Relatório Final - Desenvolvimento de um jogo *First Person Shooter*

Disciplina: Fundamentos de Computação Gráfica
Alunos: Éder Monteiro e Tiago Binz

1. Introdução

A aplicação gráfica desenvolvida para o projeto final da disciplina consiste em um jogo *First Person Shooter*, ou mais informalmente, um *Doom-like*.

Infernal Rampage é um jogo que tenta emular as funcionalidades mais básicas dos *FPS* clássicos dos anos 1990, como Wolfenstein e Doom, onde o jogador deve derrotar uma horda de demônios que estão invadindo a Terra, eliminando o *Archdemon*, o comandante das legiões satânicas que querem destruir a humanidade.

Nas próximas seções são apresentados os detalhes da implementação do jogo, explicando como cada requisito do trabalho foi atingido.

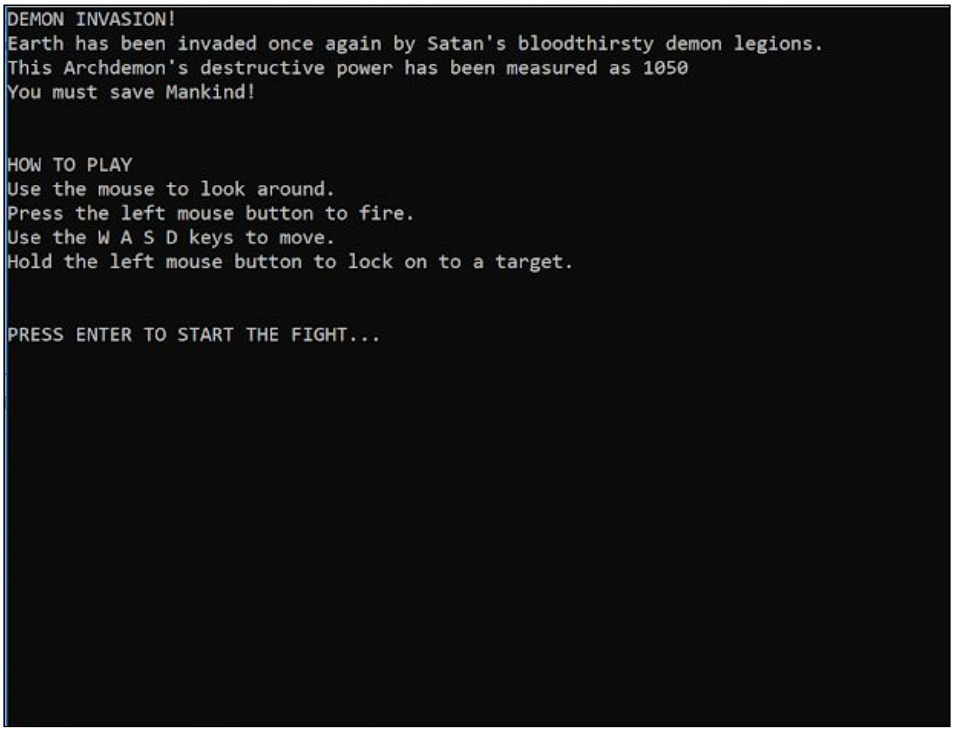

Figura 1: Tela inicial do jogo

2. Implementação

O jogo foi implementado utilizando a linguagem de programação C++ com OpenGL. Uma bibliotecas não vistas em aula foi utilizadas para a implementação do sistema de sons do jogo. Essa biblioteca é detalhada no decorrer desta seção.

A seguir, são listados os requisitos do projeto, com a explicação de como cada requisito foi atendido na implementação proposta.

2.1. Interação em tempo real e lógica de controle

A interação com o jogo implementado é aceitável. Não houve nenhuma queda de *frame rate* que influenciasse negativamente no *gameplay*, e todos os comandos são responsivos o bastante para prover uma jogabilidade satisfatória. Além disso, o jogo apresenta uma lógica de controle não-trivial, pois permite uma interação com diferentes tipos de inimigos e objetos através do mouse e do teclado, proporcionando uma certa profundidade à aplicação.

2.2. Transformações geométricas, projeções e coordenadas da câmera

A aplicação implementa transformações geométricas através de um sistema de *Game Objects* e *Transforms*. A câmera é definida através de uma subclasse de *Game Object*, o *PlayerController*, que possui informações como *Near Plane*, *Far Plane* e Field of View.

2.3. Malhas poligonais complexas

Os inimigos do jogo são os exemplos de malhas poligonais complexas mais óbvios. Há dois modelos diferentes de inimigos: os *minions*, inimigos mais fracos e em maior quantidade uma partida; e o *Archdemon*, o inimigo principal a ser combatido no jogo. Há ainda a malha que representa a arma do jogador. A Figura 2 mostra um *minion* do jogo junto da arma do jogador.

Figura 2: Inimigo a ser combatido no jogo

2.4. Transformações geométricas de objetos virtuais

A classe *Input* fornece informações para qualquer objeto sobre a entrada atual que o usuário está dando (movimento para frente, tiro, movimento da câmera,

etc.). A classe *PlayerController* chama funções de transação e rotação do objeto através dessas informações. Enquanto o jogador segura alguma das teclas de movimento, a arma usa uma função seno para se deslocar para cima e pra baixo, dando noção de movimento com passos.

2.5. Controle de câmera

Duas câmeras virtuais foram implementadas. A câmera livre é a câmera padrão do jogo, onde o jogador pode apontar para qualquer direção do cenário. A câmera *look-at* é usada no sistema de *lock* no inimigo, ou seja, o jogador pode mirar especificamente em um inimigo, e a câmera vai seguir o movimento desse objeto.

2.6. Objeto virtual com mais de uma instância

No código fonte da aplicação existe uma diferença entre um *Game Object* e uma *Mesh*. Um *Game Object* desenha uma instância de uma Mesh que só precisa ser carregada uma vez pelo programa, que recebe um “nome”, ou “índice” pelo qual ela pode ser referenciada.

Por exemplo, os *Game Objects* podem ter *Game Objects* filhos, possibilitando transformações hierárquicas, como no caso da arma ser filha do jogador.

2.7. Intersecção entre objetos virtuais

Foram implementados três tipos de intersecções, sendo elas:

- Cápsula-esfera: implementa a colisão entre um personagem (jogador ou inimigo) e um projétil.
- Cápsula-plano: implementa a colisão entre o chão e o jogador, além de causar a morte do jogador quando ele caminha para cima da lava.
- Plano-esfera: implementa a colisão entre um projétil e o chão do cenário.

Figura 3: Power up para recarregar munição

2.8. Modelos de iluminação

A seguir, a listagem de quais modelos foram utilizados, e para quais situações.

- Lambert: todos objetos do jogo, exceto a arma
- Blinn Phong: somente a arma do jogador

- Gourad (interpolação): todos objetos do jogo, exceto arma e power up
- Phong (interpolação): arma do jogador e power up

2.9. Mapeamento de texturas

O mapeamento das texturas foi feito no programa Blender. Objetos trazidos de outros jogos já tinham parte dessa informação pronta. Foi decidido que, para o jogo ficar mais dinâmico, os mapeamentos UV deveriam ser alterados em tempo de execução, para simular movimento da lava, contar o número de balas do jogador como no jogo Halo e apresentar efeitos de explosões animadas. Na Figura 4, temos exemplos de texturas para o cenário (lava, chão) e para a arma e sua quantidade de balas restante.

Figura 4: *Archdemon* em destaque

2.10. Curvas de Bézier

Curvas de Bézier foram utilizadas para implementar a movimentação dos inimigos. O *Archdemon* se movimenta sempre utilizando a mesma curva. Os outros inimigos escolhem aleatóriamente uma uma das curvas definidas quando são instanciados para realizar seu movimento.

2.11. Animação de movimento baseada no tempo

Foram utilizados os conceitos vistos em aula para implementar as animações do jogo. No loop principal da classe *World*, todos objetos são renderizados, e realizam uma chamada da função *Update*, recebendo como parâmetro o *Delta Time*, ou seja, a quantidade de tempo em segundos que se passou desde o último quadro.

A aplicação foi executada em três máquinas diferentes: duas máquinas com Windows e uma com Linux, sendo uma delas de propriedade da UFRGS, disponíveis aos alunos nos laboratórios. Nos três casos, o jogo não apresentou nenhuma mudança visível de execução e movimentação de objetos entre uma máquina e outra.

Extras

O jogo possui um pequeno conjunto de efeitos sonoros. Esses efeitos são referentes ao som dos tiros da arma do jogador, da reação do jogador ao tomar dano e da morte do *Archdemon*. Além disso, uma trilha sonora foi adicionada, combinando com a estética do jogo. A implementação dos efeitos sonoros foi feita utilizando a biblioteca irrKlang.

Além do sistema de efeitos sonoros, o jogo pode ter sua dificuldade configurada. Para isso, é necessário modificar o arquivo DemonInvasion.txt, presente no diretório . Os parâmetros modificáveis são mostrados na Figura 5.

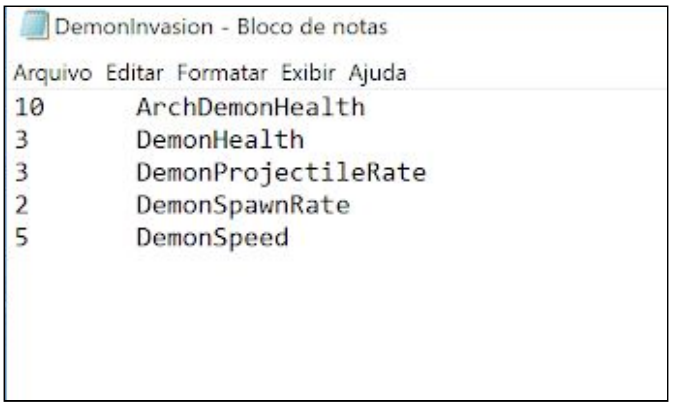

Figura 5: Arquivo de configuração de dificuldade

Os parâmetros têm os seguintes efeitos no jogo:

- ArchDemonHealth: Atribui a vida do inimigo principal, definindo o quão difícil será derrotá-lo
- DemonHealth: Atribui a vida dos demais inimigos
- DemonProjectileRate: Define a frequência com que os inimigos irão atacar o jogador
- DemonSpawnRate: Define a frequência que novos inimigos são criados
- DemonSpeed: Define a velocidade de todos inimigos menos o *Archdemon*

3. Manual

Os controles do jogo são simples, remetendo aos controles do clássico Doom, de maneira mais simplificada. A seguir, a lista de comandos:

- Mouse: olha ao redor do cenário. Os tiros da arma vão na direção do centro da tela
- Botão esquerdo do mouse: atira projéteis com a arma
- W, A, S, D: move para frente, para trás, para a esquerda e para a direita, respectivamente
- Botão direito do mouse: trava a mira em um inimigo específico

4. Compilação e execução

O projeto do *CodeBlocks* fornecido no arquivo *zip* de entrega contém tudo o que é necessário para compilar e executar o jogo implementado.

O projeto do *CodeBlocks* realiza o *link* da biblioteca irrKlang para que os efeitos sonoros sejam carregados corretamente. O arquivo *dll* dessa biblioteca se encontra no diretório *lib*.

O único requisito é ter a IDE *CodeBlocks* instalada, para conseguir abrir o projeto, compilar e executar através da IDE.