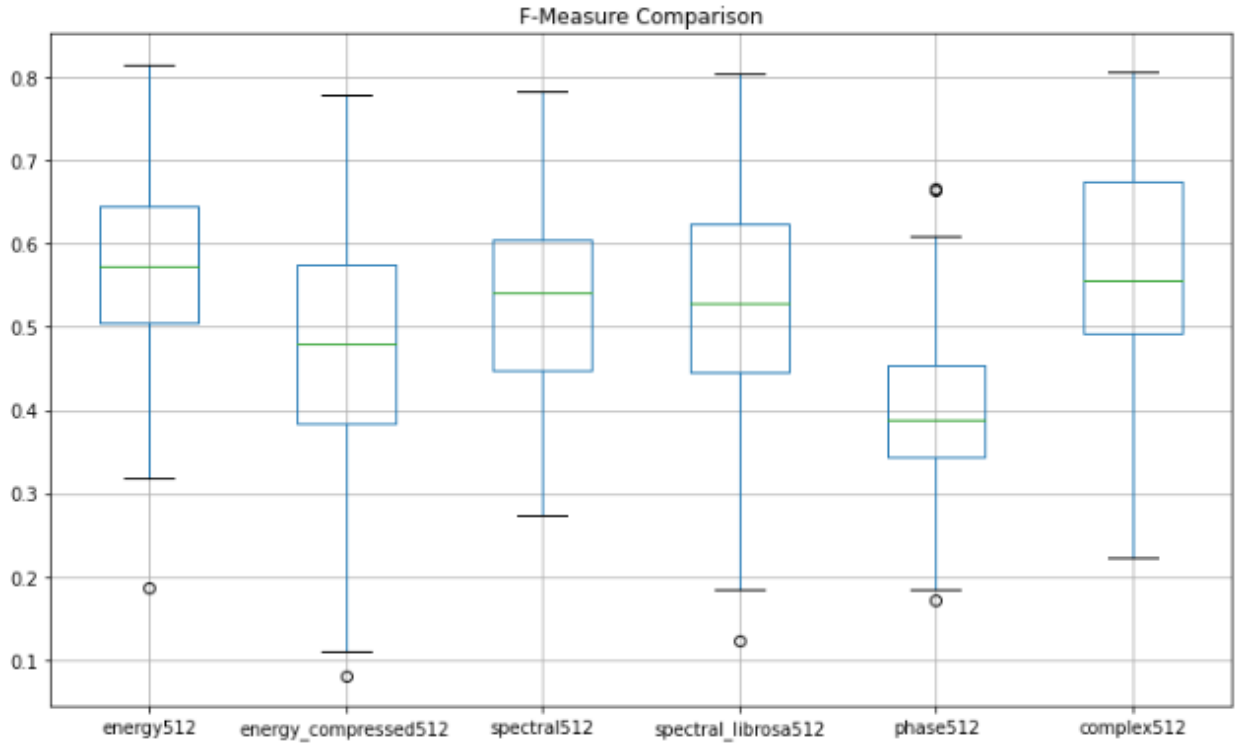

Figura 3.3: *Distribuição de F-Measure para cada função de novidade*

3.1.8 Combinando Classificadores

É possível enfrentar o problema de detecção de onsets como um problema de classificação: para cada janela de tempo de um sinal, decidir se existe um onset ou não. Por conta disso, faz sentido pensar em combinações de classificadores.

Por exemplo, considere dois classificadores: C1 e C2. Se C1 possui valor alto de Recall mas baixo Precision, teríamos um classificador que identifica muitos falso-positivos, isto é, classifica como onset algo que não é. Essa performance pode ser melhorada através da combinação com outro classificador C2 que tenha maior Precision, gerando um novo classificador C3 dado por:

$$C3(t) := C1(t) \wedge C2(t)$$

Ou seja, C3 só detectaria um onset num dado intervalo de tempo se tanto C1 como C2 detectassem esse onset. Da mesma forma, é possível compor classificadores utilizando outros operadores booleanos como $\vee$.

Repetindo as análises da seção anterior para os valores de Precision e Recall, obteve-se, para Precision 3.4, a melhor média de 56,2%, utilizando função de novidade espectral. Para Recall 3.5, a melhor média foi de 71,2% utilizando função de novidade complexa.

Interessante notar então que os valores de Precision são muito piores que os de Recall: Ou seja, temos muito falso-positivos. Com base nisso, foram testados mais 4 classificadores:

- $complex\_and\_spectral := complex512 \wedge spectral512$
- $energy\_and\_spectral := energy512 \wedge spectral512$
- $energy\_and\_complex := energy512 \wedge complex512$

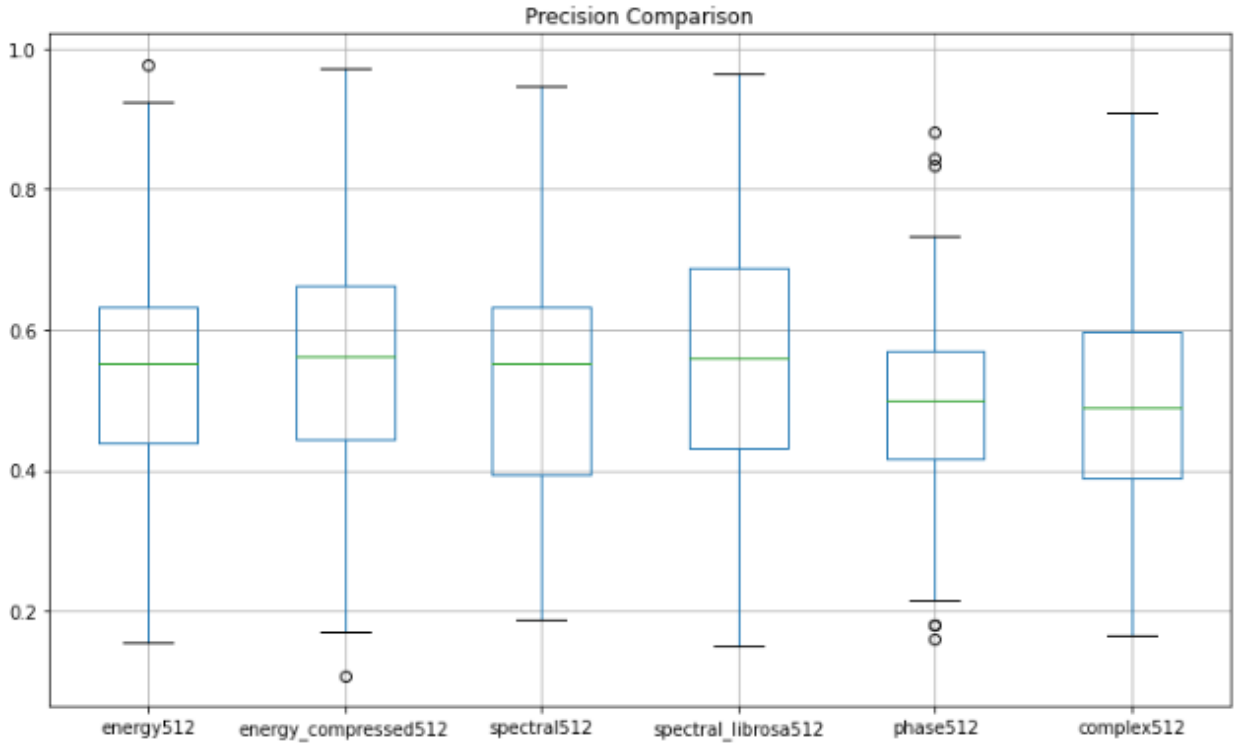

Figura 3.4: *Distribuição de Precision para cada função de novidade*

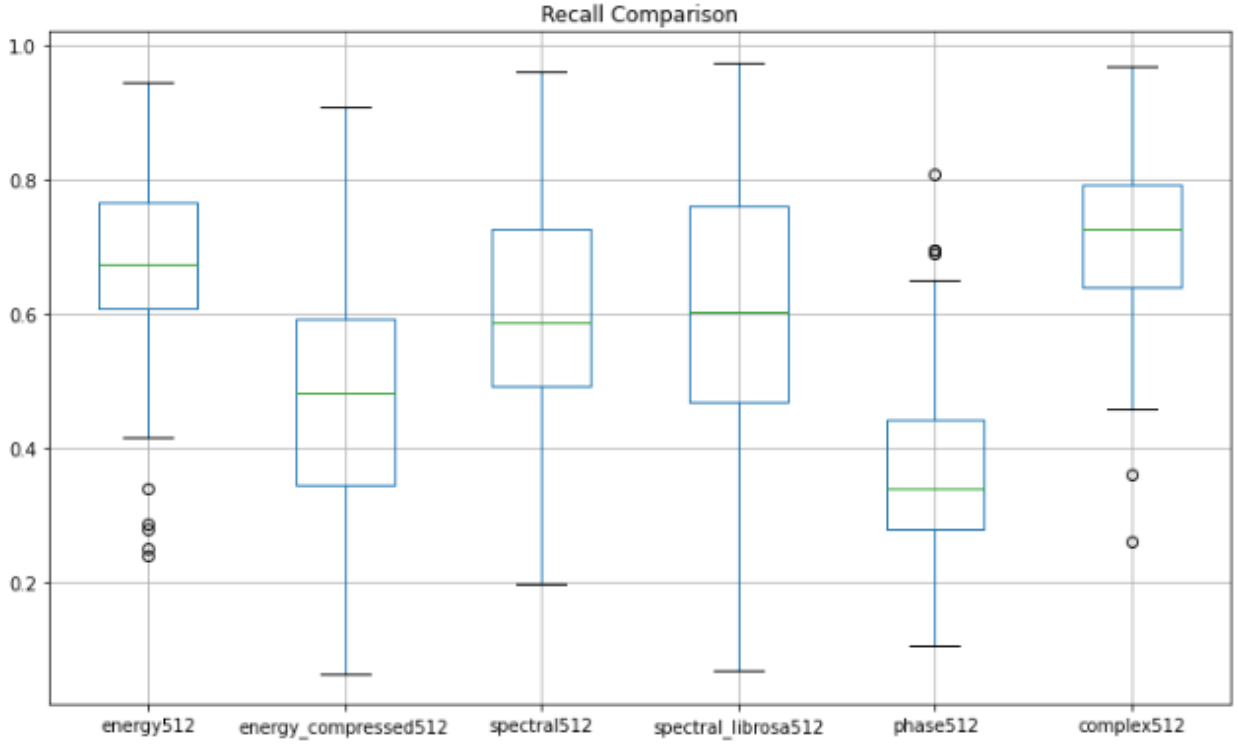

Figura 3.5: *Distribuição de Recall para cada função de novidade*

- $energy\_and\_complex\_and\_spectral := energy512 \wedge complex512 \wedge spectral512$

3.2 Resultados Finais

O classificador que obteve melhores avaliações foi o $energy512 \wedge complex512$ (label energy-complex-intersect) 3.6, com média de 56,5%, inferior ao melhor resultado dos classificadores separados. Para fins de investigação, decidiu-se, então, testar mais quatro classificadores:

- $complex\_or\_spectral := complex512 \vee spectral512$
- $energy\_or\_spectral := energy512 \vee spectral512$
- $energy\_or\_complex := energy512 \vee complex512$
- $energy\_or\_complex\_and\_spectral := energy512 \vee complex512 \vee spectral512$

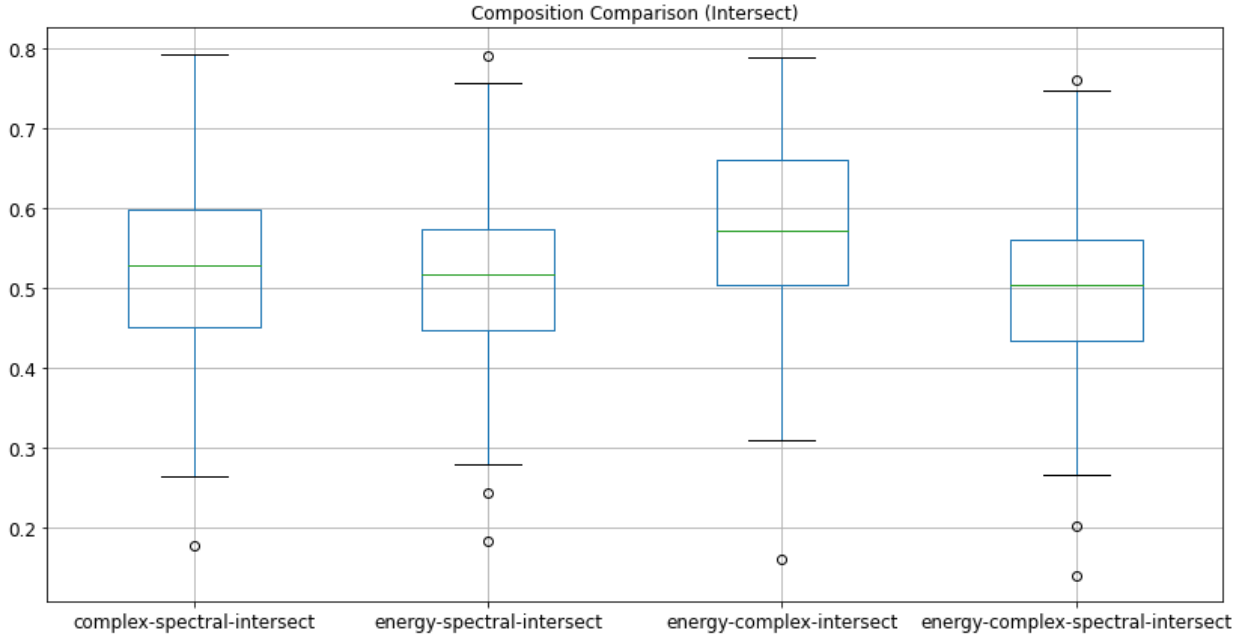

Figura 3.6: *Distribuição de F-measure para cada intersecção de classificadores*

A melhor média de F-measure obtida pelos classificadores formados pela união de outros classificadores foi de 60% 3.7, através do classificador $energy512 \vee complex512$ (label energy-complex-join).

Conclui-se portanto que a composição de classificadores possibilitou um aumento pouco expressivo da performance do algoritmo de detecção de *onsets*, indo de 57,4% para 60%.

3.3 O Experimento: *Postmortem*

Com o objetivo de entender melhor a causa dos resultados pouco satisfatórios da seção anterior, resolveu-se investigar de forma mais aprofundada a relação entre os *beatmaps* de *ground truth* e os *beatmaps* gerados para duas músicas: A primeira, de *id* 874856, cuja medida de *F-measure* foi uma das maiores obtidas (81,2%), e a segunda, de *id* 772255, cuja medida *F-measure* foi uma das piores (44,72%).

Para a primeira música, constatou-se que a **dificuldade** do *beatmap* de *ground truth* contribuiu muito para a boa performance dos algoritmos, enquanto que para a segunda, verificou-se que a restrição do domínio de frequências analisados nas funções de novidade espectral e complexas também contribuiu para resultado melhor.

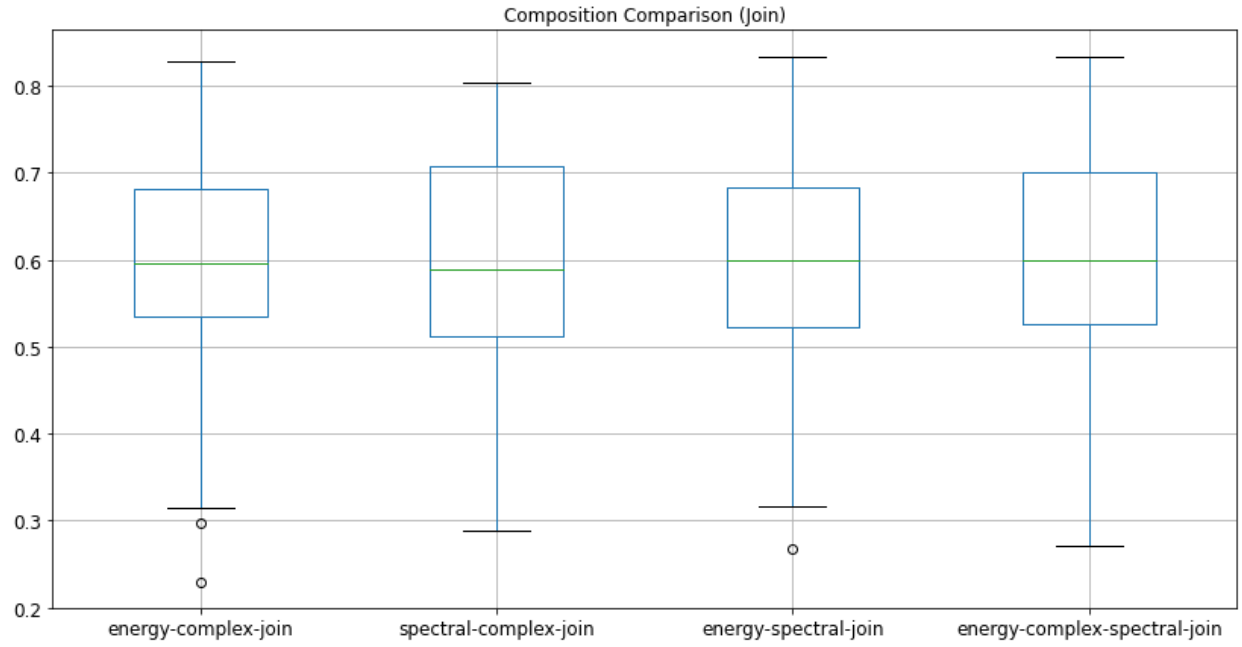

Figura 3.7: *Distribuição de F-measure para cada união de classificadores*

3.3.1 Influência da Dificuldade

No jogo de ritmo osu! uma mesma música pode ter diversas **versões** de *beatmaps* diferentes, cada uma delas associada a uma dificuldade. Porém, diferente dos jogos de ritmos tradicionais, as dificuldades em osu! não seguem um padrão intuitivo, como Fácil, Normal e Dificíl. O responsável pelo *beatmap* pode nomear a dificuldade da maneira que desejar. Por isso, algumas músicas, por exemplo, acabam tendo sua maior dificuldade com nomes como "*Extreme*", "*Insane*", *Advanced* ou *"Dangerous"*.

Na composição do *dataset* para este experimento, tomou-se o cuidado de selecionar apenas *beatmaps* aprovados pela comunidade, isto é, que foram avaliados de forma positiva por seus jogadores, presumivelmente porque a sincronia com a música foi considerada adequada. Porém, na hora de selecionar as dificuldades, escolheu-se sempre aquelas que tivessem a palavra "*Hard*", ou, quando não existisse, a palavra "*Normal*". Isso significa que a dificuldade dos *beatmaps* desse *dataset* não é nem um pouco uniforme 3.8.

Para a música de *id* 874856, cuja dificuldade usada foi a maior possível (aqui denominada *Very Hard*) foi testado então a função de novidade **energética**, que havia obtido o melhor desempenho, em uma versão de menor dificuldade (aqui denominada *Normal*). O resultado foi uma queda acentuada de performance, com valor de *F-measure* caindo para 65,09% 4.11.

3.3.2 Influência da Faixa de Frequências

Os algoritmos utilizados na seção anterior têm a finalidade de resolver o problema de detecção de *onsets*, sem distinguir o instrumento de origem. Para músicas polifônicas, esperar que o jogador realize uma ação para cada *onset* produzido por cada *instrumento* tornaria o jogo muito difícil, devido a enorme quantidade de *onsets* em uma música comum e o tempo curto entre eles. Por isso, jogos de ritmo costumam seguir apenas a **melodia principal**, que normalmente é a voz em músicas com vocais, mas que também pode ser tocada por outros instrumentos.

Para a música de *id* 772255, através da escuta do áudio sobreposto por uma sonificação em cliques nos *onsets* apontados pelo *beatmap* de *groundtruth*, determinou-se que os *onsets*

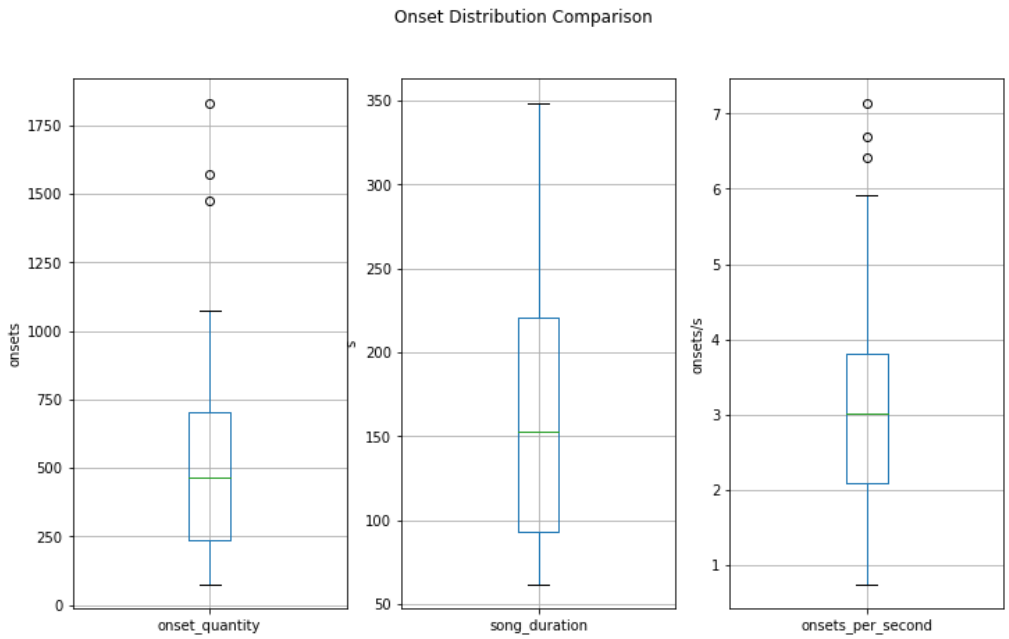

Figura 3.8: *Distribuição de onsets: o primeiro boxplot contém dados sobre a quantidade de onsets em cada música; o segundo contém a distribuição das durações e o último o número médio de onsets por segundo.*

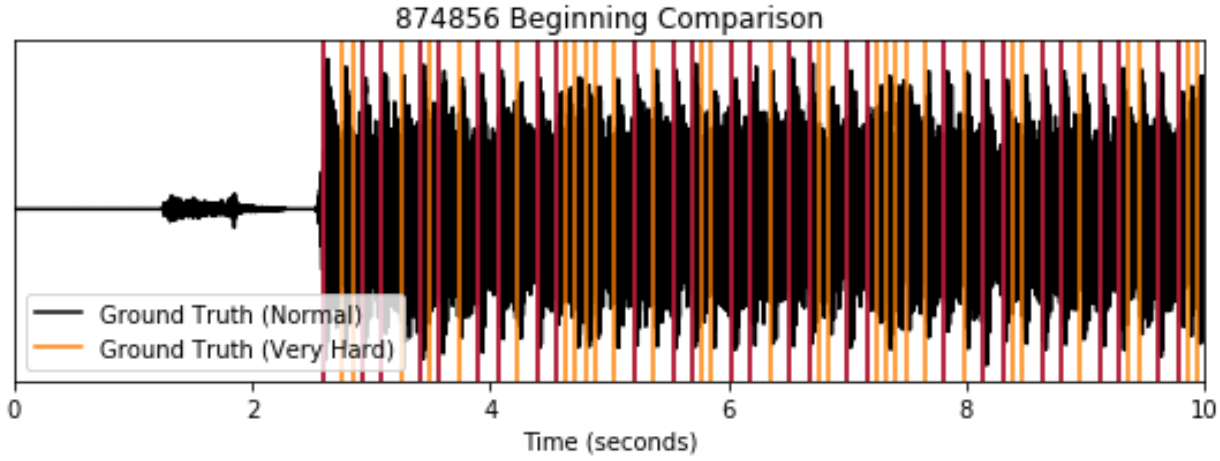

Figura 3.9: *Comparação entre os ground truths das dificuldades Normal e Very Hard para o início da música de id 874586. É notável a queda no número de onsets.*

correspondiam quase unicamente aos *onsets* produzidos pela **voz** da cantora.

A música em questão possui um arranjo simples: composto apenas por uma bateria, um violoncelo, um piano e uma voz. Com a finalidade de detectar apenas *onsets* produzidos pela voz, foram modificadas as funções de novidade **espectral** e **complexa** para detectar mudanças apenas numa faixa específica de frequências.

Encontrar essa faixa foi um processo tanto investigativo quando de tentativa e erro: inicialmente analisou-se o espectro da música em um analisador *online*[6], de forma a encontrar a faixa de frequências no qual a voz se situava 3.10 e a partir disso algumas faixas de frequências foram testadas.

A faixa que proporcionou melhor resultado foi a de 10800 a 11200 Hz, com *F-measure* de

[6] https://academo.org/demos/spectrum-analyzer/

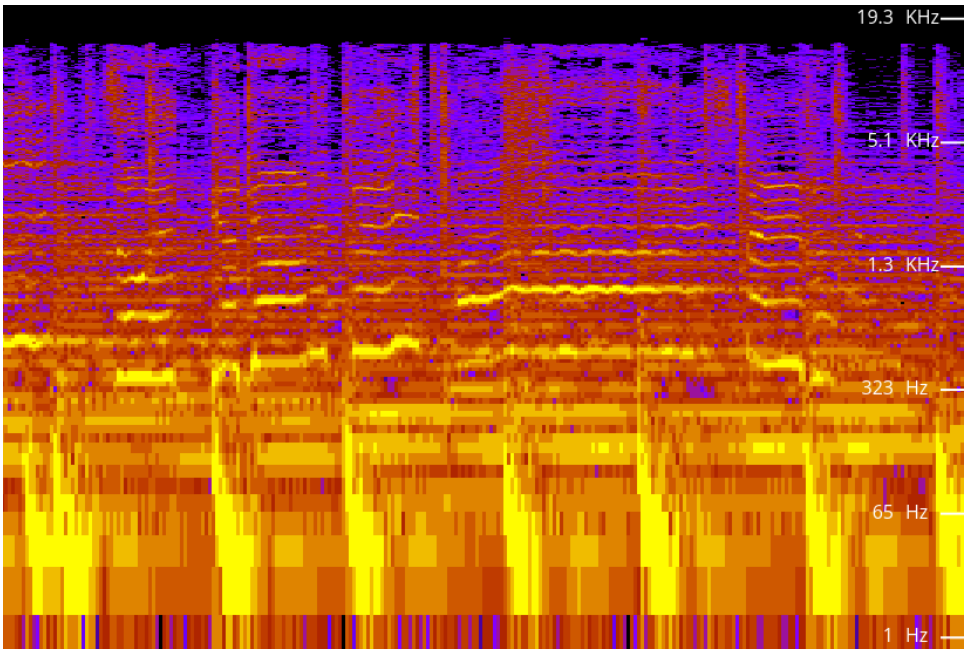

Figura 3.10: *Espectrograma de um trecho do refrão da música de id 772255. Na faixa entre 10 e 100 Hz concentra-se a energia da bateria, entre 200 e 800 Hz o piano, e a voz principalmente entre 400 e 4000 Hz.*

67,35%. Tal faixa situa-se muito acima do campo de frequências fundamentais produzidas pela voz feminina (entre 200 e 1000 Hz)[7], porém verifica-se no espectrograma a existência de alguns **harmônicos** nessa faixa. Frequências abaixo desse limiar faziam com que *onsets* produzidos pelo cimbal da bateria e por notas mais agudas do piano também fossem detectados 3.11.

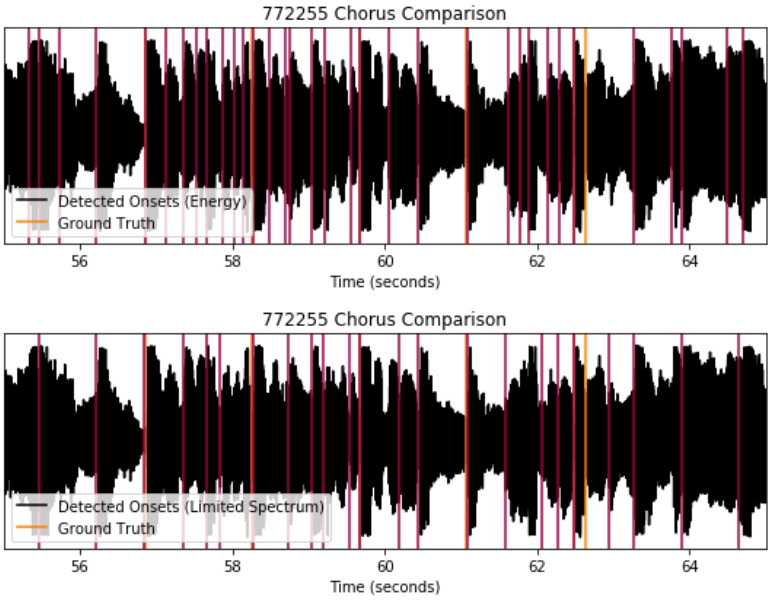

Figura 3.11: *Comparação entre o ground truth e onsets detectados usando função de novidade energética e espectral com banda limitada. Os onsets de instrumentos percussivos que eram detectados antes não estão mais presentes.*

[7] https://en.wikipedia.org/wiki/Vocal_range

Capítulo 4

Desenvolvimento dos Protótipos

Com o objetivo de validar o algoritmo de *beatmapper* desenvolvido anteriormente e explorar novas formas de jogabilidade baseadas em música, foram desenvolvidos dois protótipos: o primeiro é um jogo de ritmo no estilo *osu!*, seguindo os moldes de um jogo de ritmo tradicional, enquanto o segundo não se encaixa nesses moldes.

Ambos protótipos foram desenvolvidos utilizando o motor de jogos Godot, software livre, de código aberto e em ativo desenvolvimento [1]. Essa escolha se deu por dois fatores: primeiro, familiaridade da autora com a engine, que já a utilizou em projetos anteriores; segundo, existência de um plugin Leblond (2020) que permite o uso de bibliotecas *python*, linguagem utilizada na fase de construção do *beatmapper*, portanto possibilitando o reaproveitamento de código.

Com os protótipos prontos, colheu-se opiniões a respeito da jogabilidade dos mesmos, com o propósito de avaliar e comparar suas qualidades.

4.1 *Star Connect*

O conceito por trás do Star Connect é o de tratar cada nota de uma música como uma estrela no céu. As ações do jogador "conectam"essas estrelas, formando uma bela constelação ao final da música.

4.1.1 Arquitetura

Para importar músicas, o jogador deve colocar seus arquivos .mp3 numa pasta chamada "Songs". A decisão de usar arquivos .mp3 se deu por conta dos metadados normalmente contidos em arquivos desse tipo, com informações sobre título da música, artista, álbum, etc.

Para o carregamento das músicas, decidiu-se por uma aborgadem "preguiçosa", sendo cada música processada apenas quando o jogador selecioná-la. Ao carregar uma música pela primeira vez, a mesma precisa se processada pelo *beatmapper*, o que pode levar tempo considerável. Primeiro, extrai-se as informações dos metadados do .mp3 para identificar a música. Depois, utiliza-se do utilitário FFMPEG [2] para converter o .mp3 para um .ogg, pois a Godot não consegue tocar arquivos .mp3. Finalmente, utiliza-se a biblioteca *librosa* em conjunto com os códigos da seção anterior para extrair os *onsets*, armazenando seus tempos numa lista, dependendo da dificuldade selecionada.

[1] https://godotengine.org
[2] https://ffmpeg.org/

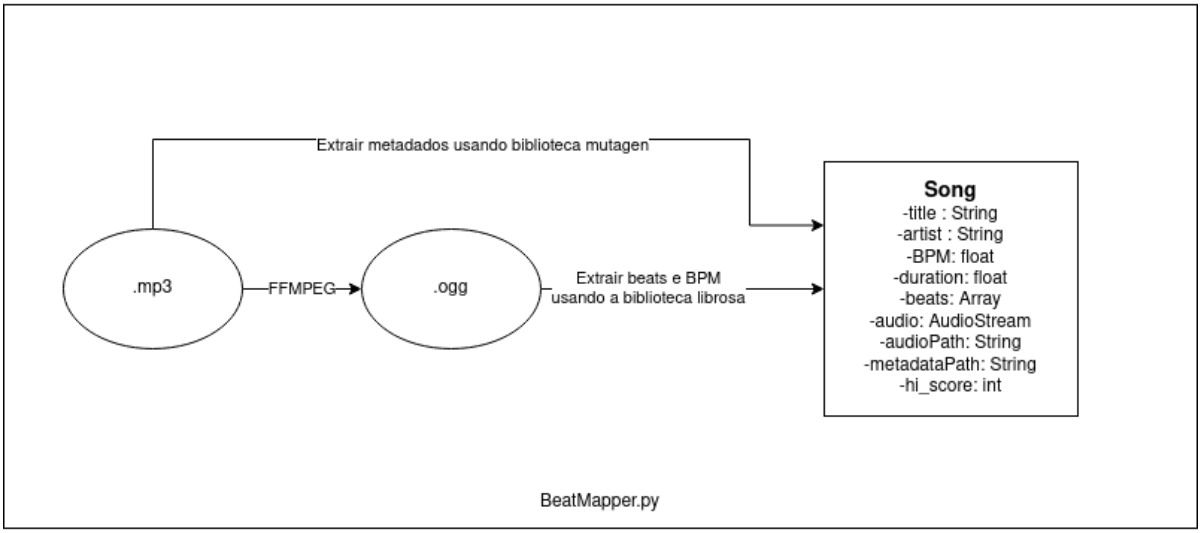

Figura 4.1: *Funcionamento do beatmapper e dados de uma música*

4.1.2 Jogabilidade

A tela inicial do jogo possui algumas informações sobre como o jogo deve ser jogado no menu "How to Play"4.2. Ao apertar "Play", o jogador se depara com um menu de seleção de músicas. Selecionando uma música, o cenário se transforma num céu inicialmente vazio.

O objetivo é simples: estrelas azuis e vermelhas surgem na tela, inicialmente com bordas pretas largas ao seu redor. À medida que o tempo passa as bordas vão diminuindo, até coincidirem com o contorno estrela, momento em que o jogador deve apertar um botão (A se for vermelha, S se for azul). O timing em que essas ações são realizadas coincidem com os as notas do *beatmap* da música.

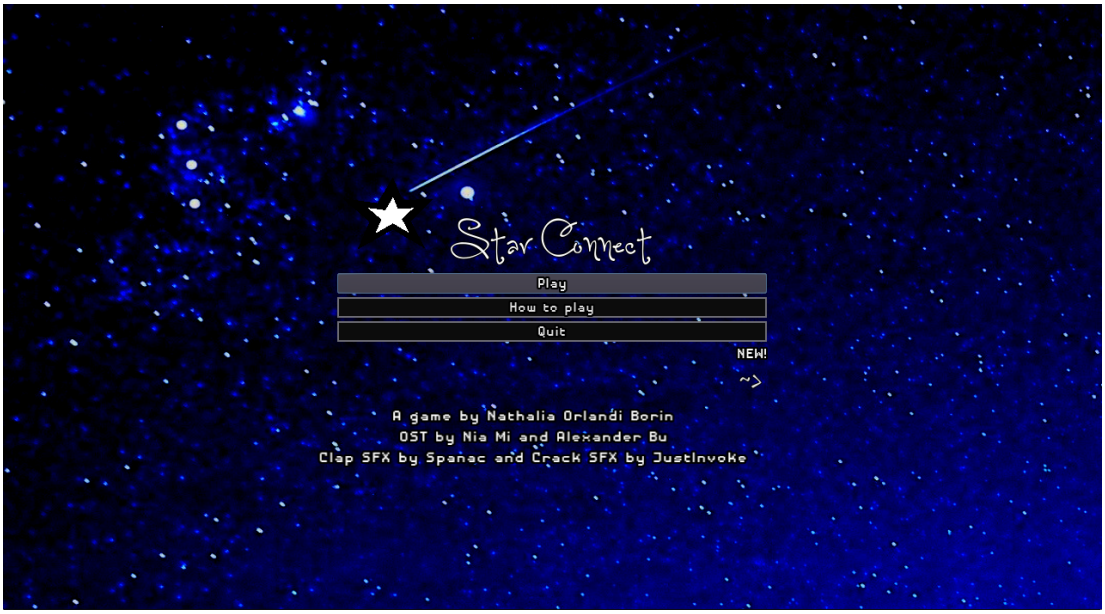

Figura 4.2: *Menu principal do Star Connect*

Para cada nota que o jogador acerta, cria-se uma linha branca entre a última nota acertada e a atual, de modo que as ações do jogador vão criando um caminho de estrelas pelo céu, ou uma constelação. Ao fim da música, é apresentado ao jogador uma imagem do percurso realizado em sua totalidade.

A performance do jogador é calculada com base na diferença entre o tempo de sua ação e o tempo da nota correspondente no *beatmap*. Os resultados possíveis em ordem crescente de performance são *MISS*, *BAD*, *GOOD*, *GREAT* e *EXCELLENT*. Cada nota acertada contribui com a pontuação, sendo a contribuição maior quanto mais acurada ação do jogador.

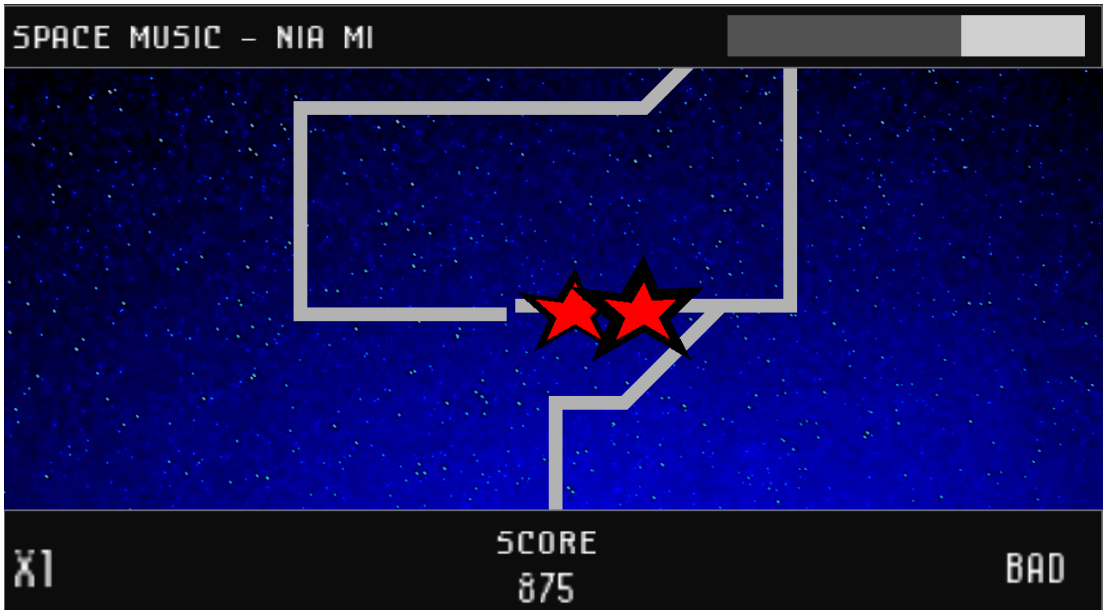

Figura 4.3: *Jogabilidade do Star Connect: Na parte superior da tela encontra-se informações sobre a música tocada e sua duração, enquanto na parte inferior observa-se informações referente a performance do jogador.*

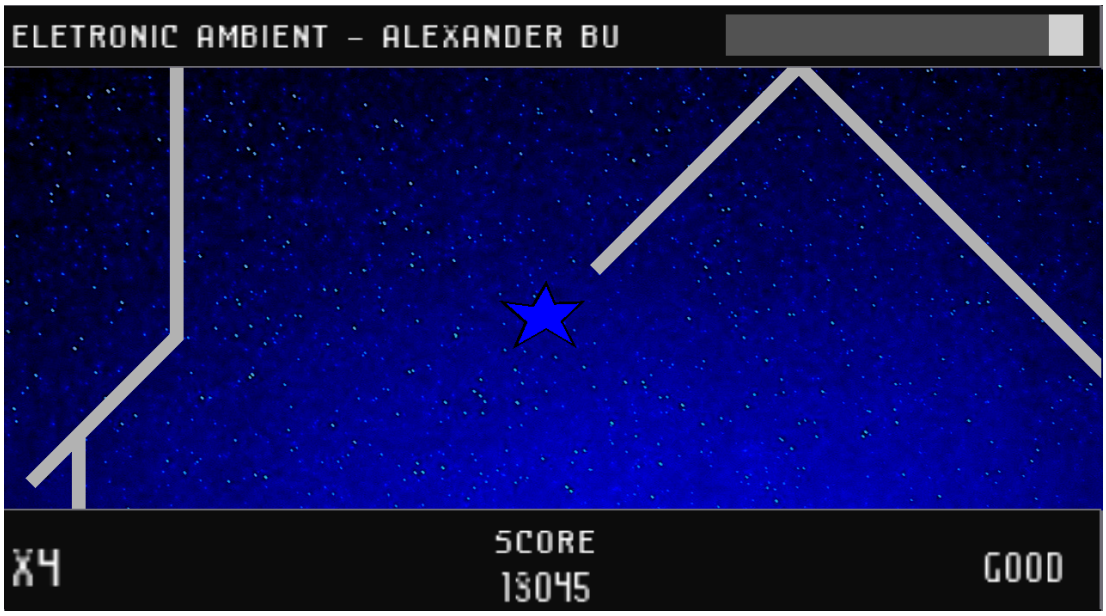

Figura 4.4: *Jogabilidade do Star Connect: No canto inferior esquerdo, o multiplicador X4 indica que o jogador acertou diversas notas seguidas, sem errar, recebendo mais pontos com as notas que acerta.*

Este protótipo possui 4 opções de dificuldade: Easy, Normal, Hard e Very Hard, sendo seus mapas gerados da seguinte forma:

1. **Easy:** $Onsets_{energy} \wedge Onsets_{spectral}$ - Intersecção garante menor número de notas para esta dificuldade.

2. **Normal:** *Beats* - Dificuldade padrão, composta por beats em intervalos regulares.

3. **Hard:** $Onsets_{energy} \vee Onsets_{complex}$ - União entre um classificador que produz bastante *onsets* (*Energy*) com um que produz poucos (*Complex*)

4. **Very Hard:** $Onsets_{energy} \vee Onsets_{spectral}$ - União entre dois classificadores que produzem bastante *onsets*.

4.2 *Star Shooter*

4.2.1 Arquitetura

Para importar músicas, da mesma forma como no protótipo anterior, o jogador deve colocar seus arquivos .mp3 numa pasta chamada "Songs". Sendo os mapas gerados segundo a mesma abordagem "preguiçosa"descrita anteriormente.

4.2.2 Jogabilidade

A tela inicial do jogo possui algumas informações sobre como o jogo deve ser jogado no menu "How to Play"4.5. Ao apertar "Play", assim como no protótipo anterior, o jogador se depara com um menu de seleção de músicas. Selecionando uma música, o cenário se transforma num céu com uma nave controlada pelo jogador no centro da câmera.

Figura 4.5: *Menu principal do Star Shooter*

O objetivo do jogador é sobreviver: para isso, ele deve manobrar sua nave de forma a desviar das estrelas que estão "caindo"do topo da tela. Cada estrela surge no tempo de uma nota do *beatmap* gerado para a música 4.6.

O jogador possui um número limitado de vidas e ganha mais pontos de acordo com o tempo que sobrevive e o número de vidas restantes. Com o objetivo de tornar o jogo mais interessante, optou-se também por adicionar uma mecânica de risco e recompensa, na qual o jogador ganha mais pontos por "quase ser atingido", isto é, ao se aproximar o suficiente de uma estrela cadente sem realmente colidir com a mesma 4.7.

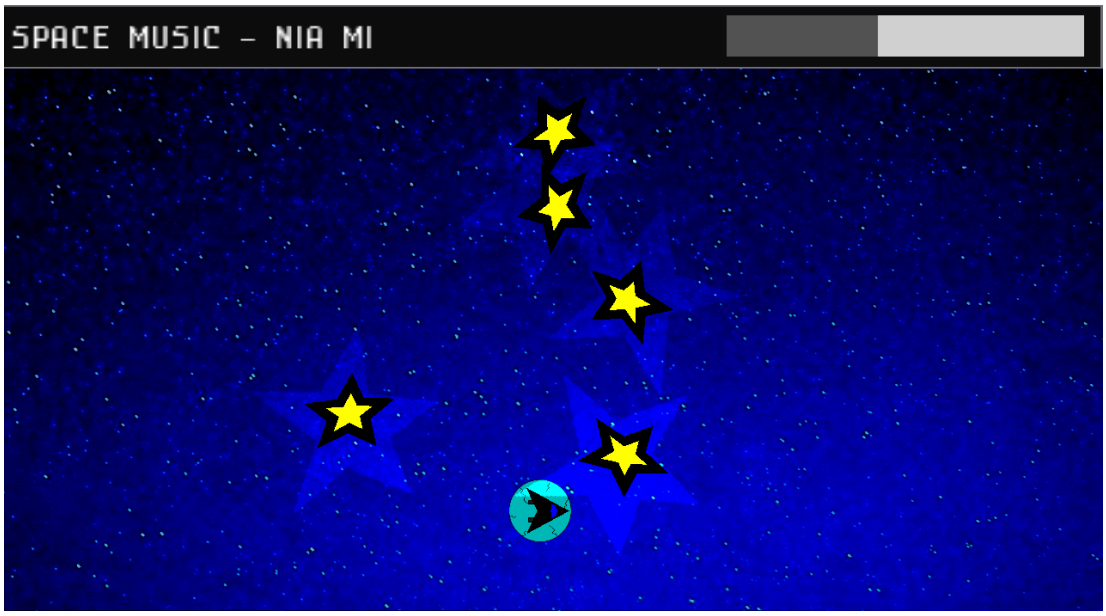

Figura 4.6: *Jogabilidade do Star Shooter: o escudo rachado indica que o jogador já perdeu uma de suas vidas. No canto superior esquerdo da tela se encontra o nome da música e no canto superior direito uma barra que indica seu progresso.*

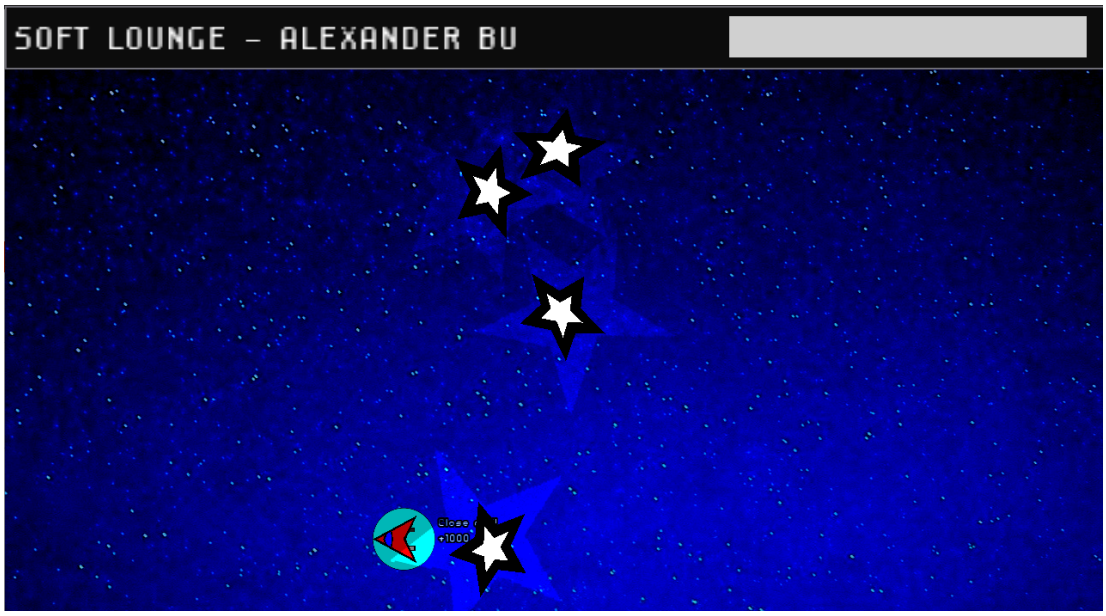

Figura 4.7: *Mecânica de risco e recompensa: na imagem, o jogador ganhou um bônus de 1000 pontos por passar perto de uma estrela*

Assim como o anterior, este protótipo possui 4 opções de dificuldade: Easy, Normal, Hard e Very Hard, sendo seus mapas gerados da seguinte forma:

1. **Easy:** $Beats$
2. **Normal:** $Onsets_{spectral}$
3. **Hard:** $Onsets_{spectral} \vee Onsets_{complex}$
4. **Very Hard:** $Onsets_{energy} \vee Onsets_{spectral}$

4.3 Resultados

A qualidade dos protótipos foi medida através de um questionário via Google Forms, respondido por 14 jogadores. As perguntas buscavam avaliar 5 fatores principais: experiência prévia dos jogadores com jogos de ritmo, usabilidade do jogo (dificuldade de executar o jogo, importar músicas e compreender as mecânicas principais), dificuldade do jogo (dificuldade de executar as mecânicas principais), diversão do jogo e sincronismo percebido, além de tentar encontrar possível correlação entre o conhecimento prévio de uma música e a performance do jogador na respectiva fase.

Buscou-se também identificar a aplicação dos algoritmos de detecção de *onsets* nas diferentes dificuldades de jogo, através de perguntas que avaliam as dificuldades melhor e pior avaliadas pelos jogadores.

A última pergunta do questionário visa também identificar a preferência dos jogadores por um dos protótipos.

4.3.1 Experiência com jogos de ritmo

Essa categoria possui apenas duas perguntas: a primeira visa avaliar a experiência do entrevistado com jogos de ritmo, categorizando-as em "Nunca joguei", "Joguei poucos", "Joguei alguns"ou "Jogo com frequência".

Interessante notar que 100% dos entrevistados já haviam jogado jogos de ritmo 4.8. O jogo mais conhecido dentre os entrevistados foi *Guitar Hero*, presente em 92% das respostas a pergunta sobre experiência anterior com jogos de ritmo. Outros jogos muito citados foram *Rock Band* e *osu!*.

Figura 4.8: *Dos respondentes, 28,6% jogam jogos de ritmo com frequência, 50% jogaram alguns e os demais jogaram poucos.*

4.3.2 Usabilidade

Essa categoria possui também duas perguntas: a primeira visa identificar possíveis problemas na hora de rodar o jogo e importar músicas, enquanto a segunda é uma avaliação do processo de importar músicas, categorizando-o numa escala que vai de "Muito Ruim"a "Muito Bom".

Todos os jogadores conseguiram rodar o jogo sem problemas, no entanto 21,4% reportaram problemas para importar músicas 4.9. Esses problemas se deram principalmente por

conta do uso de uma dependência externa (ffmpeg) para converter as músicas de .mp3 para .ogg. Adicionalmente, uma pessoa reportou problemas por conta da versão da biblioteca *numpy* instalada. Esse problema foi resolvido usando uma versão um pouco mais antiga do *numpy*.

Quanto ao processo de importar músicas, a média das avaliações foi 3 (Regular) 4.10. O principal problema citado foi lentidão na hora de carregar as músicas.

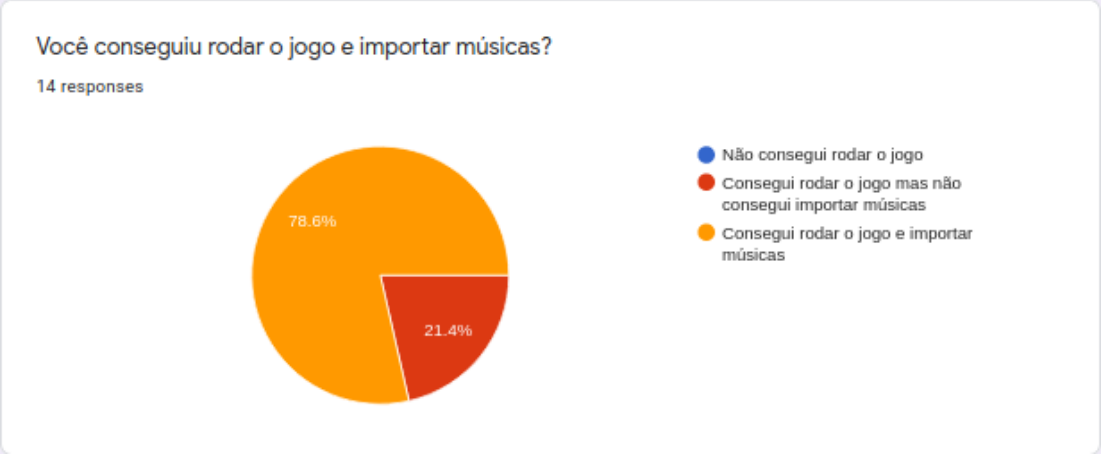

Figura 4.9: *78,6% dos jogadores conseguiram rodar o jogo e importar suas músicas e o demais 21,4% conseguiram rodar o jogo mas não conseguiram importar suas músicas.*

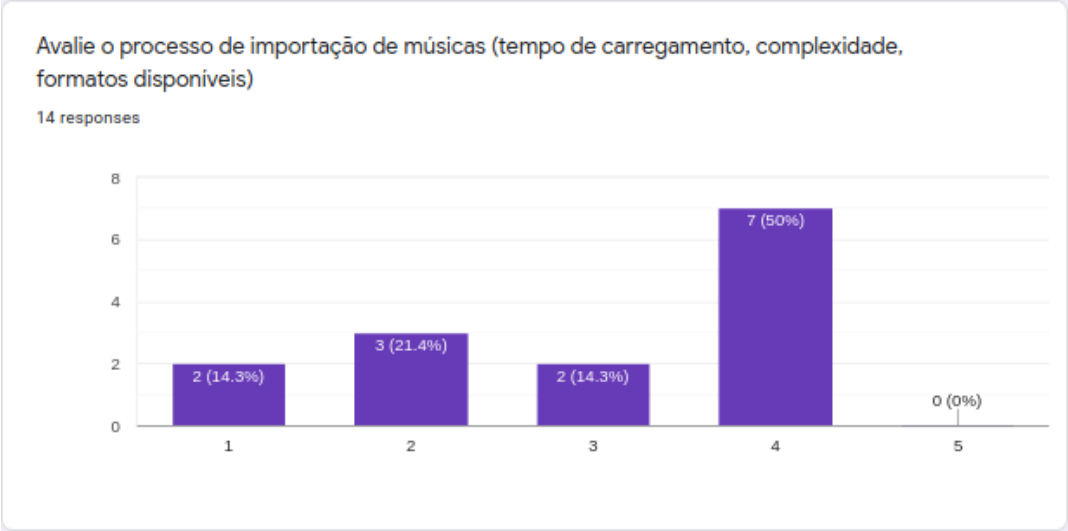

Figura 4.10: *Numa escala de 1 a 5, 14.3% classificaram o processo de importar músicas como 1 (Muiro Ruim), 21,4% como 2 (Ruim), 14,3% como 3 (Regular) e 50% como 4 (Bom).*

4.3.3 Dificuldade

Essa categoria possui duas perguntas para cada protótipo. A primeira busca avaliar a dificuldade geral reportada pelos jogadores ao testarem a dificuldade *"Normal"*. A segunda busca identificar os algoritmos mais e menos adequados para construção de *beatmappers* em cada protótipo, através do levantamento das dificuldades que os jogadores acharam mais e menos divertidas.

A dificuldade padrão *"Normal"* foi avaliada pelos jogadores numa escala de 1 (Muito Fácil) a 5 (Muito Difícil). Para o *Star Connect*, a dificuldade média foi de 3,35, o que está bem próximo de uma dificuldade Regular, o que é desejável. Já para o *Star Shooter* a variância das respostas foi maior, mas a média foi de 3,14 4.11.

A maior variância pode ser explicada pela má compreensão das mecânica de risco e recompensa, pois os jogadores que avaliaram o jogo como "Fácil"reportaram que para sobreviver, bastava ficar num dos cantos da tela, onde as estrelas não atingiam a nave. Isso é uma estratégia válida mas implica em pontuações mais baixas.

As maiores dificuldades encontradas pelos jogadores do *Star Connect* foram "Sobreposição de estrelas"(reportado por 53.8% dos jogadores), "Muitas notas na tela"(reportado pro 46.2% dos jogadores) e "Pouco tempo para reagir"(reportado por 38.5% dos jogadores).

Já as maiores dificuldades encontradas pelos jogadores do *Star Shooter* foram "Controle da espaçonave insatisfatório"(reportado por 57.1% dos jogadores) e "Hitboxes imprecisas"(reportado por 42.9% dos jogadores).

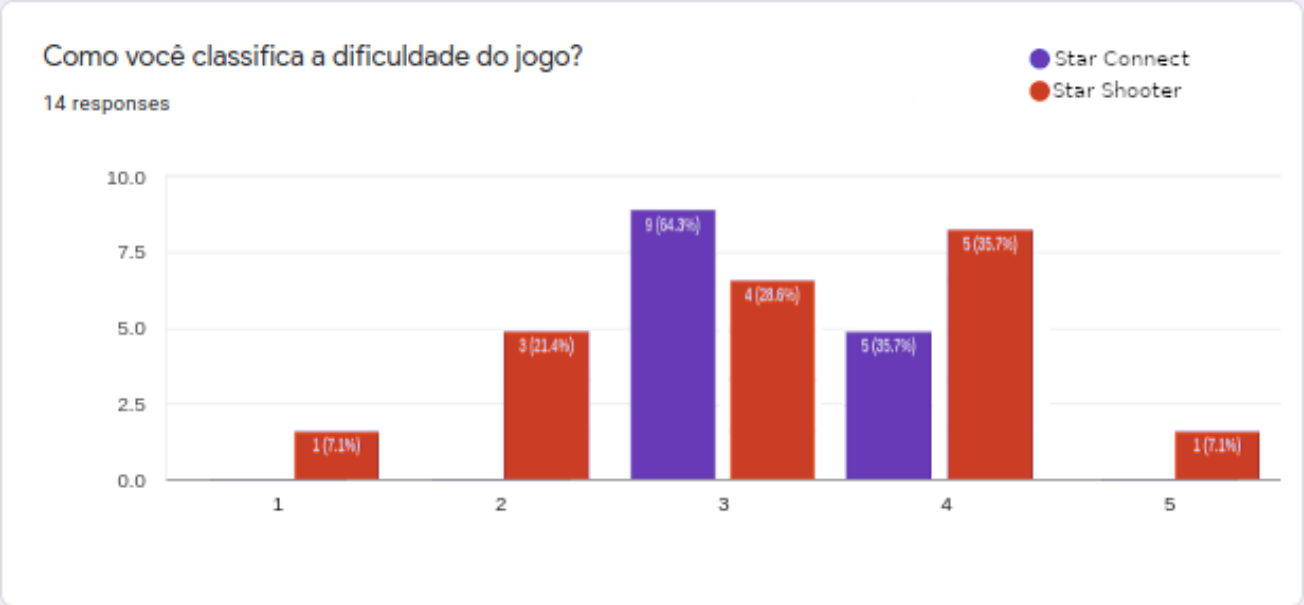

Figura 4.11: *Comparação entre o nível de dificuldade "Normal"dos dois protótipos*

As dificuldades melhores avaliadas no *Star Connect* foram *Hard*, *Normal* e *Very Hard*, preferidas por 57,1%, 28,6% e 14,3% dos jogadores, respectivamente. Já as, piores avaliadas foram *Easy*, *Very Hard* e *Normal*, avaliados negativamente por 78,6%, 14,3% e 71% 4.12 dos jogadores.

As dificuldades melhores avaliadas no *Star Shooter* foram *Hard*, *Normal* e *Very Hard*, preferidas por 64,3%, 28,6% e 7,1% dos jogadores, respectivamente. Já as, piores avaliadas foram *Easy* e *Normal*, avaliados negativamente por 92,9% e 7,1% dos jogadores 4.13.

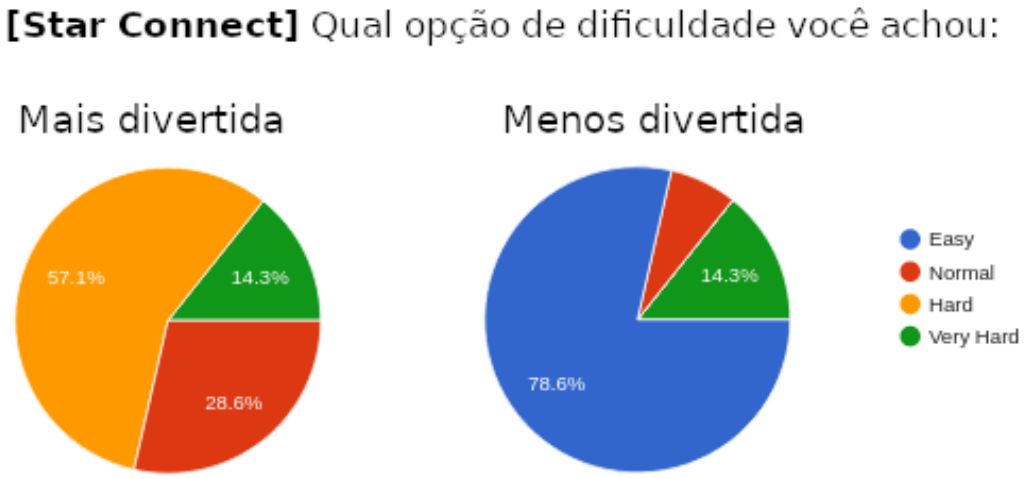

Figura 4.12: *Dificuldades melhor e pior avaliadas pelos jogadores do Star Connect*

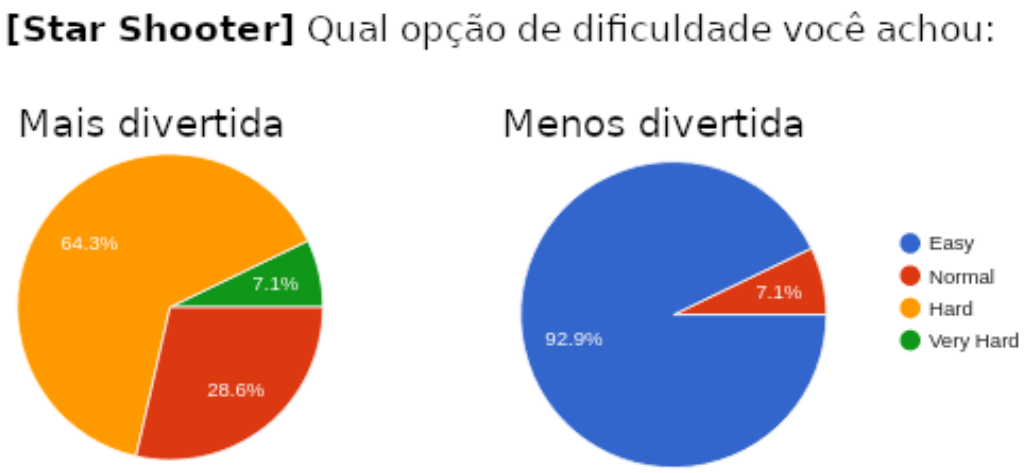

Figura 4.13: *Dificuldades melhor e pior avaliadas pelos jogadores do Star Shooter*

4.3.4 Diversão

Essa categoria possui apenas uma pergunta por protótipo e busca avaliar a diversão dos jogadores numa escala de 1 (Nem um pouco divertido) até 5 (Muito divertido). Nessa escala, a média do *Star Connect* foi de 3,85 e do *Star Shooter* foi 2,92 4.11. Existe aqui uma correlação entre jogadores que reportaram baixa dificuldade e jogadores que reportaram pouca diversão, possivelmente pelo motivo citado anteriormente de má compreensão das mecânicas de risco e recompensa.

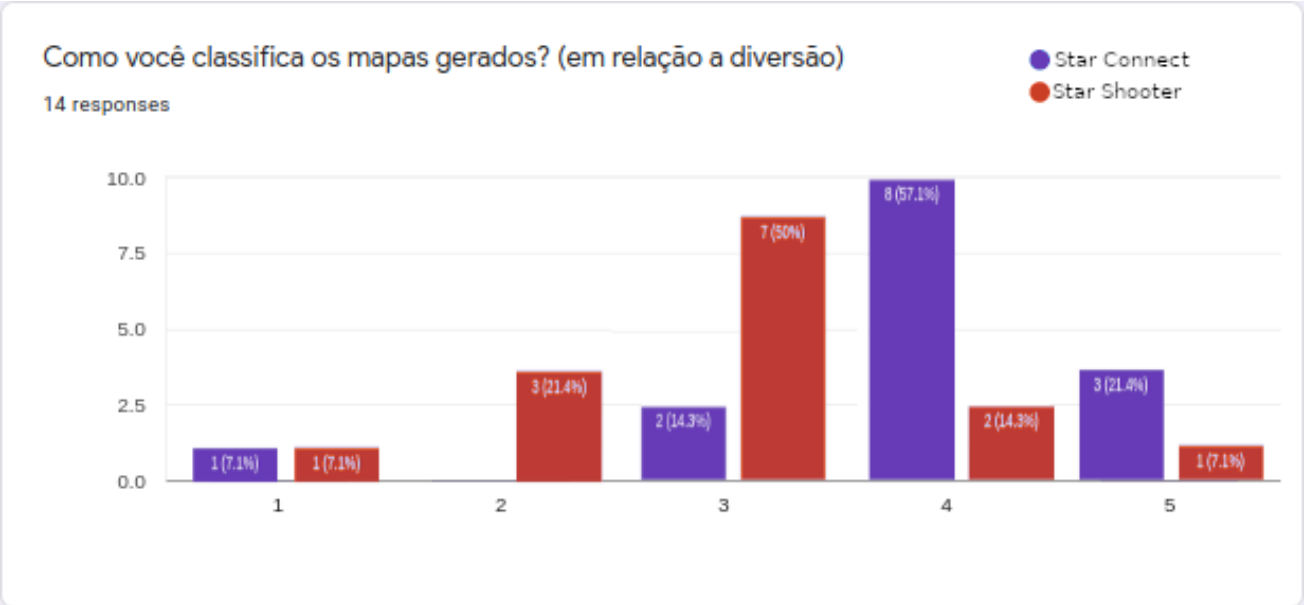

Figura 4.14: *Comparação entre a diversão reportada pelos jogadores ao jogar os dois protótipos.*

4.3.5 Sincronia

Essa categoria possui apenas uma pergunta por protótipo e busca avaliar a sincronia percebida pelos jogadores entre suas ações e a música do jogo numa escala de 1 (Nem um pouco sincronizado) até 5 (Muito sincronizado). Nessa escala, a média do *Star Connect* foi de 3,5 e do *Star Shooter* também foi de 3,5 4.15.

Numa escala de 1 a 5, uma média de 3,5 foi considerado um valor satisfatório para os protótipos, mas esse valor pode certamente ser melhorado usando algoritmos melhores para o *beatmapper* e também aumentando o número de elementos dentro do jogo que são influenciados pela música.

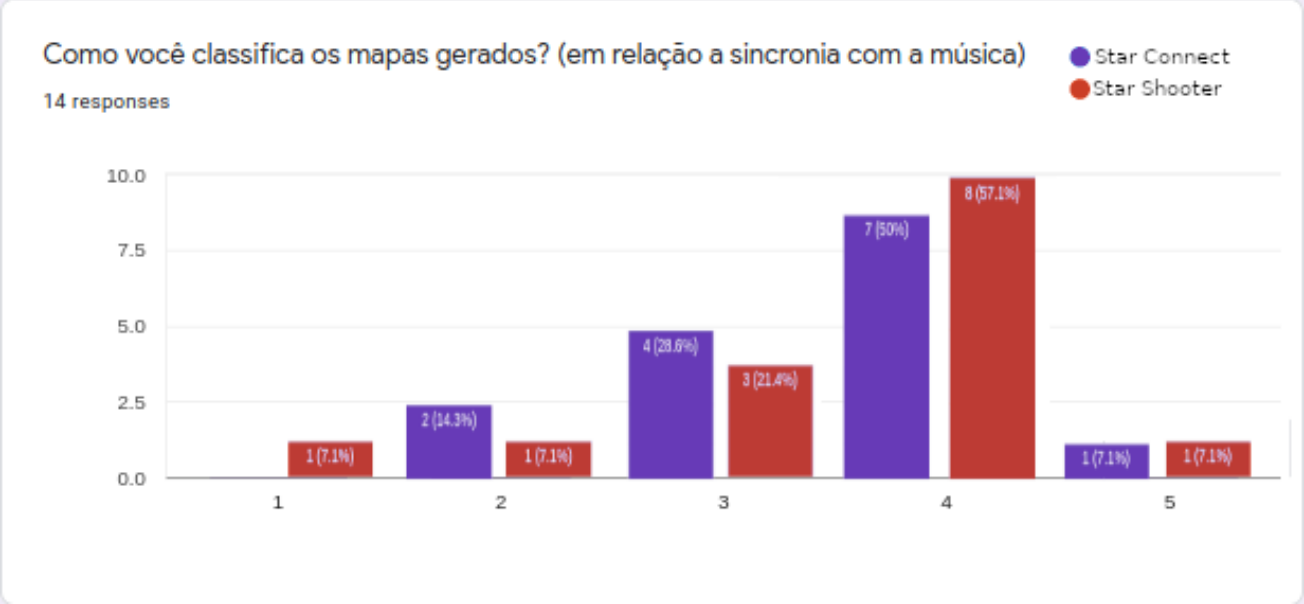

Figura 4.15: *Comparação entre a sincronia reportada pelos jogadores ao jogar os dois protótipos.*

4.3.6 Comparação

A pergunta final busca avaliar a preferência dos jogadores entre os dois protótipos desenvolvidos. O *Star Connect* aparece como vencedor claro, com 78,6% dos votos 4.16.

Pode ser especulada aqui alguma correlação entre a maioria dos jogadores já jogaram e estarem acostumados com jogos de ritmo tradicionais e a preferência pelo protótipo mais tradicional.

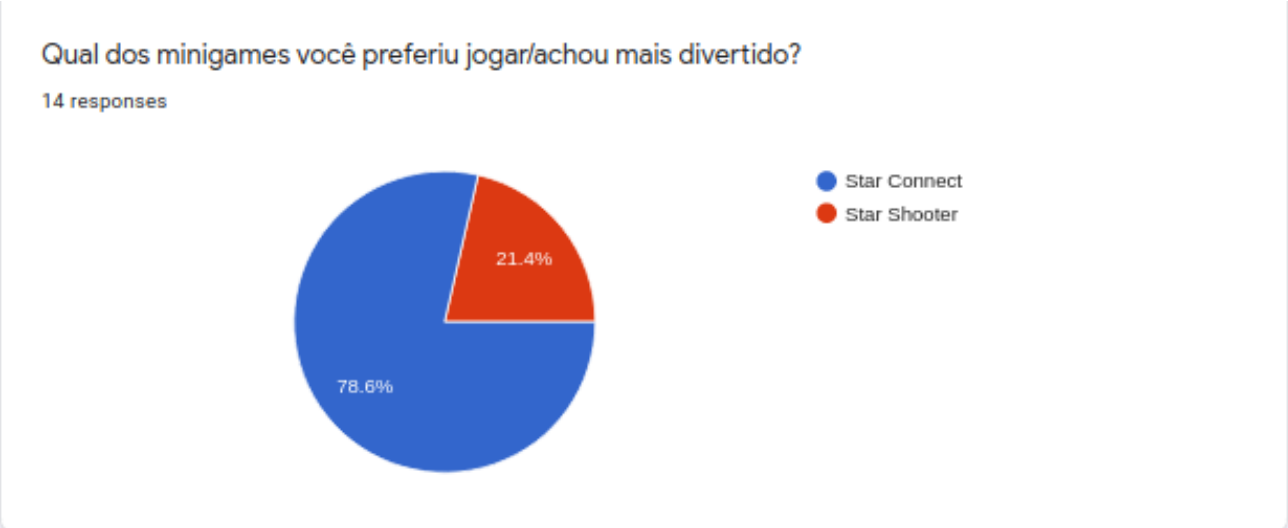

Figura 4.16: *78,6% dos jogadores preferiram o Star Connect e 21,4% acharam o Star Shooter mais divertido.*

4.4 *Postmortem*

Após análise das respostas aos questionários, foram elencados alguns possíveis pontos de melhoria para uma eventual versão final dos jogos *Star Connect* e *Star Shooter*.

Uma primeira iniciativa que vale para ambos os protótipos seria **melhorar processo de build e release:** O principal problema decorrente de utilizar o plugin para rodar *python* na Godot está na dificuldade em exportar o jogo para plataformas diferentes. O plugin empacota junto ao jogo um versão de *python* específica, com executável e pacotes instalados próprios. Isso significa que, para instalar as bibliotecas necessárias, é preciso instalá-las para cada sistema operacional separadamente. Por conta dessa limitação, não foi possível exportar o jogo para a plataforma OSX pois não havia um computador MAC disponível para isso.

As demais melhorias são específicas para cada um dos protótipos, portanto estão em seções separadas:

4.4.1 Star Connect

- **Melhorar *beatmaps* de algumas dificuldades:** As dificuldades *Easy* e *Very Hard* foram as piores avaliadas por diferentes motivos: A *Easy* gerava poucas notas, às vezes deixando intervalos de mais de 20 segundos entre as notas, tornando a experiência monótona para o jogador. Uma possível melhoria seria complementar o beatmap criado com os *beats* da música quando nenhum beat for detectado em grandes intervalos. Alternativamente, tomar como base um *beatmap* mais complexo, como do *Normal* ou *Hard* e remover algumas notas seguindo algum critério também é uma opção. Quanto

ao *Very Hard*, o problema é o oposto: o *beatmapper* gerou muitas notas em intervalos curtos de tempo, sendo muito difícil para o jogador executar todas as ações esperadas. Para esse caso, algum critério para eliminar notas com uma distância muito pequena entre si no tempo seria suficiente para melhorar a experiência de jogo.

- **Melhorar distribuição espacial e temporal das estrelas:** A reclamação mais frequente, sobreposição de estrelas pode ser resolvida usando um critério melhor de espacialização das estrelas, armazenando em memória quais posições já estão ocupadas para não repeti-las. Outra melhoria possível seria adicionar um diferencial visual na nota seguinte para ajudar o jogador a localizar mais rapidamente a próxima ação que deve executar. Da mesma forma, outro problema comum reportado (pouco tempo para reagir) poderia ser resolvido com um tempo maior entre o instante do surgimento de uma nota na tela e a ação esperada do jogador.

- **Pensar em alternativas independente de cores:** Fazer com que o jogador tenha que saber qual botao pressionar de acordo com a cor da estrela é algo prejudicial para pessoas que não conseguem diferenciar cores. Visando a inclusão dessas pessoas, poderiam ser utilizadas estrelas de formatos diferentes para diferenciar os botões.

4.4.2 Star Shooter

- **Melhorar comunicação da mecânica de risco e recompensa:** Diversos jogadores afirmaram que o jogo era fácil demais sem engajar nas mecânicas de risco e recompensa, o que significa que elas foram comunicadas de forma insatisfatória. Para resolver isso, a seção "How to Play"do protótipo poderia ser melhorada e obrigatoriamente apresentada ao jogador da primeira vez que ele entrasse no jogo, para garantir que todos os jogadores vissem esse tutorial.

- **Melhorar algoritmo de distribuição espacial das estrelas:** No momento, a distribuição do ponto de origem de cada estrela cadente é semi-aleatória, com maior tendência de estrelas caírem próximas umas das outras. Isso pode fazer com que muitas estrelas seguidas caiam sempre no mesmo lugar, deixando o jogador seguro sem precisar se movimentar muito. Mudar essa distribuição parece um próximo passo natural, assim como se possível atrelá-la à música de alguma forma, talvez com influência de seu *pitch*.

- **Melhorar movimentação do jogador:** Muitos jogadores reportaram problemas com o movimento da nave, embora não tenha ficado claro qual exatamente é o problema. De qualquer forma, variáveis como a velocidade máxima do jogador, sua aceleração e a inércia do movimento poderiam ser melhores ajustadas.

- **Aumentar variabilidade das estrelas:** Alguns jogadores comentaram que seria interessante se as estrelas possuíssem maior variabilidade que não seja puramente estética. Algumas possibilidades são: estrelas vindo de outros cantos da tela, estrelas de tamanhos diferentes, estrelas que rotacionam ou caem em velocidades diferentes, estrelas que alteram sua trajetória, ou até mesmo estrelas que na verdade recuperam a vida do jogador. These beatmaps were used in the development of two rhythm game prototypes: one in the vein of traditional rhythm games and the other with original gameplay. These prototypes were tested by 14 people, which surveyed the usability, fun and perceived synchronization between the game and their music. In a five point scale, the prototypes obtained the average value of 3.5 for the synchronization facto

Capítulo 5

Conclusões

5.1 Considerações Finais

A partir das avaliações do protótipo *Star Connect*, este trabalho mostra que é possível obter resultados razoáveis utilizando-se métodos automatizados de extração de *features* musicais para a criação de fases de jogos de ritmo tradicionais. Para a criação desse *beatmapper*, os únicos eventos utilizados foram os *onsets* e *beats* de uma música e os métodos utilizados baseiam-se unicamente em técnicas de processamento de áudio, não dependendo portanto da existência de grandes *datasets*, como uma solução que utiliza *machine learning*. Os algoritmos propostos ainda podem ser muito otimizados e aprimorados, mas constituem um bom ponto de partida para a criação de sistemas mais complexos.

Adicionalmente, a partir das avaliações do *Star Shooter*, demonstra-se a viabilidade de desenvolvimento e o interesse de jogadores por jogos de ritmo que fogem do padrão tradicional do uso do *beatmap* como *ground truth*. Novamente, as mecânicas utilizadas no jogo poderiam ter sido melhores desenvolvidas e não caracterizam uma limitação do uso da música na construção de desafios interessantes de jogo.

Conclui-se portanto que os objetivos iniciais do trabalho foram concluídos com sucesso, abrindo as portas para o desenvolvimento de jogos de ritmo ainda mais criativos e originais, sem que seja necessário que seus desenvolvedores passem horas construindo e ajustando manualmente os *beatmaps* de cada fase.

5.2 Trabalhos Futuros

Pretende-se continuar aprimorando os protótipos *Star Connect* e *Star Shooter* com as melhorias apresentadas no *postmortem* da seção anterior. Adicionalmente, novas formas de atrelar a jogabilidade de um jogo a uma música ainda podem ser exploradas. Algumas ideias que haviam sido levantadas durante o *brainstorm* que levou ao desenvolvimento do *Star Shooter* mas que foram ultimamente descartadas foram: releituras de jogos clássicos do *arcade* como *Snake* e *Pong*, com movimentos quantizados segundo os *beats* de uma música, assim como uma releitura do jogo "Gênio", no qual o jogador precisa repetir padrões rítmicos que seriam gerados a partir de suas músicas.

Além disso, outras informações musicais além dos *onsets* e *beats* como: *offsets* Liang e H. (2015), *pitches* da melodia principal e espacialização dos sons Tzanetakis e K. (2007) são *features* que podem dar origem a ideias de jogabilidade interessantes. Especificamente sobre o uso da melodia, destaca-se a pesquisa de Salamon (2013), que levou ao desenvolvimento do algoritmo *melodia*. Esse algoritmo encontra-se facilmente disponível para uso através de uma

biblioteca de *python* [1], podendo-ser rapidamente usado para experimentos e prototipagem.

[1] https://www.upf.edu/web/mtg/melodia

Referências Bibliográficas

C. Raffel e Ellis(2014) E. J. Humphrey J. Salamon O. Nieto D. Liang C. Raffel, B. McFee e D. P. W. Ellis. mir_eval: A transparent implementation of common mir metrics. Em *15th International Conference on Music Information Retrieval*. Citado na pág. 15

Chris Donahue e McAuley(2017) Zachary C. Lipton Chris Donahue e Julian McAuley. Dance dance convolution. Em *34th International Conference on Machine Learning*. Citado na pág. 1

Iazetta(2020) Fernando Henrique Iazetta. Tutoriais de Áudio e acústica. http://www2.eca.usp.br/prof/iazzetta/tutor/, 2020. Último acesso em 01/12/2020. Citado na pág. 7

Kähärä(2018) Lassi Kähärä. Producing adaptive music for non-linear media. Dissertação de Mestrado, Tampere University of Applied Sciences, Finland. Citado na pág. 12

Leblond(2020) Emmanuel Leblond. Godot python. https://github.com/touilleMan/godot-python, 2020. Version 0.50.0. Citado na pág. 23

Liang e H.(2015) Su L. Yang Y. Liang, C. e Lin H. Musical offset detection of pitched instruments: The case of violin. Em *15th International Conference on Music Information Retrieval*. Citado na pág. 35

Mcfee *et al.*(2015) Brian Mcfee, Colin Raffel, Dawen Liang, Daniel Ellis, Matt Mcvicar, Eric Battenberg e Oriol Nieto. librosa: Audio and music signal analysis in python. *Proceedings of the 14th Python in Science Conference*. doi: 10.25080/majora-7b98e3ed-003. Citado na pág. 15

musictheory.net(2020) musictheory.net. Music theory lessons. https://www.musictheory.net/lessons, 2020. Último acesso em 01/12/2020. Citado na pág. 4

Müller(2015) Meinard Müller. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer International Publishing, 1º edição. Citado na pág. 3, 13

Müller e Zalkow(2019) Meinard Müller e Frank Zalkow. Fmp notebooks: Educational material for teaching and learning fundamentals of music processing. Em *International Conference on Music Information Retrieval (ISMIR)*. Citado na pág. 13

Pichlmair e Kayali(2007) Martin Pichlmair e Fares Kayali. Levels of sound: On the principles of interactivity in music video games. Em *Digital Games Research Association 2007 Conference*. Citado na pág. 1

Rebelo(2016) Ruben Rodrigues Rebelo. Building a music rhythm video game. Dissertação de Mestrado, Instituto Superior Técnico, Universidade de Lisboa, Portugal. Citado na pág. 1

Salamon(2013) J. Salamon. *Melody Extraction from Polyphonic Music Signals*. Tese de Doutorado, Universitat Pompeu Fabra, Barcelona, Spain. Citado na pág. 35

Society(2019) Oxford Artificial Intelligence Society. Deepsaber: Generating beat saber levels using machine learning. https://oxai.org/2019/07/20/deepsaber.html, 2019. Último acesso em 01/12/2020. Citado na pág. 1

Tzanetakis e K.(2007) Jones R. Tzanetakis, G. e McNally K. Stereo panning features for classifying recording production style. Em *International Conference on Music Information Retrieval (ISMIR)*. Citado na pág. 35