Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

MAC0499 - Trabalho de Formatura Supervisionado

Beat'n Strike

Desenvolvedores:
Alexandre Barbosa de Macedo
Daniel Grecco Machado
Fábio Tsuguta Matsumoto

Orientador:
Prof. Dr. Flávio Soares Correa da Silva

Sumário

I. Parte Técnica

Capítulo 1

Introdução

Jogos de Computador é um tipo de software que mais abrange as áreas da computação. Inteligência artificial, computação gráfica, programação orientada a objetos, estrutura de dados, computação musical, entre outras, são exemplos das principais competências para o desenvolvimento deste tipo de software. Em função desta complexidade, uma das áreas mais importantes para o desenvolvimento de jogos é a engenharia de software, pois um planejamento e modelagem adequados são fundamentais para unir harmoniosamente todas estas diferentes áreas afim de criar jogos de qualidade.

A idéia do projeto é desenvolver um jogo, desde seu conceito até um demo jogável, que envolva várias das competências do curso de computação, além de explorar alguns aspectos desta área em específico.

O jogo em questão chama-se *Beat'n Strike* que relaciona planos sensoriais diferentes, a audição e a visão. O intuito é transmitir a sensação musical para o jogador através da interface gráfica, ou seja o que queremos fazer é "desenhar", com formas geométricas e cores, a música escolhida pelo jogador durante a partida. Para tal, foi necessário descobrir como extrair algumas características fundamentais da música, além de interpretar e decodificar arquivos MP3.

A plataforma escolhida para o jogo foi o Windows®, utilizando para o desenvolvimento o *framework* XNA junto com a IDE Visual Studio 2008.

1.1 Objetivos do Trabalho

O objetivo deste projeto é desenvolver um jogo para Windows®, com possível, porém distante, extensão para o console XBOX 360, utilizando em especial técnicas de engenharia de software, além de outras áreas da computação, como computação gráfica, computação musical e programação orientada a objetos.

1.2 Organização dos Capítulos

Os capítulos estão organizados da seguinte maneira:

- **Capítulo 2 - Ambiente de Desenvolvimento:** apresentação do ambiente e ferramentas utilizadas no desenvolvimento do projeto;

- **Capítulo 3 - O Jogo:** descrição do funcionamento e dinâmica do jogo *Beat'n Strike*;
- **Capítulo 4 - Desenvolvimento:** apresentação de como foi o processo de desenvolvimento do jogo.
- **Capítulo 5 - Portando *Beat'n Strike* para XBOX 360:** descreve o procedimento e resultados obtidos durante a portação do jogo para o console XBOX 360.
- **Capítulo 6 - Conclusão:** apresentação dos resultados obtidos.

Capítulo 2

Ambiente de Desenvolvimento

Neste capítulo, iremos apresentar as tecnologias utilizadas no projeto.

2.1 Ferramentas

O projeto foi desenvolvido na plataforma Windows®, junto com o IDE Visual Studio 2008 distribuído pela Microsoft®. Além disso, a linguagem de programação utilizada foi a linguagem C# [10] (*C sharp*) que utiliza a base .NET criada pela Microsoft®.

Também usamos a *High Level Shading Language*[13] (HLSL), uma linguagem de *shader*[25] proprietária, criada pela Microsoft®. Um *shader* é um conjunto de instruções com o objetivo de calcular efeitos de renderização diretamente no hardware gráfico, com um alto grau de flexibilidade.

2.2 *Frameworks*

2.2.1 XNA[18]

XNA (*XNA's Not Acronymed*) é um *framework* baseado na plataforma .NET, distribuído pela Microsoft® com o objetivo de facilitar o desenvolvimento de jogos. Ele foi criado com um extenso conjunto de classes e bibliotecas, específicas para programação de jogos, encapsulando detalhes de programação de baixo nível envolvidos na criação de jogos. Desta forma permite ao desenvolvedor focar no conteúdo e na diversão proporcionada pelo jogo. Além disso, este *framework* foi desenvolvido pensando na total ou máxima portabilidade de código entre as plataformas Windows® e XBOX 360[14], sendo o jogo executado com mínima ou nenhuma modificação. Esta portabilidade somente é possível pois o XNA é executado sobre uma máquina virtual (*Common Language Runtime*[11]) otimizada para jogos. A linguagem de programação que possui suporte ao XNA é a linguagem C#.

A versão do *framework* XNA utilizado neste trabalho foi a versão 3.0.

Capítulo 3

O Jogo

3.1 Introdução

Atualmente, a indústria de jogos se tornou uma área de entretenimento tão lucrativa e grande como a do cinema. Isso torna o desenvolvimento de jogos um campo multidisciplinar, envolvendo diversas áreas diferentes. Um jogo projetado por uma grande empresa não inclui somente programadores e designers gráficos em sua equipe, também inclui roteiristas, músicos, engenheiros de áudio, etc. Todos coordenados por um produtor.

Com o objetivo de criar um jogo minimalista, porém interessante, e sabendo de nossas limitações, uma equipe com apenas três pessoas, composta apenas por programadores sem talento artístico, decidimos criar um conceito de jogo que não dependesse dessas características. Chegamos a duas idéias, a primeira foi utilizar uma música escolhida pelo jogador como o *background* sonoro. Desse modo, não teríamos que nos preocupar em compor uma trilha sonora, e o jogador ainda ouviria uma música que provavelmente gosta. Também poderíamos extrair informações da música, de forma a influenciar no jogo.

A segunda idéia foi utilizar apenas desenhos geométricos simples nos elementos gráficos do jogo. Isso eliminaria a necessidade de elaborar texturas, algo em que não tínhamos muita habilidade. Assim desenharíamos utilizando primitivas gráficas diretamente na GPU.

3.2 Características Principais

As características do jogo que mais se destacam:

Uso da música: Hoje em dia, encontramos muitos jogos em que a música exerce um papel fundamental na dinâmica do jogo. Há jogos como *Guitar Hero*®[1] ou *Rock Band*®[6], em que o objetivo é simular que o jogador está tocando um instrumento musical. Neste tipo de jogo, deve-se acertar uma sequência de "notas" no tempo correto, de acordo com a música que está sendo tocada. Isso elevou a música de elemento secundário, apenas trilha sonora, para elemento principal no *gameplay*.

Há ainda jogos como o *Audiosurf*[5], que se assemelha mais com o conceito do nosso jogo, onde escolhe-se previamente uma música, que será responsável pela geração de obstáculos na tela. O jogador deve coletar os obstáculos para aumentar a pontuação.

Figura 3.1: Imagem do jogo *Audiosurf*

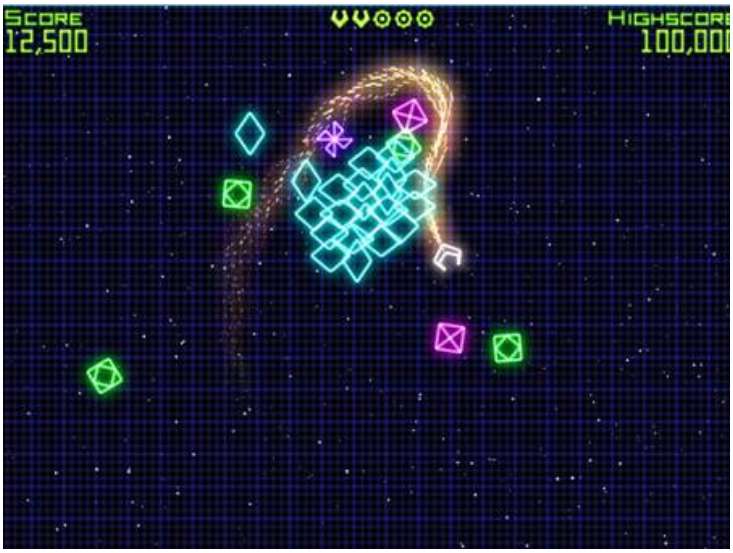

Figura 3.2: Imagem do jogo *Geometry Wars*

Uso de primitivas geométricas: Como dito anteriormente, resolvemos utilizar formas geométricas simples nos elementos do jogo. Isso foi inspirado no jogo *Geometry Wars*[2], que apesar de possuir um aspecto simples, produz um resultado graficamente bonito.

3.3 Gênero

Não há um gênero bem definido para este tipo de jogo. Uma possível definição seria classificar como um híbrido entre *rhythm game*, que são jogos que utilizam elementos musicais, geralmente o ritmo, e *shooter*, que são os conhecidos jogos de nave que devem atirar nos inimigos para sobreviver. A dinâmica do jogo será explicada em seguida, mas podemos pensar que neste jogo o próprio "cometa" faz o papel dos tiros, devendo colidir com os obstáculos.

3.4 Plataforma

O jogo desenvolvido roda em Windows®, e futuramente será estendido para outras plataformas, como o XBOX 360.

3.4.1 Windows®

O sistema operacional Windows® é a plataforma mais utilizada para jogos atualmente, devido a sua grande flexibilidade de hardware e larga utilização. Para rodar jogos feitos em XNA no Windows®, é necessário no mínimo ter uma placa de vídeo que suporte *Shader Model* 1.1[25] e Directx 9.0c[12].

3.4.2 XBOX 360

O XBOX 360 é o segundo *videogame* produzido pela Microsoft® e o sucessor do XBOX. Ele compete com o PlayStation 3[24] da Sony e com o Wii[20] da Nintendo como parte da sétima geração de *videogames*. Por dentro o XBOX 360 possui um processador, desenvolvido pela IBM, com três núcleos simétricos e uma frequência de 3.2 GHz cada. Além disso, ao todo o processador é capaz de processar seis *threads* simultaneamente, duas para cada núcleo. O processamento gráfico é gerenciado pela placa de vídeo ATI Xenos que possui 10 MB de memória eRAM (*embedded* RAM) e é capaz de renderizar 500 milhões de triângulos por segundo. O XBOX 360 possui 512 MB como memória principal e a média do desempenho geral do sistema de ponto flutuante é de 1 Teraflop.

Figura 3.3: Console Xbox 360 da Microsoft

3.5 Dinâmica do Jogo

O jogo consiste de uma arena fixa de duas dimensões. O jogador controla um objeto que pode mover-se livremente até os limites da tela, deixando um pequeno rastro de partículas, lembrando um cometa. Além de controlar a movimentação, o jogador também pode alterar a cor atual deste "cometa", pressionando os botões do controle ou através do teclado. Existem quatro cores possíveis – amarelo, vermelho, verde e azul – e a mudança entre elas ocorre ao se pressionar o botão da cor respectiva, ou seja, o botão azul para a cor azul e assim por diante.

Obstáculos surgem nos limites da tela, de acordo com a música previamente escolhida. Eles podem aparecer em quatro formas e cores diferentes. As cores são as mesmas que o cometa pode assumir. As formas são triângulo, quadrado, losango e hexágono. Embora a

Figura 3.4: Controle do XBOX 360 da Microsoft

música determine quando e onde um obstáculo deve aparecer, a cor e forma são aleatórias. Optamos por este modelo pois queríamos que ao mesmo tempo ficasse claro que a música estava gerando os obstáculos, houvesse um caráter aleatório, para aumentar a dificuldade e o desafio ao jogador, deixando o jogo menos previsível.

O objetivo do jogador é colidir com os obstáculos de mesma cor do cometa e evitar os de cores diferentes. Há um elemento multiplicador na pontuação, quanto mais o jogador acertar obstáculos de uma mesma cor em sequência, mais pontos irá ganhar, indo até o limite de oito obstáculos. Então para maximizar a sua pontuação, um jogador deve ter reflexos rápidos e pegar o máximo possível de obstáculos de mesma cor em sua proximidade antes de mudar para uma outra cor. Caso colida com um obstáculo de cor diferente, ele perde pontos e sentirá uma vibração no controle indicando o erro.

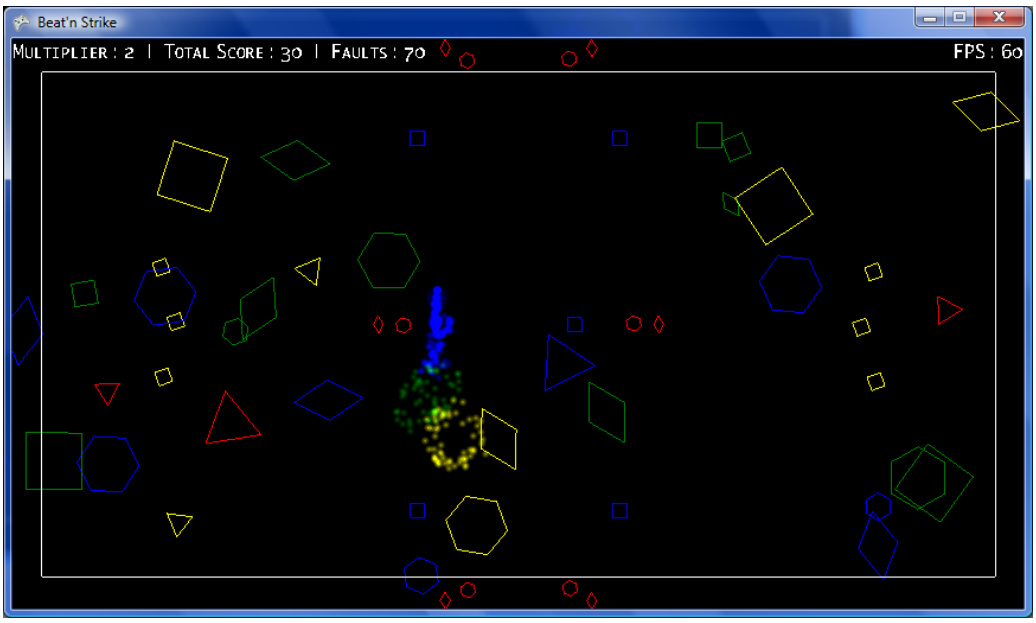

Figura 3.5: Imagem do jogo desenvolvido, demonstrando sua dinâmica

Capítulo 4

Desenvolvimento

Neste capítulo apresentaremos como planejamos o desenvolvimento do jogo *Beat'n Strike*, quais conceitos utilizamos e como cada parte do projeto foi implementada.

4.1 Planejamento

Para fornecer controle, estabilidade e organização , além de poder construir um software de qualidade, decidimos adotar um processo de software, isto é, uma estrutura ou roteiro para as tarefas que são necessárias no desenvolvimento de um sistema. Como o nível de detalhamento do processo adotado depende do tipo de software que está sendo construído, no nosso caso um jogo, utilizamos o modelo prescritivo de processo incremental/iterativo, incorporando algumas características de metodologia ágil.

4.1.1 Modelo Incremental

A ideia por trás deste modelo é desenvolver um software de modo incremental, permitindo ao desenvolvedor utilizar o que foi aprendido durante o processo de desenvolvimento da fase inicial. Este aprendizado vem do desenvolvimento e do uso do sistema, quando possível. Passos chaves no processo são iniciados com uma simples implementação de um subconjunto dos requerimento do software e iterativamente melhorias são implementadas até finalmente termos o software completo. O passo inicial cria a base do sistema, chamada de núcleo do projeto. O objetivo deste passo é criar um produto no qual o usuário já possa interagir. Ele deve oferecer uma amostra mínima de funcionalidades dos requisitos do sistema, sendo que modificações no projeto são feitas e novas características funcionais adicionadas nas iterações seguintes[22].

4.1.2 Metodologia Ágil

Junto com o modelo incremental utilizamos algumas características de metodologias ágeis, tanto de *Extreme Programming* (XP)[8] quanto de *Scrum*[8].

Extreme Programming (XP)

As características de XP utilizadas neste projeto foram:

- **Programação pareada:** Toda a codificação, durante o projeto, foi realizada em pares, para assim tentar minimizar o número de ocorrência de erros, além de disseminar o conhecimento entre os integrantes do grupo;
- **Integração contínua:** Foi amplamente utilizada pois muito do que implementamos foram pequenos módulos ou características adicionais.

Scrum

As características de *Scrum* utilizadas neste projeto foram:

- **Pequenas equipes de trabalho:** Para maximizar a comunicação, minimizar a supervisão e maximizar o compartilhamento de conhecimento;
- **Adaptabilidade:** O processo precisa ser suficientemente maleável tanto para modificações técnicas quanto de negócios;
- **Incrementos:** O processo produz pequenos incrementos de software que podem ser inspecionados, ajustados, testados, documentados e expandidos;
- **Divisão clara do trabalho:** O trabalho de desenvolvimento e o pessoal que o realiza é dividido em partições claras de baixo acoplamento ou em pacotes.

4.1.3 Divisão do Projeto

Para o desenvolvimento deste projeto resolvemos dividí-lo em três grandes partes, cada uma seguindo o modelo híbrido de desenvolvimento citado na seção acima.

A três partes principais são:

- **Áudio:** Nesta parte o *decoder/player* de MP3 foi desenvolvido;
- **Gráfico e Física:** Utilizando o *framework* XNA desenvolvemos o gráfico e física do jogo;
- **Integração:** Nesta última etapa integramos a parte de áudio com a parte gráfica, definimos o *gameplay* e a interface de navegação (menus) do jogo.

4.2 Áudio

Como dito anteriormente, a proposta é transmitir a sensação musical para o jogador através da interface do jogo, ou seja o que queremos fazer é "desenhar" a música escolhida pelo jogador durante a partida. Para tal, precisamos descobrir como podemos extrair da música algumas de suas características e representá-las no decorrer do jogo.

4.2.1 Música

Não existe um consenso total sobre a definição de o que venha a ser música, mas pode-se dizer que é uma combinação de sons e de silêncios, ritmadas, que se desenvolvem com o tempo. Esta combinação inclui variações nas características do som tais como: altura, duração, intensidade e timbre; bem como características de organização temporal: ritmo, melodia (sequencialmente) ou harmonia (simultaneamente).

Elementos que caracterizam uma música:

- **Melodia:** é uma sucessão coerente de sons e silêncios que se desenvolvem em uma sequência linear com identidade própria. É a voz principal que dá sentido a uma composição e encontra apoio musical na harmonia e no ritmo.
- **Harmonia:** conceito musical relacionado com a emissão simultânea de diferentes frequências. Ela trabalha com as sonoridades resultantes da sobreposição de diferentes notas.
- **Ritmo:** é um acontecimento sonoro, tenha ele altura definida ou não, que acontece numa certa regularidade temporal, ou seja, o ritmo é a pulsação da música. Sem ritmo não há música.
- **Altura:** ou tom (na escala musical) diz respeito à maneira como o ouvido humano percebe as frequências fundamentais. As baixas frequências são percebidas como sons graves e as mais altas como sons agudos. O ouvido humano consegue perceber sons aproximadamente entre 20 Hz e 20000 Hz.
- **Timbre:** característica sonora que nos permite distinguir fontes diferentes de sons soando na mesma frequência (fundamental). Esta percepção é possível quando uma mesma frequência tem forma de onda diferente.
- **Intensidade:** ou volume é a percepção da amplitude da onda sonora. A percepção desta intensidade varia de acordo com a altura da nota.
- **Duração:** é o tempo de uma nota ou entre as notas formando o ritmo.

São todas estas informações que, somadas, proporcionam uma sensação agradável ao ouvinte, contudo é preciso escolher algumas destas características para representá-las no jogo. Uma destas escolhas se dá naturalmente, o ritmo, que também é intrínseco ao jogo e por esta razão compartilhará do ritmo da música. Além disto, outra característica de fácil percepção é a altura. A variação de frequência, que é uma das mais marcantes propriedades da uma música, no jogo, será responsável pela definição dos obstáculos.

Com isto temos, influenciando diretamente na interface do jogo, duas características de diferentes categorias: a altura como propriedade do som e o ritmo que é parte integrante da organização temporal da música.

4.2.2 Formatos de Entrada

Uma vez definidas as informações que vamos extrair da música precisamos agora descobrir uma maneira eficiente de obtê-las. Para isto vamos analisar alguns formatos de áudio digital que servirão como entrada. A escolha do formato é extremamente importante pois a partir de cada formato diferente podemos obter diferentes características mais facilmente.

Pulse Code Modulation (PCM)

É o formato digital de áudio mais primitivo e não envolve nenhum tipo de compressão, a partir dele é possível obter as intensidades do som com facilidade. Se observarmos em sequência os valores armazenados no decorrer do tempo veremos uma onda sonora (figura ao lado), resultante da soma das várias frequências que compõe a música, que reproduz quanta pressão o som exerce sobre o meio de propagação.

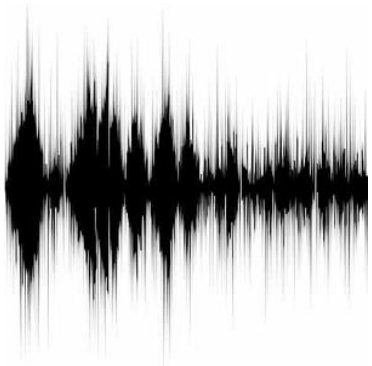

Figura 4.1: Onda sonora

Como já discutido anteriormente, as informações que queremos extrair da música são ritmo e altura, porém estas não podem ser obtidas diretamente a partir do formato PCM sem a utilização de transformações matemáticas sobre a função da onda sonora. Uma alternativa à todas estas contas é o formato MP3.

MP3

Desenvolvido inicialmente por um grupo alemão chamado Fraunhofer & Thomson em conjunto com a universidade da Erlangen (Professor Dieter Seitzer), foi oficialmente padronizada em 1993 pela Organização de padrões internacional (*International Standards Organization*, ou ISO)[7]. Também conhecido como MPEG-1 Audio Layer 3, foi um dos primeiros tipos de compressão de áudio a manter a qualidade sonora da música com perdas quase imperceptíveis à audição humana. Isto é, para a maioria dos ouvintes, um áudio comprimido utilizando a compressão MP3 soa tão fiel quanto o áudio original, ainda que projetado para reduzir significavelmente a quantidade de dados que o representam (compressão com perda). Isto somente é possível pois o compactador aproveita o conhecimento das imperfeições ou limitações da audição humana para eliminar informações sem afetar o que ouvimos, conseguindo assim um grande nível de compressão. Este conhecimento sobre a nossa audição é chamado de modelo psicoacústico humano (um ouvido virtual) e é carregado em todos os codificadores, cada um possuindo um modelo diferente portanto cada *encoder* produz um resultado diferente. O nome, claro, corresponde a extensão encontrada nos arquivos MP3, um dos formatos de áudio digital mais populares, senão o mais popular, da atualidade. Aliada a esta popularidade, este formato ainda nos fornece uma maneira rápida de obtermos os dados que precisamos graças ao processo de compressão. Estas vantagens nos levaram à escolha deste formato para entrada no jogo[19].

4.2.3 Extração das informações

Como dito anteriormente, a escolha do formato favorece a extração das informações de maneira que precisamos fazer menos cálculos no decorrer da partida. Uma vez que temos que tocar a música durante o jogo, nos comprometemos a decodificar o arquivo MP3. E é exatamente durante o processo de *decoding* que vamos obter as características da música. A seguir vamos explicar um pouco do padrão MP3 e os processos de codificação e decodificação de maneira a ilustrar melhor como tiramos proveito destes processos.

Encoding do MP3

> It's one of the most popular and controversial file types in the world, but most explanations of its workings are either blindingly simplistic or overwhelmingly technical.
>
> – *Thomas Wilburn*

O padrão MPEG-1 Layer III é bem especificado, mas não contém uma definição precisa para um *encoder*[9]. O processo que vamos descrever é bem superficial, porém possui informações suficientes para a finalidade do projeto. A figura abaixo ilustra a base do processo de codificação.

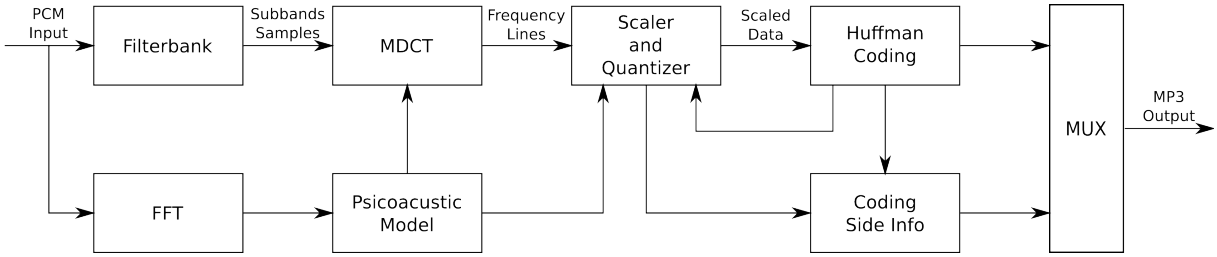

Figura 4.2: Diagrama do processo de codificação do MP3.

Inicialmente vamos partir de um arquivo de áudio descomprimido (PCM). Neste arquivo vamos encontrar amostras do som de um determinado "instante" (intervalo bem pequeno de tempo), estas amostras são chamadas de *samples*. Cada uma destas amostras passa inicialmente através de um conjunto de filtros chamado *filterbank*. Estes filtros dividem o áudio em 32 bandas que ainda são divididas por um fator valendo 18, perfazendo portanto 576 bandas menores. Estas bandas não tem tamanho fixo, ou seja, durante o *encoding*, informações baseadas no modelo psicoacústico humano determinam o tamanho da cada uma delas influenciando diretamente na fidelidade que se deseja obter no som por banda de frequência.

A partir deste ponto dois processos paralelos darão inicio série de cálculos matemáticos: *Fast Fourier Transforms* (FFT) e *Modified Discrete Cosine Transform* (MDCT). O FFT analisa as informações contidas em cada banda para alimentar o modelo psicoacústico humano do *encoder*. Este utiliza o modelo pra responder algumas questões, por exemplo: esta banda será sobreposta por outra mais alta? O áudio é relativamente constante ou

variável? As respostas para estas e outras perguntas serão utilizadas pelo *encoder* para descobrir qual informação pode ser removida com segurança, ou seja, sem causar perda sensível de qualidade com relação ao ouvido humano.

Paralelamente o MDCT transforma cada banda num conjunto de valores espectrais. Espectro sonoro é o conjunto de sons parciais ordenados a partir de uma frequência fundamental, seguindo uma relação frequência/amplitude. O espectro de um som complexo (som composto por mais de uma frequência) caracteriza graficamente a forma da onda sonora que o define. Este gráfico é composto de barras, cada uma delas representando a amplitude de uma das frequências componentes do som analisado. Como a compressão é baseada na audição humana, o espectro só contempla as frequências audíveis, ou seja, variam de 20Hz até 20KHz.

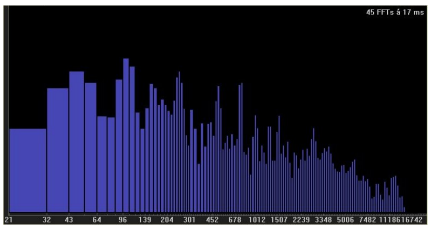

Figura 4.3: Espetrograma de uma música

Ao contrário dos valores iniciais que representam a perturbação oscilante do meio de propagação no decorrer do tempo, após o MDCT obtemos uma representação do som como energia em relação à um intervalo de frequências. Como a representação do som como forma de espectro é a maneira que mais se assemelha à maneira com que o ouvido humano interpreta o som, muitos *encoders* utilizam esta transformação para alimentar seus modelos psicoacústicos. É por esta razão que, em geral, nos identificamos com aquelas "barrinhas" dos *players* populares. É desta informação que vamos precisar no momento da decodificação do arquivo MP3.

Com dados suficientes para "reconstruir" o MP3, o processo seguinte é comprimir propriamente o arquivo. Para realizar esta tarefa o *encoder* utiliza duas abordagens: a compressão baseada no modelo psicoacústico, que é muito eficiente em músicas mais complexas com muitos componentes misturados; e a compressão tradicional que é mais eficiente quando utilizada em sons mais simples. É combinando ambos os métodos que o formato MP3 alcança grandes níveis de compressão.

A primeira fase da compressão chama-se requantização. Durante esta fase os 576 valores obtidos após o MDCT são divididos por um valor chamado requantizador e em seguida classificados em 22 bandas chamadas de *scalefactor bands* para serem escalados tanto para mais quanto para menos levando em consideração a precisão necessária fornecida pela saída do FFT. O processo envolve arredondamentos que, durante o processo de decodificação, serão transformados em ruído, portanto esta conta simples exige um equilíbrio entre boa compressão com perda em forma de ruído e uma compressão menor que garanta uma fidelidade razoável.

Em seguida os dados serão submetidos a um próximo estágio de compressão, desta vez sem perda. Esta fase aplica o algoritmo de Huffman para codificar as informações. *Huffman coding* é um algoritmo probabilístico que utiliza códigos de comprimento variável. Ele se baseia na frequência estimada da ocorrência dos símbolos a serem codificados. A cada símbolo do alfabeto é atribuído um código binário que o representará. Este código tem tamanho variável sendo que os mais curtos representam os símbolos do alfabeto

que mais ocorrem. Para evitar confusão no momento da decodificação, nenhum código é prefixo do outro. Para que os códigos sejam atribuídos aos símbolos de maneira conveniente, o algoritmo constrói uma árvore binária completa. A seguir vamos exemplificar o funcionamento do processo.

Inicialmente deve-se conhecer todos os símbolos e sua probabilidade de ocorrência, ou seja, o algoritmo precisa da distribuição de probabilidade dos símbolos a serem codificados.

S1	S2	S3	S4	S5	S6
0,13	0,05	0,17	0,08	0,36	0,21

Figura 4.4: Símbolos com distribuição de probabilidade.

Em seguida os símbolos são ordenados por ordem de prioridade. A cada passo são selecionados os dois últimos para formarem um novo símbolo. Este processo se repete até que reste apenas um único símbolo, que será a raiz de nossa árvore binária.

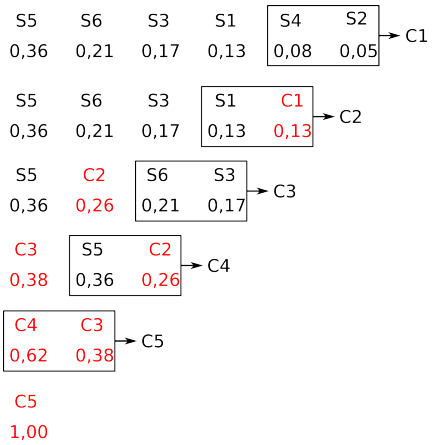

Figura 4.5: Geração de símbolos intermediários.

Ao final do processo a árvore binária é organizada com os nós gerados nos passos anteriores.

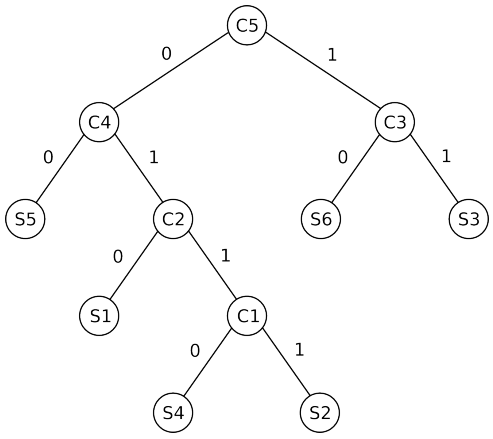

Figura 4.6: Árvore gerada.

O código de um símbolo é determinado pelo caminho que se percorre até sua folha.

S1 - 010
S2 - 0111
S3 - 11
S4 - 0110
S5 - 00
S6 - 10

Figura 4.7: Atribuição dos códigos.

Após aplicadas as compressões, obtivemos um bloco de dados de áudio correspondente a um *frame* do arquivo de MP3. Esta porção de dados binários vem acompanhada de um cabeçalho que chamamos de *frame header* contendo algumas informações relevantes sobre o bloco de dados que representa. O *frame* ainda contém informações sobre erros e dados opcionais. Abaixo temos a descrição de um *frame* completo.

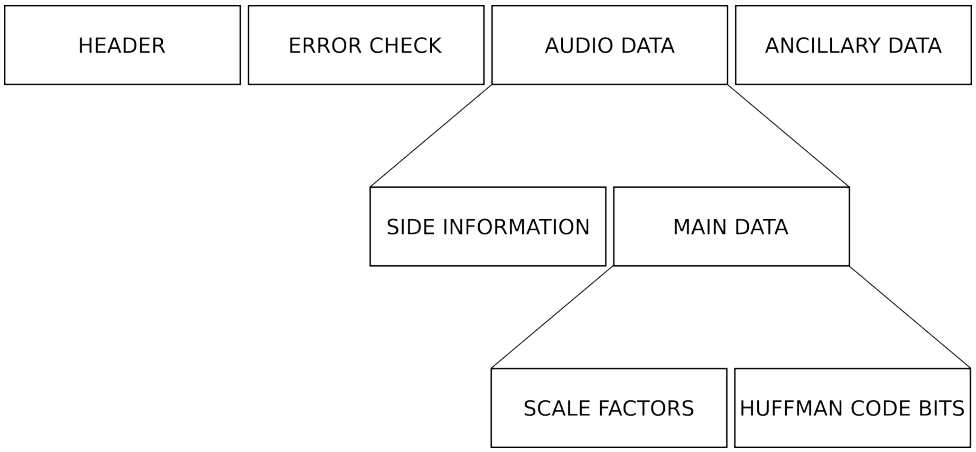

Figura 4.8: Estrutura do *frame*.

- **Header** - Tem o tamanho de 4 bytes. Possui uma marcação de sincronização para que o decodificador possa encontrar o início do *frame*, informações sobre o *layer* (*layer* III no caso do MP3), taxa de bits, taxa de amostragem de frequência, e modo (mono, stereo).
- **Error check** - Contém 16 bits para uma verificação opcional de erros de codificação.
- **Side information** - Contém as informações necessárias para a decodificação do *Main Data*, entre elas: informação de seleção dos *scale factors*, ganho global que será aplicado em todas as amostras, especificação da tabela de Huffman para decodificação das amostras contidas no *frame*.
- **Main data** - Contém os *scale factors* que foram utilizados e os valores das amostras codificadas usando códigos de Huffman.
- Ancillary data - Este campo é reservado para a adição de informações extras como: nome do autor, nome do álbum, nome da faixa, etc.

O resultado da codificação do MP3 é uma sequência de *frames* iguais ao descrito acima armazenados no arquivo com extensão .mp3.

Decoding do MP3

A decodificação, por outro lado, é cuidadosamente especificada no padrão. O *decoder* trabalha o reverso, isto é, ele lê o arquivo binário, descomprime as amostras de frequências, reconstrói as amostras baseadas nas informações de como o modelo removeu conteúdo e transforma-as para o domínio de tempo como no arquivo original no formato PCM[4].

O diagrama em blocos a seguir mostra o fluxo de dados dentro de um *decoder* MP3.

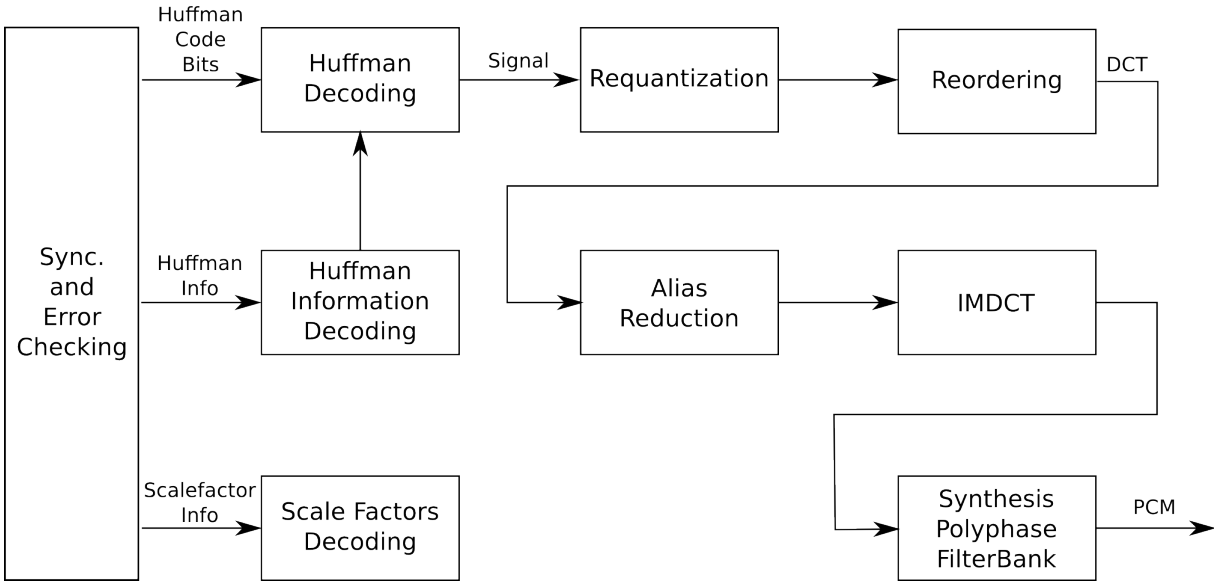

Figura 4.9: Diagrama do processo de decodificação do MP3.

Os dados de entrada são divididos em *frames* e sua corretude pode ser verificada usando CRC (*Cyclic Redundancy Code*). Em seguida, usando as informações de seleção dos *scale factors* contidas no *side information*, os valores de escala são decodificados no bloco *Scale factor decoding*. Paralelamente no bloco *Huffman Info Decoding* a árvore binária utilizada para a codificação de Huffman será recuperada e desta vez será utilizada para decodificar as amostras das subbandas no bloco *Huffman Decoding*. Os *scale factors* são utilizados para re-escalar as amostras de cada subbanda, este processo chama-se *Re-quantization* que reverte a quantização aplicada no processo de codificação. Neste passo algumas perdas de arredondamento são esperadas, contudo foram cuidadosamente dimensionadas para que não influenciem sensivelmente no resultado final.

Uma reordenação pode ser realizada durante o processo de codificação para agrupar valores parecidos o que facilita a compressão por Huffman. Se ela foi aplicada, é preciso revertê-la durante decodificação do MP3, este reagrupamento é feito no bloco que chamamos de *Reordering*.

Alias reduction é um bloco que corrige imperfeições decorrentes da codificação. Os 32 filtros do *filter bank* causaram um efeito de *aliasing* no sinal que será amenizado nesta fase da decodificação. Neste ponto temos o espectro que será utilizado para extrairmos as informações que precisamos da música.

Quase no fim, aplicamos o IMDCT (*Inverse Modified Discrete Cosine Transform*) que é a inversa da MDCT. Neste passo o espectro é transformado de volta para o domínio

de tempo originário do formato PCM. Após a transformação obtivemos a música original a menos de supressões de áudio realizadas pelo modelo psicoacústico do *encoder* e arredondamentos decorrentes da quantização.

O último passo é "somar" as 32 bandas para formar a onda PCM. A decodificação não precisa ser completada para a reprodução do áudio, a música pode ser reproduzida durante a decodificação.

4.2.4 Algoritmo de detecção de batidas

O sistema auditivo humano determina o ritmo da música detectando uma pseudo-periódica sucessão de batidas. Este sinal interceptado pelo ouvido contém uma certa energia, esta energia é convertida em impulsos elétricos que são interpretados pelo cérebro. É claro que quanto maior for a energia interceptada, mais forte o som vai parecer. Intuitivamente podemos dizer que uma batida ocorre quando a sua energia é muito maior que o histórico do som. A idéia é ensinar a máquina a detectar batidas quando o som variar muito bruscamente. O algoritmo que implementamos é um detector estatístico de batidas baseado nesta definição[21].

Neste ponto já temos todas as informações que precisamos, o que precisamos é manipulá-las de maneira a atender nossas necessidades. Com o espectro em mãos o primeiro passo foi repartir o domínio em bandas de tamanhos convenientes, e classificá-las segundo o modelo da audição humana. Esta classificação se baseia no fato de que a grande maioria das músicas possuirem espectros parecidos se comparados com relação a faixa de frequência que ocupa dentro da faixa audível humana (20Hz - 20Khz), isto é, todas elas ocupam predominantemente o primeiro quarto de espectro. Isto implica que esta região terá mais subdivisões do que as demais. As sete bandas são:

- Sub Bass (20Hz - 60Hz);
- Lower Bass (60Hz - 140Hz);
- Bass (140Hz - 250Hz);
- Lower Midrange (250Hz - 1500Hz);
- Upper Midrange (1500Hz - 3500Hz);
- Presence (3500Hz - 6000Hz);
- Brilliance (6000Hz - 20000Hz);

Quando condensamos estes sete valores através do cálculo das médias o que obtivemos foi um gráfico semelhante ao que pode ser visualizado em muitos *players* populares. Precisamente este gráfico é a "fotografia" da música que vamos tirar a cada 24ms aproximadamente.

No decorrer da música guardamos estas "fotografias" em um *buffer* de tamanho variável, ou seja, cada banda tem em seu próprio *buffer* um histórico das últimas amplitudes

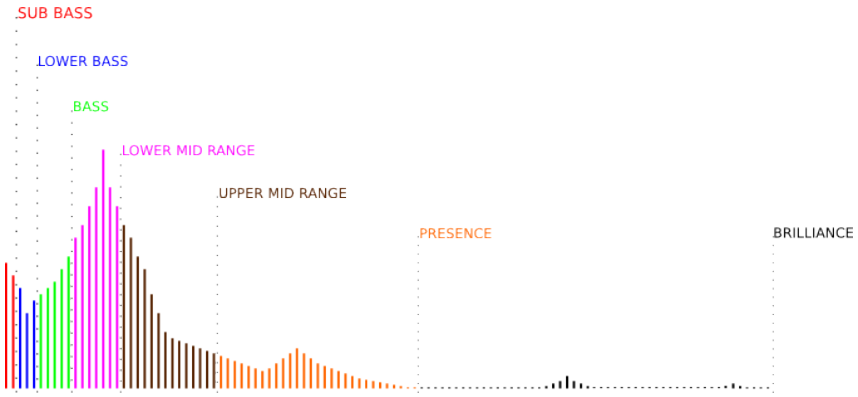

Figura 4.10: Classificação do espectro em bandas.

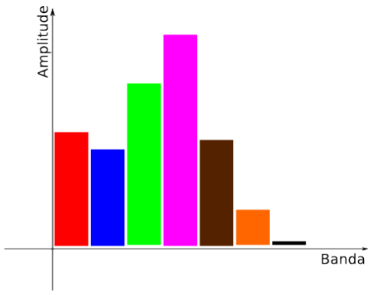

Figura 4.11: Média das 7 bandas.

registradas relativas a um período pré-determinado em segundos (diferente para cada banda). A cada novo registro, comparamos sua amplitude com a média registrada nos últimos segundos multiplicada por uma constante de sensibilidade que age como uma espécie de desvio padrão. Se o valor da amplitude extrapolar o valor calculado a partir do histórico então consideramos que uma batida ocorreu. A figura a seguir ilustra o processo.

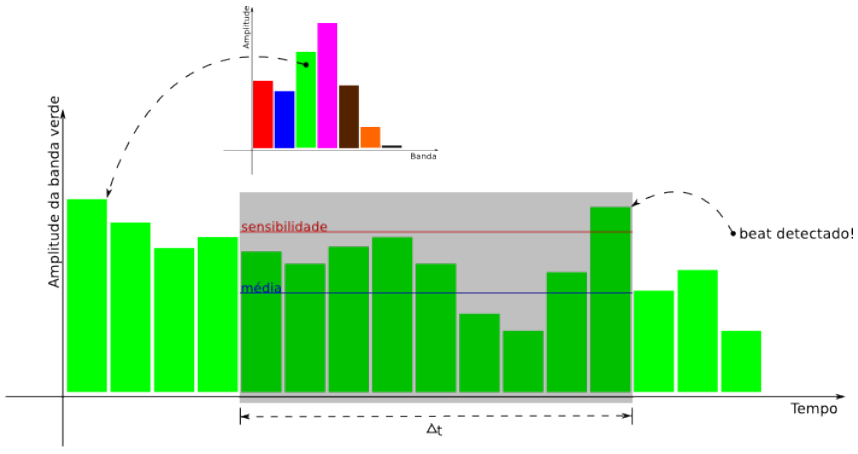

Figura 4.12: Detecção das batidas.

Uma constante diferente para cada banda é selecionada para ajustar a sensibilidade do detector de batidas assim como o tamanho do *buffer* que também pode ser ajustado para qualquer valor em segundos aumentando ou diminuindo o histórico de uma determinada banda. Esta flexibilidade é importante pois se uma música tem muitas variações no seu

ritmo, é importante manter um menor número informações no *buffer* para que a detecção das batidas tenha um efeito menos global possibilitando assim que o algoritmo se adapte rapidamente a estas variações. Desta maneira é possível determinar qual banda gera mais batidas e definir perfis para cada estilo musical.

4.3 Gráficos e Física

O XNA é um *framework* de alto nível e orientado a objetos, que encapsula os detalhes de baixo nível envolvidos na criação de jogos e garante que as diferenças entre plataformas, caso haja a necessidade de portabilidade, sejam resolvidas facilmente.

Uma das diferenças entre as plataformas, XBOX 360 e Windows®, se trata da versão do .NET *framework*, da qual o XNA depende. No caso do Windows®, é utilizado o .NET *framework* 2.0, enquanto o XBOX 360 utiliza o .NET *Compact Framework* 2.0, uma versão mais simplificada, onde apenas um subconjunto das bibliotecas do *framework* similar utilizado em Windows® estão disponíveis. Como tínhamos o objetivo de portar para o XBOX 360, isso orientou o desenvolvimento para utilizar o mínimo possível de estruturas de dados prontas do sistema, logo implementamos nossas próprias estruturas usando *arrays* na maior parte dos casos. Dessa forma, não dependíamos tanto das bibliotecas do .NET, e facilitaríamos a portabilidade para o XBOX 360.

O XNA permite tanto a criação de jogos em 2D como em 3D. Começamos a desenvolver o jogo inicialmente em 2D, devido à jogabilidade, exposta no capítulo anterior, ser em duas dimensões. Porém, há uma limitação, quando trabalhamos em 2D no XNA devemos carregar sempre texturas (*sprites*), não sendo possível desenhar primitivas diretamente. Nesse caso até poderíamos utilizar um software de desenho vetorial para desenhar os elementos, e apenas carregar as imagens criadas, mas isso traria uma inconveniência para o tratamento das colisões, que será explicada mais a frente. Então optamos por desenvolver em 3D, deixando uma das dimensões – a profundidade, ou Z – sem uso.

Geralmente ao se desenvolver jogos, sempre temos um *loop* principal, responsável pela atualização e gerenciamento do estado do jogo. No XNA, temos os métodos *Update* e *Draw*, localizados na classe *Game*, e que são responsáveis pela atualização e pelo desenho na tela, respectivamente. Além disso, o XNA possui uma forma bem modular de integrar as partes do código, através da classe *GameComponent*. Um *GameComponent* possui seu próprio método *Update*, que é automaticamente atrelado ao loop principal. Depois da chamada à *Update* da classe *Game*, todos os *GameComponents* associados tem seu método *Update* chamados também. Há ainda o *DrawableGameComponent*, que além de possuir o método *Update*, possui o *Draw* que também fica associado à chamada do método *Draw* da classe principal. Utilizando estas características, desenvolvemos diversos componentes baseados na necessidade do jogo[23].

Uma visão geral dos GameComponents desenvolvidos:

InputHandler: responsável pela verificação da entrada feita através do controle do XBOX 360 e do teclado;

VibrationHandler: responsável pelo gerenciamento da vibração do controle do XBOX 360;

GameStateManager: responsável pelo gerenciamento das telas do jogo;

SpectrumAnalyzer: responsável pela análise do espectro da música;

BeatDetectorManager: responsável pela análise do ritmo da música.

Uma visão geral dos DrawableGameComponents:

Comet: objeto controlado pelo jogador;

FrameBurst: quadros localizados ao redor da tela, atualizados conforme a música;

ObstacleManager: responsável por criar, gerenciar e verificar a colisão dos obstáculos;

GameState: a tela de jogo;

ParticleSystem: utilizados no Comet e nas explosões causadas quando ocorre colisão entre o cometa e os obstáculos.

4.3.1 Sistema de Partículas

Para criar o cometa e as explosões, utilizamos uma técnica conhecida como *Point Sprites*. Um *point sprite* é um vértice que possui uma textura, que pode variar de tamanho, e cuja face sempre está voltada para a câmera[3]. O uso de *point sprites* melhora a performance principalmente por três motivos:

- um *point sprite* usa apenas um vértice, economizando espaço de armazenamento. O modo habitual seria mapear a textura para um triângulo ou retângulo, dessa maneira gastando três ou quatro vértices, respectivamente;
- por ser uma técnica recorrente em sistemas de partículas, há uma maneira de mapear automaticamente no código responsável pela renderização dos objetos na tela , evitando a necessidade armazenar as coordenadas para o mapeamento utilizado na transformação da textura em 2D para o mundo do jogo, em 3D;
- como os *point sprites* sempre estão voltados para a câmera, não há necessidade de código para ajustar o ângulo de visão por diferentes direções.

4.3.2 Física

Para o "cometa" explicado anteriormente ter um aspecto mais convincente, ou seja, deixar um rastro de partículas conforme se movimenta, foi necessário implementar um conjunto de funções simples. As classes que compõem a física do jogo são:

PhysicsHelper: contém diversos métodos estáticos, responsáveis principalmente pela cinemática das partículas, ou seja, calcula a aceleração, velocidade e posição delas;

IForce: ao se criar um sistema de partículas, podemos associar a força à qual ele estará sujeito, que deve ser uma implementação da interface IForce;

DireccionalForce: implementa a interface IForce, é uma força direcional, funciona de maneira semelhante a um "vento", ou seja, movimenta as partículas numa única direção;

PunctualForce: também implementa a interface IForce. Funciona como a gravidade, gerando uma força de atração ou repulsão num determinado ponto. Apesar de termos implementado no início, acabamos não utilizando esse tipo de força no jogo final.

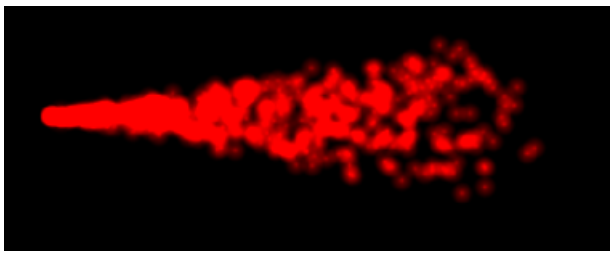

Figura 4.13: Cometa sofrendo ação da força direcional.

4.3.3 Colisão

Anteriormente foi dito que o jogo foi feito em 3D pois isso iria facilitar a detecção das colisões. Em 2D, deveríamos carregar a textura para o nosso obstáculo, e nesse caso não teríamos as coordenadas dos vértices, apenas a posição do *sprite*, assim poderíamos no máximo fazer uma aproximação ao retângulo que contém a textura, tornando a detecção da colisão não muito precisa. Poderíamos utilizar outras técnicas como colisão por *pixel* (usar um mapeamento dos *pixels* da tela), porém não queríamos adicionar complexidade desnecessária. Ao utilizar 3D, deixamos de utilizar texturas para os obstáculos, e passamos a desenhar diretamente as primitivas (no caso, linhas) através dos métodos nativos presentes no XNA, que utiliza a GPU para isso. Nesse caso, estamos guardando diretamente a geometria de cada objeto que temos, e precisamos apenas de matemática para calcular as colisões. Por isso os obstáculos são todos figuras geométricas convexas, pois é mais fácil verificar se um ponto está ou não no interior deste tipo de polígono.

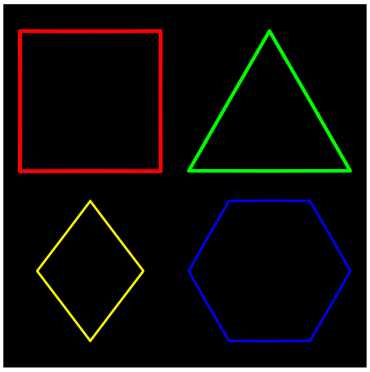

Figura 4.14: Forma dos obstáculos.

4.4 Integração

Após a conclusão da parte do áudio e da gráfica, precisávamos integrar ambas as partes. Já tínhamos um *decoder* funcional, capaz de analisar e tocar os MP3's. Quanto à parte gráfica, também tínhamos um jogo funcional, sendo que os obstáculos ainda eram gerados aleatoriamente. Foi necessário criar uma ponte, deixando o espectro do *decoder* disponível para a classe que gerencia os obstáculos. Assim tínhamos as seguintes classes:

BeatDetector: classe responsável por um intervalo do espectro, que verifica se houve "batidas" durante a execução da música;

BeatManager: gerencia um conjunto de BeatDetector's, e passa a informação para o ObstacleManager criar os obstáculos;

SpectrumAnalyzer: contém a informação do espectro presente no *decoder*. É usado no *frame* ao redor da tela.

4.4.1 *Gameplay*

Ajustar o *gameplay* se mostrou uma tarefa realmente complicada. Mesmo que a música gerasse os obstáculos, queríamos uma aleatoriedade para aumentar o desafio ao jogador. Caso ficasse muito previsível a forma que os obstáculos surgissem, o jogo ficaria muito fácil.

Inicialmente utilizávamos todas as sete bandas resultantes da subdivisão do espectro, mostrada na seção sobre o áudio. Decidimos sempre criar um obstáculo quando fosse registrada uma "batida" em determinada banda. Porém, isso resultou numa enorme quantidade de obstáculos, o que prejudicava a jogabilidade. Quisemos restringir isso, e uma das alternativas foi colocar um tempo mínimo de intervalo para cada banda, desse modo, mesmo que houvesse uma batida, apenas seria criado um novo obstáculo caso tivesse passado o tempo mínimo do intervalo.

Num primeiro momento, deixamos tanto formas, posições e cores de formas aleatórias, apenas o momento quando era gerado um obstáculo era determinado pela batida da

música, e o lado pelo qual este mesmo obstáculo aparecia era determinado de acordo com o intervalo do espectro onde ocorria a batida. Porém julgamos que o resultado ficou bem mais aleatório do que queríamos. Em momentos especifícos de uma música, quando se concentrava em um dos intervalos do espectro, até conseguíamos notar a diferença. Porém na maior parte do tempo, tínhamos vários obstáculos, e ficava difícil fazer a associação entre o que se via na tela de jogo e a música que estava tocando.

Decidimos restringir isso, e deixamos a posição em que surgiam obstáculos de forma mais fixa, e também fixamos as formas geométricas referentes a cada intervalo do espectro. A parte aleatória fica apenas por parte das cores. Essa abordagem gerou um resultado melhor que o inicial, e foi a que utilizamos na versão final.

Quanto aos detalhes das regras:

- a primeiras três bandas do espectro (*Subbass, Lowbass, Bass*) não são usadas para gerar obstáculos. Elas apenas influenciam o *frame* ao redor da tela, com isso quisemos passar a impressão dos espectros (geralmente em barras) que aparecem comumente em *players*.
- as bandas *LowerMidrange* e *UpperMidrange* são responsáveis pelos obstáculos maiores que aparecem pela horizontal e pela vertical, as formas são quadrados e hexágonos respectivamente. Como nessa área se concentra a maior parte das músicas, também incluímos uma nova restrição, apenas seria criado um obstáculo por vez, entre as duas. Assim, quando ambas registram uma batida, calculamos onde a alteração foi mais significativa, e criamos um obstáculo para a banda responsável por ela.
- As demais bandas são responsáveis pelos obstáculos menores e mais rápidos. A banda *Presence* é responsável pelos obstáculos que surgem a partir das diagonais com forma de losango. Com a banda *Brilliance*, por ser a mais rara em que acontecem grandes variações, resolvemos criar mais obstáculos. Quando é registrado uma batida nessa área, criamos seis obstáculos iguais a partir das laterais com forma de triângulos, que se movem de maneira simétrica uns aos outros.

As imagens a baixo mostram o resultado obtido e ilustram como cada tipo de obstáculo é gerado.

4.4.2 Interface

No final, restava desenvolver a interface e o gerenciamento das telas de jogo. Os estados são gerenciados usando uma pilha. Geralmente o estado ativo, é aquele presente no topo da pilha, embora nós possamos ter vários estados funcionando ao mesmo tempo. Por exemplo, quando estamos durante o jogo, podemos pausar, empilhando um novo estado (pausado) sobre o estado atual (jogando). Mesmo assim, o estado anterior continua sendo desenhado na tela, apenas não é atualizado.

As classes responsáveis pelos estados do jogo são:

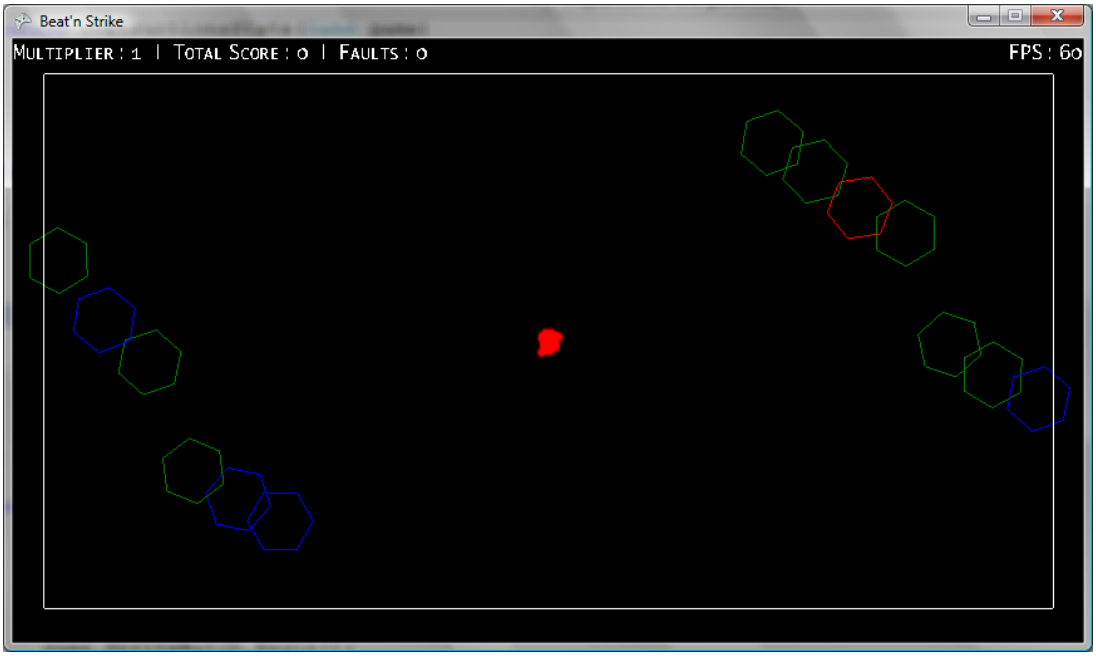

Figura 4.15: Obstáculos verticais gerados pelo intervalo *Lower Midrange*.

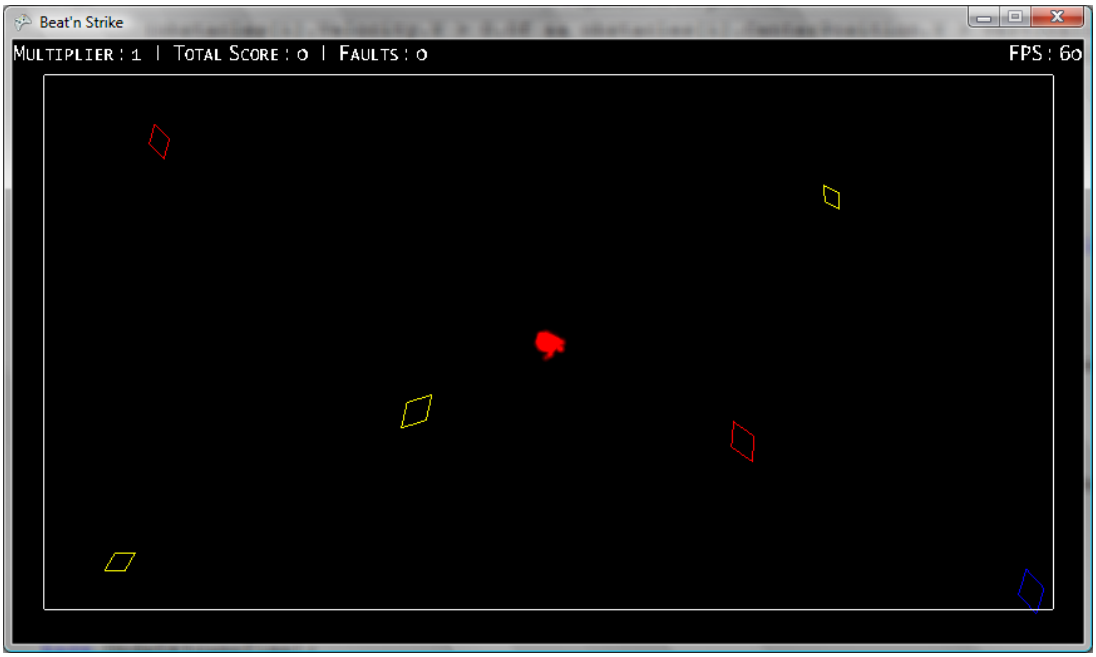

Figura 4.16: Obstáculos diagonais gerados pelo intervalo *Presence*.

GameStateManager: guarda a pilha dos estados do jogo, e é a responsável pelo gerenciamento dela. Sempre que o estado no topo é alterado, é enviado uma mensagem a todos os estados presentes na pilha.

GameState: estado de jogo que é armazenado no GameStateManager. Todas as classes abaixo herdam sua funcionalidade desta classe.

TitleIntroState: tela de início de jogo. A partir dela só é acessível o StartMenuState.

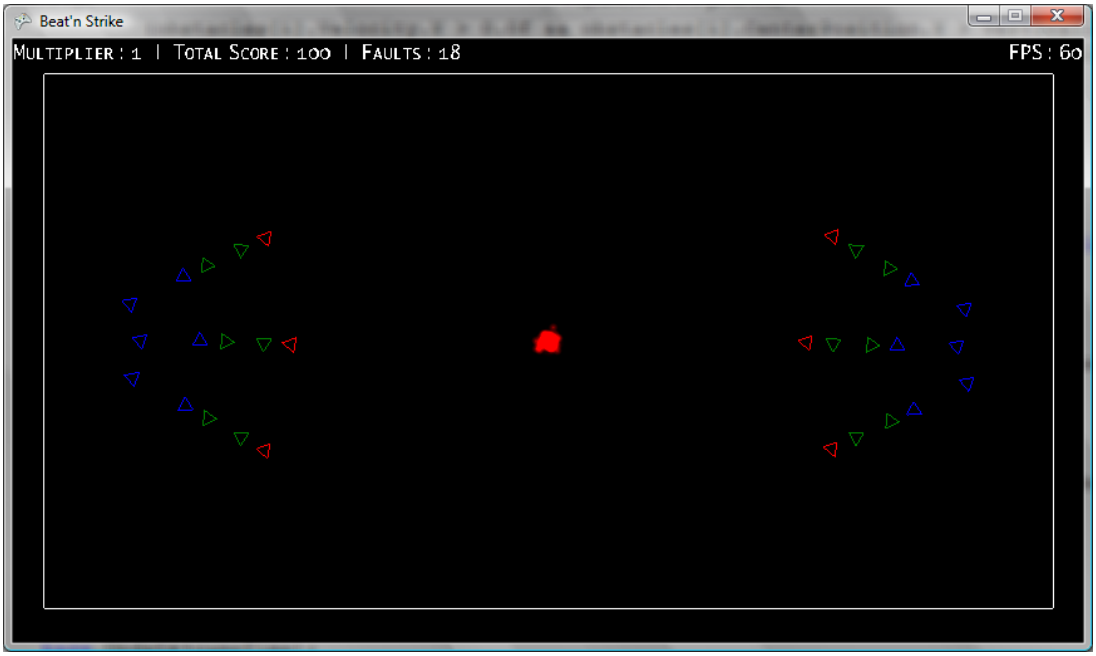

Figura 4.17: Obstáculos variados gerados pelo intervalo *Brilliance*.

StartMenuState: tela de menu principal, onde é possível escolher a música para o jogo. A partir dela podemos acessar o InstructionsState ou o PlayingState. Também podemos sair do jogo.

InstructionsState: tela que mostra as instruções de jogo. A partir dela, só se pode voltar ao StartMenuState.

PlayingState: tela onde ocorre o jogo propriamente dito. A partir dela, podemos ir para o PausedMenuState, ou SongOverState.

PausedMenuState: tela de pause, onde é mostrado um menu. Temos a opção de voltar a jogar, (PlayingState), de voltar ao menu principal (StartMenuState), ou de encerrar o jogo.

SongOverState: tela mostrada assim que a música é encerrada. São mostradas estatísticas sobre o desempenho do jogador. A partir dela só podemos ir para o RetryMenuState.

RetryMenuState: tela onde o jogador pode escolher jogar de novo com a mesma música, escolher uma nova, ou sair do jogo. A partir dela podemos ir para o PlayingState.

A figura a seguir ilustra as possíveis transições das telas.

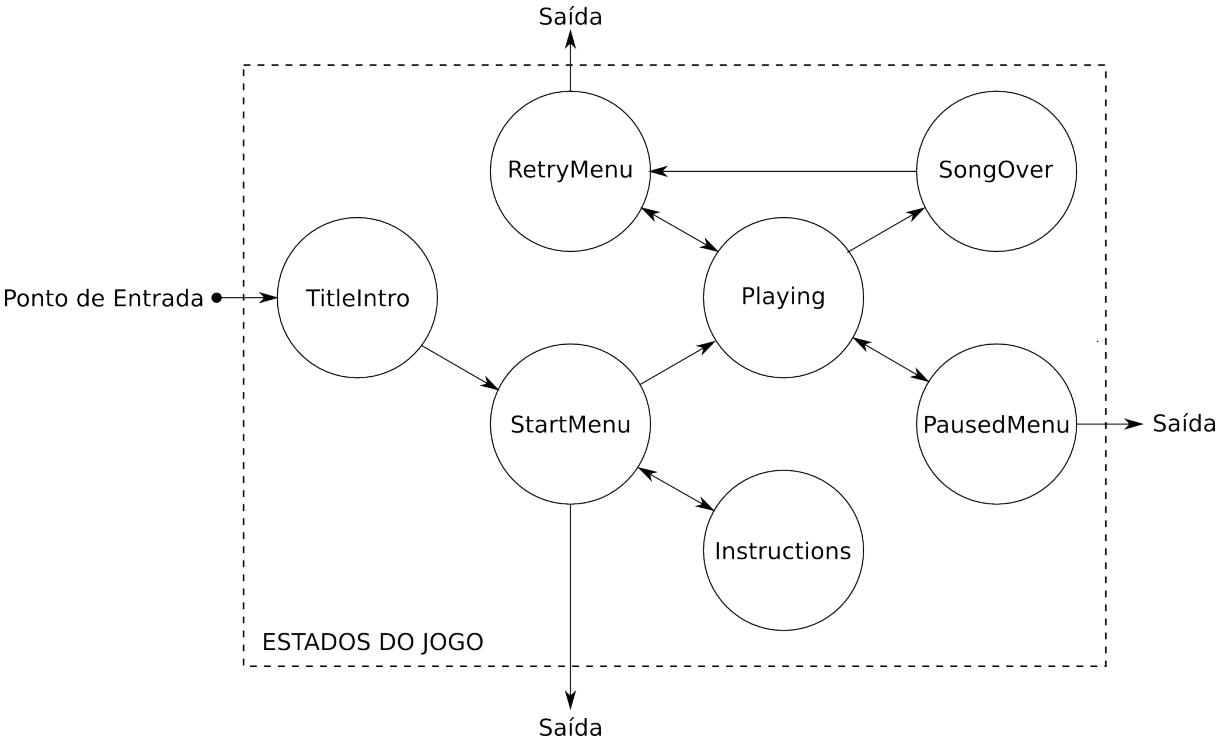

Figura 4.18: Transições das telas do jogo.

Capítulo 5

Portando *Beat'n Strike* para XBOX 360

Neste capítulo discutiremos o processo, as dificuldades e a conclusão que tivemos para portar o jogo *Beat'n Strike* para o console XBOX 360.

5.1 Conceitos Preliminares

5.1.1 Serviços *online*

Xbox LIVE[15]

Xbox LIVE é um serviço *online*, acessado pelo Xbox 360, oferecido pela Microsoft® que disponibiliza partidas de jogos *multiplayer* e mídia digital, como vídeos, músicas e jogos. Através do *Xbox LIVE Marketplace*[16] é possível comprar jogos, baixar demostrações e pacotes de atualização.

XNA Creators Club[17]

XNA Creators Club é um local onde os criadores de jogos para XBOX 360 podem enviar seus jogos feitos usando o *framework* XNA, caso aprovados, os jogos podem ser distribuídos através *Xbox LIVE Marketplace*. Para criar uma conta no XNA Creators Club e poder enviar jogos é necessário pagar uma taxa de 99 dólares para um plano de 12 meses e 49 dólares para um plano de 4 meses. A Microsoft também oferece uma versão para teste (*Trial*) para estudantes, através do MSDNAA (MSDN Academic Alliance).

5.1.2 Bibliotecas

DirectSound

Desenvolvido pela Microsoft®, DirectSound é um componente da biblioteca DirectX[12] presente no sistema operacional Windows®. O DirectSound oferrece uma interface entre aplicativos e o dispositivo de áudio, permitindo programas reproduzirem sons e música. Além de prover o serviço de passar os dados de áudio para a placa de som, ele oferece muitos outras características como gravação e mixagem de sons; adição de efeitos sonoros; captura sons de entradas de áudio como o microfone; posicionamento de sons no espaço 3D entre outros.

No jogo *Beat'n Strike* utilizamos o DirectSound juntamente com o decoder de mp3 para tocar as músicas do jogo.

Xaudio 2

Com a mesma função do DirectSound, o Xaudio 2 é uma evolução do Xaudio, uma API desenvolvida somente para o XBOX 360 para o processamento de sinais digitais, com um diferencial: Xaudio 2 é uma API de audio de baixo nivel *Cross-patform*, isto é, uma biblioteca que funciona tanto em Windows® quanto em Xbox, diferente do Xaudio, que funciona apenas no XBOX 360 e do DirectSound que funciona apenas em Windows®. Desenvolvida pela Microsoft® esta biblioteca foi adicionada no March 2008 DirectX SDK (Kit de desenvolvimento para DirectX), possuindo como alvo as plataformas Windows® (XP, Vista) e XBOX 360.

No jogo *Beat'n Strike* iríamos utilizar esta biblioteca para criar uma versão para o Xbox 360.

5.2 O Processo

5.2.1 Requisitos

Os requisitos mínimos para converter e executar um jogo desenvolvido utilizando o *framework* XNA[18] e a linguagem C# para o console XBOX 360 são:

- O console XBOX 360 juntamente com um HD (disco rígido próprio para o console);
- Cabo de rede ou modem Wi-Fi para ter acesso a uma rede LAN;
- Uma conta no Xbox LIVE;
- Uma conta no XNA Creators Club.

Figura 5.1: O console XBOX 360, o HD e o controle, respectivamente.

5.2.2 Como Converter o Jogo

Para converter o jogo três passos básicos são necessários, são eles:

- **Adicionar o XBOX 360 como dispositivo em um computador:** Utilizando o cabo de rede, conecta-se o XBOX 360 a mesma rede LAN do computador onde foi desenvolvido o jogo. Para o console ser reconhecido como um dispositivo, inicialmente é necessária uma chave de acesso, esta fornecida pelo próprio XBOX 360. Para se gerar esta chave é necessário o programa *XNA Game Studio Connect* baixado via o console através do *Xbox LIVE Marketplace*, para isso são essenciais as contas no Xbox LIVE e no XNA Creators Club. Com a chave de acesso em mãos, no computador, utilizando o programa *XNA Game Studio Device Center*, disponibilizado com o *framework* XNA, podemos adicionar o XBOX 360 como dispositivo de Hardware. A imagem abaixo mostra a tela do programa *XNA Game Studio Connect* em execução com o XBOX 360 sendo reconhecido pelo computador.
- **Converter um projeto para ser executado no XBOX 360:** No IDE Visual Studio 2008 basta clicar com o botão direito do mouse sobre o nome do projeto e escolher a opção *Create Copy of Project for Xbox 360*.
- **Compilar o projeto:** Caso esteja tudo certo com o projeto, basta compilá-lo e este será executado no XBOX 360.

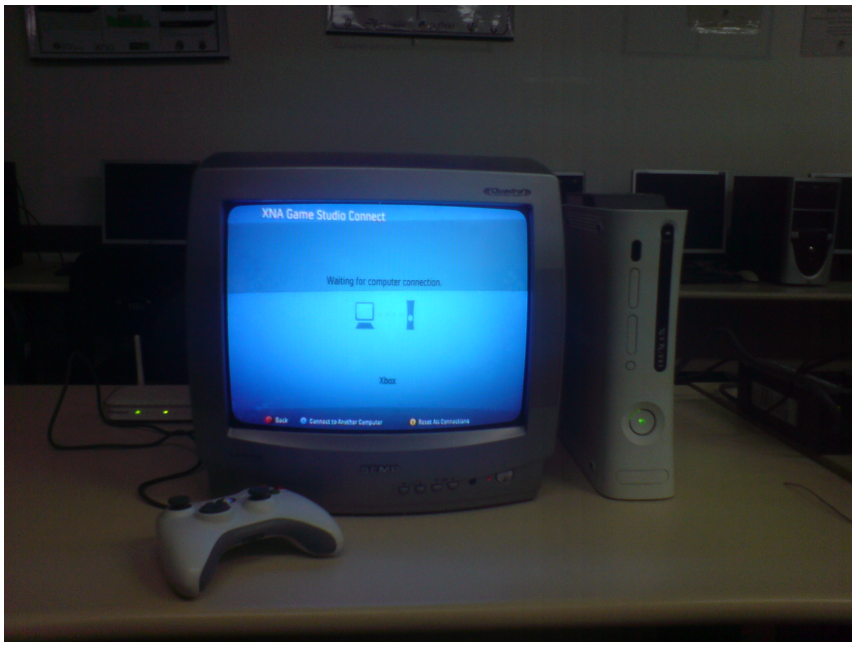

Figura 5.2: Programa *XNA Game Studio Connect* em execução.

5.2.3 Problemas Iniciais

O primeiro problema que enfrentamos foi a parte do cadastro ao abrir as contas no *Xbox LIVE* e no *XNA Creators Club*, foi necessária duas tentativas para conseguirmos criar um usuário que pudesse acessar a *Xbox LIVE Marketplace*. Este problema foi causado, pois o Brasil não esta na lista de países que possuem acesso aos serviços da *Xbox LIVE*, pois a Microsoft® não oferece suporte. Descobrindo isto da pior forma, criamos o segundo cadastro escolhendo como país de origem um dos países com acesso ao serviço.

Resolvido o primeiro problema passamos a seguir o processo descrito na seção anterior. Enquanto seguíamos o processo tudo estava indo bem, porém foi na última etapa ,compilar o projeto, que começamos a ter sérios problemas, pois uma séries de erros de execução apareceram.

Analisando os erros descobrimos que eles ocorriam em duas partes do sistema, no sistema de arquivos, e na biblioteca de áudio. Os erros no sistema de arquivos nós já tínhamos previsto que ocorreria, pois no jogo na versão para Windows® uma janela de arquivo aparece no momento em que o jogador for selecionar um arquivo MP3. Porém, este tipo de recurso não está disponível no XBOX 360. Já os erros na biblioteca de áudio foram inesperados. Com um pouco mais de pesquisa vimos que o acesso a dispositivos de áudio no Windows® é diferente do acesso de dispositivos de áudio no XBOX 360, pois os dois sistemas utilizam bibliotecas diferentes, o Windows® usa o DirectSound e o XBOX 360 usa o Xaudio 2.

5.2.4 Soluções Propostas Iniciais

Para os erros no sistema de arquivo, poderíamos modificar o modo de seleção de músicas. Desta forma poderíamos testar o conceito do *Beat'n Strike* e facilitar o processo de portabilidade. Como mudança, decidimos que as músicas seriam disponibilizadas com o jogo, isto é, o jogador deixaria de acessar o seu acervo pessoal, para acessar um acervo de músicas limitado e fixo. Com esta solução temporária, não precisaríamos da janela de arquivos.

Quanto aos erros na parte da biblioteca de áudio, como na versão para Windows® utilizamos o DirectSound, para resolver o problema deveríamos refatorar nossa biblioteca para executar a saída gerada pelo nosso *decoder* utilizando o Xaudio 2, mas como descreveremos mais a frente esta tarefa se tornou inviável.

5.3 Problemas Enfrentados e Dificuldades

Após muita pequisa infelizmente descobrimos que não é possível portar o jogo *Beat'n Strike* para o XBOX 360.

O *framework* XNA, apesar de ajudar no desenvolvimento de jogos, para tarefas de baixo-nível com acesso a dispositivos de hardware, do XBOX 360 principalmente, ele se mostra não muito eficaz. E apesar de ser possível executar um arquivo MP3 com o XNA,

como este framework faz isto de maneira muito alto-nível, oferecendo opção apenas para tocar, "pausar" e parar a música, ele não é viável para gerar os dados para a detecção de batidas.

Apesar do Xaudio 2 ser desenvolvido para ser usado tando em Windows® quanto em XBOX 360 sua escassa documentação e o fato de não dar suporte para a linguagem C# e sim a linguagem C++, utilizar esta biblioteca se tornou inviável para nosso propósito.

A solução para estes problemas seria usar o kit de desenvolvimento do Xbox (XDK), pois este kit possui todas as bibliotecas e um compilador para gerar um binário que execute no XBOX 360. Com um porém, ao invés de usar a linguagem C# precisaríamos reescrever todo o código para a linguagem C++ pois é esta linguagem que o XDK utiliza e da suporte.[1]

Outra possível solução seria desenvolver uma dll para fazer uma interface entre o C# e o C++, para assim acessar a biblioteca Xaudio 2, tarefa esta muito trabalhosa devido a pouca documentação e tempo, além de ser de grande risco, pois pode não funcionar.

Com todos estes problemas portar o jogo *Beat'n Strike* para o XBOX 360 tornou-se uma tarefa inviável para este projeto.

5.4 Solução Parcial

Vendo que portar *Beat'n Strike* para o Xbox 360 seria inviável e para verificar que os erros estavam somente na parte de áudio, decidimos alterar o conceito do jogo, tirando a parte da música e passando a gerar obstáculos aleatoriamente. Com isso queríamos demostrar que a parte gráfica que desenvolvemos seria perfeitamente portável para Xbox 360. Abaixo estão algumas imagens do jogo somente com a parte gráfica sendo executada.

Figura 5.3: Tela inicial do jogo.

[1] O acesso á um kit de desenvolvimento para XBOX 360 é limitado a desenvolvedores que trabalham em títulos aprovados por *publishers* licenciados pela Microsoft®.

Figura 5.4: Jogo sendo executado sem a parte de áudio.

5.5 Resultados

Inicialmente pensamos que seria tranquilo a tarefa de portar o jogo *Beat'n Strike* para XBOX 360. Acreditávamos que o principal desafio seria o acesso ao sistema de arquivos do console, mas vimos que o que realmente se tornou um problema foi uma das partes mais importantes do jogo, o áudio, devido a incompatibilidade das bibliotecas entre o Windows® e o XBOX 360 além da pouca documentação. Outro empecilho que não imaginávamos foi a linguagem, pois a biblioteca Xaudio 2 utilizam a linguagem C++ ao invés de C#, assim como o XDK.

Desta forma portar o jogo *Beat'n Strike* para XBOX 360 não é viável nesta etapa e passa a ser um grande projeto futuro, devido as tamanhas modificações.

Capítulo 6

Conclusão

Este capítulo, apresenta os resultados obtidos neste projeto, os desafios que encontramos no processo de desenvolvimento e os próximos passos a serem seguidos no futuro do *Beat'n Strike*.

6.1 Resultados

Como resultado tivemos dois softwares:

- Uma biblioteca de áudio capaz de decodificar e tocar uma música de um arquivo MP3;
- Um jogo completo, desafiador e divertido.

6.1.1 Biblioteca de Áudio

A biblioteca de áudio que implementamos pode ser base para qualquer *player*, pois ela é capaz se decodificar e tocar uma música. Porém, ela funciona somente no Windows®, não sendo compativel para o XBOX 360.

6.1.2 O jogo

O jogo *Beat'n Strike*, ao final de seu desenvolvimento, é um jogo completo no qual o jogador carrega uma música em formato MP3 e joga com o objetivo de ganhar mais pontos possíveis, colidindo o cometa com os objetos.

Abaixo algumas *screenshots* do *Beat'n Strike*.

6.2 Desafios

Durante o processo de desenvolvimento encontramos muitos desafios, principalmente na parte do áudio. Apesar do arquivo MP3 ser bem popular, encontramos muitas informações sobre o processo de *decoding*, mas não encontramos muita documentação a respeito de como implementar o algoritmo de decodificação deste tipo de arquivo. Então para contornar este problema analisamos códigos fontes de *players* de MP3 *open source* para descobrir como implementar nossa biblioteca de áudio.

Figura 6.1: Tela inicial do jogo.

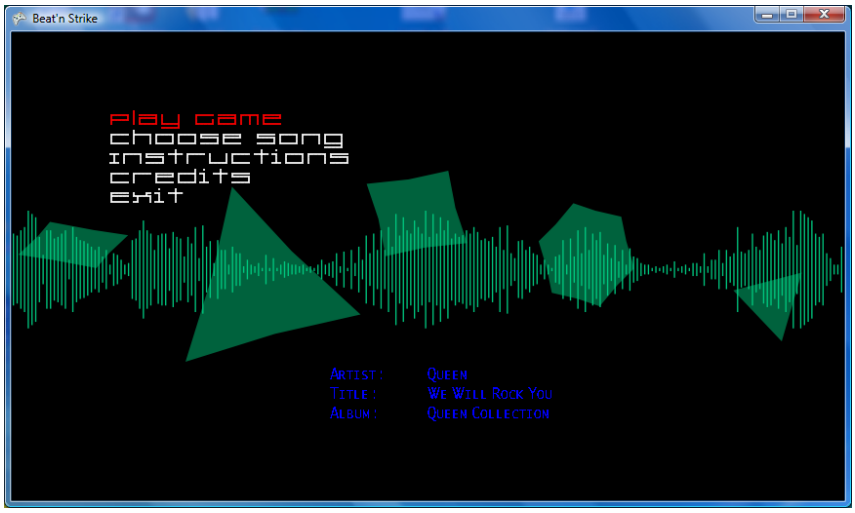

Figura 6.2: Menu Principal.

Encontrar algoritmos para detecção de batidas é uma tarefa um pouco frustrante. Apesar de parecer bem óbvio para o ouvido humano, é algo difícil de formalizar. Os melhores algoritmos para este fim são aqueles que se adequam a um problema específico. Quanto mais restrito e definido, melhor é o resultado esperado. Nós conseguimos, mais ou menos, aproximar a detecção de batidas para o nosso caso. Nosso problema era bem genérico, e incluía uma grande variação de entradas, já que a música seria escolhida pelo usuário. Por isso tivemos que cobrir todo o espectro.

Outra dificuldade que tivemos foi criar testes para o algoritmo. Apesar de utilizarmos um modelo de processo que descreve que testes são fundamentais, não criamos muitos testes devido a natureza dos dados que tínhamos, uma quantidade enorme de dados que precisam ser tratados de forma eficiente.

Por fim, portar para o console XBOX 360 se tornou um grande desafio, pois de inicio acreditávamos que isto seria simples. Porém, devido a diversos problemas com a incompatibilidade de bibliotecas de áudio entre o Windows e o XBOX 360, não foi possível realizar esta tarefa.

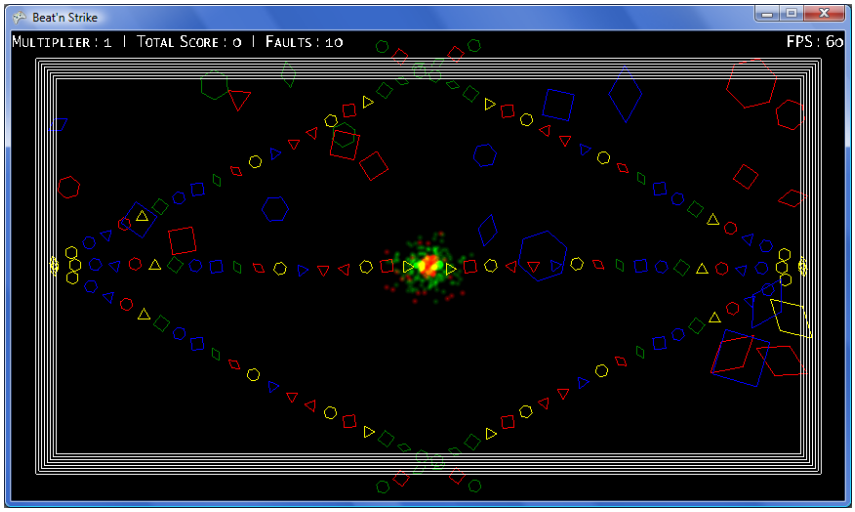

Figura 6.3: Tela do jogo.

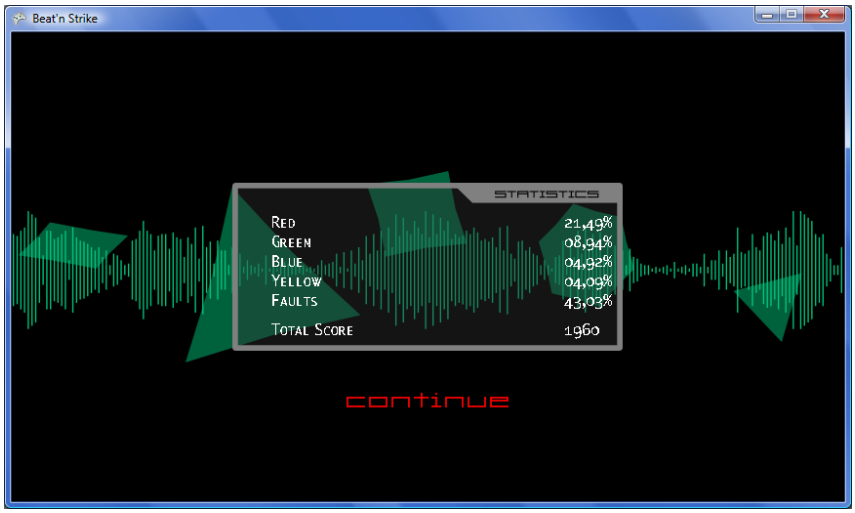

Figura 6.4: Estatística e Pontuação ao final de uma partida.

6.3 Próximos Passos

Apesar de termos um jogo completo, muitas melhorias e características podem ser adicionadas ao jogo. Entre elas estão:

- Salvar o placar, tanto localmente quanto *online*;
- Criar diferentes objetivos para o jogo, assim como itens especiais para aumentar a diversão do jogo;
- Melhorar os gráficos do jogo, adicionando mais efeitos e explosões;
- Permitir escolha de dificuldades;
- Adicionar um modo *multiplayer*, onde de dois a quatro jogadores podem jogar simultaneamente;

- Permitir que o jogador crie novos dos modos de jogo;
- Melhorar o algoritmo de detecção de batidas na música.

Como passo futuro pretendemos implementar estas melhorias e talvez tentar exportar o conceito do jogo para plataformas móveis, como iPhone e Android.

II. Parte Subjetiva

Capítulo 7

Parte Subjetiva

7.1 Alexandre Barbosa de Macedo

Sempre tive vontade de desenvolver um jogo. Durante o curso, tivemos oportunidades de desenvolver alguns jogos como EP em algumas disciplinas, porém eu ainda queria fazer um jogo de alto nível, que pudesse se equiparar ao de alguma pequena produtora, por exemplo. Por isso, fazer um jogo como trabalho de formatura pareceu um bom caminho. Além de me divertir e fazer algo que eu desejava, poderia aproveitar o projeto em uma disciplina.

7.1.1 Desafios e frustrações

As principais dificuldades que encontrei:

- Falta de informações sobre o MP3. Isso não deve ser entendido como uma falta de documentação sobre o MP3. É muito fácil encontrar a especificação do formato na internet, sobre como interpretar os *headers*, etc, mas não há muita informação sobre os algoritmos envolvidos no processo de *encoding/decoding*. Estudamos implementações de *decoders* para tentar aprender, mas poucos são voltados a aprendizado, e muitas vezes incluíam diversas técnicas para melhorar a performance, o que confundia o entendimento do código. Foi necessário muita pesquisa e estudo para entender cada etapa do processo, que não é muito simples.

- Pouca informação sobre o HLSL, a linguagem de *shader* da *Microsoft*. De novo, isto não deve ser entendido como uma falta de documentação. Há muitos tutoriais e exemplos de código, porém eu os considerei "desbalanceados". Geralmente, ou o exemplo era muito simples, e não ajudava muito, ou era muito complicado, bem além do que necessitávamos, fazendo com que entendêssemos muito pouco. E utilizar o HLSL era a única opção para implementar sistema de partículas. Isso fez com que perdêssemos um tempo além do necessário para implementar algo que considerávamos simples.

- Administrar as prioridades. Tínhamos várias idéias, e devíamos ter cuidado para não nos empolgar demais. O próprio conceito do jogo mudou durante o desenvolvimento, e várias coisas que pensamos inicialmente, deixamos de fora do jogo final. Era necessário terminar o projeto a tempo, mesmo que significasse excluir algumas funcionalidades.

- Ajustar o *gameplay* foi um grande desafio. Queria deixar clara a impressão de que a música de fato está gerando o cenário. De início não havia me contentado muito com o resultado, porém com o tempo a mais que nos foi dado, considerei o resultado satisfatório. Ainda não está perfeito, pois depende muito da música escolhida pelo jogador, e tentamos achar um "meio termo", que funcionasse de maneira razoável para diferentes tipos de música.
- Portar para o XBOX 360. A primeira frustração foi ter perdido minha conta do *XNA Creators Club* ao descobrir que a *XBOX Live* não oferece o serviço para o Brasil. Tivemos que utilizar outra conta, mudando a localidade para os EUA. A segunda foi não ter conseguido portar o jogo plenamente. Apesar de conseguir rodar a parte gráfica, o jogo não tem a mesma dinâmica e diversão da versão que roda no PC.

7.1.2 Disciplinas relevantes e observações

Entre as disciplinas cursadas que eu considerei mais importantes para a realização deste projeto estão:

MAC0122 - Princípios de Desenvolvimento de Algoritmos É a disciplina base dos cursos da computação, onde aprendemos o fundamental de estruturas de dados e algoritmos.

MAT0139 - Álgebra Linear para Computação Foi muito importante quando fiz Computação Gráfica. Diversos conceitos como produto escalar ou produto interno são fundamentais quando estamos trabalhando em um sistema de coordenadas 3D. Transformações lineares também são importantes para o mapeamento de texturas, etc.

MAC0323 - Estruturas de Dados Foi uma disciplina importante, pois não apenas nos diz qual estrutura é mais apropriada para cada tipo de problema, também nos fez pensar e adaptar nossas próprias estruturas de dados.

MAC0211 - Laboratório de Programação I Primeira disciplina em que fizemos um jogo. Além de despertar o interesse, ajudou em diversos conceitos mais gerais, e permitiu adquirir uma pequena experiência na área.

MAC0441 - Programação Orientada a Objetos Importante, pois o C# também é uma linguagem orientada a objetos. Diversas técnicas puderam ser aproveitadas durante o desenvolvimento, além de buscar sempre algumas qualidades no código, como a diminuição do acoplamento, etc.

MAC0332 - Engenharia de Software Disciplina importante para dar uma visão geral do projeto. Ajudou durante o planejamento e a organização, tanto do estudo quanto do desenvolvimento.

MAC0420 - Introdução à Computação Gráfica Disciplina que ajudou muito. Além de ajudar com os conceitos teóricos, nesta disciplina usamos *OpenGL* para desenvolver os EPs, que apesar de diferente do XNA, guarda muitas semelhanças. Diversas coisas mais práticas como definir volume de visualização, posicionar a câmera, etc, foram úteis ao trabalhar com o XNA.

MAT0221 - Cálculo Diferencial e Integral IV Em cálculo 4 aprendemos sobre séries e sequências. Apesar de não aplicarmos diretamente, foi importante para entender os conceitos por trás do *encoding* do MP3, como a *Fast Fourier Transform* e a *Modified Discrete Cosine Transform*.

MAE0121 - Introdução à Probabilidade e Estatística I Disciplina em que aprendemos os conceitos básicos, como média, variância, desvio padrão, etc. Foram importantes durante o desenvolvimento do algoritmo que verifica o ritmo da música.

FAP0126- Física I Disciplina que fala sobre cinemática e dinâmica, que foram usadas para dar um aspecto mais real quando desenvolvemos o sistema de partículas.

7.1.3 Planos Futuros

Gostei bastante de desenvolver o *Beat'n Strike*, e por isso não pretendo parar o desenvolvimento. Ainda podemos melhorar a parte gráfica do jogo e adicionar diversas funcionalidades que pensamos durante o desenvolvimento, mas que não incluímos para não comprometer o prazo de entrega.

Apesar de não incluir o áudio, ver o jogo rodando no XBOX foi bem empolgante. A experiência deixa vontade de criar mais jogos para esta plataforma.

Outra coisa que planejamos é portar o jogo para outras plataformas, relacionadas principalmente a *Smart Phones*, como o *Android* ou *iPhone*.

7.2 Daniel Grecco Machado

A primeira impressão quando pensei em desenvolver um jogo foi de que seria algo de pouca relevância acadêmica, principalmente por isso optamos por adicionar algum diferencial no jogo que acabou sendo o áudio. Mas este sentimento rapidamente mudou, ou seja, depois muito estudo a respeito de áudio e computação gráfica e ainda com o desenvolvimento precoce do projeto fui percebendo a quantidade grande de aplicações das disciplinas do BCC de que iríamos utilizar.

7.2.1 Desafios e frustrações

O objetivo era desenvolver um jogo completo ainda que minimalista do ponto de vista da arte gráfica. Este ponto me preocupou um pouco pois durante o curso poucas vezes desenvolvi um projeto desde a sua concepção até a arte final incluindo IHC de modo que o projeto pudesse ser distribuído, ao mesmo tempo este aspecto soou como um desafio dado que na maioria das oportunidades do curso o foco foi trabalhar no *core* das aplicações desenvolvendo algoritmos eficientes, respeitando padrões e convenções, o que foi extremamente importante para o projeto do jogo, contudo vi no desenvolvimento do jogo a oportunidade de consolidar estes conhecimentos produzindo uma aplicação do início ao fim.

Outro desafio surgiu durante os estudos de ferramentas e algoritmos de manipulação de arquivos mp3. As informações que queríamos extrair da música só poderiam ser obtidas "no meio" do processo de decodificação o que nos forçou a desenvolver um *decoder* próprio. Para isto precisaríamos de muita documentação. Como mencionado em uma das seções o padrão MPEG1 é muito bem definido mas os algoritmos de manipulação bem como a implementação das funções matemáticas envolvidas nos processos de codificação e decodificação não são bem documentadas. Após vários dias pesquisando livros que pudessem suprir estas dificuldades descobrimos um livro que parecia resolver o nosso problema pois era citado como referência em vários outros livros, porém a biblioteca da USP não dispunha de um exemplar deste e a única opção era comprá-lo pelo amazon.com. Mas esta alternativa não existia pois a data máxima de entrega do livro extrapolava a entrega do Trabalho Supervisionado de Formatura além do valor ser um tanto caro. A solução foi adotar um decodificador *open source* (JavaZoomLayerIII) como base para desenvolvermos um próprio.

Aos poucos as partes do jogo começavam a ficar prontas e a empolgação em ver o projeto crescendo aumentava. Logo muitas idéias novas surgiam a cada momento porém o comprometimento de completar o jogo nos impediu de levá-las adiante. Neste aspecto a ajuda do orientador foi muito importante, afinal, estávamos desenvolvendo um projeto de formatura antes de qualquer outra coisa e esta empolgação rapidamente nos tiraria do foco principal.

Apesar da grande dificuldade de manipular o som, o resultado final foi muito satisfatório. A audição humana lida muito naturalmente com as batidas das músicas, e ensinar isto ao computador se mostrou um desafio muito grande. Na verdade nós conseguimos definir estes fenômenos musicais e capturá-los porém representá-los de maneira a "desenhar" a música na tela foi extremamente complicado e tomou muito tempo não pela complexidade dos códigos envolvidos, pois não eram, mas pelas inúmeras tentativas diferentes de representação.

7.2.2 Disciplinas relevantes e observações

MAC0122 - Princípios de Desenvolvimento de Algoritmos Acredito que esta matéria tenha sido a mais importante por formar a base para o entendimento de todas as outras. Um exemplo direto da aplicação desta matéria no projeto é a codificação de Huffman que utiliza árvore binária completa na sua implementação.

MAT0139 - Álgebra Linear para Computação No desenvolvimento de jogos os conceitos de manipulação de vetores no espaço é essencial. Esta matéria teve várias aplicações principalmente com funções auxiliares de parte gráfica: Norma, produto escalar, etc.

MAC0323 - Estruturas de Dados As estruturas de dados estão presentes em todos os lugares do projeto mas foi cuidadosamente aplicada no sistema de partículas. Um exemplo claro foi o uso de *structs* e filas circulares para o armazenamento de partículas pois a interface que nos permitia "conversar" com a GPU nos restringiu ao uso destas estruturas primitivas.

MAC0211 - Laboratório de Programação I Foi o primeiro contato com o desenvolvimento de um projeto completo além de introduzir a ferramenta para desenvolvimento colaborativo *svn* que foi largamente utilizado durante todo o projeto.

MAC0338 - Análise de Algoritmos Num jogo em tempo real é muito importante para manter os quadros por segundo aceitáveis. Durante o desenvolvimento nos preocupamos em implementar algoritmos eficientes de maneira a não comprometer o desempenho do jogo.

MAC0441 - Programação Orientada a Objetos Imprescindível em qualquer aplicação que utilize linguagens orientadas a objetos, a utilização de padrões facilitam o desenvolvimento e a depuração além de permitir uma flexibilidade grande quando se trata de melhorias e modificações futuras. Um exemplo direto é o padrão *observer* que foi utilizado no gerenciamento de estados.

MAC0332 - Engenharia de Software Outra disciplina que foi extremamente útil, nos ajudou a organizar o desenvolvimento baseado em uma equipe pequena com prazos curtos.

MAC0420 - Introdução à Computação Gráfica Os conceitos aprendidos nesta disciplina foram amplamente explorados para manipulação de câmera, vetores, efeitos, etc. Não há dúvidas a respeito da utilidade deste tópico no projeto do jogo.

MAE0121 - Introdução à Probabilidade e Estatística I Durante o desenvolvimento do algoritmo de detecção da batidas usamos análises estatísticas. Já na codificação do mp3, noções de distribuição de probabilidades foram úteis para estudar a codificação de Huffman, etc.

FAP0126 - Física I Utilizamos noções de física básica para projetar as forças e os movimentos dos corpos.

MAC0342 - Laboratório de Programação Extrema Muitos dos conceitos de grande relevância aplicados durante o desenvolvimento do projeto foram ensinados nesta diciplina.

A maioria dos conhecimentos utilizados são básicos, mas em conjunto abrangem quase todas as áreas estudadas no curso de Ciências da Computação. Não mencionei computação musical por não ter tido oportunidade de cursá-la, porém muitos dos estudos foram dirigidos abordando esta área.

7.2.3 Planos Futuros

A idéia agora é trabalhar na parte estética e no gameplay para melhorar a sensação de jogabilidade e aprimorar a experiência de jogo do jogador. Apesar das dificuldades de portar, ainda que parcialmente, o jogo para XBOX 360, foi muito gratificante jogá-lo a partir do console, o que nos estimulou ainda mais a desenvolver novos projetos deste tipo para esta e outras plataformas. Um dos nossos objetivos é desenvolver esta idéia em plataformas móveis como IPhone e Android.

7.3 Fábio Tsuguta Matsumoto

Ao entrar na computação nunca imaginei que faria um jogo no trabalho de formatura, porém devido a duas disciplinas que cursei durante o curso, Laboratório de Programação I e Engenharia de Software, onde os trabalhos destas disciplinas foram desenvolver um jogo, comecei a tomar gosto pelo desenvolvimento deste tipo de software.

7.3.1 Desafios e frustrações

Muitos desafios surgiram com este trabalho, o primeiro deles foi a escolha do tipo de software que queria desenvolver. Pensando junto com o Alexandre e o Daniel acabamos por escolher um jogo. Inicialmente pensamos que academicamente este tipo de software era pouco relevante, mas durante sua implementação vimos que este tipo de sistema utiliza muito do que aprendemos durante o curso de computação, além de ser um projeto que todos os integrantes do grupo tinham interesse em trabalhar.

Depois de escolhido que desenvolveríamos um jogo, começamos o pensar que tipo de jogo faríamos. Analisando nossas habilidades artísticas e influenciados por outros jogos, decidimos criar o jogo *Beat'n Strike*. Claro que do conceito até o produto final algumas alterações foram feitas.

Pessoalmente, durante o processo de desenvolvimento do jogo *Beat'n Strike* encontrei muitos outros desafios, dentre eles os que mais deram trabalho para resolver estavam ligados a parte do áudio. Como não tinha experiência neste tipo de área, tive que pesquisar e aprender muita coisa sobre música. Além disso outro desafio encontrado foi trabalhar com MP3, apesar de encontrar muita informação a respeito deste tipo de compressão de áudio, implementar um *decoder* e um *player* não foi uma tarefa fácil.

Outro desafio encontrado foi portar o jogo para o console Xbox 360 e mais uma vez foi na parte de áudio que este desafio se encontra. Como não sabíamos da diferenças de bibliotecas de áudio entre o Windows e o Xbox 360, verificar se existia alguma solução para podermos implementar a versão para o Xbox 360 levou a muita pesquisa e estudo, para mesmo assim não conseguimos encontrar ou descobrir nada viável para o tempo que possuíamos.

Com o tempo a mais que nos foi fornecido, a frustração que tinha antes com relação à geração de obstáculos foi amenizada. Acho que no tempo que foi dado este é o melhor que poderíamos fazer. Acredito que tirando um pouco da aleatoriedade que existia antes melhorou bastante a dinâmica do jogo, criando um padrão para cada música e dando a impressão de ver a música, ainda que de vez em quando, dependendo do tipo da música ainda esteja impreciso.

7.3.2 Disciplinas relevantes e observações

Esta são as disciplinas que cursei durante o curso de computação que foram fundamentais para o desenvolvimento deste projeto.

MAC0110 - Introdução à Computação: como entrei na computação sem saber muito sobre programação, esta disciplina foi fundamental, pois nela tive meu primeiro contato com linguagens de programação.

MAC0122 - Princípios de Desenvolvimento de Algoritmos: disciplina essencial, pois oferece toda a base para o desenvolvimento de algoritmos mais complexos.

MAC0211 - Laboratório de Programação I: primeira disciplina onde tive que desenvolver um grande projeto dividido em fases e onde tive contato e gosto por criar jogos, pois este era o projeto quando cursei esta disciplina.

MAC0323 - Estruturas de Dados: disciplina muito importante, pois apresenta estruturas de dados mais complexas, além de uma lógica mais profunda para o desenvolvimento de algoritmos mais eficientes. Foi muito útil neste projeto, principalmente na parte de decodificação do MP3.

MAC0332 - Engenharia de Software: disciplina que apresentou métodos e modelagens para a construção de projetos, estes utilizados neste jogo, para a organização e controle do trabalho.

MAC0342 - Laboratório de Programação Extrema: assim como em Engenharia de Software, apresentou métodos para desenvolvimento de sistemas, mas com outros princípios e abordagem, a programação ágil.

MAC0420 - Introdução a Computação Gráfica: disciplina que apresentou muitas teorias ligadas a criação, manipulação e processamento de gráficos, estas amplamente utilizadas no projeto.

MAC0441 - Programação Orientada a Objetos: disciplina na qual aprendi os padrões de desenvolvimento e a técnica de programação orientada a objetos, útil para o projeto devido ao uso da linguagem C#, uma linguagem orientada a objetos.

MAE0121 - Introdução à Probabilidade e Estatística I: disciplina na qual aprendi o básico de estatística e probabilidade, essencial para este projeto na parte de decodificação do MP3 e no algoritmo de detecção de batidas.

FAP0126 - Física I: disciplina onde aprendi e revi conceitos sobre cinemática, conceito de força e as leis de Newton, utilizados neste trabalho na implementação da Física do jogo.

7.3.3 Planos Futuros

No futuro ainda pretendo trabalhar no desenvolvimento deste jogo, mesmo como um projeto pessoal, para implementar as melhorias que podemos realizar, como adicionar outros modos de jogo, níveis de dificuldade entre outros. Quanto a divulgar o *Beat'n Strike* no *XNA Creators club*, isto pode ficar para um projeto futuro, quando tivermos acesso a um kit de desenvolvimento para o Xbox (XDK). Mesmo assim tenho a intenção de participar de campeonatos como o *Imagine Cup* e o *XNA Challenge* e tentar aplicar o conceito do jogo para outras plataformas como SmartPhones (iPhone e Android).

Referências Bibliográficas

[1] Activision: *Guitar Hero*. Disponível em: <http://hub.guitarhero.com/>. Acesso em: 28 nov. 2009.

[2] Bizarre Creations: *Geometry Wars*. Disponível em: <http://www.bizarrecreations.com/games/geometry_wars_retro_evolved/>. Acesso em: 28 nov. 2009.

[3] Carter, Chad: *Microsoft XNA Unleashed*. Sams, 2007, ISBN 0672329646.

[4] Chandraiah, Pramod e Dömer, Rainer: *Specification and Design of a MP3 Audio Decoder*. 2005.

[5] Dylan Fitterer: *Audio Surf*. Disponível em: <http://www.audio-surf.com/>. Acesso em: 28 nov. 2009.

[6] EA: *Rock Band*. Disponível em: <http://www.rockband.com/>. Acesso em: 28 nov. 2009.

[7] Hacker, Scot: *MP3: The Definitive Guide*. O'Reilly, 2000, ISBN 1565926617.

[8] Kniberg, Henrik: *Scrum and XP from the Trenches*. 2007, ISBN 9781430322641.

[9] Lagerström, Krister: *Design and Implementation of an MPEG-1 Layer III Audio Decoder*. 2001.

[10] Microsoft: *C#*. Disponível em: <http://msdn.microsoft.com/pt-pt/vcsharp/default(en-us).aspx>. Acesso em: 28 nov. 2009.

[11] Microsoft: *Common Language Runtime*. Disponível em: <http://msdn.microsoft.com/pt-br/library/8bs2ecf4.aspx>. Acesso em: 28 nov. 2009.

[12] Microsoft: *DirectX*. Disponível em: <http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx>. Acesso em: 28 nov. 2009.

[13] Microsoft: *HLSL*. Disponível em: <http://en.wikipedia.org/wiki/High_Level_Shader_Language>. Acesso em: 28 nov. 2009.

[14] Microsoft: *XBOX 360*. Disponível em: <http://www.xbox.com/pt-br>. Acesso em: 28 nov. 2009.

[15] Microsoft: *Xbox LIVE*. Disponível em: <http://www.xbox.com/en-US/LIVE/>. Acesso em: 27 jan. 2010.

[16] Microsoft: *Xbox LIVE*. Disponível em: <http://marketplace.xbox.com/en-US/>. Acesso em: 29 jan. 2010.

[17] MICROSOFT: *Xbox LIVE*. Disponível em: <http://creators.xna.com/en-US/>. Acesso em: 27 jan. 2010.

[18] MICROSOFT: *XNA*. Disponível em: <http://www.xna.com/>. Acesso em: 28 nov. 2009.

[19] MP3-TECH: *MP3*. Disponível em: <http://www.mp3-tech.org/>. Acesso em: 28 nov. 2009.

[20] NINTENDO: *Wii*. Disponível em: <http://www.nintendo.com/wii>. Acesso em: 28 nov. 2009.

[21] PATIN, Frédéric: *Beat Detection Algorithms*. Disponível em: <http://www.gamedev.net/reference/programming/features/beatdetection/>. Acesso em: 28 nov. 2009.

[22] PRESSMAN, Roger S.: *Engenharia de Software*. McGraw-Hil, 2006, ISBN 8586804576.

[23] REED, Aaron: *Learning XNA 3.0*. O'Reilly, 2008, ISBN 0596521952.

[24] SONY: *Playstation 3*. Disponível em: <http://www.playstation3.com.br/>. Acesso em: 28 nov. 2009.

[25] WIKIPEDIA: *Shader*. Disponível em: <http://en.wikipedia.org/wiki/Shader>. Acesso em: 28 nov. 2009.