CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA

FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

UMBRA - JOGO PARA ENSINO NO DESENVOLVIMENTO DE SHADERS

LEONARDO ADEMIR TONEZI DOS SANTOS

ORIENTADOR(A): PROF. DR ALLAN CESAR MOREIRA DE OLIVEIRA

Marília - SP
Dezembro/2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA

FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

UMBRA - JOGO PARA ENSINO NO DESENVOLVIMENTO DE SHADERS

LEONARDO ADEMIR TONEZI DOS SANTOS

Monografia apresentada ao Centro Universitário Eurípides de Marília como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.
Orientador(a): Prof. Dr. Allan Cesar Moreira de Oliveira

Marília - SP

Dezembro /2017

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA - UNIVEM
MANTIDO PELA FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ATA DE SESSÃO DE DEFESA DO TRABALHO DE CURSO

O Trabalho do Curso de Graduação em Ciência da Computação intitulado "UMBRA - Um Jogo para Ensino no Desenvolvimento de Shaders", elaborado por Leonardo Ademir Tonezi dos Santos, RA nº. 54020-1, 8º A-S Noturno, foi apresentado e defendido em sessão de argüição e avaliação, em 30 de novembro de 2017, nas dependências desta instituição de ensino, perante a banca examinadora formada pelos membros abaixo assinados, tendo obtido aprovação com a nota 8 (oito) e sido julgado adequado para o cumprimento do requisito legal previsto e regulamentado no Regulamento do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília - Univem.

Marília, 30 de novembro de 2017.

Orientador: **Allan Cesar Moreira de Oliveira**

Examinador 1 : **Emerson Alberto Marconato**

Examinador 2 : **Paulo Rogerio de Mello Cardoso**

Dedico este projeto a meus pais, que sempre me apoiaram.

Agradecimento

Gostaria de agradecer, primeiramente, a meus pais, Lenita de Alencar Tonezi dos Santos e Gilberto dos Santos pelo amor, suporte e incentivo ao longo dos anos.

Ao meu orientador e amigo, Prof. Dr. Allan Cesar Moreira de Oliveira, que guiou e motivou meu desenvolvimento acadêmico.

Aos Professores Bruno Marques, Maurício Duarte e Fábio Dacêncio, pelas aulas maravilhosas.

Aos meus amigos, João Ricardo, Frederico Soares, Thiago Costa, Alisson Solitto, Matheus Ferraroni e demais colegas de turma.

Por fim, a UNIVEM, que sempre deu suporte e motivação a todos os alunos.

Você deve entender que há mais de um caminho para o topo da montanha.

Miyamoto Musashi

RESUMO

Com o crescimento recente e significativo da tecnologia de processamento gráfico(GPU), a ciência vêm progredindo de uma forma absurdamente grande, trazendo muitos benefícios em áreas de estudo como da Medicina, Engenharia, Aeronáutica, etc. Com o surgimento do pipeline programável, ou seja, a permissão de programação em certos estágios do pipeline na renderização, houve o nascimento de um novo conceito chamado Shaders, que são técnicas e metodologias de programação deste pipeline. Porém, a propagação do conhecimento desta tecnologia é escasso, visto que é um método tanto recente quanto complexo, tornando árduo o desenvolvimento de novas aplicações neste cenário.

Sendo assim, tornou-se viável a implementação de uma plataforma Desktop de noviciado e treinamento ao desenvolvimento de Shaders, definindo técnicas e métodos utilizando conceitos de programação básica. O processo da gamificação vêm como um complemento para este projeto, trazendo uma espécie de “motivação” para o usuário.

Palavras-chave: Jogo, Jogo Educacional, Computação Gráfica, Shader, Plataforma, 2D, GPU, Renderização.

ABSTRACT

With the recent and significantly growth of graphics processing unit(GPU), science has been progressing in an absurdly large way, bringing many benefits in areas of study such as Medicine, Engineering, Aeronautics, etc. With the emergence of the programmable pipeline, that is, programming permission at certain stages of the rendering pipeline, that was the birth of a new concept called Shaders, which are programming techniques and methodologies of this pipeline.

However, the propagation of the knowledge of this technology is scarce, since it is a recent and complex method, making the development of new applications in this scenario arduous.

Thus, it became viable to implement a Desktop platform to novitiate and training the development of Shaders, defining techniques and methods using concepts of basic programming. The gamification process comes as a complement to this project, bringing a kind of “motivation” to the user.

Keywords: Game, Educational game, Computer Graphics, Shader, Platform, 2D, GPU, Rendering.

Lista de Figuras

Figura 2.1 – Shaders durante o pipeline de renderização..22
Figura 2.2 – Estrutura de entradas de um shader de vértices..................................23
Figura 2.3 – Ocean Water Shader...24
Figura 2.4 – Mapeamento Gama...25
Figura 2.5 – UV Mapping 26
Figura 2.6 – Diferença entre Bump e Normal Map 27
Figura 2.7 – Processo de Tesselation 28
Figura 4.1 – Demonstração dos inimigos denominados RenderBugs. 39
Figura 4.2 – Saw 39
Figura 4.3 – Turret..40
Figura 4.4 – Mac512 40
Figura 4.5 – infoSpheres 41
Figura 4.6 – Boss 41
Figura 4.7 – Questionário...42
Figura 4.8 – Shader adquirido 43
Figura 4.9 – Acessibilidade a novos locais..44
Figura 4.10 – Mapa do protótipo 45
Figura 4.11 – Arte do protótipo...46
Figura 4.12 – Script de mensagem das infoSpheres..47
Figura 5.1 - Feedback 49

LISTA DE GRÁFICOS

Gráfico 5.1 – Formulário 1 50
Gráfico 5.2 – Formulário 2 51
Gráfico 5.3 – Formulário 3 5252
Gráfico 5.4 – Formulário 4 522
Gráfico 5.5 – Formulário 5 533
Gráfico 5.6 – Formulário 6 533

LISTA DE ABREVIATURAS E SIGLAS

3D – Terceira dimensão

2D – Segunda dimensão

Cg – *C for Graphics*

CPU – *Central Processing Unit*

FPS – *Frames per Second*

GLSL – *OpenGL Shading Language*

GPU – *Graphics Processing Unit*

HLSL – *High Level Shading Language*

OPENGL – *Open Graphics Library*

RV – *Virtual Reality*

TES – *Tesselation Evaluation Shader*

TCS – *Tesselation Control Shader*

NPC – *Non Player Character*

SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO **14**

1.1 Contexto 14

1.2 Motivação e Objetivos 15

1.3 Metodologia de Desenvolvimento do Trabalho 16

1.4 Organização do Trabalho 16

CAPÍTULO 2 - SHADERS **18**

2.1 Processo de Renderização 18

2.1.1 Primitivas 18

2.1.2 Transformações Lineares 19

2.1.3 Eliminação de polígonos ou faces escondidas (culling back-faces) 19

2.1.4 Clipping 19

2.1.5 Rasterização 19

2.1.6 Hidden surface removal 20

2.1.7 Iluminação 21

2.2 Shaders 21

2.2.1 Shader de Vértices 23

2.2.2 Shader de fragmento 24

2.2.3 Tesselation Shader 27

2.2.4 Geometry Shader 28

2.2.5 Compute Shader 29

2.3 Linguagens 29

2.3.1 GLSL (OpenGL Shading Language) 29

2.3.2 HLSL (High Level Shading Language) 30

2.3.3 Cg (C for Graphics) 30

2.4 Considerações Finais 32

CAPÍTULO 3 - JOGOS EDUCACIONAIS **33**

3.1 Tipos de jogos Educacionais 34

3.2 Trabalhos Correlatos 34

3.3 Testes 36

3.4 Considerações Finais 37

CAPÍTULO 4 - DESENVOLVIMENTO DO PROTÓTIPO 38

4.1 Mecânica 38
4.2 Level Design... 45
4.3 Arte 45
4.4 Conteúdo 46
4.5 Considerações Finais 48

CAPÍTULO 5 - AVALIAÇÃO 49

CAPÍTULO 6 - CONCLUSÃO 55

6.1 Contribuições e Limitações 55
6.2 Lições aprendidas 56
6.3 Trabalhos Futuros 56

Capítulo 1

INTRODUÇÃO

1.1 Contexto

Com o crescimento recente e significativo na tecnologia de unidade de processamento gráfico (GPU), a ciência vem progredindo de uma forma grande, abrindo cada vez mais leques na área da Computação Gráfica, que nada mais é do que o processamento e tratamento de imagens, sejam elas na segunda, terceira ou mais dimensões. Algumas grandes áreas em que ela atua é na área da Medicina, por exemplo, na ajuda de análises de exames, e também em muitas das áreas de pesquisas exatas como física ou astronomia para uma representação digital de métodos matemáticos, etc.

Até meados de 2003, era utilizado o *pipeline* fixo de renderização das GPU's, ou seja, a capacidade de renderização estava inteiramente ligada a capacidade do hardware, pois não havia um controle do dado que era processado, apenas o ciclo fixo do *pipeline*, executando as instruções da própria GPU, portanto, o custo era caro quando a necessidade por uma imagem mais realística era extremamente alta.[1]

Assim iniciou-se uma corrida na área científica e tecnológica para desenvolver novos métodos para tornar a imagem mais realista, mais suave e mais flexível. Desse objetivo surgiu o *pipeline* programável, ou seja, a permissão de programação

[1] Shaders. Disponível em: http://desenvolvimentodejogos.wikidot.com/shaders. Acesso em 22 de Abril de 2017.

em certos estágios do *pipeline* na renderização, um novo conceito chamado *Shaders*, que são mini-programas que rodam diretamente na GPU possibilitando cálculos repetitivos e complexos.

Apesar disso, a propagação do conhecimento desta tecnologia é escasso, visto que é um método tanto recente quanto complexo, tornando árduo o desenvolvimento de novas aplicações neste cenário.

1.2 Motivação e Objetivos

Sendo assim, tornou-se relevante a implementação de um jogo de noviciado e treinamento ao desenvolvimento de *Shaders*, definindo técnicas e métodos utilizando conceitos de programação básica.

Dessa forma esse trabalho propõem um jogo que extraiu conhecimentos de diversos livros, como *Teaching a Shader-Based Introduction to Computer Graphics*, (SHREINER; ANGEL, 2011) e *Unity 5.x Shaders and Effects Cookbook: Master the art of Shader programming to bring life to your Unity projects* (ZUCCONI; LAMMERS, 2016) para introduzir ao aluno de uma forma mais didática e divertida. Assim, o jogo apresenta um conteúdo técnico e introdutório no que diz respeito aos *Shaders*, para fornecer o melhor material para estudo e desenvolvimento ao usuário, para que no fim ele possa não apenas conhecer os conceitos que foram transmitidos, mas também aprender a desenvolver suas próprias técnicas e passar este conhecimento adiante.

No fim, será feito um levantamento de dados através de um formulário junto ao protótipo que será enviado aos alunos de Ciência da Computação da instituição, que concluíram a disciplina de Computação Gráfica, necessária para o completo entendimento do conteúdo abordado.

Portanto, propõe-se como objetivo projetar e implementar um jogo educacional de auxílio e introdução ao desenvolvimento de *Shaders*, para que no fim dele os usuários possam ter uma noção mais estruturada sobre esse ponto específico da computação gráfica.

Para a realização do objetivo, serão necessários alguns objetivos específicos:

- Análise e Estudo de Linguagens de *Shading* de alto nível.
- Estudo sobre desenvolvimento de jogos.
- Análise de resultados apresentados.

1.3 Metodologia de Desenvolvimento do Trabalho

O plano de projeto está dividido em seis etapas descritas a seguir:

1. Pesquisa das tecnologias e ferramentas que serão utilizadas no Projeto;
    - Computação Gráfica;
    - *Shaders*;
    - *Software* Unity3D;
    - Linguagens de *Shading*;
    - Domínio de alguma linguagem de alto nível para programação dos eventos;
2. Pesquisa de trabalhos correlatos;
3. Planejamento do Jogo para Desenvolvimento de *Shaders* (*Game Design)*;
4. Implementação do Jogo para Desenvolvimento de *Shaders*;
5. Realização de Testes;
6. Levantamento de Dados;

1.4 Organização do Trabalho

Esta monografia está dividida em 6 capítulos, discriminados a seguir:

Capítulo 1: Introdução – Neste capítulo foi introduzido o contexto que se encontra o projeto, bem como sua problemática e os objetivos para poderem solucionar este caso.

Capítulo 2: *Shaders* – A seguir será apresentado o foco principal do projeto, os *Shaders*. Realizando um leve nivelamento sobre o *pipeline* de renderização gráfico, para assim aprofundar nas categorias de shaders e onde eles atuam, e depois sobre as linguagens de programação de shaders mais conhecidas e utilizadas.

Capítulo 3: Jogos Educacionais – Nesse capítulo será explicado o impacto que os jogos educacionais têm no nosso dia-a-dia atual, analisando os tipos de jogos educacionais e os trabalhos correlatos a este projeto.

Capítulo 4: Desenvolvimento do Protótipo – Será descrito o desenvolvimento do jogo, explicitando todos os pontos e desafios que o jogo apresenta quanto as mecânicas, todos os detalhes durante o *Level Design*, a ideia de como a arte no jogo será aplicada e por fim o conteúdo que será apresentado ao usuário.

Capítulo 5: Avaliação – O relato dos testes que foram realizados com os usuários e as análises a partir deles.

Capítulo 6: Conclusão – Por último, temos o capítulo referente a organização da Conclusão, onde será explicado todas as limitações encontradas, lições aprendidas e o futuro do projeto.

Capítulo 2

SHADERS

Neste capítulo será abordado o assunto principal do projeto, os *shaders*, porém, para adentrar nesse conteúdo é necessário fazer um leve nivelamento sobre o processo de renderização.

Portanto, primeiramente será apresentado os processos de renderização do *pipeline* gráfico, logo após os vários tipos de *shaders* e como eles funcionam, para então explicar sobre as linguagens de *shading*.

2.1 Processo de Renderização

O *pipeline* de renderização, na computação gráfica, é o método de conversão de dados em uma imagem mais próxima possível do realismo humano. Ele pode ser resumido como o cálculo da geometria das cenas, assim como as primitivas dos objetos, suas texturas e cores, como elas se comportam com o ambiente e como a iluminação está configurada, assim como a perspectiva da câmera na cena.

Este processo de realismo virtual envolve sete etapas distintas, não necessariamente chaves, pois, nem todas são utilizadas em todas as aplicações.

2.1.1 Primitivas

Nesta fase, é construído o modelo que conterá todas as informações necessárias para a modelagem do objeto e continuação do processo de realismo visual. Utilizando técnicas de modelagens e transformações lineares.

2.1.2 Transformações Lineares

Nesta fase é aplicada algumas transformações lineares (escalonamento, rotações e translações) ao modelo para que seja possível a visualização deste objeto na perspectiva tridimensional, em dispositivos bidimensionais. Resumindo, esta fase é responsável pela projeção e perspectiva do modelo.

2.1.3 Eliminação de polígonos ou faces escondidas (*culling back-faces*)

Nesta fase, são desconsideradas propriedades da cena que não serão mostradas, para que não haja gastos de processamento que não serão utilizados.

No procedimento de remoção dos polígonos ou faces escondidas, mais conhecido como *culling back-faces,* é analisado a posição relativa entre os objetos que compõem a cena com o observador, de forma que os polígonos que não estiverem no campo de visão do observador sejam removidos.

2.1.4 Clipping

Mais conhecido como clipping ou *viewing frustum culling*, esta fase é responsável por desconsiderar partes da cena que não serão mostradas, como, por exemplo, algum elemento que está muito longe do observador, ou muito perto, tornando o processo mais realístico, pois a CPU não possui noção do que é visível ao nosso raio de visibilidade.

Uma boa estratégia de clipping é importante no desenvolvimento de jogos, pois, maximiza a quantidade de FPS, ou *frames* por segundo, (com a remoção de objetos que não serão necessários, dando mais espaço para a memória tratar o modelo realístico) tanto quanto sua qualidade gráfica.

2.1.5 Rasterização

Para que a imagem possa ser transmitida e representada na segunda dimensão, ou seja, em monitores e telas, é necessário o método de rasterização. Ele possibilita a conversão de um projeto tridimensional qualquer em uma representação matricial onde é possível ser armazenada na memória de um dispositivo *raster*,

utilizado em grande parte dos dispositivos de entrada e saída, como filmadoras digitais, *scanners* e impressoras.

A rasterização dos polígonos funciona da seguinte forma: os polígonos representados na memória pela lista de seus vértices são projetados na tela do dispositivo pela projeção de cada um desses vértices. Assim, os polígonos são rasterizados, primeiramente, por todos os seus lados. Isto é realizado através do cálculo da intersecção de cada uma das linhas horizontais do vídeo, chamadas *scan-lines*.

2.1.6 Hidden surface removal

O principal motivo dos métodos de remoção de superfícies ocultas é o melhor entendimento da cena, ou seja, um realismo mais profundo ao usuário. Desta forma, muitos algoritmos foram criados com base em: quantidade de memória, tempo de processamento e utilidade. Desses algoritmos, o que mais obteve destaque e é utilizado até hoje é o método chamado *z-buffer*.

Umas das limitações que o método *z-buffer* apresenta refere-se a memória e processamento, pois, requer a alocação de até dois *buffers*, em memória, com dimensões iguais à câmera de apresentação, normalmente *buffers* chamados de Imagem e Profundidade. Sendo o primeiro responsável pelo “rascunho” utilizado durante os cálculos de visibilidade, portanto, sendo opcional, e o segundo responsável por armazenar a distância de cada *pixel*, da tela de rascunho, ao plano de projeção, sendo também chamado de *z-Buffer*.

Os *pixels* que são projetados na tela de rascunho são calculados pela distância em relação ao plano de projeção, para então formarem os polígonos que serão projetados na cena. Caso a distância for inferior à distância armazenada no *buffer* de Profundidade, os valores desse *pixel* substituem os valores que foram armazenados anteriormente na tela de rascunho, então a nova profundidade é armazenada no *z-buffer* para aquela posição.

2.1.7 Iluminação

A última etapa do processo de renderização se trata da iluminação, ou seja, como elas se comportam com a cena em questão e também como cada elemento da cena se comporta com elas, trazendo assim um alto nível de realismo para o projeto.

Como tratamos a iluminação recebida e emitida de cada elemento da cena, é dito que os emissores (lâmpada, fogo, estrelas) são fontes de luz, e são distinguidos por suas intensidades e frequências. Já os objetos que são classificados como refletores de luz, geralmente são os objetos que terão a aparência realística, pois, como propriedades são caracterizados pela aparência da superfície como cor, polimento e material. A seleção desses dois tipos de objetos vai depender do contexto do projeto, como, por exemplo, um ambiente futurista utilizaria diversas emissões artificiais de luz, enquanto um ambiente se aproximando do realismo utilizaria bastante da reflexão e refração dos objetos da cena.

2.2 Shaders

O grande problema que as GPU's de antigamente enfrentavam era a questão do *pipeline* fixo, ou seja, não era possível executar outras funções além das que eram programadas neste *pipeline*. Por isso a computação gráfica não conseguia obter resultados tão realistas, e assim surgiram os *shaders* para solucionar este problema.

Sendo executados diretamente na GPU, tornando o processo mais rápido, pois, possui uma capacidade de processamento superior a da CPU e executando tarefas em paralelo, com métodos que não necessitavam uma quantidade alta de polígonos, como funcionava com o pipeline fixo.

Basicamente, *shaders* são miniprogramas que rodam diretamente na placa gráfica ou GPU e que manipulam a cena 3D durante o processo de renderização, antes da imagem ser aplicada a tela. *Shaders* são capazes de criar diferentes tipos de efeitos e realismos, e são utilizados principalmente para aplicações em tempo real, como games.

Há diferentes categorias de shaders, que são executados em partes distintas do *pipeline* de renderização, como demonstra a Figura 2.1, porém, os mais importantes e que serão mais destacados neste projeto são os *shaders* de vértices (*Vertex Shader*) e os *shaders* de fragmento (*Pixel/Fragment Shader*).

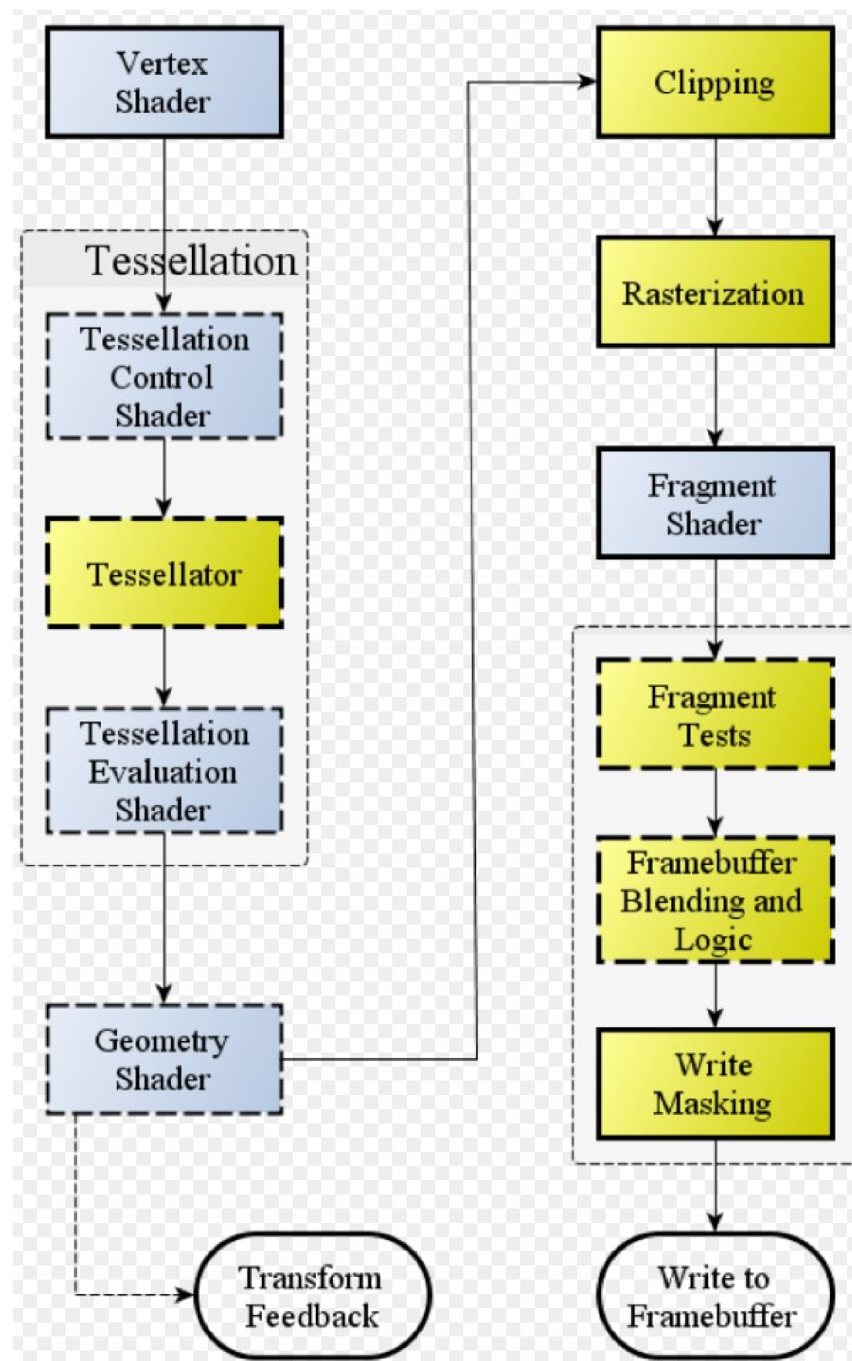

Figura 2.1 – *Shaders* durante o pipeline de renderização

Extraído de http://opengl-notes.readthedocs.io/en/latest/topics/intro/opengl-pipeline.html#vertex-shader, acessado em 19 de maio 2017.

2.2.1 *Shader* de Vértices

O primeiro *shader* a ser executado durante o pipeline, acontece antes do processo de *culling back-faces*. O *shader* de vértices é responsável por manipular os vértices dos objetos da cena, tratando-às uma a uma, porém, não são capazes de criarem novas vértices.

O *shader* de vértices trabalha com vértices individuais, um de cada vez. Ele não possui conhecimento de outros vértices que criam o modelo gráfico, assim como de qual primitiva os vértices pertencem. Para cada vértice de entrada, o *shader* retorna um vértice de saída.

Cada vértice tem um conjunto de atribuições definidas pelo usuário, por exemplo, posição, tangente, vetor normal e coordenadas de textura, como demonstrado na figura 2.2.

```
struct vertexInput {
    float4 vertex : POSITION;
    float4 tangent : TANGENT;
    float3 normal : NORMAL;
    float4 texcoord : TEXCOORD0;
    float4 texcoord1 : TEXCOORD1;
    float4 texcoord2 : TEXCOORD2;
    float4 texcoord3 : TEXCOORD3;
    float4 color : COLOR;
};
```

Figura 2.2 – Estrutura de entradas de um shader de vértices

O melhor exemplo que demonstra o quão poderoso os *shaders* de vértices podem ser e como funcionam são os efeitos de movimentação da água que temos em diversos jogos, demonstrado na figura 2.3, no qual temos apenas um plano completamente reto, porém, com o uso do *shader* de vértices é possível alterar os vértices criando um padrão de movimentação entre elas, tornando o produto final extremamente realístico.

Figura 2.3 – *Ocean Water Shader*

Extraído de https://www.youtube.com/watch?v=ok1mMxVLuUU, acessado em 21 de maio de 2017.

2.2.2 *Shader* de fragmento

O *shader* de *pixel*, ou fragmento, é o responsável por tratar das cores do modelo, e acontece após o processo de rasterização, onde os dados já estão convertidos para pixels, para que possam ser trabalhados um a um, portanto, o termo “fragmento” é usado para definir um elemento que eventualmente contribuirá para a cor final deste *pixel*.

Para que possa definir a cor do *pixel* em questão, o *shader* de fragmento deve lidar com situações como transparência, sombras, névoas e iluminação. Este *shader* é capaz de muitos efeitos, como, por exemplo, o da alteração do contraste de uma cena (é realizado utilizando apenas duas linhas de código), a figura 2.4 demonstra este tipo de efeito, no qual é chamado de mapeamento gama, as faixas representam o nível de contraste utilizado.

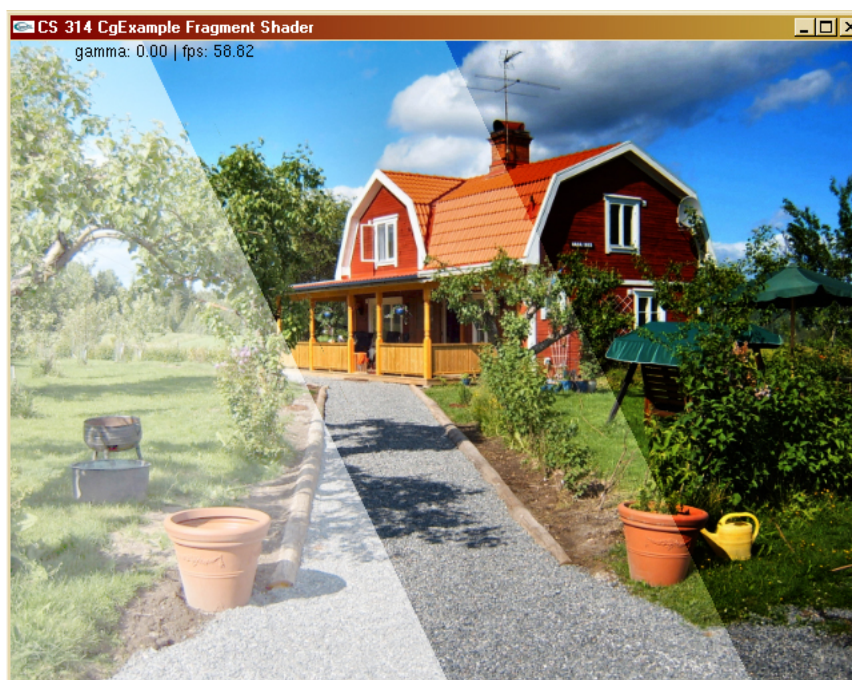

Figura 2.4 – Mapeamento Gama

Extraído de http://web.media.mit.edu/~gordonw/OpenGL/ , Wetzstein Gordon, acessado em 20 de Maio de 2017.

Um dos mais populares usos para o *shader* de *pixels* é utilizando mapeamento de texturas, ou *UV maps* (mapeamentos UV), um conceito de técnicas para que se projete uma imagem bidimensional para uma superfície tridimensional utilizando texturas, onde o U representa uma coordenada e o V outra (não poderia ser chamado de X e Y, pois, geraria confusão), de forma que ela se encaixe perfeitamente ao modelo 3D.

Um bom exemplo disso é considerar uma caixa de presente (um cubo com textura de caixa de presente), para que a textura se assemelhe perfeitamente a este modelo, será necessária uma textura com seis formas geométricas, demonstrado na figura 2.5, para que cada uma combine com cada face da caixa, afetando assim a aparência de como a geometria de malhas quando reagem à luz, tornando os modelos muito mais realistas e detalhados, sem a utilização de altas quantidades de polígonos.

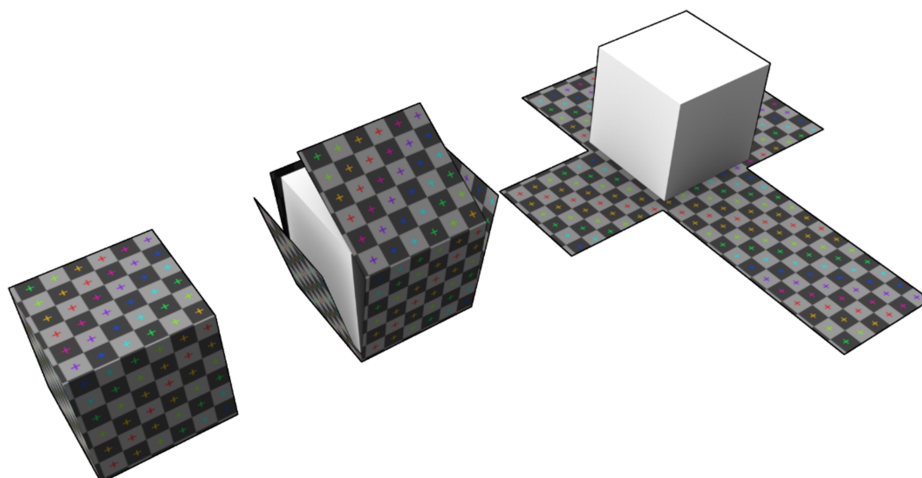

Figura 2.5 – *UV Mapping*

Extraído de http://www.wikiwand.com/en/UV_mapping, acessado em 25 de maio de 2017.

Resumindo, mapeamento UV nada mais é do que alterar seu modelo 3D, distribuindo partes dele em uma superfície com coordenadas de nome U e V, referente as coordenadas X e Y, para facilitar a aplicação ou pintura de imagens 2D.

Há diversas técnicas complementares de mapeamentos e seus diversos subtipos, porém, as duas mais utilizadas e que irei apresentar neste projeto são os mapeamentos por *bump map* e *normal map*.

A função do *bump* e *normal map* é a simulação de um relevo em uma superfície, calculando o ângulo da luz recebida para causar o efeito de sombra em uma textura qualquer e, consequentemente, trazendo uma impressão de maior profundidade de detalhes para o objeto.

O bump map é um método para simulação de relevos em uma superfície sem a necessidade de criação de novos polígonos, gerando sombra de acordo com o ângulo da câmera, e gerado através de um mapa em tons de cinza. No qual preto representa a ausência de relevo e o branco relevo total. O problema deste algoritmo é que quanto mais perto a câmera se encontra do modelo, mais perceptível é a falsidade do relevo, pois, ele apenas simula altura.

O *normal map* é uma aplicação melhorada comparada ao bump map, porém, um pouco mais complexa. Os mapas são em RGB, e além de guardarem a altura do relevo, também guardam a direção em relação a câmera, tornando muito mais eficaz e menos perceptível ao usuário, resultando num realismo e qualidade melhor. Um exemplo de comparação entre o detalhismo de ambos os métodos podem ser vistos na figura 2.6.

Figura 2.6 – Diferença entre *Bump* e *Normal* Map

Extraído de http://thezedlab.com/quick-reference-baking-normals-from-a-bump-map/ , acessado em 20 de Maio de 2017.

2.2.3 Tesselation Shader

O *shader* de *tesselation* foi inserido nas versões de OpenGL 4.0 e DirectX 11 em diante, e acontece durante o processo de vértices (estágio do *pipeline* de renderização onde as sequências de vértices são processadas através de uma série de *shaders*), onde a primitiva é subdividida em várias partes, para providenciar assim um nível maior de detalhes. Este processo ocorre em dois estágios de *shaders* e um estágio fixo do *pipeline*. O processo de subdivisão da primitiva envolve a computação de novos valores de vértices (posição, cor, coordenadas de textura, etc) para cada uma que foi gerada durante o processo. Cada estágio do *pipeline* de *tesselation* executa parte desta tarefa.

O *Tesselation Control Shader* (TCS) é responsável por controlar a intensidade da *tessellation*, o quanto deverá ser aplicado ao modelo. Além disso, ele também é responsável por gerar os dados necessários para garantir a continuidade do tesselamento, passando assim para o *Tesselator*. Lembrando que esta etapa é opcional, caso não seja executada são aplicados os valores padrões de *tesselation*.

O *Tesselator* é a parte fixa do *pipeline* que executa a ação em si. Subdividindo os modelos de acordo com o nível de intensidade declarado na etapa anterior, segue-se um exemplo na figura 2.7.

Por fim temos o *Tesselation Evaluation Shader* (TES), recebendo o modelo tesselado e registrando cada valor dos vértices para cada vértice gerado durante o processo, muito parecido com o *shader* de vértices.

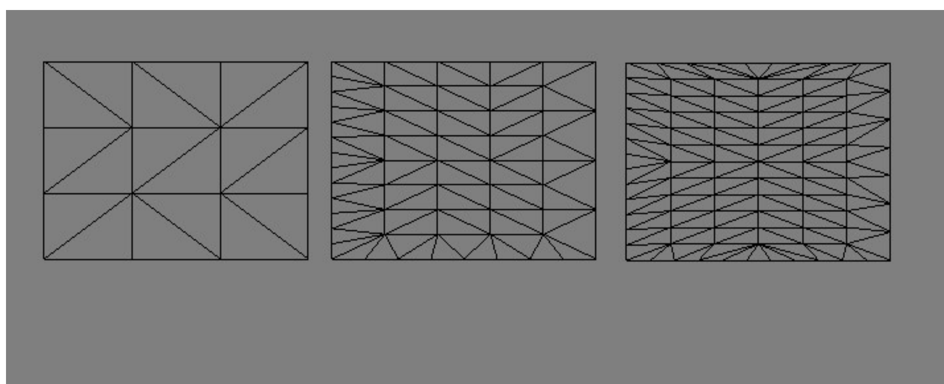

Figura 2.7 – Processo de *Tesselation*

Extraído de http://in2gpu.com/2014/07/12/tessellation-tutorial-opengl-4-3/ , Vlad Baiou, acessado em 21 de Maio de 2017.

2.2.4 *Geometry Shader*

Assim como o nome sugere, o *shader* de geometria é utilizado para manipular a geometria do objeto, ele recebe a primitiva como entrada, assim como o *shader* de vértices recebe os vértices do modelo como entrada, ele pode retornar zero ou mais primitivas como saída. Eles são codificados para aceitarem uma única primitiva como de entrada e zero ou mais primitivas de saída. Durante o processo de renderização, o *shader* de geometria acontece entre o *shader* de vértice e fragmento.

Os principais motivos para a utilização dos shaders de geometria são:

- Camadas de Renderização: Recebendo uma única primitiva e a dividindo em múltiplas camadas sem a necessidade de alterar os parâmetros de renderização e assim por diante.
- *Feedback Transform*: Geralmente utilizado quando se faz tarefas computacionais na GPU (*Compute-Shader*), será melhor explicado no capítulo 2.2.5.

2.2.5 Compute Shader

O *Compute shader* é um *shader* de propósito geral, que é utilizado fora do *pipeline* de renderização, o que significa que ele não é utilizado para desenhar uma primitiva, ou tonalizar um *pixel*. Eles são utilizados para usufruir da capacidade de processamento paralelo da GPU, realizando múltiplas tarefas, muitas delas em conjunto com *shaders* de vértice e fragmento para um resultado mais fino. É uma tecnologia extremamente recente e poderosa.

2.3 Linguagens

Os *shaders* possuem sua linguagem própria de codificação, pois, eles atuam diretamente nas placas gráficas e GPU's, ao contrário de linguagens de programação comuns, que são executadas pela CPU. Todas as linguagens de *shaders* possuem similaridades com C, para uma melhor compreensão. A seguir irei apresentar as linguagens mais utilizadas, com suas respectivas características.

2.3.1 GLSL (OpenGL *Shading Language*)

A GLSL é uma linguagem de programação com muitas semelhanças com C, porém, voltada ao desenvolvimento de shaders. Oferecida pela OpenGL, e padronizada pela Khronos, é uma linguagem muito utilizada atualmente, por sua aplicabilidade e confiabilidade que a OpenGL traz.

Sobre os tipos de dados da linguagem GLSL, é possível defini-los como:

- Reais, escalares e vetoriais: float, vec2, vec3, vec4;
- Inteiros, escalares e vetoriais: int, ivec2, ivec3, ivec4;
- Matrizes quadradas: mat2, mat3, mat4;
- Matrizes não-quadradas: mat3x2, mat 2x4, etc...
- Booleano escalar ou vetorial: bool, bvec2, bvec3, bvec4;
- Sampler para acesso às texturas.

Há alguns qualificadores, que ajudam a definir o que a variável irá armazenar para a GPU, poupando muito processamento, alguns deles:

- Uniform: Armazena algum valor “por primitiva”;
- Attribute: Armazena algum valor por vértice;
- Varying: Quando a variável é passada de um estágio do shader para o outro.

Aplicações OpenGL possuem um processamento um pouco mais lento em comparação com aplicações DirectX (HLSL) em plataformas Windows, logo, isto é realmente relativo com o projeto que se está desenvolvendo. De qualquer forma, OpenGL é a única interface aberta a todas (Windows/Mac/Unix/Alguns Consoles) plataformas. Sabendo disso, utilizar exclusivamente GLSL pode ser a melhor escolha para produtos não centrados apenas em plataformas Microsoft.

2.3.2 HLSL (*High Level Shading Language*)

High Level Shading Language é a linguagem de *shading* criada pela Microsoft, utilizada em conjunto com a API gráfica DirectX. Muito usada nos dias de hoje em jogos devido à relevância que os shaders ganharam ao longo do tempo.

Os tipos de dados da HLSL são muito semelhantes aos de C/C++, sendo que vetores de até 4 componentes são estendidos para serem tipos primitivos da linguagem. Um tipo de dado que é ausente nesta linguagem é aquele relacionado a strings, pois, em aplicações gráficas é desnecessário a utilização deste tipo.

Ao contrário de GLSL, que necessita de qualificadores, os atributos são argumentos nas funções dos shaders.

Em resumo, HLSL é tão poderosa quanto GLSL, porém, é mais aplicada a jogos ou projetos apenas desenvolvidos para produtos da Microsoft.

2.3.3 Cg (C *for Graphics*)

Por fim, temos Cg, criada pela Nvidia em parceria com a Microsoft, é muito parecida com HLSL. A maioria dos códigos em Cg irá compilar em compiladores HLSL com apenas mudanças mínimas, e vice-versa.

Porém, Cg também suporta aplicações em OpenGL, pois, vem de uma “plataforma independente”, o que a torna uma linguagem mais relevante para este projeto.

Cg possui seis tipos básicos de dados, alguns deles são iguais ao C, outros são especificamente adicionados para a programação em GPU, esses tipos são:

- Float: Um número de ponto flutuante de 32-bit;
- Half: Um número de ponto flutuante de 16-bit;
- Int: Um número inteiro de 32-bit
- Fixed: Um número fixo de 12-bit;
- Bool: Uma variável booleana;
- Sampler: Representa um objeto de textura.

Cg também possui tipos de dados em vetores e matrizes, baseados nos tipos básicos de dados, como float3, float4x4, etc. Esses tipos são mais utilizados quando se trabalha com programação gráfica 3D. Cg também possui tipos de dados em *array* e estruturas, assim como o C. Ela também suporta algumas outras coisas como interfaces, *arrays* de tamanho indefinido, etc, o que torna a codificação de *shaders* mais dinâmica e fácil.

A Unity3D, a plataforma de desenvolvimento de jogos que será utilizada no projeto, oferece uma plataforma de desenvolvimento de *shaders* chamada ShaderLab, com uma sintaxe extremamente similar com Cg, apenas com algumas adaptações específicas para a Unity3D. Por essa peculiaridade, e outras, como a facilidade de aprendizado que ela oferece para iniciantes no desenvolvimento de games devido a interface interativa, etc.

Conclui-se que Cg é a linguagem mais apropriada para este projeto, apesar de não ser mais atualizada pela Nvidia, porém, no quesito didático, é a que mais se aplica ao projeto, além de que o conhecimento que ela traz e sua similaridade com HLSL a tornam um ótimo pilar para quem está ingressando no mundo dos Shaders, além de ser uma linguagem multiplataforma.

2.4 Considerações Finais

Neste capítulo foi instruído que os *shaders* são um ponto específico, porém, de alto valor atualmente na área da Computação Gráfica, pois auxilia no tratamento direto das execuções no *pipeline* da placa gráfica ou GPU, no processo de renderização, ou sintetização de imagens.

O projeto atual irá utilizar como base a linguagem de alto nível para programação de *shaders* Cg, por ser uma linguagem multiplataforma de relevante valor didático. Os tipos de *shaders* que serão apresentados no projeto serão os mais utilizados, que são os *shaders* de vértice e fragmento, para um melhor entendimento ao usuário.

A seguir será relacionado aos trabalhos mais relevantes relacionados a este tema da Educação com o advento da Gamificação.

Capítulo 3

JOGOS EDUCACIONAIS

Aqui será apresentado o impacto que os jogos educacionais têm nos dias atuais, dando exemplos de aplicações que envolvem programação e como elas estão progredindo no ensino de escolas e universidades.

É de conhecimento mútuo que os métodos de ensino atuais estão ultrapassados, e que cada vez mais ouvimos termos como gamificação, imersão tecnológica, etc. A verdade é que apenas um método de ensino acaba se tornando entediante para os estudantes, porém, a tecnologia traz uma infinidade de maneiras para estimular nosso cérebro a aprender, e com o advento da gamificação, que pode ser definido como o uso do *design* de elementos de jogos, em um contexto não-jogável, segundo Deterding, (2011).

Em companhia a isso temos o crescimento da indústria de jogos, que hoje em dia supera a grande indústria do cinema, e se desenvolvendo ainda mais. A este respeito, nos Estados Unidos, 65% das famílias possuem alguém que joga vídeo-games regularmente, e também é notado que a indústria de jogos impacta na economia norte-americana, em 2016, ela contribuiu com US $11,7 bilhões para o PIB. Alimentando o emprego direto de 65,678 americanos[2]. No Brasil, em oito anos, o número de empresas desenvolvedoras de jogos digitais aumentou em quase 600%, e o faturamento neste setor no país cresceu 25% entre 2014 e 2016[3].

---

[2] THE ESA – 2017 Essential facts about the computer and video game industry, Disponível em:< http://www.theesa.com/article/2017-essential-facts-computer-video-game-industry/>. Acesso em: 05 jun. de 2017.

[3] G1 – Número de desenvolvedores de games cresce 600% em 8 anos, diz associação, Disponível em <http://g1.globo.com/economia/negocios/noticia/numero-de-desenvolvedores-de-games-cresce-600-em-8-anos-diz-associacao.ghtml>. Acesso em: 05 jun. de 2017.

Assim sendo, tornou-se viável o desenvolvimento de um jogo para o ensino deste conceito tão pouco explorado, que será um pilar da computação gráfica no futuro.

3.1 Tipos de jogos Educacionais

Existem diversas categorias de jogos que podem ser usados para a educação. Aqui citarei as mais relevantes:

Ação: Estes jogos têm como característica o desenvolvimento da integração das funções motores e psíquicas, melhora nos reflexos e na velocidade para resolução de problemas.

Estratégia: Jogos de cartas, tabuleiro, construção, simulação, entre outros, tem a finalidade de aguçar o senso lógico dos alunos, de modo a algum tema ou área em questão. Delimitando decisões estratégicas que superam a sorte como fator de determinação do vencedor.

Aventura: Se caracterizam por apresentar um ambiente desconhecido ao usuário e a ser explorado por ele, o que torna um ponto perfeito para aplicar um conhecimento pouco conhecido pelo ambiente em que se deseja implementar.

Portanto, foi definido este tipo de jogo a ser desenvolvido, apresentando um mundo com poucas cores, para que ao longo do caminho, diversas cores possam ser apresentadas demonstrando a diferença que os *shaders* fazem em uma aplicação gráfica.

3.2 Trabalhos Correlatos

Durante os últimos anos, a OpenGL se tornou uma API padrão no ensino utilizando a computação gráfica, seja em ciência da computação, medicina, engenharia, arquitetura, etc. E uma das últimas releases dessa empresa foi a API WebGL em JavaScript, que está disponível através do novo elemento *canvas* do

HTML5, oferecendo suporte para renderização de gráficos 2D e 3D, e foi assim que ANGEL(2017), realizou uma pesquisa demonstrando todas as mudanças que a OpenGL aplicou para se habituar cada vez mais com os métodos de aprendizado, e concluiu que a WebGL é a maior promessa do futuro para a OpenGL, ele declara:

> Embora muitos cursos introdutórios de computação gráfica ainda precisam migrar para o WebGL, three.js, ou uma combinação dos dois, os argumentos para criar a mudança são fortes. Em minhas aulas (MOOCs, Siggraph e universidade) e em minhas interações com instrutores usando meu textbook, houve uniformemente reações positivas com a utilização do WebGL e three.js.

E dessa relação em prol do crescimento da computação gráfica que o trabalho proposto também se atrela.

Em 2015, na Flórida, foi desenvolvido um jogo educacional para ensino de fundamentos da programação, devido ao fato da demanda por empregos relacionados a ciência da computação ter crescido rapidamente. O jogo entitula-se CodeCraft e é um jogo no qual utiliza uma aproximação baseada em problemas no qual os jogadores aprendem conceitos de programação resolvendo *puzzles* em um ambiente 3D imersivo(VENTURA et al., 2015). A diferença entre esse trabalho e o trabalho proposto é o conteúdo apresentado no jogo, pois, neste trabalho estamos apresentando uma parte bem específica da computação gráfica, e não fundamentos de conceitos de programação.

Em Santa Catarina, este ano ,2017, algumas escolas estão adotando jogos eletrônicos consagrados como metologia de ensino. O minecraft, por exemplo, é um jogo que cobra do jogador uma criatividade absurda[4], e hoje em dia é considerado um dos jogos mais vendidos da história, o professor de inglês, Leonardo Tubarão acrescenta:

---

[4] DC - ESCOLAS DE SC ADOTAM JOGOS ELETRÔNICOS CONSAGRADOS COMO METODOLOGIA DE ENSINO, Disponível em: <http://dc.clicrbs.com.br/sc/estilo-de-vida/noticia/2017/05/escolas-de-sc-adotam-jogos-eletronicos-consagrados-como-metodologia-de-ensino-9785029.html>.Acesso em: 23 ago. de 2017.

> "O jogo desperta atenção, foco e compreensão. A parte fantástica de usar o game para ensinar é poder sair do abstrato para o concreto, mas dentro do mundo virtual. Posso trabalhar ciências criando todo um sistema respiratório, por exemplo. É possível abordar conceitos de volume na aula de matemática. Além disso, o tempo todo eles estão exercitando o inglês, dentro da proposta do ensino bilíngue."

Além disso, durante o meu estágio na Secretaria Municipal de Educação de Marília-SP, observei que diversas escolas de educação infantil estão adotando plataformas de ensino como o Khan Academy, que auxilia em diversos temas da educação, incluindo programação. Essa plataforma, junto com muitas outras como Udemy, Udacity, Coursera, e diversas outras, são plataformas que fazem parte do complemento de ensino de muitas universidades e escolas nos EUA e outros países, desde 2012[5].

3.3 Testes

O processo de testes envolve a procura por falhas e também a jogabilidade e aceitação do jogo por parte dos usuários. Os testes geralmente são distribuídos entre todas as fases de desenvolvimento (JUNIOR, 2002, p. 12).

Os testes de funcionalidade e *bugfix* (acertar erros), em geral possuem as características das abordagens clássicas da Engenharia de Software. Mas também existe uma classe de testes que é própria para o desenvolvimento de jogos: o *playtest*. O *playtest* permite que os programadores analisem a aceitação do jogo e a reação dos jogadores (JUNIOR, 2002, p. 12). De acordo com Junior (2002), um *playtest* pode ser:

Aberto: uma versão do jogo é disponível para *download* para os usuários testarem ou marca-se um dia e local para a realização dos testes.

[5] IG – COLUMBIA E 16 UNIVERSIDADES ADEREM À PLATAFORMA DE CURSOS ONLINE, Disponível em: http://ultimosegundo.ig.com.br/educacao/2012-09-19/columbia-e-mais-16-universidades-aderem-a-plataforma-de-cursos-online.html. Acesso em: 03 set. de 2017.

Fechado: os testadores são previamente selecionados, existem os chamados “testadores profissionais”, eles são capazes de ir além de simplesmente testar e apontar o problema, eles são capazes também de dar informações mais detalhadas e sugestões que auxiliam na correção do *software*.

As abordagens de teste variam conforme a equipe e a fase de desenvolvimento, sendo que as mais conhecidas e utilizadas são:

Usuários jogam e programador observa: o programador não mantém nenhum contato com o jogador. O programador deverá apenas observar e fazer as anotações das reações e dificuldades dos jogadores. Essa metodologia se mostra eficiente na descoberta de dificuldades de jogabilidade e de partes tediosas de um game (JUNIOR, 2002, p. 12).

Usuários jogam enquanto o programador faz perguntas: o programador não apenas observa, ele também faz perguntas ao longo do jogo para o jogador. As perguntas são sobre os aspectos, a jogabilidade e a emoção do jogo. Funciona bem para jogos onde a ação é constante e dinâmica (JUNIOR, 2002, p. 12).

Usuários jogam e fazem relatório: é muito útil em *playtests* de grande porte (como os testes via internet), essa abordagem exige que o usuário jogue e faça um relatório sobre o que achou do jogo. É nesse modelo que se encaixam bem os “testadores profissionais”. Porém, os desenvolvedores devem se atentar ao problema de vazamento de informações. É comum acontecer de versões piratas de jogos serem disponibilizados na Internet, meses antes de seu lançamento, fruto de cópias de seções de testes (JUNIOR, 2002, p. 12).

3.4 Considerações Finais

Conclui-se que os jogos educacionais estão tomando cada vez mais parte do dia-a-dia de escolas e universidades, trazendo um conteúdo didático e, ao mesmo tempo, divertido ao estudante.

A seguir será apresentado o desenvolvimento do protótipo.

Capítulo 4

DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo será explicado todas as etapas realizadas durante o desenvolvimento do projeto.

4.1 Mecânica

A mecânica do jogo é referente a jogabilidade do protótipo, ou seja, tudo o que pode e não pode ser feito pelo usuário e *NPC's,* ou “Personagem Não-Jogável”(Personagem que não pode ser controlado por um jogador mas se envolve de alguma forma com o enredo do jogo), como eles irão interagir com o ambiente, etc. A mecânica decidida seguirá o subgênero de ação-aventura denominado *MetroidVania*(por ser inspirada na série de jogos *Metroid* e *Castlevania*), no qual o jogo possui um grande mapa interconectado que o jogador poderá explorar, porém, com alguns locais que estarão “bloqueados” até o jogador atingir um certo nível no progresso do jogo, ou obter alguma habilidade após derrotar algum chefe.

Ele seguirá a perspectiva 2D em plataforma, pois, é o mais conhecido e popular sistema de jogos de todos os tempos, para uma maior assimilação do usuário ao projeto. Lembrando que o foco está no conteúdo apresentado, e não na complexidade de sua mecânica, portanto, a simplicidade nesse caso é a opção mais plausível e que preenche da melhor forma o objetivo do projeto.

O jogador poderá pular e andar, com os botões ‘espaço’ e as setas do teclado, eliminando os inimigos denominados *RenderBugs*, ou “*bugs* de renderização”, pulando em cima deles, elemento retirado de muitos jogos clássicos, como *Mario* ou *Sonic*, para que o usuário assimile a mecânica de forma natural. Os inimigos são demonstrados na figura 4.1. Ele possuirá cinco vidas, que estarão no canto superior esquerdo da tela, e os corações refletem quantas vezes ele poderá

levar danos, quando não houver mais corações ele irá perder uma vida, e voltar ao último *checkpoint* registrado. Caso ele perca todas as vidas, ele terá a opção de recomeçar o jogo ou sair dele.

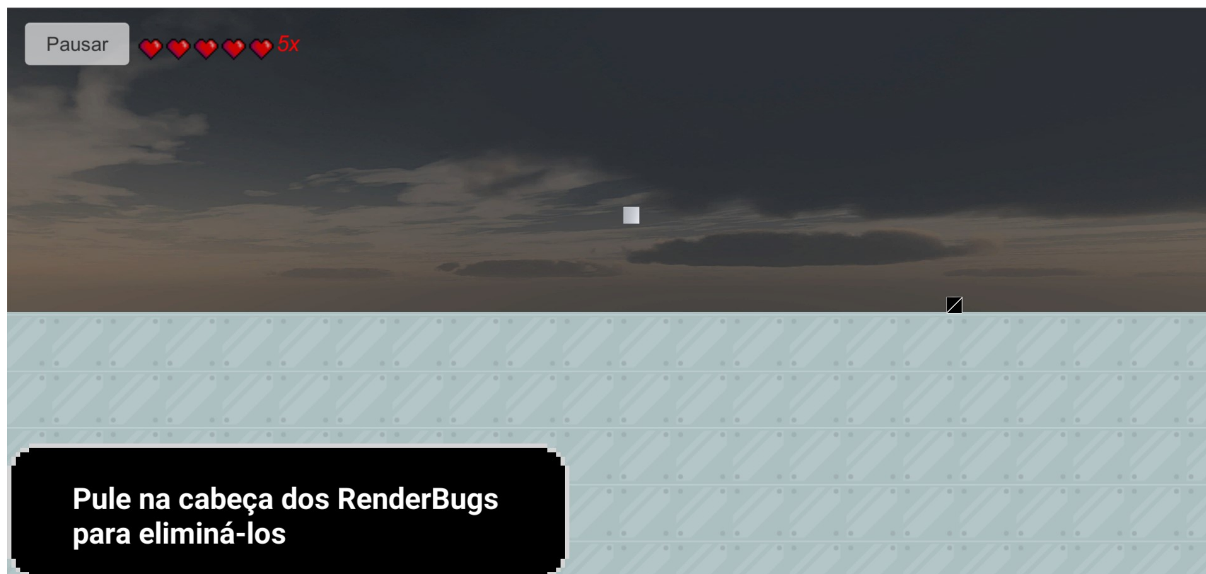

Figura 4.1 – Demonstração dos inimigos denominados *RenderBugs.*

Os *RenderBugs* poderão causar dano ao jogador de duas formas, o tocando ou com os projéteis que serão lançados por eles. Além deles, também existem inimigos que não podem ser destruídos, que devem ser apenas desviados, tornando o jogo mais desafiador e divertido. Um deles é a *Saw*, ou "Serra", um inimigo que estará em alguns pontos do mapa apenas para atrapalhar o jogador, e poderão se mover também, caso o jogador encoste em alguma delas, ele perderá um coração. A *Saw* é demonstrada na figura 4.2.

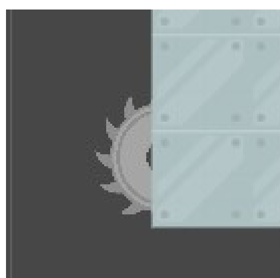

Figura 4.2 – *Saw*

Outro inimigo que irá atrapalhar o jogador é a *Turret,* ou "Torre", que ficará em um local que o jogador não possa alcançar (teto, por exemplo), e irá disparar

projéteis que causarão dano ao jogador, retirando um coração. Ela é demonstrada na figura 4.3.

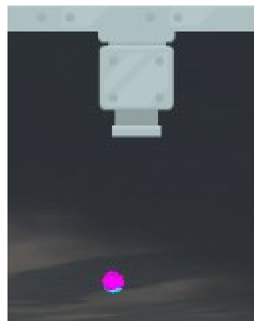

Figura 4.3 – *Turret*

Por fim, teremos o Mac512 (demonstrada na figura 4.4), que irá se movimentar e disparar projéteis no jogador, causando dano a ele. Ela também poderá causar dano ao encostar no jogador. Os projéteis deles terão a mesma forma que eles mesmos, porém, com tamanho reduzido.

Figura 4.4 – *Mac512*

O objetivo principal do jogo situa-se em encontrar as *infoSpheres* (demonstrado na figura 4.5), ou “esferas de informação” que lhe concederão conhecimento sobre os *shaders*, através de uma caixa de mensagem que aparecerá ao personagem tocar na esfera. Para no fim de cada chefe, ele conseguir responder o questionário que surgirá, adquirindo uma nova habilidade e um novo *shader* que será aplicado ao personagem.

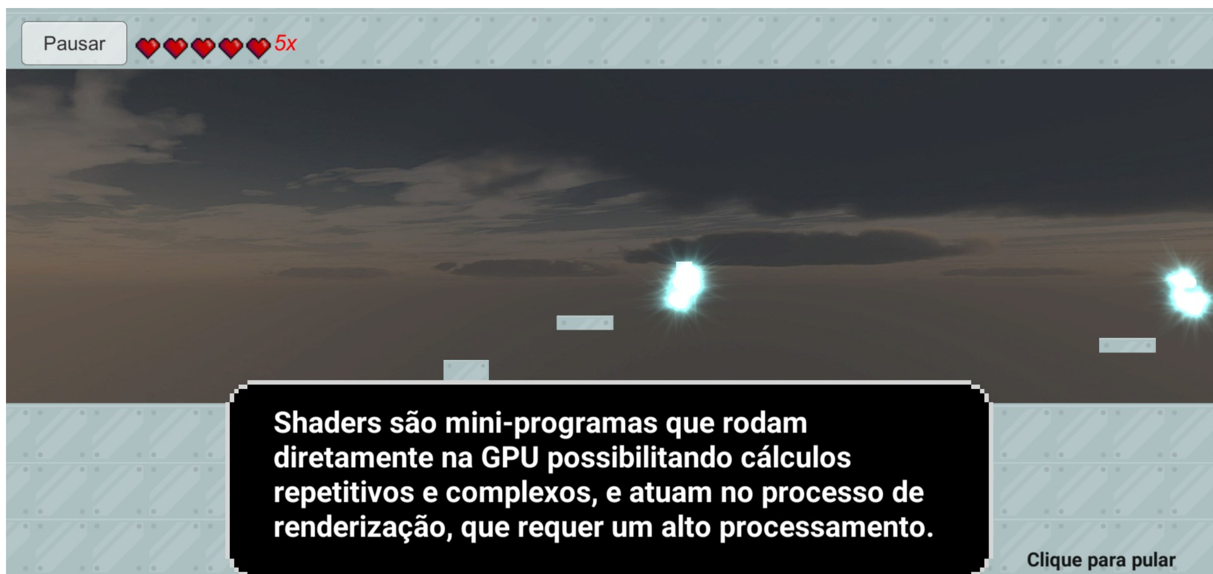

Figura 4.5 – *infoSpheres*

Cada chefe possuirá um comportamento diferente, referente ao ambiente que estiver, e ao *shader* que ele possuirá, o primeiro chefe, por exemplo, ficará em movimento até que alcance uma certa posição para lançar diversos projéteis em direções aleatórias (mostrado na figura 4.6), e poderá ser derrotado da mesma forma que os *RenderBugs*, pulando em cima dele.

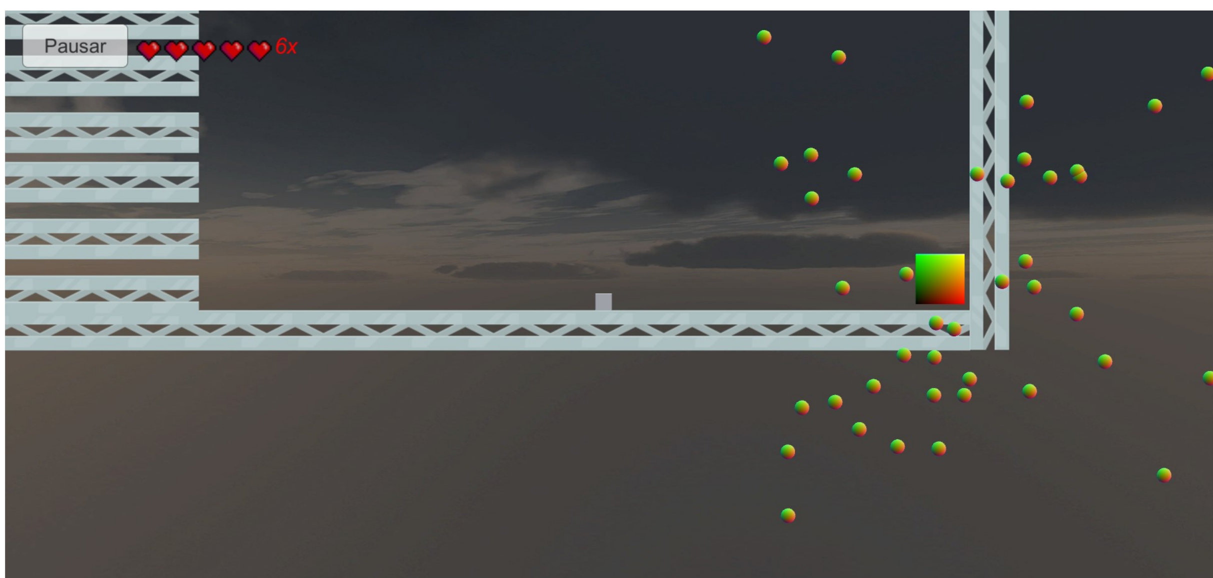

Figura 4.6 – *Boss*

Assim que o chefe for derrotado, o jogador deverá responder corretamente um questionário sobre *shaders* (mostrado na figura 4.7), que estarão contidas nas *infoSpheres* espalhadas pelo mapa. Essas questões envolvem desde conceitos gerais sobre *shaders*, até questões específicas sobre Cg, a *shading language* que foi ensinada,etc. As perguntas foram baseadas nas *infoSpheres* que serão disponibilizadas ao usuário. Todas as questões estão disponibilizadas no Apêndice.

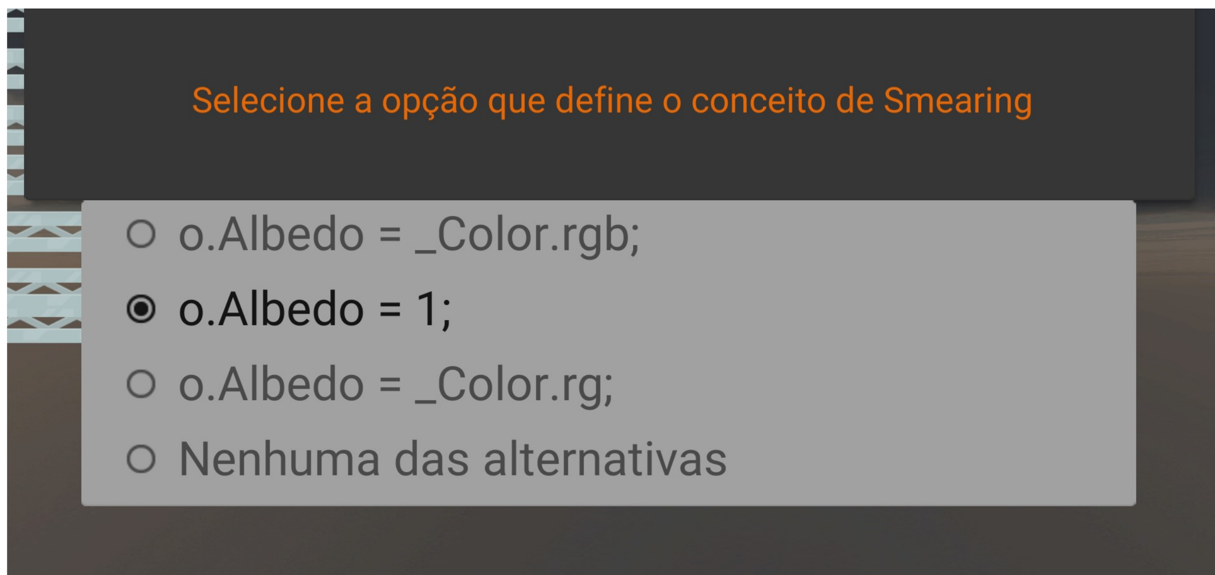

Figura 4.7 – Questionário

Caso o personagem acerte pelo menos 3 das 5 questões aleatórias que serão apresentadas, ele vence e adquire o *shader* que o chefe possuía (para uma melhor visualização deste *shader*, será possível a rotação do personagem no eixo Y, alterando o sentido com as teclas ‘Q’ e ‘E’), demonstrado na figura 4.8. Caso ele perca, ele perde uma vida e volta ao último *checkpoint* registrado.

Figura 4.8 – *Shader* adquirido

Assim que adquirir o novo *shader*, ele poderá acessar áreas do mapa que não poderia, pois, necessitava de uma certa habilidade que o *shader* poderia proporcionar. Neste caso, o *shader* concedeu ao personagem um super pulo, assim como uma plataforma adicional, tornando possível o alcance a locais que estavam muito altos, ou inacessíveis sem o *shader* (demonstrado na figura 4.9).

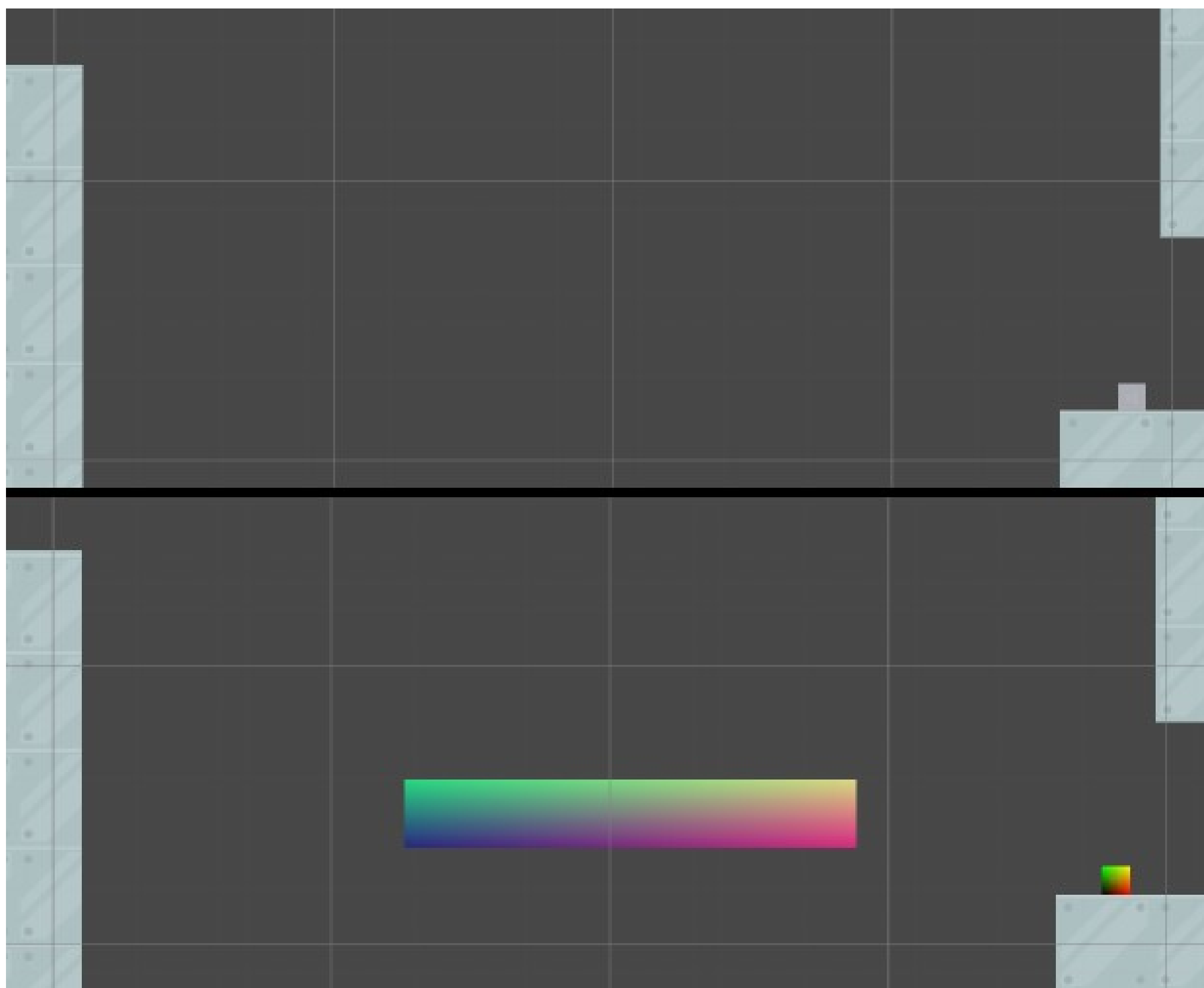

Figura 4.9 – Acessibilidade a novos locais

Não há um sistema de pontuação ou fases, pois, tudo acontece em um único mapa. Portanto, a motivação do jogador se encontra em adquirir mais e mais *infoSpheres* para poder derrotar os chefes e adquirir suas habilidades.

4.2 Level Design

Os mapas nesse tipo de jogo são famosos por serem bem grandes, dando uma vasta quantidade de desafios e um desejo ainda maior por exploração ao jogador. O mapa do projeto está demonstrado na figura abaixo (Figura 4.10). Possui diversas áreas onde o jogador deverá ter cautela e reflexos aguçados para conseguir conquistar todas as *infoSpheres*.

Figura 4.10 – Mapa do protótipo

4.3 Arte

A arte contida no jogo é de um tema futurista monótono, com grande parte dos objetos possuindo cores neutras, a ideia que deseja a ser passada é que estamos num ambiente desprovido de *shaders*, onde os chefes tomaram todos eles para si e deixaram o ambiente com um tom triste. Ao derrotá-los, o mundo começa a ter mais cores e efeitos, dando ênfase no poder que os *shaders* tem no mundo dos jogos e efeitos especiais, além disso, as *infoSpheres* são esferas de luz, indicando

mais uma vez que o conhecimento sobre os *shaders* pode dar vida a um ambiente digital. A figura 4.11 demonstra sobre a arte no protótipo.

Figura 4.11 – Arte do protótipo

4.4 Conteúdo

Atualmente são 23 infoSpheres espalhadas pelo mapa, cada uma trazendo um conhecimento diverso e importante sobre os *shaders*.

A interação delas com o jogador funciona da seguinte forma: Assim que o jogador colide com alguma delas, o tempo é paralisado, e é ativado dois objetos principais, um painel e um texto que possuirá a informação do shader referente a *infoSphere* que foi atingida, o script está demonstrado na figura 4.12. Assim que o tempo é congelado e os dois objetos são ativados, é acionado uma *Coroutine* (funções que conseguem pausar a execução e voltar o controle para a Unity e logo após continuar a execução do *frame* que foi pausado) com um contador para que o jogador possa ler todo o conteúdo (diferentes *infoSpheres* possuem diferentes contadores), porém, caso ele leia rapidamente ou queria pular alguma informação que já saiba, ele só precisa clicar com o botão esquerdo do *mouse*.

```
public class msg : MonoBehaviour {

    public GameObject txt,txtCliq, panel;
    public float time = 3;
    private AudioSource som;

    private void Start()
    {
        som = gameObject.GetComponent<AudioSource>();
    }

    void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Player")
        {
            txtCliq.SetActive(true);
            txt.SetActive(true);
            panel.SetActive(true);
            som.Play();
            StartCoroutine(DisappearBoxAfter(time));

        }
    }

    IEnumerator DisappearBoxAfter(float waitTime)
    {
        Time.timeScale = 0.00000001f;
        float pauseEndTime = Time.realtimeSinceStartup + waitTime;
        while (Time.realtimeSinceStartup < pauseEndTime)
        {
            yield return 0;
            if (Input.GetButton("Fire1"))
            {
                Time.timeScale = 1;
                txt.SetActive(false);
                panel.SetActive(false);
                txtCliq.SetActive(false);
                transform.GetChild(0).gameObject.SetActive(false);
            }
        }
        Time.timeScale = 1;
        txt.SetActive(false);
        panel.SetActive(false);
        transform.GetChild(0).gameObject.SetActive(false);
    }
```

Figura 4.12 – Script de mensagem das *infoSpheres*

No protótipo atual foram cadastradas 9 questões que serão geradas aleatoriamente ao usuário, todas baseadas nas *infoSpheres* contidas no mapa, 23 inimigos (exceto os inimigos fixos como torres de projeteis, serras, e outros, pois, estes não podem ser eliminados) e um chefe.

4.5 Considerações Finais

Concluindo, neste capítulo foi explicado todo o *game design* por trás do protótipo, dividindo em 4 etapas: mecânica, *level design*, arte e conteúdo.

A seguir, será demonstrado os testes realizados para fim de levantamento de análises.

Capítulo 5

AVALIAÇÃO

Como o projeto apresenta um conteúdo bem específico, para apenas aqueles que possuem conhecimento sobre o pipeline de renderização e uma base sobre a computação gráfica em si, foi selecionado o *playtest* Fechado, com 30 alunos que concluíram as aulas de computação gráfica na instituição de ensino, oferecendo o protótipo e um formulário através de um grupo formado pelos alunos em uma rede social, com algumas perguntas importantes sobre aprendizado e o game em si. Como cada pessoa utilizou sua máquina pessoal, não foi possível especificar as máquinas, porém, nenhuma delas relatou problemas na execução do software. Esse formulário procurou sanar questões sobre o quão benéfico foi o jogo para o usuário, comparando a livros ou vídeo-aulas. O formulário também é exibido ao usuário após eliminar o chefe, como demonstrado na figura 5.1.

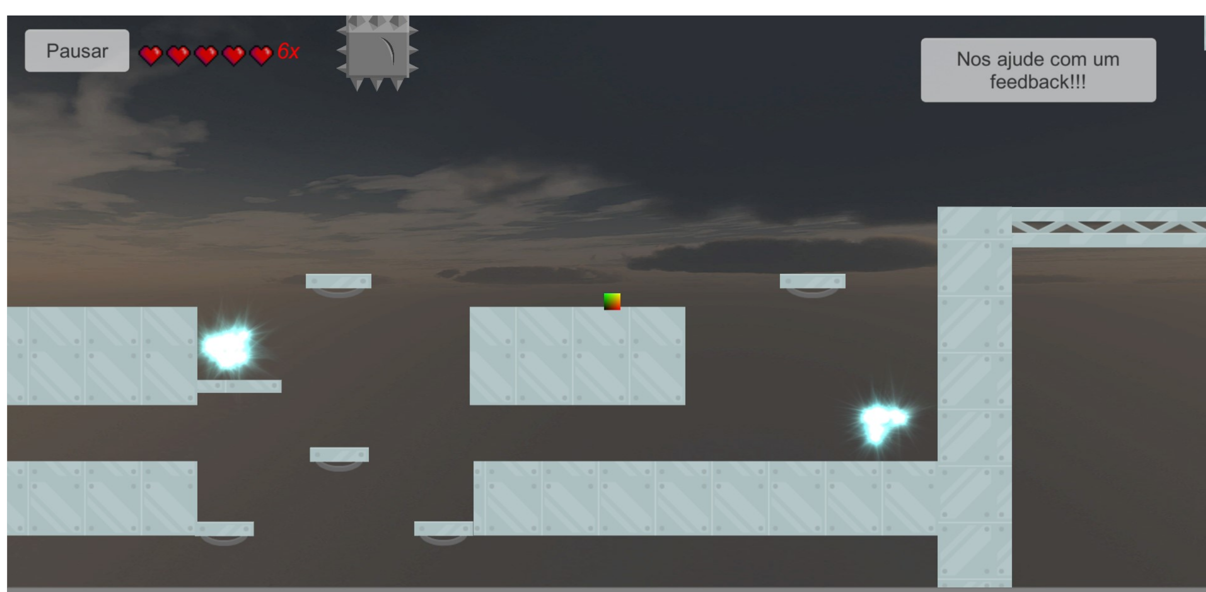

Figura 5.1 - *Feedback*

As primeiras respostas surgiram em torno de 15 a 20 minutos, o tempo médio para se concluir o jogo.

Como este é um protótipo inicial do projeto, a quantidade de informação sobre os *shaders* é pequena, visto que fora necessária uma introdução a este conceito, portanto, em comparação ao livro *Unity 5.x Shaders and Effects Cookbook: Master the art of Shader programming to bring life to your Unity projects*(ZUCCONI; LAMMERS, 2016), que foi utilizado para construção da fundamentação teórica para desenvolvimento do projeto, a finalização deste jogo consegue trazer, em questão de conteúdo, a dois capítulos, o que levaria cerca de 30 minutos, porém, com diferentes exemplos, e uma didática mais divertida, é possível aprender sobre *shaders* em menos de 20 minutos com o protótipo.

1° Pergunta – Costuma estudar com livros e conteúdos impressos?

30 respostas

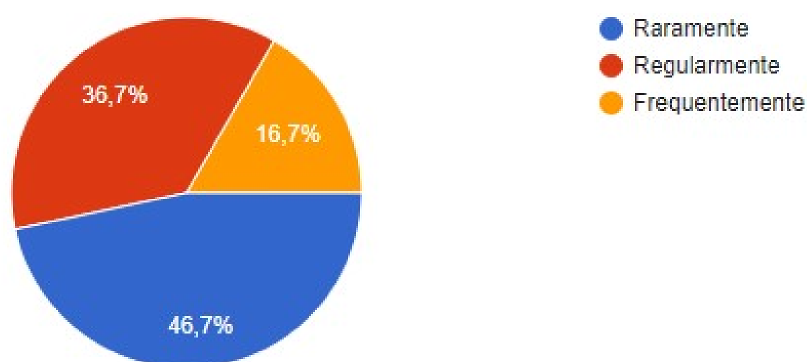

Gráfico 5.1 – Formulário 1

Com base nessas informações, podemos concluir que os meios de estudo antigo, como o papel, estão cada vez mais ultrapassados e menos

utilizados, apesar de serem extremamente detalhados e com um conteúdo mais aprofundado.

2° Pergunta – Costuma estudar com vídeo-aulas e tutoriais em vídeo?

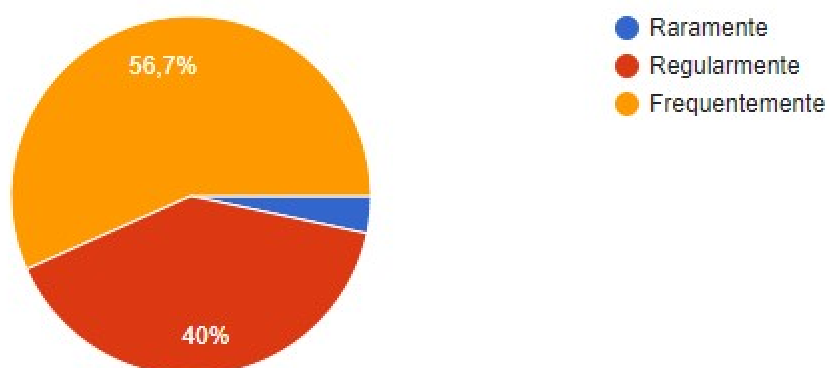

Gráfico 5.2 – Formulário 2

Nessa resposta, já vemos como os meios digitais estão mais frequentes atualmente, como as aulas gravadas em vídeo, lives de estudo e meios mais visuais e práticos do conteúdo.

3° Pergunta – Já estudou através de jogos ou aplicativos educacionais?

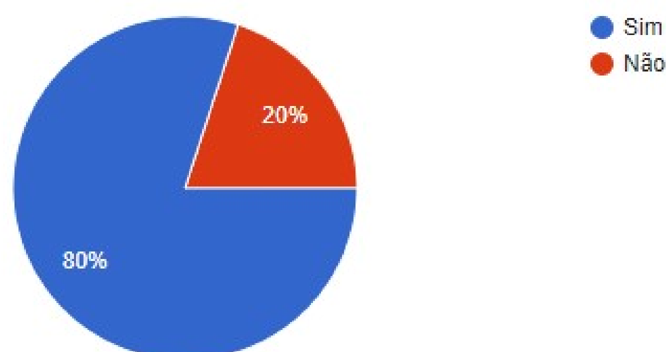

Gráfico 5.3 – Formulário 3

Os jogos educacionais estão cada vez mais presentes nos dias de hoje, seja por aplicativos de línguas ou ciências, até jogos para ensinar linguagens de programação.

4° Pergunta – Se sim, qual a experiência?

Gráfico 5.4 – Formulário 4

Para aqueles que experienciaram aplicativos educacionais, podemos notar que a maioria não conseguiu se manter focado por muito tempo, portanto, é necessário estudar meios melhores de gamificação para recompensar e prender mais a atenção do usuário no ensino, de forma que ele se auto-discipline para o estudo.

5° Pergunta – Você possuía algum conhecimento sobre Shaders?

30 respostas

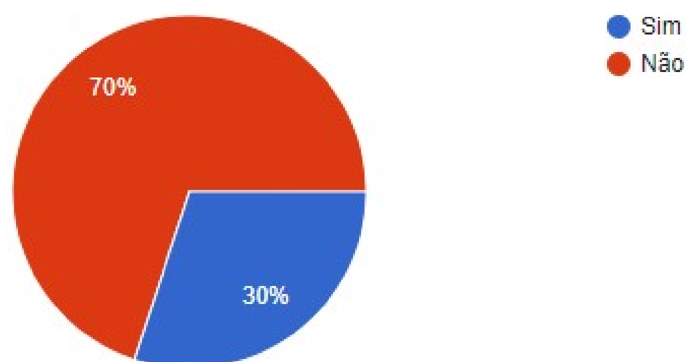

Gráfico 5.5 – Formulário 5

Conclui-se nessa resposta o que foi citado anteriormente, o conceito de *Shaders* é pouco conhecido e compartilhado, mesmo entre aqueles que tem uma bagagem de conhecimento sobre computação gráfica.

6° Pergunta – O jogo te ajudou, de certa forma, a expandir ou adquirir esse conhecimento?

30 respostas

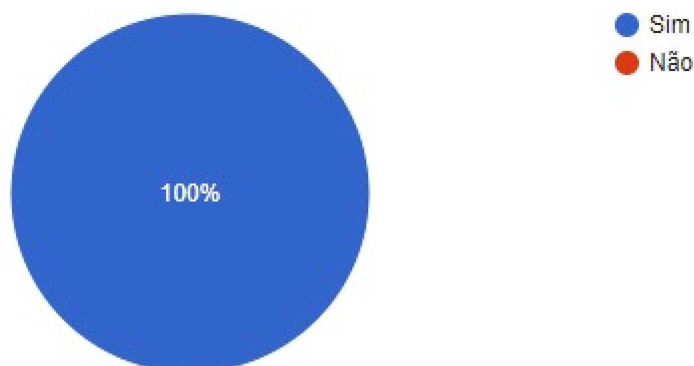

Gráfico 5.6 – Formulário 6

Com base nessas informações, podemos concluir que: O jogo, mesmo em seu protótipo, foi capaz de cumprir seus objetivos, trazendo um conhecimento específico sobre essa área pouco distribuída no Brasil.

Além dessas questões, foi criada uma referente a melhorias do projeto, não obrigatória. Foram registradas apenas duas respostas a ela. A primeira não era nenhuma ideia nova para o projeto, apenas a parabenização de como ele está, a segunda era referente a uma ideia futura, a possibilidade de área para programação em tempo real dos *shaders*, até o momento, não foi possível adicionar este elemento ao projeto, porém, caso haja trabalhos futuros para este, essa ideia será a primeira a ser adicionada.

Capítulo 6

CONCLUSÃO

Com as pesquisas realizadas foram notados a vantagem do uso de aplicativos educacionais direcionados a um tema em específico, tornando todo o processo de aprendizado pouco entediante e mais divertido, dando assim mais motivação ao aluno.

Porém, ainda assim este processo pode acabar se tornando entediante, e servir como distração ao usuário, a menos que sejam estudadas ferramentas de *design* de jogos, para que o jogador encontre motivação para realizar as tarefas propostas pelo aplicativo. Os jogos educacionais aplicados a conhecimentos específicos podem se tornar um poderoso instrumento de ensino para aqueles que desejam aprender esse conhecimento em ascensão.

Foram estudados desde diversas ferramentas de desenvolvimento de jogos até aos conceitos de *shading language* que poderiam ser passadas de forma mais didática ao usuário. Ao final foi desenvolvido um protótipo que conseguiu cumprir com sua proposta inicial, a de produzir uma plataforma educacional de auxílio e introdução ao desenvolvimento de *Shaders*, para que no fim do *game* os usuários possam ter uma noção mais estruturada sobre esse ponto específico da computação gráfica.

6.1 Contribuições e Limitações

Durante o desenvolvimento do projeto, surgiu a ideia de apresentar uma área de programação aos shaders dentro da própria aplicação, sendo que o que fosse programado lá, teria impacto no jogo de forma direta. No entando, não foram encontrados meios para que fosse possível a utilização dessa ideia. A Unity3D não possui um interpretador de códigos em tempo real, e a criação de um poderia levar

tempo suficiente para exceder o tempo limite do prazo para entrega do projeto, portanto, essa é uma ideia que pode vir a florescer em tempos futuros.

Como este tipo de jogo escolhido possui um vasto mapa, a criação do l*evel design* tomou um longo tempo e não foi possível a adição de mais chefes, e consequentemente, mais conhecimento sobre os *shaders*, porém, como todo o *background* já foi criado, tudo o que vier a partir dele será gerado naturalmente se investido mais tempo no projeto. O maior desafio que era a criação de um jogo educacional de auxílio e introdução ao desenvolvimento de *Shaders*, citados na introdução do projeto, conseguiu ser concretizado.

6.2 Lições aprendidas

O início ao estudo dos conceitos de *shading* provou-se uma das maiores dificuldades. Por necessitar de um conhecimento prévio sobre como o *pipeline* de renderização funciona e por envolver diversos cálculos matemáticos, tornou-se um grande desafio. Contudo, assim que assimilamos como cada bloco e função de código funciona, o progresso no aprendizado desse conceito se torna muito mais rápido.

6.3 Trabalhos Futuros

Há diversos temas e projetos que podem ser usados para o futuro, com base nesse trabalho.

Um deles é a própria continuação dessa ideia, levando o jogo a um nível superior, adicionando elementos como:

- API's de programação em tempo real durante o game, onde elas teriam impacto direto no ambiente que o jogador se encontrava;
- Criação de *sprites* de todas as unidades do game;
- Um mapa vasto com *shaders* extremamente realísticos, para aumentar a motivação do jogador em querer aprender esse conceito;

- Aprofundar ainda mais o material de ensino sobre os *shaders*, apresentando outros tipos como o *Tesselation* ou *Geometry Shaders*;
- Novas *shading languages* como a *GLSL* ou HLSL;
- Apresentar exemplos com cinemáticas realistas, relacionando com jogos que ganharam destaque na indústria por conta de seus gráficos, como *Crysis* da CryTek e *GTA V,* da Rockstar Games;
- Adição de mais chefes com habilidades diferentes que possam ser passadas ao jogador;
- A capacidade de selecionar as *infoSpheres* coletadas de onde estiver, funcionando como um "inventário";
- A possibilidade de interação do personagem e chefes através de falas;
- A tradução do conteúdo para outros idiomas como inglês ou espanhol;

Resumindo, a hipótese de tornar este protótipo em um jogo completo, que possa ser utilizado não apenas no Brasil, mas pelo mundo todo.

Com o conhecimento que foi adquirido sobre jogos e *shaders* abre-se um leque enorme de possibilidades para aplicações futuras, desde interfaces dinâmicas para aplicativos empresariais até sistemas em RV para ajudar em pesquisas científicas em áreas como da Medicina, Engenharia, etc. Com a ferramenta dos *shaders* a possibilidade de se apresentar algo digital em tempo real torna-se muito mais viável.

REFERÊNCIAS

ANGEL, The Case for Teaching Computer Graphics with WebGL: A 25-Year Perspective, University of New Mexico, 2017.

CG PROGRAMMING/UNITY, Disponível em:< https://en.wikibooks.org/wiki/Cg_Programming/Unity>. Acesso em: 15 mai. De 2017.

CONCI, Aura.; AZEVEDO, Eduardo.; LETA, Fabiana R. Computação Gráfica: Teoria e Prática. V.2. Rio de Janeiro: Editora Campus. 2007.

DC - ESCOLAS DE SC ADOTAM JOGOS ELETRÔNICOS CONSAGRADOS COMO METODOLOGIA DE ENSINO, Disponível em: <http://dc.clicrbs.com.br/sc/estilo-de-vida/noticia/2017/05/escolas-de-sc-adotam-jogos-eletronicos-consagrados-como-metodologia-de-ensino-9785029.html>. Acesso em: 23 ago. de 2017.

DETERDING citation, Disponível em: http://dl.acm.org/citation.cfm?doid=2181037.2181040. Acesso em: 04 jun. de 2017.

G1 – Número de desenvolvedores de games cresce 600% em 8 anos, diz associação, Disponível em <http://g1.globo.com/economia/negocios/noticia/numero-de-desenvolvedores-de-games-cresce-600-em-8-anos-diz-associacao.ghtml>. Acesso em: 05 jun. de 2017.

IG – COLUMBIA E 16 UNIVERSIDADES ADEREM À PLATAFORMA DE CURSOS ONLINE, Disponível em: http://ultimosegundo.ig.com.br/educacao/2012-09-19/columbia-e-mais-16-universidades-aderem-a-plataforma-de-cursos-online.html. Acesso em: 03 set. de 2017.

LOWE, Jen.; VIVO Patricio Gonzalez. The book of shaders. Disponível em:< https://thebookofshaders.com>. Acesso em: 01 mai. De 2017.

OPENGL WIKI, Disponível em:< https://www.khronos.org/opengl/wiki/Main_Page>. Acesso em: 13 mai. De 2017.

SHADERS, Disponível em: http://desenvolvimentodejogos.wikidot.com/shaders. Acesso em 22 de Abril de 2017.

SHREINER,Dave; ANGEL, Ed. Teaching a Shader-Based Introduction to Computer Graphics. Published in IEEE Computer Graphics and Applications, 2011, On pages 9-13. Disponível em: http://ieeexplore.ieee.org/document/5719035/ , Acessado em 22 de outubro de 2016

THE ESA – 2017 Essential facts about the computer and video game industry, Disponível em:< http://www.theesa.com/article/2017-essential-facts-computer-video-game-industry/>. Acesso em: 05 jun. de 2017.

VENTURA et al., Development of a Video Game that Teaches the Fundamentals of Computer Programming, Florida, 2015.

ZUCCONI A., LAMMERS K. Unity 5.x Shaders and Effect Cookbook: Master the art of Shader programming to bring life to your Unity projects. Birmingham: Packt Publishing Ltd. 2016.

Apêndice

QUESTIONÁRIO

Marque a opção que NÃO é uma diretiva de compilação

- #pragma target 4.0
- #pragma only_vertex
- #pragma vertex "nome da função"
- #pragma only_renderers opengl

Em Cg e HLSL, há um conceito chamado semânticas, defina-o:

- Um método de programação em um shader de pixels
- Maneiras de reduzir a capacidade de processamento gráfico sem reduzir a qualidade
- Meio de dizer a GPU como executar os dados através do pipeline gráfico
- Técnicas de texturização

Qual o procedimento necessário para criar novas vértices no shader de vértices?

- create new vert = fixed4();
- float4 v = new vert();
- O shader de vértices não é capaz de criar novas vértices
- Nenhuma das alternativas

A Microsoft é responsável pela criação de qual shader language?

- GLSL
- HLSL
- Cg
- Todas as alternativas

Selecione a opção que define o conceito de Smearing

- o.Albedo = _Color.rgb;
- o.Albedo = 1;
- o.Albedo = _Color.rg;
- Nenhuma das alternativas

Para que servem as Tags em Cg/HLSL?

- Organizar os shaders
- Setar tipos nas funções
- Mostrar como e quando esperam ser renderizados
- Programar o background do objeto

Para que serve o bloco Properties{} em Cg?

- Definir as propriedades do objeto 3D
- Referenciar elementos de interface
- Configurar as diretivas de compilação
- Delimitar as funções da textura

Qual o procedimento necessário para criar novas vértices no shader de vértices?

- create new vert = fixed4();
- float4 v = new vert();
- O shader de vértices não é capaz de criar novas vértices
- Nenhuma das alternativas

Qual o nome da função fornecida pela Unity que é utilizada quando nenhum shader roda no hardware em questão?

- Back
- FallBack
- returnTo
- Return